Algorithmic Number Theory Symposium XVII, Groningen, July 6-10, 2026

DETERMINISTIC METHODS FOR FINDING ELEMENTS OF LARGE MULTIPLICATIVE ORDER

DAVID HARVEY AND MARKUS HITTMEIR

Abstract. We revisit the problem of rigorously and deterministically finding elements of large order in the multiplicative group of integers modulo a natural number N. Solving this problem is an essential step in several recent deterministic algorithms for factoring N, including the currently fastest ones. In 2018, the second author gave an algorithm that for a given target order $D \geqslant N^{2/5}$, finds either an element of order exceeding D, or a nontrivial divisor of N, or proves that N is prime. The running time was

$$O\left(\frac{D^{1/2}}{(\log\log D)^{1/2}}\log^2 N\right)$$

bit operations, asymptotically the same as the cost of computing the order of a *single* element using Sutherland's optimisation of the classical babystep-giantstep method. Subsequent work by several authors weakened the hypothesis $D \geqslant N^{2/5}$ to $D \geqslant N^{1/6}$. In this paper, we show that the hypothesis may be dropped altogether. Moreover, if N is prime, we can guarantee returning an element of order exceeding D, rather than a proof that N is prime.

1. Introduction

Let $N \geqslant 2$ be an integer and consider the multiplicative group $\mathbb{Z}_N^*$ of invertible residues modulo N. The order of an element $\alpha \in \mathbb{Z}_N^*$, denoted $\mathrm{ord}_N(\alpha)$, is defined to be the smallest positive integer k such that $\alpha^k = 1$.

In this paper we are interested in the following *large order problem*: given N and a positive integer D, find (if possible) an element of $\mathbb{Z}_N^*$ such that $\mathrm{ord}_N(\alpha) > D$. This problem appears in many practical applications, including for example cryptographic schemes based on the discrete logarithm problem [DH76, Elg85] and the generation of pseudorandom numbers [BM84].

For these and other real-world applications it is entirely reasonable to attack the large order problem via probabilistic and/or heuristic methods. However, in this paper we are mainly interested in *rigorous*, *deterministic* algorithms, with our principal motivation being the task of rigorous and deterministic *integer factorisation*. We will return to this application in a few moments, but let us first briefly discuss what can be said about the large order problem in a wider algorithmic context.

1.1. Heuristic and randomised algorithms.

If our goal is merely to find an element that with *high probability* has order larger than D, then we may take a *Monte Carlo* approach and simply select some $\alpha \in \mathbb{Z}_N^*$ at random. The success probability depends on the size of D and the density of elements with order larger

2020 *Mathematics Subject Classification.* Primary 11Y16, Secondary 11Y05.

The second author was supported by the Research Council of Norway (grant 357539).

than D. This procedure is often effective for small values of D, and even for larger D when $\phi(N)$ (the order of the group $\mathbb{Z}_N^*$) has few small prime divisors.

It is possible to boost the success probability considerably if the prime factorisation $N = p_1^{e_1} \cdots p_k^{e_k}$ is known. (This includes the case that N is itself prime.) Indeed, after computing $\phi(N)$ and finding all of its "small" prime divisors q, we may easily determine the q-part of $\mathrm{ord}_N(\alpha)$ for a given $\alpha \in \mathbb{Z}_N^*$ by computing $\alpha^{\phi(N)/q^j} \pmod{p_i^{e_i}}$ for various i and j. By sampling repeatedly, we can easily find an element $\alpha \in \mathbb{Z}_N^*$ with maximal q-power order for all of these q. For such α, it becomes overwhelmingly likely that it has maximal q-power order for the remaining (unknown but large) prime divisors q of $\phi(N)$.

