Chapter 5

Algorithmic complexity analysis

The aim of this chapter is to describe algorithms for understanding important aspects of the structure of functional graphs of generalized cyclotomic mappings of finite fields in detail and analyze their complexities. In Section 5.1, we set the ground by describing our computational model, the so-called dual model, in detail and introducing some important auxiliary concepts and results. We note that this dual model consists of carefully keeping track of three distinct parameters – the bit operations, elementary quantum gates and conversions from bits to qubits and vice versa – separately, which is, to the authors' knowledge, a novel approach and may be of independent, wider interest for readers working in quantum complexity analysis. Section 5.2 consists of the proof of Theorem 5.1.9, which provides complexity bounds for three fundamental algorithmic problems and may be considered the main result of this chapter. As mentioned in the introduction, it is an open problem how to encode the overall structure of the functional graph of a generalized cyclotomic mapping compactly; in particular, these results do *not* provide an efficient general algorithm for deciding whether the functional graphs of two given generalized cyclotomic mappings are isomorphic. However, in Section 5.3, we discuss four special cases in which this isomorphism problem can be solved efficiently.

5.1 Framework and auxiliary results

Throughout this chapter, we assume that f is an index d generalized cyclotomic mapping of $\mathbb{F}_q$, given in cyclotomic form (1.1), where either

- each a_i is specified as the field element 0 or as a power of a common, unknown primitive element ω of $\mathbb{F}_q$, or
- we explicitly know the minimal polynomial $P(T)$ over the prime subfield $\mathbb{F}_p$ of such an ω, and the a_i are represented as elements of $\mathbb{F}_p[T]/(P(T))$.

The main goal in this chapter is to analyze the complexities of the following algorithmic problems.

Problem 1. Given f, compute a compact parametrization of a CRL-list $\mathcal{L}$ of f (we note that $|\mathcal{L}|$ equals the number of cycles of f on its periodic points, which may be superpolynomial in $\log q$, so we want to avoid listing $\mathcal{L}$ element-wise).

Problem 2. Given f, compute a partition-tree register of f in the sense of Definition 5.1.2 below.

Problem 3. Given f, a partition-tree register of f, and a pair (r, l) such that $r \in \mathbb{F}_q$ is f-periodic and l is the cycle length of r under f, compute a compact description of the cyclic sequence of rooted tree isomorphism types from formula (3.4) (which characterizes the digraph isomorphism type of the connected component of Γ_f that contains r).

A partition-tree register of f is a standardized way of storing information about the arithmetic partitions $\mathcal{P}_i$ constructed in Section 3.3 and the rooted trees associated with their blocks. To define it, we first introduce the following auxiliary concept.

Definition 5.1.1. A *recursive tree description list* is a finite sequence $(\mathfrak{D}_n)_{n=0,1,\ldots,N}$ of sets that has an associated (unique) ordered sequence $(\mathfrak{T}_n)_{n=0,1,\ldots,N}$ of pairwise distinct, finite rooted tree isomorphism types such that the following hold.

(1) $\mathfrak{T}_0$ is the trivial rooted tree isomorphism type, and $\mathfrak{D}_0 = \emptyset$.

(2) For $n \geq 1$, each rooted tree attached in $\mathfrak{T}_n$ to the root of $\mathfrak{T}_n$ is isomorphic to $\mathfrak{T}_m$ for some $m \in \{0, 1, \ldots, n-1\}$. Moreover, $\mathfrak{D}_n$ is the set of all pairs (m, k_m), where $m \in \{0, 1, \ldots, n-1\}$ is an index for which $\mathfrak{T}_m$ is attached to the root of $\mathfrak{T}_n$ at least once, and k_m is the multiplicity with which it is attached.

In a recursive tree description list, each set $\mathfrak{D}_n$ can be viewed as a compact description of $\mathfrak{T}_n$, referring to the rooted trees attached to the root of $\mathfrak{T}_n$ with their (earlier) indices m, rather than their full descriptions. The idea of encoding isomorphism types of rooted trees via numbers ("tree indices") to get more compact descriptions of larger rooted trees is not new; it appears, for example, in the decision algorithm for isomorphism of directed rooted trees described in [6, Example 3.2 on p. 84]. In contrast to that algorithm, which is linear in the number of vertices, we do not list tree indices m repeatedly, but rather, we specify their multiplicities k_m. In situations such as ours, where entire sets (here: arithmetic partition blocks) of vertices can be dealt with simultaneously, this modification is crucial to ensure the efficiency of our algorithms relative to their smaller input length (which lies in $O(d \log q)$). In implementations, we assume that each $\mathfrak{D}_n$ is represented by an array (ordered list) of pairs (m, k_m), sorted by increasing m. Moreover, m and k_m, both of which are at most q, are to be represented by bit strings of length $\lfloor \log_2 q \rfloor + 1$ (please note, however, that we use other conventions for the related notion of a type-I tree register, introduced in Definition 5.3.2.1 (1)). We observe that with these conventions, all bit strings representing an element (m, k_m) of $\mathfrak{D}_n$ (for some n) have the same length, and the ordering of the elements of $\mathfrak{D}_n$ by increasing m corresponds to the lexicographic ordering of those bit string encodings.

Equipped with the concept of a recursive tree description list, we can define partition-tree registers of generalized cyclotomic mappings of finite fields as follows, using notations introduced in Section 3.3. We note that in this algorithmic chapter, we

frequently identify arithmetic partitions with specific spanning congruence sequences of them.

Definition 5.1.2. Let f be an index d generalized cyclotomic mapping of $\mathbb{F}_q$. For an $\bar{f}$-periodic index $i \in \{0, 1, \ldots, d-1\}$, we recall that i_t for $t \in \mathbb{Z}$ denotes the unique $\bar{f}$-periodic index in $\{0, 1, \ldots, d-1\}$ such that $(\bar{f}_{|\mathrm{per}(\bar{f})})^t(i) = i_t$. A *partition-tree register of* f is an ordered pair of the form

$$((\mathcal{Z}_i)_{i=0,1,\ldots,d-1}, ((\mathfrak{D}_n, (S_{n,i})_{i=0,1,\ldots,d}))_{n=0,1,\ldots,N})$$

such that the following hold.

(1) For each $i = 0, 1, \ldots, d-1$, $\mathcal{Z}_i$ is the following.
   (a) If i is $\bar{f}$-transient, then $\mathcal{Z}_i = \mathcal{P}_i$, given through a spanning congruence sequence of length $m_i \in \mathbb{N}_0$.
   (b) If i is $\bar{f}$-periodic, then $\mathcal{Z}_i$ is an $(H_i + 2)$-tuple $(\mathcal{X}_{i,h})_{h=-1,0,\ldots,H_i}$ such that
      (i) $\mathcal{X}_{i,-1} = (\theta_{i,h}(x))_{h=1,2,\ldots,H_i}$, and
      (ii) $\mathcal{X}_{i,h}$ for $h = 0, 1, \ldots, H_i$ is (a spanning congruence sequence for) the arithmetic partition $\lambda^h_{i_{-h}}(\mathcal{R}_{i_{-h}})$, of length $n_{i_{-h}} \in \mathbb{N}_0$.

(2) The sequence $(\mathfrak{D}_n)_{n=0,1,\ldots,N}$ is a recursive tree description list, with associated rooted tree isomorphism type sequence $(\mathfrak{T}_n)_{n=0,1,\ldots,N}$, such that the $\mathfrak{T}_n$ are just those rooted tree isomorphism types that are of one of the forms
   (a) $\mathrm{Tree}_i(\mathcal{P}_i, \vec{\nu}^{(\mathcal{P}_i)})$ for some $\bar{f}$-transient i and some $\vec{\nu}^{(\mathcal{P}_i)} \in \{\emptyset, \neg\}^{m_i}$ such that the block $\mathcal{B}(\mathcal{P}_i, \vec{\nu}^{(\mathcal{P}_i)})$ is non-empty;
   (b) $\mathrm{Tree}_{\Gamma_f}(0_{\mathbb{F}_q})$; or
   (c) $\mathrm{Tree}^{(h)}_i(\mathcal{P}_{i,h}, \vec{\nu}^{(\mathcal{P}_{i,h})})$ for some $\bar{f}$-periodic $i \neq d$, some $h \in \{0, 1, \ldots, H_i\}$ and some $\vec{\nu}^{(\mathcal{P}_{i,h})} \in \{\emptyset, \neg\}^{n_{i_0}+n_{i_{-1}}+\cdots+n_{i_{-h}}}$ such that $\mathcal{B}(\mathcal{Q}_{i,h}, \vec{\nu}^{(\mathcal{P}_{i,h})} \diamond \vec{\xi}_{i,h})$, which is the set of all points in $\mathcal{B}(\mathcal{P}_{i,h}, \vec{\nu}^{(\mathcal{P}_{i,h})})$ of $\mathfrak{h}$-value h, is non-empty.

(3) The objects $S_{n,i}$ satisfy the following.
   (a) If i is $\bar{f}$-transient, then
   $$S_{n,i} = \{\vec{\nu}^{(\mathcal{P}_i)} \in \{\emptyset, \neg\}^{m_i} : \mathcal{B}(\mathcal{P}_i, \vec{\nu}^{(\mathcal{P}_i)}) \neq \emptyset \text{ and } \mathrm{Tree}_i(\mathcal{P}_i, \vec{\nu}^{(\mathcal{P}_i)}) \cong \mathfrak{T}_n\}.$$
   (b) If $i = d$, then $S_{n,i} = S_{n,d} \in \{\emptyset, \neg\}$ is the logical sign associated with the truth value of the isomorphism relation $\mathrm{Tree}_{\Gamma_f}(0_{\mathbb{F}_q}) \cong \mathfrak{T}_n$.
   (c) If $i \neq d$ is $\bar{f}$-periodic, then $S_{n,i} = (S_{n,i,h})_{h=0,1,\ldots,H_i}$, where
   $$S_{n,i,h} = \{\vec{\nu}^{(\mathcal{P}_{i,h})} \in \{\emptyset, \neg\}^{n_{i_0}+n_{i_{-1}}+\cdots+n_{i_{-h}}} : \mathcal{B}(\mathcal{Q}_{i,h}, \vec{\nu}^{(\mathcal{P}_{i,h})} \diamond \vec{\xi}_{i,h}) \neq \emptyset \text{ and } \mathrm{Tree}^{(h)}_i(\mathcal{P}_{i,h}, \vec{\nu}^{(\mathcal{P}_{i,h})}) \cong \mathfrak{T}_n\}.$$

In implementations, we assume that each set $S_{n,i}$ for $\bar{f}$-transient i, as well as each set $S_{n,i,h}$ for $\bar{f}$-periodic $i < d$ and $h \in \{0, 1, \ldots, H_i\}$, is represented by a lexicographically ordered array of bit strings, where a bit 0 stands for $\emptyset$ and a bit 1 stands for $\neg$. We note that while a partition-tree register for f does not explicitly mention the partitions $\mathcal{P}_i = \mathcal{Q}_{i,H_i}$ for $\bar{f}$-periodic indices $i < d$, it is easy to read off their spanning congruence sequences and associated rooted trees from the register. Namely,

- the concatenation of the congruence sequences in $\mathcal{Z}_i$ spans $\mathcal{P}_i$; and
- by Proposition 3.3.5, the rooted tree associated with a block $\mathcal{B}(\mathcal{P}_i, \vec{\nu}^{(\mathcal{P}_i)})$ of $\mathcal{P}_i$ is of the form $\mathrm{Tree}_i^{(h)}(\mathcal{P}_{i,h}, \vec{\nu}^{(\mathcal{P}_{i,h})})$ for suitable $h \in \{0, 1, \ldots, H_i\}$ and

$$\vec{\nu}^{(\mathcal{P}_{i,h})} \in \{\emptyset, \neg\}^{n_{i_0}+n_{i_{-1}}+\cdots+n_{i_{-h}}}.$$

  The relevant parameters h and $\vec{\nu}^{(\mathcal{P}_{i,h})}$ can be read off from the logical sign tuple $\vec{\nu}^{(\mathcal{P}_i)}$ that characterizes the block of $\mathcal{P}_i$.

Before we proceed with the actual complexity analysis of Problems 1–3, we make some comments, starting with a discussion of our computational model.

As was already hinted at in Section 2.4, in order to even stand a chance of achieving polynomial runtime for our algorithms, we need quantum computers at least for certain subtasks, such as whenever a modular multiplicative order or a discrete logarithm needs to be computed. That being said, we only relegate certain well-defined tasks, for which efficient quantum algorithms are already known, to quantum computers, while the rest of our algorithms can be performed on a classical computer. Therefore, we use the following two computational models:

- a bit operation model with several kinds of queries for the tasks for which no efficient algorithms are known on a classical computer. In this model, which we henceforth refer to as the *query model* (and an algorithm in that model is a *query algorithm*), the complexity is measured as a tuple that tracks the amount of bit operations used outside queries and the amount of times each kind of query is called for;
- a model in which classical and quantum computers are used in tandem and can "feed" their outputs to each other; we refer to algorithms built like that as *dual algorithms*, and to the model as the *dual model*.

For the quantum side of the computations in the dual model, we specifically use the quantum *circuit* model, so whenever we speak of *quantum complexity*, we mean quantum *(elementary) gate* complexity. All quantum algorithms which we use are based on Shor's seminal paper [68], and all of them are *Las Vegas algorithms*, i.e., their runtime on a given input varies randomly, and their specified bit operation cost, gate complexity and number of conversions from bits to qubits and vice versa are to be understood as *expected* values. This also means that as a whole, all of our dual

algorithms are Las Vegas algorithms, and all parts of their specified complexities are expected values only.

On the other hand, for the "classical" side of the computations in either model, we use a bit operation model based on random memory access in the vein of [6, Section 1.2], in which memory access takes $O(N)$ bit operations if N is the bit length of the address (index) of the memory register that needs to be accessed. For example, accessing the stored value of a variable y_k, where k is a non-negative integer takes $O(\log k)$ bit operations – the entire memory address consists of a bit encoding for the letter "y" (which is assumed to be of length $O(1)$), concatenated with the standard binary representation of k. We thus assume that "jumping" to a place in memory after its address has been scanned is free. In addition to accessing memory registers by reading in their addresses, we also assume that we can save certain positions within a register through placing pointers (of which we have a finite amount, though we do not specify a concrete bound on their number), which enables us to jump back to that specific position (bit) in memory at a cost of $O(1)$ bit operations. Moreover, we assume that it takes $O(1)$ bit operations to move to a neighboring position in memory, including to the next entry of an array. We refer to the classical part of our complexity as *classical complexity* or (synonymously) *bit operations*.

Now, it is well known (see, e.g., [84, Section IV.3]) that each classical circuit has an equivalent quantum circuit in which the number of elementary gates is only larger by at most a constant factor. Based on this, it may seem tempting to just use circuits for both kinds of computations in the dual model, so that the classical part could be subsumed (without changing the Landau O-class of the gate complexity) in the quantum part, and it appears that this is the usual approach for quantum complexity analysis. For example, in [37, Sections 7.3 and 7.4], the complexity analysis of specific quantum algorithms (i.e., those that do not involve operations in a black-box group) only consists of counting the involved number of quantum gates, while the bit operation (or classical gate) cost of pre- and post-processing is ignored. For the algorithms in [37, Sections 7.3 and 7.4], this is perfectly fine, as that classical cost is a big-O of the quantum gate count regardless of whether bit operations or classical gates are used for measurement. However, our algorithms do involve a significantly larger classical cost than quantum gates, as can already be seen in the complexity bounds from Lemma 5.1.6; we note that while these are essentially algorithms from [37, Sections 7.3 and 7.4], they do end up with a relatively large classical cost if one wants them to be Las Vegas algorithms (due to the use of the AKS primality test). Hence, keeping track of the classical cost (whether bit operations or classical gates) and quantum gates separately seems natural, especially since the actual time cost of each quantum gate in a large-scale physical implementation of a quantum computer is not known at this point.

As for why we use bit operations (and not gates) in the classical part of our computations, we note that our algorithms for solving Problems 2 and 3 involve a copious

amount of "bookkeeping", i.e., memory access, and in any of the two circuit models, memory access is generally costly. Indeed, let us assume that, say, in the classical circuit model, we wish to access the value of a previously computed variable $y_k \in \{0, 1\}$, where the index $k \in \{1, \ldots, N\}$ is also a result of an earlier computation. When building the circuit, we do not know a priori which of the associated N wires carries the relevant information, and so this needs to be processed via a subcircuit that takes as input those N wires and the $O(\log N)$ wires carrying (the bit representation of) k. But each elementary gate only accepts $O(1)$ input bits, so the said subcircuit performing the memory access must consist of at least cN elementary gates for some constant $c > 0$, as opposed to the $O(\log N)$ cost of memory access for the analogous problem in our chosen bit operation model.

When communication between the classical and quantum part of a dual algorithm happens, classical bit strings $\vec{x} \in \{0, 1\}^N$ need to be converted into the corresponding qubit registers $|\vec{x}\rangle$ and vice versa. In our algorithms dealing with generalized cyclotomic mappings of $\mathbb{F}_q$, the bit length N is in $O(\log q)$ for each such conversion. Because it is not clear how costly such conversions are, it is of interest to count them separately (for both conversion directions together) in what we call the *conversion complexity* of the corresponding dual algorithm. The copying of converted information over to the next classical computer or quantum circuit, respectively, as well as the measurement taken at the end of a quantum circuit, are considered a part of the respective conversion process, and we do not track their cost separately. For standardization purposes, we assume that both the original input and final output of a dual algorithm are classical bit strings. In particular, the complexity of a pure quantum algorithm that is viewed as a dual algorithm involves two conversions (one each at the beginning and end of the algorithm) in addition to the quantum gate count.

Let us also talk about Grover's quantum algorithm for unstructured database search from [26]. This algorithm is famous for providing a quadratic speedup over the classical linear search algorithm, and given the aforementioned copious amount of bookkeeping in our algorithms, it seems natural to use it. However, there are some subtleties to take into account here, which ultimately led the authors to decide against the inclusion of Grover's algorithm in our analysis. The usual complexity analysis for Grover's algorithm assumes that the list to be searched (or rather, the associated characteristic function χ for the piece of information we want to find in the list) is given as a certain unitary operator U_χ, called *phase inversion*, which is subsequently used as a part of the quantum circuit for the algorithm and treated as an oracle. The celebrated Grover complexity of $O(\sqrt{N})$ for searching a list of length N refers to the number of times U_χ (and another, so-called phase shift operator) is applied before the final measurement. However, we are interested in gate complexities, so the gate complexity of U_χ needs to be included as an additional factor. Now, χ could be any function $\{0, 1, \ldots, N-1\} \to \{0, 1\}$ (the oft-used assumption that $\chi(j) = 1$ for a unique index j does not apply to our case), which we may also view as a (partial)

Boolean function in $n = \lfloor \log_2 N \rfloor$ variables. This means that in order for the quantum (gate) complexity of Grover's algorithm to "beat" the bit operation complexity of linear search, the worst-case quantum gate complexity of an n-variable Boolean function would need to be in $o(2^{n/2})$, and it is not clear whether this holds. We do note that it is known that the worst-case *classical* gate complexity of a Boolean function in n variables is of order of magnitude $2^n/n$ (Shannon, [67, Theorems 6 and 7 on pp. 76f.]), and that it is only a certain power away from the worst-case quantum gate complexity of an n-variable Boolean function (Beals et al., [12]).

We observe that our treatment of quantum algorithms in the dual model is idealized in the sense that we ignore the possibility of errors due to hardware failure and quantum noise. Like many authors, we do so relying on the celebrated quantum threshold theorem, the morale of which is that once the failure rate per elementary gate can be pushed beneath a certain, constant threshold, arbitrarily robust quantum algorithms can be constructed at little extra cost compared to their idealized counterparts. This theorem dates back to a paper of Shor [69], though the version stated there is weaker than what the theorem is known as today. Several variants of the stronger version (depending on the error model used) were proved independently by Aharonov and Ben-Or [4], Knill, Laflamme and Zurek [39], and Kitaev [38], respectively. The survey [25], in which the theorem is stated as Theorem 10, provides a unified proof of it.

The preceding discussion motivates the following definition of the notions of algorithmic complexity which our results in this chapter refer to.

Definition 5.1.3. We introduce the following concepts and notations.

(1) We denote by $\{0,1\}^{<\infty}$ the set of all finite bit strings. Formally,

$$\{0,1\}^{<\infty} = \bigcup_{n \in \mathbb{N}_0} \{0,1\}^n.$$

(2) An *algorithmic problem* is a function $\mathfrak{L}$ defined on a subset $\mathfrak{L}_{\text{in}}$ of $\{0,1\}^{<\infty}$ and mapping each bit string $\vec{x} \in \mathfrak{L}_{\text{in}}$ to some non-empty finite subset $\mathfrak{L}(\vec{x}) \subseteq \{0,1\}^{<\infty}$.

(3) In the situation of statement (2), the elements of $\mathfrak{L}_{\text{in}}$ are called the *admissible inputs for* $\mathfrak{L}$, and for each $\vec{x} \in \mathfrak{L}_{\text{in}}$, the elements of $\mathfrak{L}(\vec{x})$ are called the *admissible outputs for* $\vec{x}$ *(with respect to* $\mathfrak{L}$*)*.

(4) Let $\mathfrak{L}$ be an algorithmic problem, and let $y, y_1, y_2, \ldots, y_n$ be non-negative real parameters associated with the admissible inputs for $\mathfrak{L}$; formally, y and the y_j are functions $\mathfrak{L}_{\text{in}} \to [0,\infty)$.

(a) A tuple $\vec{\mathcal{C}}^{(\text{qry})} = (\mathcal{C}_{\text{class}}, \mathcal{C}_{\text{fdl}}, \mathcal{C}_{\text{mdl}}, \mathcal{C}_{\text{mord}}, \mathcal{C}_{\text{prt}})$ each entry of which is a function $[0,\infty)^n \to [0,\infty)$ is called a *y-bounded query complexity of* $\mathfrak{L}$

(with respect to $y_1, \ldots, y_n$) if there is a query algorithm which on each input $\vec{x} \in \mathfrak{L}_{\text{in}}$ produces an admissible output for $\vec{x}$ using

- $O(\mathcal{C}_{\text{class}}(y_1(\vec{x}), y_2(\vec{x}), \ldots, y_n(\vec{x})))$ bit operations outside the queries listed below;
- $O(\mathcal{C}_{\text{fdl}}(y_1(\vec{x}), y_2(\vec{x}), \ldots, y_n(\vec{x})))$ queries to compute a discrete logarithm in a finite field of size at most $y(\vec{x})$;
- $O(\mathcal{C}_{\text{mdl}}(y_1(\vec{x}), y_2(\vec{x}), \ldots, y_n(\vec{x})))$ queries to compute, for given $x, z \in (\mathbb{Z}/m\mathbb{Z})^*$, where $m < y(\vec{x})^2$, the modular discrete logarithms $\log_x^{(k)}(z)$, where
$$k \in \{m, p^{v_p(m)} : p \mid m\},$$
outputting a list consisting of the pair $(m, \log_x^{(m)}(z))$ and the quadruples
$$(p, v_p(m), p^{v_p(m)}, \log_x^{(p^{v_p(m)})}(z))$$
for all primes $p \mid m$;
- $O(\mathcal{C}_{\text{mord}}(y_1(\vec{x}), y_2(\vec{x}), \ldots, y_n(\vec{x})))$ queries to compute, for a given unit $x \in (\mathbb{Z}/m\mathbb{Z})^*$, where $m < y(\vec{x})^2$, the multiplicative orders $\text{ord}_k(x)$, where $k \in \{m, p^{v_p(m)} : p \mid m\}$, outputting a list consisting of the pair $(m, \text{ord}_m(x))$ and the quadruples
$$(p, v_p(m), p^{v_p(m)}, \text{ord}_{p^{v_p(m)}}(x))$$
for all primes $p \mid m$;
- $O(\mathcal{C}_{\text{prt}}(y_1(\vec{x}), y_2(\vec{x}), \ldots, y_n(\vec{x})))$ queries to find a primitive root $\text{r}^{(p)}$ modulo each odd prime power divisor $p^{v_p(m)} > 1$ for some integer $m < y(\vec{x})$, outputting the corresponding list of quadruples
$$(p, v_p(m), p^{v_p(m)}, \text{r}^{(p)}).$$

(b) A *y-bounded Las Vegas dual complexity for* $\mathfrak{L}$ is a triple
$$\vec{\mathcal{C}}^{(\text{LV})} = (\mathcal{C}_{\text{class}}, \mathcal{C}_{\text{quant}}, \mathcal{C}_{\text{conv}})$$
each entry of which is a function
$$[0, \infty)^n \to [0, \infty)$$
such that there is an (idealized) dual algorithm which on each input $\vec{x} \in \mathfrak{L}_{\text{in}}$ terminates after an expected number of

- $O(\mathcal{C}_{\text{class}}(y_1(\vec{x}), y_2(\vec{x}), \ldots, y_n(\vec{x})))$ bit operations,
- $O(\mathcal{C}_{\text{quant}}(y_1(\vec{x}), y_2(\vec{x}), \ldots, y_n(\vec{x})))$ elementary quantum gates, and

- $O(\mathcal{C}_{\mathrm{conv}}(y_1(\vec{x}), y_2(\vec{x}), \ldots, y_n(\vec{x})))$ conversions of length $O(\log y(\vec{x}))$ bit strings into qubit registers and of length $O(\log y(\vec{x}))$ qubit registers into bit strings,

producing an admissible output for $\vec{x}$.

In our algorithms, the value of $y(\vec{x})$ from Definition 5.1.3 is always equal to the corresponding field size q. While our definition of the query model does not explicitly include integer factorization queries, they are subsumed in either of modular discrete logarithm queries or multiplicative order queries. Indeed, in order to factor $m \in \mathbb{N}^+$ with $m < y(\vec{x})^2$, one can simply make the query to compute the multiplicative order modulo m of $x := 1 \in (\mathbb{Z}/m\mathbb{Z})^*$. Beside the pair $(m, 1)$, the resulting output consists of the quadruples $(p, \nu_p(m), p^{\nu_p(m)}, 1)$, where p ranges over the prime divisors of m, from which it is straightforward to read off the prime factorization of m. Likewise, one could make a modular discrete logarithm query with $x := y := 1$.

The assumption from Definition 5.1.3 (2) that each admissible input for $\mathfrak{L}$ should only have finitely many admissible outputs is without loss of generality for our analysis. It simplifies the formulation of Lemma 5.1.4 below, which is straightforward to prove and basically states that query complexities behave additively with respect to composition of algorithmic problems, which is defined as follows. If $\mathfrak{L}$ and $\mathfrak{L}'$ are algorithmic problems such that $\mathfrak{L}(\vec{x}) \subseteq \mathfrak{L}'_{\mathrm{in}}$ for each $\vec{x} \in \mathfrak{L}_{\mathrm{in}}$, then the *composition of* $\mathfrak{L}$ *and* $\mathfrak{L}'$, written $\mathfrak{L}\mathfrak{L}'$ or $\mathfrak{L}' \circ \mathfrak{L}$, is the algorithmic problem with input set $\mathfrak{L}_{\mathrm{in}}$ that is defined via $(\mathfrak{L}' \circ \mathfrak{L})(\vec{x}) := \bigcup \mathfrak{L}'(\mathfrak{L}(\vec{x}))$ (the union of the sets that make up the element-wise image of $\mathfrak{L}(\vec{x})$ under $\mathfrak{L}'$, i.e., all finite bit strings which are admissible outputs, with respect to $\mathfrak{L}'$, for some admissible output for $\vec{x}$ with respect to $\mathfrak{L}$).

Lemma 5.1.4. *Let $\mathfrak{L}$ and $\mathfrak{L}'$ be algorithmic problems such that $\mathfrak{L}(\vec{x}) \subseteq \mathfrak{L}'_{\mathrm{in}}$ for each $\vec{x} \in \mathfrak{L}_{\mathrm{in}}$, and let $y, y_1, y_2, \ldots, y_n$, respectively, $y', y'_1, y'_2, \ldots, y'_{n'}$ be non-negative real parameters that are associated with the admissible inputs of $\mathfrak{L}$, respectively, $\mathfrak{L}'$. For $j' = 1, 2, \ldots, n'$, we define*

$$\begin{aligned} y_{n+j'} : \mathfrak{L}_{\mathrm{in}} &\to [0, \infty), \\ \vec{x} &\mapsto \max\{y'_{j'}(\vec{y}) : \vec{y} \in \mathfrak{L}(\vec{x})\}. \end{aligned}$$

Because the composition $\mathfrak{L}\mathfrak{L}'$ has input set $\mathfrak{L}_{\mathrm{in}}$, we may view each y_j for $1 \leq j \leq n + n'$ as a parameter for $\mathfrak{L}\mathfrak{L}'$. Moreover, we define

$$\begin{aligned} y^+ : \mathfrak{L}_{\mathrm{in}} &\to [0, \infty), \\ \vec{x} &\mapsto \max\{y(\vec{x}), y'(\vec{y}) : \vec{y} \in \mathfrak{L}(\vec{x})\}. \end{aligned}$$

Finally, we let $\vec{\mathcal{C}}^{(\mathrm{qry})}$, respectively, $\vec{\mathcal{C}'}^{(\mathrm{qry})}$, be a y-bounded query complexity for $\mathfrak{L}$ with respect to $y_1, \ldots, y_n$, respectively, a y'-bounded query complexity for $\mathfrak{L}'$ with respect to $y'_1, \ldots, y'_{n'}$. For $k = 1, \ldots, 5$, we denote by $\mathcal{C}_k$, respectively, $\mathcal{C}'_k$, the k-th

entry of $\vec{\mathcal{C}}^{(\mathrm{qry})}$*, respectively,* $\vec{\mathcal{C}'}^{(\mathrm{qry})}$*. Then, defining*

$$\begin{aligned}&\mathcal{C}_k^+ : [0,\infty)^{n+n'} \to [0,\infty),\\ &(z_1,\dots,z_{n+n'}) \mapsto \mathcal{C}_k(z_1,\dots,z_n) + \mathcal{C}'_k(z_{n+1},\dots,z_{n+n'}),\end{aligned}$$

the tuple $\vec{\mathcal{C}}^{+(\mathrm{qry})} := (\mathcal{C}_1^+,\dots,\mathcal{C}_5^+)$ *is a* y^+*-bounded query complexity for* $\mathfrak{L}\mathfrak{L}'$ *with respect to* $y_1,\dots,y_{n+n'}$*.*

On the other hand, for each given algorithmic problem $\mathfrak{L}$ and non-negative real parameter y associated with the admissible inputs for $\mathfrak{L}$, a y-bounded Las Vegas dual complexity for $\mathfrak{L}$ can be derived from a y-bounded query complexity for $\mathfrak{L}$ as long as y can be bounded in terms of the other parameters y_j; see Lemma 5.1.7 below.

As usual, when specifying complexities in a concrete situation, we identify functions with their defining terms. For example, if d and q are the relevant parameters associated with our inputs, we may specify a q-bounded Las Vegas dual complexity as

$$(d^3 \log q,\ d \log^{2+o(1)} q,\ \log^{1+o(1)} q),$$

rather than introduce names for the functions in the three components. In this context, we also note that $\log^k x$ always denotes the arithmetic power $(\log x)^k$, *not* the function value at x of the k-fold iterate of log. We always spell iterated logarithms out ($\log\log$, $\log\log\log$, etc.).

The following lemma, which is used throughout this chapter, provides the complexities of some fundamental algorithmic problems. For more information, we refer to the classical books [11, 75].

Lemma 5.1.5. *The following hold.*

(1) *Addition and subtraction of integers of absolute value less than* m*, as well as addition and subtraction modulo* m *cost* $O(\log m)$ *bit operations each.*

(2) *Addition and subtraction in the finite field* $\mathbb{F}_q$ *cost* $O(\log q)$ *bit operations each.*

(3) *Multiplication of positive integers less than* m*, multiplication modulo* m *and division of positive integers less than* m *with remainder each respectively cost* $O(\log^{1+o(1)} m)$ *bit operations.*

(4) *Let* $x \in (\mathbb{Z}/m\mathbb{Z})^*$*. The computation of* $\mathrm{inv}_m(x)$*, the multiplicative inverse of* x *modulo* m*, costs* $O(\log^{1+o(1)} m)$ *bit operations.*

(5) *Multiplication, multiplicative inversion and division in the finite field* $\mathbb{F}_q$ *each have a bit operation cost of* $O(\log^{1+o(1)} q)$*.*

(6) *Let* $x \in \mathbb{Z}/m\mathbb{Z}$ *and* $e \in \mathbb{N}_0$*. The computation of the power* x^e *modulo* m *has a bit operation cost of* $O(\log(e)\log^{1+o(1)} m)$*.*

(7) *Let $x \in \mathbb{F}_q$ and $e \in \mathbb{N}_0$. The computation of x^e costs $O(\log(e)\log^{1+o(1)} q)$ bit operations.*

(8) *The computations of the* gcd *and* lcm *of two positive integers that are at most m cost $O(\log^{1+o(1)} m)$ bit operations each.*

(9) *Checking deterministically whether a given positive integer m is a prime has a bit operation cost of $O(\log^{6+o(1)} m)$.*

(10) *An array of n bit strings, each of length k, can be lexicographically sorted with $O(kn \log n)$ bit operations.*

(11) *We assume given two lexicographically sorted arrays of bit strings (not necessarily all of the same length) and consider the algorithmic problem of finding the lexicographically sorted version of their concatenation (i.e., the problem of* merging *those sorted arrays). For $j = 1, 2$, say the j-th array has N_j entries, and the sum of the bit lengths of the strings stored in it is $l^{(j)}_{\text{total}}$. Then those two sorted arrays can be merged within $O(N_1 + N_2 + l^{(1)}_{\text{total}} + l^{(2)}_{\text{total}})$ bit operations (and thus within $O(l^{(1)}_{\text{total}} + l^{(2)}_{\text{total}})$ bit operations if all bit strings in question are non-empty).*

Proof. For statement (1), it is well known (and easy to check) that using the schoolbook algorithms for addition and subtraction yields the specified complexities.

For statement (2), let $q = p^m$. We refer to [51, Table 2.8 on p. 84], and note that the only $(\mathbb{Z}/p\mathbb{Z})$-operations involved in an addition/subtraction in $\mathbb{F}_q$ are modular additions/subtractions. Therefore, statement (1) implies that the cost of addition and subtraction in $\mathbb{F}_q$ is in $O(m \log p) = O(\log q)$, as required.

For statement (3), it follows from the Schönhage–Strassen algorithm [65] or the (slightly faster) algorithm [30] by Harvey and van der Hoeven that the multiplication of two positive integers less than m costs $O(\log^{1+o(1)} m)$ bit operations. Moreover, integer division with remainder also costs $O(\log^{1+o(1)} m)$ bit operations if the Newton–Raphson algorithm is used for it; see [2, Section 1.3]. Multiplication modulo m can be done by performing a (non-modular) multiplication of the two integers in question (resulting in a number with $O(\log(m^2)) = O(\log m)$ bits), followed by a modular reduction (which is a part of division with remainder). In total, multiplication modulo m thus also only requires $O(\log^{1+o(1)} m)$ bit operations.

For statement (4), we note that inversion modulo m can be done with the Extended Euclidean algorithm, which takes $O(\log^{1+o(1)} m)$ bit operations when using an accelerated variant of it due to Schönhage (based on earlier ideas of Knuth) [64]; see also [83], which provides a generalization of this and may be more accessible due to being written in English.

For statement (5), we note that $\mathbb{F}_q$ is given as $(\mathbb{Z}/p\mathbb{Z})[T]/(P(T))$, where $P(T)$ is a monic primitive irreducible polynomial of degree $m := \log_p q$. In order to multiply two elements $Q_1(T) + (P(T))$ and $Q_2(T) + (P(T))$ of $\mathbb{F}_q$, one computes the

polynomial product $Q_1(T)Q_2(T) \in (\mathbb{Z}/p\mathbb{Z})[T]$, then determines its remainder upon division by $P(T)$. As observed in [76, second paragraph in Section 2], fast methods for multiplication of polynomials over $\mathbb{Z}/p\mathbb{Z}$ of degree at most n, as well as for divisions with remainder of such polynomials, take $O(n^{1+o(1)})$ operations (additions, subtractions, multiplications, multiplicative inversions) in $\mathbb{Z}/p\mathbb{Z}$. This corresponds to a bit operation cost of $O(n^{1+o(1)} \log^{1+o(1)} p)$ by statements (1) and (3). It follows that the computation of $Q_1(T)Q_2(T)$, and the subsequent computation of its remainder modulo $P(T)$, both take $O(m^{1+o(1)} \log^{1+o(1)} p) = O(\log^{1+o(1)} q)$ bit operations, as needed for the asserted complexity bound on multiplication in $\mathbb{F}_q$ to hold.

Now, because a division in $\mathbb{F}_q$ consists of a multiplicative inversion followed by a multiplication, it suffices to argue that the bit operation cost of multiplicative inversion in $\mathbb{F}_q$ is in $O(\log^{1+o(1)} q)$ to conclude the proof of this statement. Assuming that $P(T) \nmid Q(T)$, the multiplicative inversion of $Q(T)$ modulo $P(T)$ may be performed by writing $1 = \gcd(P(T), Q(T))$ as a $(\mathbb{Z}/p\mathbb{Z})[T]$-linear combination of $P(T)$ and $Q(T)$, and reducing the scalar of $Q(T)$ in this linear combination modulo $P(T)$. The algorithm described in [6, Section 8.9] uses $O(\log^{1+o(1)}(p^m) \cdot \log m) = O(\log^{1+o(1)} q)$ bit operations to compute $\gcd(P(T), Q(T))$ (according to [6, Theorem 8.19]). In the process, one may store the (2×2)-matrices $R_1, R_2, \ldots, R_K$ (with coefficients in $(\mathbb{Z}/p\mathbb{Z})[T]$) that are output (in the listed order) by the $K \in O(\log m)$ calls of the HGCD procedure from [6, Figure 8.7 on p. 304]. Let $P_0(T) = P(T)$, $P_1(T) = Q(T)$, $P_2(T), \ldots, P_N(T) = \gcd(P(T), Q(T)) = 1$ be the successive remainders appearing in the classical, "slow" version of the Euclidean algorithm applied to $(P(T), Q(T))$, and for $t \in \{0, 1, \ldots, N-1\}$, let $Q_t(T)$ be the quotient of the polynomial division of $P_t(T)$ by $P_{t+1}(T)$, which satisfies $\deg Q_t(T) = \deg P_t(T) - \deg P_{t+1}(T)$. By [6, statement on p. 303 that the output of HGCD is of the form R_{0j}, and definition of R_{ij} before Example 8.10 on p. 302], each of the matrices R_j is a product of matrices of the form

$$\begin{pmatrix} 0 & 1 \\ 1 & -Q_t(T) \end{pmatrix}$$

for pairwise distinct t. Therefore, each entry of $R_j(T)$ is a polynomial in $(\mathbb{Z}/p\mathbb{Z})[T]$ of degree at most

$$\sum_{t=0}^{N-1} \deg Q_t(T) = \sum_{t=0}^{N-1} (\deg P_t(T) - \deg P_{t+1}(T)) = \deg P(T) - \deg 1 = \deg P(T).$$

Moreover, by [6, Lemma 8.5 (a)], for each $k \in \{1, 2, \ldots, K\}$, one has

$$R_k R_{k-1} \cdots R_1 \cdot \begin{pmatrix} P(T) \\ Q(T) \end{pmatrix} = \begin{pmatrix} P_j(T) \\ P_{j+1}(T) \end{pmatrix}$$

for some $j = j(k) \in \{0, 1, \ldots, N-1\}$, and specifically

$$R_K R_{K-1} \cdots R_1 \cdot \begin{pmatrix} P(T) \\ Q(T) \end{pmatrix} = \begin{pmatrix} P_{N-1}(T) \\ P_N(T) \end{pmatrix} = \begin{pmatrix} P_{N-1}(T) \\ 1 \end{pmatrix}.$$

This latter equality yields an expression of 1 as a linear combination of $P(T)$ and $Q(T)$, in which the (reduction modulo $P(T)$ of the) scalar of $Q(T)$ is equal to the (reduction modulo $P(T)$ of the) lower right coefficient of the (2×2)-matrix $R_K R_{K-1} \cdots R_1$. It takes $O(K) \subseteq O(\log m)$ additions, multiplications and divisions with remainder in $(\mathbb{Z}/p\mathbb{Z})[T]$ to compute this matrix product, which also corresponds to a bit operation cost of $O(\log m \cdot \log^{1+o(1)} p^m) = O(\log^{1+o(1)} q)$, as required.

For statement (6), we note that using "Square and Multiply", the power $x^e \bmod m$ can be computed with $O(\log e)$ multiplications modulo m, so statement (3) yields the claim.

For statement (7), the proof is analogous to the one for statement (6), but using statement (5) in place of statement (3).

For statement (8), one may use the Euclidean algorithm to compute a greatest common divisor and refer to [64] or [83]. Moreover, $\operatorname{lcm}(x, y) = xy/\gcd(x, y)$, so statement (3) completes the proof of this claim.

For statement (9), the asserted complexity is achieved by a variant of the AKS primality test devised by Lenstra and Pomerance, see [3, 45].

For statement (10), we refer the reader to [6, Algorithm 3.1 on pp. 78f.], observing that the variable m from that algorithm has the value 2 in our situation. We note that while [6, Theorem 3.1 on p. 79] states that this algorithm costs $O(kn)$ bit operations, this uses an assumption which our computational model does not share. Namely, [6, Algorithm 3.1 on pp. 78f.] uses a "pointer" for each bit string, which must not be confused with the way we use that word. In our model, a pointer is a short-cut to jump to a previously saved point in memory using only $O(1)$ bit operations, and we may only use $O(1)$ of these pointers (i.e., the number of pointers used must not tend to ∞ as the input length tends to ∞). On the other hand, in [6, Algorithm 3.1 on pp. 78f.], the word "pointer" appears to denote what we would call the memory address of the respective bit string. It is stated explicitly in [6, Algorithm 3.1 on pp. 78f.] that the authors of that book assume that a pointer in their sense can be processed (i.e., stored and used to jump to the respective bit string) within $O(1)$ bit operations. However, in our model, since n of these memory addresses are needed, it takes $O(\log n)$ bit operations to process an address, which leads to the additional factor $\log n$ in our cost.

For statement (11), we observe that it is easy to prove that the merging algorithm [40, Algorithm M on p. 158] achieves this complexity, as long as pointers (in our sense of the word) are used to immediately jump back to saved positions in the arrays, which avoids additional logarithmic factors in the complexity. In fact, in the discussion from [40, p. 159], it is stated that the achieved complexity is in $O(N_1 + N_2)$, but

this uses the assumption that each stored bit string has constantly bounded bit length (causing $l^{(j)}_{\text{total}} \in O(N_j)$ for $j = 1, 2$). ∎

The next lemma essentially provides the Las Vegas dual complexities of the query problems from our query model. It is used to translate query complexities into Las Vegas dual complexities; see Lemma 5.1.7 below.

Lemma 5.1.6. *The following hold.*

(1) *The prime factorization of the positive integer m can be performed with a Las Vegas dual algorithm with m-bounded complexity*

$$(\log^{7+o(1)} m, \log^{3+o(1)} m, \log m).$$

(2) *The computation of the multiplicative order of $x \in (\mathbb{Z}/m\mathbb{Z})^*$ can be performed with a Las Vegas dual algorithm with (expected) m-bounded dual complexity*

$$(\log^{7+o(1)} m, \log^{3+o(1)} m, \log m).$$

(3) *Let m be a positive integer. For $x, y \in (\mathbb{Z}/m\mathbb{Z})^*$, the modular discrete logarithm $\log_x^{(m)}(y)$ can be computed with a Las Vegas dual algorithm with m-bounded complexity*

$$(\log^{7+o(1)} m, \log^{3+o(1)} m, \log m).$$

(4) *Let q be a prime power. For $x, y \in \mathbb{F}_q^*$, the discrete logarithm $\log_x(y)$ can be computed with a Las Vegas dual algorithm with q-bounded complexity*

$$(\log^{3+o(1)} q, \log^{3+o(1)} q, \log q).$$

(5) *Let p be an odd prime, and k a positive integer. On input (p, k), a primitive root modulo p^k can be found with a Las Vegas dual algorithm with p-bounded complexity* $(\log^{7+o(1)} p, \log^{3+o(1)} p, \log p)$.

Proof. For the proofs of statements (1) and (2), we follow the approach described in [37, Section 7.3], which is originally due to Miller [53] and Shor [68]. A different approach, as kindly pointed out by one of the reviewers, would be to reduce those problems to special instances of the hidden subgroup problem, which would in fact work for all five problems and offer a different explanation for the quantum parts of the complexities (see [37, Corollary 7.5.3]).

For the approach of Miller and Shor, we need to first analyze the complexity of the order-finding algorithm from [37, p. 137], which uses quantum circuits combined with some classical post-processing. It should be noted that this algorithm admits absolute bounds on (i.e., not just expected values of) the different parts of its complexity, but the output is only correct with probability at least 0.399, making this a *Monte Carlo algorithm*, not the Las Vegas algorithm we wish to construct in the end.

First, we need to analyze the complexity of the continued fractions algorithm (it is mentioned in [37, Theorem 7.1.7] that this algorithm has polynomial complexity without specifying the degree). Let us assume given a floating point number (in binary format) that represents the rational number $k/2^n$, where $k \in \{0, 1, \ldots, 2^n - 1\}$. Then there is a sequence of fractions

$$\mathrm{conv}_j(k, 2^n) = \frac{\mathrm{num}_j(k, 2^n)}{\mathrm{den}_j(k, 2^n)}$$

for $j = 1, 2, \ldots, N \in O(n)$, the *(principal) convergents of* $k/2^n$, which are optimal approximations of $k/2^n$ relative to the size of their denominators $\mathrm{den}_j(k, 2^n) \leq 2^n$; for details, we refer the reader to [37, Theorem 7.1.7 and Exercise 7.1.7] and [42, Chapter I]. An important property which we need later is that any reduced integer fraction y/z such that

$$\left| \frac{k}{2^n} - \frac{y}{z} \right| \leq \frac{1}{2z^2}$$

is one of the convergents of $k/2^n$; see [42, Corollary 2 on p. 11]. By [42, Theorem 1 on p. 2], each of the two sequences $(\mathrm{num}_j(k, 2^n))_{j=1,\ldots,N}$ and $(\mathrm{den}_j(k, 2^n))_{j=1,\ldots,N}$ is defined through a simple recursion (involving $O(1)$ integer additions and multiplications in each recursion step) in terms of the so-called *continued fractions coefficients of* $k/2^n$, which are just the integer quotient values in the divisions that occur upon applying the Euclidean algorithm to $(2^n, k)$. In view of [20, Problem 31-2 posed on p. 937] (see also [17] for a worked out solution of this problem using a telescopic sum argument), one can compute and store the continued fraction coefficients of $k/2^n$ using $O(n^2)$ bit operations. Following that, the computation of $\mathrm{num}_j(k, 2^n)$ and $\mathrm{den}_j(k, 2^n)$ for all relevant j takes another $O(n \cdot n^{1+o(1)}) \subseteq O(n^{2+o(1)})$ bit operations by statements (1) and (3) of Lemma 5.1.5.

Having analyzed the continued fractions algorithm, let us now turn to the order-finding algorithm described in [37, p. 137]. In accordance with our notation, we assume that this algorithm is used to find the multiplicative order of x modulo m (in [37], the variable a, respectively, N, is used in place of x, respectively, m). The algorithm starts by computing $m' := \lceil 2 \log_2 m \rceil + 1$, which takes $O(\log m)$ bit operations. We observe that $\mathrm{ord}(x)$, the multiplicative order of x modulo m, is at most $2^{(m'-1)/2}$. After this, we need to initialize two m'-qubit registers, which in our dual model formally takes $O(\log m)$ bit operations for printing the length m' bit strings $0 \cdots 0$ and $0 \cdots 01$, followed by $2 \in O(1)$ conversions of these strings into the corresponding qubit registers $|0\rangle^{\otimes m'}$ and $|0 \cdots 01\rangle$. Steps 4–6 of the algorithm in [37, p. 137] are applications of quantum circuits to those registers, which according to the analysis in [37, pp. 138f.] consist of $O(\log^{2+o(1)} m)$ elementary gates. Next, the measurement described in step 7 of [37, p. 137] corresponds to one more conversion in our model (this time from qubits to classical bits), and with high probability, it leads to a "good estimate" (see below) $k_1/2^{m'}$ of a random integer multiple $t/\mathrm{ord}_m(x)$

of $1/\operatorname{ord}_m(x)$, with $t \in \{0, 1, \ldots, \operatorname{ord}_m(x) - 1\}$. Considering step 8 of [37, p. 137] next, we believe that there is a mistake in the formulation of this step, the first sentence of which should in our opinion read (using the notation from there) "Use the continued fractions algorithm to obtain integers $c_1 \geq 0$ and r_1 with $1 \leq r_1 \leq 2^{(n-1)/2}$ such that $|x_1/2^n - c_1/r_1| \leq 1/2^{n+1}$." In any case, this is a formulation that works. Indeed, switching back to our notation, as long as the output $k_1/2^{m'}$ of step 7 is a good estimate of $t/\operatorname{ord}(x)$ for some $t \in \{0, 1, \ldots, \operatorname{ord}(x) - 1\}$, it follows (by the definition of "good estimate" from [37, beginning of Section 7.1.1], see in particular [37, Theorem 7.1.4]) that

$$\left| \frac{t}{\operatorname{ord}(x)} - \frac{k_1}{2^{m'}} \right| \leq \frac{1}{2^{m'+1}} < \frac{1}{2^{m'}} \leq \frac{1}{2\operatorname{ord}(x)^2}$$

whence, as noted above, we have $t/\operatorname{ord}(x) = \operatorname{conv}_j(k_1, 2^{m'})$ for some j by [42, Corollary 2 on p. 11]. By our above analysis of the continued fractions algorithm, one can thus find, using $O(\log^{2+o(1)} m)$ bit operations, an index j and associated values $\operatorname{num}_j(k_1, 2^{m'})$ and $\operatorname{den}_j(k_1, 2^{m'})$ with $\operatorname{den}_j(k_1, 2^{m'}) \leq 2^{(m'-1)/2}$ such that

$$\left| \operatorname{conv}_j(k_1, 2^{m'}) - \frac{k_1}{2^{m'}} \right| = \left| \frac{\operatorname{num}_j(k_1, 2^{m'})}{\operatorname{den}_j(k_1, 2^{m'})} - \frac{k_1}{2^{m'}} \right| \leq \frac{1}{2^{m'+1}},$$

unless we had bad luck with regard to the output of step 7. If so, it makes sense to abandon the computations and output "FAIL", as specified in [37, step 8 on p. 137]. Now, it follows that

$$\begin{aligned} \left| \frac{t}{\operatorname{ord}(x)} - \operatorname{conv}_j(k_1, 2^{m'}) \right| &\leq \left| \frac{t}{\operatorname{ord}(x)} - \frac{k_1}{2^{m'}} \right| + \left| \frac{k_1}{2^{m'}} - \operatorname{conv}_j(k_1, 2^{m'}) \right| \\ &\leq \frac{1}{2^{m'+1}} + \frac{1}{2^{m'+1}} = \frac{1}{2^{m'}} \\ &\leq \min\left(\frac{1}{2\operatorname{ord}(x)^2}, \frac{1}{2\operatorname{den}_j(k_1, 2^{m'})^2} \right). \end{aligned}$$

Using [37, Exercise 7.1.7 (b)], this implies that $t/\operatorname{ord}(x) = \operatorname{conv}_j(k_1, 2^{m'})$, as required for the correctness of the algorithm from [37, p. 137]. Step 9 of [37, p. 137] is just a repetition of steps 1–8, hence does not make a difference for the O-class of the complexity. Finally, steps 10 and 11 of [37, p. 137] take $O(\log^{2+o(1)} m)$ bit operations by statements (6) and (8) of Lemma 5.1.5. In summary, the order-finding algorithm from [37, p. 137] may be viewed as a dual algorithm which, on input $x \in (\mathbb{Z}/m\mathbb{Z})^*$ and m, outputs the multiplicative order of x modulo m with probability at least 39.9% (see [37, Theorem 7.3.2]), and does so taking $O(\log^{2+o(1)} m)$ bit operations and elementary quantum gates each, as well as $O(1)$ conversions of $O(\log m)$-bit strings to $O(\log m)$-qubit registers or vice versa. As noted in [37, Theorem 7.3.2] (and is clear from step 11), unless the output of that algorithm is "FAIL", it always outputs at

least an integer multiple of $\mathrm{ord}(x)$. This concludes the preparation for the proofs of statements (1) and (2), which we tackle next.

For statement (1), we assume given a positive integer m. We wish to obtain the prime factorization of m. Formally, we wish to output the list of pairs $(p, \nu_p(m))$, where p ranges over the prime divisors of m. First, we describe and analyze a deterministic (classical) algorithm that decides whether m is a power of a single prime p and, if so, outputs $(p, \nu_p(m))$; see also [37, Exercise 7.3.3]. This algorithm is, in turn, based on a deterministic routine that decides whether a given $m \in \mathbb{N}^+$ is a power n^k of a positive integer $n < m$ and, if so, outputs (n, k) for the smallest possible value of $k \geq 2$. We note that if $m = n^k$ for some $n \in \{1, 2, \ldots, m-1\}$ and some $k \in \mathbb{N}^+$, then $k \leq \lfloor \log_2 m \rfloor$. Therefore, we loop over $k = 2, 3, \ldots, \lfloor \log_2 m \rfloor$, and for each fixed value of k, we perform a binary search for n, using the strict monotonicity of the function $x \mapsto x^k$. More specifically, we initialize $n := 2$, and as long as $n^k < m$, we double n until $n^k = m$ or $n^k > m$. In the latter case, we start a binary search between $\frac{n}{2}$ and n. With this approach, the values of the powers n^k which we compute never exceed $2^{\lfloor \log_2 m \rfloor} m \leq m^2$, whence each individual power computation in the process takes

$$O(\log(k) \log^{1+o(1)}(m^2)) = O(\log \log m \log^{1+o(1)} m) = O(\log^{1+o(1)} m)$$

bit operations by statement (6) of Lemma 5.1.5. Because we loop over $O(\log m)$ values of k, and for each k, the binary search for n has $O(\log m)$ iterations, it follows that it takes

$$O(\log m \cdot \log m \cdot \log^{1+o(1)} m) = O(\log^{3+o(1)} m)$$

bit operations to find the minimal working value of k and associated $n = \sqrt[k]{m}$, or see that they do not exist.

Let us now describe a procedure to check whether a given $m \in \mathbb{N}^+$ is a prime power and, if so, write it as such. First, by repeating the above procedure with the loop going backward (i.e., for $k = \lfloor \log_2 m \rfloor, \lfloor \log_2 m \rfloor - 1, \ldots, 2$), one can write $m = n^k$ for the *maximal* $k \in \{1, 2, \ldots, \lfloor \log_2 m \rfloor\}$ such that m has an integer k-th root, also taking $O(\log^{3+o(1)} m)$ bit operations as above. The problem is then reduced to checking whether n is a prime, which takes $O(\log^{6+o(1)} n) \subseteq O(\log^{6+o(1)} m)$ bit operations by statement (9) of Lemma 5.1.5. In summary, we have a deterministic routine with complexity in $O(\log^{6+o(1)} m)$ for deciding whether m is a prime power and, if so, writing it as such.

Shor's general Las Vegas dual approach for factoring $m \in \mathbb{N}^+$ using reduction ideas of Miller is outlined in [37, pp. 132f.]. We start by splitting off the factor $2^{\nu_2(m)}$ from m. Because m is given in its binary representation, this only takes $O(\log^{1+o(1)} m)$ bit operations, accounting for $O(\nu_2(m)) \subseteq O(\log m)$ increases of a counter that remains in $O(\log m)$ throughout (and thus has $O(\log \log m) \subseteq O(\log^{o(1)} m)$ bits). Now, we set $m' := m/2^{\nu_2(m)}$. In the rest of our proof of statement (1), we will

only be dealing with *odd* positive integers. We describe a Las Vegas routine that decides whether a given odd positive integer n is a prime power, then does the following:

- if n is a prime power, it writes n as $p^{v_p(n)}$;
- if n is not a prime power, it finds a factorization of n of the form $n = n' \cdot n''$, where $1 < n', n'' < n$.

We already described above how to decide whether $n = p^{v_p(n)}$ and, if so, write it as such using $O(\log^{6+o(1)} n)$ bit operations, so we start by applying that routine and may henceforth assume that it returned that n is *not* a prime power. Following [37, p. 133], we wish to draw an integer $x \in \{2, 3, \ldots, n-1\}$ uniformly at random. Letting $N := \lfloor \log_2(n-3) \rfloor + 1$, we aim to draw the N-bit integer $y \in \{0, 1, \ldots, n-3\}$ uniformly at random, then set $x := y + 2$. To draw y, we initialize an N-qubit register to $|0\rangle^{\otimes N}$, then pass it through an N-dimensional Hadamard circuit (with elementary gate complexity $N \in O(\log n)$) to get a uniform superposition of all N-bit strings, so that a simple measurement returns (the binary representation of) a random integer in $\{0, 1, \ldots, 2^N - 1\}$. The probability that this integer lies in the range for y is at least $1/2$, so we only need to iterate this procedure an expected number of $O(1)$ times until we get a suitable value for y, using $O(\log n)$ bit operations and elementary quantum gates as well as $O(1)$ conversions to and from $O(\log n)$-bit strings. Following that, we compute $x = y + 2$ and $\gcd(x, n)$, taking $O(\log^{1+o(1)} n)$ bit operations by statements (1) and (8) of Lemma 5.1.5. If $\gcd(x, n) > 1$, we may output the factorization $n = n' \cdot n''$ with $n' = \gcd(x, n)$, taking just another $O(\log^{1+o(1)} n)$ bit operations to compute n'' by division, and are done. Otherwise, x is a (uniformly random) unit modulo n, and we proceed to apply the order-finding routine from [37, p. 137] to get an output o which is either a number or the string "FAIL", and is equal to $\mathrm{ord}(x)$ with probability at least 0.399. If o is "FAIL", we repeat this routine on the same value of x until we get an output that is actually a number (only $O(1)$ repetitions needed by expectancy). Then, if o is *not* even, we abandon this value of x and choose y anew (because we want $2 \mid \mathrm{ord}(x)$, and even if o may not be equal to $\mathrm{ord}(x)$, it is an integer multiple of $\mathrm{ord}(x)$, as was noted above) until we get an x such that either $\gcd(x, n) > 1$ or the associated alleged multiplicative order $o \in \mathbb{N}^+$ is even. This, too, only requires an expected number of $O(1)$ attempts, because for a randomly selected $x \in (\mathbb{Z}/n\mathbb{Z})^*$, the order of x is even with probability at least $1/2$. We then compute $z := x^{o/2} \bmod n$, taking $O(\log^{2+o(1)} n)$ bit operations by statement (6) of Lemma 5.1.5. Because n is not a prime power, we have $\gcd(z-1, n) > 1$ with probability at least $1/2$, so after expectedly $O(1)$ more tries, we will indeed have found a nontrivial factorization of n. Taking into account the complexity of the order-finding routine from [37, p. 137] which we analyzed above, this process expectedly takes $O(\log^{6+o(1)} n)$ bit operations, $O(\log^{2+o(1)} n)$ elementary quantum gates, and $O(1)$ conversions to and from $O(\log n)$-bit strings.

Let us now return to our problem of factoring the odd positive integer $m' = m/2^{v_2(m)}$. Through iteratively applying the factor-finding routine we just described, which needs to be applied $O(\log m)$ times, statement (1) follows (the bit operation cost of some necessary deterministic post-processing, such as adding the exponents of primes appearing in multiple obtained factors, is clearly subsumed under $O(\log^{7+o(1)} m)$).

For statement (2), we assume given a modulus m and a unit $x \in (\mathbb{Z}/m\mathbb{Z})^*$, and we wish to give a *Las Vegas* algorithm that computes $\mathrm{ord}(x)$. For this, we first factor m, using the m-bounded Las Vegas dual complexity $(\log^{7+o(1)} m, \log^{3+o(1)} m, \log m)$ by statement (1). Knowing the factorization of m allows us to compute the Euler totient function value $\phi(m)$ and a factorization thereof within the same m-bounded Las Vegas dual complexity. Now, for each prime $p \mid \phi(m)$, we can work out $v_p(\mathrm{ord}(x))$ as the smallest $v_p \in \{0, 1, \ldots, v_p(\phi(m))\}$ such that

$$x^{p^{v_p}\phi(m)_{p'}} \equiv 1 \pmod{m},$$

where $\phi(m)_{p'} := \phi(m)/p^{v_p(\phi(m))}$. More specifically, we can use a binary search for finding $v_p(\mathrm{ord}(x))$, which according to statement (6) of Lemma 5.1.5 results in a cost of

$$O(\log\log m \cdot \log^{2+o(1)} m) = O(\log^{2+o(1)} m)$$

bit operations for finding $v_p(\mathrm{ord}(x))$ for a single p, hence of $O(\log^{3+o(1)} m)$ bit operations for finding all of these valuations. Finally, we compute $\mathrm{ord}(x)$ itself as the product of all prime powers $p^{v_p(\mathrm{ord}(x))}$, where p ranges over the prime divisors of $\phi(m) \in \mathrm{ord}(x)\mathbb{Z}$. This takes another $O(\log^{3+o(1)} m)$ bit operations, thus proving statement (2).