If however our goal is to return an element of order larger than D with *certainty* (a *Las Vegas* algorithm), then the key issue is to be able to actually compute the order of elements, or otherwise certify that the order is large. If we know the complete prime factorisation of both N and $\phi(N)$, then we can compute orders of elements in (deterministic) polynomial time. Thus for example if N is prime, the probability of a random $\alpha \in \mathbb{Z}_N^*$ being a primitive root is $\phi(N-1)/(N-1) \gg 1/\log\log N$, so if we know the factorisation of $N-1$, we can find a primitive root in randomised polynomial time by sampling $\alpha \in \mathbb{Z}_N^*$ repeatedly until we succeed.

If we insist on deterministic algorithms, then we may be able to get away with heuristic assumptions as a substitute for randomness. For example, under the Extended Riemann Hypothesis it is known that for N prime there exists a primitive root less than $(\log N)^{O(1)}$ [Wan61, Sho90]. In this setting we can find a primitive root in deterministic polynomial time, again assuming that the factorisation of $N-1$ is known.

It is clear from the preceding discussion that if we want certified output, then regardless of whether we allow randomness or unproved hypotheses, the real bottleneck is the integer factorisation step. Of course, no polynomial time factorisation algorithm is known, even allowing probabilistic or heuristic methods. Algorithms such as the quadratic sieve and the general number field sieve [Rie94, Wag13] achieve heuristic runtime complexities subexponential in $\log N$. The latter can (conjecturally) find the factorisations of N and $\phi(N)$ in expected time

$$L_N\left[1/3, (64/9)^{1/3}\right] = \exp\left(\left((64/9)^{1/3} + o(1)\right)(\log N)^{1/3}(\log\log N)^{2/3}\right).$$

From a practical point of view, it is this last bound that determines the difficulty of the large order problem in its most general form.

1.2. **Rigorous and deterministic algorithms.** To motivate the approach to the large order problem taken in this paper, we will first explain the crucial role it has played in recent work on deterministic integer factorisation.

Until recently, the best complexity bounds for rigorously factoring an integer N were of the form $N^{1/4+o(1)}$, based on methods going back to Pollard [Pol74], Strassen [Str77] and Coppersmith [Cop97]. (Here, and for the rest of the paper, time complexity always refers to the number of steps performed by a deterministic multitape Turing machine. For a detailed description of this computational model, see [Pap94, Ch. 2] or [AHU75, §1.6].) The first improvement on these bounds by a superpolynomial factor (i.e., larger than any fixed power of $\log N$) appeared in [Hit18], and was based on the idea of solving the discrete logarithm problem $\alpha^x = \alpha^{N+1}$ in $\mathbb{Z}_N^*$ for a given element α. If N is a product of two distinct primes, say $N = pq$, then the solution $x = p + q$ immediately reveals the two factors of N.

However, we can only find this solution if $\operatorname{ord}_N(\alpha)$ is sufficiently large. Therefore, the paper also included an algorithm that, for $D \geqslant N^{2/5}$, either finds an element of order at least D, or a nontrivial divisor of N, or proves N prime [Hit18, Alg. 6.2]. Notice that this algorithm does not always actually solve the large order problem for (N, D); if N is composite, it is allowed to return a nontrivial divisor of N instead. This is of course entirely reasonable in the context of a factorisation algorithm.

The algorithm has two stages. In the first stage, we search for the order of small elements $\beta = 2, 3, \ldots$ via Sutherland's optimisation of the classical babystep-giantstep method [Sha71, Sut07], assuming that the order is at most D. If for some element we fail to compute the order, then the order exceeds D and we are done. If the order $m \coloneqq \operatorname{ord}_N(\beta)$ is actually found, we try to use it to compute a nontrivial divisor of N, by examining $\gcd(\beta^{m/r} - 1, N)$ for prime divisors r of m. If that fails, we deduce that m divides $p-1$ for every prime factor p of N. As we compute the exact orders of these small elements, and continue to fail to factorise N, we accumulate considerable information about the prime factors of N. Eventually, we recover sufficient information to proceed to the second stage of the algorithm, in which we attempt to factorise N directly. If that fails too, we obtain a proof that N is prime. It is noteworthy that the time complexity of this procedure is asymptotically the same as testing whether a single element has order at most D, namely