For the proofs of statements (3) and (4), we need some preparations again. In [37, Section 7.4], a general approach for computing discrete logarithms, working for elements chosen from any black-box group G with a unique encoding of each element, is discussed. For given $x, y \in G$ such that $y = x^t$ for some $t \in \mathbb{Z}$ and the order of x in G is known, this approach returns with high probability the unique $t \in \{0, 1, \ldots, \mathrm{ord}(x) - 1\}$ such that $y = x^t$, which is called the *discrete logarithm (in G) of y with base x*, written $\log_x(y)$. However, if y is *not* a power of x in G, it seems that this approach does not provide a means of confirming this *with certainty*, as is required for the Las Vegas algorithms we desire. This means that in addition to the discrete logarithm algorithm from [37, Section 7.4], we need a Las Vegas routine for checking whether y is a power of x in the first place. We analyze these algorithms one after the other, starting with the routine for computing $\log_x(y)$ in case y is a power of x for $G = (\mathbb{Z}/m\mathbb{Z})^*$ or $G = \mathbb{F}_q^*$. As in Section 2.4, we extend the notation $\log_x(y)$ to arbitrary $x, y \in G$ by setting $\log_x(y) := \infty$ if y is *not* a power of x.

An important observation is that the discrete logarithm algorithm described in [37, p. 144] only works if $\mathrm{ord}(x)$ is a prime (see [37, second paragraph after formula (7.4.2.) on p. 143]). For the general case, we follow "Method 1" from [37, pp. 244f., starting after Corollary A.2.2]. We start by setting $m_0 := 1$ and $\mathrm{rem}_0 := 0$. Then, certainly, $\log_x(y) \equiv \mathrm{rem}_0 \pmod{m_0}$. The aim is to recursively define integers

$$m_1 \mid m_2 \mid \cdots \mid m_N = \mathrm{ord}(x)$$

and $\mathrm{rem}_1, \mathrm{rem}_2, \ldots, \mathrm{rem}_N$ with $\mathrm{rem}_N = \log_x(y)$ such that $\log_x(y) \equiv \mathrm{rem}_k \pmod{m_k}$ throughout. As noted in [37, p. 244, right after Corollary A.2.2], running the algorithm from [37, p. 144] allows us to work out m_{k+1} and rem_{k+1} from $m_k < \mathrm{ord}(x)$ and rem_k with high probability (and no risk of getting incorrect values for them, only "FAIL"). In each step, this involves

- $O(1)$ arithmetic operations covered in statements (1–8) of Lemma 5.1.5, which account for $O(\log^{2+o(1)} m)$ bit operations if $G = (\mathbb{Z}/m\mathbb{Z})^*$, respectively, for $O(\log^{2+o(1)} q)$ bit operations if $G = \mathbb{F}_q^*$;
- $O(\log^{2+o(1)} m)$, respectively, $O(\log^{2+o(1)} q)$, elementary quantum gates; and
- $O(1)$ conversions to and from $O(\log m)$-bit, respectively, $O(\log q)$-bit, strings.

As noted in [37, p. 245], the number N of iterations of this loop is in $O(\log \mathrm{ord}(x))$, and thus in $O(\log m)$, respectively, $O(\log q)$. Therefore, we can compute $\log_x(y)$ in case it is not ∞ and $\mathrm{ord}(x)$ is known using the following m-bounded, respectively, q-bounded, Las Vegas dual complexity:

- $(\log^{3+o(1)} m, \log^{3+o(1)} m, \log m)$ if $G = (\mathbb{Z}/m\mathbb{Z})^*$;
- $(\log^{3+o(1)} q, \log^{3+o(1)} q, \log q)$ if $G = \mathbb{F}_q^*$.

This concludes our preparation for the proofs of statements (3) and (4).

For statement (3), we assume that $x, y \in (\mathbb{Z}/m\mathbb{Z})^*$ are given. In order to compute $\log_x(y)$, we first check whether y is a power of x in the first place. We start by factoring m, which takes m-bounded Las Vegas dual complexity

$$(\log^{7+o(1)} m, \log^{3+o(1)} m, \log m)$$

by statement (1). For a prime divisor p of m, we set $v_p := v_p(m)$ and $m_p := p^{v_p}$. We wish to compute $\log_x^{(m_p)}(y)$, the discrete logarithm modulo m_p of y with base x, for each prime $p \mid m$, and in the following two paragraphs, we describe how to do so. We use the notation $\mathrm{ord}_n(z)$ to denote the multiplicative order of z modulo n.

First, we assume that $p > 2$. We compute $\mathrm{ord}_{m_p}(x)$ and $\mathrm{ord}_{m_p}(y)$, taking m_p-bounded Las Vegas dual complexity $(\log^{7+o(1)} m_p, \log^{3+o(1)} m_p, \log m_p)$ by statement (2). Because the unit group $(\mathbb{Z}/m_p\mathbb{Z})^*$ is cyclic, we have that y is a power of x modulo m_p if and only if $\mathrm{ord}_{m_p}(y)$ divides $\mathrm{ord}_{m_p}(x)$. By statement (3) of Lemma 5.1.5, it only takes $O(\log^{1+o(1)} m_p)$ bit operations to check this. If this

divisibility does *not* hold, then $\log_x^{(m_p)}(y)$ is ∞, and so is $\log_x^{(m)}(y)$, so we are done. Otherwise, we compute $\log_x^{(m_p)}(y)$ using the Las Vegas routine from [37, p. 144 and Appendix A.2], which takes m_p-bounded Las Vegas dual complexity $(\log^{3+o(1)} m_p, \log^{3+o(1)} m_p, \log m_p)$; we note that at this point, we do know $\operatorname{ord}_{m_p}(x)$ because it was computed beforehand.

Now we assume that $p = 2$. We proceed in a similar manner to when $p > 2$, namely by first checking whether $\log_x^{(m_2)}(y)$ is ∞ and, if not, computing its precise integer value at the cost of an m_2-bounded Las Vegas dual complexity of

$$(\log^{3+o(1)} m_2, \log^{3+o(1)} m_2, \log m_2),$$

or $(\log^{7+o(1)} m_2, \log^{3+o(1)} m_2, \log m_2)$ if $\operatorname{ord}_{m_2}(x)$ has not been computed at that point. Checking whether $\log_x^{(m_2)}(y) = \infty$ is a bit more complicated than for $p > 2$, though, because $(\mathbb{Z}/m_2\mathbb{Z})^*$ is not necessarily cyclic. We may assume that $m_2 > 2$ (otherwise, $\log_x^{(m_2)}(y)$ is simply equal to 1), and we distinguish some cases.

- If $x \equiv y \equiv 1 \pmod 4$, which only takes $O(1)$ bit operations to check because x and y are given in binary, then x and y both lie in the cyclic subgroup of $(\mathbb{Z}/m_2\mathbb{Z})^*$ generated by the unit 5 (which is equal to the unit 1 if $m_2 = 4$). Therefore, just as for $p > 2$, we have that y is a power of x if and only if $\operatorname{ord}_{m_2}(y) \mid \operatorname{ord}_{m_2}(x)$.
- If $x \equiv 1 \pmod 4$ and $y \equiv 3 \pmod 4$, then y cannot be a power of x modulo m_2.
- If $x \equiv 3 \pmod 4$ and $y \equiv 1 \pmod 4$, then y is a power of x modulo m_2 if and only if y is a power of x^2 modulo m_2. Because $x^2 \equiv 1 \pmod 4$, we conclude that y is a power of x modulo m_2 if and only if $\operatorname{ord}_{m_2}(y) \mid \operatorname{ord}_{m_2}(x^2)$.
- If $x \equiv y \equiv 3 \pmod 4$, then y is a power of x modulo m_2 if and only if $-y$ is a power with odd exponent of $-x$ modulo m_2. Therefore, in order to check whether y is a power of x modulo m_2, we first check whether $\operatorname{ord}_{m_2}(-y) \mid \operatorname{ord}_{m_2}(-x)$. If not, then y is certainly *not* a power of x modulo m_2. Otherwise, we compute $\log_{-x}^{(m_2)}(-y)$ and check whether it is odd.

In summary, since

$$\sum_{p \mid m} \log^k m_p \in O(\log^k m)$$

for all real exponents $k \geq 1$, we conclude that computing $\log_x^{(m_p)}(y)$ for each prime $p \mid m$ takes m-bounded Las Vegas dual complexity $(\log^{7+o(1)} m, \log^{3+o(1)} m, \log m)$ for all p together. As noted above, if any of these "primary discrete logarithms" is ∞, then y is *not* a power of x modulo m, i.e., $\log_x^{(m)}(y) = \infty$, and we are done. Otherwise, noting that for each integer t, we have

$$y \equiv x^t \pmod m \text{ if and only if } y \equiv x^t \pmod{m_p} \text{ for all primes } p \mid m,$$

we find that $\log_x^{(m)}(y) < \infty$ if and only if the system of congruences

$$t \equiv \log_x^{(m_p)}(y) \pmod{\operatorname{ord}_{m_p}(x)} \text{ for all primes } p \mid m$$

in the variable t is consistent, in which case $\log_x^{(m)}(y)$ is its unique solution between 0 and $\operatorname{lcm}(\{\operatorname{ord}_{m_p}(x) : p \mid m\}) - 1 = \operatorname{ord}_m(x) - 1$. We already computed the right-hand sides and moduli of this system of congruences, except possibly $\operatorname{ord}_{m_2}(x)$, which takes m-bounded Las Vegas dual complexity $(\log^{7+o(1)} m, \log^{3+o(1)} m, \log m)$ to compute. Following that, we can check the consistency of this system using the equivalence of statements (1) and (2) in Proposition 2.2.1, which requires $O(\log m)$ subtractions, gcd computations and divisions of integers less than m, hence can be done using $O(\log^{2+o(1)} m)$ bit operations. Should the system be consistent, the integer value of $\log_x^{(m)}(y)$ may be determined by solving the system (which is deterministic and can certainly be done with $O(\log^{7+o(1)} m)$ bit operations, but we do not go into the details of this here), or by computing $\operatorname{ord}_m(x)$ and running the aforementioned routine from [37, p. 144 and Appendix A.2] for another m-bounded Las Vegas dual complexity of $(\log^{7+o(1)} m, \log^{3+o(1)} m, \log m)$. This concludes the proof of statement (3).

For statement (4), we assume that $x, y \in \mathbb{F}_q^*$ are given. In accordance with the two formats (specified at the beginning of this section) in which generalized cyclotomic mappings may be given, we consider two distinct versions of the computational problem of finding $\log_x(y)$:

- version 1: x, y are given as powers of a common, unspecified primitive element ω of $\mathbb{F}_q$;
- version 2: a primitive irreducible polynomial $P(T)$ over $\mathbb{Z}/p\mathbb{Z}$ of degree $\log_p q$ is known, and $\mathbb{F}_q$ is to be viewed as $(\mathbb{Z}/p\mathbb{Z})[T]/(P(T))$, with x, y given as elements of this quotient ring in standard form (i.e., as polynomials over $\mathbb{Z}/p\mathbb{Z}$ in the variable T and of degree less than $\log_p q$).

In either scenario, we show that one can check whether y is a power of x and, if so, work out the integer value of $\log_x(y)$ using q-bounded Las Vegas dual complexity $(\log^{3+o(1)} q, \log^{3+o(1)} q, \log q)$ altogether. Indeed, we note that for any given primitive element $\omega \in \mathbb{F}_q$, the function $\log_\omega : \mathbb{F}_q^* \to \mathbb{Z}/(q-1)\mathbb{Z}$ is a group isomorphism. In the first version of the computational problem, an (unspecified) value for ω was already fixed, and in the second version, we set $\omega := T$ (viewed as an element of $(\mathbb{Z}/p\mathbb{Z})[T]/(P(T))$). In either case, we can work out $\log_\omega(x)$ and $\log_\omega(y)$; in the first scenario, x and y are literally given as powers of ω, and in the second scenario, we apply the routine from [37, p. 144 and Appendix A.2] to compute $\log_\omega(x)$ and $\log_\omega(y)$, which works because x and y are powers of ω and we know that $\operatorname{ord}(\omega) = q - 1$. In either case, those two discrete logarithms can be computed with q-bounded Las Vegas dual complexity $(\log^{3+o(1)} q, \log^{3+o(1)} q, \log q)$ (in fact, in

the first scenario, they only need to be copied from the input, requiring $O(\log q)$ bit operations only).

After finding $\log_\omega(x)$ and $\log_\omega(y)$, the problem is reduced to checking whether $\log_\omega(y)$ is a multiple of $\log_\omega(x)$ in $\mathbb{Z}/(q-1)\mathbb{Z}$, i.e., whether

$$\log_\omega(y) \equiv k \cdot \log_\omega(x) \pmod{q-1} \tag{5.1}$$

for some $k \in \mathbb{Z}$. But this is the case if and only if $\gcd(\log_\omega(x), q-1) \mid \log_\omega(y)$, which can be checked using $O(\log^{1+o(1)} q)$ bit operations. If so, then $\log_x(y)$ is the unique solution $k \in \{0, 1, \ldots, q-2\}$ of congruence (5.1), that is, modulo $q-1$

$$\log_x(y) = \frac{\log_\omega(y)}{\gcd(\log_\omega(x), q-1)} \cdot \mathrm{inv}_{\frac{q-1}{\gcd(\log_\omega(x), q-1)}} \left(\frac{\log_\omega(x)}{\gcd(\log_\omega(x), q-1)} \right),$$

which can be evaluated with a final batch of $O(\log^{1+o(1)} q)$ bit operations. This concludes the proof of statement (4).

For statement (5), we first aim to find a primitive root modulo p. A polynomial-time probabilistic classical algorithm for doing so is [22, Algorithm 1], which is a refinement of an earlier algorithm by Bach [10], itself based on Itoh's idea of using partial factorizations of $p-1$ to find a primitive root with high probability. We follow this approach, but since we can factor $p-1$ completely, our situation is easier. We start by factoring $p-1$, taking p-bounded Las Vegas dual complexity $(\log^{7+o(1)} p, \log^{3+o(1)} p, \log p)$ by statement (1). Say $p - 1 = \prod_{j=1}^{K} p_j^{v_j}$ is the said factorization. For each prime divisor p_j of $p-1$, we wish to find a unit $y_j \in (\mathbb{Z}/p\mathbb{Z})^* = \{1, 2, \ldots, p-1\}$ such that $p_j^{v_j}$ divides $\mathrm{ord}_p(y_j)$. Equivalently, y_j should *not* be a p_j-th power in $\mathbb{Z}/p\mathbb{Z}$. The proportion of units that satisfy this is $1 - \frac{1}{p_j} \geq \frac{1}{2}$, so if we pick $y_j \in (\mathbb{Z}/p\mathbb{Z})^*$ at random, then check whether $y_j^{(p-1)/p_j} \not\equiv 1 \pmod{p}$, it only takes an expected number of $O(1)$ tries until we succeed at finding y_j. As in the proof of statement (1), we perform this random drawing of y_j using a $(\lfloor \log_2(p-2) \rfloor + 1)$-qubit Hadamard circuit. This means that the expected p-bounded Las Vegas dual complexity of finding y_j is $(\log^{2+o(1)} p, \log p, 1)$ for a single j, and $(\log^{3+o(1)} p, \log^2 p, \log p)$ for all $j = 1, 2, \ldots, K$ together. Once the y_j have been found, the unit

$$\mathrm{r} := \prod_{j=1}^{K} y_j^{(p-1)/p_j^{v_j}} \in (\mathbb{Z}/p\mathbb{Z})^*,$$

which can be computed with an additional $O(\log^{3+o(1)} p)$ bit operations, is a primitive root modulo p. Indeed, the j-th factor in this product has order $p_j^{v_j}$, and because the numbers $p_j^{v_j}$ are pairwise coprime and the group $(\mathbb{Z}/p\mathbb{Z})^*$ is abelian, this entails that $\mathrm{ord}_p(\mathrm{r}) = \prod_{j=1}^{K} p_j^{v_j} = p - 1$.

From r, it is not difficult to construct a primitive root r^+ modulo p^k. Indeed, if $k = 1$, we just set $r^+ := r$. Moreover, a primitive root modulo p^2 is a primitive root modulo p^k for each $k \geq 2$, and either r or $r + p$ is a primitive root modulo p^2. Hence, if $k > 1$, we simply check whether $r^{p-1} \equiv 1 \pmod{p^2}$, taking $O(\log^{2+o(1)} p^2) = O(\log^{2+o(1)} p)$ bit operations. If so, we set $r^+ := r + p$, otherwise we set $r^+ := r$. This concludes the proof of statement (5) and of Lemma 5.1.6 as a whole. ∎

Our first application of Lemma 5.1.6 is the following aforementioned result on converting a query complexity into a Las Vegas dual complexity.

Lemma 5.1.7. *Let $\mathfrak{L}$ be an algorithmic problem, and let $y, y_1, \ldots, y_n$ be non-negative real parameters associated with the admissible inputs for $\mathfrak{L}$ such that*

$$y(\vec{x}) \leq h(y_1(\vec{x}), \ldots, y_n(\vec{x}))$$

for all $\vec{x} \in \mathfrak{L}_{\text{in}}$, where h is a fixed function $[0, \infty)^n \to [0, \infty)$. Moreover, let

$$\vec{\mathcal{C}}^{\text{qry}} = (\mathcal{C}_{\text{class}}, \mathcal{C}_{\text{fdl}}, \mathcal{C}_{\text{mdl}}, \mathcal{C}_{\text{mord}}, \mathcal{C}_{\text{prt}})$$

be a y-bounded query complexity of $\mathfrak{L}$ with respect to $y_1, \ldots, y_n$. Then

$$\vec{\mathcal{C}} = (\mathcal{C}'_{\text{class}}, \mathcal{C}_{\text{quant}}, \mathcal{C}_{\text{conv}}),$$

with $\mathcal{C}'_{\text{class}}, \mathcal{C}_{\text{quant}}, \mathcal{C}_{\text{conv}} : [0, \infty)^n \to [0, \infty)$ as defined below, is a y-bounded Las Vegas dual complexity of $\mathfrak{L}$. We write $\vec{z}$ shorthand for $(z_1, z_2, \ldots, z_n) \in [0, \infty)^n$.

$$\begin{aligned}
\mathcal{C}'_{\text{class}}(\vec{z}) &:= \mathcal{C}_{\text{class}}(\vec{z}) + \log^{7+o(1)}(h(\vec{z})) \cdot (\mathcal{C}_{\text{mdl}}(\vec{z}) + \mathcal{C}_{\text{mord}}(\vec{z}) + \mathcal{C}_{\text{prt}}(\vec{z})) \\
&\quad + \log^{3+o(1)}(h(\vec{z}))\mathcal{C}_{\text{fdl}}(\vec{z}); \\
\mathcal{C}_{\text{quant}}(\vec{z}) &:= \log^{3+o(1)}(h(\vec{z})) \cdot (\mathcal{C}_{\text{mdl}}(\vec{z}) + \mathcal{C}_{\text{fdl}}(\vec{z}) + \mathcal{C}_{\text{mord}}(\vec{z}) + \mathcal{C}_{\text{prt}}(\vec{z})); \\
\mathcal{C}_{\text{conv}}(\vec{z}) &:= \log(h(\vec{z})) \cdot (\mathcal{C}_{\text{mdl}}(\vec{z}) + \mathcal{C}_{\text{fdl}}(\vec{z}) + \mathcal{C}_{\text{mord}}(\vec{z}) + \mathcal{C}_{\text{prt}}(\vec{z})).
\end{aligned}$$

Proof. This follows easily from Lemma 5.1.6 and the definitions of the involved concepts. For example, when computing $\mathcal{C}'_{\text{class}}(\vec{z})$, we not only have to take into account the bit operations spent outside the special queries, which are represented by the summand $\mathcal{C}_{\text{class}}(\vec{z})$, but also those coming from the queries, using the algorithms discussed in the proofs of Lemma 5.1.6 to fulfill those queries. For example, on input $\vec{x}$, each of the $O(\mathcal{C}_{\text{fdl}}(y_1(\vec{x}), \ldots, y_n(\vec{x})))$ finite field discrete logarithm queries needed in the course of computing an admissible output for $\vec{x}$ is about computing a discrete logarithm in a finite field of size at most $h(y_1(\vec{x}), \ldots, y_n(\vec{x}))$. Therefore, by Lemma 5.1.6 (4), these "fdl queries" together account for

$$O(\mathcal{C}_{\text{fdl}}(y_1(\vec{x}), \ldots, y_n(\vec{x})) \cdot \log^{3+o(1)}(h(y_1(\vec{x}), \ldots, y_n(\vec{x}))))$$

bit operations, whence the inclusion of the summand $\mathcal{C}_{\mathrm{fdl}}(\vec{z}) \cdot \log^{3+o(1)}(h(\vec{z}))$ in the definition of $\mathcal{C}'_{\mathrm{class}}(\vec{z})$. For the other three kinds of queries, one can use Lemma 5.1.6 together with the fact that

$$\sum_{p|m} \log^t (p^{v_p(m)}) \in O(\log^t m)$$

for each positive integer m and each real exponent $t \geq 1$. For example, for a "prt query", we need to find a primitive root modulo $p^{v_p(m)}$ for each odd prime p dividing m. We do so by first factoring m, then applying the algorithm from Lemma 5.1.6 (5). By Lemma 5.1.6 (1,5), the number of bit operations needed in the process is in

$$O\Big(\log^{7+o(1)} m + \sum_{2<p|m} \log^{7+o(1)} p\Big) \subseteq O\Big(\log^{7+o(1)} m + \sum_{p|m} \log^{7+o(1)} \big(p^{v_p(m)}\big)\Big)$$
$$= O(\log^{7+o(1)} m) \subseteq O(\log^{7+o(1)} y(\vec{x})) \subseteq O(\log^{7+o(1)} h(y_1(\vec{x}), \ldots, y_n(\vec{x}))),$$

which also subsumes the cost of computing $p^{v_p(m)}$ from p and $v_p(m)$ for all p by Lemma 5.1.5 (6). The number of elementary quantum gates, respectively of bit-qubit conversions, needed in the process may be dealt with analogously. Moreover, an analogous approach works for "mdl queries" and "mord queries", where one also needs to factor m (see Lemma 5.1.6 (1)) and (due to the upper bound of $y(\vec{x})^2$ on m from Definition 5.1.3 (4,a)) ends up with an argument of $h(y_1(\vec{x}), \ldots, y_n(\vec{x}))^2$ in the logarithm power, but this may be replaced by $h(y_1(\vec{x}), \ldots, y_n(\vec{x}))$ without changing the O-class of the overall expression. ∎

In view of Lemma 5.1.7, we mostly work with query complexities from here on, only converting them to Las Vegas dual complexities in some main results. In order to solve the three algorithmic problems on index d generalized cyclotomic mappings f from the beginning of this section using the theory developed in this memoir, we first need to compute the induced function $\bar{f} : \{0, 1, \ldots, d\} \to \{0, 1, \ldots, d\}$ and, for each $i \in \{0, 1, \ldots, d-1\}$ such that the coefficient a_i in the cyclotomic form (1.1) of f is non-zero, we need to compute the affine map A_i of $\mathbb{Z}/s\mathbb{Z}$ that encodes the restriction $f_{|C_i} : C_i \to C_{\bar{f}(i)}$ under the identification of C_j with $\mathbb{Z}/s\mathbb{Z}$ via the bijection ι_j described in our introduction. Our next goal is to analyze the query complexity of these tasks.

Proposition 5.1.8. *Given f, one can compute the induced function $\bar{f}$ and the associated affine maps A_i with q-bounded query complexity*

$$(d \log^{1+o(1)} q, d, 0, 0, 0).$$

Proof. With regard to $\bar{f}$, we know that $\bar{f}(d) = d$, so only the values $\bar{f}(i)$ for $i \in \{0, 1, \ldots, d-1\}$ need to be computed. These are $O(d)$ cases. By our discussion in

the introduction, we have $\bar{f}(i) = (e_i + r_i i) \bmod d$, where $e_i = \log_\omega(a_i)$, and by our assumptions from the beginning of this section, this discrete logarithm is either directly specified with a_i, or we compute it with a (finite) field discrete logarithm (fdl) query. After computing e_i, it takes another $O(\log^{1+o(1)} q)$ bit operations by Lemma 5.1.5 (1,3) to evaluate $(e_i + r_i i) \bmod d$ and thus compute $\bar{f}(i)$. In total, a q-bounded query complexity of computing $\bar{f}$ is

$$(d \log^{1+o(1)} q, d, 0, 0, 0).$$

Once $\bar{f}$ has been determined, the computation of the A_i is easy; for each of the $O(d)$ values of i in question, we note that $A_i(x) = \alpha_i x + \beta_i$ for all $x \in \mathbb{Z}/s\mathbb{Z}$, where $\alpha_i, \beta_i \in \mathbb{Z}/s\mathbb{Z}$ are constants. Computing A_i just means computing α_i and β_i, and by the discussion in our introduction, we have

$$\alpha_i = r_i, \quad \beta_i = \frac{e_i + r_i i - \bar{f}(i)}{d} \bmod s.$$

We can directly read off r_i from the definition (1.1) of f, and computing β_i takes $O(\log^{1+o(1)} q)$ bit operations by Lemma 5.1.5 (1,3). Therefore, computing all α_i and β_i after $\bar{f}$ has been worked out takes $O(d \log^{1+o(1)} q)$ bit operations, and the result follows. ∎

With regard to Problem 3 from the beginning of this section, we note that solving this problem efficiently provides us with a quick understanding of each given connected component of Γ_f. While it would be more desirable to have an efficient algorithm that achieves a *global* understanding, of the isomorphism types of all connected components of Γ_f, it is not even clear what the output of such an algorithm would look like. In Definition 5.3.2.8, we introduce the concept of a tree necklace list, which is a way to list the isomorphism types of all connected components of Γ_f with their multiplicities. While such a tree necklace list is a compact encoding of the isomorphism type of Γ_f in *some* cases (e.g., the ones considered in Sections 5.3.2 and 5.3.3), it is not clear whether that is always the case; see also the discussion after Remark 5.3.2.9.

The following main result of this chapter provides q-bounded query and Las Vegas dual complexities of the three algorithmic problems from the beginning of this section. We recall that $\mathrm{mpe}(q-1) = \max_{p|q-1} \nu_p(q-1)$ denotes the maximum exponent of a prime in the prime factorization of $q-1$.

Theorem 5.1.9. *The following hold with regard to the three algorithmic problems from the beginning of this section.*

(1) *Problem 1 has q-bounded query complexity*

$$(d \log^2 d + d^2 \log^{1+o(1)} q + d \log^{2+o(1)} q + \log^{3+o(1)} q, d, d \log q, d, 1)$$

and q-bounded Las Vegas dual complexity

$$(d\log^2 d + d^2\log^{1+o(1)} q + d\log^{8+o(1)} q, d\log^{4+o(1)} q, d\log^2 q).$$

(2) *Problem 2 has q-bounded query complexity*

$$(d^3\,\mathrm{mpe}(q-1)2^{(3d^2+d)\,\mathrm{mpe}(q-1)+2d}\log^{1+o(1)} q, d, 0, 0, 0)$$

and q-bounded Las Vegas dual complexity

$$(d^3\,\mathrm{mpe}(q-1)2^{(3d^2+d)\,\mathrm{mpe}(q-1)+2d}\log^{1+o(1)} q + d\log^{3+o(1)} q,$$
$$d\log^{3+o(1)} q, d\log q).$$

Moreover, the computed tree-partition register can be chosen such that its underlying recursive tree description list is of length in

$$O(\min\{d2^{d^2\,\mathrm{mpe}(q-1)+d}, q\}),$$

with each tree description from the list being itself a list of length in

$$O(\min\{d2^{d^2\,\mathrm{mpe}(q-1)+d}, q\}),$$

each entry of which is an ordered pair of bit length in $O(\log q)$.

(3) *Problem 3 has q-bounded query complexity*

$$(d\log^{3+o(1)} q + d^3\,\mathrm{mpe}(q-1)\log^{1+o(1)} q$$
$$+ d^3\,\mathrm{mpe}(q-1)2^{d^2\,\mathrm{mpe}(q-1)+d}\log q, d, d^3\,\mathrm{mpe}(q-1), d^3\,\mathrm{mpe}(q-1), 0)$$

and q-bounded Las Vegas dual complexity

$$(d^3\,\mathrm{mpe}(q-1)\log^{7+o(1)} q + d^3\,\mathrm{mpe}(q-1)2^{d^2\,\mathrm{mpe}(q-1)+d}\log q,$$
$$d^3\,\mathrm{mpe}(q-1)\log^{3+o(1)} q, d^3\,\mathrm{mpe}(q-1)\log q).$$

We prove Theorem 5.1.9 in Section 5.2. Before that, we make some more comments on the complexities of the three algorithmic problems in Theorem 5.1.9. Firstly, while we are mostly focused on quantum complexities in this memoir, we note that knowing the query complexity of an algorithmic problem also allows one to deduce a bound on its classical Las Vegas complexity. Indeed, all four kinds of queries we are concerned with admit rigorously subexponential solution algorithms on classical computers: for integer factorization, and thus (following the proof of Lemma 5.1.6 (2)) modular multiplicative orders, as well as discrete logarithms in finite fields, see Pomerance's paper [60]. For modular discrete logarithms, see Adleman's

extended abstract [1]. Finally, primitive roots may be dealt with using Dubrois–Dumas' algorithm from [22], combined with modular multiplicative order computations to confirm the alleged primitive root produced by the Dubrois–Dumas algorithm. The upshot of this is that as long as all components of one the q-bounded query complexities in Theorem 5.1.9 are subexponential in the input size (which lies in $O(d \log q)$), then the corresponding problem admits a rigorously subexponential classical Las Vegas solution algorithm.

Secondly, considering the said components of the q-bounded query complexities in Theorem 5.1.9, a glaring question is how the inclusion of the parameter $\mathrm{mpe}(q-1)$ in some components in statements (2) and (3) affects their size. At first glance, this seems rather bad, because generally $\mathrm{mpe}(q-1) \leq \lfloor \log_2(q-1) \rfloor \sim \log_2 q$, and this bound is attained whenever q is a Fermat prime. Since in statements (2) and (3) of Theorem 5.1.9, $\mathrm{mpe}(q-1)$ occurs in the exponent of a power with base 2, this means that in the worst case, the given complexities for Problems 2 and 3 are exponential in the input length for fixed d.

That being said, it turns out that "most of the time", $\mathrm{mpe}(q-1)$ is actually bounded from above by a suitably large constant, as the following result states. This result and its proof was kindly pointed out by MathOverflow user "Dr. Pi" in a response to a question posted by the first author on MathOverflow[1].

Proposition 5.1.10. *There is an absolute constant $c_{\mathrm{mpe}} > 0$ such that for all $x \geq 2$, one has*

$$\Big(\sum_{q \leq x} 1\Big)^{-1} \sum_{q \leq x} \mathrm{mpe}(q-1) \leq c_{\mathrm{mpe}},$$

where the variable q ranges over prime powers. In particular, the following hold.

(1) *For each $\varepsilon > 0$, there is a constant $c_\varepsilon > 0$ such that for all prime powers q except an asymptotic fraction of less than ε, one has $\mathrm{mpe}(q-1) < c_\varepsilon$.*

(2) *Let $h : [0,\infty) \to [0,\infty)$ be a function such that $h(x) \to \infty$ as $x \to \infty$. Then for asymptotically almost all prime powers q, one has $\mathrm{mpe}(q-1) \leq h(q)$.*

Proof. We start by observing that the number of proper (i.e., non-prime) prime powers up to x is asymptotically equivalent to $2x^{1/2}/\log x$. Indeed, the number of prime *squares* up to x is

$$\pi(x^{1/2}) \sim \frac{x^{1/2}}{\log(x^{1/2})} = \frac{2x^{1/2}}{\log x}.$$

Moreover, a prime power $p^k \leq x$ with $k \geq 3$ satisfies $k \leq \lfloor \log_2 x \rfloor$, and for each fixed k, the number of such prime powers is at most $x^{1/k} \leq x^{1/3}$. Hence the number

[1] see https://mathoverflow.net/questions/436134/average-value-of-the-prime-omega-function-omega-on-predecessors-of-prime-powe, visited on 2 September 2025.

of all proper prime powers up to x is

$$\pi(x^{1/2}) + O(x^{1/3}\log x) \sim \frac{2x^{1/2}}{\log x}.$$

This entails the following two things.

(1) The number $\sum_{q\le x} 1$ of *all* prime powers up to x is asymptotically equivalent to $x/\log x$, same as $\pi(x)$.

(2) In the sum $\sum_{q\le x} \mathrm{mpe}(q-1)$, the total contribution stemming from *proper* prime powers is at most

$$O\Big(\log x \cdot \frac{2x^{1/2}}{\log x}\Big) = O(x^{1/2}) \subseteq o\Big(\sum_{q\le x} 1\Big).$$

We may thus focus on the contribution $\sum_{p\le x} \mathrm{mpe}(p-1)$ stemming from primes.

For each $v = 1, 2, \dots, \lfloor \log_2 x \rfloor$, we give a O-bound on the number of primes $p \le x$ with $\mathrm{mpe}(p-1) = v$. For $v = 1$, we use the trivial bound

$$O(\pi(x)) = O\Big(\frac{x}{\log x}\Big) = O\Big(\frac{x}{2\log x}\Big).$$

Now we assume that $v \ge 2$. In order to derive a bound for such v, we use the Brun–Titchmarsh theorem in its stronger form proved by Montgomery and Vaughan [55, Theorem 2]. This result states that for $\mathfrak{a} \in \mathbb{N}^+$, $\mathfrak{b} \in \mathbb{Z}$ and each real $x > \mathfrak{a}$, the number of primes $p \le x$ with $p \equiv \mathfrak{a} \pmod{\mathfrak{b}}$ is at most

$$\frac{2x}{\phi(\mathfrak{b})\log(x/\mathfrak{b})}.$$

Now, a prime $p \le x$ with $\mathrm{mpe}(p-1) = v$ is congruent to 1 modulo $\mathfrak{p}^v$ for some prime $\mathfrak{p} < x^{1/v}$. If $\mathfrak{p}^v \le x^{1/2}$ is fixed, then the Brun–Titchmarsh theorem implies that the number of primes $p \le x$ with $p \equiv 1 \pmod{\mathfrak{p}^v}$ is at most

$$\frac{2x}{\phi(\mathfrak{p}^v)\log(x/\mathfrak{p}^v)} \le \frac{4x}{\mathfrak{p}^{v-1}(\mathfrak{p}-1)\log x} \in O\Big(\frac{x}{\mathfrak{p}^v \log x}\Big).$$

On the other hand, if $x^{1/2} < \mathfrak{p}^v < x$, then the number of primes $p \le x$ with $p \equiv 1 \pmod{\mathfrak{p}^v}$ is at most $x/\mathfrak{p}^v < x^{1/2}$. It follows that the number of primes $p \le x$ with $\mathrm{mpe}(p-1) = v = 2$ is in

$$O\Big(\frac{x}{\log x} \cdot \sum_{\mathfrak{p}\le x^{1/4}} \frac{4}{\mathfrak{p}(\mathfrak{p}-1)} + x^{1/2} \cdot \frac{2x^{1/2}}{\log x}\Big) = O\Big(\frac{x}{\log x}\Big) = O\Big(\frac{x}{4\log x}\Big)$$

and, if $v > 2$, that number is in

$$O\Big(\frac{x}{\log x}\cdot\sum_{\mathfrak{p}\le x^{1/(2v)}}\frac{4}{\mathfrak{p}^{v-1}(\mathfrak{p}-1)}+x^{1/2}\cdot x^{1/3}\Big)=O\Big(\frac{x}{2^v\log x}\Big).$$

In summary, we have shown that for each $v = 1, 2, \ldots, \lfloor \log_2 x \rfloor$, the number of primes $p \le x$ with $\mathrm{mpe}(p-1) = v$ is in $O(x/(2^v \log x))$, and so

$$\sum_{p\le x}\mathrm{mpe}(p-1)\in O\Big(\sum_{v=1}^{\lfloor\log_2 x\rfloor}\frac{vx}{2^v\log x}\Big)=O\Big(\frac{x}{\log x}\sum_{v=1}^{\lfloor\log_2 x\rfloor}\frac{v}{2^v}\Big)=O\Big(\frac{x}{\log x}\Big),$$

whence

$$\Big(\sum_{q\le x}1\Big)^{-1}\sum_{q\le x}\mathrm{mpe}(q-1)\sim\Big(\sum_{p\le x}1\Big)^{-1}\sum_{p\le x}\mathrm{mpe}(p-1)$$
$$\in O\Big(\frac{1}{x/\log x}\cdot\frac{x}{\log x}\Big)=O(1),$$

which is the main statement of this proposition. The first "In particular" statement follows readily from this by observing that the quantity $(\sum_{q\le x}1)^{-1}\sum_{q\le x}\mathrm{mpe}(q-1)$ is the average value of $\mathrm{mpe}(q-1)$ on prime powers $q \le x$. Finally, the second "In particular" statement is an easy consequence of the first. ∎

For applications, finite fields of characteristic 2 are of particular interest. The authors are not aware of any rigorous results concerning the asymptotic behavior of $\mathrm{mpe}(2^v - 1)$ as $v \to \infty$, but in Table 5.1, we provide an overview of the maximum and average values of $\mathrm{mpe}(2^v - 1)$ for $v \in \{1, 2, \ldots, K\}$, where $K \in \{100, 200, \ldots, 1000\}$. This was obtained using GAP [70] and information from the Cunningham project [77]. More specifically, GAP appeared to have difficulties factoring $2^v - 1$ for

$$v \in \{929, 947, 991\},$$

but a quick consultation of the Cunningham factorization tables reveals that

$$\mathrm{mpe}(2^v - 1) = 1$$

for each of these three values of v.

Based on this, we conjecture that the average value of $\mathrm{mpe}(2^v - 1)$ for $1 \le v \le x$ is always less than 2, see Conjecture 6.1.1.

5.2 Proof of Theorem 5.1.9

We give detailed descriptions of algorithms for solving Problems 1–3 and analyze their query complexities (their Las Vegas dual complexities specified in Theorem 5.1.9

K	$\max\{\mathrm{mpe}(2^v-1) : 1 \le v \le K\}$	$K^{-1}\sum_{v=1}^{K} \mathrm{mpe}(2^v-1)$ rounded
100	4	1.28
200	5	1.325
300	5	1.3267
400	5	1.3325
500	6	1.336
600	6	1.3383
700	6	1.3371
800	6	1.34
900	6	1.3389
1000	6	1.341

Table 5.1. Maximum and average values of $\mathrm{mpe}(2^v-1)$.

follow readily using Lemma 5.1.7). The amount of details we give should make it easy to implement these algorithms. In all three cases, we first need to compute $\bar{f}$ and the affine maps A_i, which takes query complexity $(d \log^{1+o(1)} q, d, 0, 0, 0)$ by Proposition 5.1.8. We assume that this has already been done at the start of the discussion of each individual problem. Whenever a positive integer needs to be factored, we subsume this under an mdl or mord query (counted in the third, respectively, fourth, entry of a query complexity). In doing so, we prefer the former but will use the latter in situations where that is more appropriate due to us also needing to compute the multiplicative orders provided by a mord query (this is the case, for example, in the paragraph on $\kappa_{i,p}$ and other parameters in the following section and at the start of the proof of Proposition 5.3.2.3).

5.2.1 Proof of statement (1)

Quite a lot of notations are needed to provide this algorithm in full detail. For the reader's convenience, we print the names of those notations that are newly introduced in this discussion, as well as those of a few notations introduced earlier but rarely used since, in underlined form at the beginning of the respective paragraph where they first appear in this discussion. For the reading flow, these underlined parts need to be ignored. Of course, these notations are also catalogued in Table A.2 in the appendix.

Computing $\bar{\mathcal{L}}$. Before computing $\mathcal{L}$ properly, we need to compute a CRL-list $\bar{\mathcal{L}}$ for $\bar{f}$. Because $\bar{f}$ can be any function $\{0, 1, \dots, d\} \to \{0, 1, \dots, d\}$ with $\bar{f}(d) = d$, we use a general, brute-force algorithm of polynomial complexity in d for this, assuming that the indices $i \in \{0, 1, \dots, d\}$ are processed as non-negative integers in binary representation, with $\lfloor \log_2 d \rfloor + 1$ digits each. Specifically, the following algorithm

based on the idea of "burning leaves", which was kindly pointed out to the authors by one of the reviewers, achieves the computation of $\bar{\mathfrak{L}}$ using $O(d \log^2 d)$ bit operations.

We start by computing three length $d+1$ lists L_1, L_2 and L_3 where, for $j \in \{0, 1, \ldots, d\}$, the $(j+1)$-th entry $L_{1,j}$ of L_1 if $\bar{f}(j)$, and $L_{2,j} := |\bar{f}^{-1}(j)|$ and $L_{3,j} := 0$. Since the values of $\bar{f}$ have already been computed, filling L_1 with its entries is a mere copying process taking $O(d \log d)$ bit operations, and the entries of L_2 can be computed by first letting $L_{2,j} := 0$ for each j, followed by going through the entries of L_1 once, incrementing $L_{2,L_{1,j}}$ for $j = 0, 1, \ldots, d$. This computation of L_2 also takes $O(d \log d)$ bit operations, and so does the (trivial) computation of L_3 (note that we still need to print $O(\log d)$ bits for each 0 entry due to the uniform length of the bit representations).

Now, for a finite functional graph Γ, let $\beta(\Gamma)$ be the (functional) graph obtained from Γ by removing ("burning") all of its leaves. Denoting by $\bar{H}$ the maximum tree height in $\Gamma_{\bar{f}}$, we find that $\Gamma_{\bar{f}} = \beta^0(\Gamma_{\bar{f}}) \supsetneq \beta^1(\Gamma_{\bar{f}}) \supsetneq \cdots \supsetneq \beta^{\bar{H}}(\Gamma_{\bar{f}}) = \beta^{\bar{H}+1}(\Gamma_{\bar{f}})$ and $\beta^{\bar{H}}(\Gamma_{\bar{f}})$ is the subgraph of $\Gamma_{\bar{f}}$ spanned by the periodic vertices. For $h = 0, 1, \ldots, \bar{H}-1$, the vertices in $\mathrm{V}(\beta^h(\Gamma_{\bar{f}})) \setminus \mathrm{V}(\beta^{h+1}(\Gamma_{\bar{f}}))$ are called *new leaves after h burning steps*. For $h = 0$, those are simply the leaves of $\Gamma_{\bar{f}}$. We note that one can also write the set $\mathrm{V}(\beta^h(\Gamma_{\bar{f}})) \setminus \mathrm{V}(\beta^{h+1}(\Gamma_{\bar{f}}))$ as $\mathrm{im}(\bar{f}^h) \setminus \mathrm{im}(\bar{f}^{h+1})$; henceforth, we will denote this set by Layer_h for short.

The aforementioned "burning leaves" algorithm consists of computing, for each $h \in \{0, 1, \ldots, \bar{H}-1\}$, lists L'_h, L''_h and L'''_h of variable length such that L'_h is a repetition-free list of the new leaves after h burning steps (i.e., a list-representation of the set Layer_h), L''_h is a repetition-free list of the $\bar{f}$-images of vertices in L'_h, and L'''_h is a list of the same length as L''_h satisfying

$$L'''_{h,j} = |\bar{f}^{-1}(L''_{h,j}) \cap \mathrm{Layer}_h| \quad \text{for each } j.$$

Throughout this process, we will also update the list L_3 such that at the end of the step for computing L'_h, L''_h and L'''_h, each entry $L_{3,j}$ is the intersection of $\bar{f}^{-1}(j)$ with the set of vertices that are "old" leaves (i.e., elements of Layer_t for some $t < h$).

First, we describe how to obtain L'_0, L''_0 and L'''_0. Go through L_2 once; the zero entries correspond bijectively to the leaves of $\Gamma_{\bar{f}}$, and we can store them repetition-freely in L'_0 using $O(d \log d)$ bit operations. Then store the $\bar{f}$-images of the vertices in L'_0 in the list L''_0 – first with repetitions, then sort and remove repetitions in L''_0, storing the repetition multiplicities in L'''_0. By Lemma 5.1.5 (10), this process takes $O(|L'_0| \log^2 |L'_0|)$ bit operations overall. We may leave all entries of L_3 as 0 for now, since there are no old leaves yet after 0 burning steps.

Now, assume that for a given $h \in \{0, 1, \ldots, \bar{H}-2\}$, we have already computed L'_h, L''_h and L'''_h as well as adjusted L_3 such that it counts the number of old leaf pre-images after h burning steps for each vertex. We would like to compute L'_{h+1},

L''_{h+1} and L'''_{h+1} as well as readjust L_3, so it includes the "new leaves after h burning steps" as old leaves.

First, we describe how to compute L'_{h+1}, and we note that this part of the computations will also need to be done for $h = \bar{H} - 1$, as it is through the equality $L_{\bar{H}} = \emptyset$ that we know we are done (i.e., all transient vertices have been burned as leaves at some point and we have reduced the graph to its periodic vertices).

We go through the entries $L''_{h,j}$ of L''_h, and for each of them, we check whether $L'''_{h,j} + L_{3,L''_{h,j}} = L_{2,j}$. If so, this means that the pre-image $\bar{f}^{-1}(L''_{h,j})$ consists entirely of new leaves after h burning steps and older leaves, whence $L''_{h,j}$ will be a new leaf after $h+1$ burning steps, and we may thus store $L''_{h,j}$ in L'_{h+1}. If not, there is some vertex in the pre-image $\bar{f}^{-1}(L''_{h,j})$ that has never been a leaf so far and will thus "survive" the $(h+1)$-th burning step, whence $L''_{h,j}$ will *not* become a new leave after that burning step. In both cases, we update $L_{3,L''_{h,j}}$ by adding $L'''_{h,j}$ to it to account for the newly burned leaves in step $h+1$. This process of computing the new list L'_{h+1} and updating L_3 takes $O(|L''_h| \log d) \subseteq O(|L'_h| \log d)$ bit operations.

Following that (assuming that $h < \bar{H} - 1$), we may compute L''_{h+1} and L'''_{h+1} based on L'_{h+1} analogously to how we computed L''_0 and L'''_0 based on L'_0 above; this takes $O(|L'_h| \log^2 d)$ bit operations.

Overall, the process of computing the lists L'_h, L''_h and L'''_h for each h takes

$$O(d \log d) + \sum_{i=0}^{\bar{H}-2} O(|\mathrm{Layer}_h| \log^2 d) + O(|\mathrm{Layer}_{\bar{H}-1}| \log d)$$
$$\subseteq O(d \log d) + O\left(\sum_{i=0}^{\bar{H}-2} |\mathrm{Layer}_h| \log^2 d \right) = O(d \log^2 d)$$

bit operations, where the last equality uses that the sets Layer_h are pairwise disjoint.

Finally, we can compute the set of periodic points $\mathrm{per}(\bar{f})$ as the complement of $\bigcup_{h=0}^{\bar{H}-1} \mathrm{Layer}_h$ using another $O(d \log d)$ bit operations. Then we determine $\bar{\mathcal{L}}$ through iteration of $\bar{f}$ on $\mathrm{per}(\bar{f})$ by brute force. If we mark vertices as visited and keep track of the already processed initial segment of the list representing $\mathrm{per}(\bar{f})$ internally, we can do so using $O(d \log d)$ bit operations only, and in the process, we can actually store each cycle of $\bar{f}$ in full, which will be useful shortly. In total, the computations for $\bar{\mathcal{L}}$ require $O(d \log^2 d)$ bit operations.

Computing $\mathcal{L}_i$, U_i, par_i, Y_i. To compute the desired parametrization of $\mathcal{L}$, we go through the elements $(i, \ell) \in \bar{\mathcal{L}}$, with associated $\bar{f}$-cycle $(i_0, i_1, \ldots, i_{\ell-1})$, and compute a parametrization of a CRL-list $\mathcal{L}_i$ of the restriction $f_{|U_i}$, where $U_i = \bigcup_{t=0}^{\ell-1} C_{i_t}$. This works because by Proposition 3.1.1, $\mathcal{L}$ is simply the (disjoint) union of those $\mathcal{L}_i$. Specifically, we compute a formula that defines a bijective function $\mathrm{par}_i : Y_i \to \mathcal{L}_i$, where Y_i is a "simple" set depending on i. For $(i, \ell) = (d, 1)$, which is dealt with

outside the loop for the other pairs (i,ℓ), we have $\mathfrak{L}_d = \{(0_{\mathbb{F}_q}, 1)\}$, and we set $Y_d := \{(\emptyset,\emptyset)\}$ (to conform with the format the sets Y_i for $i < d$ have – each of the two $\emptyset$ is to be viewed as an empty tuple) and define

$$\mathrm{par}_d(\emptyset,\emptyset) := (0_{\mathbb{F}_q}, 1).$$

This only takes $O(\log d)$ bit operations (not $O(1)$, because the index d on the left-hand side of the definition needs to be spelled out).

Computing r_p, $\mathcal{A}_i$, $\overline{\alpha}_i$, $\overline{\beta}_i$, par'_i, $\mathfrak{L}'_i$, $\mathfrak{P}_i$. Next, we factor $s = (q-1)/d$ in a single q-bounded modular discrete logarithm (mdl) query (we remind the reader that we subsume factorizations under mdl queries). We also find a primitive root r_p modulo $p^{\nu_p(s)}$ for each odd prime divisor p of s using a single q-bounded primitive root (prt) query. Following that, we loop over the elements $(i,\ell) \in \overline{\mathfrak{L}}$ with $i < d$, and for each of them, we do the following. We compute

$$\mathcal{A}_i := A_{i_0} A_{i_1} \cdots A_{i_{\ell-1}}, \quad \mathcal{A}_i(z) = \overline{\alpha}_i z + \overline{\beta}_i.$$

This takes $O(d)$ multiplications of already computed affine maps, each of which costs $O(\log^{1+o(1)} q)$ bit operations by Lemma 5.1.5 (1,3) using the formula

$$(z \mapsto \alpha z + \beta)(z \mapsto \alpha' z + \beta') = (z \mapsto \alpha\alpha' z + \alpha'\beta + \beta').$$

Hence, in total, the computation of $\mathcal{A}_i$ takes $O(d \log^{1+o(1)} q)$ bit operations. Our next goal is to compute a parametrization $\mathrm{par}'_i : Y_i \to \mathfrak{L}'_i$ of a CRL-list $\mathfrak{L}'_i$ for $\mathcal{A}_i$, from which $\mathrm{par}_i : Y_i \to \mathfrak{L}_i$ is obtained simply by stretching all second entries (cycle lengths) of images of par'_i by the factor ℓ. As preparations for an upcoming loop over the prime divisors of s, we initialize $\mathfrak{P}_i := \emptyset$ (ultimately, $\mathfrak{P}_i$ will be a list of those prime divisors of s that do *not* divide $\overline{\alpha}_i$). We also compute $\mathrm{ord}_{p^{\nu_p(s)}}(\overline{\alpha}_i)$ for each prime divisor p of s, requiring a single q-bounded multiplicative order (mord) query.

Computing κ_p, $\overline{\mathcal{A}}_{i,p}$, $\mathrm{par}'_{i,p}$, $Y_{i,p}$, $\mathfrak{L}'_{i,p}$. Next, we loop over the prime divisors p of s, and for each of them, we do the following. First, we check whether $p \mid \overline{\alpha}_i$, and if so, we skip to the next value of p. Otherwise, we add p to $\mathfrak{P}_i$, then read off $\kappa_p := \nu_p(s)$ and p^{κ_p} from the factorization of s computed earlier. Following that, we compute $\overline{\mathcal{A}}_{i,p} := \mathcal{A}_i \bmod p^{\kappa_p}$ (that is, we compute $\overline{\alpha}_i \bmod p^{\kappa_p}$ and $\overline{\beta}_i \bmod p^{\kappa_p}$), which takes $O(\log^{1+o(1)} q)$ bit operations by Lemma 5.1.5 (3). We note that since p does not divide $\overline{\alpha}_i$, the function $\overline{\mathcal{A}}_{i,p}$ is an affine *permutation* of $\mathbb{Z}/p^{\kappa_p}\mathbb{Z}$, and from our Table 2.2, we can read off a compact parametrization $\mathrm{par}'_{i,p} : Y_{i,p} \to \mathfrak{L}'_{i,p}$ of a CRL-list $\mathfrak{L}'_{i,p}$ of $\overline{\mathcal{A}}_{i,p}$ in which all specified cycle lengths are fully factored. The details of this are given in Table 5.2 below; each numbered row of that table corresponds to the case with the same number in Table 2.2. The following paragraph introduces some more notation, which is used in Table 2.2 and needs to be computed before one is able to print a description of $\mathrm{par}'_{i,p}$.

No.	u	u'	$\mathrm{par}'_{i,p}(\vec{u})$
1	$0,\dots,\kappa_p$	$0,\dots,$ $\big(\prod_{k=1}^{\mathfrak{n}_p} \mathfrak{p}_{p,k}^{v_{p,k}-v'_{i,p,k}}\big)^{\delta_{[u<\kappa_p]}}\cdot$ $p^{\min\{\kappa_p-1-v'_{i,p},\delta_{[u<\kappa_p]}(\kappa_p-u-1)\}}$ -1	$(\mathfrak{r}_p^{u'}\,p^u+\mathfrak{f}_{i,p},$ $\big(\prod_{k=1}^{\mathfrak{n}_p} \mathfrak{p}_{p,k}^{v'_{i,p,k}}\big)^{\delta_{[u<\kappa_p]}}\cdot$ $p^{\delta_{[u<\kappa_p]}\max\{v'_{i,p}-u,0\}})$
2	$0,\dots,p^{\kappa_{i,p}}-1$	n/a	$(u,\,p^{\kappa_p-\kappa_{i,p}})$
3	$0,\dots,2^{\kappa_{i,2}}-1$	n/a	$(u,2^{\kappa_2-\kappa_{i,2}})$
4	$0,1,2$	n/a	$(u,-u^2+2u+1)$
5	$0,1,2$	n/a	$(2^u-1,\frac{1}{2}u^2-\frac{3}{2}u+2)$
6	$0,1$	n/a	$(2u,2)$
7	$-\kappa_2,\dots,\kappa_2-1$	if $u\in\{-\kappa_2,\kappa_2-1\}$: 0; otherwise: $0,\dots,2^{\kappa_2-2-\max\{v'_{i,2},u\}}-1$	if $u=\kappa_2-1$: $(\mathfrak{f}_{i,2},1)$; if $u=-\kappa_2$: $(2^{\kappa_2-1}+\mathfrak{f}_{i,2},1)$; if $0\le u<\kappa_2-1$: $(5^{u'}2^u+\mathfrak{f}_{i,2},2^{\max\{v'_{i,2}-u,0\}})$; if $-\kappa_2<u<0$: $(-5^{u'}2^{-u-1}+\mathfrak{f}_{i,2},$ $2^{\max\{v'_{i,2}+u+1,0\}})$.
8	$-v''_{i,2},\dots,v''_{i,2}$	if $u=v''_{i,2}$: $0,\dots,2^{\kappa_2-v''_{i,2}-1}$; otherwise: $0,\dots,2^{\kappa_2-v''_{i,2}-2}-1$.	if $u=v''_{i,2}$ and $u'\in\{0,2^{\kappa_2-v''_{i,2}-1}\}$: $(u'2^{v''_{i,2}}+\mathfrak{f}_{i,2},1)$; if $u=v''_{i,2}$ and $0<u'<2^{\kappa_2-v''_{i,2}-1}$: $(u'2^{v''_{i,2}}+\mathfrak{f}_{i,2},2)$; if $0\le u<v''_{i,2}$: $(5^{u'}2^u+\mathfrak{f}_{i,2},2^{v''_{i,2}-u})$; if $u<0$: $(5^{u'}2^{-u-1}+\mathfrak{f}_{i,2},2^{v''_{i,2}+u+1})$.
9	$0,\dots,2^{\kappa_{i,2}}-1$	n/a	$(u,2^{\kappa_2-\kappa_{i,2}})$
10	$1,\dots,2^{\kappa_2-v''_{i,2}-1}$	n/a	if $u=1$: $(0,2^{v''_{i,2}+1})$; otherwise: $(\bar{\beta}_i u,2^{v''_{i,2}+1})$.

Table 5.2. Explicit parametrizations of CRL-lists of affine permutations of finite primary cyclic groups.

Computing $\kappa_{i,p}$, $\mathfrak{n}_p$, $\mathfrak{p}_{p,k}$, $v_{p,k}$, $v'_{i,p,k}$, $v'_{i,p}$, $v''_{i,2}$, $\mathfrak{r}_p$, $\mathfrak{f}_{i,p}$. Recalling that $\nu_p^{(v)}(m):=\min\{\nu_p(m),v\}$, we set

$$\kappa_{i,p}:=\nu_p^{(\kappa_p)}(\bar{\beta}_i)=\nu_p^{(\kappa_p)}(\bar{\beta}_i \bmod p^{\kappa_p}),$$

which can be computed using $O(\kappa_p) \subseteq O(\log p^{\kappa_p})$ integer divisions by p, resulting in a bit operation cost of $O(\log^{2+o(1)} p^{\kappa_p})$. If $p > 2$, we next compute factorizations of $p-1$ and of $\mathrm{ord}_{p^{\kappa_p}}(\overline{\alpha}_i)$ using $2 \in O(1)$ mord queries. We spell these factorizations out as follows:

$$p - 1 = \prod_{k=1}^{\mathfrak{n}_p} \mathfrak{p}_{p,k}^{v_{p,k}}$$

and

$$\mathrm{ord}_{p^{\kappa_p}}(\overline{\alpha}_i) = \prod_{k=1}^{\mathfrak{n}_p} \mathfrak{p}_{p,k}^{v'_{i,p,k}} \cdot p^{v'_{i,p}}.$$

We note that some of the exponents $v'_{i,p,k}$ or $v'_{i,p}$ may be 0. On the other hand, if $p = 2$ (where $\mathfrak{n}_p = 0$), we write

$$\mathrm{ord}_{2^{\kappa_2}}(\overline{\alpha}_i) = 2^{v'_{i,2}},$$

which matches with the notation for $p > 2$ above, and $\mathrm{ord}_{2^{\kappa_2}}(-\overline{\alpha}_i) = 2^{v''_{i,2}}$. Usually, $v''_{i,2} = v'_{i,2}$ as $\mathrm{ord}_{2^{\kappa_2}}(\overline{\alpha}_i) = \mathrm{ord}_{2^{\kappa_2}}(-\overline{\alpha}_i)$, but if $\overline{\alpha}_i \equiv \pm 1 \pmod{2^{\kappa_2}}$, then $v'_{i,2} \in \{0, 1\}$ and $v''_{i,2} = 1 - v_{i,2}$. Finally, regardless of whether or not $p > 2$, we check whether $\overline{\mathcal{A}}_{i,p}$ has a fixed point, i.e., whether

$$\gcd(\overline{\alpha}_i - 1, p^{\kappa_p}) = \gcd((\overline{\alpha}_i \bmod p^{\kappa_p}) - 1, p^{\kappa_p}) \mid \overline{\beta_i} \bmod p^{\kappa_p},$$

which can be done using $O(\log^{1+o(1)} p^{\kappa_p})$ bit operations. We store this information, and whenever $\overline{\mathcal{A}}_{i,p}$ has a fixed point, we compute one, denoted by $\mathfrak{f}_{i,p}$, via the formula in Proposition 2.3.6, taking another $O(\log^{1+o(1)} p^{\kappa_p})$ bit operations.

Computing u, u', $\vec{u}$. We are now ready to give the tabular definition of the bijective parametrization $\mathrm{par}'_{i,p} : Y_{i,p} \to \mathcal{L}'_{i,p}$ of a CRL-list $\mathcal{L}'_{i,p}$ of $\overline{\mathcal{A}}_{i,p}$. We note that the set $Y_{i,p}$ always has one of the following two forms, which will be important later on.

- $Y_{i,p}$ is an integer interval, a general element of which is denoted by u; or
- the elements of $Y_{i,p}$ are pairs (u, u') of integers, where u ranges over an integer interval, and for each fixed value of u, the second entry u' also ranges over an integer interval.

We observe that having entire intervals of integers (or pairs of integers) subsumed under a uniform parametrization like $\mathrm{par}'_{i,p}$ is key to describing the possibly superpolynomially many elements of a CRL-list using just polynomially many bits (the core idea why Problem 1 from the beginning of this chapter can be solved at all).

To have a uniform notation, we may also denote an element of $Y_{i,p}$ by $\vec{u}$ in either of the two cases above. For example, to derive the formulas in the first case of Table 5.2, we apply the first case in Table 2.2, with $v := \kappa_p$, $t := u$, $j := u'$,

$a := \bar{\alpha}_i$ and $b := \bar{\beta}_i$. Then the range for u is clear from the range for t in Table 2.2. Concerning the asserted range for u', we note that

$$\frac{\phi(p^{\kappa_p})}{\mathrm{ord}_{p^{\kappa_p}}(\bar{\alpha}_i)} = \prod_{k=1}^{\mathfrak{n}_p} \mathfrak{p}_{p,k}^{v_{p,k}-v'_{i,p,k}} \cdot p^{\kappa_p-1-v'_{i,p}}$$

and

$$\phi(p^{\kappa_p-u}) = ((p-1)p^{\kappa_p-u-1})^{\delta_{[u<\kappa_p]}} = \left(\prod_{k=1}^{\mathfrak{n}_p} \mathfrak{p}_{p,k}^{v_{p,k}} \cdot p^{\kappa_p-u-1}\right)^{\delta_{[u<\kappa_p]}},$$

from which it can be deduced that

$$\begin{aligned}&\gcd\left(\frac{\phi(p^{\kappa_p})}{\mathrm{ord}_{p^{\kappa_p}}(\bar{\alpha}_i)}, \phi(p^{\kappa_p-u})\right)\\ &\quad= \left(\prod_{k=1}^{\mathfrak{n}_p} \mathfrak{p}_{p,k}^{v_{p,k}-v'_{i,p,k}}\right)^{\delta_{[u<\kappa_p]}} \cdot p^{\min\{\kappa_p-1-v'_{i,p},\delta_{[u<\kappa_p]}(\kappa_p-u-1)\}},\end{aligned}$$

as required. Finally, the formula for the cycle lengths (second entries of $\mathrm{par}'_{i,p}(\vec{u})$) in case 1 holds because

$$\begin{aligned}&\frac{\phi(p^{\kappa_p-u})}{\gcd\left(\frac{\phi(p^{\kappa_p})}{\mathrm{ord}_{p^{\kappa_p}}(\bar{\alpha}_i)}, \phi(p^{\kappa_p-u})\right)}\\ &\quad= \frac{\left(\prod_{k=1}^{\mathfrak{n}_p} \mathfrak{p}_{p,k}^{v_{p,k}} \cdot p^{\kappa_p-u-1}\right)^{\delta_{[u<\kappa_p]}}}{\left(\prod_{k=1}^{\mathfrak{n}_p} \mathfrak{p}_{p,k}^{v_{p,k}-v'_{i,p,k}}\right)^{\delta_{[u<\kappa_p]}} \cdot p^{\min\{\kappa_p-1-v'_{i,p},\delta_{[u<\kappa_p]}(\kappa_p-u-1)\}}}\\ &\quad= \left(\prod_{k=1}^{\mathfrak{n}_p} \mathfrak{p}_{p,k}^{v'_{i,p,k}}\right)^{\delta_{[u<\kappa_p]}} \cdot p^{\delta_{[u<\kappa_p]}(\kappa_p-u-1-\min\{\kappa_p-1-v'_{i,p},\kappa_p-u-1\})}\\ &\quad= \left(\prod_{k=1}^{\mathfrak{n}_p} \mathfrak{p}_{p,k}^{v'_{i,p,k}}\right)^{\delta_{[u<\kappa_p]}} \cdot p^{\delta_{[u<\kappa_p]}\max\{v'_{i,p}-u,0\}}.\end{aligned}$$

The other cases in Table 5.2 can be dealt with analogously. To prevent confusion among readers, we note that in cases 7 and 8 of Table 2.2, the specified CRL-list consists of several disjoint parts with different formulas. Because we want u to range over an integer interval, these have been slightly rearranged and "glued together" here. For example, in case 7 here, the ranges $\{0, 1, \ldots, \kappa_2 - 2\}$ and $\{-\kappa_2 + 1, -\kappa_2 + 2, \ldots, -1\}$ for u correspond, respectively, to the parts with representative elements $5^j 2^t + \dagger$ and $-5^j 2^t + \dagger$ in case 7 of Table 2.2. In the latter of the two segments, the range for u is not equal to the corresponding range for t in Table 2.2, which explains the variable substitution $u \to -u - 1$ although the corresponding formulas for cycle lengths in Table 2.2 are the same. Our algorithm prints and stores the parametric description of

$\mathrm{par}'_{i,p}(\vec{u})$ for all primes $p \mid s$ with $p \nmid \overline{\alpha}_i$. For a given p, this parametric description takes $O(\log p^{\kappa_p})$ bits to store (as follows by observing that it takes $O(\log n)$ bits to print the prime factorization of $n \in \mathbb{N}^+$), and so all descriptions together can be stored using $O(\log s) \subseteq O(\log q)$ bits.

Computing par''_i, $\mathcal{L}''_i$, $\mathcal{A}'_i$, s'_i, $\vec{\vec{u}}$, $\vec{u}_p$, u_p, u'_p, $\overline{Y}_i$, proj_j, $\vec{r}_i(\vec{\vec{u}})$, $r_{i,p}(\vec{u}_p)$, $B_{\vec{r}_i(\vec{\vec{u}})}$, $\mathcal{I}_{i,\vec{\vec{u}}}$. Next, using the parametrizations $\mathrm{par}'_{i,p}$ of the CRL-lists $\mathcal{L}'_{i,p}$ of $\overline{\mathcal{A}}_{i,p}$ for $p \in \mathfrak{P}_i$, we construct a parametrization $\mathrm{par}''_i : Y_i \to \mathcal{L}''_i$ of a CRL-list $\mathcal{L}''_i$ for $\mathcal{A}'_i := \mathcal{A}_i \bmod \prod_{p\in\mathfrak{P}_i} p^{\kappa_p}$. We start by setting $s'_i := \prod_{p\in\mathfrak{P}_i} p^{\kappa_p}$, which takes $O(\log^{2+o(1)} q)$ bit operations to compute, carrying out $|\mathfrak{P}_i| \in O(\log q)$ integer multiplications, each with a bit operation cost in $O(\log^{1+o(1)} q)$ (we note that the powers p^{κ_p} themselves do not need to be computed, as they are specified, alongside the pairs (p, κ_p), in the output of the mdl query that gave the factorization of s). We follow the approach described at the end of Section 2.3. More specifically, we identify $\mathbb{Z}/s'_i\mathbb{Z}$ with $\prod_{p\in\mathfrak{P}_i} \mathbb{Z}/p^{\kappa_p}\mathbb{Z}$, and $\mathcal{A}'_i$ with $\bigotimes_{p\in\mathfrak{P}_i} \overline{\mathcal{A}}_{i,p}$. We consider tuples $\vec{\vec{u}} = (\vec{u}_p)_{p\in\mathfrak{P}_i} \in \prod_{p\in\mathfrak{P}_i} Y_{i,p} =: \overline{Y}_i$. We can either write $\vec{u}_p = u_p$ or $\vec{u}_p = (u_p, u'_p)$. For $j = 1, 2$, we denote by proj_j the (class-sized) function that maps an ordered pair to its j-th entry. Associated with each parameter tuple $\vec{\vec{u}} \in \overline{Y}_i$, we have the tuple

$$\vec{r}_i(\vec{\vec{u}}) := (r_{i,p}(\vec{u}_p))_{p\in\mathfrak{P}_i} := (\mathrm{proj}_1(\mathrm{par}'_{i,p}(\vec{u}_p)))_{p\in\mathfrak{P}_i}$$

of associated cycle representatives of the $\overline{\mathcal{A}}_{i,p}$. By our discussion at the end of Section 2.3, these tuples $\vec{r}_i(\vec{\vec{u}})$ parametrize the blocks $B_{\vec{r}_i(\vec{\vec{u}})}$ of a certain partition of $\prod_{p\in\mathfrak{P}_i} \mathbb{Z}/p^{\kappa_p}\mathbb{Z}$, each block of which is a union of cycles of $\mathcal{A}'_i$. In terms of $\vec{\vec{u}}$, we wish to explicitly describe a CRL-list for the restriction of $\mathcal{A}'_i$ to $B_{\vec{r}_i(\vec{\vec{u}})}$. For this, we need to exhibit an $\vec{r}_i(\vec{\vec{u}})$-admissible indexing function $\mathcal{I}_{i,\vec{\vec{u}}}$ in the sense of Definition 2.3.8 (1) and understand its associated set of good tuples (in the sense of Definition 2.3.8 (2)).

Computing $l_{i,p,\vec{u}_p}$, $l_{i,\vec{\vec{u}}}$, $\mathfrak{P}'_i$. Now, following the definition of an admissible indexing function, the domain of definition of $\mathcal{I}_{i,\vec{\vec{u}}}$ is the set of all primes that divide at least one of the component cycle lengths $l_{i,p,\vec{u}_p} := \mathrm{proj}_2(\mathrm{par}'_{i,p}(\vec{u}_p))$ for $p \in \mathfrak{P}_i$, or, equivalently, that divide $l_{i,\vec{\vec{u}}} := \mathrm{lcm}\{l_{i,p,\vec{u}_p} : p \in \mathfrak{P}_i\}$, of which we compute a parametric definition of bit length in $O(\log^{1+o(1)} q)$ for later use by scanning the displayed parametric factorizations of the $l_{i,p,\vec{u}_p}$, taking $O(\log^{2+o(1)} q)$ bit operations. By definition, the domain of $\mathcal{I}_{i,\vec{\vec{u}}}$ is a subset of

$$\mathfrak{P}'_i := \mathfrak{P}_i \cup \pi\Big(\prod_{p\in\mathfrak{P}_i} (p-1)\Big).$$

We compute $\mathfrak{P}'_i$ as a list (with $O(\log q)$ entries), and this computation consists of $O(\log q)$ containment checks each involving $O(1)$ copying processes of bit strings

of length in $O(\log q)$, and $O(\log q)$ bit comparisons and scans of memory addresses each of length in $O(\log\log q)$. Hence, we may compute $\mathfrak{P}'_i$ using $O(\log^{2+o(1)} q)$ bit operations. In our algorithmic approach, we treat $\mathcal{I}_{i,\vec{\vec{u}}}$ as a function whose domain of definition is all of $\mathfrak{P}'_i$; the additional primes $\mathfrak{p}$ are those which do not divide any component cycle length, hence occur with valuation 0 in each component, and the value $\mathcal{I}_{i,\vec{\vec{u}}}(\mathfrak{p})$ may be chosen arbitrarily in $\mathfrak{P}_i$. We note that this change does not affect the associated notion of good tuples and ensures that the domain of $\mathcal{I}_{i,\vec{\vec{u}}}$ does not depend on $\vec{\vec{u}}$.

For each $\mathfrak{p} \in \mathfrak{P}'_i$, the value $\mathcal{I}_{i,\vec{\vec{u}}}(\mathfrak{p})$ is a prime $p' \in \mathfrak{P}_i$ (thought of as an index for a component of $\vec{\vec{u}}$) such that $\nu_{\mathfrak{p}}(l_{i,p',\vec{\vec{u}}})$ is maximal among all $\nu_{\mathfrak{p}}(l_{i,p,\vec{u}_p})$ for $p \in \mathfrak{P}_i$. We recall that we have already worked out explicit factorizations of the positive integers $l_{i,p,\vec{u}_p}$ in terms of $\vec{u}_p$ (see Table 5.2). If $\mathfrak{p} \notin \mathfrak{P}_i$, then for each $p \in \mathfrak{P}_i$, the value of $\nu_{\mathfrak{p}}(l_{i,p,\vec{u}_p})$ is constant, not depending on $\vec{u}_p$, and a scan along the length $O(\log q)$ parametric description, combined with comparisons of the relevant exponents $\nu_{\mathfrak{p}}(l_{i,p,\vec{u}_p})$, each of which has bit length in $O(\log\log q)$, lets us pick a suitable value for $\mathcal{I}_{i,\vec{\vec{u}}}(\mathfrak{p})$. For a given $\mathfrak{p} \in \mathfrak{P}'_i \setminus \mathfrak{P}_i$, this process requires $O(\log^{1+o(1)} q)$ bit operations, and carrying it out for all $\mathfrak{p} \in \mathfrak{P}'_i \setminus \mathfrak{P}_i$ takes $O(\log^{2+o(1)} q)$ bit operations.

Computing $\mathcal{J}_{i,p,k}$, $\mathfrak{m}_{i,p}$, $\mathcal{J}'_{i,p,k}$. We still need to discuss the approach when $\mathfrak{p} \in \mathfrak{P}_i$. Even then, $\nu_{\mathfrak{p}}(l_{i,p,\vec{u}_p})$ does not depend on $\vec{u}_p$ unless $p = \mathfrak{p}$, in which case one of the following applies.

- $\nu_p(l_{i,p,\vec{\vec{u}}})$ also does not depend on $\vec{u}_p$ (see, e.g., case 2 in Table 5.2), and we can compute a constant value for $\mathcal{I}_{i,\vec{\vec{u}}}(p)$ as described above.
- $\nu_p(l_{i,p,\vec{\vec{u}}})$ does depend on $\vec{u}_p$, in the following way: it only depends on u_p (not u'_p), and one can partition the range for u_p into at most five subintervals $\mathcal{J}_{i,p,1}, \ldots, \mathcal{J}_{i,p,\mathfrak{m}_{i,p}}$ (case 8 in Table 5.2 does require $\mathfrak{m}_{i,p} = 5$) such that in case $u_p \in \mathcal{J}_{i,p,k}$ for a fixed $k \in \{1, \ldots, \mathfrak{m}_{i,p}\}$, the value of $\nu_p(l_{i,p,\vec{\vec{u}}})$ is either constant, or given by a linear expression in u_p, or given by an expression that is the maximum among a linear expression in u_p and 0. This allows us to specify a subinterval (in fact, an initial or terminal segment) $\mathcal{J}'_{i,p,k}$ of $\mathcal{J}_{i,p,k}$ (with constant boundary points) such that $\mathcal{I}_{i,\vec{\vec{u}}}(p)$ may be chosen as p if $u_p \in \mathcal{J}'_{i,p,k}$, whereas $\mathcal{I}_{i,\vec{\vec{u}}}(p)$ must be chosen as a different constant value in $\mathfrak{P}_i$ (the same for each k) if $u_p \in \mathcal{J}_{i,p,k} \setminus \mathcal{J}'_{i,p,k}$. For each given p, writing down an explicit definition of $\mathcal{I}_{i,\vec{\vec{u}}}(p)$ (which consists of a case distinction with at most two cases) requires us to scan the parametric descriptions of the component images $\mathrm{par}'_{i,p}(\vec{u}_p)$ and perform some low-cost computations such as additions or subtractions between exponents of primes (which are numbers of bit length in $O(\log\log q)$). For all relevant values of $\mathfrak{p}$ together, this can be done using $O(\log^{2+o(1)} q)$ bit operations.

We note that for each given $\mathfrak{p} \in \mathfrak{P}'_i$, the parametric definition of $\mathcal{I}_{i,\vec{\vec{u}}}$ which we just derived has bit length in $O(\log q)$. Therefore, and because the domain $\mathfrak{P}'_i$ of $\mathcal{I}_{i,\vec{\vec{u}}}$

has size in $O(\log q)$, it takes $O(\log^2 q)$ bits to store the parametric definitions of all function values of $\mathcal{I}_{i,\vec{u}}$.

Computing $\mathcal{I}_i$, $\mathfrak{P}_{i,p,\vec{u}}$, $\vec{k}$, k_p, $\mathfrak{d}_{i,p,\vec{u}}$, $\vec{k}'$, k'_p, $K'_{i,\vec{u}}$. Before we proceed with our argument, we need to introduce another notation. For $\mathfrak{p} \in \mathfrak{P}'_i$, if $\mathcal{I}_{i,\vec{u}}(\mathfrak{p})$ only assumes one distinct value as $\vec{u}$ ranges over $\overline{Y}_i$, we set $\mathcal{I}_i(\mathfrak{p}) := \mathcal{I}_{i,\vec{u}}(\mathfrak{p})$ for any $\vec{u} \in \overline{Y}_i$. On the other hand, if $\mathcal{I}_{i,\vec{u}}(\mathfrak{p})$ assumes two distinct values, one of which is $\mathfrak{p}$, we let $\mathcal{I}_i(\mathfrak{p})$ be the unique element of $\{\mathcal{I}_{i,\vec{u}}(\mathfrak{p}) : \vec{u} \in \overline{Y}_i\}$ that is distinct from $\mathfrak{p}$. This defines a function $\mathcal{I}_i : \mathfrak{P}'_i \to \mathfrak{P}_i$ that is independent of $\vec{u}$ and can be easily derived from the parametric definitions of the function values $\mathcal{I}_{i,\vec{u}}(\mathfrak{p})$ (taking $O(\log^{2+o(1)} q)$ bit operations). Using the function $\mathcal{I}_i$, we can give the following compact parametric definition of the pre-image of a singleton subset of $\mathfrak{P}_i$ under $\mathcal{I}_{i,\vec{u}}$:

$$\begin{aligned}\mathfrak{P}_{i,p,\vec{u}} &:= \mathcal{I}_{i,\vec{u}}^{-1}(\{p\}) \\ &= \left(\mathcal{I}_i^{-1}(\{p\}) \setminus \{\mathfrak{p} \in \mathfrak{P}_i \setminus \{p\} : u_{\mathfrak{p}} \in \bigcup_{k=1}^{\mathfrak{m}_{i,\mathfrak{p}}} \mathcal{J}'_{i,p,k}\}\right) \\ &\quad \cup \left\{\mathfrak{p} \in \{p\} : u_{\mathfrak{p}} \in \bigcup_{k=1}^{\mathfrak{m}_{i,\mathfrak{p}}} \mathcal{J}'_{i,p,k}\right\}. \end{aligned} \tag{5.2}$$

We note that our algorithm is merely producing this defining formula for $\mathfrak{P}_{i,p,\vec{u}}$ for each $p \in \mathfrak{P}_i$, which is harmless complexity-wise – even when spelling $\mathcal{I}_i^{-1}(\{p\})$ out explicitly in each case, this can be done using $O(\log^{1+o(1)} q)$ bit operations and storage space per p, hence $O(\log^{2+o(1)} q)$ bit operations and storage space altogether. One could also try to provide a case-distinction definition of $\mathfrak{P}_{i,p,\vec{u}}$, where each case corresponds to a constant value of $\mathfrak{P}_{i,p,\vec{u}}$, but this breaks the complexity, as one needs to go through $2^{O(\log q)}$ cases in general. Likewise, it is easy to check that producing each parametric definition described in the rest of this argument takes $O(\log^{2+o(1)} q)$ bit operations if one is careful enough about how to spell those parametrizations out.

Having these explicit definitions of the pre-images $\mathfrak{P}_{i,p,\vec{u}}$ is important because they are needed to set up a parametrization of the $\mathcal{I}_{i,\vec{u}}$-good tuples. We recall from above the notation $l_{i,p,\vec{u}_p}$ for the cycle length of the representative $r_{i,p}(\vec{u}_p)$ in the p-indexed component of $\vec{r}_i(\vec{u})$. An $\mathcal{I}_{i,\vec{u}}$-good tuple is a tuple $\vec{k} = (k_p)_{p \in \mathfrak{P}_i}$ with $k_p \in \mathbb{Z}/l_{i,p,\vec{u}_p}\mathbb{Z} = \{0, 1, \ldots, l_{i,p,\vec{u}_p} - 1\}$ such that k_p is divisible by

$$\mathfrak{d}_{i,p,\vec{u}} := \prod_{\mathfrak{p} \in \mathfrak{P}_{i,p,\vec{u}}} \mathfrak{p}^{\nu_{\mathfrak{p}}(l_{i,\vec{u}})}.$$

We can compute a parametric definition of $\mathfrak{d}_{i,p,\vec{u}}$ and $l_{i,p,\vec{u}_p}/\mathfrak{d}_{i,p,\vec{u}}$ using storage space per p and $O(\log^{1+o(1)} q)$ bit operations, hence $O(\log^{2+o(1)} q)$ bit operations

and storage space altogether. Moreover, we can parametrize the set of $\mathcal{I}_{i,\vec{u}}$-good tuples as follows:

$$\mathrm{Good}_{\vec{r}_i(\vec{u})}(\mathcal{I}_{i,\vec{u}}) = \left\{(k'_p \mathfrak{d}_{i,p,\vec{u}})_{p\in\mathfrak{P}_i} : \vec{k'} = (k'_p)_{p\in\mathfrak{P}_i} \in \prod_{p\in\mathfrak{P}_i} \mathbb{Z}/\frac{l_{i,p,\vec{u}_p}}{\mathfrak{d}_{i,p,\vec{u}}}\mathbb{Z} =: K'_{i,\vec{u}}\right\}.$$

Computing $r'_{i,\vec{u}}(\vec{k'})$. Now, for each $\vec{k'} = (k'_p)_{p\in\mathfrak{P}_i} \in K'_{i,\vec{u}}$ and its associated $\mathcal{I}_{i,\vec{u}}$-good tuple $(k'_p \mathfrak{d}_{i,p,\vec{u}})_{p\in\mathfrak{P}_i}$, we have the cycle representative $(\overline{\mathcal{A}}_{i,p}^{k'_p\mathfrak{d}_{i,p,\vec{u}}}(r_{i,p}(\vec{u}_p)))_{p\in\mathfrak{P}_i}$ of $\mathcal{A}'_i$, or rather, of the permutation $\bigotimes_{p\in\mathfrak{P}_i} \overline{\mathcal{A}}_{i,p}$ identified with it, in $\prod_{p\in\mathfrak{P}_i} \mathbb{Z}/p^{\kappa_p}\mathbb{Z}$. Literally, $\mathcal{A}'_i$ is defined as an affine permutation of $\mathbb{Z}/s'_i\mathbb{Z} = \mathbb{Z}/\prod_{p\in\mathfrak{P}_i} p^{\kappa_p}\mathbb{Z}$. Therefore, the actual cycle representative of $\mathcal{A}'_i$ associated with $\vec{k'}$ is

$$r'_{i,\vec{u}}(\vec{k'}) := \sum_{p\in\mathfrak{P}_i} \overline{\mathcal{A}}_{i,p}^{k'_p\mathfrak{d}_{i,p,\vec{u}}}(r_{i,p}(\vec{u}_p))\frac{s'_i}{p^{\kappa_p}}\,\mathrm{inv}_{p^{\kappa_p}}\left(\frac{s'_i}{p^{\kappa_p}}\right),$$

the unique element of $\mathbb{Z}/s'_i\mathbb{Z}$ that is congruent to $\overline{\mathcal{A}}_{i,p}^{k'_p\mathfrak{d}_{i,p,\vec{u}}}(r_{i,p}(\vec{u}_p))$ modulo p^{κ_p} for each $p \in \mathfrak{P}_i$. We note that the expression $\overline{\mathcal{A}}_{i,p}^{k'_p\mathfrak{d}_{i,p,\vec{u}}}(r_{i,p}(\vec{u}_p))$ can be spelled out explicitly as follows.

$$\overline{\mathcal{A}}_{i,p}^{k'_p\mathfrak{d}_{i,p,\vec{u}}}(r_{i,p}(\vec{u}_p)) = \begin{cases} r_{i,p}(\vec{u}_p) + k'_p\mathfrak{d}_{i,p,\vec{u}}\overline{\beta}_i, & \text{if } \overline{\alpha}_i = 1, \\ \overline{\alpha}_i^{k'_p\mathfrak{d}_{i,p,\vec{u}}} r_{i,p}(\vec{u}_p) + \overline{\beta}_i \frac{\overline{\alpha}_i^{k'_p\mathfrak{d}_{i,p,\vec{u}}}-1}{\overline{\alpha}_i-1}, & \text{otherwise,} \end{cases}$$

where the fraction in the second case is to be understood as an integer division, but the expression as a whole represents an element of $\mathbb{Z}/p^{\kappa_p}\mathbb{Z}$ (one needs to identify the integer value with its reduction modulo p^{κ_p}). It takes $O(\log^{2+o(1)} q)$ bit operations and storage space to compute and store the parametric definition of $r'_{i,\vec{u}}$.

Computing L_i. At last, we can now provide the parametric definitions for the CRL-list $\mathcal{L}''_i$ of $\mathcal{A}'_i$ and, subsequently, for the CRL-list $\mathcal{L}'_i$ of $\mathcal{A}_i$. Namely, $Y_i := \bigcup_{\vec{u}\in\overline{Y}_i} (\{\vec{u}\} \times K'_{i,\vec{u}})$, and for $(\vec{u}, \vec{k'}) \in Y_i$, we set

$$\mathrm{par}''_i(\vec{u}, \vec{k'}) := (r'_{i,\vec{u}}(\vec{k'}), l_{i,\vec{u}}).$$

Then $\mathcal{L}''_i = \{\mathrm{par}''_i(\vec{u}, \vec{k'}) : (\vec{u}, \vec{k'}) \in Y_i\}$. In order to obtain par'_i and $\mathcal{L}'_i$, we simply need to lift the first entries of elements of $\mathcal{L}''_i$ (images of par''_i) from $\mathbb{Z}/s'_i\mathbb{Z}$ to $\mathbb{Z}/s\mathbb{Z}$ such that the reduction modulo s/s'_i of each lift is the unique periodic point of $\mathcal{A}_i$ mod (s/s'_i) in $\mathbb{Z}/(s/s'_i)\mathbb{Z}$. By Lemma 2.1.14, we can compute that periodic point as follows. Let L_i denote the smallest non-negative integer such that

$$\gcd(\alpha_i^{L_i}, s) = \prod_{p|\gcd(\alpha_i,s)} p^{\kappa_p},$$

which satisfies

$$L_i = \max_{p|\gcd(\overline{\alpha}_i,s)} \left\lceil \frac{\nu_p(s)}{\nu_p(\overline{\alpha}_i)} \right\rceil = \max_{p|\gcd(\overline{\alpha}_i,s)} \left\lceil \frac{\nu_p(s)}{\nu_p(\overline{\alpha}_i \bmod p^{\nu_p(s)})} \right\rceil \leq \mathrm{mpe}(s) \in O(\log q)$$

and may be found by computing, for each $p \mid \gcd(\overline{\alpha}_i, s)$, the value $\overline{\alpha}_i \bmod p^{\nu_p(s)}$ (for $\nu_p(s)$, one consults the factorization of s computed above), then finding $\nu_p(\overline{\alpha}_i \bmod p^{\nu_p(s)})$ with a binary search between 0 and $\nu_p(s)$ (each step of which involves a power and a gcd computation). Altogether, this costs

$$O\Big(\log q \cdot \Big(\log^{1+o(1)} q + \sum_{p|\gcd(\overline{\alpha}_i,s)} (\log\log q \cdot \log^{2+o(1)} p^{\nu_p(s)})\Big)\Big) = O(\log^{3+o(1)} q)$$

bit operations by Lemma 5.1.5 (6,8). The unique periodic point of $\mathcal{A}_i \bmod (s/s'_i)$ is the reduction of

$$\sum_{z=0}^{L_i-1} \overline{\alpha}_i^z \overline{\beta}_i = \begin{cases} L_i \overline{\beta}_i, & \text{if } \overline{\alpha}_i = 1, \\ \frac{\overline{\alpha}_i^{L_i}-1}{\overline{\alpha}_i - 1} \overline{\beta}_i, & \text{otherwise,} \end{cases}$$

modulo s/s'_i and may be computed in $O(\log^{2(1+o(1))} q) = O(\log^{2+o(1)} q)$ bit operations using that $\overline{\alpha}_i^{L_i}$ is of bit length in $O(\log^2 q)$. We obtain the following formula for par'_i (which has the domain of definition Y_i, same as par''_i) such that $\mathcal{L}'_i = \{\mathrm{par}'_i(\vec{u}, \vec{k}') : (\vec{u}, \vec{k}') \in Y_i\}$:

$$\mathrm{par}'_i(\vec{u}, \vec{k}') = \left(r'_{i,\vec{u}}(\vec{k}') \frac{s}{s'_i} \mathrm{inv}_{s'_i}\left(\frac{s}{s'_i}\right) + \sum_{z=0}^{L_i-1} \overline{\alpha}_i^z \overline{\beta}_i s'_i \, \mathrm{inv}_{s/s'_i}(s'_i), l_{i,\vec{u}} \right).$$

Printing this parametric definition of $\mathcal{L}'_i$ takes $O(\log^{2+o(1)} q)$ bits of storage space. As mentioned before, the (bijective) parametrization $\mathrm{par}_i : Y_i \to \mathcal{L}_i$ of the CRL-list $\mathcal{L}_i$ of $f_{|U_i}$ can be obtained by stretching the second entries of the images of par'_i by the factor $\ell = \ell_i$ (the $\bar{f}$-cycle length of i). That is,

$$\mathrm{par}_i(\vec{u}, \vec{k}') = \left(r'_{i,\vec{u}}(\vec{k}') \frac{s}{s'_i} \mathrm{inv}_{s'_i}\left(\frac{s}{s'_i}\right) + \sum_{z=0}^{L_i-1} \overline{\alpha}_i^z \overline{\beta}_i s'_i \, \mathrm{inv}_{s/s'_i}(s'_i), \ell \cdot l_{i,\vec{u}} \right)$$

and $\mathcal{L}_i = \{\mathrm{par}_i(\vec{u}, \vec{k}') : (\vec{u}, \vec{k}') \in Y_i\}$.

Finally, the expression $\mathrm{par}_i(\vec{u}, \vec{k}')$, where

- $i \in \mathrm{proj}_1(\overline{\mathcal{L}})$;
- $\vec{u} \in \overline{Y}_i$ (with $\overline{Y}_d := \{\emptyset\}$); and
- $\vec{k}' \in K'_{i,\vec{u}}$ (with $K'_{d,\emptyset} := \{\emptyset\}$)

forms the desired bijective parametrization of a CRL-list $\mathfrak{L}$ of f, which can be pasted together from the results of earlier computations using $O(d \log^{2+o(1)} q)$ bit operations.

In what follows, we conclude this subsection with an overview of the steps of this algorithm. At the end of the description of each step, we specify its q-bounded query complexity (QC); in the case of a loop, this is obtained component-wise by computing the sum of the entries in the corresponding components of the query complexities of the iteration steps of the loop, if applicable replacing the resulting expression by a simpler one that generates the same O-class, and multiplying it with a O-bound on the number of iterations of the loop. It follows from this overview that the query complexity of Problem 1 is as specified in statement (1) of Theorem 5.1.9, and the formula for the Las Vegas dual complexity follows from this and Lemma 5.1.7.

1 Compute the induced function $\bar{f}$ on $\{0, 1, \ldots, d\}$ and the affine maps A_i of $\mathbb{Z}/s\mathbb{Z}$.
QC: $(d \log^{1+o(1)} q, d, 0, 0, 0)$.

2 Compute a CRL-list $\bar{\mathfrak{L}}$ for $\bar{f}$, storing the cycles of $\bar{f}$ in full in the process.
QC: $(d \log^2 d, 0, 0, 0, 0)$.

3 Compute and store the parametrization $\mathrm{par}_d : Y_d \to \mathfrak{L}_d$, where $Y_d = \{(\emptyset, \emptyset)\}$ and $\mathrm{par}_d(\emptyset, \emptyset) = (0_{\mathbb{F}_q}, 1)$.
QC: $(\log d, 0, 0, 0, 0)$.

4 Compute and factor $s = (q-1)/d$.
QC: $(\log^{1+o(1)} q, 0, 1, 0, 0)$.

5 Find a primitive root r_p modulo $p^{\nu_p(s)}$ for each odd prime $p \mid s$.
QC: $(\log q, 0, 0, 0, 1)$.

6 For each $(i, \ell) \in \bar{\mathfrak{L}} \setminus \{(d, 1)\}$, with associated $\bar{f}$-cycle $(i_0, i_1, \ldots, i_{\ell-1})$, which was already computed in step 2, do the following.
QC: $(d^2 \log^{1+o(1)} q + d \log^{2+o(1)} q + \log^{3+o(1)} q, 0, d \log q, d, 0)$.

6.1 Compute the forward cycle product $\mathcal{A}_i = A_{i_0} A_{i_1} \cdots A_{i_{\ell-1}} : z \mapsto \bar{\alpha}_i z + \bar{\beta}_i$.
QC: $(d \log^{1+o(1)} q, 0, 0, 0, 0)$.

6.2 Initialize $\mathfrak{P}_i := \emptyset$.
QC: $(\log d, 0, 0, 0, 0)$.

6.3 Compute $\mathrm{ord}_{p^{\nu_p(s)}}(\bar{\alpha}_i)$ for each prime $p \mid s$.
QC: $(\log q, 0, 0, 1, 0)$.

6.4 For each prime $p \mid s$, do the following.
QC: $(\log^{2+o(1)} q, 0, \log q, 0, 0)$.

6.4.1 Check whether $p \mid \bar{\alpha}_i$, and if not, skip to the next p.
QC: $(\log^{1+o(1)} q, 0, 0, 0, 0)$.

6.4.2 Add p to $\mathfrak{P}_i$ as a new element.
QC: $(\log q, 0, 0, 0, 0)$.

6.4.3 Read off $\kappa_p = \nu_p(s)$ and p^{κ_p} from the factorization of s computed in step 4.
QC: $(\log q, 0, 0, 0, 0)$.

6.4.4 Compute $\overline{\mathcal{A}}_{i,p} = \mathcal{A}_i \bmod p^{\kappa_p}$, i.e., compute $\overline{\alpha}_i \bmod p^{\kappa_p}$ and $\overline{\beta}_i \bmod p^{\kappa_p}$.
QC: $(\log^{1+o(1)} q, 0, 0, 0, 0)$.

6.4.5 Compute $\kappa_{i,p} := \nu_p^{(\kappa_p)}(\overline{\beta}_i) = \nu_p^{(\kappa_p)}(\overline{\beta}_i \bmod p^{\kappa_p})$, using $O(\kappa_p)$ divisions by p and increasing a counter.
QC: $(\log^{2+o(1)} p^{\kappa_p}, 0, 0, 0, 0)$.

6.4.6 If $p > 2$ then do the following.
QC: $(\log q + \log^{1+o(1)} p^{\kappa_p}, 0, 1, 0, 0)$.

6.4.6.1 Compute factorizations of $p-1$ and of $\operatorname{ord}_{p^{\kappa_p}}(\overline{\alpha}_i)$:

$$p - 1 = \prod_{k=1}^{\mathfrak{n}_p} \mathfrak{p}_{p,k}^{\nu_{p,k}} \quad \text{and} \quad \operatorname{ord}_{p^{\kappa_p}}(\overline{\alpha}_i) = \prod_{k=1}^{\mathfrak{n}_p} \mathfrak{p}_{p,k}^{\nu'_{i,p,k}} \cdot p^{\nu'_{i,p}}.$$

QC: $(\log q, 0, 1, 0, 0)$.

6.4.6.2 Check whether $\overline{\mathcal{A}}_{i,p}$ has a fixed point and, if so, store this information and compute a fixed point $\mathfrak{f}_{i,p}$ of it according to Proposition 2.3.6. The check can be done by testing whether $\gcd((\overline{\alpha}_i \bmod p^{\kappa_p}) - 1, p^{\kappa_p})$ divides $\overline{\beta}_i \bmod p^{\kappa_p}$.
QC: $(\log^{1+o(1)} p^{\kappa_p}, 0, 0, 0, 0)$.

6.4.7 Else do the following.
QC: $(\log^{2+o(1)} 2^{\kappa_2}, 0, 0, 0, 0)$.

6.4.7.1 Compute $\nu'_{i,2} = \nu_2(\operatorname{ord}_{2^{\kappa_2}}(\overline{\alpha}_i))$ and $\nu''_{i,2} = \nu_2(\operatorname{ord}_{2^{\kappa_2}}(-\overline{\alpha}_i))$. To avoid making another mord query, we note that $\nu''_{i,2} = \nu'_{i,2}$ unless

$$\overline{\alpha}_i \equiv \pm 1 \pmod{2^{\kappa_2}},$$

in which case $\nu''_{i,2} = 1 - \nu'_{i,2}$.
QC: $(\log^{2+o(1)} 2^{\kappa_2}, 0, 0, 0, 0)$.

6.4.7.2 Check whether $\overline{\mathcal{A}}_{i,2}$ has a fixed point and, if so, store this information and compute a fixed point $\mathfrak{f}_{i,2}$ of it (cf. step 6.4.6.2).
QC: $(\log^{1+o(1)} 2^{\kappa_2}, 0, 0, 0, 0)$.

6.4.8 Spell out a definition of the bijective parametrization $\mathrm{par}'_{i,p} : Y_{i,p} \to \mathscr{L}'_{i,p}$ of a CRL-list $\mathscr{L}'_{i,p}$ of $\overline{\mathcal{A}}_{i,p}$ in which all specified cycle lengths are fully factored, referring to Table 5.2. This requires checking which of the cases

from Table 2.2 applies, and we stored part of the information relevant for this in steps 6.4.6.2 and 6.4.7.2. We note that a general element of $Y_{i,p}$ is denoted by $\vec{u}_p$ and is either equal to u_p or (u_p, u'_p), where u_p and u'_p are integer parameters, with u_p ranging over a fixed interval, and u'_p ranging over an interval for each fixed value of u_p (with explicit formulas for the interval bounds in terms of u_p).
QC: $(\log p^{\kappa_p}, 0, 0, 0, 0)$.

6.5 Compute $s'_i = \prod_{p\in\mathfrak{P}_i} p^{\kappa_p}$.
QC: $(\log^{2+o(1)} q, 0, 0, 0, 0)$.

6.6 Compute a parametric definition of $l_{i,\vec{\vec{u}}}$, the cycle length of $\mathcal{A}'_i$ (or rather, of the permutation $\bigotimes_{p\in\mathfrak{P}_i} \bar{\mathcal{A}}_{i,p}$ on $\prod_{p\in\mathfrak{P}_i} \mathbb{Z}/p^{\kappa_p}\mathbb{Z}$ identified with it) on the point $\vec{r}_i(\vec{\vec{u}})$ represented by $\vec{\vec{u}} = (\vec{u}_p)_{p\in\mathfrak{P}_i}$. This can be done through scanning the parametric definitions of the fully factored cycle lengths

$$l_{i,p,\vec{u}_p} = \mathrm{proj}_2(\mathrm{par}'_{i,p}(\vec{u}_p)),$$

of which $l_{i,\vec{\vec{u}}}$ is the least common multiple, and performing low-cost operations on numbers of bit length in $O(\log\log q)$.
QC: $(\log^{2+o(1)} q, 0, 0, 0, 0)$.

6.7 Set $\mathfrak{P}'_i := \mathfrak{P}_i \cup \pi(\prod_{p\in\mathfrak{P}_i}(p-1))$, using $O(\log q)$ containment checks each involving $O(1)$ copying processes of bit strings of length $O(\log q)$, and $O(\log q)$ bit comparisons and scans of memory addresses each of length $O(\log\log q)$.
QC: $(\log^{2+o(1)} q, 0, 0, 0, 0)$.

6.8 For $\mathfrak{p} \in \mathfrak{P}'_i$, compute a parametric definition of the function value $\mathcal{I}_{i,\vec{\vec{u}}}(\mathfrak{p})$ of the $\vec{r}_i(\vec{\vec{u}})$-admissible indexing function $\mathcal{I}_{i,\vec{\vec{u}}}$. This definition consists of a case distinction with at most two cases (and constant value of $\mathcal{I}_{i,\vec{\vec{u}}}(\mathfrak{p})$ in each case). For $\mathfrak{p} \notin \mathfrak{P}_i$, there is always only one case, and for $\mathfrak{p} \in \mathfrak{P}_i$, the cases depend on the containment of $u_\mathfrak{p}$ in a union of certain intervals (at most five such intervals per $\mathfrak{p}$). Moreover, whenever there are two cases, one of them corresponds to $\mathcal{I}_{i,\vec{\vec{u}}}(\mathfrak{p}) = \mathfrak{p}$. Whenever there is only one case, set $\mathcal{I}_i(\mathfrak{p}) := \mathcal{I}_{i,\vec{\vec{u}}}(\mathfrak{p})$ for any given $\vec{\vec{u}}$, otherwise let $\mathcal{I}_i(\mathfrak{p})$ be the unique element of $\mathcal{I}_{i,\vec{\vec{u}}}(\mathfrak{p})$ that is distinct from $\mathfrak{p}$.
QC: $(\log^{2+o(1)} q, 0, 0, 0, 0)$.

6.9 For $p \in \mathfrak{P}_i$, compute a parametric description of the pre-image set

$$\mathfrak{P}_{i,p,\vec{\vec{u}}} := \mathcal{I}^{-1}_{i,\vec{\vec{u}}}(\{p\}),$$

using formula (5.2). In this parametric description, the inclusion of primes $\mathfrak{p} \in \mathfrak{P}'_i \setminus \mathfrak{P}_i$ in $\mathfrak{P}_{i,p,\vec{\vec{u}}}$ is independent of $\vec{\vec{u}}$, whereas primes $p' \in \mathfrak{P}_i$ each have

a condition, in terms of a disjunction of bounds on $u_{p'}$ corresponding to the intervals mentioned in step 6.8, for whether $p' \in \mathfrak{P}_{i,p,\vec{\bar{u}}}$.
QC: $(\log^{2+o(1)} q, 0, 0, 0, 0)$.

6.10 Based on step 6.9, compute parametric descriptions of

$$\mathfrak{d}_{i,p,\vec{\bar{u}}} = \prod_{\mathfrak{p}\in\mathfrak{P}_{i,p,\vec{\bar{u}}}} \mathfrak{p}^{\nu_{\mathfrak{p}}(l_{i,\vec{\bar{u}}})} = \prod_{\mathfrak{p}\in\mathfrak{P}_{i,p,\vec{\bar{u}}}} \mathfrak{p}^{\nu_{\mathfrak{p}}(l_{i,p,\bar{u}_p})}$$

and $l_{i,p,\vec{\bar{u}}}/\mathfrak{d}_{i,p,\vec{\bar{u}}}$ for each $p \in \mathfrak{P}_i$. In view of step 6.9, this can be achieved using suitable Kronecker deltas in the exponents.
QC: $(\log^{2+o(1)} q, 0, 0, 0, 0)$.

6.11 Compute the parametric description

$$r_{i,\vec{\bar{u}}}(\vec{k'}) = \sum_{p\in\mathfrak{P}_i} \bar{\mathcal{A}}_{i,p}^{k'_p \mathfrak{d}_{i,p,\vec{\bar{u}}}}(r_{i,p}(\vec{u}_p)) \frac{s'_i}{p^{\kappa_p}} \operatorname{inv}_{p^{\kappa_p}}\left(\frac{s'_i}{p^{\kappa_p}}\right)$$

of the cycle representative $r_{i,\vec{\bar{u}}}(\vec{k'})$ of $\mathcal{A}_{i'}$ associated with the parameter tuple $(\vec{\bar{u}}, \vec{k'})$, where $\vec{\bar{u}} \in \overline{Y}_i$ and

$$\vec{k'} = (k'_p)_{p\in\mathfrak{P}_i} \in K'_{i,\vec{\bar{u}}} = \prod_{p\in\mathfrak{P}_i} \mathbb{Z}/(l_{i,p,\bar{u}_p}/\mathfrak{d}_{i,p,\vec{\bar{u}}})\mathbb{Z}.$$

In this expression, the (inexplicit) affine map iterate value $\bar{\mathcal{A}}_{i,p}^{k'_p \mathfrak{d}_{i,p,\vec{\bar{u}}}}(r_{i,p}(\vec{u}_p))$ is to be substituted with the explicit formula

$$\bar{\mathcal{A}}_{i,p}^{k'_p \mathfrak{d}_{i,p,\vec{\bar{u}}}}(r_{i,p}(\vec{u}_p)) = \begin{cases} r_{i,p}(\vec{u}_p) + k'_p \mathfrak{d}_{i,p,\vec{\bar{u}}} \bar{\beta}_i, & \text{if } \bar{\alpha}_i = 1, \\ \bar{\alpha}_i^{k'_p \mathfrak{d}_{i,p,\vec{\bar{u}}}} r_{i,p}(\vec{u}_p) + \bar{\beta}_i \frac{\bar{\alpha}_i^{k'_p \mathfrak{d}_{i,p,\vec{\bar{u}}}} - 1}{\bar{\alpha}_i - 1}, & \text{otherwise.} \end{cases}$$

QC: $(\log^{2+o(1)} q, 0, 0, 0, 0)$.

6.12 Compute

$$L_i = \max_{p \mid \gcd(\bar{\alpha}_i, s)} \left\lceil \frac{\nu_p(s)}{\nu_p(\bar{\alpha}_i)} \right\rceil = \max_{p \mid \gcd(\bar{\alpha}_i, s)} \left\lceil \frac{\nu_p(s)}{\nu_p(\bar{\alpha}_i \bmod p^{\nu_p(s)})} \right\rceil,$$

the smallest non-negative integer such that $\gcd(\bar{\alpha}_i^{L_i}, s) = \prod_{p \mid \gcd(\bar{\alpha}_i, s)} p^{\kappa_p}$. To do so, for each $p \mid \gcd(\bar{\alpha}_i, s)$, compute $\bar{\alpha}_i \bmod p^{\nu_p(s)}$ with a division, then find $\nu_p(\bar{\alpha}_i \bmod p^{\nu_p(s)})$ with a binary search between 0 and $\nu_p(s)$.
QC: $(\log^{3+o(1)} q, 0, 0, 0, 0)$

6.13 Compute the parametric description

$$\operatorname{par}_i(\vec{\bar{u}}, \vec{k'}) = \left(r'_{i,\vec{\bar{u}}}(\vec{k'}) \frac{s}{s'_i} \operatorname{inv}_{s'_i}\left(\frac{s}{s'_i}\right) + \sum_{z=0}^{L_i - 1} \bar{\alpha}_i^z \bar{\beta}_i s'_i \operatorname{inv}_{s/s'_i}(s'_i), \ell \cdot l_{i,\vec{\bar{u}}} \right)$$

of the element of $\mathfrak{L}_i$ (a CRL-list of $f_{|U_i}$, where $U_i = \bigcup_{t=0}^{\ell-1} C_{i_t}$) associated with $(\vec{\vec{u}}, \vec{k}')$.
QC: $(\log^{2+o(1)} q, 0, 0, 0, 0)$.

7 Output the parametric description $\mathrm{par}_i(\vec{\vec{u}}, \vec{k}')$ of the element of $\mathfrak{L}$ (a CRL-list of f) associated with $(i, \vec{\vec{u}}, \vec{k}')$, where $i \in \mathrm{proj}_1(\bar{\mathfrak{L}})$, $\vec{\vec{u}} \in \bar{Y}_i$ and $\vec{k}' \in K'_{i,\vec{\vec{u}}}$ (with the convention that $\bar{Y}_d = \{\emptyset\}$ and $K'_{d,\emptyset} = \{\emptyset\}$), then halt.
QC: $(d \log^{2+o(1)} q, 0, 0, 0, 0)$.

5.2.2 Proof of statement (2)

We note that the only part of our algorithm for Problem 2 where a quantum computer is required is at the beginning, when $\bar{f}$ and the A_i need to be computed. The rest of the algorithm, which we describe henceforth, uses bit operations only.

In addition to computing $\bar{f}$ and the A_i, and as at the beginning of the proof of statement (1), we need to compute the different "layers" of indices $i \in \{0, 1, \ldots, d-1\}$ according to their containment in the iterated images of $\bar{f}$, requiring $O(d \log^2 d)$ bit operations overall.

We follow the approach from Section 3.3, proceeding in three successive steps.

Step 1: transient i. We aim to compute

- $\mathcal{Z}_i = \mathcal{P}_i$ for all $\bar{f}$-transient $i \in \{0, 1, \ldots, d-1\}$;
- a list of rooted tree descriptions $(\mathfrak{D}_0, \mathfrak{D}_1, \ldots, \mathfrak{D}_{N_1})$ that covers all isomorphism types of rooted trees of the form $\mathrm{Tree}_i(\mathcal{P}_i, \vec{\nu}^{(\mathcal{P}_i)})$ for $\bar{f}$-transient $i \in \{0, 1, \ldots, d-1\}$ and logical sign tuples $\vec{\nu}^{(\mathcal{P}_i)} \in \{\emptyset, \neg\}^{m_i}$ such that $\mathcal{B}(\mathcal{P}_i, \vec{\nu}^{(\mathcal{P}_i)}) \neq \emptyset$; and
- the corresponding logical sign tuple data $S_{n,i}$.

At any given point in the algorithm (not just in this step), the set of all $n \in \mathbb{N}_0$ for which $\mathfrak{D}_n$ is defined is an initial segment $\{0, 1, \ldots, N'\}$ of $\mathbb{N}_0$, denoted by $\mathcal{N}$ (a variable that gets updated throughout the process).

In order to carry out the computations listed above, we proceed by recursion on $h_i = \mathrm{ht}(\mathrm{Tree}_{\Gamma_{\bar{f}}}(i))$. First, we assume that $h_i = 0$. Then, in accordance with Section 3.3, we set $\mathcal{P}_i := \mathfrak{P}(\emptyset)$ for all such i, introduce the trivial rooted tree isomorphism type $\mathfrak{T}_0$ via its description $\mathfrak{D}_0 := \emptyset$, and set $S_{0,i} := \{\emptyset\}$ (with $\emptyset$ to be viewed as the empty logical sign tuple), while all $S_{n,i}$ for values $n > 0$ introduced later will be defined as the empty set. This settles the case $h_i = 0$.

Now we assume that $h_i = h > 0$, and that all transient indices j with $h_j < h$ have been taken care of. The first thing we need to do for each given i is to find the $\bar{f}$-pre-images $j_1, j_2, \ldots, j_K$ of i, which requires $O(d \log d)$ bit operations. Following that, we compute a spanning congruence sequence for $\mathcal{P}_i = \bigwedge_{t=1}^{K} \mathfrak{P}'(\mathcal{P}_{j_t}, A_{j_t})$.

This involves simple arithmetic operations (including gcd computations) and requires

$$O\left(\sum_{t=1}^{K} m_{j_t} \log^{1+o(1)} q\right) \subseteq O(d \log^{1+o(1)} q)$$

bit operations. Subsequently, we go through the logical sign tuples $\vec{\nu}^{(\mathcal{P}_i)} \in \{\emptyset, \neg\}^{m_i}$ in lexicographic order, check whether $\mathcal{B}(\mathcal{P}_i, \vec{\nu}^{(\mathcal{P}_i)}) \neq \emptyset$, and if so, compute a compact description $\mathfrak{D}$ of $\mathrm{Tree}_i(\mathcal{P}_i, \vec{\nu}^{(\mathcal{P}_i)})$. For checking whether the block $\mathcal{B}(\mathcal{P}_i, \vec{\nu}^{(\mathcal{P}_i)})$ is non-empty, we note that by the argument before Proposition 3.3.2, the cardinality $|\mathcal{B}(\mathcal{P}_i, \vec{\nu}^{(\mathcal{P}_i)})|$ is equal to the distribution number $\sigma_{\mathcal{P}_i,\mathbf{0}}(\vec{\nu}^{(\mathcal{P}_i)}, (\emptyset, \ldots, \emptyset))$, where $\mathbf{0}$ is the constant zero function $\mathbb{Z}/s\mathbb{Z} \to \mathbb{Z}/s\mathbb{Z}$. To see how costly the computation of this distribution number is, we refer to the following lemma.

Lemma 5.2.2.1. *Let $\mathcal{P}$ be an arithmetic partition of $\mathbb{Z}/m\mathbb{Z}$, given by an explicit spanning m-congruence sequence of length $c \in \mathbb{N}^+$. Moreover, let A be an affine function $\mathbb{Z}/m\mathbb{Z} \to \mathbb{Z}/m\mathbb{Z}$. Then for any given logical sign tuples $\vec{\nu}^{(\mathcal{P})}$ and $\vec{\nu}^{(\mathcal{P}')}$, of length c and $c+1$, respectively, it takes $O(c2^c \log^{1+o(1)} m)$ bit operations to compute the single distribution number $\sigma_{\mathcal{P},A}(\vec{\nu}^{(\mathcal{P})}, \vec{\nu}^{(\mathcal{P}')})$.*

Proof. According to the formula in Lemma 2.2.2, computing $\sigma_{\mathcal{P},A}(\vec{\nu}^{(\mathcal{P})}, \vec{\nu}^{(\mathcal{P}')})$ requires us to add up the summands $(-1)^{|J|}\kappa_{\mathcal{P},A}(\vec{\nu}^{(\mathcal{P})}, \vec{\nu}^{(\mathcal{P}')}, J)$ for all $J \subseteq J_-(\vec{\nu}^{(\mathcal{P})})$, and there are $O(2^c)$ such summands. Computing a single such summand consists of

- a simple look-up of the last component of $\vec{\nu}^{(\mathcal{P}')}$ (bit operation cost: $O(\log c)$ for scanning the corresponding memory address);
- $O(c)$ integer divisibility checks following a gcd computation and subtraction, of total bit operation cost $O(c \log^{1+o(1)} m)$ by Lemma 5.1.5 (1,3,8);
- checking whether the two subsets $J_+(\vec{\nu}^{(\mathcal{P})}) \cup J$ and $J_-(\vec{\nu}^{(\mathcal{P}')})$ of $\{1, 2, \ldots, c\}$ are disjoint, which involves look-ups of entries of $\vec{\nu}^{(\mathcal{P})}$ and $\vec{\nu}^{(\mathcal{P}')}$ and takes $O(c)$ bit operations in total if pointers are used; and
- performing $O(c)$ gcd computations, integer divisions and lcm computations, of total complexity $O(c \log^{1+o(1)} m)$.

Therefore, computing all these summands $(-1)^{|J|}\kappa_{\mathcal{P},A}(\vec{\nu}^{(\mathcal{P}_{j_t})}, \vec{\nu}^{(\mathcal{P}'_{j_t})}, J)$ will take $O(c2^c \log^{1+o(1)} m)$ bit operations, which majorizes the cost of adding these summands up and is thus also the complexity of computing $\sigma_{\mathcal{P},A}(\vec{\nu}^{(\mathcal{P})}, \vec{\nu}^{(\mathcal{P}')})$. ∎

In particular, computing $|\mathcal{B}(\mathcal{P}_i, \vec{\nu}^{(\mathcal{P}_i)})|$ to check whether that block of $\mathcal{P}_i$ is empty costs

$$O(m_i 2^{m_i} \log^{1+o(1)} q) \subseteq O(d 2^d \log^{1+o(1)} q)$$

bit operations.

Let us now assume that $\mathcal{B}(\mathcal{P}_i, \vec{\nu}^{(\mathcal{P}_i)})$ turned out to be non-empty. Then we wish to compute a compact description $\mathfrak{D}$ of $\mathrm{Tree}_i(\mathcal{P}_i, \vec{\nu}^{(\mathcal{P}_i)})$. To do so, we write $\vec{\nu}^{(\mathcal{P}_i)} = \diamond_{t=1}^{K} \vec{\nu}^{(\mathcal{P}'_{j_t})}$ with $\vec{\nu}^{(\mathcal{P}'_{j_t})} \in \{\emptyset, \neg\}^{m_{j_t}+1}$. By Proposition 3.3.1, we may set

$$\mathfrak{D} := \left\{ \left(n, \sum_{t=1}^{K} \sum_{\vec{\nu}^{(\mathcal{P}_{j_t})} \in S_{n,j_t}} \sigma_{\mathcal{P}_{j_t}, A_{j_t}}(\vec{\nu}^{(\mathcal{P}_{j_t})}, \vec{\nu}^{(\mathcal{P}'_{j_t})}) \right) : n \in \mathcal{N} \right\} \setminus (\mathbb{N} \times \{0\}).$$

We note that the range of the summation index in Proposition 3.3.1 includes logical sign tuples $\vec{\nu}^{(\mathcal{P}_{j_t})}$ for which $\mathcal{B}(\mathcal{P}_{j_t}, \vec{\nu}^{(\mathcal{P}_{j_t})})$ is empty, but these may be ignored (as we do here), because all corresponding distribution numbers $\sigma_{\mathcal{P}_{j_t}, A_{j_t}}(\vec{\nu}^{(\mathcal{P}_{j_t})}, \vec{\nu}^{(\mathcal{P}'_{j_t})})$ are 0. According to Lemma 5.2.2.1, computing a single one of the distribution numbers $\sigma_{\mathcal{P}_{j_t}, A_{j_t}}(\vec{\nu}^{(\mathcal{P}_{j_t})}, \vec{\nu}^{(\mathcal{P}'_{j_t})})$ takes

$$O(2^{m_{j_t}} m_{j_t} \log^{1+o(1)} q),$$

bit operations. Observing that for each fixed $t \in \{1, 2, \ldots, K\}$, one has

$$\bigcup \{S_{n,j_t} : n \in \mathcal{N}\} \subseteq \{\emptyset, \neg\}^{m_{j_t}},$$

we end up with a total bit operation cost of

$$O\left(\sum_{t=1}^{K} (2^{m_t} \cdot 2^{m_{j_t}} m_{j_t} \log^{1+o(1)} q) \right) \subseteq O(d 4^d \log^{1+o(1)} q)$$

for computing $\mathfrak{D}$, which majorizes the cost of checking whether $\mathcal{B}(\mathcal{P}_i, \vec{\nu}^{(\mathcal{P}_i)}) \neq \emptyset$. Also, if we go through the numbers $n \in \mathcal{N}$ in increasing order when computing $\mathfrak{D}$, the array representing $\mathfrak{D}$ has its elements ordered by increasing n, as it should.

Next, we need to check whether the rooted tree $\mathfrak{T}$ described by $\mathfrak{D}$ occurs among the rooted trees $\mathfrak{T}_n$, described by $\mathfrak{D}_n$, which have already been introduced. The number of those trees is at most the total number of distinct (non-empty) blocks in all $\mathcal{P}_j$, where j is $\bar{f}$-transient, and that number is in $O(\min\{d 2^d, q\})$. Since each $\mathfrak{D}_n$ as well as $\mathfrak{D}$ is a lexicographically sorted list of length in $O(\min\{d 2^d, q\})$ every entry of which is a bit string of length in $O(\log q)$, it takes $O(d^2 4^d \log q)$ bit operations to check whether $\mathfrak{D} = \mathfrak{D}_n$ for some n. Should that be the case, we add $\vec{\nu}^{(\mathcal{P}_i)}$ to $S_{n,i}$ as a new element (at the end of the array, which leads to that array being lexicographically ordered). Otherwise, we create $\mathfrak{D}$ as a new tree description $\mathfrak{D}_{n'}$, where $n' = \max \mathcal{N} + 1$, and initialize

$$S_{n',j} := \begin{cases} \{\vec{\nu}^{(\mathcal{P}_i)}\}, & \text{if } j = i, \\ \emptyset, & \text{otherwise.} \end{cases}$$

For a given $\bar{f}$-transient i such that $h_i = h$, this loop takes

$$O(2^{m_i}(d4^d \log^{1+o(1)} q + d^2 4^d \log q))$$

bit operations. Now, distinct indices i with $h_i = h$ have disjoint iterated pre-image sets under $\bar{f}$, whence the sum of the numbers m_i for all such i is at most d. Therefore, we get a total bit operation cost of

$$O(d8^d \log^{1+o(1)} q + d^2 8^d \log q)$$

for dealing with all i such that h_i has a given value. Dealing with all $\bar{f}$-transient i in total takes

$$O(d^2 8^d \log^{1+o(1)} q + d^3 8^d \log q)$$

bit operations.

Step 2: $i = d$. Now that the $\bar{f}$-transient indices i have been taken care of, one can compute a description $\mathfrak{D}$ of $\mathrm{Tree}_{\Gamma_f}(0_{\mathbb{F}_q})$ following Proposition 3.3.2, which is similar to a single iteration of the loop in step 1 and takes $O(d4^d \log^{1+o(1)} q)$ bit operations. Afterward, we check whether $\mathfrak{D}$ occurs among the existing descriptions $\mathfrak{D}_n$ (introduced in step 1), which (analogously to step 1) takes $O(d^2 4^d \log q)$ bit operations. If so, we set $S_{n,d} := \emptyset$ (positive logical sign) for the corresponding unique n, and $S_{m,d} := \neg$ for all other m. If not, we introduce $\mathfrak{D}$ as a new tree description $\mathfrak{D}_{n'}$, where $n' = \max \mathcal{N} + 1$, and set $S_{n',d} := \emptyset$ and $S_{m,d} := \neg$ for all $m < n'$. The overall complexity of this step is majorized by the one of step 1.

Step 3: $\bar{f}$-periodic $i < d$. Finally, we discuss $\bar{f}$-periodic indices $i \in \{0, 1, \ldots, d-1\}$. Let us use the notation from step 3 in Section 3.3. For instance, $(i_0, i_1, \ldots, i_{\ell-1})$ is the $\bar{f}$-cycle of $i = i_0$, and we have $i_t = i_{t \bmod \ell}$ for $t \in \mathbb{Z}$ as well as $i' = i_{-1}$.

We begin by computing H_i, the maximum tree height in Γ_f above periodic vertices in cosets of the form C_{i_t} for $t \in \mathbb{Z}$, for each $\bar{f}$-periodic i. Following the argument in Section 3.3, we recall that $H_i \leq \ell \,\mathrm{mpe}(s) \leq \ell \lfloor \log_2 s \rfloor$. Moreover, if we denote for fixed $k \in \mathbb{Z}$ by $h'_{i,k}$ the smallest positive integer h' such that

$$\gcd\left(\prod_{t=0}^{h'-1} \alpha_{i_{k-h'+t}}, s\right) = \gcd\left(\prod_{t=0}^{h'-2} \alpha_{i_{k-h'+t}}, s\right), \tag{5.3}$$

then

$$H_i = \max\{h'_{i,k} : k = 0, 1, \ldots, \ell - 1\} - 1.$$

Before we enter a loop over k to find $h'_{i,k}$, we compute $\bar{\alpha}_i = \prod_{t=0}^{\ell-1} \alpha_{i_t} \bmod s$, taking $O(\ell \log^{1+o(1)} s) \subseteq O(d \log^{1+o(1)} q)$ bit operations. We then enter the loop over $k = 0, 1, \ldots, \ell - 1$. For each k, we aim to find the correct value of $h'_{i,k}$ using a

binary search in the range between 1 and $\ell\lfloor \log_2 s \rfloor + 1$. Let us assume that we fixed a tentative value h'. Then for $H' \in \{h'-1, h'-2\}$, we have

$$\prod_{t=0}^{H'} \alpha_{i_{k-h'+t}} = (\overline{\alpha}_i)^{\lfloor (H'+1)/\ell \rfloor} \cdot \prod_{t=0}^{(H'+1) \bmod \ell - 1} \alpha_{i_{k-h'+t}},$$

which can be computed modulo s for both values of H' using

$$O(\log\log s \log^{1+o(1)} s + \ell \log^{1+o(1)} s) = O(\ell \log^{1+o(1)} s) \subseteq O(d \log^{1+o(1)} q)$$

bit operations. Following that, we compute and compare the two gcds from equation (5.3), which takes $O(\log^{1+o(1)} s) \subseteq O(\log^{1+o(1)} q)$ bit operations. This binary search has $O(\log(\ell \log s)) \subseteq O(\log d + \log\log q)$ iterations per k, and there are $\ell \in O(d)$ values of k to deal with. In total, the computation of H_i takes

$$O(d \cdot (\log d + \log\log q) \cdot d \log^{1+o(1)} q) = O(d^2 \log d \log^{1+o(1)} q)$$

bit operations for each individual i, and

$$O(d^3 \log d \log^{1+o(1)} q)$$

bit operations for all $\bar{f}$-periodic $i < d$ together.