$$O\left(\frac{D^{1/2}}{(\log\log D)^{1/2}} \log^2 N\right). \tag{1.1}$$

The 2018 algorithm for finding elements of large order was later used as a subroutine in a series of works [Hit21, Har21, HH22] that improved the runtime of deterministic factorisation to $N^{1/5+o(1)}$. The original assumption $D \geqslant N^{2/5}$ was good enough to directly apply the original algorithm in these factorisation methods, but only just. The hypothesis on D was subsequently improved on two occasions, first to $D \geqslant N^{1/4+o(1)}$ by Gao, Feng, Hu and Pan [GFHP25, Lem. 3.5] and then to $D \geqslant N^{1/6}$ by Oznovich and Volk [OV26, Thm. 1.1]. These improvements are based on a better bound for the number of consecutive elements with the same order m (namely, $O(\sqrt{m})$ instead of m), leading to a faster progression in the loop over the elements $\beta = 2, 3, \ldots$ in the first stage of the algorithm. Both of these papers (and additionally [HH24]) also considered the case of N having an r-power divisor, which allows the assumption to be relaxed further to $D \geqslant N^{1/6r}$ [OV26, Thm. 4.2]. Furthermore, both [GFHP25] and [OV26] present improved techniques for the second stage, in which N is factorised based on a large known factor of $p-1$ for all primes $p \mid N$.

All of the results mentioned so far impose a fairly restrictive (in fact, fully exponential) lower bound hypothesis on D. The main result of this paper is an algorithm that solves the same problem in essentially the same amount of time, but without any restriction on D at all.

Theorem 1.1. *There exists an algorithm with the following properties. It takes as input integers $N \geqslant 3$ and $D \geqslant 1$ with $D < N - 1$. It outputs either some $\alpha \in \mathbb{Z}_N^*$ with $\operatorname{ord}_N(\alpha) > D$ or a nontrivial divisor of N. Its time complexity is*

$$O\left(\frac{D^{1/2} \log D}{(\log\log D)^{1/2}} \log N\right). \tag{1.2}$$

Note that the expression $\log\log D$ does not literally make sense for $D \leqslant 2$; in cases like this, the reader should mentally substitute $\log\log\max(D, 3)$. It can be shown that the space complexity in Theorem 1.1 is given by

$$O\left(\frac{D^{1/2}}{(\log\log D)^{1/2}}\log N\right),$$

but we will not carry out the details of this analysis.

The reader will notice a small improvement in time complexity between (1.1) and (1.2), namely that one factor of $\log N$ has been replaced by $\log D$. In the first version of this paper, Theorem 1.1 was stated with the previous bound (1.1) instead of (1.2). An anonymous referee subsequently pointed out that the analysis of Sutherland's order-finding algorithm could be improved slightly in the Turing model (see Lemma 2.1), leading to the tighter bound (1.2).

We point out several interesting features of this result:

- Unlike its predecessors, this algorithm never returns "N is prime". In particular, if N is prime, it *always* returns $\alpha \in \mathbb{Z}_N^*$ with $\mathrm{ord}_N(\alpha) > D$ (and thus solves the large order problem as originally stated). This is interesting from a theoretical point of view, as it permits rigorously finding not only primitive roots but also elements of order exceeding D, even for small values of D. (When D is sufficiently large, namely for $D \geqslant N^{1/2+\varepsilon}$, it was already known how to find elements of order at least D in time $O(D^{1/2+\varepsilon})$, as one can find a primitive root in time $O(N^{1/4+\varepsilon})$ [Shp96].) It is also worth mentioning that when D is very small, the runtime may be even less than the cost of testing N for primality via a deterministic polynomial time primality test, such as the AKS test [AKS04].
- An important consequence of the theorem is that in the context of *any* deterministic factoring algorithm that runs in exponential time, finding elements of large order should no longer be considered a bottleneck, regardless of the exponent.
- Theorem 1.1 outperforms the (heuristic) approach mentioned in Section 1.1 whenever
$$D \ll L_N\left[1/3, 2(64/9)^{1/3}\right].$$
For such inputs D, we are not aware of a general-purpose Las Vegas algorithm that, without obtaining and using the prime factorisation of N, certifies $\mathrm{ord}_N(\alpha) > D$ faster than the deterministic bound given in (1.2).