After finding all H_i, we aim to compute $\mathcal{Z}_i = (\mathcal{X}_{i,h})_{h=-1,0,\ldots,H_i}$ for each $\bar{f}$-periodic i. We do so by computing $\mathcal{X}_{i,h}$ for all $\bar{f}$-periodic i together successively for $h = -1, 0, \ldots, \mathfrak{H} := \max\{H_i : i \in \mathrm{per}(\bar{f}) \setminus \{d\}\}$ (for each fixed value of h, we skip those i such that $h > H_i$). Now, $\mathcal{X}_{i,-1} = (\theta_{i,h}(x))_{h=1,2,\ldots,H_i}$ consists of H_i congruences, and according to the definition of $\theta_{i_t,h}(x)$, these congruences can be computed recursively using simple arithmetic in each step. Per i, this takes

$$\begin{aligned} O((H_i+1)\log^{1+o(1)} q) &\subseteq O((d\,\mathrm{mpe}(s)+1)\log^{1+o(1)} q) \\ &\subseteq O(d\,\mathrm{mpe}(q-1)\log^{1+o(1)} q) \end{aligned}$$

bit operations, and for all $\bar{f}$-periodic i together, it takes $O(d^2\,\mathrm{mpe}(q-1)\log^{1+o(1)} q)$ bit operations to compute $\mathcal{X}_{i,-1}$. The computation of $\mathcal{X}_{i,0} = \mathcal{R}_i$ is analogous to the one of $\mathcal{Z}_j = \mathcal{P}_j$ for $\bar{f}$-transient j (see step 1), taking $O(d \log^{1+o(1)} q)$ bit operations per i, and $O(d^2 \log^{1+o(1)} q)$ for all $\bar{f}$-periodic i together. Following that, the spanning congruence sequence for

$$\mathcal{X}_{i,h} = \lambda^h_{i_{-h}}(\mathcal{R}_{i_{-h}}) = \lambda(\lambda^{h-1}_{i_{-h}}(\mathcal{R}_{i_{-h}}), A_{i_{-1}}) = \lambda(\mathcal{X}_{i_{-1},h-1}, A_{i_{-1}})$$

is obtained recursively for $h = 1, 2, \ldots, H_i$ through processing the one for $\mathcal{X}_{i_{-1},h-1}$ using simple arithmetic, again taking complexity $O(d^2 \log^{1+o(1)} q)$ in each step for

all i together. Overall, the bit operation cost of computing $\mathcal{Z}_i$ after each H_i has been worked out is in

$$\begin{aligned} &O(d^2 \operatorname{mpe}(q-1) \log^{1+o(1)} q + (d \operatorname{mpe}(s) + 1) \cdot d^2 \log^{1+o(1)} q) \\ &\quad \subseteq O(d^3 \operatorname{mpe}(q-1) \log^{1+o(1)} q) \end{aligned}$$

for all $\bar{f}$-periodic $i < d$ together.

Finally, we need to

- extend the list of rooted tree descriptions $\mathfrak{D}_n$ produced in steps 1 and 2 to its final version, which additionally contains descriptions of all rooted tree isomorphism types of the form $\operatorname{Tree}_i^{(h)}(\mathcal{P}_{i,h}, \vec{\nu}^{(\mathcal{P}_{i,h})})$, where $i < d$ is $\bar{f}$-periodic, $h \in \{0, 1, \ldots, H_i\}$, $\vec{\nu}^{(\mathcal{P}_{i,h})} \in \{\emptyset, \neg\}^{n_{i-h}+n_{i-h+1}+\cdots+n_{i_0}}$, and $\mathcal{B}(\mathcal{Q}_{i,h}, \vec{\nu}^{(\mathcal{P}_{i,h})} \diamond \vec{\xi}_{i,h}) \neq \emptyset$; and
- compute the associated logical sign sets $S_{n,i,h}$.

We do so recursively in the same manner as before, i.e., computing successively for $h = 0, 1, \ldots, \mathfrak{H}$ the relevant data for *all* corresponding $\bar{f}$-periodic $i < d$ together. For $h = 0$, where $\mathcal{P}_{i,h} = \mathcal{R}_i$, this is basically identical to the corresponding computations in step 1 and has the same overall bit operation cost, in $O(d^2 8^d \log^{1+o(1)} q + d^3 8^d \log q)$.

Now we assume that $h > 0$ and that all smaller values have been taken care of. We loop (in lexicographic order) over the logical sign tuples $\vec{\nu}^{(\mathcal{P}_{i,h})}$ with $\sum_{\mathfrak{t}=0}^{h} n_{i-\mathfrak{t}}$ entries, noting that there are $O(2^{(h+1)d})$ such tuples. For each tuple $\vec{\nu}^{(\mathcal{P}_{i,h})}$, we first check whether $\mathcal{B}(\mathcal{Q}_{i,h}, \vec{\nu}^{(\mathcal{P}_{i,h})} \diamond \vec{\xi}_{i,h})$ is non-empty. Because the length of the spanning congruence sequence for $\mathcal{Q}_{i,h}$ which we use here is at most $(h+1)d + H_i$, Lemma 5.2.2.1 implies that this check takes

$$O(((h+1)d + H_i) 2^{(h+1)d+H_i} \log^{1+o(1)} q)$$

bit operations. If this block of $\mathcal{Q}_{i,h}$ is indeed non-empty, we need to compute a compact description $\mathfrak{D}$ of $\operatorname{Tree}_i(\mathcal{P}_{i,h}, \vec{\nu}^{(\mathcal{P}_{i,h})})$. Let $j_1, j_2, \ldots, j_K$ be the $\bar{f}$-transient pre-images of i under $\bar{f}$ (which take $O(d \log d)$ bit operations to determine). Writing $\vec{\nu}^{(\mathcal{P}_{i,h})} = \diamond_{\mathfrak{t}=0}^{h} \vec{o'_{\mathfrak{t}}}$ with $\vec{o'_{\mathfrak{t}}} \in \{\emptyset, \neg\}^{n_{i-\mathfrak{t}}}$, and $\vec{o'_0} = \diamond_{t=1}^{K} \vec{\nu}^{(\mathcal{P}'_{j_t})}$ with $\vec{\nu}^{(\mathcal{P}'_{j_t})} \in \{\emptyset, \neg\}^{m_{j_t}+1}$, we may choose $\mathfrak{D}$ as follows according to Propositions 3.3.3 and 3.3.4:

$$\begin{aligned} \mathfrak{D} := \Bigg\{ \Bigg(m, &\sum_{t=1}^{K} \sum_{\vec{\nu}^{(\mathcal{P}_{j_t})} \in S_{m,j_t}} \sigma_{\mathcal{P}_{j_t}, A_{j_t}}\left(\vec{\nu}^{(\mathcal{P}_{j_t})}, \vec{\nu}^{(\mathcal{P}'_{j_t})}\right) \\ &+ \sum_{k=0}^{h-1} \sum_{\vec{\nu}^{(\mathcal{P}_{i',k})} \in S_{m,i',k}} \sigma_{\mathcal{Q}_{i',k}, A_{i'}}\left(\vec{\nu}^{(\mathcal{P}_{i',k})} \diamond \vec{\xi}_{i',k}, \diamond_{\mathfrak{t}=1}^{k+1} \vec{o'_{\mathfrak{t}}} \diamond \vec{\xi_{i,h}}\right) \Bigg) : m \in \mathcal{N} \Bigg\} \\ \setminus (\mathbb{N} \times \{0\}). \end{aligned}$$

Computing the first sum,

$$\sum_{t=1}^{K} \sum_{\vec{\nu}^{(\mathcal{P}_{j_t})} \in S_{m,j_t}} \sigma_{\mathcal{P}_{j_t},A_{j_t}}\left(\vec{\nu}^{(\mathcal{P}_{j_t})}, \vec{\nu}^{(\mathcal{P}'_{j_t})}\right),$$

for all $m \in \mathcal{N}$ *together* is analogous to the corresponding argument in step 1 (also applying Lemma 5.2.2.1 with $c := m_{j_t}$ and $m := q$) and takes $O(d4^d \log^{1+o(1)} q)$ bit operations. As for the complexity of computing the second sum,

$$\sum_{k=0}^{h-1} \sum_{\vec{\nu}^{(\mathcal{P}_{i',k})} \in S_{m,i',k}} \sigma_{\mathcal{Q}_{i',k},A_{i'}}(\vec{\nu}^{(\mathcal{P}_{i',k})} \diamond \vec{\xi}_{i',k}, \diamond_{t=1}^{k+1} \vec{o}'_t \diamond \vec{\xi}_{i,h}),$$

we note that for fixed k and $\vec{\nu}^{(\mathcal{P}_{i',k})}$, Lemma 5.2.2.1 with $c := (k+1)d + H_i$ and $m := q$ implies that computing the single distribution number

$$\sigma_{\mathcal{Q}_{i',k},A_{i'}}(\vec{\nu}^{(\mathcal{P}_{i',k})} \diamond \vec{\xi}_{i',k}, \diamond_{t=1}^{k+1} \vec{o}'_t \diamond \vec{\xi}_{i,h})$$

takes

$$O(2^{(k+1)d+H_i}((k+1)d + H_i)\log^{1+o(1)} q)$$

bit operations, whence

$$\begin{aligned} &O(2^{(k+1)d} \cdot 2^{(k+1)d+H_i}((k+1)d + H_i)\log^{1+o(1)} q) \\ &= O(2^{2(k+1)d+H_i}((k+1)d + H_i)\log^{1+o(1)} q) \end{aligned}$$

bit operations are needed for computing all of these numbers for a fixed k *and all* m *together*. Computing the second sum in its entirety for all m together takes

$$\begin{aligned} &O\left(\sum_{k=0}^{h-1} 2^{2(k+1)d+H_i}((k+1)d + H_i)\log^{1+o(1)} q\right) \\ &\subseteq O(2^{2hd+H_i}(hd + H_i)\log^{1+o(1)} q), \end{aligned}$$

bit operations, which majorizes the overall bit operation costs for computing the first sum and for checking whether $\mathcal{B}(\mathcal{P}_i, \vec{\nu}^{(\mathcal{P}_i)})$, and thus is also the cost of computing $\mathfrak{D}$.

Next, we need to check if $\mathfrak{D}$ occurs among the already introduced descriptions $\mathfrak{D}_n$ (for $n \in \mathcal{N}$). The number $|\mathcal{N}|$ of these descriptions is at most the sum of the numbers of distinct (non-empty) blocks in arithmetic partitions of one of the forms

- $\mathcal{P}_j$, where j is $\bar{f}$-transient, or
- $\mathcal{P}_{j,k}$, where $j < d$ is $\bar{f}$-periodic and $k \leq h$,

and that sum is in

$$O\left(d2^d + d\sum_{k=0}^{h} 2^{(k+1)d}\right) = O(d2^{(h+1)d}).$$

Moreover, each of the descriptions $\mathfrak{D}_n$ for $n \in \mathcal{N}$ as well as $\mathfrak{D}$ is a lexicographically sorted list of length in $O(\min\{d2^{(h+1)d}, q\})$ each entry of which is a bit string of length in $O(\log q)$, so it takes $O(d^2 4^{(h+1)d} \log q)$ bit operations to check if $\mathfrak{D} = \mathfrak{D}_n$ for some $n \in \mathcal{N}$. If so, we add $\vec{v}^{(\mathcal{P}_{i,h})}$ to $S_{n,i,h}$ as a new element. Otherwise, we create $\mathfrak{D}$ as a new tree description $\mathfrak{D}_{n'}$, where $n' = \max \mathcal{N} + 1$, and initialize

$$S_{n',j,h} := \begin{cases} \{\vec{v}^{(\mathcal{P}_{i,h})}\}, & \text{if } j = i, \\ \emptyset, & \text{otherwise.} \end{cases}$$

For a given $\bar{f}$-periodic $i < d$, this loop has a bit operation cost in

$$\begin{aligned} &O(2^{(h+1)d} \cdot (2^{2hd+H_i}(hd + H_i)\log^{1+o(1)} q + 4^{(h+1)d} d^2 \log q)) \\ &\quad \subseteq O(2^{3hd+H_i+d}(hd + H_i)\log^{1+o(1)} q + 8^{(h+1)d} d^2 \log q), \end{aligned}$$

and doing this for all such i for a fixed value of h costs

$$O(d(hd + \mathfrak{H})2^{3hd+\mathfrak{H}+d} \log^{1+o(1)} q + d^3 8^{(h+1)d} \log q)$$

bit operations. In total, the bit operation cost of step 3 is in

$$\begin{aligned} &O(d^3 \operatorname{mpe}(q-1)\log^{1+o(1)} q + d^2 8^d \log^{1+o(1)} q + d^3 8^d \log q \\ &+ \sum_{h=1}^{\mathfrak{H}} (d(hd + \mathfrak{H})2^{3hd+\mathfrak{H}+d} \log^{1+o(1)} q + d^3 8^{(h+1)d} \log q)) \\ &\quad \subseteq O(d^3 \operatorname{mpe}(q-1)\log^{1+o(1)} q + d^2 8^d \log^{1+o(1)} q + d^3 8^d \log q \\ &\qquad + d(d+1)\mathfrak{H} 2^{3\mathfrak{H}d+\mathfrak{H}+d} \log^{1+o(1)} q + d^3 8^{(\mathfrak{H}+1)d} \log q) \\ &\quad \subseteq O(d^3 \operatorname{mpe}(q-1) 2^{(3d^2+d)\operatorname{mpe}(q-1)+d} \log^{1+o(1)} q + d^3 8^{(d \operatorname{mpe}(q-1)+1)d} \log q) \\ &\quad \subseteq O(d^3 \operatorname{mpe}(q-1) 2^{(3d^2+d)\operatorname{mpe}(q-1)+2d} \log^{1+o(1)} q), \end{aligned}$$

which majorizes the costs of steps 1 and 2 and thus is the overall bit operation cost of the algorithm for Problem 2, as asserted. The claims on the length $|\mathcal{N}|$ of the constructed recursive tree description list, as well as on the memory costs of the individual rooted tree descriptions, can be deduced as follows. Through applying the above bound on $|\mathcal{N}|$ that holds throughout the h-th iteration of the loop with $h = \mathfrak{H} \leq d \operatorname{mpe}(q-1)$, we get

$$|\mathcal{N}| \in O(d2^{(\mathfrak{H}+1)d}) \subseteq O(d2^{(d \operatorname{mpe}(q-1)+1)d}) = O(d2^{d^2 \operatorname{mpe}(q-1)+d}).$$

Moreover, $|\mathcal{N}| \leq q$ because by construction, each rooted tree isomorphism type in the associated recursive tree description list is of the form $\mathrm{Tree}_{\Gamma_f}(x)$ for some $x \in \mathbb{F}_q = \mathrm{V}(\Gamma_f)$. Each individual tree description $\mathfrak{D}_n$ is a set consisting of pairs of the form (m, k), where the first entries m are pairwise distinct elements of $\mathcal{N}$, whence the length of $\mathfrak{D}_n$ as a list is at most $|\mathcal{N}| \in O(\min\{d2^{d^2\,\mathrm{mpe}(q-1)+d}, q\})$. Finally, the bit cost of storing an individual pair (m, k) is in $O(\log q)$, as required.

We conclude this subsection with a detailed overview of the steps of this algorithm in the form of pseudocode, using the same format as at the end of Section 5.2.1.

1 Compute the induced function $\bar{f}$ on $\{0, 1, \ldots, d\}$ and the affine maps A_i of $\mathbb{Z}/s\mathbb{Z}$.
QC: $(d \log^{1+o(1)} q, d, 0, 0, 0)$.

2 For $i \in \{0, 1, \ldots, d-1\}$, let

$$h_i := \begin{cases} \mathrm{ht}(\mathrm{Tree}_{\Gamma_{\bar{f}}}(i)), & \text{if } i \text{ is } \bar{f}\text{-transient}, \\ \infty, & \text{otherwise.} \end{cases}$$

For each attainable value h of h_i, compute the associated list Layer_h of indices i.
QC: $(d \log^2 d, 0, 0, 0, 0)$.

3 Set $\mathcal{N} := \emptyset$.
QC: $(1, 0, 0, 0, 0)$.

4 Set $\vec{\mathfrak{D}} := \emptyset$.
QC: $(1, 0, 0, 0, 0)$.

5 Set $\vec{\imath} := \emptyset$.
QC: $(1, 0, 0, 0, 0)$.

6 For $h = 0, 1, \ldots, \max\{h_i : i \in \{0, 1, \ldots, d-1\} \setminus \mathrm{per}(\bar{f})\}$, do the following.
QC: $(d^2 8^d \log^{1+o(1)} q + d^3 8^d \log q, 0, 0, 0, 0)$.

6.1 If $h = 0$, then do the following.

6.1.1 Set $N' := 0$.
QC: $(1, 0, 0, 0, 0)$.

6.1.2 Add N' to $\mathcal{N}$ as a new element.
QC: $(1, 0, 0, 0, 0)$.

6.1.3 Set $\mathfrak{D}_0 := \emptyset$, and add it to $\vec{\mathfrak{D}}$ as a new element.
QC: $(1, 0, 0, 0, 0)$.

6.1.4 For each $i \in \mathrm{Layer}_0$, do the following.
QC: $(d \log d, 0, 0, 0, 0)$.

6.1.4.1 Set $\mathcal{Z}_i := \mathcal{P}_i := \mathfrak{P}(\emptyset)$ and $m_i := 0$.
QC: $(\log d, 0, 0, 0, 0)$.

6.1.4.2 Set $S_{0,i} := \{\emptyset\}$.
QC: $(\log d, 0, 0, 0, 0)$.

6.1.4.3 Add i to $\vec{\imath}$ as a new element.
QC: $(\log d, 0, 0, 0, 0)$.

6.2 Else do the following.

6.2.1 For each $i \in \mathrm{Layer}_h$, do the following.
QC: $(d8^d \log^{1+o(1)} q + d^2 8^d \log q, 0, 0, 0, 0)$.

6.2.1.1 Compute the $\bar{f}$-pre-images $j_1, j_2, \ldots, j_K$ of i.
QC: $(d \log d, 0, 0, 0, 0)$.

6.2.1.2 Compute a spanning congruence sequence, of length m_i, for $\mathcal{Z}_i := \mathcal{P}_i := \bigwedge_{t=1}^{K} \mathfrak{P}'(\mathcal{P}_{j_t}, A_{j_t})$.
QC: $(d \log^{1+o(1)} q, 0, 0, 0, 0)$.

6.2.1.3 For each $n \in \mathcal{N}$, do the following.
QC: $(d2^d \log q, 0, 0, 0, 0)$.

6.2.1.3.1 Initialize $S_{n,i} := \emptyset$.
QC: $(\log q, 0, 0, 0, 0)$.

6.2.1.4 For each $\vec{\nu}^{(\mathcal{P}_i)} \in \{\emptyset, \neg\}^{m_i}$, do the following.
QC: $(2^{m_i}(d4^d \log^{1+o(1)} q + d^2 4^d \log q), 0, 0, 0, 0)$.

6.2.1.4.1 Check whether $|\mathcal{B}(\mathcal{P}_i, \vec{\nu}^{(\mathcal{P}_i)})| = \sigma_{\mathcal{P}_i,\emptyset}(\vec{\nu}^{(\mathcal{P}_i)}, (\emptyset, \ldots, \emptyset)) = 0$, and if so, skip to the next tuple $\vec{\nu}^{(\mathcal{P}_i)}$.
QC: $(d2^d \log^{1+o(1)} q, 0, 0, 0, 0)$.

6.2.1.4.2 Writing $\vec{\nu}^{(\mathcal{P}_i)} = \diamond_{t=1}^{K} \vec{\nu}^{(\mathcal{P}'_{j_t})}$ with $\vec{\nu}^{(\mathcal{P}'_{j_t})} \in \{\emptyset, \neg\}^{m_{j_t}+1}$, compute

$$\mathfrak{D} := \left\{ \left(n, \sum_{t=1}^{K} \sum_{\vec{\nu}^{(\mathcal{P}_{j_t})} \in S_{n,j_t}} \sigma_{\mathcal{P}_{j_t}, A_{j_t}}(\vec{\nu}^{(\mathcal{P}_{j_t})}, \vec{\nu}^{(\mathcal{P}'_{j_t})}) \right) : n \in \mathcal{N} \right\} \setminus (\mathbb{N} \times \{0\}),$$

a compact description of $\mathrm{Tree}_i(\mathcal{P}_i, \vec{\nu}^{(\mathcal{P}_i)})$.
QC: $(d4^d \log^{1+o(1)} q, 0, 0, 0, 0)$.

6.2.1.4.3 Check whether $\mathfrak{D} = \mathfrak{D}_n$ for some (unique) $n \in \mathcal{N}$, and store this information (the truth value and, if applicable, n).
QC: $(d^2 4^d \log q, 0, 0, 0, 0)$.

6.2.1.4.4 If $\mathfrak{D} = \mathfrak{D}_n$ for some $n \in \mathcal{N}$, then do the following.

6.2.1.4.4.1 Add $\vec{\nu}^{(\mathcal{P}_i)}$ to $S_{n,i}$ as a new element.
QC: $(\log q + d, 0, 0, 0, 0)$.

6.2.1.4.5 Else do the following.

6.2.1.4.5.1 Set $N' := N' + 1$, and add it to $\mathcal{N}$ as a new element.
QC: $(\log q, 0, 0, 0, 0)$.

6.2.1.4.5.2 Set $\mathfrak{D}_{N'} := \mathfrak{D}$, and add it to $\vec{\mathfrak{D}}$ as a new element.
QC: $(d\,2^d \log q, 0, 0, 0, 0)$.

6.2.1.4.5.3 For $j \in \vec{\imath}$, do the following.
QC: $(d \log q, 0, 0, 0, 0)$.

6.2.1.4.5.3.1 Set $S_{N',j} := \emptyset$.
QC: $(\log q, 0, 0, 0, 0)$.

6.2.1.4.5.4 Set $S_{N',i} := \{\vec{v}^{(\mathcal{P}_i)}\}$.
QC: $(\log q + d, 0, 0, 0, 0)$.

6.2.1.5 Add i to $\vec{\imath}$ as a new element.
QC: $(\log d, 0, 0, 0, 0)$.

7 Compute the $\bar{f}$-transient $\bar{f}$-pre-images $j_1, j_2, \ldots, j_K$ of d.
QC: $(d \log d, 0, 0, 0, 0)$.

8 Compute

$$\mathfrak{D} := \left\{ \left(n, \sum_{t=1}^{K} \sum_{\vec{v}^{(\mathcal{P}_{j_t})} \in S_{n,j_t}} \sigma_{\mathcal{P}_{j_t},\mathbf{0}}(\vec{v}^{(\mathcal{P}_{j_t})}, (\emptyset, \ldots, \emptyset)) \right) : n \in \mathcal{N} \right\} \setminus (\mathbb{N} \times \{0\}),$$

a compact description of $\mathrm{Tree}_{\Gamma_f}(0_{\mathbb{F}_q})$.
QC: $(d\,4^d \log^{1+o(1)} q, 0, 0, 0, 0)$.

9 Check whether $\mathfrak{D} = \mathfrak{D}_n$ for some (unique) $n \in \mathcal{N}$, and store this information (the truth value and, if applicable, n).
QC: $(d^2 4^d \log q, 0, 0, 0, 0)$.

10 If $\mathfrak{D} = \mathfrak{D}_n$ for some $n \in \mathcal{N}$, then do the following.

10.1 Set $S_{n,d} := \emptyset$, and $S_{m,d} := \neg$ for all $m \in \mathcal{N} \setminus \{n\}$.
QC: $(d\,2^d \log q, 0, 0, 0, 0)$.

11 Else do the following.

11.1 Set $N' := N' + 1$, and add it to $\mathcal{N}$ as a new element.
QC: $(\log q, 0, 0, 0, 0)$.

11.2 Set $\mathfrak{D}_{N'} := \mathfrak{D}$, and add it to $\vec{\mathfrak{D}}$ as a new element.
QC: $(d\,2^d \log q, 0, 0, 0, 0)$.

11.3 For $j \in \vec{\imath} = \{0, 1, \ldots, d-1\} \setminus \mathrm{per}(\bar{f})$, do the following.
QC: $(d \log q, 0, 0, 0, 0)$.

11.3.1 Set $S_{N',j} := \emptyset$.
QC: $(\log q, 0, 0, 0, 0)$.

11.4 Set $S_{N',d} := \emptyset$, and $S_{n,d} := \neg$ for all $n \in \mathcal{N} \setminus \{N'\}$.
QC: $(d\,2^d \log q, 0, 0, 0, 0)$.

12 For $i \in \mathrm{per}(\bar{f}) \setminus \{d\}$, do the following.
QC: $(d^3 \log d \log^{1+o(1)} q, 0, 0, 0, 0)$.

12.1 Compute H_i, the maximum tree height in Γ_f above periodic vertices in cosets of the form C_{i_t}, where $t \in \mathbb{Z}$. See the discussion above for details.
QC: $(d^2 \log d \log^{1+o(1)} q, 0, 0, 0, 0)$.

13 Compute $\mathfrak{H} := \max\{H_i : i \in \mathrm{per}(\bar{f}) \setminus \{d\}\}$.
QC: $(d \log d, 0, 0, 0, 0)$.

14 For $h = -1, 0, 1, \ldots, \mathfrak{H}$, do the following.
QC: $(d^3\,\mathrm{mpe}(q-1) \log^{1+o(1)} q, 0, 0, 0, 0)$.

14.1 If $h = -1$, then do the following.

14.1.1 For $i \in \mathrm{per}(\bar{f}) \setminus \{d\}$, do the following.
QC: $(d^2\,\mathrm{mpe}(q-1) \log^{1+o(1)} q, 0, 0, 0, 0)$.

14.1.1.1 Compute and store the $\bar{f}$-pre-images of i, including the periodic one, i'.
QC: $(d \log d, 0, 0, 0, 0)$.

14.1.1.2 Compute $\mathcal{X}_{i,-1} := (\theta_{i,k}(x))_{k=1,2,\ldots,H_i}$.
QC: $(d\,\mathrm{mpe}(q-1) \log^{1+o(1)} q, 0, 0, 0, 0)$.

14.2 Else do the following.

14.2.1 If $h = 0$, then do the following.

14.2.1.1 For $i \in \mathrm{per}(\bar{f}) \setminus \{d\}$, do the following.
QC: $(d^2 \log^{1+o(1)} q, 0, 0, 0, 0)$.

14.2.1.1.1 Compute $\mathcal{X}_{i,0} = \mathcal{R}_i = \bigwedge_{t=1}^{K} \mathfrak{P}'(\mathcal{P}_{j_t}, A_{j_t})$, with a spanning sequence of length n_i, where $j_1, j_2, \ldots, j_K$ are the $\bar{f}$-transient $\bar{f}$-pre-images of i (this can be handled analogously to steps 6.2.1.1 and 6.2.1.2).
QC: $(d \log^{1+o(1)} q, 0, 0, 0, 0)$.

14.2.2 Else do the following.

14.2.2.1 For $i \in \mathrm{per}(\bar{f}) \setminus \{d\}$, do the following.
QC: $(d^2 \log^{1+o(1)} q, 0, 0, 0, 0)$.

14.2.2.1.1 If $h \le H_i$, then compute $\mathcal{X}_{i,h} = \lambda(\mathcal{X}_{i',h-1}, A_{i'})$ (we observe that as a spanning congruence sequence, $\mathcal{X}_{i,h}$ has length $n_{i'}$, same as $\mathcal{X}_{i',h-1}$). Otherwise, skip to the next value of i.
QC: $(d \log^{1+o(1)} q, 0, 0, 0, 0)$.

15 For $i \in \mathrm{per}(\bar{f}) \setminus \{d\}$, do the following.
QC: $(d^3\,\mathrm{mpe}(q-1) \log q, 0, 0, 0, 0)$.

15.1 Set $\mathcal{Z}_i := (\mathcal{X}_{i,h})_{h=-1,0,\ldots,H_i}$.
QC: $(d^2 \operatorname{mpe}(q-1)\log q, 0, 0, 0, 0)$.

16 For $h = 0, 1, \ldots, \mathfrak{H}$, do the following.
QC: $(d^3 \operatorname{mpe}(q-1)2^{(3d^2+d)\operatorname{mpe}(q-1)+2d}\log^{1+o(1)} q, 0, 0, 0, 0)$.

16.1 Initialize all sets $S_{n,i,h}$, where $n \in \mathcal{N}$ and $i \in \operatorname{per}(\bar{f}) \setminus \{d\}$ such that $H_i \geq h$, to be $\emptyset$.
QC: $(d^2 2^{hd}, 0, 0, 0, 0)$.

16.2 If $h = 0$, then do the following.

16.2.1 Extend $\mathcal{N}$ and the associated list $\vec{\mathfrak{D}}$ of rooted tree descriptions $\mathfrak{D}_n$ such that all rooted tree isomorphism types of the form

$$\operatorname{Tree}_i(\mathcal{P}_{i,0}, \vec{\nu}^{(\mathcal{P}_{i,0})}) = \operatorname{Tree}_i\left(\mathcal{R}_i, \bigcup_{t=1}^{K} C_{j_t}, \vec{\nu}^{(\mathcal{P}_{i,0})}\right)$$

for $i \in \operatorname{per}(\bar{f}) \setminus \{d\}$ are covered, and compute the associated logical sign tuple sets $S_{n,i,0}$. This is analogous to step 6.2.1.4, but carried out for the $O(d)$ values of $i \in \operatorname{per}(\bar{f}) \setminus \{d\}$ together.
QC: $(d^2 8^d \log^{1+o(1)} q + d^3 8^d \log q, 0, 0, 0, 0)$.

16.3 Else do the following.

16.3.1 For each $i \in \operatorname{per}(\bar{f}) \setminus \{d\}$, do the following.
QC: $(d(hd + \mathfrak{H})2^{3hd+\mathfrak{H}+d}\log^{1+o(1)} q + d^3 8^{(h+1)d}\log q, 0, 0, 0, 0)$.

16.3.1.1 Check whether $h \leq H_i$. If not, skip to the next i.
QC: $(\log q, 0, 0, 0, 0)$.

16.3.1.2 Recalling that $\mathcal{P}_{i,h} = \bigwedge_{t=0}^{h} \mathcal{X}_{i,t}$, do the following for each $\vec{\nu}^{(\mathcal{P}_{i,h})} \in \{\emptyset, \neg\}^{n_{i_0}+n_{i_{-1}}+\cdots+n_{i_{-h}}}$.
QC: $((hd + H_i)2^{3hd+H_i+d}\log^{1+o(1)} q + d^2 8^{(h+1)d}\log q, 0, 0, 0, 0)$.

16.3.1.2.1 Recalling that $\mathcal{Q}_{i,h} = \mathcal{P}_{i,h} \wedge \mathcal{U}_i$, where $\mathcal{U}_i = \mathfrak{P}'(\theta_{i,k}(x) : k = 1, 2, \ldots, H_i)$ (and the definition of $\vec{\xi}_{i,h} \in \{\emptyset, \neg\}^{H_i}$ from page 55), check whether

$$|\mathcal{B}(\mathcal{Q}_{i,h}, \vec{\nu}^{(\mathcal{P}_{i,h})} \diamond \vec{\xi}_{i,h})| = \sigma_{\mathcal{Q}_{i,h},\mathbf{0}}(\vec{\nu}^{(\mathcal{P}_{i,h})} \diamond \vec{\xi}_{i,h}, (\emptyset, \ldots, \emptyset)) = 0,$$

and if so, skip to the next tuple $\vec{\nu}^{(\mathcal{P}_{i,h})}$.
QC: $(((h+1)d + H_i)2^{(h+1)d+H_i}\log^{1+o(1)} q, 0, 0, 0, 0)$.

16.3.1.2.2 Writing $\vec{\nu}^{(\mathcal{P}_{i,h})} = \diamond_{t=0}^{h} \vec{o'_t}$ with $\vec{o'_t} \in \{\emptyset, \neg\}^{n_{i_{-t}}}$, and

$$\vec{o'_0} = \diamond_{t=1}^{K} \vec{\nu}^{(\mathcal{P}'_{j_t})}$$

with $\vec{\nu}^{(\mathcal{P}'_{j_t})} \in \{\emptyset, \neg\}^{n_{j_t}+1}$ (where $j_1, j_2, \ldots, j_K$ are the $\bar{f}$-transient $\bar{f}$-pre-images of i, computed in step 14.1.1.1), compute

$$\mathfrak{D} := \left\{ \left(m, \sum_{t=1}^{K} \sum_{\vec{\nu}^{(\mathcal{P}_{j_t})} \in S_{m,j_t}} \sigma_{\mathcal{P}_{j_t}, A_{j_t}}(\vec{\nu}^{(\mathcal{P}_{j_t})}, \vec{\nu}^{(\mathcal{P}'_{j_t})}) + \sum_{k=0}^{h-1} \sum_{\vec{\nu}^{(\mathcal{P}_{i',k})} \in S_{m,i',k}} \sigma_{\mathcal{Q}_{i',k}, A_{i'}}(\vec{\nu}^{(\mathcal{P}_{i',k})} \diamond \vec{\xi}_{i',k}, \diamond_{\mathfrak{t}=1}^{k+1} \vec{o}'_{\mathfrak{t}} \diamond \vec{\xi}_{i,h}) \right) : m \in \mathcal{N} \right\} \setminus (\mathbb{N} \times \{0\}).$$

QC: $(2^{2hd+H_i}(hd + H_i)\log^{1+o(1)} q, 0, 0, 0, 0)$.

16.3.1.2.3 Check whether $\mathfrak{D} = \mathfrak{D}_n$ for some (unique) $n \in \mathcal{N}$, and store this information (the truth value and, if applicable, n).
QC: $(d^2 4^{(h+1)d} \log q, 0, 0, 0, 0)$.

16.3.1.2.4 If $\mathfrak{D} = \mathfrak{D}_n$ for some $n \in \mathcal{N}$, then do the following.

16.3.1.2.4.1 Add $\vec{\nu}^{(\mathcal{P}_{i,h})}$ to $S_{n,i,h}$ as a new element.
QC: $(\log q + (h+1)d, 0, 0, 0, 0)$.

16.3.1.2.5 Else do the following.

16.3.1.2.5.1 Set $N' := N' + 1$, and add it to $\mathcal{N}$ as a new element.
QC: $(\log q, 0, 0, 0, 0)$.

16.3.1.2.5.2 Set $\mathfrak{D}_{N'} := \mathfrak{D}$, and add it to $\vec{\mathfrak{D}}$ as a new element.
QC: $(d 2^{(h+1)d} \log q, 0, 0, 0, 0)$.

16.3.1.2.5.3 For $j \in \vec{\imath} = \{0, 1, \ldots, d-1\} \setminus \mathrm{per}(\bar{f})$, do the following.
QC: $(d \log q, 0, 0, 0, 0)$.

16.3.1.2.5.3.1 Set $S_{N',j} := \emptyset$.
QC: $(\log q, 0, 0, 0, 0)$.

16.3.1.2.5.4 Set $S_{N',d} := \neg$.
QC: $(\log q, 0, 0, 0, 0)$.

16.3.1.2.5.5 For $k = 0, 1, \ldots, h-1$, do the following.
QC: $(dh \log q, 0, 0, 0, 0)$.

16.3.1.2.5.5.1 For $j \in \mathrm{per}(\bar{f}) \setminus \{d\}$, do the following.
QC: $(d \log q, 0, 0, 0, 0)$.

16.3.1.2.5.5.1.1 Set $S_{N',j,k} := \emptyset$.
QC: $(\log q, 0, 0, 0, 0)$.

16.3.1.2.5.6 For $j \in \mathrm{per}(\bar{f}) \setminus \{d\}$, do the following.
QC: $(d \log q + (h+1)d^2, 0, 0, 0, 0)$.

16.3.1.2.5.6.1 Set

$$S_{N',j,h} := \begin{cases} \{\vec{v}^{(\mathcal{P}_{i,h})}\}, & \text{if } j = i, \\ \emptyset, & \text{otherwise.} \end{cases}$$

QC: $(\log q + (h+1)d, 0, 0, 0, 0)$.

17 For $n \in \mathcal{N}$, do the following.
QC: $(d^4 \operatorname{mpe}(q-1)4^{d^2 \operatorname{mpe}(q-1)+d}, 0, 0, 0, 0)$; please note that although the QC term for step 16 only involves d^3, not d^4, it still majorizes this.

17.1 For $i \in \mathrm{per}(\bar{f}) \setminus \{d\}$, do the following.
QC: $(d^3 \operatorname{mpe}(q-1)2^{d^2 \operatorname{mpe}(q-1)+d}, 0, 0, 0, 0)$.

17.1.1 Set $S_{n,i} := (S_{n,i,h})_{h=0,1,\ldots,H_i}$.
QC: $(d^2 \operatorname{mpe}(q-1)2^{d^2 \operatorname{mpe}(q-1)+d}, 0, 0, 0, 0)$ for copying, using that

$$\begin{aligned} \sum_{h=0}^{H_i} 2^{(h+1)d}(h+1)d &\in O((H_i+1)d2^{(H_i+1)d}) \\ &\subseteq O(d^2 \operatorname{mpe}(q-1)2^{d^2 \operatorname{mpe}(q-1)+d}) \end{aligned}$$

18 Output the partition-tree register $((\mathcal{Z}_i)_{i=0,1,\ldots,d-1}, ((\mathfrak{D}_n, (S_{n,i})_{i=0,1,\ldots,d}))_{n\in\mathcal{N}})$, and halt.
QC: $(d^2 4^{d^2 \operatorname{mpe}(q-1)+d} \log q + d^4 \operatorname{mpe}(q-1)4^{d^2 \operatorname{mpe}(q-1)+d}, 0, 0, 0, 0)$ for copying, using that the bit storage cost of $\mathcal{X}_{i,-1} = (\theta_{i,h}(x))_{h=1,2,\ldots,H_i}$ for fixed i is in

$$O(H_i \log q) \subseteq O(d \operatorname{mpe}(q-1) \log q),$$

the bit storage cost of $\mathcal{X}_{i,h}$ for fixed i and h is in $O((h+1)d \log q)$, the total bit storage cost of the recursive tree description list $(\mathfrak{D}_n)_{n\in\mathcal{N}}$ is in $O(|\mathcal{N}|^2 \log q) \subseteq O(d^2 4^{d^2 \operatorname{mpe}(q-1)+d} \log q)$, the total bit storage cost of the $S_{n,i}$ for $n \in \mathcal{N}$ and $i \in \{0, 1, \ldots, d\}$ is in $O(d^4 \operatorname{mpe}(q-1)4^{d^2 \operatorname{mpe}(q-1)+d})$ (the bit operation cost of step 17; the combined storage cost of the $S_{n,i}$ for $i \notin \mathrm{per}(\bar{f})$ or $i = d$ is majorized by that for $i \in \mathrm{per}(\bar{f}) \setminus \{d\}$), and

$$\begin{aligned} &O(dH_i \log q + d \sum_{h=0}^{H_i} (h+1)d \log q + d^2 4^{d^2 \operatorname{mpe}(q-1)+d} \log q \\ &\quad + d^4 \operatorname{mpe}(q-1)4^{d^2 \operatorname{mpe}(q-1)+d}) \\ &\quad \subseteq O(d^2 4^{d^2 \operatorname{mpe}(q-1)+d} \log q + d^4 \operatorname{mpe}(q-1)4^{d^2 \operatorname{mpe}(q-1)+d}). \end{aligned}$$

5.2.3 Proof of statement (3)

We follow the approach of Section 3.4. If $r = 0_{\mathbb{F}_q}$, then we search for the unique $n \in \mathcal{N} = \{0, 1, \ldots, N\}$ such that $S_{n,d} = \emptyset$ (as opposed to $\neg$). This process takes $O(N) \subseteq O(d\,2^{d^2\,\mathrm{mpe}(q-1)+d})$ bit operations – see statement (2) for this bound on N. Once n has been found, one may simply read off the description $\mathfrak{D}_n$ of $\mathfrak{T}_n = \mathrm{Tree}_{\Gamma_f}(0_{\mathbb{F}_q})$ and output the description $[\mathfrak{D}_n]$ of the cyclic sequence in question, which takes $O(d\,2^{d^2\,\mathrm{mpe}(q-1)+d}\log q)$ bit operations for copying.

Henceforth, we assume that $r \neq 0_{\mathbb{F}_q}$. We recall from the proof of statement (4) of Lemma 5.1.6 that there is a natural choice ω for a primitive element of $\mathbb{F}_q$. We compute $\mathfrak{l} := \log_\omega(r)$ in a single fdl query (if (r, l) is obtained as an element of the CRL-list of f that is parametrized by the algorithm for Problem 1, then r is literally specified as a power of ω, and this computation may be skipped). Then we set $i := \mathfrak{l} \bmod d$ (taking $O(\log^{1+o(1)} q)$ bit operations to compute by Lemma 5.1.5 (3)), so that $r \in C_i$, and determine the cycle $(i_0, i_1, \ldots, i_{\ell-1})$ of $i = i_0$ under $\bar{f}$, which costs $O(d \log d)$ bit operations (see also the beginning of the argument for statement (1)). Moreover, we set

$$r'_{i_0} = r'_i := \iota_i^{-1}(r) = \frac{\mathfrak{l} - i}{d} \in \mathbb{Z}/s\mathbb{Z} \quad \text{and} \quad r'_{i_t} := A_{i_{t-1}}(r'_{i_{t-1}}) \text{ for } t = 1, 2, \ldots, \ell - 1,$$

which takes $O(d \log^{1+o(1)} q)$ bit operations altogether. Finally, for $t = 0, 1, \ldots, \ell - 1$, we compute

$$\mathcal{A}_{i_t} := A_{i_t} A_{i_{(t+1) \bmod \ell}} \cdots A_{i_{(t+\ell-1) \bmod \ell}},$$

which takes $O(d^2 \log^{1+o(1)} q)$ bit operations for all t together (see also the beginning of the argument for statement (1)).

Until further notice, we assume that t is fixed. In the notation of Section 3.4, we have $\mathcal{A}_{i_t}(x) = \overline{\alpha}_{i_t} x + \overline{\beta}_{i_t}$. We compute $\gcd(\overline{\alpha}_{i_t}, s)$, find the smallest non-negative integer L_{i_t} such that $\gcd(\overline{\alpha}_{i_t}^{L_{i_t}}, s) = \gcd(\overline{\alpha}_{i_t}^{L_{i_t}+1}, s) =: s''_{i_t}$, compute $s'_{i_t} := s/s''_{i_t}$ and $\mathfrak{f}_{i_t} := \sum_{z=0}^{L_{i_t}-1} \overline{\alpha}_{i_t}^z \overline{\beta}_{i_t} \bmod s''_{i_t}$ (the unique periodic point of the reduction of $\mathcal{A}_{i_t}$ modulo s''_{i_t}). Together, these tasks can be performed within $O(\log^{3+o(1)} q)$ bit operations if binary search is used to find L_{i_t} (see also the paragraph in the proof of statement (1), where L_i is introduced).

In what follows, we focus on the $\mathcal{A}_{i_t}$-cycle of r'_{i_t} (which corresponds to the cycle of $f^t(r)$ under f^ℓ). For each $j \in \{1, 2, \ldots, m_{i_t}\}$, we compute the affine discrete logarithm

$$\mathfrak{l}_{i_t,j} := \log_{\mathcal{A}_{i_t}}^{(\mathfrak{a}_{i_t,j})}(r'_{i_t}, \mathfrak{b}_{i_t,j})$$

(as defined in Section 2.4), which takes $O(m_{i_t}) \subseteq O(d^2\,\mathrm{mpe}(q-1))$ mdl queries and $O(m_{i_t} \log^{1+o(1)} q) \subseteq O(d^2\,\mathrm{mpe}(q-1)\log^{1+o(1)} q)$ bit operations by the reduction

argument in Section 2.4. For the bound $m_{i_t} \in O(d^2 \operatorname{mpe}(q-1))$, we recall from Section 3.3 that

$$\mathcal{P}_{i_t} = \mathcal{Q}_{i_t, H_{i_t}} = \bigwedge_{k=0}^{H_{i_t}} \lambda_{i_{t-k}}^k(\mathcal{R}_{i_{t-k}}) \wedge \mathcal{U}_{i_t},$$

and that the length of the "standard" spanning congruence sequence of $\lambda_{i_{t-k}}^k(\mathcal{R}_{i_{t-k}})$ (stored in the partition-tree register as $\mathcal{X}_{i_t,k}$) is $n_{i_{t-k}} \in O(d)$, while the one of $\mathcal{U}_{i_t} = \mathfrak{P}(\theta_{i_t,h}(x) : h = 0, 1, \ldots, H_{i_t})$, which is stored as $\mathcal{X}_{i_t,-1}$, has length H_i. Moreover, $H_i \in O(d \operatorname{mpe}(q-1))$.

In addition to computing $\mathfrak{l}_{i_t,j}$ for every $j \in \{1, 2, \ldots, m_{i_t}\}$, we also compute, for each j such that $\mathfrak{l}_{i_t,j} < \infty$, the cycle length $l_{i_t,j}$ of r'_{i_t} under $\mathcal{A}_{i_t}$ modulo $\mathfrak{a}_{i_t,j}$ – again, by the reduction argument of Section 2.4, this takes $O(m_{i_t}) \subseteq O(d^2 \operatorname{mpe}(q-1))$ mord queries and $O(m_{i_t} \log^{1+o(1)} q) \subseteq O(d^2 \operatorname{mpe}(q-1) \log^{1+o(1)} q)$ bit operations.

Now, observe that by the definition of the affine discrete logarithm, for each index $j \in \{1, 2, \ldots, m_{i_t}\}$, the equality $\mathfrak{l}_{i_t,j} = \infty$ is equivalent to the j-th spanning congruence of $\mathcal{P}_{i_t}$, $x \equiv \mathfrak{b}_{i_t,j} \pmod{\mathfrak{a}_{i_t,j}}$, being constantly false along the $\mathcal{A}_{i_t}$-cycle of r'_{i_t}. In that case, we say that the index j is *of type* I and set $v_j := \neg$.

Moreover, if $\mathfrak{l}_{i_t} < \infty$, then the congruence $x \equiv \mathfrak{b}_{i_t,j} \pmod{\mathfrak{a}_{i_t,j}}$ is true for exactly one point on the *modular* $\mathcal{A}_{i_t}$-cycle of r'_{i_t} modulo $\mathfrak{a}_{i_t,j}$. Hence, for each index $j \in \{1, 2, \ldots, m_{i_t}\}$, this congruence is constantly true along the (non-modular) $\mathcal{A}_{i_t}$-cycle of r'_{i_t} if and only if $\mathfrak{l}_{i_t,j} < \infty$ and $l_{i_t,j} = 1$. In that case, we say that the index j is *of type* II and set $v_j := \emptyset$.

An index $j \in \{1, 2, \ldots, m_{i_t}\}$ that is neither of type I nor of type II is said to be *of type* III. For such indices, we leave the value of v_j open. Note that in the notation of Section 3.4, the type-III indices $j \in \{1, 2, \ldots, m_{i_t}\}$ are precisely the elements of the set I_{i_t} (of indices for which the truth value of the j-th spanning congruence of $\mathcal{P}_{i_t}$ is constant along the $\mathcal{A}_{i_t}$-cycle of r'_{i_t}).

We note that there are other, sufficient but not necessary conditions for the j-th spanning congruence of $\mathcal{P}_{i_t}$ having a constant truth value along the $\mathcal{A}_{i_t}$-cycle of r'_{i_t}, based on the observation (already made in Section 3.4) that each point on that cycle is congruent to $\mathfrak{f}_{i_t}$ modulo s''_{i_t}:

- if $\mathfrak{f}_{i_t} \not\equiv \mathfrak{b}_{i_t,j} \pmod{\gcd(\mathfrak{a}_{i_t,j}, s''_{i_t})}$, then the j-th spanning congruence of $\mathcal{P}_{i_t}$ is always false along the cycle.
- if $\mathfrak{a}_{i_t,j} \mid s''_{i_t}$ and $\mathfrak{f}_{i_t} \equiv \mathfrak{b}_{i_t,j} \pmod{\mathfrak{a}_{i_t,j}}$, then the j-th spanning congruence of $\mathcal{P}_{i_t}$ is always true along the cycle.

Those conditions do not require a quantum computer to be checked efficiently, and in practice, they may speed up an implementation of the algorithm we are currently discussing. However, they do not improve the worst-case complexity of the algorithm, and so we will ignore them in this theoretical discussion.

We recall that the f-cycle length l of r is given to us as part of the input. Associated with each $j \in \{1, 2, \ldots, m_{i_t}\} \setminus I_{i_t}$, we have the (l/ℓ)-congruence $y \equiv \mathfrak{l}_{i_t,j} \pmod{l_{i_t,j}}$, which holds precisely for those $y \in \mathbb{Z}/(l/\ell)\mathbb{Z}$ such that the j-th spanning congruence of $\mathcal{P}_{i_t}$ becomes true when substituting $x := \mathcal{A}_{i_t}^{y}(r'_{i_t})$. As in Section 3.4, we set

$$\mathcal{P}^{(i_t)} := \mathfrak{P}(y \equiv \mathfrak{l}_{i_t,j} \pmod{l_{i_t,j}} : j \in \{1, 2, \ldots, m_{i_t}\} \setminus I_{i_t}),$$

an arithmetic partition of $\mathbb{Z}/(l/\ell)\mathbb{Z}$. The logical sign tuples which parametrize the blocks of $\mathcal{P}^{(i_t)}$ are from the set $\{\emptyset, \neg\}^{\{1,2,\ldots,m_{i_t}\}\setminus I_{i_t}}$ and are of the form

$$\vec{\nu} = (\nu_j)_{j \in \{1,\ldots,m_{i_t}\}\setminus I_{i_t}}.$$

For such a tuple $\vec{\nu}$, we set $\vec{\nu}^{+} := (\nu_j)_{j=1,2,\ldots,m_{i_t}}$ (the positions indexed by $j \in I_{i_t}$ are filled with the constant logical signs for such j that were defined above). Conversely, for $\vec{\nu'} \in \{\emptyset, \neg\}^{m_{i_t}}$, we denote by $\vec{\nu'}^{-}$ the projection of $\vec{\nu'}$ to $\{\emptyset, \neg\}^{\{1,2,\ldots,m_{i_t}\}\setminus I_{i_t}}$.

It follows that if $y \in \mathbb{Z}/(l/\ell)\mathbb{Z}$ is contained in the block $\mathcal{B}(\mathcal{P}^{(i_t)}, \vec{\nu})$ of $\mathcal{P}^{(i_t)}$, then $\mathcal{A}_{i_t}^{y}(r'_{i_t})$ is always contained in the block $\mathcal{B}(\mathcal{P}_{i_t}, \vec{\nu}^{+})$ of $\mathcal{P}_{i_t}$, whence

$$\mathrm{Tree}_{\Gamma_f}(f^{t+\ell y}(r)) = \mathfrak{T}_{n'_{i_t}(\vec{\nu})},$$

where $n'_{i_t}(\vec{\nu})$ is the unique $n \in \{0, 1, \ldots, N\}$ such that $\vec{\nu}^{+} \in S_{i_t,n}$.

We remind the reader that we wish to output a compact description of the cyclic sequence of rooted tree isomorphism types that characterizes the connected component of Γ_f containing r. In order to do so, we compute sets $\overline{S}_{n,i_t}$ for $n \in \{0, 1, \ldots, N\}$ (and our still fixed t) that are defined as follows. The elements of $\overline{S}_{n,i_t}$ are precisely those logical sign tuples $\vec{\nu} \in \{\emptyset, \neg\}^{I_{i_t}}$ for which $n'_{i_t}(\vec{\nu}) = n$.

Let us describe how to compute these sets $\overline{S}_{n,i_t}$. We go through the $N + 1 \in O(d\,2^{d^2\,\mathrm{mpe}(q-1)+d})$ values for n (see also the argument for $r = 0_{\mathbb{F}_q}$ at the beginning of this subsection), and for each n, we go through the $|S_{n,i_t,H_{i_t}}|$ logical sign tuples $\vec{\nu''}$ in $S_{n,i_t,H_{i_t}}$ (the last entry of S_{n,i_t}) in their listed order, which is lexicographic. For each $\vec{\nu''}$, we set $\vec{\nu'} := \vec{\nu''} \diamond \vec{\xi}_{i_t,H_{i_t}} = (\nu'_j)_{j=1,2,\ldots,m_{i_t}}$. We check whether $\nu'_j = \nu_j$ for all $j \in I_{i_t}$ (which takes $O(m_{i_t} \log q) \subseteq O(d^2\,\mathrm{mpe}(q-1)\log q)$ bit operations). If not, we move on to the next $\vec{\nu''}$ and associated $\vec{\nu'}$, otherwise we add $\vec{\nu'}^{-}$ to $\overline{S}_{n,i_t}$ as a new element (again taking $O(m_{i_t} \log q) \subseteq O(d^2\,\mathrm{mpe}(q-1)\log q)$ bit operations, for copying). Because $N \in O(d\,2^{d^2\,\mathrm{mpe}(q-1)+d})$ and $\sum_{n=0}^{N} |S_{n,i_t,H_{i_t}}| \leq 2^{(H_{i_t}+1)d} \leq 2^{d^2\,\mathrm{mpe}(q-1)+d}$, and we only need $O(1)$ bit operations to deal with an n for which $S_{n,i_t} = \emptyset$, we conclude that the overall bit operation cost of computing the sets $\overline{S}_{n,i_t}$ (for our fixed t) is in

$$O(d\,2^{d^2\,\mathrm{mpe}(q-1)+d} + m_{i_t}\,2^{(H_{i_t}+1)d}\log q) \subseteq O(d^2\,\mathrm{mpe}(q-1)2^{d^2\,\mathrm{mpe}(q-1)+d}\log q),$$

and the constructed arrays representing those sets are lexicographically ordered.

Let us now finally "unfix" t again. For all the $O(d)$ values of $t \in \{0, 1, \ldots, \ell-1\}$ together, a q-bounded query complexity of all computations we described after agreeing to fix t is

$$(d \log^{3+o(1)} q + d^3 \operatorname{mpe}(q-1) \log^{1+o(1)} q + d^3 \operatorname{mpe}(q-1) 2^{d^2 \operatorname{mpe}(q-1)+d} \log q, \\ 0, d^3 \operatorname{mpe}(q-1), d^3 \operatorname{mpe}(q-1), 0). \tag{5.4}$$

Here, the first summand in the bit operation component, as well as the specified queries, cover the cost of everything except the computation of the sets $\overline{S}_{n,i_t}$ themselves, which is instead covered by the second summand in the bit operation component (without needing any queries). Considering the query complexities of all other involved computations (see the detailed steps of the algorithm below), we find that only the second entry, 0, in formula (5.4) needs to be replaced by d in order to obtain a valid q-bounded query complexity of the entire algorithm for solving Problem 3.

As in the previous two subsections, we conclude with some pseudocode for this algorithm, which includes q-bounded query complexities for each step.

1 If $r = 0_{\mathbb{F}_q}$, then search for the unique $n \in \mathcal{N} = \{0, 1, \ldots, N\}$ such that $S_{n,d} = \emptyset$, and output the following and halt: "The cyclic sequence of rooted tree isomorphism types which encodes the digraph isomorphism type of the connected component in question is $[\mathfrak{D}_n]$."
QC: $(d 2^{d^2 \operatorname{mpe}(q-1)+d} \log q, 0, 0, 0, 0)$.

2 Compute the induced function $\bar{f}$ on $\{0, 1, \ldots, d\}$ and the affine maps A_i of $\mathbb{Z}/s\mathbb{Z}$.
QC: $(d \log^{1+o(1)} q, d, 0, 0, 0)$.

3 Let ω be the "natural" choice of primitive element of $\mathbb{F}_q$ (see the proof of Lemma 5.1.6 (4)), and determine $\mathfrak{l} := \log_\omega(r)$.
QC: $(\log q, 1, 0, 0, 0)$.

4 Set $i := \mathfrak{l} \bmod d$.
QC: $(\log^{1+o(1)} q, 0, 0, 0, 0)$.

5 Determine the cycle $(i_0, i_1, \ldots, i_{\ell-1})$ of $i = i_0$ under $\bar{f}$.
QC: $(d \log d, 0, 0, 0, 0)$.

6 Compute $r'_{i_0} := (\mathfrak{l} - i)/d$ and $r'_{i_t} := A_{i_{t-1}}(r'_{i_{t-1}})$ for $t = 1, 2, \ldots, \ell - 1$.
QC: $(d \log^{1+o(1)} q, 0, 0, 0, 0)$.

7 For $t = 0, 1, \ldots, \ell - 1$, compute $\mathcal{A}_{i_t} := A_{i_t} A_{i_{(t+1) \bmod \ell}} \cdots A_{i_{(t+\ell-1) \bmod \ell}} : x \mapsto \overline{\alpha}_{i_t} x + \overline{\beta}_{i_t}$.
QC: $(d^2 \log^{1+o(1)} q, 0, 0, 0, 0)$.

8 For each $t = 0, 1, \ldots, \ell - 1$, do the following. QC:

$$(d \log^{3+o(1)} q + d^3 \operatorname{mpe}(q-1) \log^{1+o(1)} q + d^3 \operatorname{mpe}(q-1) 2^{d^2 \operatorname{mpe}(q-1)+d} \log q, \\ 0, d^3 \operatorname{mpe}(q-1), d^3 \operatorname{mpe}(q-1), 0).$$

8.1 Compute $\gcd(\overline{\alpha}_{i_t}, s)$.
QC: $(\log^{1+o(1)} q, 0, 0, 0, 0)$.

8.2 Find the smallest non-negative integer L_{i_t} such that

$$\gcd(\overline{\alpha}_{i_t}^{L_{i_t}}, s) = \gcd(\overline{\alpha}_{i_t}^{L_{i_t}+1}, s) =: s''_{i_t}.$$

QC: $(\log^{3+o(1)} q, 0, 0, 0, 0)$.

8.3 Compute $s'_{i_t} := s/s''_{i_t}$.
QC: $(\log^{1+o(1)} q, 0, 0, 0, 0)$.

8.4 Compute $\mathfrak{f}_{i_t} := \sum_{z=0}^{L_{i_t}-1} \overline{\alpha}_{i_t}^z \overline{\beta}_{i_t} \bmod s''_{i_t}$. In doing so, do *not* add up all the summands, but use the geometric sum formula

$$\sum_{z=0}^{L_{i_t}-1} \overline{\alpha}_{i_t}^z = \begin{cases} \frac{\overline{\alpha}_{i_t}^{L_{i_t}}-1}{\overline{\alpha}_{i_t}-1}, & \text{if } \overline{\alpha}_{i_t} \neq 1, \\ L_{i_t}\overline{\alpha}_{i_t}, & \text{if } \overline{\alpha}_{i_t} = 1, \end{cases}$$

and the fact that $L_{i_t} \in O(\log q)$.
QC: $(\log^{2+o(1)} q, 0, 0, 0, 0)$.

8.5 Read off $\mathcal{P}_{i_t} = \mathfrak{P}(x \equiv \mathfrak{b}_{i_t,j} \pmod{\mathfrak{a}_{i_t,j}} : j = 1, 2, \ldots, m_{i_t})$ from the given partition-tree register.
QC: $(d^2 \operatorname{mpe}(q-1) \log q, 0, 0, 0, 0)$.

8.6 For $j = 1, 2, \ldots, m_{i_t}$, do the following.
QC: $(d^2 \operatorname{mpe}(q-1) \log^{1+o(1)} q, 0, d^2 \operatorname{mpe}(q-1), d^2 \operatorname{mpe}(q-1), 0)$.

8.6.1 Compute the affine discrete logarithm

$$\mathfrak{l}_{i_t,j} := \log_{\mathcal{A}_{i_t}}^{(\mathfrak{a}_{i_t,j})}(r'_{i_t}, \mathfrak{b}_{i_t,j}).$$

If $\mathfrak{l}_{i_t,j} = \infty$, set $\nu_j := \neg$, $\mathrm{Type}_{\mathrm{I}}(j) := \mathrm{true}$, $\mathrm{Type}_{\mathrm{II}}(j) := \mathrm{false}$ and

$$\mathrm{Type}_{\mathrm{III}}(j) := \mathrm{false}.$$

Otherwise, just set $\mathrm{Type}_{\mathrm{I}}(j) := \mathrm{false}$.
QC: $(\log^{1+o(1)} q, 0, 1, 0, 0)$.

8.6.2 If $\mathrm{Type}_{\mathrm{I}}(j) = \mathrm{false}$, compute the cycle length $l_{i_t,j}$ of r'_{i_t} under $\mathcal{A}_{i_t}$ modulo $\mathfrak{a}_{i_t,j}$. If $l_{i_t,j} = 1$, set $\nu_j := \emptyset$, $\mathrm{Type}_{\mathrm{II}}(j) := \mathrm{true}$ and $\mathrm{Type}_{\mathrm{III}}(j) := \mathrm{false}$. Otherwise, set $\mathrm{Type}_{\mathrm{II}}(j) := \mathrm{false}$ and $\mathrm{Type}_{\mathrm{III}}(j) := \mathrm{true}$.
QC: $(\log^{1+o(1)} q, 0, 0, 1, 0)$.

8.7 Set

$$\mathcal{P}^{(i_t)} := \mathfrak{P}(y \equiv l_{i_t,j} \pmod{\mathfrak{l}_{i_t,j}} : 1 \leq j \leq m_{i_t}, \mathrm{Type}_{\mathrm{III}}(j) = \mathrm{true}).$$

QC: $(d^2 \operatorname{mpe}(q-1) \log q, 0, 0, 0, 0)$.

8.8 For each $n \in \mathcal{N} = \{0, 1, \ldots, N\}$, do the following.
QC: $(d^2 \operatorname{mpe}(q-1)2^{d^2 \operatorname{mpe}(q-1)+d} \log q, 0, 0, 0, 0)$.

8.8.1 Initialize the set $\overline{S}_{n,i_t}$ to be $\emptyset$.
QC: $(\log q, 0, 0, 0, 0)$.

8.8.2 For each $\vec{\nu}'' \in S_{n,i_t,H_{i_t}}$ (last entry of S_{n,i_t} from the partition-tree register from the input), do the following.
QC: $(d^2 \operatorname{mpe}(q-1)2^{d^2 \operatorname{mpe}(q-1)+d} \log q, 0, 0, 0, 0)$.

8.8.2.1 Set $\vec{\nu}' := \vec{\nu}'' \diamond \vec{\xi}_{i_t,H_{i_t}} = (\nu'_j)_{j=1,2,\ldots,m_{i_t}}$.
QC: $(d^2 \operatorname{mpe}(q-1) \log q, 0, 0, 0, 0)$.

8.8.2.2 Check whether $\nu'_j = \nu_j$ for all $j \in \{1, 2, \ldots, m_{i_t}\}$ such that one of $\text{Type}_{\text{I}}(j)$, $\text{Type}_{\text{II}}(j)$ or $\text{Type}_{\text{III}}(j)$ is true. If so, add $\vec{\nu}'^{-}$ (which is $\vec{\nu}'$ with all components corresponding to one of the three types deleted) to $\overline{S}_{n,i_t}$ as a new element.
QC: $(d^2 \operatorname{mpe}(q-1) \log q, 0, 0, 0, 0)$.

9 Output the following: "Consider an iterate $f^z(r)$, where $z \in \mathbb{Z}/l\mathbb{Z}$. Let $t := z \bmod \ell$ and $y := (z-t)/\ell \in \mathbb{Z}/(l/\ell)\mathbb{Z}$. Depending on t, we have the arithmetic partition $\mathcal{P}^{(i_t)}$ of $\mathbb{Z}/(l/\ell)\mathbb{Z}$, and the block of $\mathcal{P}^{(i_t)}$ in which y is contained controls the isomorphism type of $\text{Tree}_{\Gamma_f}(f^z(r))$. More precisely, for each $n \in \mathcal{N}$, we have the set $\overline{S}_{n,i_t}$ of logical sign tuples such that $\text{Tree}_{\Gamma_f}(f^z(r)) = \mathfrak{T}_n$ if and only if $y \in \mathcal{B}(\mathcal{P}^{(i_t)}, \vec{\nu})$ for some $\vec{\nu} \in \overline{S}_{n,i_t}$. The arithmetic partitions $\mathcal{P}^{(i_t)}$ and associated logical sign tuple sets $\overline{S}_{n,i_t}$ are as follows.", followed by printing (the computed spanning congruence sequence of) $\mathcal{P}^{(i_t)}$ and the sets $\overline{S}_{n,i_t}$ for each $t = 0, 1, \ldots, \ell - 1$. Then halt.
QC: $(d^3 \operatorname{mpe}(q-1)2^{d^2 \operatorname{mpe}(q-1)+d} \log q, 0, 0, 0, 0)$.

5.3 The isomorphism problem for functional graphs of generalized cyclotomic mappings

In this section, we consider the algorithmic problem of deciding whether the functional graphs of two given generalized cyclotomic mappings of $\mathbb{F}_q$ (each of a fixed index, but not necessarily both of the same index) are isomorphic digraphs. We note that thanks to Babai [9] (see also Helfgott's expository article [32], or its English translation [31]), a general algorithm for deciding the isomorphism of (di)graphs is known that is quasi-polynomial in the number of vertices. Assuming that the graphs in question are given by specifying their edges (as ordered or unordered pairs of vertices) individually, this algorithm is quasi-polynomial in the input length (one can assume without loss of generality that there are no isolated vertices, and thus that the number of vertices and the number of edges are within quadratic bounds of each other).

However, we do not specify our functional graphs Γ_f in this form; rather, we specify f in its cyclotomic form (1.1), so our input length is only in $O(d \log q)$, while the vertex number is $|\mathbb{F}_q| = q$. We believe that it is a hard problem to decide whether the functional graphs of two generalized cyclotomic mappings of $\mathbb{F}_q$, each of an index that is at most d, are isomorphic (i.e., that this problem is not generally solvable in polynomial time in $\log q$ for fixed d). However, for some special cases, efficient algorithms can be developed, and it is the purpose of this section to present such special cases and the associated decision algorithms.

5.3.1 Special case: Index 1

Generalized cyclotomic mappings of $\mathbb{F}_q$ of index 1 are the same as monomial mappings $x \mapsto ax^r$, where $a \in \mathbb{F}_q$ and $r \in \{1, \dots, q-1\}$ (if one has $r_0 = 0$ in formula (1.1) with $d = 1$, then one may replace it by $q-1$ to get a formula that works on all of $\mathbb{F}_q$). To get the class of all monomial mappings of $\mathbb{F}_q$, one must include the case "$r = 0$" (in which $0_{\mathbb{F}_q}$ is not necessarily fixed), and we do so in this subsection. Thanks to earlier work of Deng [21], it is easy to decide whether two monomial mappings of $\mathbb{F}_q$, say $f : x \mapsto ax^r$ and $f' : x \mapsto a'x^{r'}$, have isomorphic functional graphs, as we explain below.

First, we note that if f is constant, which happens if and only if $r = 0$ or $a = 0_{\mathbb{F}_q}$, then $\Gamma_f \cong \Gamma_{f'}$ if and only if f' is constant as well, i.e., if and only if $r' = 0$ or $a' = 0_{\mathbb{F}_q}$. Checking whether this special case applies only takes $O(\log q)$ bit operations (for scanning the values of a, a', r, r').

We may thus assume that r, r', a, a' all are non-zero. Because $a, a' \neq 0_{\mathbb{F}_q}$, we have $f(\mathbb{F}_q^*) \cup f'(\mathbb{F}_q^*) \subseteq \mathbb{F}_q^*$. Following the procedure described in our introduction, we can associate with f, respectively, f', an affine map A, respectively, A', of $\mathbb{Z}/(q-1)\mathbb{Z}$ such that $\Gamma_f \cong \Gamma_{f'}$ if and only if $\Gamma_A \cong \Gamma_{A'}$. Computing A and A' has q-bounded query complexity $(\log^{1+o(1)} q, 1, 0, 0, 0)$; beside some simple arithmetic, it requires the computation of the discrete logarithms of a and a' with base ω (the "natural choice" of primitive element of $\mathbb{F}_q$ – see the proof of Lemma 5.1.6 (4)).

In order to decide whether $\Gamma_A \cong \Gamma_{A'}$, we use the following result, which is a variant of [21, Theorem 11].

Theorem 5.3.1.1. *Let $m = p_1^{v_1} \cdots p_K^{v_K}$ be a positive integer with its prime factorization displayed. Let $a, a', b, b' \in \mathbb{Z}/m\mathbb{Z}$, and let us denote by A, respectively, A', the affine map $x \mapsto ax + b$, respectively, $x \mapsto a'x + b'$, of $\mathbb{Z}/m\mathbb{Z}$. Moreover, we define*

- *$\mathfrak{Y} := \{p_j : j \in \{1, 2, \dots, K\}, p_j \nmid a, A \bmod p_j^{v_j}$ has a fixed point$\}$, and analogously for $\mathfrak{Y}'$, with a' and A' in place of a and A; and*
- *l, respectively, l', to be the minimal cycle length of A, respectively, A', on its respective periodic points.*

Then $\Gamma_A \cong \Gamma_{A'}$ *if and only if both of the following hold:*

(1) $\gcd(a, m) = \gcd(a', m)$, *and*

(2) $\mathfrak{Y} = \mathfrak{Y}'$, $l = l'$, $\operatorname{ord}_{p^{\nu_p(m)}}(a^l) = \operatorname{ord}_{p^{\nu_p(m)}}((a')^l)$ *for all* $p \in \mathfrak{Y}$, *and if* $2 \in \mathfrak{Y}$ *and* $\nu_2(m) > 1$, *then* $\operatorname{ord}_4(a^l) = \operatorname{ord}_4((a')^l)$.

Proof. This is the same as [21, Theorem 11] except that in condition (2), we do not demand that $\operatorname{ord}_t(a^l) = \operatorname{ord}_t((a')^l)$ for *all* divisors t of m whose prime divisors are in $\mathfrak{Y}$. However, our condition (2) is enough, because if it holds and $t = \prod_{p\in\mathfrak{Y}} p^{\nu'_p}$ divides m, then for all $p \in \mathfrak{Y}$, one has

$$\operatorname{ord}_{p^{\nu'_p}}(a^l) = \operatorname{ord}_{p^{\nu'_p}}((a')^l), \tag{5.5}$$

and thus

$$\operatorname{ord}_t(a^l) = \operatorname{lcm}_{p\in\mathfrak{Y}} \operatorname{ord}_{p^{\nu'_p}}(a^l) = \operatorname{lcm}_{p\in\mathfrak{Y}} \operatorname{ord}_{p^{\nu'_p}}((a')^l) = \operatorname{ord}_t((a')^l).$$

To see that formula (5.5) holds, we make a case distinction.

- If $p > 2$, let us fix a primitive root r of $\mathbb{Z}/p^{\nu_p(m)}\mathbb{Z}$. Because

  $$\operatorname{ord}_{p^{\nu_p(m)}}(a^l) = \operatorname{ord}_{p^{\nu_p(m)}}((a')^l),$$

  it follows that modulo $p^{\nu_p(m)}$, we have $a^l = \mathrm{r}^k$ and $(a')^l = \mathrm{r}^{k'}$, where

  $$\gcd(k, \phi(p^{\nu_p(m)})) = \gcd(k', \phi(p^{\nu_p(m)})).$$

  Since $\phi(p^{\nu'_p})$ divides $\phi(p^{\nu_p(m)})$, it follows that

  $$\gcd(k, \phi(p^{\nu'_p})) = \gcd(k', \phi(p^{\nu'_p})),$$

  whence a^l and $(a')^l$ are also of the same multiplicative order modulo $p^{\nu'_p}$, as required.

- If $p = 2$, then formula (5.5) holds by assumption if $\nu_2(m) \le 2$, so let us assume that $\nu_2(m) \ge 3$. We may also assume that $\nu'_2 < \nu_2(m)$, because there is nothing to show if $\nu'_2 = \nu_2(m)$. By the structure of $\mathbb{Z}/2^{\nu_2(m)}\mathbb{Z}$, modulo $2^{\nu_2(m)}$ we can write $a^l = (-1)^{\varepsilon}5^k$ and $(a')^l = (-1)^{\varepsilon'}5^{k'}$ with $\varepsilon, \varepsilon' \in \{0, 1\}$. Since a^l and $(a')^l$ have (by assumption) the same multiplicative order modulo 4, we infer that $\varepsilon = \varepsilon'$. It is not hard to see that if $\operatorname{ord}_{2^{\nu_2(m)}}(a^l) = \operatorname{ord}_{2^{\nu_2(m)}}((a')^l) \in \{1, 2\}$, then $\operatorname{ord}_{2^{\nu'_2}}(a^l) = \operatorname{ord}_{2^{\nu'_2}}((a')^l) = 1$. Indeed, this is clear if both orders modulo $2^{\nu_2(m)}$ are equal to 1, and if both orders modulo $2^{\nu_2(m)}$ are equal to 2, then $\operatorname{ord}_{2^{\nu_2(m)}}(5^k), \operatorname{ord}_{2^{\nu_2(m)}}(5^{k'})$ both divide 2, whence due to $\nu'_2 < \nu_2(m)$, one has $5^k \equiv 5^{k'} \equiv 1 \pmod{2^{\nu'_2}}$, as follows by comparing the multiplicative orders of 5

modulo $2^{v_2(m)}$ and modulo $2^{v_2'}$, respectively. We may thus assume that the common multiplicative order modulo $2^{v_2(m)}$ of a^l and $(a')^l$ is strictly greater than 2. Then

$$\operatorname{ord}_{2^{v_2(m)}}(5^k) = \operatorname{ord}_{2^{v_2(m)}}(a^l) = \operatorname{ord}_{2^{v_2(m)}}((a')^l) = \operatorname{ord}_{2^{v_2(m)}}(5^{k'}),$$

and with an analogous argument to the one for "$p > 2$", we conclude that

$$\operatorname{ord}_{2^{v_2'}}(5^k) = \operatorname{ord}_{2^{v_2'}}(5^{k'}).$$

Therefore,

$$\begin{aligned}\operatorname{ord}_{2^{v_2'}}(a^l) &= \operatorname{lcm}(\operatorname{ord}_{2^{v_2'}}((-1)^\varepsilon), \operatorname{ord}_{2^{v_2'}}(5^k)) \\ &= \operatorname{lcm}(\operatorname{ord}_{2^{v_2'}}((-1)^\varepsilon), \operatorname{ord}_{2^{v_2'}}(5^{k'})) \\ &= \operatorname{ord}_{2^{v_2'}}((a')^l),\end{aligned}$$

as required. ∎

Using Theorem 5.3.1.1, we can prove the following.

Corollary 5.3.1.2. *Let m be a positive integer, and let A and A' be affine maps of $\mathbb{Z}/m\mathbb{Z}$. Deciding whether $\Gamma_A \cong \Gamma_{A'}$ takes m-bounded query complexity*

$$(\log^{2+o(1)} m, 0, \log m, 1, 0)$$

and m-bounded Las Vegas dual complexity

$$(\log^{8+o(1)} m, \log^{4+o(1)} m, \log^2 m).$$

Proof. We use the notation from Theorem 5.3.1.1, including that $A(x) = ax + b$ and $A'(x) = a'x + b'$. We argue that the conditions given in Theorem 5.3.1.1 can be verified within the specified m-bounded query complexity (the asserted Las Vegas dual complexity then follows readily using Lemma 5.1.7). First, we compute and compare $\gcd(a, m)$ and $\gcd(a', m)$, which takes $O(\log^{1+o(1)} m)$ bit operations by Lemma 5.1.5 (8). Next, we factor $m = p_1^{v_1} \cdots p_K^{v_K}$ using a single m-bounded mdl query (see also the beginning of Section 5.2).

Following that, we compute the sets $\mathfrak{Y}$ and $\mathfrak{Y}'$ and compare them. For this, we observe that $A \bmod p_j^{v_j}$, respectively, $A' \bmod p_j^{v_j}$, has a fixed point if and only if the congruence $(a-1)x \equiv -b \pmod{p_j^{v_j}}$, respectively, $(a'-1)x \equiv -b' \pmod{p_j^{v_j}}$, is solvable, which holds if and only if $\gcd(a-1, p_j^{v_j}) \mid b$, respectively, $\gcd(a'-1, p_j^{v_j}) \mid b'$. To check whether this holds, we read off the binary representation of $p_j^{v_j}$ from the output of the above mdl query, then carry out the relevant arithmetic in either case, which takes $O(\log^{1+o(1)} m)$ bit operations for a single j by Lemma 5.1.5 (1,3,8).

Because there are $O(\log m)$ distinct values of j, it takes $O(\log^{2+o(1)} m)$ bit operations altogether to compute $\mathfrak{Y}$ and $\mathfrak{Y}'$ and check whether they are equal.

If $\mathfrak{Y} = \mathfrak{Y}'$, we next compute the minimal cycle lengths l and l' and check if they are equal. It follows from our Table 2.2 (or [15, Tables 3 and 4], in which cycle types, not CRL-lists, of affine maps of finite primary cyclic groups are displayed and from which the cycle lengths can be read off more directly) that modulo a prime power, the cycle lengths of an affine permutation are linearly ordered under divisibility. Therefore, l, respectively, l', is the least common multiple of the smallest cycle lengths of A, respectively of A', modulo the $p_j^{v_j}$ for those $j \in \{1, 2, \dots, K\}$ such that $p_j \nmid a$, respectively, $p_j \nmid a'$. Those minimal cycle lengths can be computed according to our Table 2.2 (or [15, Tables 3 and 4]). More specifically, we go through $j = 1, 2, \dots, K$, and for each of these values, we do the following.

- We check whether $p_j \nmid a$, respectively, $p_j \nmid a'$, taking $O(\log^{1+o(1)} m)$ bit operations.
- If so, we check which case in Table 2.2 (or [15, Table 3 or 4, respectively]) applies, which also takes $O(\log^{1+o(1)} m)$ bit operations.
- Finally, we compute the minimal cycle length according to the case-specific formula and factor it, using one mdl query and $O(\log^{1+o(1)} m)$ bit operations.

For all j together, this process takes m-bounded query complexity

$$(\log^{2+o(1)} m, 0, \log m, 0, 0).$$

Following this, we determine the least common multiple of the cycle lengths. These lengths are already factored, so one only needs to go through the $O(\log m)$ primes dividing at least one of those cycle lengths (we note that each of those primes is a divisor of $p(p-1)$ for some prime $p \mid m$), and for each of them, we compute the largest exponent with which it occurs. For this, we need to scan the obtained factorization of the minimal cycle length modulo $p_j^{v_j}$, which is a bit string of length in $O(\log p_j^{v_j})$, for each j, and we need to compare the stored intermediate maximum with one of the prime exponents in it, which is a bit string of length in $O(\log\log p_j^{v_j})$. Altogether, the computation of l and l' takes $O(\log^{2+o(1)} m)$ bit operations, and checking whether l and l' are equal takes a mere $O(\log m)$ bit operations.

If $l = l'$, we next compute $a^l \bmod m$ and $(a')^l \bmod m$, taking $O(\log^{2+o(1)} m)$ bit operations. Then, for each $p \in \mathfrak{Y}$, we compute $\mathrm{ord}_{p^{v_p(m)}}(a^l)$ and $\mathrm{ord}_{p^{v_p(m)}}((a')^l)$, which can be done for all p together with just two mord queries, and check if they are equal, which takes $O(\log m)$ bit operations for all p together. Finally, if necessary, we compute $\mathrm{ord}_4(a^l)$ and $\mathrm{ord}_4((a')^l)$ and check if they are equal, which can be done with $O(\log^{1+o(1)} m)$ bit operations (no queries necessary). ∎

With regard to our application to generalized cyclotomic mappings of degree 1, we note the following consequence of Corollary 5.3.1.2.

Corollary 5.3.1.3. *Let f and f' be monomial mappings of $\mathbb{F}_q$, given in polynomial form. Within q-bounded query complexity*

$$(\log^{2+o(1)} q, 1, \log q, 1, 0),$$

or q-bounded Las Vegas dual complexity

$$(\log^{8+o(1)} q, \log^{4+o(1)} q, \log^2 q)$$

one can decide whether $\Gamma_f \cong \Gamma_{f'}$.

Proof. As explained at the beginning of this subsection, one first deals with the case where at least one of f or f' is constant through simple scans of the input, taking $O(\log q)$ bit operations. If not, then

$$f(x) = ax^r \quad \text{and} \quad f'(x) = a'x^{r'},$$

where a, a', r, r' all are non-zero. Under logarithmization, the restriction of f, respectively, f', to $\mathbb{F}_q^*$ corresponds to the affine map

$$A(x) = rx + \log_\omega(a),$$

respectively, $A'(x) = r'x + \log_\omega(a')$, of $\mathbb{Z}/(q-1)\mathbb{Z}$, and computing these affine maps takes q-bounded query complexity $(\log q, 1, 0, 0, 0)$. Finally, one applies the algorithm from the proof of Corollary 5.3.1.2 to check within q-bounded query complexity $(\log^{2+o(1)} q, 0, \log q, 1, 0)$ whether $\Gamma_A \cong \Gamma_{A'}$, which is equivalent to

$$\Gamma_f \cong \Gamma_{f'}.$$

■

Comparing our Theorem 5.3.1.1 with Deng's original version [21, Theorem 11], we note that the additional simplification of essentially only having to check the equality

$$\mathrm{ord}_t(a^l) = \mathrm{ord}_t((a')^l)$$

for those divisors t of m that are of the form $p^{v_p(m)}$, as opposed to all divisors of m, is essential to achieve a polynomial complexity in the associated algorithm. This is because the number $\tau(m)$ of distinct (positive) divisors of m may be superpolynomial in $\log m$.

On the other hand, Dirichlet proved a result which implies that the average value of τ over the initial segment $\{1, 2, \ldots, N\}$ of $\mathbb{N}^+$ is asymptotically equivalent to $\log N$ [8, Theorem 3.3], so in particular, the set of positive integers m for which $\tau(m) \geq \log^{1+\varepsilon} m$ has asymptotic density 0 for each constant $\varepsilon > 0$, because otherwise,

if the said asymptotic density is $\delta > 0$, there are infinitely many $N \in \mathbb{N}^+$ such that

$$\begin{aligned} N^{-1} \sum_{m \le N} \tau(m) &\ge N^{-1} \sum_{m=1}^{\lfloor \delta N \rfloor} \log^{1+\varepsilon} m \\ &\ge (2N)^{-1} \frac{\delta}{2} N \log^{1+\varepsilon} \left(\frac{\delta}{2} N \right) = \frac{\delta}{4} \log^{1+\varepsilon} \left(\frac{\delta}{2} N \right) \\ &\ge \frac{\delta}{8} \log^{1+\varepsilon} N \gg \log N, \end{aligned}$$

a contradiction. Concerning the average value of $\tau(q-1)$, where q ranges over prime powers, we have the following result, the "In particular" statement of which will be used in Section 5.3.2. In this context, the authors would like to thank Ofir Gorodetsky, who kindly pointed out Halberstam's crucial paper [27] and parts of the proof of Proposition 5.3.1.4 in an answer to a question posted by the first author on MathOverflow[2].

Proposition 5.3.1.4. *For each $\varepsilon > 0$, there are constants $c_\varepsilon, c'_\varepsilon > 0$ such that the following hold for all but an asymptotic fraction of less than ε of all prime powers q:*

(1) $\mathrm{mpe}(q-1) < c_\varepsilon$; *and*

(2) *the number of distinct prime divisors of $q-1$ is less than $c'_\varepsilon \log\log q$.*

In particular, for all such prime powers q, one has $\tau(q-1) < \log^{c''_\varepsilon} q$, where

$$c''_\varepsilon := \log(c_\varepsilon + 1) c'_\varepsilon.$$

Proof. Throughout this proof, the variable q ranges over prime powers, while p ranges over primes. Statement (1) is the same as Proposition 5.1.10 (1). For statement (2), as usual, we denote by $\omega(m)$ the number of distinct prime divisors of $m \in \mathbb{N}^+$. Halberstam proved that

$$\Big(\sum_{p \le x} 1 \Big)^{-1} \sum_{p \le x} \omega(p-1) \sim \log\log x \tag{5.6}$$

as $x \to \infty$, see [27, Theorem 1]. Now, let $\varepsilon > 0$ be fixed, and let us assume that for some constant $c > 0$, one has $\omega(q-1) \ge c \log\log q$ for an asymptotic fraction of at least ε of all prime powers q. We need to bound c in terms of ε in order to prove statement (2). Now, because proper prime powers are a density 0 subset of all prime powers (see the proof of Proposition 5.1.10), we conclude that also for an asymptotic fraction of at least ε of all *primes* p, one has $\omega(p-1) \ge c \log\log p$. We fix a large

[2] see https://mathoverflow.net/questions/436134/average-value-of-the-prime-omega-function-omega-on-predecessors-of-prime-powe, visited on 2 September 2025.

enough $x \geq 2$ such that for a fraction of at least $\varepsilon/2$ of all primes $p \leq x$, one has $\omega(p-1) \geq c \log\log p$. Let us denote by p_m the m-th prime number for $m \in \mathbb{N}^+$. Because $p_m \geq (m \log m)/2$ for large enough m, and the total number of primes $p \leq x$ is at least $x/(2\log x)$, it follows that

$$\sum_{p \leq x} \omega(p-1) \geq \sum_{m=1}^{\lfloor \varepsilon x/(4\log x)\rfloor} c \log\log p_m \geq \sum_{m=\lceil \varepsilon x/(8\log x)\rceil}^{\lfloor \varepsilon x/(4\log x)\rfloor} c \log\log\left(\frac{1}{2} m \log m\right)$$
$$\geq \frac{\varepsilon x}{16 \log x} \cdot c \log\log\left(\frac{1}{2} \cdot \frac{\varepsilon x}{8\log x} \cdot \log\left(\frac{\varepsilon x}{8\log x}\right)\right),$$

whence

$$\left(\sum_{p \leq x} 1\right)^{-1} \sum_{p \leq x} \omega(p-1)$$
$$\geq \left(2\frac{x}{\log x}\right)^{-1} \cdot \frac{\varepsilon x}{16 \log x} \cdot c \log\log\left(\frac{1}{2} \cdot \frac{\varepsilon x}{8\log x} \cdot \log\left(\frac{\varepsilon x}{8\log x}\right)\right)$$
$$\sim \frac{\varepsilon c}{32} \log\log x,$$

which implies

$$\left(\sum_{p \leq x} 1\right)^{-1} \sum_{p \leq x} \omega(p-1) \geq \frac{\varepsilon c}{64} \log\log x$$

if x is large enough. Hence, in order to not contradict Halberstam's (5.6), we must have $c \leq 64/\varepsilon$, an upper bound on c in terms of ε, as required in order for statement (2) to hold.

Finally, for the "In particular" statement, we note that because

$$\tau(q-1) = (v_1+1)(v_2+1)\cdots(v_K+1)$$

if $q - 1 = p_1^{v_1} p_2^{v_2} \cdots p_K^{v_K}$ is the prime factorization of $q-1$, one can bound $\tau(q-1)$ from above as follows:

$$\tau(q-1) \leq (\mathrm{mpe}(q-1)+1)^{\omega(q-1)} \leq (c_\varepsilon + 1)^{c'_\varepsilon \log\log q}$$
$$= \exp(c'_\varepsilon \log\log q \log(c_\varepsilon + 1))$$
$$= (\log q)^{\log(c_\varepsilon+1)c'_\varepsilon}.$$

■

In addition to thanking Ofir Gorodetsky, the authors would also like to thank one of the anonymous reviewers, who pointed out that it is even possible to bound the average value of $\tau(q-1)$ for prime powers $q \leq x$ using the Titchmarsh divisor theorem, which states that for each constant $a \in \mathbb{Z}$, one has

$$\sum_{p \leq x} \tau(p+a) = C_1(a) + O\left(\frac{x \log\log x}{\log x}\right),$$

see, e.g., Halberstam's short proof [28] (using the Bombieri–Vinogradov theorem and the Brun–Titchmarsh inequality) or the earlier proof of Linnik [46] using the dispersion method. Indeed, using that $\sum_{p\le x} 1 \sim \frac{x}{\log x}$, this implies that

$$\Big(\sum_{p\le x} 1\Big)^{-1} \sum_{p\le x} \tau(p-1) \in O(\log x),$$

and since the number of proper prime powers up to x is in $O(\sqrt{x}\log x)$ and τ grows asymptotically more slowly than any power function, the same O-bound holds true when primes p are replaced by prime powers q. In particular, we get the following stronger version of the "In particular" of Proposition 5.3.1.4.

Proposition 5.3.1.5. *Let $\mathfrak{g} : [0,\infty) \to \mathbb{R}$ be monotonically increasing and such that $\mathfrak{g}(x) \to \infty$ as $x \to \infty$. Then, as $x \to \infty$, the proportion of prime powers $q \le x$ such that $\tau(q-1) \ge \mathfrak{g}(q-1)$ tends to 0.*

Proof. Assume otherwise. Then there is an $\varepsilon > 0$ such that for arbitrarily large $x \in [0,\infty)$, the proportion of prime powers $q \le x$ with $\tau(q-1) \ge \mathfrak{g}(x)$ is at least ε. Since at least $\frac{\varepsilon}{2}x$ of those prime powers must be greater than or equal to $\frac{\varepsilon}{2}x$, it follows that there is are arbitrarily large positive real numbers x such that

$$\Big(\sum_{q\le x} 1\Big)^{-1} \sum_{q\le x} \tau(q-1) \ge \frac{\mathfrak{g}\big(\frac{\varepsilon}{2}x\big)\cdot\frac{\varepsilon}{2}x}{2x/\log x} = \frac{\varepsilon}{2}\log x \cdot \mathfrak{g}\Big(\frac{\varepsilon}{2}x\Big),$$

contradicting that the average value of $\tau(q-1)$ is in $O(\log x)$. ∎

We conclude this subsection with two batches of pseudocode. First, we give pseudocode for the algorithm that checks whether $\Gamma_A \cong \Gamma_{A'}$, where $A : x \mapsto ax+b$ and $A' : x \mapsto a'x+b'$ are affine maps of $\mathbb{Z}/m\mathbb{Z}$ (see Corollary 5.3.1.2 and its proof). As in the previous section, we specify the (m-bounded) query complexity (QC) of each step.

1 Compute $\gcd(a,m)$ and $\gcd(a',m)$, and check whether they are equal. If not, output "false" and halt.
QC: $(\log^{1+o(1)} m, 0, 0, 0, 0)$.

2 Factor $m = p_1^{v_1}\cdots p_K^{v_k}$.
QC: $(\log m, 0, 1, 0, 0)$.

3 For each $j = 1, 2, \ldots, K$, do the following.
QC: $(\log^{2+o(1)} m, 0, 0, 0, 0)$.

3.1 Check whether it is the case that $p_j \nmid a$ and $\gcd(a-1, p_j^{v_j}) \mid b$. If so, set $\mathrm{test}_j :=$ true, otherwise set $\mathrm{test}_j :=$ false.
QC: $(\log^{1+o(1)} m, 0, 0, 0, 0)$.

3.2 Check whether it is the case that $p_j \nmid a'$ and $\gcd(a'-1, p_j^{v_u}) \mid b'$. If so, set $\text{test}'_j :=$ true, otherwise set $\text{test}'_j :=$ false.
QC: $(\log^{1+o(1)} m, 0, 0, 0, 0)$.

3.3 If $\text{test}_j \neq \text{test}'_j$, then output "false" and halt.
QC: $(\log \log m, 0, 0, 0, 0)$.

4 For each $j = 1, 2, \ldots, K$, do the following.
QC: $(\log^{2+o(1)} m, 0, \log m, 0, 0)$.

4.1 Check whether $p_j \nmid a$. If not, set $\text{Test}_j :=$ false and skip to step 4.4. Otherwise, set $\text{Test}_j :=$ true.
QC: $(\log^{1+o(1)} m, 0, 0, 0, 0)$.

4.2 Check which case in Table 2.2 (or [15, Table 3 or 4, respectively]) applies to $A \bmod p_j^{v_j}$, using simple arithmetic.
QC: $(\log^{1+o(1)} m, 0, 0, 0, 0)$.

4.3 Determine the minimal cycle length l_j of $A \bmod p_j^{v_j}$ according to Table 2.2 (or [15, Table 3 or 4, respectively]) and factor it.
QC: $(\log^{1+o(1)} m, 0, 1, 0, 0)$.

4.4 Check whether $p_j \nmid a'$. If not, set $\text{Test}'_j :=$ false and skip to the next j. Otherwise, set $\text{Test}'_j :=$ true.
QC: $(\log^{1+o(1)} m, 0, 0, 0, 0)$.

4.5 Check which case in Table 2.2 (or [15, Table 3 or 4, respectively]) applies to $A' \bmod p_j^{v_j}$, using simple arithmetic.
QC: $(\log^{1+o(1)} m, 0, 0, 0, 0)$.

4.6 Determine the minimal cycle length l'_j of $A' \bmod p_j^{v_j}$ according to Table 2.2 (or [15, Table 3 or 4, respectively]) and factor it.
QC: $(\log^{1+o(1)} m, 0, 1, 0, 0)$.

5 Compute

$$l := \operatorname{lcm}(l_j : 1 \leq j \leq K, \text{Test}_j = \text{true})$$

and

$$l' := \operatorname{lcm}(l'_j : 1 \leq j \leq K, \text{Test}'_j = \text{true})$$

using the factorizations of the l_j and l'_j computed in steps 4.3 and 4.6 above.
QC: $(\log^{2+o(1)} m, 0, 0, 0, 0)$.

6 Check whether $l = l'$. If not, output "false" and halt.
QC: $(\log m, 0, 0, 0, 0)$.

7 Compute $a^l \bmod m$ and $(a')^l \bmod m$.
QC: $(\log^{2+o(1)} m, 0, 0, 0, 0)$.

8 Compute $\mathrm{ord}_{p^{v_p(m)}}(a^l)$, respectively, $\mathrm{ord}_{p^{v_p(m)}}((a')^l)$, for all primes $p \mid m$ such that $p \nmid a$, respectively, $p \nmid a'$, in particular for all

$$p \in \mathfrak{Y} = \mathfrak{Y}' = \{p_j : \mathrm{test}_j = \mathrm{true}\}.$$

QC: $(\log m, 0, 0, 1, 0)$.

9 Check whether $\mathrm{ord}_{p^{v_p(m)}}(a^l) = \mathrm{ord}_{p^{v_p(m)}}((a')^l)$ for all $p \in \mathfrak{Y}$. If not, output "false" and halt.
QC: $(\log m, 0, 0, 0, 0)$.

10 If $4 \mid m$, do the following.

10.1 Check whether $\mathrm{ord}_4(a^l) = \mathrm{ord}_4((a')^l)$. If not, output "false" and halt.
QC: $(\log^{1+o(1)} m, 0, 0, 0, 0)$.

10.2 Output "true" and halt.
QC: $(1, 0, 0, 0, 0)$.

11 Else do the following.

11.1 Output "true" and halt.
QC: $(1, 0, 0, 0, 0)$.

Finally, we provide pseudocode for the algorithm that checks whether $\Gamma_f \cong \Gamma_{f'}$ for monomial mappings $f : x \mapsto ax^r$ and $f' : x \mapsto a'x^{r'}$ of $\mathbb{F}_q$ (where $a, a' \in \mathbb{F}_q$ and $r, r' \in \{0, 1, \ldots, q-1\}$).

1 If $a = 0_{\mathbb{F}_q}$ or $r = 0$, then do the following.

1.1 Check whether it is the case that $a' = 0_{\mathbb{F}_q}$ or $r' = 0$. If so, output "true" and halt. Otherwise, output "false" and halt.
QC: $(\log q, 0, 0, 0, 0)$.

2 Else do the following.

2.1 Check whether it is the case that $a' = 0_{\mathbb{F}_q}$ or $r' = 0$. If so, output "false" and halt.
QC: $(\log q, 0, 0, 0, 0)$.

2.2 Compute $b := \log_\omega(a)$ and $b' := \log_\omega(a')$, where ω is the "natural" choice of primitive element of $\mathbb{F}_q$ (see the proof of Lemma 5.1.6 (4)). Denote by A, respectively, A', the affine map of $\mathbb{Z}/(q-1)\mathbb{Z}$ given by the formula $A(x) = rx + b$, respectively, $A'(x) = r'x + b'$.
QC: $(\log q, 1, 0, 0, 0)$.

2.3 Use the algorithm from above to check whether

$$\Gamma_A \cong \Gamma_{A'}.$$

If so, output "true" and halt. Otherwise, output "false" and halt.
QC: $(\log^{2+o(1)} q, 0, \log q, 1, 0)$.

5.3.2 Special case: Trees only depend on the coset

In this subsection, we discuss two special classes of generalized cyclotomic mappings of $\mathbb{F}_q$ such that if f_1 and f_2 each belong to one of those classes (not necessarily both to the same) and are of index d_1 and d_2, respectively, then it can be decided whether $\Gamma_{f_1} \cong \Gamma_{f_2}$ in a q-bounded query complexity each entry of which is polynomial in the parameters $\max\{d_1, d_2\}$, $\log q$ and $\tau(q-1)$ (the number of divisors of $q-1$). In particular, the q-bounded Las Vegas dual complexity of this problem is always subexponential in the input size $O(\max\{d_1, d_2\} \log q)$, as $\tau(m) \in o(m^\varepsilon)$ for each $\varepsilon > 0$ [8, p. 296]. Moreover, in view of Proposition 5.3.1.4 or 5.3.1.5 with $\mathfrak{g}(x) = \log x$, for instance, for each $\varepsilon > 0$, one has that for all but an asymptotic fraction of less than ε of all finite fields $\mathbb{F}_q$, the said q-bounded Las Vegas dual complexity is polynomial in the input size (and the polynomial degree does, by Proposition 5.3.1.5, *not* depend on ε).

The two classes of generalized cyclotomic mappings f of $\mathbb{F}_q$, say of index d, which we consider are as follows.

- *Class 1*: f maps each coset C_i for $i \in \{0, 1, \ldots, d-1\}$ either to $C_d = \{0_{\mathbb{F}_q}\}$ or bijectively to $C_{\bar{f}(i)}$ (i.e., whenever the affine function A_i of $\mathbb{Z}/s\mathbb{Z}$ is well defined, it is a permutation of $\mathbb{Z}/s\mathbb{Z}$). If this happens, we say that f is *of special type* I. This is the same situation as in Section 4.3.
- *Class 2*: The induced function $\bar{f}$ is a permutation of $\{0, 1, \ldots, d\}$ (i.e., f permutes the cosets of C). If this happens, we say that f is *of special type* II. Using the notation of Section 3.2, this means that $\Gamma_f = \Gamma_{\mathrm{per}}$, and so the discussion from that section applies.

The crucial property which these two cases share is that the rooted trees above periodic vertices in Γ_f only depend on the block C_i, for $i \in \{0, 1, \ldots, d\}$, in which these vertices lie, as follows from Lemma 4.3.1 and Theorem 3.2.1 (or Proposition 2.1.8), respectively. This allows us to produce a comparatively compact description of the digraph isomorphism type of Γ_f. To that end, it is helpful to adapt the notion of a partition-tree register, introduced in Definition 5.1.2, as follows.

Definition 5.3.2.1. Let f be an index d generalized cyclotomic mapping of $\mathbb{F}_q$.

(1) We assume that f is of special type I, so that *all* vertices (not just f-periodic ones) in a given block C_i have isomorphic rooted trees above them in Γ_f. A *type*-I *tree register for* f is an ordered sequence $((\mathfrak{D}_n, S_n))_{n=0,1,\ldots,N}$ such that

   (a) the sets $\mathfrak{D}_n$ form a recursive tree description list, with associated rooted tree isomorphism types $\mathfrak{T}_n$ (see Definition 5.1.1) and

   (b) for each n, the set S_n is nonempty and consists precisely of those $i \in \{0, 1, \ldots, d\}$ such that $\mathrm{Tree}_{\Gamma_f}(x) \cong \mathrm{Tree}_{\Gamma_{\bar{f}}}(i) \cong \mathfrak{T}_n$ for any $x \in C_i$.

(2) We assume that f is of special type II, so that for each $i \in \{0, 1, \ldots, d-1\}$, the rooted tree isomorphism type $\mathrm{Tree}_{\Gamma_f}(x)$ for $x \in C_i$ only depends on i and the $\mathfrak{h}$-value of x (see Section 4.1, page 66 onward). A *type*-II *tree register for* f is an ordered sequence $((\mathfrak{D}_n, S_n))_{n=0,1,\ldots,N}$ such that

(a) the sets $\mathfrak{D}_n$ form a recursive tree description list, with associated rooted tree isomorphism types $\mathfrak{T}_n$;

(b) the $\mathfrak{T}_n$ are just those isomorphism types that occur among the rooted trees of the form $\mathrm{Expand}(\mathfrak{T}_{i,h})$ for $i \in \{0, 1, \ldots, d-1\}$ and

$$h \in \{0, 1, \ldots, H_i\}$$

(see Section 4.1, page 66 onward, for the definition of the $\mathfrak{T}_{i,h}$); and

(c) for each n, one has $S_n = (\mathrm{ht}(\mathfrak{T}_n), S_{n,\mathrm{trans}}, S_{n,\mathrm{per}})$, where

$$S_{n,\mathrm{trans}} = \{i \in \{0, 1, \ldots, d-1\} : \mathrm{ht}(\mathfrak{T}_n) < H_i \text{ and } \mathfrak{T}_n = \mathrm{Expand}(\mathfrak{T}_{i,\mathrm{ht}(\mathfrak{T}_n)})\}$$

and

$$S_{n,\mathrm{per}} = \{i \in \{0, 1, \ldots, d-1\} : \mathfrak{T}_n = \mathrm{Expand}(\mathfrak{T}_{i,H_i})\}.$$

In an implementation, we assume that the sets S_n from Definition 5.3.2.1 (1), as well as the sets $S_{n,\mathrm{trans}}$ and $S_{n,\mathrm{per}}$ from Definition 5.3.2.1 (2), are represented by *sorted* arrays each entry of which is a binary digit representation of a number $i \in \{0, 1, \ldots, d\}$ with bit length exactly $\lfloor \log_2 d \rfloor + 1$. Moreover, in a type-I tree register, only one of the descriptions $\mathfrak{D}_n$ corresponds to $\mathrm{Tree}_{\Gamma_f}(0_{\mathbb{F}_q})$, and it is the only description in which the second entries of elements may be larger than d. For the sake of efficiency, we make the convention that all descriptions $\mathfrak{D}_n$ except that one use $\lfloor \log_2 d \rfloor + 1$ digits for representing each entry m or k_m of an element (m, k_m) of $\mathfrak{D}_n$. On the other hand, in the description corresponding to $\mathrm{Tree}_{\Gamma_f}(0_{\mathbb{F}_q})$, we use $\lfloor \log_2 d \rfloor + 1$ digits for the first entry m, and $\lfloor \log_2 q \rfloor + 1$ digits for the second entry k_m. In contrast to this, in a type-II tree register, we know that $m \leq d^2 \,\mathrm{mpe}(q-1) + d \leq d^2 \lfloor \log_2 q \rfloor + d$ (see the first paragraph in the proof of Lemma 5.3.2.2 (4) below), while there is a priori no upper bound on k_m other than the trivial one, q. Hence, in such a register, we use $\lfloor \log_2(d^2 \lfloor \log_2 q \rfloor + d) \rfloor + 1$ digits for representing m, and $\lfloor \log_2 q \rfloor + 1$ digits for representing k_m. In either case, the entries of a given description $\mathfrak{D}_n$ all have the same bit length, and different descriptions $\mathfrak{D}_n$ use the same bit length for the first entries of their elements. As before, we assume that the array representing a given description $\mathfrak{D}_n$ is lexicographically ordered (corresponding to the ordering of the elements (m, k_m) of $\mathfrak{D}_n$ by increasing m).

We observe that in Definition 5.3.2.1 (2,c), the set $S_{n,\mathrm{trans}}$ consists precisely of those $i \in \{0, 1, \ldots, d-1\}$ such that $\mathfrak{T}_n = \mathrm{Tree}_{\Gamma_f}(x)$ for all $x \in C_i$ with $\mathfrak{h}(x) = \mathrm{ht}(\mathfrak{T}_n) < H_i$, regardless of whether such x exist; we recall from Example 3.2.2 that

not necessarily all values in $\{0, 1, \ldots, H_i - 1\}$ are assumed by $\mathfrak{h}$ on the f-transient points in a given coset C_i. We also remind the reader that f-transient $x \in C_i$ are characterized by the inequality $\mathfrak{h}(x) < H_i$, and that for all such x, one has

$$\mathrm{ht}(\mathrm{Tree}_{\Gamma_f}(x)) = \mathrm{ht}(\mathfrak{T}_{i,\mathfrak{h}(x)}) = \mathfrak{h}(x);$$

see the recursive definition of the $\mathfrak{T}_{i,h}$ in Section 4.1, page 66 onward. Moreover, the set $S_{n,\mathrm{per}}$ consists of those $i \in \{0, 1, \ldots, d-1\}$ such that $\mathfrak{T}_n$ is the unique isomorphism type $\mathrm{Tree}_{\Gamma_f}(x) = \mathrm{Expand}(\mathfrak{T}_{i,H_i})$ for f-periodic $x \in C_i$ (and $\mathrm{Expand}(\mathfrak{T}_{i,H_i})$ is *not* necessarily of height H_i, but it is of height $\mathcal{H}_i$).

Next, we discuss the following important lemma.

Lemma 5.3.2.2. *Let f be an index d generalized cyclotomic mapping of $\mathbb{F}_q$, given in cyclotomic form (1.1).*

(1) *Checking whether f is of special type* I *takes q-bounded query complexity*

$$(d \log^{1+o(1)} q, d, 0, 0, 0).$$

(2) *If f is of special type* I*, then a type-*I *tree register $\mathfrak{R} = ((\mathfrak{D}_n, S_n))_{n=0,1,\ldots,N}$ for f with $N \in O(d)$ can be computed within q-bounded query complexity*

$$(d^3 \log^2 d + d \log^{1+o(1)} q, d, 0, 0, 0).$$

(3) *Checking whether f is of special type* II *has q-bounded query complexity*

$$(d \log^2 d + d \log^{1+o(1)} q, d, 0, 0, 0).$$

(4) *If f is of special type* II*, then a type-*II *tree register $\mathfrak{R} = ((\mathfrak{D}_n, S_n))_{n=0,1,\ldots,N}$ for f with $N \in O(d^2 \,\mathrm{mpe}(q-1))$ can be computed within q-bounded query complexity*

$$(d^5 \log^2 d \,\mathrm{mpe}(q-1)^3 + d^5 \,\mathrm{mpe}(q-1)^3 \log q + d^2 \,\mathrm{mpe}(q-1) \log^{1+o(1)} q, d, 0, 0, 0).$$

In particular, it can be computed within q-bounded query complexity

$$(d^{5+o(1)} \log^4 q, d, 0, 0, 0).$$

Proof. For statement (1), we first compute $\bar{f}$ and the affine maps $A_i : x \mapsto \alpha_i x + \beta_i$, which requires q-bounded query complexity $(d \log^{1+o(1)} q, d, 0, 0, 0)$ by Proposition 5.1.8. We note that f is of special type I if and only if $\gcd(\alpha_i, s) = 1$ for all i such that A_i is well defined (i.e., such that the coefficient $a_i \in \mathbb{F}_q$ in the cyclotomic form (1.1) of f is non-zero), which one can check with $O(d \log^{1+o(1)} q)$ bit operations by Lemma 5.1.5 (3,8).

For statement (2), we start by computing $\bar{f}$, the number $s = (q-1)/d$ and (as in Section 5.2.1, page 116) the "layers"

$$\mathrm{Layer}_h := \begin{cases} \mathrm{im}(\bar{f}^h) \setminus \mathrm{im}(\bar{f}^{h+1}), & \text{if } h \in \{0, 1, \ldots, \bar{H}-1\}, \\ \mathrm{im}(\bar{f}^{\bar{H}}) = \mathrm{per}(\bar{f}), & \text{if } h = \infty, \end{cases}$$

of $\bar{f}$ with respect to iteration in sorted form and without multiple entries, where $\bar{H}$ is the maximum tree height in $\Gamma_{\bar{f}}$. Altogether, this takes q-bounded query complexity $(d \log^{1+o(1)} q + d \log^2 d, d, 0, 0, 0)$, and we may also compute the cycles of $\bar{f}$ in the process. After this, we start building the register. At any given point during that process, we have at least a "partial register" as an intermediate result, which includes definitions of descriptions $\mathfrak{D}_n$ of rooted trees $\mathfrak{T}_n$ for all $n \in \mathcal{N}$, an initial segment of $\mathbb{N}_0$. Because each $n \in \mathcal{N}$ has a non-empty subset S_n of $\{0, 1, \ldots, d\}$ associated with it and those sets are pairwise disjoint, we conclude that $|\mathcal{N}| \leq d + 1 \in O(d)$. In particular, $N \in O(d)$ in the end, as asserted.

Now, to build the register, we do the following: first, we compute the pre-image set $\bar{f}^{-1}(\{i\})$ in sorted form, which just requires looping over the known value table of $\bar{f}$ once, thus taking $O(d \log d)$ bit operations only. Now, for a fixed index $i \in \mathrm{Layer}_h$, let $j_1, j_2, \ldots, j_K$ be the $\bar{f}$-*transient* pre-images of i under $\bar{f}$; if $h < \infty$, those are simply all pre-images of i, and if $h = \infty$, one can determine them by additionally identifying the unique $\bar{f}$-pre-image of i in $\mathrm{per}(\bar{f}) = \mathrm{Layer}_\infty$; since the cycles of $\bar{f}$ are known and one can store the Boolean information which indices lie in Layer_∞ by scanning once over each of the other layers beforehand using $O(d \log d)$ bit operations, this only takes $O(\log d)$ bit operations per i, hence $O(d \log d)$ bit operations overall.

In what follows, we assume that i is fixed. Each j_t lies in a unique layer $\mathrm{Layer}_{h_{j_t}}$ with $h_{j_t} < h_i = h$, and so there is a unique non-negative integer $\bar{n}_{j_t} \in \mathcal{N}$ such that $j_t \in S_{\bar{n}_{j_t}}$. Computing $\bar{n}_{j_t}$ takes $O(|\mathcal{N}| \cdot \log^2 d) \subseteq O(d \log^2 d)$ bit operations for a single t, hence $O(d^2 \log^2 d)$ bit operations altogether (for this fixed value of i). Now, by Lemma 4.3.1, the rooted tree above any $x \in C_i$ is isomorphic to

$$\begin{cases} \mathrm{Tree}_{\Gamma_{\bar{f}}}(i) \cong \sum_{t=1}^{K} \mathfrak{T}^+_{\bar{n}_{j_t}}, & \text{if } i < d, \\ \sum_{t=1}^{K} s\mathfrak{T}^+_{\bar{n}_{j_t}}, & \text{if } i = d, \end{cases}$$

and so we may choose the following compact description $\mathfrak{D}$ for this tree.

- If $i < d$, we set

$$\mathfrak{D} := \{(n, m) : n \in \{\bar{n}_{j_t} : 1 \leq t \leq K\}, m = |\{t \in \{1, \ldots, K\} : \bar{n}_{j_t} = n\}| > 0\}.$$

- If $i = d$, we set

$$\mathfrak{D} := \{(n, sm) : n \in \{\bar{n}_{j_t} : 1 \leq t \leq K\}, m = |\{t \in \{1, \ldots, K\} : \bar{n}_{j_t} = n\}| > 0\}.$$

Computing $\mathfrak{D}$ after the numbers $\bar{n}_{j_t}$ have been determined requires us to create a list of the *distinct* values of the $\bar{n}_{j_t}$ and their multiplicities, which can be done in $O(K \log K \log d) \subseteq O(d \log^2 d)$ bit operations when using the sorting algorithm from Lemma 5.1.5 (10). If $i < d$, this is also the overall complexity of computing $\mathfrak{D}$ for that i, whereas if $i = d$, the complexity of computing $\mathfrak{D}$ is in $O(d \log^2 d + d \log^{1+o(1)} q)$, since it also involves integer multiplications. After $\mathfrak{D}$ has been computed, we check whether there is an $n \in \mathcal{N}$ such that $\mathfrak{D} = \mathfrak{D}_n$, which takes

$$O(|\mathcal{N}| \cdot d \log d) \subseteq O(d^2 \log d)$$

bit operations regardless of the value of i. Indeed, if $i = d$, for which the bit length of the second entries of elements of $\mathfrak{D}$ is not necessarily in $O(\log d)$, one can proceed as follows. Observing that those second entries can only be that large for this one value of i, one first checks whether $s > d$, which can be done with a mere $O(\log d)$ bit operations (we note that s itself was already computed at the beginning). If so, one knows that $\mathfrak{D} \neq \mathfrak{D}_n$ for any $n \in \mathcal{N}$; otherwise, the bit length of the second entries of $\mathfrak{D}$ is in $O(\log d^2) = O(\log d)$ even for $i = d$, and one can proceed as for $i < d$. In any case, if $\mathfrak{D} = \mathfrak{D}_n$ for some $n \in \mathcal{N}$, then we add i to S_n as a new element by merging the sorted lists corresponding to the sets S_n and $\{i\}$, which takes $O(d \log d)$ bit operations by Lemma 5.1.5 (11). Otherwise, we create $\mathfrak{D}$ as a new description $\mathfrak{D}_{n'}$, where $n' = \max \mathcal{N} + 1$, and initialize $S_{n'} := \{i\}$.

Since we need to carry out the computations described after declaring i to be fixed for all such i, the total bit operation cost of these computations is in $O(d^3 \log^2 d + d \log^{1+o(1)} q)$, and so the overall q-bounded query complexity of computing a type-I tree register for f is as asserted.

For statement (3), we simply compute $\bar{f}$ and check whether $\mathrm{im}(\bar{f}) = \{0, 1, \ldots, d\}$. The former of these two tasks takes q-bounded query complexity

$$(d \log^{1+o(1)} q, d, 0, 0, 0)$$

by Proposition 5.1.8, and the latter takes $O(d \log^2 d)$ bit operations (see the beginning of the argument in Section 5.2.1).

For statement (4), we first compute $\bar{f}$, the affine maps $A_i : x \mapsto \alpha_i x + \beta_i$ and (as in Section 5.2.1) a CRL-list $\bar{\mathcal{L}}$ of $\bar{f}$ and the cycles of $\bar{f}$, taking q-bounded query complexity in $(d \log^{1+o(1)} q + d \log^2 d, d, 0, 0, 0)$. Following that, we start building the tree register, and as in the proof of statement (2), in dependency of a given point in that process, we denote by $\mathcal{N}$ the initial segment of $\mathbb{N}_0$ consisting of all n for which $\mathfrak{D}_n$ is defined at that point. Because the associated rooted tree isomorphism types $\mathfrak{T}_n$ are pairwise distinct and are elements of the set $\{\mathrm{Expand}(\mathfrak{T}_{i,h}) : i \in \{0, 1, \ldots, d - 1\}, h \in \{0, 1, \ldots, H_i\}\}$, we have

$$|\mathcal{N}| \leq d \cdot (\max_i H_i + 1) \leq d^2 \,\mathrm{mpe}(q - 1) + d \in O(d^2 \,\mathrm{mpe}(q - 1))$$

(for the bound on H_i, see formula (3.2) in Section 3.3). In particular,

$$N \in O(d^2 \operatorname{mpe}(q-1))$$

in the end, as asserted.

To build the register, we go through the elements $(i,\ell) \in \bar{\mathcal{L}}$ with $i < d$ (we note that $\operatorname{Tree}_{\Gamma_f}(0_{\mathbb{F}_q})$ is simply trivial and is not even recorded in the register by definition), and for each of them, we do the following. First, we compute the exact value of H_i, the maximum tree height above a periodic vertex in $\bigcup_{t=0}^{\ell-1} C_{i_t}$, where $(i_0, i_1, \ldots, i_{\ell-1})$ with $i = i_0$ is the $\bar{f}$-cycle of i. We note that by Theorem 3.2.1 and the paragraph before it (which was worked out in detail in Section 4.1), we have the following. For each $t \in \{0,1,\ldots,\ell-1\}$, the trees above periodic vertices in C_{i_t} are pairwise isomorphic and thus of a common height $\mathcal{H}_{i_t}$. We have $H_i = \max\{\mathcal{H}_{i_t} : t = 0,1,\ldots,\ell-1\}$, so we compute the numbers $\mathcal{H}_{i_t}$ in order to get H_i. At this point, we note that in fact, in Section 5.2.2, we already described how to find H_i through a binary search. However, here we are also interested in storing the procreation numbers $\operatorname{proc}_{i_t,k}$ for $t \in \{0,1,\ldots,\ell-1\}$ and $k \in \{1,2,\ldots,\mathcal{H}_{i_t}\}$, whence we do *not* use binary search to skip steps.

We remind the reader that for arbitrary $t \in \mathbb{Z}$, the notation i_t is shorthand for $i_{t \bmod \ell}$. For $t = 0,1,\ldots,\ell-1$ and successively for $k = 1,2,\ldots$, we compute (and store) the procreation number (see Theorem 3.2.1)

$$\operatorname{proc}_{i_t,k} = \frac{\gcd(\prod_{r=0}^{k-1} \alpha_{i_{t-k+r}}, s)}{\gcd(\prod_{r=0}^{k-2} \alpha_{i_{t-k+r}}, s)}$$

until $\operatorname{proc}_{i_t,k} = 1$ for the first time for a given t, which happens precisely when $k = \mathcal{H}_{i_t} + 1$. If we store the values of the two products appearing in the formula for $\operatorname{proc}_{i_t,k}$, then the computation of each $\operatorname{proc}_{i_t,k}$ only involves $O(1)$ multiplications and thus has a bit operation cost in $O(\log^{1+o(1)} q)$ by Lemma 5.1.5 (3,8). Therefore, and because $\mathcal{H}_{i_t} \in O(\ell \operatorname{mpe}(q-1))$, we can compute each individual $\mathcal{H}_{i_t}$ using $O(\ell \operatorname{mpe}(q-1) \log^{1+o(1)} q)$ bit operations, whence the computation of H_i in total takes $O(\ell^2 \operatorname{mpe}(q-1) \log^{1+o(1)} q)$ bit operations.

Once H_i has been computed, we start adding the information associated with the $\bar{f}$-cycle of i to our tree register. More precisely, we do the following successively for $h = 0,1,\ldots,H_i$. If $h = 0$, then $\operatorname{Expand}(\mathfrak{T}_{i_t,h})$ is trivial for all t, so in case $\mathcal{N} = \emptyset$ (which only happens for the first pair $(i,\ell) \in \bar{\mathcal{L}}$ we consider), we set $\mathfrak{D}_0 := \emptyset$, causing $\mathfrak{T}_0$ to be the trivial rooted tree (in particular, $\operatorname{ht}(\mathfrak{T}_0) = 0$), and we initialize some variables as follows.

- We set

$$S_{0,\text{trans}} := \begin{cases} \{i_0, i_1, \ldots, i_{\ell-1}\}, & \text{if } H_i > 0, \\ \emptyset, & \text{otherwise.} \end{cases}$$

- We set $S_{0,\mathrm{per}} := \{i_t : t \in \{0, 1, \ldots, \ell - 1\}, \mathcal{H}_{i_t} = 0\}$.

We remind the reader that we want the arrays representing $S_{0,\mathrm{trans}}$ and $S_{0,\mathrm{per}}$ to be sorted, so one should apply the sorting algorithm from Lemma 5.1.5 (10), which takes $O(d \log^2 d)$ bit operations for each array.

In the other case, where $\mathcal{N} \neq \emptyset$, we do the following.

- If $H_i > 0$, we add $i_0, i_1, \ldots, i_{\ell-1}$ to the already defined set $S_{0,\mathrm{trans}}$ as new elements (technically speaking, we sort $\{i_0, i_1, \ldots, i_{\ell-1}\}$ and merge it with $S_{0,\mathrm{trans}}$).
- We also add all indices i_t, for $t \in \{0, 1, \ldots, \ell - 1\}$, such that $\mathcal{H}_{i_t} = 0$ to the already defined set $S_{0,\mathrm{per}}$ as new elements.

Using Lemma 5.1.5 (10,11), one sees that dealing with the case $h = 0$ as a whole only takes $O(\ell \log^2 d + d \log d)$ bit operations (for copying information and sorting/merging, as well as simple look-ups of the values $\mathcal{H}_{i_t}$).

Now we assume that $h \geq 1$. The edge-weighted rooted tree (isomorphism type) $\mathfrak{T}_{i_t,h}$ is drawn at the end of Section 4.1 (we draw the reader's attention to the case distinction between $h < H_i$ and $h = H_i$), and we compute the description

$$\mathfrak{D} = \mathfrak{D}(i_t, h)$$

of $\mathrm{Expand}(\mathfrak{T}_{i_t,h})$ as follows. First, we set

$$h' := \begin{cases} h, & \text{if } h < H_i, \\ \mathcal{H}_{i_t}, & \text{if } h = H_i, \end{cases}$$

which is the height of $\mathrm{Expand}(\mathfrak{T}_{i_t,h})$. It is also the number of edges in $\mathfrak{T}_{i_t,h}$ that have the root as their terminal vertex (i.e., the unweighted in-degree of that root). We do note that some of these edges may have weight 0. For $k = 0, 1, \ldots, h' - 1$, we compute

$$\mathrm{w}_k := \begin{cases} \mathrm{w}_{i_t,k} = \mathrm{proc}_{i_t,k+1} - \mathrm{proc}_{i_t,k+2}, & \text{if } h = H_i, \text{ or } h < H_i \text{ and } k < h' - 1, \\ \mathrm{proc}_{i_t,h}, & \text{if } h < H_i \text{ and } k = h' - 1, \end{cases}$$

which is the weight of the $(k + 1)$-th edge in $\mathfrak{T}_{i_t,h}$ (counted from the left in the drawing) that has the root as its terminal vertex. These computations only require

$$O(\ell \cdot h' \cdot \log q) \subseteq O(\ell^2 \,\mathrm{mpe}(q - 1) \log q)$$

bit operations for all t together. We may then set

$$\mathfrak{D} := \{(\bar{n}_{i_t,k}, \mathrm{w}_k) : k = 0, 1, \ldots, h' - 1\},$$

where $\bar{n}_{i_t,k}$ is the unique $n \in \mathcal{N}$ such that

$$\mathfrak{T}_n = \mathrm{Expand}(\mathfrak{T}_{i_t,k}),$$

i.e., such that $i_t \in S_{n,\mathrm{trans}}$ and $\mathrm{ht}(\mathfrak{T}_n) = k$. Assuming that the heights of the various $\mathfrak{T}_n$ are stored whenever the register is updated, the computation of $\mathfrak{D}$ takes

$$\begin{aligned} &O(h' \cdot (|\mathcal{N}| \cdot \log^2 d + \log\log q)) \\ &\quad \subseteq O(\ell \operatorname{mpe}(q-1) \cdot (d^2 \operatorname{mpe}(q-1) \cdot \log^2 d + \log\log q)) \\ &\quad \subseteq O(\ell d^2 \log^2 d \operatorname{mpe}(q-1)^2 + \ell \operatorname{mpe}(q-1) \log\log q) \end{aligned}$$

bit operations for a single t (needed for determining the $\bar{n}_{i_t,k}$), hence

$$O(\ell^2 d^3 \log^2 d \operatorname{mpe}(q-1)^3 + \ell^2 d \operatorname{mpe}(q-1)^2 \log\log q)$$

bit operations for all t and h together.