The main idea behind the improved algorithm is based on estimates for the density of integers that factor completely into primes $q \leqslant B$ for a given bound B, the so-called *B-smooth numbers*. The algorithm has a broadly similar structure to its predecessors. In the first stage, we attempt to compute the order of all elements $\beta = 2, 3, \ldots, B$, for a suitable choice of B. If we have not yet found an element of large order, or a factor of N via GCDs as before, then we have computed some $M \leqslant D$ such that all these elements satisfy the congruence $\beta^M \equiv 1 \pmod{p}$ for every prime factor p of N, and such that $M \mid p-1$ for every such p. The key observation is that not only these β, but *all B-smooth integers* in the interval $1 \leqslant n \leqslant p$ satisfy $n^M \equiv 1 \pmod{p}$. But the latter congruence has at most M solutions modulo p, so by applying well-known lower bounds for the density of B-smooth numbers, we obtain drastically improved estimates for how fast M grows as we progress through the main loop over β. This strategy incidentally leads to a

considerable simplification in the second stage of the algorithm: it suffices to use trial division to check for divisors along the arithmetic progression $kM + 1$ instead of applying more elaborate techniques.

The rest of the paper is structured as follows. Section 2 discusses preliminaries such as the babystep-giantstep method for finding orders of elements, and bounds for the density of B-smooth numbers. Then the main theorem is proved in Section 3.

Remark 1.2. After the initial version of our paper was made public, we became aware of independent concurrent work by Itamar Nir, which was subsequently posted on arXiv [Nir26]. Nir obtains a result similar to but weaker than our Theorem 1.1, the main difference being that it requires the *subexponential* bound $D > \exp(\sqrt{2 \log N \log\log N})$. His argument makes similar use of smoothness bounds, and according to Nir is simpler than our proof of Theorem 1.1.

Acknowledgments. The authors would like to thank the anonymous referee mentioned earlier, whose comments led to the tightening of the bound in Theorem 1.1, and also the other anonymous referees whose suggestions helped to improve the presentation of these results.

2. Preliminaries

We sometimes use the notation $f \ll g$ as a synonym for $f = O(g)$. The notation $f \asymp g$ means that both $f \ll g$ and $g \ll f$ hold.

We will frequently use standard results on the complexity of integer arithmetic: we may compute sums and differences of n-bit integers in time $O(n)$, products in time $O(n \log n)$ [HvdH21], quotients and remainders in time $O(n \log n)$ [vzGG13, Ch. 9], k-th roots with remainder for fixed k in time $O(n \log n)$ [BZ11, §1.5], and greatest common divisors (including finding the corresponding cofactors in the Bézout identity) in time $O(n \log^2 n)$ [vzGG13, Ch. 11].

Let $N \geqslant 2$. Elements of the ring $\mathbb{Z}_N$ are always represented by their residue in the interval $[0, N)$. Given $\alpha \in \mathbb{Z}_N$ and an integer $n \geqslant 1$, we may compute α^n in time $O(\log n \log N \log\log N)$ using the repeated squaring method [vzGG13, §4.3].

We will occasionally need to compute approximations for logarithms and exponentials of real numbers. We use standard algorithms to approximate these functions to n-bit precision in time $O(n \log^2 n)$ [BZ11, §4.8], omitting the straightforward but tedious analysis of rounding errors.