Once $\mathfrak{D}$ has been computed, we need to check whether it already occurs among the $\mathfrak{D}_n$ for $n \in \mathcal{N}$ (and update the register accordingly). If we sort $\mathfrak{D}$ lexicographically, we may compare it with a given $\mathfrak{D}_n$ through linear comparison of entries, and so checking whether $\mathfrak{D} = \mathfrak{D}_n$ for some $n \in \mathcal{N}$ takes

$$\begin{aligned} &O(h' \log h' \log q + |\mathcal{N}| h' \log q) \subseteq O(d^2 \operatorname{mpe}(q-1) \cdot \ell \operatorname{mpe}(q-1) \cdot \log q) \\ &\quad = O(\ell d^2 \operatorname{mpe}(q-1)^2 \log q) \end{aligned}$$

bit operations for a single t, hence

$$\begin{aligned} O(\ell H_i \cdot \ell d^2 \operatorname{mpe}(q-1)^2 \log q) &\subseteq O(\ell^2 d^3 \operatorname{mpe}(q-1)^3 \log q) \\ &\subseteq O(\ell d^4 \operatorname{mpe}(q-1)^3 \log q) \end{aligned}$$

bit operations for all t and h together. This last O-expression dominates every other bit operation cost given in this complexity analysis except the cost

$$O(\ell^2 \operatorname{mpe}(q-1) \log^{1+o(1)} q) \subseteq O(\ell d \operatorname{mpe}(q-1) \log^{1+o(1)} q)$$

of computing H_i, and the total cost $O(\ell^2 d^3 \log^2 d \operatorname{mpe}(q-1)^3)$ of computing the descriptions $\mathfrak{D}$. Therefore, the bit operation cost of these computations is in

$$O(\ell d \operatorname{mpe}(q-1) \log^{1+o(1)} q + \ell d^4 \operatorname{mpe}(q-1)^3 \log q + \ell^2 d^3 \log^2 d \operatorname{mpe}(q-1)^3)$$

for a given (i, ℓ) and h. Using that $\sum_{(i,\ell)\in\bar{\mathfrak{L}}} \ell = d$, the total bit operation cost of computing the type-II tree register for f is in

$$\begin{aligned} &O(d \log^{1+o(1)} q + d^2 \log^2 d \\ &\quad + \sum_{(i,\ell)\in\bar{\mathfrak{L}}} (\ell d \operatorname{mpe}(q-1) \log^{1+o(1)} q + \ell d^4 \operatorname{mpe}(q-1)^3 \log q \\ &\qquad + \ell^2 d^3 \log^2 d \operatorname{mpe}(q-1)^3)) \\ &\quad \subseteq O(d^2 \operatorname{mpe}(q-1) \log^{1+o(1)} q + d^5 \operatorname{mpe}(q-1)^3 \log q + d^5 \log^2 d \operatorname{mpe}(q-1)^3), \end{aligned}$$

as asserted. The "In particular" statement holds because $\operatorname{mpe}(q-1) \in O(\log q)$. ∎

So far, everything has been of a q-bounded query complexity that is polynomial in $\log q$ and d. The quantity $\tau(q-1)$, which is generally superpolynomial in $\log q$, enters through the following auxiliary result.

Proposition 5.3.2.3. *Let m be a positive integer, and let $A : x \mapsto ax + b$ be an affine map of $\mathbb{Z}/m\mathbb{Z}$. The cycle type of $A_{|\mathrm{per}(A)}$, denoted by $\mathrm{CT}(A_{|\mathrm{per}(A)})$, can be computed within m-bounded query complexity*

$$(\tau(m)\log^{2+o(1)} m + \tau(m)^2 \log m, 0, \log m, 1, 0).$$

Proof. We start with two suitable m-bounded mord queries, one suitable m-bounded mdl query and a computation of the remainder of a upon division by 4 (which merely consists of scanning the two least significant digits of a). Together, these allow us to factor m, compute $\mathrm{ord}_{p^{\nu_p(m)}}(a)$ for all primes $p \mid m$ such that $p \nmid a$, factor $a-1$ and (only if $\nu_2(m) \geq 2$) compute the (unique) exponents $\varepsilon \in \{0,1\}$ and

$$e \in \{0, 1, \ldots, 2^{\nu_2(m)-2}\}$$

such that $a \equiv (-1)^\varepsilon 5^e \pmod{2^{\nu_2(m)}}$ (first, one determines ε simply by looking at the value of a modulo 4, and then e can be determined with an mdl query). The m-bounded query complexity of these computations is $(\log m, 0, 1, 1, 0)$.

Letting $m'' := \prod_{p|\gcd(a,m)} p^{\nu_p(m)}$ and $m' := m/m''$ (which we do not need to actually compute), we observe the following. Because $A \bmod m''$ has a unique periodic point (see Lemma 2.1.14), we find that $\mathrm{CT}(A_{|\mathrm{per}(A)}) = \mathrm{CT}(A \bmod m')$, and so we compute the latter. This allows us to assume without loss of generality that A is a permutation of $\mathbb{Z}/m\mathbb{Z}$. For each prime $p \mid m$, we set $A_{(p)} := A \bmod p^{\nu_p(m)}$. Since A is given via its coefficients a and b, computing $A_{(p)}$ takes a mere $O(\log^{1+o(1)} m)$ bit operations per p for obtaining the remainders of a and b upon division by $p^{\nu_p(m)}$. Hence, computing all reductions $A_{(p)}$ takes $O(\log^{2+o(1)} m)$ bit operations.

Formulas for $\mathrm{CT}(A_{(p)})$ were given in [15, Tables 3 and 4], and since we know $\mathrm{ord}_{p^{\nu_p(m)}}(a)$ for all $p \mid m$ from our initial mord query, these formulas allow us to compute $\mathrm{CT}(A_{(p)})$ for a given p within m-bounded query complexity

$$(\log^{2+o(1)}(p^{\nu_p(m)}) + \nu_p(m)\log^{1+o(1)}(p^{\nu_p(m)}), 0, 1, 0, 0).$$

Indeed, taking a closer look at those formulas, we see that apart from the information computed in the first paragraph of this proof, we only need to compute a single power and carry out some simpler arithmetic (taking $p^{\nu_p(m)}$-bounded query complexity $(\log^{2+o(1)}(p^{\nu_p(m)}), 0, 1, 0, 0)$), followed by $O(\nu_p(m))$ iterations of a loop, each consisting of $O(1)$ basic arithmetic operations taking $O(\log^{1+o(1)}(p^{\nu_p(m)}))$ bit operations each. For all p together, computing $\mathrm{CT}(A_{(p)})$ has m-bounded query complexity $(\log^{2+o(1)} m, 0, \log m, 0, 0)$. We also note that each $\mathrm{CT}(A_{(p)})$ is a monomial with at most $\nu_p(m)+1$ factors, and that our computation process allows us to store $\mathrm{CT}(A_{(p)})$ with all cycle lengths fully factored.

Now, we may compute $\mathrm{CT}(A)$ via the formula $\mathrm{CT}(A) = \divideontimes_{p|m} \mathrm{CT}(A_{(p)})$, where $\divideontimes$ denotes the Wei–Xu product from [86, Definition 2.2 on pp. 182f.]. This can be done by looping over the $O(\prod_{p|m}(v_p(m)+1)) \subseteq O(\tau(m))$ tuples formed by choosing one variable power in the factorization of each $\mathrm{CT}(A_{(p)})$ and computing the Wei–Xu product of those variable powers (which is itself a variable power) according to [86, formula (2.9) in Lemma 2.3 (b)]. Doing so requires us to compute the least common multiple of the involved cycle lengths, which takes $O(\log^{2+o(1)} m)$ bit operations because those cycle lengths are already fully factored (see also the paragraph on the computation of l in the proof of Corollary 5.3.1.2), followed by $O(\log m)$ integer multiplications and divisions for computing the exponent, which also take $O(\log^{2+o(1)} m)$ bit operations together. In total, the process of computing all relevant Wei–Xu products of variable powers takes $O(\tau(m)\log^{2+o(1)} m)$ bit operations. Once this is done, we need to compute the product of those variable powers, which means that $O(\tau(m))$ times, we need to multiply a monic monomial with $O(\tau(m))$ distinct variable power factors, each with index and exponent in $\{1, 2, \ldots, m\}$, with a single such variable power. Each such multiplication takes $O(\tau(m)\log m)$ bit operations, so the overall complexity of these computations, which result in $\mathrm{CT}(A)$, is in $O(\tau(m)^2 \log m)$. ∎

Remark 5.3.2.4. By our proof of Proposition 5.3.2.3, the cycle type of an affine map A of $\mathbb{Z}/m\mathbb{Z}$ is a product of at most $\tau(m)$ variable powers, and so A has at most $\tau(m)$ distinct cycle lengths.

As far as lower bounds on the maximum number of distinct cycle lengths of an affine map of $\mathbb{Z}/m\mathbb{Z}$ are concerned, let us fix a positive integer K and primes $2 < p_1 < p_2 < \cdots < p_K$ such that $\gcd(p_j, p_{j'} - 1) = 1$ for $1 \leq j < j' \leq K$ (such primes exist for each K by Dirichlet's theorem on primes in arithmetical progressions, see [8, Chapter 7]). For variable positive integers $v_1, v_2, \ldots, v_K$, we set $m := p_1^{v_1} \cdots p_K^{v_K}$ and consider an automorphism $A : x \mapsto ax$ of $\mathbb{Z}/m\mathbb{Z}$ such that a is a primitive root modulo $p_j^{v_j}$ for each j (it is possible to choose a like this because of the Chinese remainder theorem). By [15, Table 3], the cycle lengths of A mod $p_j^{v_j}$ are just the numbers of the form $(p_j - 1)p_j^{v_j'}$, where $v_j' \in \{0, 1, \ldots, v_j - 1\}$. Therefore, the cycle lengths of A are just the numbers of the form $\prod_{j=1}^{K}(p_j - 1) \cdot \prod_{j=1}^{K} p_j^{v_j'}$, whence A has $v_1 \cdots v_K$ distinct cycle lengths. We observe that this cycle length count is asymptotically equivalent to

$$(v_1 + 1) \cdots (v_K + 1) = \tau(m)$$

if $\min\{v_1, \ldots, v_K\} \to \infty$. Moreover, we note that

$$\log m = v_1 \log p_1 + \cdots + v_K \log p_K \leq (v_1 + \cdots + v_K) \log p_K.$$

Now, let us assume that $v_1 = v_2 = \cdots = v_K =: v \to \infty$. Then the number of distinct

cycle lengths of A is

$$v^K = \left(\frac{\log p_K \cdot K \cdot v}{\log p_K \cdot K}\right)^K \geq \left(\frac{\log m}{\log p_K \cdot K}\right)^K$$
$$= \frac{\log^K m}{(\log p_K \cdot K)^K} = c(K, p_K) \cdot \log^K m,$$

where $c(K, p_K) := (K \log p_K)^{-K}$. Because we can construct such a class of examples for each K, the maximum number of distinct cycle lengths of an affine map of $\mathbb{Z}/m\mathbb{Z}$ is in general not bounded from above by a polynomial in $\log m$.

Before we proceed further, we need another auxiliary concept and result.

Definition 5.3.2.5. Let $\vec{\mathfrak{x}} = (\mathfrak{x}_1, \mathfrak{x}_2, \ldots, \mathfrak{x}_n)$ be a finite sequence. A *period length of* $\vec{\mathfrak{x}}$ is a positive divisor m of n such that $\vec{\mathfrak{x}} = \diamond_{t=1}^{n/m}(\mathfrak{x}_1, \mathfrak{x}_2, \ldots, \mathfrak{x}_m)$, where $\diamond$ denotes concatenation (as in Section 3.3). The smallest positive integer that is a period length of $\vec{\mathfrak{x}}$ is denoted by $\mathrm{minperl}(\vec{\mathfrak{x}})$.

Remark 5.3.2.6. We note the following concerning Definition 5.3.2.5.

(1) The number $\mathrm{minperl}(\vec{\mathfrak{x}})$ is well defined because at the very least, the length n of $\vec{\mathfrak{x}}$ is a period length of it.

(2) All elements in $[\vec{\mathfrak{x}}]$, the cyclic equivalence class of the sequence $\vec{\mathfrak{x}}$ (see the second paragraph after Definition 1.4) have the same period lengths, in particular the same minperl-value, as $\vec{\mathfrak{x}}$. We denote this common minperl-value by $\mathrm{minperl}([\vec{\mathfrak{x}}])$.

We can bound the complexity of computing $\mathrm{minperl}(\vec{\mathfrak{x}})$ as follows.

Lemma 5.3.2.7. *Let $\vec{\mathfrak{x}} \in \{0, 1, \ldots, N-1\}^n$, given as a length n list of non-negative integers of bit length l_{bit}. Then* $\mathrm{minperl}(\vec{\mathfrak{x}})$ *can be computed using*

$$O(n \log n \log\log n(\log n + l_{\mathrm{bit}})) \subseteq O(n^{1+o(1)} l_{\mathrm{bit}})$$

bit operations.

Proof. We start by determining n, the number of entries of the tuple $\vec{\mathfrak{x}}$, in its binary representation with $\lfloor \log_2 n \rfloor + 1$ bits. The corresponding incrementation process involves altering

- only 1 bit in case the number to increment is even, i.e., in $O(\frac{n}{2})$ of the cases;
- only 2 bits in case the number to increment is congruent to 1 modulo 4, i.e., in $O(\frac{n}{4})$ of the cases;

and so on. Hence, the number of necessary bit operations to compute n is in

$$O\Big(\sum_{k=1} \infty \frac{k\,n}{2^k}\Big) = O(n).$$

Following that, we factor n deterministically. The current record for the bit operation cost of this is $O(n^{1/5+o(1)})$ due to Harvey and Hittmeir [29] (we could also use an mdl query for this factorization, but then the algorithm is not entirely classical, and $n^{1/5+o(1)}$ is majorized by other terms in this analysis anyway). The rest of the algorithm is analogous to the deterministic procedure for computing a modular multiplicative order described in the proof of Lemma 5.1.6 (2). More specifically, if $n = p_1^{v_1} \cdots p_K^{v_K}$ is the obtained factorization of n, then for $j = 1, 2, \ldots, K$, we perform a binary search to find the smallest $v_j' \in \mathbb{N}_0$ such that $p_j^{v_j'} \prod_{k \neq j} p_k^{v_k}$ is a period length of $\vec{\mathfrak{x}}$, which implies that $v_j' = v_{p_j}(\mathrm{minperl}(\vec{\mathfrak{x}}))$. For each given j, it takes $O(\log v_j) \subseteq O(\log \log n)$ iterations of the binary search loop until v_j' is found, and each iteration costs $O(n(\log n + l_{\mathrm{bit}}))$ bit operations. Because $K \in O(\log n)$, this means that the total bit operation cost of computing the numbers v_j' is in

$$O(n \log n \log \log n (\log n + l_{\mathrm{bit}})),$$

and $\mathrm{minperl}(\vec{\mathfrak{x}}) = \prod_{j=1}^{K} p_j^{v_j'}$ takes $O(\log n \cdot \log^{2+o(1)} n) = O(\log^{3+o(1)} n)$ bit operations to compute by Lemma 5.1.5 (3,6). ∎

We now give the precise definition of the compact description of the isomorphism type of Γ_f we aim to obtain.

Definition 5.3.2.8. Let f be a function $X \to X$, where X is some finite set, and let $\vec{\mathfrak{F}} = (\mathfrak{F}_n)_{n=0,1,\ldots,N}$ be a sequence of pairwise distinct finite rooted tree isomorphism types that contains all isomorphism types of the form $\mathrm{Tree}_{\Gamma_f}(x)$ for $x \in \mathrm{per}(f)$. The *tree necklace list for f relative to $\vec{\mathfrak{F}}$* is the unique set $\{([\vec{\mathfrak{n}}_k], l_k, \mathfrak{m}_k) : k = 1, 2, \ldots, N'\}$ of triples such that the following hold.

(1) $[\vec{\mathfrak{n}}_k] = [\mathfrak{n}_{k,1}, \mathfrak{n}_{k,2}, \ldots, \mathfrak{n}_{k,\ell_k'}]$ is a cyclic sequence of numbers in $\{0, 1, \ldots, N\}$ such that $\mathrm{minperl}([\vec{\mathfrak{n}}_k]) = \ell_k'$.

(2) l_k and $\mathfrak{m}_k$ are positive integers, and l_k is a multiple of ℓ_k'.

(3) In Γ_f, there are precisely $\mathfrak{m}_k$ connected components corresponding to the cyclic sequence of rooted tree isomorphism types

$$[\diamond_{t=1}^{l_k/\ell_k'}(\mathfrak{F}_{\mathfrak{n}_{k,1}}, \mathfrak{F}_{\mathfrak{n}_{k,2}}, \ldots, \mathfrak{F}_{\mathfrak{n}_{k,\ell_k'}})].$$

(4) For each connected component of Γ_f, there is a $k \in \{1, 2, \ldots, N'\}$ such that the said connected component corresponds to

$$[\diamond_{t=1}^{l_k/\ell_k'}(\mathfrak{F}_{\mathfrak{n}_{k,1}}, \mathfrak{F}_{\mathfrak{n}_{k,2}}, \ldots, \mathfrak{F}_{\mathfrak{n}_{k,\ell_k'}})].$$

If f is an index d generalized cyclotomic mapping of the finite field $\mathbb{F}_q$ such that f is of special type I or II, respectively, and if $\mathfrak{R} = ((\mathfrak{D}_n, S_n))_{n=0,1,\ldots,N}$ is a type-I or -II tree register for f, then associated with $\mathfrak{R}$, we have the sequence $(\mathfrak{F}_n)_{n=0,1,\ldots,N}$

of rooted tree isomorphism types where $\mathfrak{I}_n$ has the compact description $\mathfrak{D}_n$. In that case, the tree necklace list for f relative to $\vec{\mathfrak{I}}$ is also called one *relative to* $\mathfrak{R}$.

Remark 5.3.2.9. We make the following comments concerning Definition 5.3.2.8.

(1) The uniqueness of the tree necklace list for f relative to $\vec{\mathfrak{I}}$ is not hard to prove, but it does require that $\mathrm{minperl}([\vec{\mathfrak{n}}_k]) = \ell'_k$ for all k. For example, without this property, for any $\vec{\mathfrak{I}}$ of length $N+1 \geq 2$, both $\{([0,1],4,1)\}$ and $\{([0,1,0,1],4,1)\}$ would be tree necklace lists relative to $\vec{\mathfrak{I}}$ for a suitable function f.

(2) For $j = 1, 2$, let X_j be a finite set and f_j a function $X_j \to X_j$. Moreover, let $\vec{\mathfrak{I}}$ be a finite sequence of pairwise distinct, finite rooted tree isomorphism types such that for $j = 1, 2$, each $\mathrm{Tree}_{\Gamma_{f_j}}(x)$ for $x \in \mathrm{per}(f_j)$ occurs in $\vec{\mathfrak{I}}$. For $j = 1, 2$, let $\mathfrak{N}_j$ be the unique tree necklace list for f_j relative to $\vec{\mathfrak{I}}$. It is not hard to prove that $\mathfrak{N}_1 = \mathfrak{N}_2$ if and only if $\Gamma_{f_1} \cong \Gamma_{f_2}$.

(3) Let f be an index d generalized cyclotomic mapping of $\mathbb{F}_q$ that is of special type I or II. We need to fix suitable bit string encodings of the elements of a tree necklace list for f, making the following conventions. By assumption, if $i \in \{0, 1, \ldots, d\}$ has cycle length ℓ under f, then the cyclic sequence of rooted tree isomorphism types encoding the connected component of Γ_f containing any f-periodic vertex from C_i has minimal period some divisor of ℓ. In particular, the said minimal period is always at most d. Moreover, we assume that $\vec{\mathfrak{I}}$ stems from a recursive tree description list $\vec{\mathfrak{D}}$ that is part of a type-I or -II tree register $\mathfrak{R}$ for f. In $\vec{\mathfrak{D}}$, there is a common bit length to represent numbers from $\{0, 1, \ldots, N\}$ (see the remarks after Definition 5.3.2.1); we denote that bit length by l_{bit} and observe that

$$l_{\mathrm{bit}} = \begin{cases} \lfloor \log_2 d \rfloor + 1 \in O(\log d), & \text{if } \mathfrak{R} \text{ has type I,} \\ \lfloor \log_2(d^2 \lfloor \log_2 q \rfloor + d) \rfloor + 1 \in O(\log d + \log\log q), & \text{if } \mathfrak{R} \text{ has type II.} \end{cases}$$

A cyclic sequence $[\vec{\mathfrak{n}}] = [\mathfrak{n}_1, \mathfrak{n}_2, \ldots, \mathfrak{n}_{\ell'}]$ as above is the first entry of an element of the tree necklace list for f relative to $\mathfrak{R}$; we assume that the associated ordered sequence $\vec{\mathfrak{n}} = (\mathfrak{n}_1, \mathfrak{n}_2, \ldots, \mathfrak{n}_{\ell'})$ is lexicographically minimal among all ordered sequences in its cyclic equivalence class $[\vec{\mathfrak{n}}]$. We encode $[\vec{\mathfrak{n}}]$ as follows. We take the ordered sequence $\vec{\mathfrak{n}}$ and fill it up with entries -1 (a dummy value) until it has d entries. We then print a bit string that is a concatenation of encodings of the entries of this extended sequence. We use $l_{\mathrm{bit}} + 1$ bits to denote each entry, where an entry other than -1 is represented by an ancillary bit 1, followed by the length l_{bit} binary digit representation of that entry from $\mathfrak{R}$. On the other hand, an entry -1 is represented by a string of $l_{\mathrm{bit}} + 1$ zeroes. For example, if $l_{\mathrm{bit}} = 3$ and $d = 5$, then the bit string encoding

of $[6, 3, 4] = [3, 4, 6]$ in the corresponding tree necklace list is

$$10111100111000000000.$$

On the other hand, the second and third entries of elements of any tree necklace list for f are positive integers that are at most q, and we simply use their standard binary representations with $\lfloor \log_2 q \rfloor + 1$ digits to represent them; these may be directly concatenated with the bit string encoding of $[\vec{\mathfrak{n}}]$. With these conventions, all elements of a given tree necklace list for f are bit strings of the same bit length, namely

$$d(l_{\text{bit}} + 1) + 2(\lfloor \log_2 q \rfloor + 1) \in \begin{cases} O(d \log d + \log q), & \text{if } \mathfrak{R} \text{ has type I,} \\ O(d \log d + d \log\log q + \log q), & \text{if } \mathfrak{R} \text{ has type II,} \end{cases}$$

which allows us to use the sorting algorithm from Lemma 5.1.5 (10) to sort them lexicographically. Moreover, the lexicographic ordering of those bit strings corresponds to the "natural" lexicographic ordering of the elements of the associated (abstract) tree necklace list (using the lexicographic ordering among lexicographically minimal representatives of cyclic sequences in the first component, and the usual linear ordering of integers in the second and third component). It should be noted that our approach involves some padding, and this could be avoided through using [6, Algorithm 3.2 on pp. 80f.], which is a more general lexicographic sorting algorithm that does not require the bit strings from the input to be of a common length. However, in terms of the O-class of the complexity bounds we derive, it does not make a difference.

Remark 5.3.2.9 (2) guarantees that tree necklace lists are *injective* descriptions of digraph isomorphism types of functional graphs, but they may not always be compact. Indeed, they contain as many elements as there are distinct isomorphism types of connected components of the said functional graph, and in the case of the functional graph Γ_f of a generalized cyclotomic mapping f of $\mathbb{F}_q$ of a fixed index d, the maximum number of such connected components is at least the maximum number of distinct cycle lengths which an affine permutation of $\mathbb{Z}/s\mathbb{Z}$ can achieve. That latter number can, a priori, be superpolynomial in $s = (q-1)/d$ (and thus in q if d is fixed), see Remark 5.3.2.4. We note however, that the moduli considered in Remark 5.3.2.4 are of a special form, and it is not clear whether the construction from Remark 5.3.2.4 can still be carried out if, additionally, all constructed moduli must be of the form $(q-1)/d$ for some prime power q with $d \mid q-1$, where $d \in \mathbb{N}^+$ is fixed. Moreover, by our Proposition 5.3.1.4, as long as one is willing to exclude a small positive asymptotic fraction of prime powers, then $\tau(q-1)$, which is an upper bound on the

number of distinct cycle lengths of an affine map of $\mathbb{Z}/s\mathbb{Z}$ (see the proof of Proposition 5.3.2.3), *is* polynomial in $\log q$. Even for such prime powers q, the number of distinct isomorphism types of connected components of Γ_f itself could be superpolynomial in $\log q$, however. We leave the problem of finding a concrete infinite class of examples that confirms this open; see also Problems 6.3.3 and 6.3.4.

Nonetheless, if f is of special type I or II, then the following key result implies that one may compute a tree register $\mathfrak{R}$ for f and, subsequently, compute and print the tree necklace list for f relative to $\mathfrak{R}$ within a q-bounded query complexity that is polynomial in $\log q$, d and $\tau(q-1)$.

Theorem 5.3.2.10. *Let f be an index d generalized cyclotomic mapping of $\mathbb{F}_q$.*

(1) *We assume that f is of special type* I. *Then one can compute within a q-bounded query complexity of*

$$(d^3\log^2 d + d^3\tau(q-1)^2\log q + d^2\log^{1+o(1)} q + d\tau(q-1)\log^{2+o(1)} q, d, d\log q, d, 0),$$

or a q-bounded Las Vegas dual complexity of

$$(d^3\log^2 d + d^3\tau(q-1)^2\log q + d^2\log^{1+o(1)} q + d\tau(q-1)\log^{2+o(1)} q + d\log^{8+o(1)} q, d\log^{4+o(1)} q, d\log^2 q),$$

*a type-*I *tree register $\mathfrak{R} = ((\mathfrak{D}_n, S_n))_{n=0,1,\ldots,N}$ for f with $N \in O(d)$, as well as the tree necklace list of f relative to $\mathfrak{R}$.*

(2) *We assume that f is of special type* II. *Then one can compute within a q-bounded query complexity of*

$$(d^5\log^2 d\,\mathrm{mpe}(q-1)^3 + (d^5\,\mathrm{mpe}(q-1)^3 + d^3\tau(q-1)^2)\log q + d^2\,\mathrm{mpe}(q-1)\log^{1+o(1)} q + d\tau(q-1)\log^{2+o(1)} q, d, d\log q, d, 0),$$

or a q-bounded Las Vegas dual complexity of

$$(d^5\log^2 d\,\mathrm{mpe}(q-1)^3 + (d^5\,\mathrm{mpe}(q-1)^3 + d^3\tau(q-1)^2)\log q + d^2\,\mathrm{mpe}(q-1)\log^{1+o(1)} q + d\tau(q-1)\log^{2+o(1)} q + d\log^{8+o(1)} q, d\log^{4+o(1)} q, d\log^2 q),$$

*a type-*II *tree register $\mathfrak{R} = ((\mathfrak{D}_n, S_n))_{n=0,1,\ldots,N}$ of f with*

$$N \in O(d^2\,\mathrm{mpe}(q-1)),$$

as well as the tree necklace list of f relative to $\mathfrak{R}$.

Proof. We prove both statements simultaneously, referring with "case I", respectively, "case II", to the situation described in statement (1), respectively, (2). First, we compute $\bar{f}$, the affine maps A_i, and a tree register $\mathfrak{R}$ for f of the desired type and with the asserted bound on N, taking q-bounded query complexity

- $(d^3 \log^2 d + d \log^{1+o(1)} q, d, 0, 0, 0)$ in case I,
- $(d^2 \operatorname{mpe}(q-1) \log^{1+o(1)} q + d^5 \operatorname{mpe}(q-1)^3 \log q + d^5 \log^2 d \operatorname{mpe}(q-1)^3, d, 0, 0, 0)$ in case II

by Proposition 5.1.8 and Lemma 5.3.2.2 (2,4). Then we compute a CRL-list $\bar{\mathfrak{L}}$ of $\bar{f}$ together with the cycles of $\bar{f}$, taking $O(d \log^2 d)$ bit operations by the argument at the beginning of Section 5.2.1. For each $(i, \ell) \in \bar{\mathfrak{L}}$, letting $(i_0, i_1, \dots, i_{\ell-1})$ with $i = i_0$ denote the $\bar{f}$-cycle of i determined earlier, we compute the tree necklace list $\mathfrak{N}_i$, relative to $\mathfrak{R}$, for the restriction of f to $\bigcup_{t=0}^{\ell-1} C_{i_t}$ as follows.

If $i = d$, we simply set $\mathfrak{N}_i := \{([\mathfrak{n}], 1, 1)\}$, where $\mathfrak{n} \in \{0, 1, \dots, N\}$ is the positive integer that represents $\operatorname{Tree}_{\Gamma_f}(0_{\mathbb{F}_q})$ in $\mathfrak{R}$. We observe that in case II, where $\operatorname{Tree}_{\Gamma_f}(0_{\mathbb{F}_q})$ is trivial, one has $\mathfrak{n} = 0$ necessarily, whereas in case I, the number $\mathfrak{n}$ is uniquely characterized by the inclusion $d \in S_{\mathfrak{n}}$, and can thus be determined with $O(d \log d)$ bit operations (using that each $\mathfrak{n}$ is represented by a bit string of length in $O(\log d)$).

Now we assume that $i < d$. Then we compute $\mathcal{A}_i = A_{i_0} A_{i_1} \cdots A_{i_{\ell-1}}$, taking $O(d \log^{1+o(1)} q)$ bit operations (see Section 5.2.1, page 118), as well as the cycle type $\operatorname{CT}((\mathcal{A}_i)_{|\operatorname{per}(\mathcal{A}_i)}) = x_1^{e_{i,1}} x_2^{e_{i,2}} \cdots x_s^{e_{i,s}}$ (where some of the $e_{i,j}$ may be 0), taking q-bounded query complexity

$$(\tau(q-1) \log^{2+o(1)} q + \tau(q-1)^2 \log q, 0, \log q, 1, 0)$$

by Proposition 5.3.2.3. Moreover, we compute the number sequence

$$\vec{\mathfrak{n}}_i = (\mathfrak{n}_{i_0}, \mathfrak{n}_{i_1}, \dots, \mathfrak{n}_{i_{\ell-1}}),$$

where $\mathfrak{n}_{i_t}$ is uniquely characterized by the inclusion

$$\begin{cases} i_t \in S_{\mathfrak{n}_{i_t}}, & \text{in case I,} \\ i_t \in S_{\mathfrak{n}_{i_t}, \operatorname{per}}, & \text{in case II,} \end{cases}$$

i.e., $\mathfrak{n}_{i_t}$ is the positive integer that represents, in the register $\mathfrak{R}$, the unique rooted tree isomorphism type above periodic vertices of Γ_f that are contained in C_{i_t}. The number of bit operations it takes to determine (and set the value of) $\mathfrak{n}_{i_t}$ for all $t = 0, 1, \dots, \ell - 1$ can be bounded as follows:

- in case I, it is in

$$O\left(d \cdot \sum_{n=0}^{N} |S_n| \cdot \log d + d\, l_{\text{bit}}\right) \subseteq O(d^2 \log d);$$

- in case II, it is in

$$O\left(d\left(N+\sum_{n=0}^{N}|S_{n,\mathrm{per}}|\right)\log d + d\,l_{\mathrm{bit}}\right) \subseteq O(d^3 \log d\,\mathrm{mpe}(q-1) + d\log\log q).$$

Next, we overwrite $\vec{\mathfrak{n}}_i$ with the unique lexicographically minimal number sequence in the same cyclic equivalence class. We can do by spelling the $O(\ell) \subseteq O(d)$ cyclic shifts of $\vec{\mathfrak{n}}_i$ out, then ordering them lexicographically as in Lemma 5.1.5 (10) and taking the first sequence in the sorted list. This process takes

$$\begin{aligned} O(d\log d \cdot d\,l_{\mathrm{bit}}) &= O(d^2 \log d\, l_{\mathrm{bit}}) \\ &\subseteq \begin{cases} O(d^2\log^2 d), & \text{in case I,} \\ O(d^2\log^2 d + d^2 \log d \log\log q), & \text{in case II} \end{cases} \end{aligned}$$

bit operations. Following that, we compute $\mathrm{minperl}(\vec{\mathfrak{n}}_i)$, which by Lemma 5.3.2.7 takes the following amount of bit operations:

$$\begin{aligned} &O(d\log d\log\log d\cdot(\log d + l_{\mathrm{bit}})) \\ &\subseteq \begin{cases} O(d\log^2 d\log\log d), & \text{in case I,} \\ O(d\log d\log\log d(\log d + \log\log q)), & \text{in case II.} \end{cases} \end{aligned}$$

We observe that the following is a valid choice for $\mathfrak{N}_i$:

$$\mathfrak{N}_i := \{([\mathfrak{n}_{i_0}, \mathfrak{n}_{i_1}, \ldots, \mathfrak{n}_{i_{\mathrm{minperl}(\vec{\mathfrak{n}}_i)-1}}], \ell\cdot l', e_{i,l'}) : l' \in \{1,2,\ldots,s\}, e_{i,l'} > 0\}.$$

For further processing, rather than store $\mathfrak{N}_i$ literally as this list, it is more advantageous to store $\vec{\mathfrak{n}}'_i := [\mathfrak{n}_{i_0}, \mathfrak{n}_{i_1}, \ldots, \mathfrak{n}_{i_{\mathrm{minperl}(\vec{\mathfrak{n}}_i)-1}}]$ and the list

$$\mathfrak{N}'_i := \{(\ell\cdot l', e_{i,l'}) : l' \in \{1,2,\ldots,s\}, e_{i,l'} > 0\}$$

separately, which takes

$$\begin{aligned} &O(\mathrm{minperl}(\vec{\mathfrak{n}}_i) l_{\mathrm{bit}} + \tau(q-1)\log^{1+o(1)} q) \\ &\subseteq \begin{cases} O(d\log d + \tau(q-1)\log^{1+o(1)} q), & \text{in case I,} \\ O(d(\log d + \log\log q) + \tau(q-1)\log^{1+o(1)} q), & \text{in case II} \end{cases} \end{aligned}$$

bit operations for carrying out the multiplications $\ell\cdot l'$ and copying data. This concludes our analysis of how to compute $\mathfrak{N}_i$, which in summary for each given i takes q-bounded query complexity

$$\begin{aligned} &(\tau(q-1)\log^{2+o(1)} q + (d+\tau(q-1))\log^{1+o(1)} q \\ &\quad + \tau(q-1)^2\log q + E(d,q), 0, \log q, 1, 0), \end{aligned}$$

where

$$E(d,q) := Nd\log d + d^2\log d\, l_{\text{bit}}$$
$$\in \begin{cases} O(d^2\log^2 d), & \text{in case I,} \\ O(d^3\log d\,\operatorname{mpe}(q-1) + d^2\log^2 d + d^2\log d\log\log q), & \text{in case II} \end{cases}$$

and the additional factor $\tau(q-1)$ in the bit operation cost is because there are only $\tau(s) \le \tau(q-1)$ distinct cycle lengths by the proof of Proposition 5.3.2.3 (see also the first few lines in Remark 5.3.2.4). For all i together, the q-bounded query complexity is the (component-wise) d-fold of that.

Finally, we need to compute the actual tree necklace list $\mathfrak{N}$ for f relative to $\mathfrak{R}$. We start by setting $\mathfrak{M} := \mathfrak{N} := \emptyset$ and $\vec{\mathfrak{n}}' := \{(\vec{\mathfrak{n}}'_i, i, \ell) : (i,\ell) \in \bar{\mathcal{L}}\}$ (where $\vec{\mathfrak{n}}'_d := [\mathfrak{n}]$, the first entry of the unique triple in $\mathfrak{N}_d$), followed by sorting $\vec{\mathfrak{n}}'$ lexicographically. Altogether, this takes $O(d^2\log d\, l_{\text{bit}})$ bit operations. Throughout the subsequently described process, $\mathfrak{N}$ is a (lexicographically sorted) initial segment of the tree necklace list that will be output, and $\mathfrak{M}$ is an initial segment of the list obtained by deleting repeated entries in the multiset $\{\vec{\mathfrak{n}}'_i : (i,\ell) \in \bar{\mathcal{L}}\}$. In particular, $\mathfrak{M}$ has $O(d)$ elements, each of which is a cyclic sequence of length in $O(d)$ each entry of which has bit length in $O(l_{\text{bit}})$ and which is given by its lexicographically minimal representative. We observe that two such cyclic sequences are equal if and only if their representatives are equal, so it takes $O(d\, l_{\text{bit}})$ bit operations to verify such an equality.

Now, we go through the triples $(\vec{\mathfrak{n}}'_i, i, \ell) \in \vec{\mathfrak{n}}'$, and for each of them, we do the following. First, we check whether $\vec{\mathfrak{n}}'_i \in \mathfrak{M}$, which takes $O(d^2 l_{\text{bit}})$ bit operations. If so, we skip to the next triple in $\vec{\mathfrak{n}}'$, otherwise we proceed as follows. We add $\vec{\mathfrak{n}}'_i$ to $\mathfrak{M}$ as a new element (which takes $O(d\, l_{\text{bit}})$ bit operations for copying), and set $\mathfrak{V} := \mathfrak{N}'_i$, taking $O(\tau(q-1)\log q)$ bit operations.

Then, we go through the elements $(\vec{\mathfrak{n}}'_j, j, \ell') \in \vec{\mathfrak{n}}'$ that come after $(\vec{\mathfrak{n}}'_i, i, \ell)$, and for each of them, we do the following. We check whether $\vec{\mathfrak{n}}'_j = \vec{\mathfrak{n}}'_i$, which takes $O(d\, l_{\text{bit}})$ bit operations. If not, we skip to the next value of $(\vec{\mathfrak{n}}'_j, j, \ell')$, otherwise we proceed as follows. We go through the elements (l,k) of $\mathfrak{N}'_j$, and for each of them, we check whether l occurs as the first entry of some pair (l,k') of $\mathfrak{V}$; each such check takes $O(|\mathfrak{V}|\log q) \subseteq O(d\,\tau(q-1)\log q)$ bit operations. If so, we replace the unique element of $\mathfrak{V}$ of the form (l,k') by $(l,k+k')$; otherwise, we add (l,k) to $\mathfrak{V}$ as a new element.

Overall, the described loop over the elements of $\mathfrak{N}'_j$ takes

$$O(|\mathfrak{N}'_j| \cdot d\,\tau(q-1)\log q) \subseteq O(d\,\tau(q-1)^2\log q)$$

bit operations, and thus the loop over $(\vec{\mathfrak{n}}'_j, j, \ell')$ takes $O(d^2 l_{\text{bit}} + d^2\tau(q-1)^2\log q)$ bit operations. Once that loop is finished, we complete the loop over $(\vec{\mathfrak{n}}'_i, i, \ell) \in \vec{\mathfrak{n}}'$ by adding, for each $(l,k) \in \mathfrak{V}$, the triple $([\vec{\mathfrak{n}}'_i], l, k)$ to $\mathfrak{N}$ as a new element – this copying

process takes

$$O(|\mathfrak{V}| \cdot (d\, l_{\text{bit}} + \log q)) \subseteq O(d\,\tau(q-1)(d\, l_{\text{bit}} + \log q))$$

bit operations.

At the end of the loop over $(\vec{\mathfrak{n}}'_i, i, \ell) \in \vec{\mathfrak{n}}'$, the variable $\mathfrak{N}$ has its desired value, and the overall bit operation cost of this loop is in

$$O(d \cdot (d^2 l_{\text{bit}} + d^2\tau(q-1)l_{\text{bit}} + d^2\tau(q-1)^2 \log q)) = O(d^3\tau(q-1)^2 \log q). \blacksquare$$

Finally, we discuss the complexity of the digraph isomorphism problem. Let f_1 and f_2 be generalized cyclotomic mappings of $\mathbb{F}_q$, each of one of the special types I or II. We note that neither do f_1 and f_2 need to have the same index, nor are they necessarily both of the same special type. In order to decide whether $\Gamma_{f_1} \cong \Gamma_{f_2}$, we would like to compare a computed tree necklace list for f_1 with one for f_2. To that end, those tree necklace lists must be "synchronized", so that each n denotes the same rooted tree isomorphism type $\mathfrak{T}_n$ in each case. Here is a precise definition.

Definition 5.3.2.11. Let $\vec{\mathfrak{D}} = (\mathfrak{D}_n)_{n=0,1,\ldots,N}$ and $\vec{\mathfrak{D}}' = (\mathfrak{D}'_n)_{n=0,1,\ldots,N'}$ be recursive tree description lists, with associated rooted tree isomorphism type sequences

$$(\mathfrak{T}_n)_{n=0,1,\ldots,N} \text{ and } (\mathfrak{T}'_n)_{n=0,1,\ldots,N'}.$$

A *synchronization of* $\vec{\mathfrak{D}}$ *and* $\vec{\mathfrak{D}}'$ is a pair $(\vec{\mathfrak{D}}^+, \mathrm{i})$ such that the following hold.

(1) $\vec{\mathfrak{D}}^+ = (\mathfrak{D}^+_n)_{n=0,1,\ldots,N^+}$ is a recursive tree description list of which $\vec{\mathfrak{D}}$ is an initial segment (in particular, $N \leq N^+$). We denote by $(\mathfrak{T}^+_n)_{n=0,1,\ldots,N^+}$ the unique rooted tree isomorphism type sequence associated with $\vec{\mathfrak{D}}^+$.

(2) i is a function $\{0, 1, \ldots, N'\} \to \{0, 1, \ldots, N^+\}$ with

$$\{N+1, N+2, \ldots, N^+\} \subseteq \mathrm{im}(\mathrm{i})$$

such that for each $n \in \{0, 1, \ldots, N'\}$, one has $\mathfrak{T}'_n \cong \mathfrak{T}^+_{\mathrm{i}(n)}$.

In an implementation, we assume that each description $\mathfrak{D}_n$ or $\mathfrak{D}'_n$ is sorted by increasing first entries of its elements. We also assume that each of $\vec{\mathfrak{D}}$ and $\vec{\mathfrak{D}}'$ uses a common bit length, denoted by l_{bit} and l'_{bit}, respectively, for the binary representations of the numbers n. Because $N^+ \leq N + N'$, we use $\max\{l_{\text{bit}}, l'_{\text{bit}}\} + 1 \in O(l_{\text{bit}} + l'_{\text{bit}})$ bits for the numbers n in the synchronization $\vec{\mathfrak{D}}^+$. We note that the definition of a synchronization is asymmetric in the sense that a synchronization of $\vec{\mathfrak{D}}$ and $\vec{\mathfrak{D}}'$ is not necessarily also one of $\vec{\mathfrak{D}}'$ and $\vec{\mathfrak{D}}$. Complexity-wise, the following lemma shows that it is slightly more advantageous to have $N' \leq N$.

Lemma 5.3.2.12. *Let* $\vec{\mathfrak{D}} = (\mathfrak{D}_n)_{n=0,1,\ldots,N}$ *and* $\vec{\mathfrak{D}}' = (\mathfrak{D}'_n)_{n=0,1,\ldots,N'}$ *be recursive tree description lists such that for all* n,

- *each second entry of an element of $\mathfrak{D}_n$ or $\mathfrak{D}'_n$ is represented by a bit string of length at most m (a quantity that does not depend on n);*
- *each first entry of each element of $\mathfrak{D}_n$, respectively of $\mathfrak{D}'_n$, is represented by a bit string of length exactly l_{bit}, respectively, l'_{bit}, and*
- *within a given description $\mathfrak{D}_n$, respectively, $\mathfrak{D}'_n$, the second entries of elements of that description have a common bit length (so all pairs in $\mathfrak{D}_n$, respectively in $\mathfrak{D}'_n$, have the same bit length, which lies in $O(l_{\mathrm{bit}} + m)$, respectively in $O(l'_{\mathrm{bit}} + m)$).*

It takes $O((NN' \min\{N, N'\} + (\max\{N, N'\})^2 + (N')^2 l'_{\mathrm{bit}})(l_{\mathrm{bit}} + l'_{\mathrm{bit}} + m))$ bit operations to compute a synchronization of $\vec{\mathfrak{D}}$ and $\vec{\mathfrak{D}}'$.

Proof. Let $\mathfrak{I}_n$, respectively, $\mathfrak{I}'_n$, be the rooted tree isomorphism type described by $\mathfrak{D}_n$, respectively, by $\mathfrak{D}'_n$. In order to compute the synchronization, we proceed in a loop over $n = 0, 1, \ldots, N'$. At each given point in the process, the description $\mathfrak{D}^+_k$ (of the rooted tree isomorphism type $\mathfrak{I}^+_k$) is defined for all $k \in \mathcal{N}$, an initial segment of $\mathbb{N}_0$ that starts out as $\{0, 1, \ldots, N\}$, with $\mathfrak{D}^+_k := \mathfrak{D}_k$ for each $k \in \{0, 1, \ldots, N\}$, and will be $\{0, 1, \ldots, N^+\}$ in the end. We note that it takes

$$O(N^2(l_{\mathrm{bit}} + \log m)) \subseteq O((\max\{N, N'\})^2(l_{\mathrm{bit}} + l'_{\mathrm{bit}} + m))$$

bit operations (spent copying) to set $\mathcal{N}$ and the $\mathfrak{D}^+_k$ for $k \in \{0, 1, \ldots, N\}$ up. At any given point in the algorithm, the descriptions $\mathfrak{D}^+_k$ form a recursive tree description list denoted by $\vec{\mathfrak{D}}^+$ (which will have the desired value in the end). We also keep updating the value of $\mathrm{i} : \{0, 1, \ldots, n\} \to \mathcal{N}$, which starts out as the empty function $\emptyset$.

For $n = 0$, where $\mathfrak{D}'_n = \emptyset$ and its associated rooted tree is trivial, we simply set $\mathrm{i}(0) := 0$ without updating $\mathcal{N}$. Now let us assume that $n \geq 1$. Based on $\mathfrak{D}'_n$, we compute a new rooted tree description $\mathfrak{D}$ through replacing the first entry $k < n \leq N'$ of each given pair in $\mathfrak{D}'_n$ by $\mathrm{i}(k)$. Because $\mathfrak{D}'_n$ contains at most $n + 1 \in O(n)$ distinct pairs and we are handling non-negative integers of bit length in $O(l_{\mathrm{bit}} + l'_{\mathrm{bit}})$ here, computing $\mathfrak{D}$ as an unsorted list takes $O(n \cdot (l_{\mathrm{bit}} + l'_{\mathrm{bit}})) \subseteq O(N'(l_{\mathrm{bit}} + l'_{\mathrm{bit}}))$ bit operations overall (for a given n), and another $O(N' l'_{\mathrm{bit}}(l_{\mathrm{bit}} + l'_{\mathrm{bit}} + m))$ bit operations for sorting $\mathfrak{D}$.

Once $\mathfrak{D}$ has been computed in sorted form, we need to check whether the rooted tree $\mathfrak{I}'_n$ described by it with respect to $\vec{\mathfrak{D}}^+$ already occurs among the $\mathfrak{I}^+_k$. If $k > N$, then $\mathfrak{I}^+_k \cong \mathfrak{I}'_l$ for some $l \in \{0, 1, \ldots, n-1\}$, and thus $\mathfrak{I}^+_k \ncong \mathfrak{I}'_n$, as the isomorphism types $\mathfrak{I}'_t$ are pairwise distinct by assumption. Therefore, we only need to check the isomorphism $\mathfrak{I}^+_k \cong \mathfrak{I}'_n$ for $k \leq N$, where it is equivalent to $\mathfrak{I}_k \cong \mathfrak{I}'_n$ and further to $\mathfrak{D}_k = \mathfrak{D}$. For a given k, it takes $O(\min\{N, N'\}(l_{\mathrm{bit}} + l'_{\mathrm{bit}} + m))$ bit operations to check with a linear scan whether $\mathfrak{D}_k = \mathfrak{D}$ (using that both $\mathfrak{D}_k$ and $\mathfrak{D}$ are sorted), and so it can be checked with $O(N \min\{N, N'\}(l_{\mathrm{bit}} + l'_{\mathrm{bit}} + m))$ bit operations whether $\mathfrak{I}'_n \cong \mathfrak{I}^+_k$ for a (unique) $k \in \{0, 1, \ldots, N\}$. If so, we set $\mathrm{i}(n) := k$ without updating $\mathcal{N}$;

otherwise, we extend $\mathcal{N}$ by the new element $n' := \max \mathcal{N} + 1$ and set $\mathfrak{D}^+_{n'} := \mathfrak{D}$ and $\mathrm{i}(n) := n'$.

The overall bit operation cost of the described loop is in $O((NN' \min\{N, N'\} + (N')^2 l'_{\mathrm{bit}})(l_{\mathrm{bit}} + l'_{\mathrm{bit}} + m))$. We conclude the algorithm by outputting $(\vec{\mathfrak{D}}^+, \mathrm{i})$, where

$$\vec{\mathfrak{D}}^+ := (\mathfrak{D}^+_k)_{k \in \mathcal{N}}.$$

This takes

$$\begin{aligned} &O((N + N') \max\{N, N'\}(l_{\mathrm{bit}} + l'_{\mathrm{bit}} + m)) \\ &= O((\max\{N, N'\})^2 (l_{\mathrm{bit}} + l'_{\mathrm{bit}} + m)) \end{aligned}$$

bit operations for copying. ∎

Corollary 5.3.2.13. *Let f_1 and f_2 be generalized cyclotomic mappings of a common finite field $\mathbb{F}_q$, say of index d_1 and d_2, respectively, and set $d := \max\{d_1, d_2\}$.*

(1) *If each f_j is of special type* I *or* II *(not necessarily both of the same type), then it takes q-bounded query complexity*

$$\begin{aligned} &((d^6 \operatorname{mpe}(q-1)^3 + d^3 \tau (q-1)^2) \log q + d^4 \operatorname{mpe}(q-1)^2 \log^{1+o(1)} q \\ &\quad + d\,\tau(q-1) \log^{2+o(1)} q, d, d \log q, d, 0), \end{aligned}$$

or q-bounded Las Vegas dual complexity

$$\begin{aligned} &((d^6 \operatorname{mpe}(q-1)^3 + d^3 \tau (q-1)^2) \log q + d^4 \operatorname{mpe}(q-1)^2 \log^{1+o(1)} q \\ &\quad + d\,\tau(q-1) \log^{2+o(1)} q + d \log^{8+o(1)} q, d \log^{4+o(1)} q, d \log^2 q), \end{aligned}$$

to decide whether $\Gamma_{f_1} \cong \Gamma_{f_2}$.

(2) *If both f_j are of special type I, then it takes q-bounded query complexity*

$$\begin{aligned} &(d^3 \log^2(d) + d^3 \tau (q-1)^2 \log q + d^2 \log^{1+o(1)} q \\ &\quad + d\,\tau(q-1) \log^{2+o(1)} q, d, d \log q, d, 0), \end{aligned}$$

or q-bounded Las Vegas dual complexity

$$\begin{aligned} &(d^3 \log^2(d) + d^3 \tau (q-1)^2 \log q + d^2 \log^{1+o(1)} q \\ &\quad + d\,\tau(q-1) \log^{2+o(1)} q + d \log^{8+o(1)} q, d \log^{4+o(1)} q, d \log^2 q), \end{aligned}$$

to decide whether $\Gamma_{f_1} \cong \Gamma_{f_2}$.

Proof. We denote the situation in statement (1) by "case (1)", and the one in statement (2) by "case (2)"; these must *not* be confused with cases I and II from the proof of Theorem 5.3.2.10. First, for $j = 1, 2$, we compute a suitable tree register $\mathfrak{R}_j$ of f_j

with $N_j + 1$ entries, together with an associated tree necklace list $\mathfrak{N}_j$ for f_j. By Theorem 5.3.2.10, this takes q-bounded query complexity

$$\begin{aligned}&(d^5 \log^2 d \operatorname{mpe}(q-1)^3 + (d^5 \operatorname{mpe}(q-1)^3 + d^3\tau(q-1)^2) \log q\\ &\quad + d^2 \operatorname{mpe}(q-1) \log^{1+o(1)} q + d\tau(q-1) \log^{2+o(1)} q,\\ &\quad d, d \log q, d, 0),\end{aligned}$$

in case (1), or

$$\begin{aligned}&(d^3 \log^2 d + d^3\tau(q-1)^2 \log q + d^2 \log^{1+o(1)} q + d\tau(q-1) \log^{2+o(1)} q,\\ &\quad d, d \log q, d, 0),\end{aligned}$$

in case (2).

Following that, we synchronize the underlying recursive tree description lists of the $\mathfrak{N}_j$. By Lemma 5.3.2.12, applied with $m := \log q$, and using the facts that $l_{\mathrm{bit}} \in O(\log q)$ and $l_{\mathrm{bit}}, l'_{\mathrm{bit}} \in O(\log d + \log\log q)$ in either case and that

$$\min\{N_1, N_2\} \leq \max\{N_1, N_2\} \in \begin{cases} O(d^2 \operatorname{mpe}(q-1)), & \text{in case (1),}\\ O(d), & \text{in case (2),}\end{cases}$$

we see that this can be done using

$$\begin{aligned}&O((d^6 \operatorname{mpe}(q-1)^3 + d^4 \operatorname{mpe}(q-1)^2 \log\log q)(\log d + \log\log q + \log q))\\ &\quad = O((d^6 \operatorname{mpe}(q-1)^3 + d^4 \operatorname{mpe}(q-1)^2 \log\log q) \log q)\end{aligned}$$

bit operations in case (1), or $O(d^3 \log q)$ bit operations in case (2). Following our convention on the bit lengths of indices n of descriptions $\mathfrak{D}_n^+$ in synchronizations, the computed synchronization uses a common bit length l_{bit}^+, which lies in $O(\log d + \log\log q)$ in case (1), and in $O(\log d)$ in case (2).

Next, based on $\mathfrak{N}_2$, we compute a modified tree necklace list $\mathfrak{N}'_2$ for f_2 with respect to the computed synchronization $(\vec{\mathfrak{D}}^+, \mathfrak{i})$ through replacing each number $\mathfrak{n}$ occurring as an entry in one of the cyclic sequences in $\mathfrak{N}_2$ by $\mathfrak{i}(\mathfrak{n})$, then replacing the underlying ordered sequence with the lexicographically minimal representative in the same cyclic equivalence class. Computing $\mathfrak{N}'_2$ in unsorted form takes

$$O(|\mathfrak{N}_2| \cdot (d l_{\mathrm{bit}}^+ + \log q) + d \cdot d \log d\, l_{\mathrm{bit}}^+) \subseteq O(d^2\tau(q-1) \log q + d^2 \log d \log q)$$

bit operations, and following that, we sort $\mathfrak{N}'_2$ lexicographically, which takes

$$O(d^3\tau(q-1) \log q)$$

bit operations through successively merging the $O(d)$ segments corresponding to a common, rewritten first entry (see Lemma 5.1.5 (11)).

Finally, we need to check whether $\mathfrak{N}_2' = \mathfrak{N}_1$, which only takes a linear scan thanks to $\mathfrak{N}_2'$ and $\mathfrak{N}_1$ both being lexicographically sorted (we note that the bit lengths of indices n in $\mathfrak{N}_1$ may not be the same as those in $\mathfrak{N}_2'$, but that is of course not a problem). The bit operation cost of this is in

$$O(\min\{|\mathfrak{N}_1|, |\mathfrak{N}_2'|\} \cdot (d\, l_{\text{bit}}^{+} + \log q)) \subseteq O(d\,\tau(q-1) \cdot d \log q) = O(d^2 \tau(q-1) \log q).$$

■

As we did throughout Section 5.2 and in Section 5.3.1, we conclude this subsection with pseudocode for all relevant algorithms introduced in it, specifying the query complexity (q-bounded or m-bounded, depending on the context) of each step. We start with the algorithm from Lemma 5.3.2.2 (1), which decides whether a given index d generalized cyclotomic mapping f of $\mathbb{F}_q$ is of special type I.

1 Compute the affine maps $A_i : x \mapsto \alpha_i x + \beta_i$ of $\mathbb{Z}/s\mathbb{Z}$ associated with f.
QC: $(d \log^{1+o(1)} q, d, 0, 0, 0)$.

2 For each $i \in \{0, 1, \ldots, d-1\}$, do the following.
QC: $(d \log^{1+o(1)} q, 0, 0, 0, 0)$.

2.1 If $a_i = 0_{\mathbb{F}_q}$, then skip to the next i.
QC: $(\log d, 0, 0, 0, 0)$.

2.2 If $\gcd(\alpha_i, s) > 1$, then output "false" and halt.
QC: $(\log^{1+o(1)} q, 0, 0, 0, 0)$.

3 Output "true" and halt.

Next, we list the steps of the algorithm from Lemma 5.3.2.2 (2), which computes, for a given index d generalized cyclotomic mapping f of $\mathbb{F}_q$ that is of special type I, a type-I tree register $\mathfrak{R} = ((\mathfrak{D}_n, S_n))_{n=0,1,\ldots,N}$ for f with $N \in O(d)$.

1 Compute $s = (q-1)/d$, the induced function $\bar{f}$ on $\{0, 1, \ldots, d\}$ and the layers Layer_h, for $h \in \{0, 1, \ldots, \bar{H} - 1, \infty\}$, of $\bar{f}$ with respect to iteration.
QC: $(d \log^2 d + d \log^{1+o(1)} q, d, 0, 0, 0)$.

2 Set $\mathcal{N} := \emptyset$.
QC: $(1, 0, 0, 0, 0)$.

3 For each $i \in \{0, 1, \ldots, d\}$, determine its (ordered) list of $\bar{f}$-pre-images and its $\bar{f}$-transient pre-images.
QC: $(d \log d, 0, 0, 0, 0)$.

4 For each $h = 0, 1, \ldots, \bar{H} - 1, \infty$, do the following.
QC: $(d^3 \log^2 d + d \log^{1+o(1)} q, 0, 0, 0, 0)$.

4.1 If $h = 0$ or $\bar{H} = 0$, then do the following.

4.1.1 Add 0 to $\mathcal{N}$ as a new (the first) element, set $\mathfrak{D}_0 := \emptyset$ and $S_0 := \text{Layer}_h$.
QC: $(d \log d, 0, 0, 0, 0)$.

4.1.2 If $\bar{H} = 0$, then output $\mathfrak{R} := ((\mathfrak{D}_0, S_0))$ and halt.
QC: $(d \log d, 0, 0, 0, 0)$.

4.2 Else do the following.

4.2.1 For each $i \in \mathrm{Layer}_h$, letting $j_1, j_2, \ldots, j_K$ denote its $\bar{f}$-transient $\bar{f}$-pre-images, do the following.
QC: $(|\mathrm{Layer}_h|d^2 \log^2 d + \delta_{d \in \mathrm{Layer}_h} d \log^{1+o(1)} q, 0, 0, 0, 0)$.

4.2.1.1 For each $t = 1, 2, \ldots, K$, do the following.
QC: $(d^2 \log^2 d, 0, 0, 0, 0)$.

4.2.1.1.1 Determine the unique number $\bar{n}_{j_t} \in \mathcal{N}$ such that $j_t \in S_{\bar{n}_{j_t}}$.
QC: $(d \log^2 d, 0, 0, 0, 0)$.

4.2.1.2 If $i < d$, then do the following.

4.2.1.2.1 Set

$$\mathfrak{D} := \{(n, m) : n \in \{\bar{n}_{j_t} : 1 \le t \le K\}, \\ m = |\{t \in \{1, \ldots, K\} : \bar{n}_{j_t} = n\}| > 0\}.$$

QC: $(d \log^2 d, 0, 0, 0, 0)$.

4.2.1.3 Else do the following.

4.2.1.3.1 Set

$$\mathfrak{D} := \{(n, sm) : n \in \{\bar{n}_{j_t} : 1 \le t \le K\}, \\ m = |\{t \in \{1, \ldots, K\} : \bar{n}_{j_t} = n\}| > 0\}$$

QC: $(d \log^2 d + d \log^{1+o(1)} q, 0, 0, 0, 0)$.

4.2.1.4 Check whether there is a (unique) $n \in \mathcal{N}$ such that $\mathfrak{D} = \mathfrak{D}_n$, and store this information (the truth value and n).
QC: $(d^2 \log d, 0, 0, 0, 0)$.

4.2.1.5 If $\mathfrak{D} = \mathfrak{D}_n$ for some $n \in \mathcal{N}$, then do the following.

4.2.1.5.1 Merge the sorted lists $\{i\}$ and S_n.
QC: $(d \log d, 0, 0, 0, 0)$.

4.2.1.6 Else do the following.

(a) Set $n' := \max \mathcal{N} + 1$, and add n' to $\mathcal{N}$ as a new element.
QC: $(\log d, 0, 0, 0, 0)$.

(b) Set $\mathfrak{D}_{n'} := \mathfrak{D}$, and initialize $S_{n'} := \{i\}$.
QC: $(d \log d, 0, 0, 0, 0)$.

5 Output $\mathfrak{R} := ((\mathfrak{D}_n, S_n))_{n=0,1,\ldots,\max \mathcal{N}}$ and halt.
QC: $(d^2 \log d + d \log q, 0, 0, 0, 0)$.

Next, we give the (simple) pseudocode for the algorithm from Lemma 5.3.2.2 (3), which checks whether a given index d generalized cyclotomic mapping f of $\mathbb{F}_q$ is of special type II.

1 Compute the induced function $\bar{f} : \{0, 1, \ldots, d\} \to \{0, 1, \ldots, d\}$.
QC: $(d \log^{1+o(1)} q, d, 0, 0, 0)$.

2 Sort $\mathrm{im}(\bar{f})$, and check whether it is equal to $\{0, 1, \ldots, d\}$.
QC: $(d \log^2 d, 0, 0, 0, 0)$.

The algorithm from Lemma 5.3.2.2 (4), for computing a type-II tree register for a given index d generalized cyclotomic mapping f of $\mathbb{F}_q$ that is of special type II, has the following pseudocode.

1 Compute the induced function $\bar{f}$ and the affine maps $A_i : x \mapsto \alpha_i x + \beta_i$.
QC: $(d \log^{1+o(1)} q, d, 0, 0, 0)$.

2 Compute a CRL-list $\bar{\mathcal{L}}$ of $\bar{f}$ and, in the process, store the cycles of $\bar{f}$.
QC: $(d \log^2 d, 0, 0, 0, 0)$.

3 Set $\mathcal{N} := \emptyset$.
QC: $(1, 0, 0, 0, 0)$.

4 For each $(i, \ell) \in \bar{\mathcal{L}}$, do the following.
QC: $(d^5 \log^2 d \,\mathrm{mpe}(q-1)^3 + d^5 \,\mathrm{mpe}(q-1)^3 \log q + d^2 \,\mathrm{mpe}(q-1) \log^{1+o(1)} q$, $0, 0, 0, 0)$.

4.1 For each $t = 0, 1, \ldots, \ell - 1$, do the following.
QC: $(\ell^2 \,\mathrm{mpe}(q-1) \log^{1+o(1)} q, 0, 0, 0, 0)$.

4.1.1 Set $\mathrm{prod}_{\mathrm{upper}} := \alpha_{i_{t-1}}$.
QC: $(\log q, 0, 0, 0, 0)$.

4.1.2 Set $\mathrm{prod}_{\mathrm{lower}} := 1$.
QC: $(1, 0, 0, 0, 0)$.

4.1.3 For each $k = 1, 2, \ldots$ (no explicit upper bound for this for-loop a priori, though by design it will stop at $k = \mathcal{H}_{i_t} + 1 \in O(\ell \,\mathrm{mpe}(q-1))$ – for the "big-O", see bound (3.2)), do the following.
QC: $(\ell \,\mathrm{mpe}(q-1) \log^{1+o(1)} q, 0, 0, 0, 0)$.

4.1.3.1 Compute

$$\mathrm{proc}_{i_t,k} := \frac{\gcd(\mathrm{prod}_{\mathrm{upper}}, s)}{\gcd(\mathrm{prod}_{\mathrm{lower}}, s)}.$$

QC: $(\log^{1+o(1)} q, 0, 0, 0, 0)$.

4.1.3.2 If $\mathrm{proc}_{i_t,k} = 1$, then do the following.

4.1.3.2.1 Set $\mathcal{H}_{i_t} := k - 1$.
QC: $(\log d + \log\log q, 0, 0, 0, 0)$.

4.1.3.2.2 Exit the loop for k, and skip to the next t.
QC: $(1, 0, 0, 0, 0)$.

4.1.3.3 Else do the following.

(a) Set $\mathrm{prod}_{\mathrm{lower}} := \mathrm{prod}_{\mathrm{upper}}$.
QC: $(\log q, 0, 0, 0, 0)$.

(b) Set $\mathrm{prod}_{\mathrm{upper}} := \mathrm{prod}_{\mathrm{upper}} \cdot \alpha_{i_{t-k-1}}$.
QC: $(\log^{1+o(1)} q, 0, 0, 0, 0)$.

4.2 Compute $H_i := \max\{\mathcal{H}_{i_t} : t = 0, 1, \ldots, \ell - 1\}$.
QC: $(\ell(\log d + \log\log q), 0, 0, 0, 0)$.

4.3 For $h = 0, 1, \ldots, H_i$, do the following.
QC: $(\ell^2 d^3 \log^2 d \,\mathrm{mpe}(q-1)^3 + \ell d^4 \,\mathrm{mpe}(q-1)^3 \log q, 0, 0, 0, 0)$.

4.3.1 If $h = 0$, then do the following.

4.3.1.1 If $\mathcal{N} = \emptyset$, then do the following.

4.3.1.1.1 Set $\mathfrak{D}_0 := \emptyset$ and $\mathrm{ht}_0 := 0$.
QC: $(1, 0, 0, 0, 0)$.

4.3.1.1.2 If $H_i > 0$, then set $S_{0,\mathrm{trans}} := \{i_0, i_1, \ldots, i_{\ell-1}\}$, sorted. Otherwise, set $S_{0,\mathrm{trans}} := \emptyset$.
QC: $(\ell \log^2 d, 0, 0, 0, 0)$.

4.3.1.1.3 Set $S_{0,\mathrm{per}} := \{i_t : \mathcal{H}_{i_t} = 0\}$, sorted.
QC: $(\ell \log^2 d, 0, 0, 0, 0)$.

4.3.1.2 Else do the following.

4.3.1.2.1 If $H_i > 0$, then sort $\{i_0, i_1, \ldots, i_{\ell-1}\}$ and merge it with $S_{0,\mathrm{trans}}$.
QC: $(\ell \log^2 d + d \log d, 0, 0, 0, 0)$.

4.3.1.2.2 Create a list of all indices i_t, for $t = 0, 1, \ldots, \ell - 1$, such that $\mathcal{H}_{i_t} = 0$, then sort it and merge it with $S_{0,\mathrm{per}}$.
QC: $(\ell \log^2 d + d \log d, 0, 0, 0, 0)$.

4.3.2 Else do the following.

4.3.2.1 For $t = 0, 1, \ldots, \ell - 1$, do the following.
QC: $(\ell^2 d^2 \log^2 d \,\mathrm{mpe}(q-1)^2 + \ell d^3 \,\mathrm{mpe}(q-1)^2 \log q, 0, 0, 0, 0)$.

4.3.2.1.1 If $h < H_i$, then set $h' := h$; otherwise, set $h' := \mathcal{H}_{i_t}$.
QC: $(\log d + \log\log q, 0, 0, 0, 0)$.

4.3.2.1.2 For $k = 0, 1, \ldots, h' - 1$, set

$$w_k := \begin{cases} \mathrm{proc}_{i_t,k+1} - \mathrm{proc}_{i_t,k+2}, & \text{if } h = H_i, \text{ or } h < H_i \text{ and } k < h' - 1, \\ \mathrm{proc}_{i_t,h}, & \text{if } h < H_i \text{ and } k = h' - 1. \end{cases}$$

QC: $(\ell \,\mathrm{mpe}(q-1) \log q, 0, 0, 0, 0)$.

4.3.2.1.3 For $k = 0, 1, \ldots, h' - 1$, find $\bar{n}_{i_t,k}$, the unique $n \in \mathcal{N}$ such that $i_t \in S_{n,\mathrm{trans}}$ and $\mathrm{ht}_n = k$.
QC: $(\ell d^2 \log^2 d \operatorname{mpe}(q-1)^2 + \ell \operatorname{mpe}(q-1) \log\log q, 0, 0, 0, 0)$.

4.3.2.1.4 Set $\mathfrak{D} := \{(\bar{n}_{i_t,k}, w_k) : k = 0, 1, \ldots, h' - 1\}$, sorted lexicographically.
QC: $(\ell \operatorname{mpe}(q-1)(\log \ell + \log \operatorname{mpe}(q-1)) \log q, 0, 0, 0, 0)$.

4.3.2.1.5 Check whether $\mathfrak{D} = \mathfrak{D}_n$ some (unique) $n \in \mathcal{N}$, and if so, store this information (the truth value and n).
QC: $(\ell d^2 \operatorname{mpe}(q-1)^2 \log q, 0, 0, 0, 0)$.

4.3.2.1.6 If $\mathfrak{D} = \mathfrak{D}_n$ for some $n \in \mathcal{N}$, then do the following.

4.3.2.2.6.1 If $h < H_i$, then merge the sorted lists $\{i_t\}$ and $S_{n,\mathrm{trans}}$. Otherwise, merge the sorted lists $\{i_t\}$ and $S_{n,\mathrm{per}}$.
QC: $(d \log d, 0, 0, 0, 0)$.

4.3.2.1.7 Else do the following.

(1) Set $n' := \max \mathcal{N} + 1$, $\mathfrak{D}_{n'} := \mathfrak{D}$, $\mathrm{ht}_{n'} := h'$, and add n' to $\mathcal{N}$ as a new element.
QC: $(d \operatorname{mpe}(q-1) \log q, 0, 0, 0, 0)$.

(2) Initialize $S_{n',\mathrm{trans}} := \emptyset$ and $S_{n',\mathrm{per}} := \emptyset$.
QC: $(\log d + \log\log q, 0, 0, 0, 0)$

(3) If $h < H_i$, then add i_t to $S_{n',\mathrm{trans}}$ as a new element. Otherwise, add i_t to $S_{n',\mathrm{per}}$ as a new element.
QC: $(\log d + \log\log q, 0, 0, 0, 0)$.

5 Compute $\mathfrak{H} := \max\{H_i : (i, \ell) \in \bar{\mathcal{L}}\}$.
QC: $(d(\log d + \log\log q), 0, 0, 0, 0)$.

6 For $n \in \mathcal{N}$, do the following.
QC: $(d^3 \log d \operatorname{mpe}(q-1) + d^2 \operatorname{mpe}(q-1) \log\log q, 0, 0, 0, 0)$.

6.1 Set $S_n := (\mathrm{ht}_n, S_{n,\mathrm{trans}}, S_{n,\mathrm{per}})$.
QC: $(d \log d + \log\log q, 0, 0, 0, 0)$.

7 Output $\mathfrak{R} := ((\mathfrak{D}_n, S_n))_{n \in \mathcal{N}}$ and halt.
QC: $(d^4 \operatorname{mpe}(q-1)^2 \log q, 0, 0, 0, 0)$.

Next, we give pseudocode for the algorithm from Proposition 5.3.2.3, which computes the cycle type $\mathrm{CT}(A_{|\mathrm{per}(A)})$ for a given affine map $A : x \mapsto ax + b$ of $\mathbb{Z}/m\mathbb{Z}$.

1 Factor m, compute $\mathrm{ord}_{p^{\nu_p(m)}}(a)$ for all primes $p \mid m$ with $p \nmid a$, factor $a - 1$, and if $\nu_2(m) \geq 2$, compute $\varepsilon \in \{0, 1\}$ and $e \in \{0, 1, \ldots, 2^{\nu_2(m)} - 2\}$ such that $a \equiv (-1)^\varepsilon 5^e \pmod{2^{\nu_2(m)}}$.
QC: $(\log m, 0, 1, 1, 0)$.

2 For each prime $p \mid m$ such that $p \nmid a$, do the following.
QC: $(\log^{2+o(1)} m, 0, \log m, 0, 0)$.

2.1 Compute $A_{(p)} := A \bmod p^{\nu_p(m)}$.
QC: $(\log^{1+o(1)} m, 0, 0, 0, 0)$.

2.2 Using [15, Tables 3 and 4], compute

$$\mathrm{CT}(A_{(p)}) = x_{\bar{l}_{p,1}}^{e_{p,1}} x_{\bar{l}_{p,2}}^{e_{p,2}} \cdots x_{\bar{l}_{p,K_p}}^{e_{p,K_p}} \quad (\text{all } e_{p,j} > 0)$$

with all cycle lengths $\bar{l}_{p,j}$ fully factored. This involves factoring $\mathrm{ord}_{p^{\nu_p(m)}}(a)$, a single power computation, and $O(\nu_p(m))$ instances of simpler arithmetic.
QC: $(\log^{2+o(1)} p^{\nu_p(m)} + \nu_p(m) \log^{1+o(1)} p^{\nu_p(m)}, 0, 1, 0, 0)$.

3 For each $\vec{j} = (j_p)_{p|m, p\nmid a} \in \prod_{p|m,p\nmid a} \{1, 2, \ldots, K_p\}$, do the following.
QC: $(\tau(m) \log^{2+o(1)} m, 0, 0, 0, 0)$.

3.1 Compute the Wei–Xu product of variable powers

$$\mathrm{WX}(\vec{j}) := \divideontimes_{p|m,p\nmid a} x_{\bar{l}_{p,j_p}}^{e_{p,j_p}} = x_{\mathrm{lcm}(\bar{l}_{p,j_p}:p|m,p\nmid a)}^{\prod_{p|m,p\nmid a}(e_{p,j_p}\bar{l}_{p,j_p})/\mathrm{lcm}(\bar{l}_{p,j_p}:p|m,p\nmid a)}.$$

QC: $(\log^{2+o(1)} m, 0, 0, 0, 0)$.

4 Compute and output

$$\mathrm{CT}(A) = \divideontimes_{p|m,p\nmid a} \mathrm{CT}(A_{(p)}) = \prod_{\vec{j}} \mathrm{WX}(\vec{j}),$$

then halt.
QC: $(\tau(m)^2 \log m, 0, 0, 0, 0)$.

The following is pseudocode for the algorithm from Lemma 5.3.2.7, serving to compute $\mathrm{minperl}(\vec{x})$ for given $\vec{x} \in \{0, 1, \ldots, N-1\}^n$, where each $n \in \{0, 1, \ldots, N-1\}$ is given with bit length l_{bit}.

1 Compute the binary representation of n
QC: $(n, 0, 0, 0, 0)$.

2 Factor $n = p_1^{v_1} \cdots p_K^{v_K}$ deterministically.
QC: $(n^{1/5+o(1)}, 0, 0, 0, 0)$.

3 For each $j = 1, 2, \ldots, K$, do the following.
QC: $(n \log n \log\log n (\log n + l_{\mathrm{bit}}))$.

3.1 Using binary search, find $v'_j = \nu_{p_j}(\mathrm{minperl}(\vec{x}))$ as the smallest

$$v \in \{0, 1, \ldots, v_j\}$$

such that $p_j^v \prod_{k\neq j} p_k^{v_k}$ is a period length of $\vec{x}$.
QC: $(n \log\log n (\log n + l_{\mathrm{bit}}))$.

4 Compute and output

$$\operatorname{minperl}(\vec{x}) = \prod_{j=1}^{K} p_j^{v'_j},$$

then halt.
QC: $(\log^{3+o(1)} n, 0, 0, 0, 0)$.

Next, we give pseudocode for Theorem 5.3.2.10, which is concerned with computing not only a tree register, but also an associated tree necklace list for a given index d generalized cyclotomic mapping f of $\mathbb{F}_q$ that is of special type I or II. Because the procedures for the two cases are analogous, we just give one algorithm that deals with both simultaneously.

1 Check whether f is of special type I and store this information. In the process, also compute and store $\bar{f}$ and the affine maps A_i for later use.
QC: $(d \log^{1+o(1)} q, d, 0, 0, 0)$.

2 If f is *not* of special type I, then check whether f is of special type II and store this information.
QC: $(d \log^2 d, 0, 0, 0, 0)$ because $\bar{f}$ has already been computed.

3 If f is neither of special type I nor II, then output "fail" and halt.
QC: $(1, 0, 0, 0, 0)$.

4 If f is of special type I, then do the following.

4.1 Compute a type-I tree register $\mathfrak{R} = ((\mathfrak{D}_n, S_n))_{n=0,1,\ldots,N}$ for f with $N \in O(d)$.
QC: $(d^3 \log^2 d + d \log^{1+o(1)} q, 0, 0, 0, 0)$ because $\bar{f}$ and the A_i have already been computed.

5 Else do the following.

5.1 Compute a type-II tree register $\mathfrak{R} = ((\mathfrak{D}_n, S_n))_{n=0,1,\ldots,N}$ for f with

$$N \in O(d^2 \operatorname{mpe}(q-1)),$$

where $S_n = (\mathrm{ht}_n, S_{n,\mathrm{trans}}, S_{n,\mathrm{per}})$.
QC: $(d^2 \operatorname{mpe}(q-1) \log^{1+o(1)} q + d^5 \operatorname{mpe}(q-1)^3 \log q + d^5 \log^2 d \operatorname{mpe}(q-1)^3, 0, 0, 0, 0)$ because $\bar{f}$ and the A_i have already been computed.

6 Compute a CRL-list $\bar{\mathfrak{L}}$ of $\bar{f}$ and, in the process, store the cycles of $\bar{f}$.
QC: $(d \log^2 d, 0, 0, 0, 0)$.

7 For each $(i, \ell) \in \bar{\mathfrak{L}}$, with associated $\bar{f}$-cycle $(i_0, i_1, \ldots, i_{\ell-1})$, do the following.
QC: $(d\tau(q-1) \log^{2+o(1)} q + d^2 \log^{1+o(1)} q + d\tau(q-1)^2 \log q + Nd^2 \log d + d^3 \log d\, l_{\mathrm{bit}}, 0, d \log q, d, 0)$.

7.1 If $i = d$, then do the following.

7.1.1 If f is of special type I, then do the following.

7.1.1.1 Set $\mathfrak{n}$ to be the unique $n \in \{0, 1, \ldots, N\}$ such that $d \in S_n$.
QC: $(d \log d, 0, 0, 0, 0)$.

7.1.2 Else do the following.

7.1.2.1 Set $\mathfrak{n} := 0$.
QC: $(1, 0, 0, 0, 0)$.

7.1.3 Set $\mathfrak{N}_d := \{([\mathfrak{n}], 1, 1)\}$, $\vec{\mathfrak{n}}'_d := [\mathfrak{n}]$ and $\mathfrak{N}'_d := \{(1, 1)\}$, then skip to the next pair (i, ℓ).
QC: $(\log d, 0, 0, 0, 0)$.

7.2 Else do the following.

7.2.1 Compute $\mathcal{A}_i := A_{i_0} A_{i_1} \cdots A_{i_{\ell-1}}$.
QC: $(d \log^{1+o(1)} q, 0, 0, 0, 0)$.

7.2.2 Compute $\mathrm{CT}((\mathcal{A}_i)_{|\mathrm{per}(\mathcal{A}_i)}) = x_1^{e_{i,1}} x_2^{e_{i,2}} \cdots x_s^{e_{i,s}}$.
QC: $(\tau(q-1) \log^{2+o(1)} q + \tau(q-1)^2 \log q, 0, \log q, 1, 0)$.

7.2.3 If f is of special type I, then do the following.

7.2.3.1 For each $n = 0, 1, \ldots, N$, do the following.
QC: $(d^2 \log d, 0, 0, 0, 0)$.

7.2.3.1.1 For each $t = 0, 1, \ldots, \ell - 1$, do the following.
QC: $(|S_n| d \log d, 0, 0, 0, 0)$.

7.2.3.1.1.1 If $i_t \in S_n$, then set $\mathfrak{n}_{i_t} := n$.
QC: $(|S_n| \log d, 0, 0, 0, 0)$.

7.2.4 Else do the following.

7.2.4.1 For each $n = 0, 1, \ldots, N$, do the following.
QC: $(d^3 \log d \ \mathrm{mpe}(q-1) + d\, l_{\mathrm{bit}}, 0, 0, 0, 0)$.

7.2.4.1.1 For each $t = 0, 1, \ldots, \ell - 1$, do the following.
QC: $(\max\{1, |S_{n,\mathrm{per}}|\} d \log d + |S_{n,\mathrm{per}} \cap \{i_0, \ldots, i_{\ell-1}\}| l_{\mathrm{bit}}, 0, 0, 0, 0)$.

7.2.4.1.1.1 If $i_t \in S_{n,\mathrm{per}}$, then set $\mathfrak{n}_{i_t} := n$.
QC: $(\max\{1, |S_{n,\mathrm{per}}|\} \log d + l_{\mathrm{bit}}, 0, 0, 0, 0)$.

7.2.5 Set $\vec{\mathfrak{n}}_i := (\mathfrak{n}_{i_0}, \mathfrak{n}_{i_1}, \ldots, \mathfrak{n}_{i_{\ell-1}})$.
QC: $(d\, l_{\mathrm{bit}}, 0, 0, 0, 0)$.

7.2.6 Overwrite $\vec{\mathfrak{n}}_i$ with the lexicographically smallest sequence in the same cyclic equivalence class.
QC: $(d^2 \log d\, l_{\mathrm{bit}}, 0, 0, 0, 0)$.

7.2.7 Compute $\mathrm{minperl}(\vec{\mathfrak{n}}_i)$.
QC: $(d \log d \log\log d (\log d + l_{\mathrm{bit}}), 0, 0, 0, 0)$.

7.2.8 Set

- $\vec{\mathfrak{n}}'_i := [\mathfrak{n}_{i_0}, \mathfrak{n}_{i_1}, \ldots, \mathfrak{n}_{i_{\text{minperl}(\vec{\mathfrak{n}}_i)-1}}]$,
- $\mathfrak{N}'_i := \{(\ell \cdot l', e_{i,l'}) : l' \in \{1, 2, \ldots, s\}, e_{i,l'} > 0\}$, and
- $\mathfrak{N}_i := \{\vec{\mathfrak{n}}'_i\} \times \mathfrak{N}'_i$.

QC: $(d l_{\text{bit}} + \tau(q-1)\log^{1+o(1)} q, 0, 0, 0, 0)$.

8 Set $\mathfrak{M} := \mathfrak{N} := \emptyset$.
QC: $(1, 0, 0, 0, 0)$.

9 Set $\vec{\mathfrak{n}}' := \{(\vec{\mathfrak{n}}'_i, i, \ell) : (i, \ell) \in \overline{\mathcal{L}}\}$, and sort it lexicographically.
QC: $(d^2 \log d l_{\text{bit}}, 0, 0, 0, 0)$.

10 For each $(\vec{\mathfrak{n}}'_i, i, \ell) \in \vec{\mathfrak{n}}'$, do the following.
QC: $(d^3\tau(q-1)^2 \log q, 0, 0, 0, 0)$.

10.1 Check whether $\vec{\mathfrak{n}}'_i \in \mathfrak{M}$, and if so, skip to the next triple $(\vec{\mathfrak{n}}'_i, i, \ell)$.
QC: $(d^2 l_{\text{bit}}, 0, 0, 0, 0)$.

10.2 Add $\vec{\mathfrak{n}}'_i$ to $\mathfrak{M}$ as a new element.
QC: $(d l_{\text{bit}}, 0, 0, 0, 0)$.

10.3 Set $\mathfrak{V} := \mathfrak{N}'_i$.
QC: $(\tau(q-1)\log q, 0, 0, 0, 0)$.

10.4 For each $(\vec{\mathfrak{n}}'_j, j, \ell') \in \vec{\mathfrak{n}}'$ that comes after $(\vec{\mathfrak{n}}'_i, i, \ell)$, do the following.
QC: $(d^2 l_{\text{bit}} + d^2\tau(q-1)^2 \log q, 0, 0, 0, 0)$.

10.4.1 Check whether $\vec{\mathfrak{n}}'_i = \vec{\mathfrak{n}}'_j$, and if not, skip to the next triple $(\vec{\mathfrak{n}}'_j, j, \ell')$.
QC: $(d l_{\text{bit}}, 0, 0, 0, 0)$.

10.4.2 For each $(l, k) \in \mathfrak{N}'_j$, do the following.
QC: $(d\tau(q-1)^2 \log q, 0, 0, 0, 0)$.

10.4.2.1 Check whether l occurs as the first entry of some pair $(l, k') \in \mathfrak{V}$, and store this information.
QC: $(d\tau(q-1)\log q, 0, 0, 0, 0)$.

10.4.2.2 If $(l, k') \in \mathfrak{V}$ for some k', then do the following.

10.4.2.2.1 Replace the unique element of $\mathfrak{V}$ of the form (l, k') by $(l, k + k')$.
QC: $(\log q, 0, 0, 0, 0)$.

10.4.2.3 Else do the following.

10.4.2.3.1 Add (l, k) to $\mathfrak{V}$ as a new element.
QC: $(\log q, 0, 0, 0, 0)$.

10.5 For each $(l, k) \in \mathfrak{V}$, do the following.
QC: $(d\tau(q-1)(d l_{\text{bit}} + \log q), 0, 0, 0, 0)$.

10.5.1 Add $([\vec{\mathfrak{n}}'_i], l, k)$ to $\mathfrak{N}$ as a new element.
QC: $(d l_{\text{bit}} + \log q, 0, 0, 0, 0)$.

11 Output $\mathfrak{R}$ and $\mathfrak{N}$, and halt.
QC: $(N^2 \log q + d\tau(q-1)(d\, l_{\text{bit}} + \log q), 0, 0, 0, 0)$.

The following is pseudocode for the algorithm from Lemma 5.3.2.12. For given recursive tree description lists $\vec{\mathfrak{D}} = (\mathfrak{D}_n)_{n=0,1,\ldots,N}$ and $\vec{\mathfrak{D}}' = (\mathfrak{D}'_n)_{n=0,1,\ldots,N'}$ satisfying the assumptions of Lemma 5.3.2.12, this algorithm computes a synchronization $(\vec{\mathfrak{D}}^+, \mathrm{i})$ of $\vec{\mathfrak{D}}$ and $\vec{\mathfrak{D}}'$.

1 Set $\mathfrak{R} := \{0, 1, \ldots, N\}$, and for $k \in \mathfrak{R}$, set $\mathfrak{D}_k^+ := \mathfrak{D}_k$. Moreover, let i be the empty function $\emptyset$.
QC: $(N^2(l_{\text{bit}} + m), 0, 0, 0, 0)$.

2 For each $n = 0, 1, \ldots, N'$, do the following.
QC: $((NN' \min\{N, N'\} + (N')^2 l'_{\text{bit}})(l_{\text{bit}} + l'_{\text{bit}} + m), 0, 0, 0, 0)$.

2.1 If $n = 0$, then do the following.

2.1.1 Set $\mathrm{i}(0) := 0$.
QC: $(1, 0, 0, 0, 0)$.

2.2 Else do the following.

2.2.1 Let $\mathfrak{D}$ be the set of pairs obtained from $\mathfrak{D}'_n$ through replacing each first entry k of each pair in $\mathfrak{D}'_n$ by $\mathrm{i}(k)$ (one may simply overwrite the corresponding entries of $\mathfrak{D}'_n$, so one does not need to handle the second entries of bit length in $O(m)$).
QC: $(N'(l_{\text{bit}} + l'_{\text{bit}}), 0, 0, 0, 0)$.

2.2.2 Sort $\mathfrak{D}$.
QC: $(N' l'_{\text{bit}}(l_{\text{bit}} + l'_{\text{bit}} + m))$.

2.2.3 Check whether $\mathfrak{D} = \mathfrak{D}_k^+$ for some (unique) $k \in \{0, 1, \ldots, N\}$, and store this information (the truth value and k).
QC: $(N \min\{N, N'\}(l_{\text{bit}} + l'_{\text{bit}} + m), 0, 0, 0, 0)$.

2.2.4 If $\mathfrak{D} = \mathfrak{D}_k^+$ for some $k \in \{0, 1, \ldots, N\}$, then do the following.

2.2.4.1 Set $\mathrm{i}(n) := k$.
QC: $(l_{\text{bit}} + l'_{\text{bit}}, 0, 0, 0, 0)$.

2.2.5 Else do the following.

2.2.5.1 Set $n' := \max \mathcal{N} + 1$, add n' to $\mathcal{N}$ as a new element, set $\mathfrak{D}_{n'}^+ := \mathfrak{D}$ and $\mathrm{i}(n) := n'$.
QC: $(\min\{N, N'\}(l_{\text{bit}} + l'_{\text{bit}} + m), 0, 0, 0, 0)$.

3 Set $\vec{\mathfrak{D}}^+ := (\mathfrak{D}_n^+)_{n \in \mathcal{N}}$, output $(\vec{\mathfrak{D}}^+, \mathrm{i})$ and halt.
QC: $((N + N') \max\{N, N'\}(l_{\text{bit}} + l'_{\text{bit}} + m), 0, 0, 0, 0)$.

Finally, we provide pseudocode for the algorithm from Corollary 5.3.2.13. For given generalized cyclotomic mappings f_1 and f_2 of $\mathbb{F}_q$, of index d_1 and d_2, respectively, such that each f_j is of special type I or II (not necessarily both of the same

special type), this algorithm decides whether $\Gamma_{f_1} \cong \Gamma_{f_2}$. Throughout this discussion, we have $d := \max\{d_1, d_2\}$.

1 For $j = 1, 2$, do the following.
QC:
- $(d \log^{1+o(1)} q, d, 0, 0, 0)$ if f_1 and f_2 are both of special type I;
- $(d \log^2 d + d \log^{1+o(1)} q, d, 0, 0, 0)$ otherwise.

1.1 Check whether f_j is of special type I, and store this information as well as the induced function $\overline{f_j}$ and the affine maps on $\mathbb{Z}/((q-1)/d_j)\mathbb{Z}$ associated with f_j.
QC: $(d \log^{1+o(1)} q, d, 0, 0, 0)$.

1.2 If f_j is *not* of special type I, then check whether f_j is of special type II, and store this information.
QC: $(d \log^2 d, 0, 0, 0, 0)$, because $\overline{f_j}$ and the affine maps have already been computed.

1.3 If f_j is neither of special type I nor II, then output "fail" and halt.
QC: $(1, 0, 0, 0, 0)$.

2 For $j = 1, 2$, do the following.
QC:
- $(d^3 \log^2 d + d^3 \log d\,\tau(q-1) + d^3\tau(q-1)^2 \log q + d^2 \log^{1+o(1)} q + d\,\tau(q-1) \log^{2+o(1)} q, d, d \log q, d, 0)$ if f_1 and f_2 are both of special type I;
- $(d^5 \log^2 d\,\mathrm{mpe}(q-1) + (d^5\,\mathrm{mpe}(q-1)^3 + d^3\tau(q-1)^2) \log q + d^2\,\mathrm{mpe}(q-1) \log^{1+o(1)} q + d\,\tau(q-1) \log^{2+o(1)} q, d, d \log q, d, 0)$ otherwise.

2.1 If f_j is of special type I, then do the following.

2.1.1 Compute a type-I tree register $\Re_j$ of f_j, and the tree necklace list $\Re_j$ for f_j relative to $\Re_j$.
QC: $(d^3 \log^2 d + d^3\tau(q-1)^2 \log q + d^2 \log^{1+o(1)} q + d\,\tau(q-1) \log^{2+o(1)} q, d, d \log q, d, 0)$.

2.2 Else do the following.

2.2.1 Compute a type-II tree register $\Re_j$ of f_j, and the tree necklace list $\Re_j$ for f_j relative to $\Re_j$.
QC: $(d^5 \log^2 d\,\mathrm{mpe}(q-1)^3 + (d^5\,\mathrm{mpe}(q-1)^3 + d^3\tau(q-1)^2) \log q + d^2\,\mathrm{mpe}(q-1) \log^{1+o(1)} q + d\,\tau(q-1) \log^{2+o(1)} q, d, d \log q, d, 0)$.

3 For $j = 1, 2$, let $\vec{\mathfrak{D}}^{(j)} = (\mathfrak{D}_n^{(j)})_{n=0,1,\dots,N_j}$ be the underlying recursive tree description list of $\mathfrak{R}_j$. Compute a synchronization $(\vec{\mathfrak{D}}^+, \mathfrak{i})$ of $\vec{\mathfrak{D}}^{(1)}$ and $\vec{\mathfrak{D}}^{(2)}$.
QC:
- $(d^3 \log q, 0, 0, 0, 0)$ if f_1 and f_2 are both of special type I;
- $(d^6 \operatorname{mpe}(q-1)^3 \log q + d^4 \operatorname{mpe}(q-1)^2 \log^{1+o(1)} q, 0, 0, 0, 0)$ otherwise.

4 Create a modified version $\mathfrak{R}_2'$ of $\mathfrak{R}_2$ by replacing each entry $\mathfrak{n}$ of each first entry $[\vec{\mathfrak{n}}]$ of an element $([\vec{\mathfrak{n}}], l, \mathfrak{m}) \in \mathfrak{R}_2$ by $\mathfrak{i}(\mathfrak{n})$, then overwriting each of the resulting $O(d)$ distinct first entries of triples in the list with the lexicographically minimal number sequence in the same cyclic equivalence class, and finally sorting $\mathfrak{R}_2'$ lexicographically by merging the $O(d)$ distinct segments corresponding to the same first entry of triples in $\mathfrak{R}_2'$.
QC: $(d^3 \tau(q-1) \log q, 0, 0, 0, 0)$.

5 Check whether $\mathfrak{R}_2' = \mathfrak{R}_1$, output the corresponding truth value, and halt.
QC: $(d^2 \tau(q-1) \log q, 0, 0, 0, 0)$.

5.3.3 Short-term block behavior and the special case where all cycles are short

Let f be an index d generalized cyclotomic mapping of $\mathbb{F}_q$. For an f-periodic $x \in \mathbb{F}_q$ and $t \in \mathbb{Z}$, we set $x^{(t)} := (f_{|\mathrm{per}(f)})^t(x)$, and we recall the notation $i_t := (\bar{f}_{|\mathrm{per}(\bar{f})})^t(i)$ for $\bar{f}$-periodic $i \in \{0, 1, \dots, d-1\}$ and $t \in \mathbb{Z}$, as well as $i' := i_{-1}$.

In Section 3.3, for each $i \in \{0, 1, \dots, d-1\}$, we constructed an arithmetic partition $\mathcal{P}_i$ of C_i such that for $x \in C_i$, the isomorphism type of $\mathrm{Tree}_{\Gamma_f}(x)$ only depends on the $\mathcal{P}_i$-block in which x is contained. Here, we refine this construction. We recall that $\mathcal{P}_i = \mathcal{Q}_{i,H_i}$, where H_i is the maximum tree height in Γ_{per} (the induced subgraph of Γ_f on $\bigcup_{j \in \mathrm{per}(\bar{f})} C_j$) above an f-periodic point in $\bigcup_{t \in \mathbb{Z}} C_{i_t}$. The arithmetic partition $\mathcal{Q}_{i,h}$ of C_i is defined for all $h \in \mathbb{N}_0$ (even though we only considered it for $h \in \{0, 1, \dots, H_i\}$ in Section 3.3) and satisfies

$$\mathcal{Q}_{i,h+1} = \mathcal{R}_i \wedge \mathfrak{P}'(\mathcal{Q}_{i',h}, A_{i'}). \tag{5.7}$$

This formula is key to our construction. Indeed, the refined arithmetic partition of C_i which we consider here is simply $\mathcal{Q}_{i,H_i+L-1}$ for some $L \in \mathbb{N}^+$, as opposed to $\mathcal{P}_i = \mathcal{Q}_{i,H_i}$. While the blocks of $\mathcal{P}_i$ control the isomorphism types of rooted trees in Γ_f above (periodic) vertices in C_i, the following more general statement holds for $\mathcal{Q}_{i,H_i+L-1}$.

Lemma 5.3.3.1. *Let $x \in C_i$ be f-periodic, and let $L \in \mathbb{N}_0$. The $\mathcal{Q}_{i,H_i+L-1}$-block in which x is contained uniquely determines the (length L) sequence*

$$(\mathrm{Tree}_{\Gamma_f}(x^{(t)}))_{t=0,-1,\dots,-L+1}$$

of rooted tree isomorphism types.

Proof. We can write the $\mathcal{Q}_{i,H_i+L-1}$-block of x as $\mathcal{B}(\mathcal{Q}_{i,H_i+L-1}, \diamond_{t=0}^{H_i+L-1}\vec{o'_t}\diamond\vec{\xi}_{i,H_i})$, where $\vec{o'_t} \in \{\emptyset, \neg\}^{n_{i-t}}$. Noting that

$$\mathcal{B}(\mathcal{Q}_{i,H_i+L-1}, \diamond_{t=0}^{H_i+L-1}\vec{o'_t} \diamond \vec{\xi}_{i,H_i}) \subseteq \mathcal{B}(\mathcal{P}_i, \diamond_{t=0}^{H_i}\vec{o'_t} \diamond \vec{\xi}_{i,H_i}),$$

we see that $\mathrm{Tree}_{\Gamma_f}(x) = \mathrm{Tree}_i(\mathcal{P}_i, \diamond_{t=0}^{H_i}\vec{o'_t} \diamond \vec{\xi}_{i,H_i})$ is uniquely determined.

By formula (5.7), we know that for each block $\mathcal{B}(\mathcal{Q}_{i',H_i+L-2}, \diamond_{t=0}^{H_i+L-2}\vec{o_t} \diamond \vec{\xi})$ of $\mathcal{Q}_{i',H_i+L-2}$, the number of f-pre-images of x in that block is the constant

$$\sigma_{\mathcal{Q}_{i',H_i+L-2},A_{i'}}(\diamond_{t=0}^{H_i+L-2}\vec{o_t} \diamond \vec{\xi}, \diamond_{t=1}^{H_i+L-1}\vec{o'_t} \diamond \vec{\xi}_{i,H_i}). \tag{5.8}$$

Now we assume that $\vec{\xi} = \vec{\xi}_{i',H_i}$ (which is actually the same as $\vec{\xi}_{i,H_i}$). The union of all blocks of $\mathcal{Q}_{i',H_i+L-2}$ of the form $\mathcal{B}(\mathcal{Q}_{i',H_i+L-2}, \diamond_{t=0}^{H_i+L-2}\vec{o_t} \diamond \vec{\xi}_{i',H_i})$ (where $\vec{o_t}$ ranges over $\{\emptyset, \neg\}^{n_{i-t-1}}$ for each $t \in \{0, 1, \ldots, H_i + L - 2\}$) is just the subset of $C_{i'}$ consisting of all f-periodic points in it. Since x has precisely one f-periodic pre-image (which lies in $C_{i'}$), it follows that the value of (5.8) for $\vec{\xi} = \vec{\xi}_{i',H_i}$ is 0 for all $\diamond_{t=0}^{H_i+L-2}\vec{o_t} \in \{\emptyset, \neg\}^{n_{i-1}+n_{i-2}+\cdots+n_{i-H_i-L+1}}$ except for one, for which the constant (5.8) has value 1. If $\diamond_{t=0}^{H_i+L-2}\vec{o_t}$ is that unique logical sign tuple, then the unique f-periodic pre-image $x^{(-1)}$ of $x \in \mathcal{B}(\mathcal{Q}_{i,H_i+L-1}, \diamond_{t=0}^{H_i+L-1}\vec{o'_t} \diamond \vec{\xi}_{i,H_i})$ always lies in $\mathcal{B}(\mathcal{Q}_{i',H_i+L-2}, \diamond_{t=0}^{H_i+L-2}\vec{o_t} \diamond \vec{\xi}_{i',H_i})$, and so

$$\mathrm{Tree}_{\Gamma_f}(x^{(-1)}) \cong \mathrm{Tree}_{i'}(\mathcal{P}_{i'}, \diamond_{t=0}^{H_i}\vec{o_t} \diamond \vec{\xi}_{i',H_i})$$

is also uniquely determined. Continuing this process inductively, we get the statement of the lemma. ∎

For the purposes of our later complexity analysis, we need a more explicit version of Lemma 5.3.3.1. To each $\bar{f}$-periodic $i \in \{0, 1, \ldots, d-1\}$ and each $L \in \mathbb{N}^+$, we associate the set

$$\mathcal{O}_{i,L} := \{\diamond_{t=0}^{H_i+L-1}\vec{o'_t} \in \{\emptyset, \neg\}^{n_{i_0}+n_{i_{-1}}+\cdots+n_{i_{-H_i-L+1}}} : \mathcal{B}(\mathcal{Q}_{i,H_i+L-1}, \diamond_{t=0}^{H_i+L-1}\vec{o'_t} \diamond \vec{\xi}_{i,H_i}) \neq \emptyset\}$$

of logical sign tuples that correspond to a non-empty block of $\mathcal{Q}_{i,H_i+L-1}$ consisting of f-periodic points. The proof of Lemma 5.3.3.1 shows that as long as $L \geq 2$, we may implicitly define a (surjective) function $\mathfrak{u}_{i,L} : \mathcal{O}_{i,L} \to \mathcal{O}_{i',L-1}$ via

$$\sigma_{\mathcal{Q}_{i',H_i+L-2},A_{i'}}(\mathfrak{u}_{i,L}(\diamond_{t=0}^{H_i+L-1}\vec{o'_t}) \diamond \vec{\xi}_{i',H_i}, \diamond_{t=1}^{H_i+L-1}\vec{o'_t} \diamond \vec{\xi}_{i,H_i}) = 1.$$

Then for each (f-periodic) $x \in \mathcal{B}(\mathcal{Q}_{i,H_i+L-1}, \diamond_{t=0}^{H_i+L-1}\vec{o'_t} \diamond \vec{\xi}_{i,H_i})$, the unique f-periodic pre-image $x^{(-1)}$ of x, which lies in $C_{i'}$, is contained in

$$\mathcal{B}(\mathcal{Q}_{i',H_i+L-2}, \mathfrak{u}_{i,L}(\diamond_{t=0}^{H_i+L-1}\vec{o'_t}) \diamond \vec{\xi}_{i',H_i}).$$

Denoting by $\mathrm{proj}_{i,L}$ the projection

$$\mathcal{O}_{i,L} \to \{\emptyset, \neg\}^{n_{i_0}+n_{i_{-1}}+\cdots+n_{i_{-H_i}}}, \quad \diamond_{t=0}^{H_i+L-1}\vec{o}'_t \mapsto \diamond_{t=0}^{H_i}\vec{o}'_t,$$

we therefore have the following more explicit version of Lemma 5.3.3.1.

Lemma 5.3.3.2. *Let $L \in \mathbb{N}^+$, and let $x \in C_i$ be f-periodic, say contained in*

$$\mathcal{B}(\mathcal{Q}_{i,H_i+L-1}, \diamond_{t=0}^{H_i+L-1}\vec{o}'_t).$$

Then for each $k = 0, -1, \ldots, -L+1$, we have

$$\mathrm{Tree}_{\Gamma_f}(x^{(k)}) \cong$$
$$\mathrm{Tree}_{i_k}(\mathcal{P}_{i_k}, (\mathrm{proj}_{i_k,L+k} \circ \mathfrak{u}_{i_{k+1},L+k+1} \circ \mathfrak{u}_{i_{k+2},L+k+2} \circ \cdots \circ \mathfrak{u}_{i_0,L})(\diamond_{t=0}^{H_i+L-1}\vec{o}'_t)).$$

We note that the composition of functions of the form $\mathfrak{u}_{i_t,L+t}$ in the formula in Lemma 5.3.3.2 is empty if $k = 0$ (index-wise, it is supposed to ascend from 1 to 0, which is nonsensical). Specifically, Lemma 5.3.3.2 for $k = 0$ states that

$$\mathrm{Tree}_{\Gamma_f}(x) = \mathrm{Tree}_{\Gamma_f}(x^{(0)}) = \mathrm{Tree}_{i_0}(\mathcal{P}_{i_0}, \mathrm{proj}_{i_0,L}(\diamond_{t=0}^{H_i+L-1}\vec{o}'_t)),$$

for $k = -1$, it states that

$$\mathrm{Tree}_{\Gamma_f}(x^{(-1)}) = \mathrm{Tree}_{i_{-1}}(\mathcal{P}_{i_{-1}}, \mathrm{proj}_{i_{-1},L-1} \circ \mathfrak{u}_{i_0,L}(\diamond_{t=0}^{H_i+L-1}\vec{o}'_t)),$$

and so on.

In what follows, let us assume that all cycle lengths of f are at most L. We consider an $\bar{f}$-periodic index i of cycle length ℓ. By Lemma 5.3.3.2, for each $x \in C_i$, the block of $\mathcal{Q}_{i,H_i+L-1}$ in which x is contained together with the precise f-cycle length of x completely determines the digraph isomorphism type of the connected component of Γ_f containing x. By adding suitable s-congruences to the spanning congruences of $\mathcal{Q}_{i,H_i+L-1}$, we can construct a finer arithmetic partition, denoted by $\mathcal{W}_{i,L}$ below, each block of which consists of points of a common f-cycle length. Hence, for each f-periodic point $x \in C_i$, the digraph isomorphism type of the connected component of Γ_f containing x is completely determined by the $\mathcal{W}_{i,L}$-block containing x.

Let us discuss the details of how to construct $\mathcal{W}_{i,L}$. We recall from Section 3.3 that for each $l \in \mathbb{N}^+$, the restriction of f^l to $C_{i_{-l}}$, which maps to C_i, is represented by the affine map $\mathcal{A}_{i,l} : x \mapsto \bar{\alpha}_{i,l}x + \bar{\beta}_{i,l}$ (formulas for $\bar{\alpha}_{i,l}$ and $\bar{\beta}_{i,l}$ are given in the first bullet point after Proposition 3.3.3). Therefore, a point $x \in C_i$, viewed as an element of $\mathbb{Z}/s\mathbb{Z}$, is a fixed point of f^l if and only if ℓ divides l (so that $i_{-l} = i$) and $\bar{\alpha}_{i,l}x + \bar{\beta}_{i,l} \equiv x \pmod{s}$. This congruence is solvable if and only if $\gcd(s, \bar{\alpha}_{i,l} - 1) \mid \bar{\beta}_{i,l}$, in which case it is equivalent to the s-congruence

$$x \equiv -\frac{\bar{\beta}_{i,l}}{\gcd(s, \bar{\alpha}_{i,l}-1)} \cdot \mathrm{inv}_{\frac{s}{\gcd(s,\bar{\alpha}_{i,l}-1)}}\left(\frac{\bar{\alpha}_{i,l}-1}{\gcd(s, \bar{\alpha}_{i,l}-1)}\right) \left(\mathrm{mod}\ \frac{s}{\gcd(s, \bar{\alpha}_{i,l}-1)}\right),$$

which we henceforth denote by $\eta_{i,l}(x)$. We observe that $\eta_{i,l}(x)$ is only well defined when $\gcd(s, \overline{\alpha}_{i,l} - 1) \mid \overline{\beta}_{i,l}$. Let us set

$$\mathfrak{C}_{i,L} := \{l \in \{1, 2, \ldots, L\} : \ell \mid l \text{ and } \gcd(s, \overline{\alpha}_{i,l} - 1) \mid \overline{\beta}_{i,l}\}$$

and define

$$\mathcal{V}_{i,L} := \mathfrak{P}(\eta_{i,l}(x) : l \in \mathfrak{C}_{i,L}) \text{ and } \mathcal{W}_{i,L} := \mathcal{Q}_{i,H_i+L-1} \wedge \mathcal{V}_{i,L}.$$

Viewing $\mathcal{V}_{i,L}$ as an arithmetic partition of C_i, we claim that its blocks are just those subsets of C_i that consist of all points of any given f-cycle length. Indeed, let $l \in \{1, 2, \ldots, L\}$. If $l \notin \mathfrak{C}_{i,L}$, then f^l has no fixed points in C_i and, in particular, f has no points of cycle length l in C_i. On the other hand, if $l \in \mathfrak{C}_{i,L}$, then the points $x \in C_i$ of f-cycle length exactly l (if any) are just those that satisfy the congruence $\eta_{i,l'}(x)$ for precisely those $l' \in \mathfrak{C}_{i,L}$ that are multiples of l. In other words, if for $l' \in \mathfrak{C}_{i,L}$ we set

$$\nu_{l,l'} := \begin{cases} \emptyset, & \text{if } l \mid l', \\ \neg, & \text{otherwise,} \end{cases}$$

and set $\vec{\nu}_{i,L,l} := (\nu_{l,l'})_{l' \in \mathfrak{C}_{i,L}}$, then the set $\mathcal{B}(\mathcal{V}_{i,L}, \vec{\nu}_{i,L,l})$ (which may be empty) consists precisely of those $x \in C_i$ that are of f-cycle length l. In summary, we obtain the following result.

Proposition 5.3.3.3. *Let $L \in \mathbb{N}^+$ be such that all cycle lengths of f are at most L, and let $i \in \{0, 1, \ldots, d-1\}$ be $\bar{f}$-periodic. We view $\mathcal{V}_{i,L}$ and $\mathcal{W}_{i,L}$ as arithmetic partitions of C_i. Then the following hold.*

(1) *Each block of $\mathcal{V}_{i,L}$ is of one of the forms $\mathcal{B}(\mathcal{V}_{i,L}, (\neg, \neg, \ldots, \neg))$, respectively, $\mathcal{B}(\mathcal{V}_{i,L}, \vec{\nu}_{i,L,l})$ for some $l \in \mathfrak{C}_{i,L}$, and it consists precisely of the f-transient points in C_i, respectively of those f-periodic points in C_i that have f-cycle length precisely l.*

(2) *Each block of $\mathcal{W}_{i,L}$ consists either entirely of f-periodic or entirely of f-transient points. Moreover, each block of $\mathcal{W}_{i,L}$ whose elements are f-periodic is of the form*

$$\mathcal{B}(\mathcal{W}_{i,L}, \diamond_{t=0}^{H_i+L-1} \vec{o}'_t \diamond \vec{\xi}_{i,H_i} \diamond \vec{\nu}_{i,L,l})$$

for some $\vec{o}'_t \in \{\emptyset, \neg\}^{n_{i-t}}$ and $l \in \mathfrak{C}_{i,L}$, in which case for any given point x in that block, the digraph isomorphism type of the connected component of Γ_f containing x is represented by the cyclic sequence

$$[\mathrm{Tree}_{i_k}(\mathcal{P}_{i_k}, (\mathrm{proj}_{i_k,L+k} \circ \mathfrak{u}_{i_{k+1},L+k+1} \circ \mathfrak{u}_{i_{k+2},L+k+2} \circ \cdots \circ \mathfrak{u}_{i_0,L})(\diamond_{t=0}^{H_i+L} \vec{o}'_t))]_{k=-l+1,-l+2,\ldots,0}$$

of rooted tree isomorphism types.

Proposition 5.3.3.3 is the basis for proving the following theorem.

Theorem 5.3.3.4. *Let f be an index d generalized cyclotomic mapping of $\mathbb{F}_q$. Moreover, let $L \in \mathbb{N}^+$ with $L \leq q-1$ be such that all cycle lengths of f are at most L. Then, within q-bounded query complexity*

$$(8^{d^2 \operatorname{mpe}(q-1)+dL} 2^{d \operatorname{mpe}(q-1)} (d^3 L \operatorname{mpe}(q-1) + d^2 L^2) \log^{1+o(1)} q, d, 0, 0, 0),$$

and thus within q-bounded Las Vegas dual complexity

$$(8^{d^2 \operatorname{mpe}(q-1)+dL} 2^{d \operatorname{mpe}(q-1)} (d^3 L \operatorname{mpe}(q-1) + d^2 L^2) \log^{1+o(1)} q + d \log^{3+o(1)} q, d \log^{3+o(1)} q, d \log q),$$

one can compute

- *a recursive tree description list*

$$\vec{\mathfrak{D}} = (\mathfrak{D}_n)_{n=0,1,\ldots,N}$$

  with $N \in O(d 2^{d^2 \operatorname{mpe}(q-1)+d})$, whose associated sequence of rooted tree isomorphism types is denoted by $\vec{\mathfrak{T}} = (\mathfrak{T}_n)_{n=0,1,\ldots,N}$, such that indices $n \in \{0, 1, \ldots, N\}$ as well as second entries of elements of a description $\mathfrak{D}_n$ are represented by bit strings of length $l_{\text{bit}} := \lfloor \log_2 q \rfloor + 1$; and
- *the tree necklace list $\mathfrak{N}$ of f relative to $\vec{\mathfrak{T}}$, in the sense of Definition 5.3.2.8, in lexicographically sorted form, which has $O(dL 2^{d^2 \operatorname{mpe}(q-1)+dL})$ distinct elements (triples) and, by convention,*
  - *has the first entries $[\vec{\mathfrak{n}}] = [\mathfrak{n}_1, \mathfrak{n}_2, \ldots, \mathfrak{n}_{l'}]$ of its elements padded analogously to Remark 5.3.2.9 (3), but with $L - l'$ dummy entries -1, so that the bit string representation of $[\vec{\mathfrak{n}}]$ always has the length*

$$L(l_{\text{bit}} + 1) = L(\lfloor \log_2 q \rfloor + 2);$$

  - *uses $\lfloor \log_2 L \rfloor + 1$ bits to represent the second entries of its elements; and*
  - *uses $\lfloor \log_2 q \rfloor + 1$ bits for the third entries of its elements.*

Proof. First, we compute $\bar{f}$, the affine maps A_i and a partition-tree register

$$((\mathcal{Z}_i)_{i=0,1,\ldots,d-1}, ((\mathfrak{D}_n, (S_{n,i})_{i=0,1,\ldots,d}))_{n=0,1,\ldots,N})$$

for f with $N \in O(d 2^{d^2 \operatorname{mpe}(q-1)+d})$; the desired recursive tree description list $\vec{\mathfrak{D}}$ is a part of this. By Proposition 5.1.8 and Theorem 5.1.9 (2), these computations can be carried out within q-bounded query complexity

$$(d^3 \operatorname{mpe}(q-1) 2^{(3d^2+d) \operatorname{mpe}(q-1)+2d} \log^{1+o(1)} q, d, 0, 0, 0),$$

which is majorized by the asserted overall q-bounded query complexity for computing $\vec{\mathfrak{D}}$ and $\mathfrak{N}$ (it is this term which necessitates the inclusion of the factor $2^{d\,\mathrm{mpe}(q-1)}$ in the bound on the overall bit operation cost). Moreover, by the proof of Theorem 5.1.9 (2), the following are computed (and may be stored) as part of this:

- the cycles of $\bar{f}$ and a CRL-list $\bar{\mathfrak{L}}$ of $\bar{f}$;
- the parameter H_i for each $\bar{f}$-periodic index $i \in \{0, 1, \ldots, d-1\}$.

Until further notice, we assume that $(i, \ell) \in \bar{\mathfrak{L}}$ with $i < d$ is fixed (the case $i = d$ is easy to deal with separately and will be "tacked on" at the end of this proof). As usual, we let $(i_0, i_1, \ldots, i_{\ell-1})$ with $i = i_0$ be the $\bar{f}$-cycle of i, and set $i_t := i_{t \bmod \ell}$ for arbitrary $t \in \mathbb{Z}$. We analyze the bit operation cost of counting the isomorphism types of connected components of Γ_f that intersect $\bigcup_{t=0}^{d-1} C_{i_t}$ (i.e., that may be represented by a periodic vertex in one of the cosets C_{i_t}). We note that for each $t \in \{0, 1, \ldots, \ell-1\}$, one can directly read off a spanning congruence sequence for $\mathcal{U}_{i_t}$, of length $H_i \leq d\,\mathrm{mpe}(q-1)$, from the partition-tree register computed above. To proceed, we need to determine a spanning congruence sequence of $\mathcal{Q}_{i_{-t}, H_i+L-t-1}$ for $t = 0, 1, \ldots, L-1$. By the definitions of $\mathcal{R}_i$ and $\mathcal{Q}_{i,h}$ from Section 3.3, we have

$$\mathcal{Q}_{i_{-t}, H_i+L-t-1} = \mathcal{R}_{i_{-t}} \wedge \mathfrak{P}'(\mathcal{Q}_{i_{-t-1}, H_i+L-t-2}, A_{i_{-t-1}}).$$

Moreover, we observe that

$$\begin{aligned}
&\mathfrak{P}'(\mathcal{Q}_{i_{-t-1}, H_i+L-t-2}, A_{i_{-t-1}}) \\
&\quad = \mathfrak{P}'\left(\bigwedge_{j=0}^{H_i+L-t-2} \lambda^j_{i_{-t-1-j}}(\mathcal{R}_{i_{-t-1-j}}) \wedge \mathcal{U}_{i_{-t-1}}, A_{i_{-t-1}} \right) \\
&\quad = \bigwedge_{j=0}^{H_i+L-t-2} \lambda^{j+1}_{i_{-t-1-j}}(\mathcal{R}_{i_{-t-1-j}}) \wedge \mathcal{U}_{i_{-t}} \\
&\quad = \bigwedge_{j=1}^{H_i+L-t-1} \lambda^{j}_{i_{-t-j}}(\mathcal{R}_{i_{-t-j}}) \wedge \mathcal{U}_{i_{-t}},
\end{aligned}$$

whence

$$\mathcal{Q}_{i_{-t}, H_i+L-t-1} = \bigwedge_{j=0}^{H_i+L-t-1} \lambda^j_{i_{-t-j}}(\mathcal{R}_{i_{-t-j}}) \wedge \mathcal{U}_{i_{-t}}.$$

Now, from our partition-tree register, we can directly read off a spanning congruence sequence for $\lambda^j_{i_t}(\mathcal{R}_{i_t})$, of length $n_{i_t} \leq d$, for each $t = 0, 1, \ldots, \ell-1$ and each $j = 0, 1, \ldots, H_i$, which altogether takes only $O(d^3\,\mathrm{mpe}(q-1)\log q)$ bit operations for copying. Moreover, for any fixed $t \in \{0, 1, \ldots, \ell-1\}$, we can compute a spanning congruence sequence for

$$\lambda^j_{i_t}(\mathcal{R}_{i_t}) = \lambda(\lambda^{j-1}_{i_t}(\mathcal{R}_{i_t}), A_{i_{t+j-1}})$$

successively for $j = H_i + 1, H_i + 2, \ldots, H_i + L - 1$. For each given t and j, this takes $O(d \log^{1+o(1)} q)$ bit operations, and thus for all t and j together, it takes $O(d^2 L \log^{1+o(1)} q)$ bit operations. Once all of these spanning congruence sequences have been computed, one can paste together such a sequence for a single partition of the form $\mathcal{Q}_{i_{-t}, H_i+L-t-1}$ using $O((H_i + L)d \log q) \subseteq O((d^2 \operatorname{mpe}(q-1) + dL) \log q)$ bit operations. Doing so for all $t = 0, 1, \ldots, L-1$ takes $O((d^2 L \operatorname{mpe}(q-1) + dL^2) \log q)$ bit operations.

Our next goal is to compute the function

$$\mathfrak{u}_{i_{-t}, L-t} : \mathcal{O}_{i_{-t}, L-t} \to \mathcal{O}_{i_{-t-1}, L-t-1}$$

for $t = 0, 1, \ldots, L-2$. To that end, we first compute the set

$$\mathcal{O}_{i_{-t}, L-t} = \{\diamond_{k=0}^{H_i+L-t-1} \vec{o}'_k \in \{\emptyset, \neg\}^{n_{i_{-t}} + n_{i_{-t+1}} + \cdots + n_{i_{-H_i-L+1}}} : \mathcal{B}(\mathcal{Q}_{i_{-t}, H_i+L-t-1}, \diamond_{k=0}^{H_i+L-t-1} \vec{o}'_k \diamond \vec{\xi}_{i_{-t}, H_i}) \neq \emptyset\}$$

for $t = 0, 1, \ldots, L-1$. To do so, we go through the

$$O(2^{d(H_i+L-t)}) \subseteq O(2^{d^2 \operatorname{mpe}(q-1) + dL - dt})$$

tuples $\diamond_{k=0}^{H_i+L-t-1} \vec{o}'_k$, and for each of them, we compute the cardinality of the block

$$\mathcal{B}(\mathcal{Q}_{i_{-t}, H_i+L-t-1}, \diamond_{k=0}^{H_i+L-t-1} \vec{o}'_k \diamond \vec{\xi}_{i_{-t}, H_i}).$$

Following the ideas leading to Proposition 3.3.2, this cardinality is equal to the distribution number

$$\sigma_{\mathcal{Q}_{i_{-t}, H_i+L-t-1}, \mathbf{0}}(\diamond_{k=0}^{H_i+L-t-1} \vec{o}'_k \diamond \vec{\xi}_{i_{-t}, H_i}, (\emptyset, \ldots, \emptyset)) \tag{5.9}$$

of $\mathcal{Q}_{i_{-t}, H_i+L-t-1}$ under the constantly zero affine function $\mathbf{0}$. By the proof of Lemma 5.2.2.1 and the facts that

- the number of spanning congruences of $\mathcal{Q}_{i_{-t}, H_i+L-t-1}$ we are using is at most $d(H_i + L) + H_i \leq d^2 \operatorname{mpe}(q-1) + d \operatorname{mpe}(q-1) + dL \in O(d^2 \operatorname{mpe}(q-1) + dL)$,
- the subsets J we need to loop over never contain any index corresponding to a logical sign for one of the H_i spanning congruences of $\mathcal{U}_{i_{-t}}$, because $\vec{\xi}_{i_{-t}, H_i}$ consists only of positive logical signs, and
- $d(H_i + L) \leq d^2 \operatorname{mpe}(q-1) + dL$,

we conclude that the complexity of computing the distribution number (5.9) is in

$$O(2^{d^2 \operatorname{mpe}(q-1) + dL} (d^2 \operatorname{mpe}(q-1) + dL) \log^{1+o(1)} q). \tag{5.10}$$

In summary, computing all sets $\mathcal{O}_{i_{-t},L-t}$ for $t = 0, 1, \ldots, L-1$ takes

$$O\left(\sum_{t=0}^{L-1} 4^{d^2\,\mathrm{mpe}(q-1)+dL} 2^{-dt}(d^2\,\mathrm{mpe}(q-1)+dL)\log^{1+o(1)} q\right)$$
$$\subseteq O(4^{d^2\,\mathrm{mpe}(q-1)+dL}(d^2\,\mathrm{mpe}(q-1)+dL)\log^{1+o(1)} q)$$

bit operations.

Concerning the computation of the functions $\mathfrak{u}_{i_{-t},L-t}$ themselves, we note that for a given t and argument $\diamond_{k=0}^{H_i+L-t-1}\vec{o}'_k$ of $\mathfrak{u}_{i_{-t},L-t}$, the associated function value $\mathfrak{u}_{i_{-t},L-t}(\diamond_{k=0}^{H_i+L-t-1}\vec{o}'_k)$ is the unique tuple $\diamond_{k=0}^{H_i+L-t-2}\vec{o}_k \in \mathcal{O}_{i_{-t-1},L-t-1}$ such that the distribution number

$$\sigma_{\mathcal{Q}_{i_{-t-1},H_i+L-t-2},A_{i_{-t-1}}}(\diamond_{k=0}^{H_i+L-t-2}\vec{o}_k \diamond \vec{\xi}_{i_{-t-1},H_i}, \diamond_{k=1}^{H_i+L-t-1}\vec{o}'_k \diamond \vec{\xi}_{i_{-t},H_i})$$

is equal to 1 (and, according to the proof of Lemma 5.3.3.1, that distribution number is 0 for all other choices of $\diamond_{k=0}^{H_i+L-t-2}\vec{o}_k$). Therefore, in order to compute each value of every function $\mathfrak{u}_{i_{-t},L-t}$ for $t = 0, 1, \ldots, L-2$, we need to carry out

$$O\left(\sum_{t=0}^{L-2} 4^{d^2\,\mathrm{mpe}(q-1)+dL-dt} 2^{-d}\right) \subseteq O(4^{d^2\,\mathrm{mpe}(q-1)+dL})$$

computations of a distribution number of $\mathcal{Q}_{i_{-t-1},H_i+L-t-2}$ of the above form, and as above (this time using that all entries of $\vec{\xi}_{i_{-t-1},H_i}$ are the positive logical sign), the bit operation cost of each individual such distribution number computation is (5.10). Therefore, we end up with a total bit operation cost of

$$O(8^{d^2\,\mathrm{mpe}(q-1)+dL}(d^2\,\mathrm{mpe}(q-1)+dL)\log^{1+o(1)} q)$$

for computing all functions $\mathfrak{u}_{i_{-t},L-t}$.

Next, we compute the set $\mathfrak{C}_{i,L}$, and for each $l \in \mathfrak{C}_{i,L}$, we compute the s-congruence $\eta_{i,l}(x)$. By Lemma 5.1.5 (1,3,4,8), this takes $O((d+L)\log^{1+o(1)} q)$ bit operations altogether if $\mathcal{A}_{i,l} = (A_{i_0}A_{i_1}\cdots A_{i_{\ell-1}})^{l/\ell} = \mathcal{A}_i^{l/\ell}$, with linear coefficient $\overline{\alpha}_{i,l}$ and constant coefficient $\overline{\beta}_{i,l}$, is stored for each l such that $\ell \mid l$, then computed for the next larger relevant value, $l+\ell$, using the formula $\mathcal{A}_{i,l+\ell} = \mathcal{A}_{i,l}\mathcal{A}_i$ whenever $\ell \mid l$. With these computations, we have established a spanning congruence sequence for $\mathcal{W}_{i,L}$, of length at most $d(H_i+L)+H_i+L \le d^2\,\mathrm{mpe}(q-1)+d\,\mathrm{mpe}(q-1)+dL+L$. Now, we go through the $O(2^{d(H_i+L)}L) \subseteq O(2^{d^2\,\mathrm{mpe}(q-1)+dL}L)$ logical sign tuples $\diamond_{k=0}^{H_i+L-1}\vec{o}'_k \diamond \vec{\nu}_{i,L,l}$ that parametrize subsets of $\mathbb{Z}/s\mathbb{Z}$ of the form $\mathcal{B}(\mathcal{W}_{i,L}, \diamond_{k=0}^{H_i+L-1}\vec{o}'_k \diamond \vec{\xi}_{i,H_i} \diamond \vec{\nu}_{i,L,l})$ – we observe that the non-empty such sets are just those blocks of $\mathcal{W}_{i,L}$ that consist of f-periodic points. For each such tuple, we compute

$$\mathfrak{m}_i(\diamond_{k=0}^{H_i+L-1}\vec{o}'_k \diamond \vec{\nu}_{i,L,l}) := |\mathcal{B}(\mathcal{W}_{i,L}, \diamond_{k=0}^{H_i+L-1}\vec{o}'_k \diamond \vec{\xi}_{i,H_i} \diamond \vec{\nu}_{i,L,l})|$$

as the distribution number

$$\sigma w_{i,L,\mathbf{0}}(\diamond_{k=0}^{H_i+L-1}\vec{o}'_k \diamond \vec{\xi}_{i,H_i} \diamond \vec{\nu}_{i,L,l}, (\emptyset, \emptyset, \dots, \emptyset)),$$

which costs

$$\begin{aligned} &O(2^{d(H_i+L)+L}(d(H_i+L)+H_i+L)\log^{1+o(1)} q) \\ &\quad \subseteq O(2^{d^2\,\mathrm{mpe}(q-1)+dL+L}(d^2\,\mathrm{mpe}(q-1)+dL)\log^{1+o(1)} q) \end{aligned}$$

bit operations per cardinality to compute. If $\mathfrak{m}_i(\diamond_{k=0}^{H_i+L-1}\vec{o}'_k \diamond \vec{\nu}_{i,L,l}) = 0$, then we discard that case. Otherwise, we store the computed block size. Overall, this process takes

$$\begin{aligned} &O(4^{d^2\,\mathrm{mpe}(q-1)+dL}2^L(d^2L\,\mathrm{mpe}(q-1)+dL^2)\log^{1+o(1)} q) \\ &\quad \subseteq O(8^{d^2\,\mathrm{mpe}(q-1)+dL}(d^2L\,\mathrm{mpe}(q-1)+dL^2)\log^{1+o(1)} q) \end{aligned}$$

bit operations.

We recall that in view of Lemmas 5.3.3.1 and 5.3.3.2, the blocks of $\mathcal{W}_{i,L}$ that consist of f-periodic points control the digraph isomorphism type of the connected component of Γ_f containing any given point in the block. We use Lemma 5.3.3.2 to compute, for each tuple $\diamond_{k=0}^{H_i+L-1}\vec{o}'_k \diamond \vec{\nu}_{i,L,l}$ whose $\mathfrak{m}_i$-value (the associated block size) is non-zero, the unique cyclic sequence $[\mathfrak{n}_1, \mathfrak{n}_2, \dots, \mathfrak{n}_{l'}]$ with entries in $\{0, 1, \dots, N\}$ and of minimal period l' such that the cyclic sequence of rooted tree isomorphism types characterizing the corresponding digraph isomorphism type is equal to

$$[\diamond_{m=1}^{l/l'}(\mathfrak{T}_{\mathfrak{n}_1}, \mathfrak{T}_{\mathfrak{n}_2}, \dots, \mathfrak{T}_{\mathfrak{n}_{l'}})].$$

To that end, for fixed $\diamond_{k=0}^{H_i+L-1}\vec{o}'_k \diamond \vec{\nu}_{i,L,l}$, we compute the sequence

$$(\mathfrak{n}^{(-l+1)}, \mathfrak{n}^{(-l+2)}, \dots, \mathfrak{n}^{(0)}),$$

where $\mathfrak{n}^{(-t)}$ for $t \in \{0, 1, \dots, l-1\}$ is the unique index in $\{0, 1, \dots, N\}$ such that

$$\begin{aligned} \mathrm{Tree}_{i_{-t}}(\mathcal{P}_{i_{-t}}, (\mathrm{proj}_{i_{-t},L-t} &\circ \mathfrak{u}_{i_{-t+1},L-t+1} \\ &\circ \mathfrak{u}_{i_{-t+2},L-t+2} \circ \cdots \circ \mathfrak{u}_{i_0,L})(\diamond_{k=0}^{H_i+L-1}\vec{o}'_k)) \cong \mathfrak{T}_{\mathfrak{n}^{(-t)}}. \end{aligned}$$

For each fixed t, this requires us to compute the logical sign tuple

$$(\mathrm{proj}_{i_{-t},L-t} \circ \mathfrak{u}_{i_{-t+1},L-t+1} \circ \mathfrak{u}_{i_{-t+2},L-t+2} \circ \cdots \circ \mathfrak{u}_{i_0,L})(\diamond_{k=0}^{H_i+L-1}\vec{o}'_k). \tag{5.11}$$

Now, we can compute

$$(\mathfrak{u}_{i_{-t+1},L-t+1} \circ \mathfrak{u}_{i_{-t+2},L-t+2} \circ \cdots \circ \mathfrak{u}_{i_0,L})(\diamond_{k=0}^{H_i+L-1}\vec{o}'_k)$$

simply by looking up the pre-computed values of the functions $\mathfrak{u}_{i_{-j},L-j}$, which takes $O(t(H_i+L)d) \subseteq O(d^2 L \operatorname{mpe}(q-1) + dL^2)$ bit operations. The $\operatorname{proj}_{i_{-t},L-t}$-value of this tuple is a projection onto an initial segment, which can be read off using $O(d(H_i+1)) \subseteq O(d^2 \operatorname{mpe}(q-1))$ bit operations. Finally, we need to look up the number $\mathfrak{n}^{(-t)}$ in our partition-tree register – it is characterized by the inclusion

$$(\operatorname{proj}_{i_{-t},L-t} \circ \mathfrak{u}_{i_{-t+1},L-t+1} \circ \mathfrak{u}_{i_{-t+2},L-t+2} \circ \cdots \circ \mathfrak{u}_{i_0,L})(\diamond_{k=0}^{H_i+L-1} \vec{o}'_k) \in S_{\mathfrak{n}^{(-t)},i_{-t},H_i}.$$

Therefore, in order to determine and store $\mathfrak{n}^{(-t)}$, we need to go through the pairwise disjoint sets S_{n,i_{-t},H_i} for $n = 0, 1, \ldots, N$ until we find the one that contains the logical sign tuple (5.11). Because $\sum_{n=0}^{N} |S_{n,i_{-t},H_i}| \leq 2^{d(H_i+1)} \in O(2^{d^2 \operatorname{mpe}(q-1)+d})$, and each element of each set S_{n,i_{-t},H_i} is a logical sign tuple of length in $O(d(H_i+1)) \subseteq O(d^2 \operatorname{mpe}(q-1))$, it takes

$$\begin{aligned} &O(N + 2^{d^2 \operatorname{mpe}(q-1)+d} d^2 \operatorname{mpe}(q-1) + \log q) \\ &\quad \subseteq O(d^2 \operatorname{mpe}(q-1) 2^{d^2 \operatorname{mpe}(q-1)+d} + \log q) \end{aligned}$$

bit operations to determine $\mathfrak{n}^{(-t)}$ for our fixed value of t. In total, the computation of the sequence $\vec{\mathfrak{n}} = (\mathfrak{n}^{(-l+1)}, \mathfrak{n}^{(-l+2)}, \ldots, \mathfrak{n}^{(0)})$ for a fixed value of $\diamond_{k=0}^{H_i+L-1} \vec{o}'_k \diamond \vec{\nu}_{i,L,l}$ takes

$$O(d^2 L^2 \operatorname{mpe}(q-1) + dL^3 + d^2 L \operatorname{mpe}(q-1) 2^{d^2 \operatorname{mpe}(q-1)+d} + L \log q)$$

bit operations. The cyclic sequence $[\mathfrak{n}_1, \mathfrak{n}_2, \ldots, \mathfrak{n}_{l'}]$ we are looking for is simply

$$[\mathfrak{n}^{(-l+1)}, \mathfrak{n}^{(-l+2)}, \ldots, \mathfrak{n}^{(-l+l')}],$$

where $l' = \operatorname{minperl}(\vec{\mathfrak{n}})$, which can be computed in

$$O(l^{1+o(1)} l_{\text{bit}}) \subseteq O(L^{1+o(1)} \log q)$$

bit operations by Lemma 5.3.2.7. Finally, in representing this cyclic sequence, we would like to replace $(\mathfrak{n}_1, \ldots, \mathfrak{n}_{l'})$ by the lexicographically minimal ordered sequence in the same cyclic equivalence class. This takes another $O(L^2 \log L \log q)$ bit operations per such sequence. In total, the computation of the cyclic sequence

$$[\mathfrak{n}_1, \mathfrak{n}_2, \ldots, \mathfrak{n}_{l'}]$$

for all of the $O(2^{d(H_i+L)} L) \subseteq O(2^{d^2 \operatorname{mpe}(q-1)+dL} L)$ logical sign tuples $\diamond_{k=0}^{H_i+L-1} \vec{o}'_k \diamond \vec{\nu}_{i,L,l}$ takes

$$\begin{aligned} &O(2^{d^2 \operatorname{mpe}(q-1)+dL} L \cdot (d^2 L^2 \operatorname{mpe}(q-1) + dL^3 + d^2 L \operatorname{mpe}(q-1) 2^{d^2 \operatorname{mpe}(q-1)+d} \\ &+ L^2 \log L \log q)) \subseteq O(8^{d^2 \operatorname{mpe}(q-1)+dL} \log^{1+o(1)} q) \end{aligned}$$

bit operations. In summary, the bit operation cost of the computations described after fixing $(i, \ell) \in \bar{\mathcal{L}} \setminus \{(d, 1)\}$ for all of those $O(d)$ pairs (i, ℓ) together is in

$$O(8^{d^2 \operatorname{mpe}(q-1)+dL}(d^3 L \operatorname{mpe}(q-1) + d^2 L^2) \log^{1+o(1)} q).$$

At this point, we have computed, for each $(i, \ell) \in \bar{\mathcal{L}} \setminus \{(d, 1)\}$, a set $\mathfrak{N}_i$ of the form

$$\begin{aligned}\mathfrak{N}_i = \{&(\diamond_{k=0}^{H_i+L-1} \vec{o}'_k \diamond \vec{\nu}_{i,L,l}, [\mathfrak{n}_1, \mathfrak{n}_2, \ldots, \mathfrak{n}_{l'}], \mathfrak{m}_i(\diamond_{k=0}^{H_i+L-1} \vec{o}'_k \diamond \vec{\nu}_{i,L,l})) : \\ &\vec{o}'_k \in \{\emptyset, \neg\}^{n_{i-k}} \text{ for } k = 0, 1, \ldots, H_i + L - 1, l \in \mathfrak{C}_{i,L}, \\ &\text{and } \mathfrak{m}_i(\diamond_{k=0}^{H_i+L-1} \vec{o}'_k \diamond \vec{\nu}_{i,L,l}) > 0\}.\end{aligned}$$

We note that for each element of $\mathfrak{N}_i$, we have $l' \mid l$ and $\operatorname{minperl}([\mathfrak{n}_1, \ldots, \mathfrak{n}_{l'}]) = l'$. We also observe that while $\mathfrak{N}_i$ is *not* the tree necklace list, relative to $\vec{\mathfrak{F}}$, for the restriction of f to the union of all cosets C_j, where j is a vertex of the connected component of $\Gamma_{\bar{f}}$ containing i, that tree necklace list could be easily derived from $\mathfrak{N}_i$ as follows. If we fix l and $[\mathfrak{n}_1, \ldots, \mathfrak{n}_{l'}]$ and add up all the third entries of the corresponding triples in $\mathfrak{N}_i$ (i.e., of those triples where the first entry has terminal segment $\vec{\nu}_{i,L,l}$ and the second entry is $[\mathfrak{n}_1, \ldots, \mathfrak{n}_{l'}]$), then we end up with the exact number of connected components of Γ_f that are characterized by the cyclic sequence of rooted tree isomorphism types $[\diamond_{m=1}^{l/l'}(\mathfrak{F}_{\mathfrak{n}_1}, \ldots, \mathfrak{F}_{\mathfrak{n}_{l'}})]$ and are contained in the union of all cosets C_j for j in the connected component of $\Gamma_{\bar{f}}$ containing i. This observation also implies that we can compute the full tree necklace list $\mathfrak{N}$ of f relative to $\vec{\mathfrak{F}}$ as follows.

We start by setting $\mathfrak{N} := \emptyset$. Throughout the process described below, $\mathfrak{N}$ is a set of triples $([\mathfrak{n}_1, \mathfrak{n}_2, \ldots, \mathfrak{n}_{l'}], l, \mathfrak{m})$ such that

- $|\mathfrak{N}| \leq \sum_{(i,\ell) \in \bar{\mathcal{L}} \setminus \{(d,1)\}} |\mathfrak{N}_i| \in O(dL2^{d^2 \operatorname{mpe}(q-1)+dL})$;
- $\mathfrak{n}_1, \ldots, \mathfrak{n}_{l'} \in \{0, 1, \ldots, N\}$;
- $\operatorname{minperl}([\mathfrak{n}_1, \ldots, \mathfrak{n}_{l'}]) = l'$;
- $[\diamond_{m=1}^{l/l'}(\mathfrak{F}_{\mathfrak{n}_1}, \mathfrak{F}_{\mathfrak{n}_2}, \ldots, \mathfrak{F}_{\mathfrak{n}_{l'}})]$ is a cyclic sequence of rooted tree isomorphism types that characterizes at least one connected component of Γ_f that is contained in $\mathbb{F}_q^*$; and
- $\mathfrak{m} \in \mathbb{N}^+$ is the exact number of connected components of Γ_f that are contained in $\mathbb{F}_q^*$ and are characterized by $[\diamond_{m=1}^{l/l'}(\mathfrak{F}_{\mathfrak{n}_1}, \mathfrak{F}_{\mathfrak{n}_2}, \ldots, \mathfrak{F}_{\mathfrak{n}_{l'}})]$.

We remind the reader that the first entry of each triple in $\mathfrak{N}$ is represented by an ordered list of length exactly L (via padding) each entry of which has bit length in $O(\log q)$, and that the second and third entries of such a triple are represented by bit strings of length in $O(\log L)$ and $O(\log q)$, respectively. Now, to get an almost-final form of $\mathfrak{N}$, we loop over the pairs $(i, \ell) \in \bar{\mathcal{L}} \setminus \{(d, 1)\}$, and for each such pair, we loop over the elements of $\mathfrak{N}_i$; this double loop has $O(dL2^{d^2 \operatorname{mpe}(q-1)+dL})$ individual iterations. In each iteration, we add at most one new element to $\mathfrak{N}$, which explains

the above bound on $|\mathfrak{N}|$ that is valid throughout the process. Specifically, an iteration consists of the following steps. Associated with the triple in $\mathfrak{N}_i$ we are considering, we have the parameters l, which can be read off from the first entry $\diamond_{k=0}^{H_i+L} \vec{o}'_k \diamond \vec{v}_{i,L,l}$ of the triple using $O(l \log l) \subseteq O(L^{1+o(1)})$ bit operations (by scanning to find the first positive logical sign in $\vec{v}_{i,L,l}$ and incrementing a counter during that process), and $[\mathfrak{n}_1, \ldots, \mathfrak{n}_{l'}]$. We check whether these already occur as the first two entries of some element of $\mathfrak{N}$, which takes

$$O(dL2^{d^2 \operatorname{mpe}(q-1)+dL} \cdot L \log q) = O(dL^2 2^{d^2 \operatorname{mpe}(q-1)+dL} \log q)$$

bit operations. If this is the case, then we end the current iteration, having added nothing to $\mathfrak{N}$. Otherwise, we compute the sum $\mathfrak{m}$ of the third entries of all triples in $\bigcup_{(j,l') \in \bar{\mathfrak{L}} \setminus \{(d,1)\}} \mathfrak{N}_j$ that have the parameters l and $[\mathfrak{n}_1, \ldots, \mathfrak{n}_{l'}]$ associated with them. This takes

$$O(dL2^{d^2 \operatorname{mpe}(q-1)+dL}(L \log L + L \log q)) = O(dL^2 2^{d^2 \operatorname{mpe}(q-1)+dL} \log q)$$

bit operations. Then we add $([\mathfrak{n}_1, \ldots, \mathfrak{n}_{l'}], l, \mathfrak{m})$ to $\mathfrak{N}$ as a new element and end the current iteration. Overall, this double loop takes

$$\begin{aligned} &O(dL2^{d^2 \operatorname{mpe}(q-1)+dL} \cdot dL^2 2^{d^2 \operatorname{mpe}(q-1)+dL} \log q) \\ &= O(d^2 L^3 4^{d^2 \operatorname{mpe}(q-1)+dL} \log q) \\ &\subseteq O(8^{d^2 \operatorname{mpe}(q-1)+dL} \log q) \end{aligned}$$

bit operations.

At the end of the double loop, $\mathfrak{N}$ is almost equal to the tree necklace list for f relative to $\vec{\mathfrak{F}}$; only the connected component of Γ_f containing $0_{\mathbb{F}_q}$, which is characterized by the cyclic sequence $[\operatorname{Tree}_{\Gamma_f}(0_{\mathbb{F}_q})]$, has not yet been accounted for. To find the index number $\mathfrak{n} \in \{0, 1, \ldots, N\}$ of $\operatorname{Tree}_{\Gamma_f}(0_{\mathbb{F}_q})$ with respect to our partition-tree register, we note that $\mathfrak{n}$ is characterized by the equality $S_{\mathfrak{n},d} = \emptyset$ (the positive logical sign). Therefore, we only need an additional $O(N) \subseteq O(d2^{d^2 \operatorname{mpe}(q-1)+d})$ bit operations to find $\mathfrak{n}$. Then, we need to check whether $[\mathfrak{n}]$ and 1 already occur as the first two entries of some triple in $\mathfrak{N}$, which takes

$$O(dL2^{d^2 \operatorname{mpe}(q-1)+dL} \cdot \log q) \subseteq O(8^{d^2 \operatorname{mpe}(q-1)+dL} \log q)$$

bit operations. If so, we increase the third entry of that triple by 1 and halt. Otherwise, we add $([\mathfrak{n}], 1, 1)$ to $\mathfrak{N}$ as a new element.

Finally, we sort the computed array representing $\mathfrak{N}$ lexicographically. By Lemma 5.1.5 (10), since $|\mathfrak{N}| \in O(dL2^{d^2 \operatorname{mpe}(q-1)+dL})$ and each entry of the array has bit length in $O(L \log q)$, this takes

$$O(dL^2(d^2 \operatorname{mpe}(q-1) + dL)2^{d^2 \operatorname{mpe}(q-1)+dL} \log q) \subseteq O(8^{d^2 \operatorname{mpe}(q-1)+dL} \log q)$$

bit operations. We conclude by outputting $(\vec{\mathfrak{D}}, \mathfrak{N})$ and halting. ∎

Of course, Theorem 5.3.3.4 is not useful in practice unless the maximum cycle length of a generalized cyclotomic mapping can be computed efficiently. The following proposition takes care of that.

Proposition 5.3.3.5. *Let f be an index d generalized cyclotomic mapping of $\mathbb{F}_q$. The maximum cycle length of $f_{|\mathrm{per}(f)}$ can be computed within q-bounded query complexity*

$$(d \log^{2+o(1)} q + d \log^2 d, d, 1, d, 0).$$

Proof. This is similar in spirit to Proposition 5.3.2.3, but we obtain a better bound than by using that proposition directly, because there is no need to spell out entire cycle types here. First, we compute $\bar{f}$, the affine maps A_i, the cycles of $\bar{f}$ and a CRL-list $\bar{\mathcal{L}}$ of $\bar{f}$. By Proposition 5.1.8 and the beginning of Section 5.2.1, this takes q-bounded query complexity

$$(d \log^{1+o(1)} q + d \log^2 d, d, 0, 0, 0).$$

We also factor s, using a single q-bounded mdl query (i.e., spending q-bounded query complexity $(\log q, 0, 1, 0, 0)$).

Now, we loop over the pairs $(i, \ell) \in \bar{\mathcal{L}} \setminus \{(d, 1)\}$, and for each of them, we do the following. First, we compute $\mathcal{A}_i = A_{i_0} A_{i_1} \cdots A_{i_{\ell-1}} : x \mapsto \bar{\alpha}_i x + \bar{\beta}_i$, taking $O(\ell \log^{1+o(1)} q)$ bit operations (per (i, ℓ)) by the beginning of Section 5.2.1. Next, we determine the largest cycle length of $\mathcal{A}_i$ on $\mathrm{per}(\mathcal{A}_i) \subseteq \mathbb{Z}/s\mathbb{Z}$. To do so, we note that every cycle length of $\mathcal{A}_i$ is a least common multiple of cycle lengths of those primary components $\bar{\mathcal{A}}_{i,p} = \mathcal{A}_i \bmod p^{\nu_p(s)}$, where $p \mid s$ and $p \nmid \bar{\alpha}_i$. But by [15, Tables 3 and 4] (or our Table 2.2), the largest cycle length of $\bar{\mathcal{A}}_{i,p}$ is equal to the order of $\bar{\mathcal{A}}_{i,p}$ in $\mathrm{Sym}(\mathbb{Z}/p^{\nu_p(s)}\mathbb{Z})$, i.e., to the least common multiple of all cycle lengths of $\bar{\mathcal{A}}_{i,p}$. It follows that the largest cycle length of $\mathcal{A}_i$ is equal to $\mathrm{ord}(\mathcal{A}_i \bmod s_i')$, where $s_i' = \prod_{p \mid s, p \nmid \bar{\alpha}_i} p^{\nu_p(s)}$. We compute this order as follows.

We set $s_i' := 1$ and loop over the $O(\log q)$ primes p dividing s (which can be read off from the factorization of s computed above). For each p, we check whether $p \mid \bar{\alpha}_i$, taking $O(\log^{1+o(1)} q)$ bit operations by Lemma 5.1.5 (3). If so, we skip to the next p; otherwise, we overwrite $s_i' := s_i' \cdot p^{\nu_p(s)}$, taking $O(\log^{1+o(1)} q)$ bit operations for the multiplication (there is no need to compute the power $p^{\nu_p(s)}$, because it is part of the output of the mdl query used to factor s). At the end of this loop over p, which has a bit operation cost in $O(\log^{2+o(1)} q)$, the variable s_i' has the desired value. Now, we compute $\bar{\alpha}_i' := \bar{\alpha}_i \bmod s_i'$ and $\bar{\beta}_i' := \bar{\beta}_i \bmod s_i'$, taking $O(\log^{1+o(1)} q)$ bit operations, to get the affine map $\mathcal{A}_i' = \mathcal{A}_i \bmod s_i' : x \mapsto \bar{\alpha}_i' x + \bar{\beta}_i'$ of $\mathbb{Z}/s_i'\mathbb{Z}$. We wish to compute $\mathrm{ord}(\mathcal{A}_i')$. To that end, we first compute the (multiplicative) order $\mathrm{ord}_{s_i'}(\bar{\alpha}_i')$ with a q-bounded mord query. We observe that $\mathrm{ord}_{s_i'}(\bar{\alpha}_i')$ divides $\mathrm{ord}(\mathcal{A}_i')$, because $(\mathcal{A}_i')^t(x) = (\bar{\alpha}_i')^t x + c(\bar{\alpha}_i', \bar{\beta}_i', t)$ for all $x \in \mathbb{Z}/s_i'\mathbb{Z}$ and all $t \in \mathbb{Z}$. Therefore,

$$\mathrm{ord}(\mathcal{A}_i') = \mathrm{ord}_{s_i'}(\bar{\alpha}_i') \cdot \mathrm{ord}((\mathcal{A}_i')^{\mathrm{ord}_{s_i'}(\bar{\alpha}_i')}).$$

But $(\mathcal{A}'_i)^{\mathrm{ord}_{s'_i}(\overline{\alpha}'_i)}$ is the translation $x \mapsto x + \overline{\beta}''_i$ such that

$$\overline{\beta}''_i = \begin{cases} \overline{\beta}'_i, & \text{if } \overline{\alpha}'_i = 1, \\ \overline{\beta}'_i \frac{(\overline{\alpha}'_i)^{\mathrm{ord}_{s'_i}(\overline{\alpha}'_i)} - 1}{\overline{\alpha}'_i - 1}, & \text{otherwise,} \end{cases}$$

where the formula in the second case is to be evaluated in the ring $\mathbb{Z}$, although the result is to be viewed as an element of $\mathbb{Z}/s'_i\mathbb{Z}$. Because $\mathrm{ord}((\mathcal{A}'_i)^{\mathrm{ord}_{s'_i}(\overline{\alpha}'_i)})$ is equal to the additive order of $\overline{\beta}''_i$ modulo s'_i, we can work out $\mathrm{ord}(\mathcal{A}'_i)$, which coincides with the largest cycle length of $\mathcal{A}_i$, using another $O(\log^{2+o(1)} q) = O(\log^{2+o(1)}(q^2))$ bit operations (for computing $\overline{\beta}''_i \in \mathbb{Z}/s'_i\mathbb{Z}$, which may involve a power computation modulo $s'_i(\overline{\alpha}'_i - 1) \in O(q^2)$, and working out its additive order modulo s'_i via a gcd computation and a division). We conclude this loop by setting $l_i := \ell \cdot \mathrm{ord}(\mathcal{A}'_i)$, which takes $O(\log^{1+o(1)} q)$ bit operations to compute and is the largest cycle length of f on its periodic points in $\bigcup_{t=0}^{\ell-1} C_{i_t}$. This ends our description of the loop over $(i, \ell) \in \overline{\mathcal{L}} \setminus \{(d, 1)\}$, which overall takes q-bounded query complexity $(d \log^{2+o(1)} q, 0, 0, d, 0)$, using that $\sum_{(i,\ell)\in\overline{\mathcal{L}}} \ell \leq d$.

Finally, the maximum cycle length of f is simply the maximum value among the l_i for $(i, \ell) \in \overline{\mathcal{L}}$, where $l_d := 1$, which takes $O(d \log q)$ bit operations to compute. ∎

To conclude, we give the following corollary of Theorem 5.3.3.4, which can be seen as the main result of this subsection.

Corollary 5.3.3.6. *Let f_1 and f_2 be generalized cyclotomic mappings of $\mathbb{F}_q$, of index d_1 and d_2, respectively, and set $d := \max\{d_1, d_2\}$. Moreover, let $L \in \mathbb{N}^+$, and denote by L_1, respectively, L_2, the maximum cycle length of f_1, respectively of f_2, on its periodic points. Then, if $\min\{L_1, L_2\} \leq L$, it can be decided whether $\Gamma_{f_1} \cong \Gamma_{f_2}$ within q-bounded query complexity*

$$\begin{aligned} &(8^{d^2 \operatorname{mpe}(q-1)+dL} 2^{d \operatorname{mpe}(q-1)} (d^3 L \operatorname{mpe}(q-1) + d^2 L^2) \log^{1+o(1)} q \\ &\quad + d^2 4^{d^2 \operatorname{mpe}(q-1)+d} \log^2 q + d \log^{2+o(1)} q, \\ &\quad d, 1, d, 0), \end{aligned}$$

and thus within q-bounded Las Vegas dual complexity

$$\begin{aligned} &(8^{d^2 \operatorname{mpe}(q-1)+dL} 2^{d \operatorname{mpe}(q-1)} (d^3 L \operatorname{mpe}(q-1) + d^2 L^2) \log^{1+o(1)} q \\ &\quad + d^2 4^{d^2 \operatorname{mpe}(q-1)+d} \log^2 q + d \log^{2+o(1)} q + d \log^{7+o(1)} q, \\ &\quad d \log^{3+o(1)} q, d \log q). \end{aligned}$$

Proof. First, we compute L_1 and L_2, which takes q-bounded query complexity

$$(d \log^{2+o(1)} q + d \log^2 d, d, 1, d, 0)$$

by Proposition 5.3.3.5. We check whether $L_1 = L_2$, taking $O(\log q)$ bit operations. If not, then $\Gamma_{f_1} \ncong \Gamma_{f_2}$, so we may output "false" and halt. Otherwise, we continue by computing, for $j = 1, 2$,

- a recursive tree description list $\vec{\mathfrak{D}}^{(j)} = (\mathfrak{D}_n^{(j)})_{n=0,1,\ldots,N_j}$, with

$$N_j \in O(d2^{d^2 \operatorname{mpe}(q-1)+d})$$

  and associated rooted tree isomorphism type list $\vec{\mathfrak{F}}^{(j)}$; and
- the tree necklace list $\mathfrak{N}_j$ of f_j relative to $\vec{\mathfrak{F}}^{(j)}$ such that

$$|\mathfrak{N}_j| \in O(dL2^{d^2 \operatorname{mpe}(q-1)+dL}).$$

By Theorem 5.3.3.4, this can be done within q-bounded query complexity

$$(8^{d^2 \operatorname{mpe}(q-1)+dL} 2^{d \operatorname{mpe}(q-1)} (d^3 L \operatorname{mpe}(q-1) + d^2 L^2) \log^{1+o(1)} q, d, 0, 0, 0).$$

Next, we compute a synchronization $(\vec{\mathfrak{D}}^+, \mathrm{i})$ of $\vec{\mathfrak{D}}^{(1)}$ and $\vec{\mathfrak{D}}^{(2)}$, in the sense of Definition 5.3.2.11. By Lemma 5.3.2.12, this takes

$$\begin{aligned} &O((d^3 8^{d^2 \operatorname{mpe}(q-1)+d} + d^2 4^{d^2 \operatorname{mpe}(q-1)+d} \log q) \cdot \log q) \\ &= O(d^3 8^{d^2 \operatorname{mpe}(q-1)+d} \log q + d^2 4^{d^2 \operatorname{mpe}(q-1)+d} \log^2 q) \end{aligned}$$

bit operations. Following that, we overwrite each first entry $[\mathfrak{n}_1, \mathfrak{n}_2, \ldots, \mathfrak{n}_{l'}]$ in each triple in $\mathfrak{N}_2$ with $[\mathrm{i}(\mathfrak{n}_1), \mathrm{i}(\mathfrak{n}_2), \ldots, \mathrm{i}(\mathfrak{n}_{l'})]$, using lexicographically minimal representatives of cyclic equivalence classes, which results in a modified, unsorted tree necklace list $\mathfrak{N}_2'$. This takes

$$O(dL2^{d^2 \operatorname{mpe}(q-1)+dL} \cdot L^2 \log L \log q) = O(dL^3 \log L 2^{d^2 \operatorname{mpe}(q-1)+dL} \log q)$$

bit operations overall. After this, we sort $\mathfrak{N}_2'$ lexicographically, which takes

$$O(dL^2(d^2 \operatorname{mpe}(q-1) + dL)2^{d^2 \operatorname{mpe}(q-1)+dL} \log q)$$

bit operations (see also the end of the proof of Theorem 5.3.3.4). Finally, we note that $\Gamma_{f_1} \cong \Gamma_{f_2}$ if and only if $\mathfrak{N}_1 = \mathfrak{N}_2'$, so we determine the truth value of the latter, which takes

$$O(dL^2 2^{d^2 \operatorname{mpe}(q-1)+dL} \log q)$$

bit operations using a linear scan. We then output that truth value and halt. ∎

Like the previous subsections, we conclude this subsection with some pseudocode for the discussed algorithms, specifying the q-bounded query complexity (QC) of each step. We begin with the algorithm from Theorem 5.3.3.4, which on input (f, L), where f is an index d generalized cyclotomic mapping f of $\mathbb{F}_q$ such that all cycle

lengths of f are at most L, computes a pair $(\vec{\mathfrak{D}}, \mathfrak{N})$ such that $\vec{\mathfrak{D}} = (\mathfrak{D}_n)_{n=0,1,\ldots,N}$ is a recursive tree description list with $N \in O(d2^{d^2 \operatorname{mpe}(q-1)+d})$ and associated sequence of rooted tree isomorphism types $\vec{\mathfrak{T}}$, and $\mathfrak{N}$ is a tree necklace list for f relative to $\vec{\mathfrak{T}}$ with $|\mathfrak{N}| \in O(dL2^{d^2 \operatorname{mpe}(q-1)+dL+d})$.

1 Compute the induced function $\bar{f}$ on $\{0, 1, \ldots, d\}$ and the affine maps A_i of $\mathbb{Z}/s\mathbb{Z}$.
QC: $(d \log^{1+o(1)} q, d, 0, 0, 0)$.

2 Compute a partition-tree register

$$((\mathcal{Z}_i)_{i=0,1,\ldots,d-1}, ((\mathfrak{D}_n, (S_{n,i})_{i=0,1,\ldots,d}))_{n=0,1,\ldots,N})$$

of f with $N \in O(d2^{d^2 \operatorname{mpe}(q-1)+d})$. In the process, store a CRL-list $\bar{\mathfrak{L}}$ of $\bar{f}$, the cycles of $\bar{f}$, and the parameter H_i for each $\bar{f}$-periodic $i \in \{0, 1, \ldots, d-1\}$.
QC: $(d^3 \operatorname{mpe}(q-1)2^{(3d^2+d)\operatorname{mpe}(q-1)+2d} \log^{1+o(1)} q, 0, 0, 0, 0)$ because $\bar{f}$ and the A_i have already been computed.

3 Set $\vec{\mathfrak{D}} := (\mathfrak{D}_n)_{n=0,1,\ldots,N}$.
QC: $(d^2 4^{d^2 \operatorname{mpe}(q-1)+d} \log q, 0, 0, 0, 0)$.

4 For each $(i, \ell) \in \bar{\mathfrak{L}} \setminus \{(d, 1)\}$, do the following.
QC: $(8^{d^2 \operatorname{mpe}(q-1)+dL}(d^3 L \operatorname{mpe}(q-1) + d^2 L^2) \log^{1+o(1)} q, 0, 0, 0, 0)$.

4.1 For each $t = 0, 1, \ldots, \ell - 1$, do the following.
QC: $(d^3 \operatorname{mpe}(q-1) \log q + d^2 L \log^{1+o(1)} q, 0, 0, 0, 0)$.

4.1.1 From $\mathcal{Z}_{i_t}$, read off a spanning congruence sequence for $\mathcal{U}_{i_t}$ of length $H_i \le d \operatorname{mpe}(q-1)$.
QC: $(d \operatorname{mpe}(q-1) \log q, 0, 0, 0, 0)$.

4.1.2 For each $j = 0, 1, \ldots, H_i + L - 1$, do the following.
QC: $(d^2 \operatorname{mpe}(q-1) \log q + dL \log^{1+o(1)} q, 0, 0, 0, 0)$.

4.1.2.1 If $j \le H_i$, then do the following.

4.1.2.1.1 From $\mathcal{Z}_{i_{t+j}}$, read off a spanning congruence sequence for $\lambda_{i_t}^j(\mathcal{R}_{i_t})$ of length $n_{i_t} \le d$.
QC: $(d \log q, 0, 0, 0, 0)$.

4.1.2.2 Else do the following.

4.1.2.2.1 Compute a spanning congruence sequence for

$$\lambda_{i_t}^j(\mathcal{R}_{i_t}) = \lambda(\lambda_{i_t}^{j-1}(\mathcal{R}_{i_t}), A_{i_{t+j-1}})$$

of length $n_{i_t} \le d$.
QC: $(d \log^{1+o(1)} q, 0, 0, 0, 0)$.

4.2 For each $t = 0, 1, \ldots, L - 1$, do the following.
QC: $((d^2 L \operatorname{mpe}(q-1) + dL^2) \log q, 0, 0, 0, 0)$.

4.2.1 From the data stored in step 4.1, paste together a spanning congruence sequence for

$$\mathcal{Q}_{i_{-t},H_i+L-t-1} = \bigwedge_{j=0}^{H_i+L-t-1} \lambda^j_{i_{-t-j}}(\mathcal{R}_{i_{-t-j}}) \wedge \mathcal{U}_{i_{-t}}.$$

QC: $((d^2 \operatorname{mpe}(q-1) + dL) \log q, 0, 0, 0, 0)$.

4.3 For each $t = 0, 1, \ldots, L-1$, do the following.
QC: $(4^{d^2 \operatorname{mpe}(q-1)+dL}(d^2 \operatorname{mpe}(q-1) + dL) \log^{1+o(1)} q, 0, 0, 0, 0)$.

4.3.1 Set $\mathcal{O}_{i_{-t},L-t} := \emptyset$.
QC: $(\log d + \log L, 0, 0, 0, 0)$.

4.3.2 For each $\diamond_{k=0}^{H_i+L-t-1} \vec{o'_k} \in \{\emptyset, \neg\}^{n_{i_{-t}}+n_{i_{-t-1}}+\cdots+n_{i_{-H_i-L+1}}}$, do the following.
QC: $(4^{d^2 \operatorname{mpe}(q-1)+dL} 2^{-dt}(d^2 \operatorname{mpe}(q-1) + dL) \log^{1+o(1)} q, 0, 0, 0, 0)$.

4.3.2.1 Check whether

$$\sigma_{\mathcal{Q}_{i_{-t},H_i+L-t-1},\mathbf{0}}(\diamond_{k=0}^{H_i+L-t-1} \vec{o'_k} \diamond \vec{\xi}_{i_{-t},H_i}, (\emptyset, \ldots, \emptyset)) > 0,$$

and if so, add $\diamond_{k=0}^{H_i+L-t-1} \vec{o'_k}$ to $\mathcal{O}_{i_{-t},L-t}$ as a new element.
QC: $(2^{d^2 \operatorname{mpe}(q-1)+dL}(d^2 \operatorname{mpe}(q-1) + dL) \log^{1+o(1)} q, 0, 0, 0, 0)$.

4.4 For each $t = 0, 1, \ldots, L-2$, do the following.
QC: $(8^{d^2 \operatorname{mpe}(q-1)+dL}(d^2 \operatorname{mpe}(q-1) + dL) \log^{1+o(1)} q, 0, 0, 0, 0)$.

4.4.1 For each $\diamond_{k=0}^{H_i+L-t-1} \vec{o'_k} \in \mathcal{O}_{i_{-t},L-t}$, do the following.
QC: $(8^{d^2 \operatorname{mpe}(q-1)+dL} 2^{-d(t+1)} 2^{-dt}(d^2 \operatorname{mpe}(q-1) + dL) \log^{1+o(1)} q$, $0, 0, 0, 0)$.

4.4.1.1 For each $\diamond_{k=0}^{H_i+L-t-2} \vec{o_k} \in \mathcal{O}_{i_{-t-1},L-t-1}$, do the following.
QC: $(4^{d^2 \operatorname{mpe}(q-1)+dL} 2^{-d(t+1)}(d^2 \operatorname{mpe}(q-1) + dL) \log^{1+o(1)} q$, $0, 0, 0, 0)$.

4.4.1.1.1 Check whether the distribution number

$$\sigma_{\mathcal{Q}_{i_{-t-1},H_i+L-t-2},A_{i_{-t-1}}}(\diamond_{k=0}^{H_i+L-t-2} \vec{o_k} \diamond \vec{\xi}_{i_{-t-1},H_i}, \diamond_{k=1}^{H_i+L-t-1} \vec{o'_k} \diamond \vec{\xi}_{i_{-t},H_i})$$

is equal to 1. If so, set

$$\mathfrak{u}_{i_{-t},L-t}(\diamond_{k=0}^{H_i+L-t-1} \vec{o'_k}) := \diamond_{k=0}^{H_i+L-t-2} \vec{o_k}.$$

QC: $(2^{d^2 \operatorname{mpe}(q-1)+dL}(d^2 \operatorname{mpe}(q-1)+dL) \log^{1+o(1)} q, 0, 0, 0, 0)$.

4.5 Compute $\mathcal{A}_i := A_{i_0} A_{i_1} \cdots A_{i_{\ell-1}}$.
QC: $(d \log^{1+o(1)} q, 0, 0, 0, 0)$.

4.6 Set $\mathfrak{C}_{i,L} := \emptyset$.
QC: $(\log d + \log L, 0, 0, 0, 0)$.

4.7 For each $l \in \{1, 2, \ldots, L\}$, do the following.
QC: $(L \log^{1+o(1)} q, 0, 0, 0, 0)$.

4.7.1 Check whether $\ell \mid l$. If not, skip to the next l.
QC: $(\log^{1+o(1)} q, 0, 0, 0, 0)$.

4.7.2 Compute and store the affine iterate

$$\mathcal{A}_{i,l} = \mathcal{A}_i^{l/\ell} = \begin{cases} \mathcal{A}_i, & \text{if } l = \ell, \\ \mathcal{A}_{i,l-\ell} \mathcal{A}_i, & \text{if } \ell \mid l > \ell, \end{cases}$$

with linear coefficient $\overline{\alpha}_{i,l}$ and constant coefficient $\overline{\beta}_{i,l}$.
QC: $(\log q, 0, 0, 0, 0)$ if $l = \ell$ (only needing to copy information from step 4.5); $(\log^{1+o(1)} q, 0, 0, 0, 0)$ otherwise.

4.7.3 Check whether $\gcd(s, \overline{\alpha}_{i,l} - 1) \mid \overline{\beta}_{i,l}$. If so, add l to $\mathfrak{C}_{i,L}$ as a new element, and store $\eta_{i,l}(x)$ as the s-congruence

$$x \equiv -\frac{\overline{\beta}_{i,l}}{\gcd(s, \overline{\alpha}_{i,l} - 1)} \cdot \operatorname{inv}_{\frac{s}{\gcd(s, \overline{\alpha}_{i,l}-1)}} \left(\frac{\overline{\alpha}_{i,l} - 1}{\gcd(s, \overline{\alpha}_{i,l} - 1)} \right) \cdot \left(\operatorname{mod} \frac{s}{\gcd(s, \overline{\alpha}_{i,l} - 1)} \right).$$

QC: $(\log^{1+o(1)} q, 0, 0, 0, 0)$.

4.8 Set $\mathfrak{N}_i := \emptyset$.
QC: $(\log d, 0, 0, 0, 0)$.

4.9 For each $\diamond_{k=0}^{H_i+L-1} \vec{o}'_k \in \mathcal{O}_{i,L}$, do the following.
QC: $(4^{d^2 \operatorname{mpe}(q-1)+dL} 2^L (d^2 L \operatorname{mpe}(q-1) + dL^2) \log^{1+o(1)} q, 0, 0, 0, 0)$.

4.9.1 For each $l \in \mathfrak{C}_{i,L}$, do the following.
QC: $(2^{d^2 \operatorname{mpe}(q-1)+dL+L} (d^2 L \operatorname{mpe}(q-1) + dL^2) \log^{1+o(1)} q, 0, 0, 0, 0)$.

4.9.1.1 Compute the logical sign tuple $\vec{\nu}_{i,L,l}$ (see the paragraph before Proposition 5.3.3.3).
QC: $(L \log^{1+o(1)} q, 0, 0, 0, 0)$.

4.9.1.2 Set

$$\mathfrak{m}_i(\diamond_{k=0}^{H_i+L-1} \vec{o}'_k \diamond \vec{\nu}_{i,L,l}) := \sigma \mathfrak{w}_{i,L,\mathbf{0}}(\diamond_{k=0}^{H_i+L-1} \vec{o}'_k \diamond \vec{\xi}_{i,H_i} \diamond \vec{\nu}_{i,L,l}, (\emptyset, \ldots, \emptyset)).$$

QC: $(2^{d^2 \operatorname{mpe}(q-1)+dL+L} (d^2 \operatorname{mpe}(q-1) + dL) \log^{1+o(1)} q, 0, 0, 0, 0)$.

4.9.1.3 If $\mathfrak{m}_i(\diamond_{k=0}^{H_i+L-1}\vec{o}'_k \diamond \vec{\nu}_{i,L,l}) = 0$, then skip to the next l.
QC: $(d^2 \operatorname{mpe}(q-1) + dL, 0, 0, 0, 0)$.

4.9.1.4 For each $t = 0, 1, \ldots, l-1$, do the following.
QC: $(dL^3 + d^2 L \operatorname{mpe}(q-1) 2^{d^2 \operatorname{mpe}(q-1)+d} + L \log q, 0, 0, 0, 0)$.

4.9.1.4.1 Compute the logical sign tuple

$$(\operatorname{proj}_{i_{-t},L-t} \circ \mathfrak{u}_{i_{-t+1},L-t+1} \circ \mathfrak{u}_{i_{-t+2},L-t+2} \circ \cdots \circ \mathfrak{u}_{i_0,L})(\diamond_{k=0}^{H_i+L-1}\vec{o}'_k).$$

QC: $(d^2 L \operatorname{mpe}(q-1) + dL^2, 0, 0, 0, 0)$.

4.9.1.4.2 For each $n = 0, 1, \ldots, N$, do the following.
QC: $(d^2 \operatorname{mpe}(q-1) 2^{d^2 \operatorname{mpe}(q-1)+d} + \log q, 0, 0, 0, 0)$.

4.9.1.4.2.1 If the logical sign tuple computed in step 4.9.1.4.1 is an element of S_{n,i_{-t},H_i} (the last entry of the tuple $S_{n,i_{-t}}$ from the partition-tree register computed in step 2), then set $\mathfrak{n}^{(-t)} := n$ and skip to the next t.
QC: $(\max\{1, |S_{n,i_{-t},H_i}| d^2 \operatorname{mpe}(q-1)\}, 0, 0, 0, 0)$ if the condition is *not* satisfied; $(\max\{1, |S_{n,i_{-t},H_i}| d^2 \operatorname{mpe}(q-1)\} + \log q, 0, 0, 0, 0)$ if the condition *is* satisfied (the additional $O(\log q)$ bit operations are from copying the value of n; we do not need to process the $O(\log q)$-bit indices n during the loop over them, because we may jump to a neighboring address in memory in $O(1)$ bit operations).

4.9.1.5 Set $\vec{\mathfrak{n}} := (\mathfrak{n}^{(-l+1)}, \mathfrak{n}^{(-l+2)}, \ldots, \mathfrak{n}^{(0)})$.
QC: $(L \log q, 0, 0, 0, 0)$.

4.9.1.6 Set l' to be $\operatorname{minperl}(\vec{\mathfrak{n}})$.
QC: $(L^{1+o(1)} \log q, 0, 0, 0, 0)$.

4.9.1.7 Set $\vec{\mathfrak{n}}'$ to be the lexicographically minimal ordered sequence in the same cyclic equivalence class as $(\mathfrak{n}^{(-l+1)}, \mathfrak{n}^{(-l+2)}, \ldots, \mathfrak{n}^{(-l+l')})$.
QC: $(L^2 \log L \log q, 0, 0, 0, 0)$.

4.9.1.8 Add

$$(\diamond_{k=0}^{H_i+L-1}\vec{o}'_k \diamond \vec{\nu}_{i,L,l}, [\vec{\mathfrak{n}}'], \mathfrak{m}_i(\diamond_{k=0}^{H_i+L-1}\vec{o}'_k \diamond \vec{\nu}_{i,L,l}))$$

to $\mathfrak{N}_i$ as a new element.
QC: $(d^2 \operatorname{mpe}(q-1) + dL + L \log q, 0, 0, 0, 0)$.

5 Set $\mathfrak{N} := \emptyset$.
QC: $(1, 0, 0, 0, 0)$.

6 For each $(i, \ell) \in \overline{\mathfrak{L}} \setminus \{(d, 1)\}$, do the following.
QC: $(d^4L^4 \operatorname{mpe}(q-1)4^{d^2 \operatorname{mpe}(q-1)+dL} + d^2L^2 4^{d^2 \operatorname{mpe}(q-1)+dL} \log q, 0, 0, 0, 0)$.

6.1 For each $(\diamond_{k=0}^{H_i+L-1} \vec{o}'_k \diamond \vec{\nu}_{i,L,l}, [\vec{\mathfrak{n}}], \mathfrak{m}') \in \mathfrak{N}_i$, do the following.
QC: $(d^3L^4 \operatorname{mpe}(q-1)4^{d^2 \operatorname{mpe}(q-1)+dL} + dL^2 4^{d^2 \operatorname{mpe}(q-1)+dL} \log q, 0, 0, 0, 0)$.

6.1.1 From the first entry, $\diamond_{k=0}^{H_i+L-1} \vec{o}'_k \diamond \vec{\nu}_{i,L,l}$, determine the binary representation of l.
QC: $(L \log L, 0, 0, 0, 0)$.

6.1.2 Check whether $[\vec{\mathfrak{n}}]$ and l already occur as the first two entries of some element of $\mathfrak{N}$. If so, skip to the next element of $\mathfrak{N}_i$.
QC: $(dL^2 2^{d^2 \operatorname{mpe}(q-1)+dL} \log q, 0, 0, 0, 0)$.

6.1.3 Compute the sum $\mathfrak{m}$ of the third entries of all triples in $\bigcup_{(j,\ell') \in \overline{\mathfrak{L}} \setminus \{(d,1)\}} \mathfrak{N}_j$ that have the parameters l and $[\vec{\mathfrak{n}}]$ associated with them.
QC: $(dL^2 2^{d^2 \operatorname{mpe}(q-1)+dL} \log q, 0, 0, 0, 0)$.

6.1.4 Add $([\vec{\mathfrak{n}}], l, \mathfrak{m})$ to $\mathfrak{N}$ as a new element.
QC: $(L \log q, 0, 0, 0, 0)$.

7 For each $n \in \{0, 1, \ldots, N\}$, do the following.
QC: $(d 2^{d^2 \operatorname{mpe}(q-1)+d} + \log q, 0, 0, 0, 0)$.

7.1 Check whether $S_{n,d} = \emptyset$. If so, set $\mathfrak{n} := n$ and exit the loop.
QC: $(\log q, 0, 0, 0, 0)$ if the condition is satisfied (which happens only once), $(1, 0, 0, 0, 0)$ otherwise.

8 Check whether $[\mathfrak{n}]$ and 1 already occur as the first two entries of some (unique) triple in $\mathfrak{N}$, and store this information (the truth value and, if applicable, the position of that triple in $\mathfrak{N}$).
QC: $(dL 2^{d^2 \operatorname{mpe}(q-1)+dL} \log q, 0, 0, 0, 0)$.

9 If $[\mathfrak{n}]$ and 1 occur as the first two entries of a triple $([\mathfrak{n}], 1, \mathfrak{m})$ in $\mathfrak{N}$, then do the following.

9.1 Overwrite the entry $([\mathfrak{n}], 1, \mathfrak{m})$ of the list $\mathfrak{N}$ with $([\mathfrak{n}], 1, \mathfrak{m} + 1)$.
QC: $(\log q, 0, 0, 0, 0)$.

10 Else do the following.

10.1 Add $([\mathfrak{n}], 1, 1)$ to $\mathfrak{N}$ as a new element.
QC: $(L \log q, 0, 0, 0, 0)$.

11 Sort $\mathfrak{N}$ lexicographically.
QC: $(dL^2(d^2 \operatorname{mpe}(q-1) + dL) 2^{d^2 \operatorname{mpe}(q-1)+dL} \log q, 0, 0, 0, 0)$.

12 Output $(\vec{\mathfrak{D}}, \mathfrak{N})$ and halt.
QC: $((d^2 4^{d^2 \operatorname{mpe}(q-1)+d} + dL^2 2^{d^2 \operatorname{mpe}(q-1)+dL}) \log q, 0, 0, 0, 0)$.

Next, we give pseudocode for the algorithm from Proposition 5.3.3.5, which for a given index d generalized cyclotomic mapping f of $\mathbb{F}_q$ outputs the maximum cycle length of f.

1 Compute the induced function $\bar{f}$ on $\{0, 1, \ldots, d\}$ and the affine maps A_i of $\mathbb{Z}/s\mathbb{Z}$.
QC: $(d \log^{1+o(1)} q, d, 0, 0, 0)$.

2 Compute a CRL-list $\bar{\mathcal{L}}$ of $\bar{f}$ and the cycles of $\bar{f}$.
QC: $(d \log^2 d, 0, 0, 0, 0)$.

3 Factor $s = p_1^{v_1} p_2^{v_2} \cdots p_K^{v_K}$.
QC: $(\log q, 0, 1, 0, 0)$.

4 For each $(i, \ell) \in \bar{\mathcal{L}} \setminus \{(d, 1)\}$, do the following.
QC: $(d \log^{2+o(1)} q, 0, 0, d, 0)$.

4.1 Compute $\mathcal{A}_i = A_{i_0} A_{i_1} \cdots A_{i_{\ell-1}} : x \mapsto \bar{\alpha}_i x + \bar{\beta}_i$.
QC: $(\ell \log^{1+o(1)} q, 0, 0, 0, 0)$.

4.2 Set $s_i' := 1$.
QC: $(\log d, 0, 0, 0, 0)$.

4.3 For each $j = 1, 2, \ldots, K$, do the following.
QC: $(\log^{2+o(1)} q, 0, 0, 0, 0)$.

4.3.1 Check whether $p_j \mid \bar{\alpha}_i$, and if so, skip to the next j.
QC: $(\log^{1+o(1)} q, 0, 0, 0, 0)$.

4.3.2 Set $s_i' := s_i' \cdot p_j^{v_j}$.
QC: $(\log^{1+o(1)} q, 0, 0, 0, 0)$.

4.4 Set $\bar{\alpha}_i' := \bar{\alpha}_i \bmod s_i'$ and $\bar{\beta}_i' := \bar{\beta}_i \bmod s_i'$.
QC: $(\log^{1+o(1)} q, 0, 0, 0, 0)$.

4.5 Compute $\operatorname{ord}_{s_i'}(\bar{\alpha}_i')$.
QC: $(\log q, 0, 0, 1, 0)$.

4.6 Compute

$$\bar{\beta}_i'' := \begin{cases} \bar{\beta}_i', & \text{if } \bar{\alpha}_i' = 1, \\ \bar{\beta}_i' \frac{(\bar{\alpha}_i')^{\operatorname{ord}_{s_i'}(\bar{\alpha}_i')} - 1}{\bar{\alpha}_i' - 1}, & \text{otherwise.} \end{cases}$$

In the second case, do *not* compute $(\bar{\alpha}_i')^{\operatorname{ord}_{s_i'}(\bar{\alpha}_i')}$ as an integer, but compute its value modulo $s_i'(\bar{\alpha}_i' - 1)$. Then the value modulo s_i' of the entire fraction can be determined through integer division.
QC: $(\log^{2+o(1)} q, 0, 0, 0, 0)$.

4.7 Compute the additive order of $\bar{\beta}_i''$ modulo s_i', which is equal to $s_i' / \gcd(s_i', \bar{\beta}_i'')$.
QC: $(\log^{1+o(1)} q, 0, 0, 0, 0)$.

4.8 Set $l_i := \ell \cdot \mathrm{ord}_{s_i'}(\overline{\alpha}_i') \cdot s_i' / \gcd(s_i', \overline{\beta}_i'')$.
QC: $(\log^{1+o(1)} q, 0, 0, 0, 0)$.

5 Set $L := 1$.
QC: $(\log d, 0, 0, 0, 0)$.

6 For each $(i, \ell) \in \overline{\mathfrak{L}} \setminus \{(d, 1)\}$, do the following.
QC: $(d \log q, 0, 0, 0, 0)$.

6.1 If $l_i > L$, then set $L := l_i$.
QC: $(\log q, 0, 0, 0, 0)$.

7 Output L and halt.
QC: $(\log q, 0, 0, 0, 0)$.

Finally, we give pseudocode for a variant of the algorithm from Corollary 5.3.3.6. On input (L, f_1, f_2), where $L \in \mathbb{N}^+$ and f_j, for $j = 1, 2$, is a generalized cyclotomic mapping of $\mathbb{F}_q$ of index d_j, this algorithm outputs "fail" if neither the largest cycle length of f_1 nor the largest cycle length of f_2 is at most L. Otherwise, it outputs the truth value of the digraph isomorphism relation $\Gamma_{f_1} \cong \Gamma_{f_2}$.

1 For $j = 1, 2$, compute the largest cycle length L_j of f_j.
QC: $(d \log^{2+o(1)} q + d \log^2 d, d, 1, d, 0)$.

2 Check whether $\min\{L_1, L_2\} \le L$, and store this information.
QC: $(\log q, 0, 0, 0, 0)$.

3 If $\min\{L_1, L_2\} > L$, then do the following.

3.1 Output "fail" and halt.
QC: $(1, 0, 0, 0, 0)$.

4 Else do the following.

4.1 Check whether $L_1 = L_2$. If not, output "false" and halt.
QC: $(\log q, 0, 0, 0, 0)$.

5 For $j = 1, 2$, compute

- a recursive tree description list $\vec{\mathfrak{D}}^{(j)} = (\mathfrak{D}_n^{(j)})_{n=0,1,\ldots,N_j}$ with

$$N_j \in O(d 2^{d^2 \operatorname{mpe}(q-1)+d})$$

and associated rooted tree isomorphism type sequence $\vec{\mathfrak{F}}^{(j)}$; and

- the tree necklace list $\mathfrak{N}_j$ of f_j relative to $\vec{\mathfrak{F}}^{(j)}$, with

$$|\mathfrak{N}_j| \le O(dL 2^{d^2 \operatorname{mpe}(q-1)+dL}).$$

QC: $(8^{d^2 \operatorname{mpe}(q-1)+dL} 2^{d \operatorname{mpe}(q-1)} (d^3 L \operatorname{mpe}(q-1) + d^2 L^2) \log^{1+o(1)} q, d, 0, 0, 0)$.

6 Compute a synchronization $(\vec{\mathfrak{D}}^+, \mathrm{i})$ of $\vec{\mathfrak{D}}^{(1)}$ and $\vec{\mathfrak{D}}^{(2)}$.
QC: $(d^3 8^{d^2 \operatorname{mpe}(q-1)+d} \log q + d^2 4^{d^2 \operatorname{mpe}(q-1)} \log^2 q, 0, 0, 0, 0)$.

7 Set $\mathfrak{N}_2' := \emptyset$.
QC: $(1, 0, 0, 0, 0)$.

8 For each $([\mathfrak{n}_1, \mathfrak{n}_2, \ldots, \mathfrak{n}_{l'}], l, \mathfrak{m}) \in \mathfrak{N}_2$, do the following.
QC: $(dL^3 \log L 2^{d^2 \operatorname{mpe}(q-1)+dL} \log q, 0, 0, 0, 0)$.

8.1 Compute the lexicographically minimal representative of the cyclic equivalence class $[\mathfrak{n}_1, \mathfrak{n}_2, \ldots, \mathfrak{n}_{l'}]$, representing that class by it.
QC: $(L^2 \log L \log q, 0, 0, 0, 0)$.

8.2 Add $([\mathfrak{i}(\mathfrak{n}_1), \mathfrak{i}(\mathfrak{n}_2), \ldots, \mathfrak{i}(\mathfrak{n}_{l'})], l, \mathfrak{m})$ to $\mathfrak{N}_2'$ as a new element.
QC: $(L \log q, 0, 0, 0, 0)$.

9 Sort $\mathfrak{N}_2'$ lexicographically.
QC: $(dL^2(d^2 \operatorname{mpe}(q-1) + dL)2^{d^2 \operatorname{mpe}(q-1)+dL} \log q, 0, 0, 0, 0)$.

10 Check whether $\mathfrak{N}_1 = \mathfrak{N}_2'$ as sets, output the truth value of this equality, and halt.
QC: $(dL^2 2^{d^2 \operatorname{mpe}(q-1)+dL} \log q, 0, 0, 0, 0)$.