Whenever we need to store the prime factorisation of a positive integer n, say $n = \prod_i q_i^{e_i}$, we always represent it by the list of pairs (q_i, e_i), with the q_i in increasing order and all $e_i \geqslant 1$. This representation occupies space $O(\log n)$.

At the heart of all our algorithms is a minor variant of Sutherland's well-known method for computing the order of an element of a group.

Lemma 2.1. *Given as input integers $N \geqslant 2$, $D \geqslant 1$ and an element $\alpha \in \mathbb{Z}_N^*$, we may determine if $\operatorname{ord}_N(\alpha) \leqslant D$, and if so compute the exact order $m \geqslant 1$, in time*

$$O\left(\frac{r^{1/2}}{(\log\log r)^{1/2}}(\log r + \log\log N)\log N\right), \qquad \text{where } r \coloneqq \min(m, D).$$

Proof. According to [Sut07, Alg. 4.1], we may search for the order of α up to a given $T \geqslant 1$ by performing $O(T^{1/2}(\log\log T)^{-1/2})$ group operations in $\mathbb{Z}_N^*$ plus a similar number of reads/writes to a lookup table, e.g., a hash table. One cannot implement hash tables efficiently on a Turing machine, but it is shown in [Hit18,

Thm. 6.1] (see also [Hit18, Rem. 2.5]) that by replacing the use of lookup tables by sorted lists and the merge sort algorithm, one can solve this problem in

$$O\left(\frac{T^{1/2}}{(\log\log T)^{1/2}}\log^2 N\right)$$

bit operations on a Turing machine.

As pointed out by an anonymous referee, this bound can be improved slightly in the following way. Let $n \coloneqq T^{1/2}(\log\log T)^{-1/2}$. We may perform $O(n)$ group operations in $\mathbb{Z}_N^*$ using $O(n \log N \log\log N)$ bit operations, and we may sort a list of $O(n)$ elements of $\mathbb{Z}_N^*$ using $O(n \log N \log n) = O(n \log N \log T)$ bit operations. Inserting these estimates into the argument in [Hit18, Thm. 6.1], we conclude that we may search for the order of α up to T within

$$O\left(\frac{T^{1/2}}{(\log\log T)^{1/2}}(\log T + \log\log N)\log N\right)$$

bit operations.

We apply this result successively for $T = 1, 2, 4, \ldots, 2^{\lceil \log_2 D\rceil}$, terminating immediately if we find m. If m is not found, we must have $m > 2^{\lceil \log_2 D\rceil} \geqslant D$, and we return "$\mathrm{ord}_N(\alpha) > D$". The overall time complexity bound follows from the estimate

$$\begin{aligned}\sum_{i=0}^{\lceil \log_2 r\rceil} 2^{i/2}(\log\log(2^i))^{-1/2} &\ll \sum_{i=0}^{\lceil (\log_2 r)/2\rceil} 2^{i/2} + \sum_{i=\lceil (\log_2 r)/2\rceil+1}^{\lceil \log_2 r\rceil} 2^{i/2}(\log i)^{-1/2} \\ &\ll r^{1/4} + r^{1/2}(\log\log r)^{-1/2} \\ &\ll r^{1/2}(\log\log r)^{-1/2}.\end{aligned}$$

□

Lemma 2.2. *Let $N \geqslant 2$. Given as input elements $\alpha, \beta \in \mathbb{Z}_N^*$, and their orders $u \coloneqq \mathrm{ord}_N(\alpha)$ and $v \coloneqq \mathrm{ord}_N(\beta)$, together with the prime factorisations of u and v, we may compute an element $\gamma \in \mathbb{Z}_N^*$ of order $w \coloneqq \mathrm{lcm}(u, v)$, together with the prime factorisation of w, in time*

$$O((\log u + \log v)\log N \log\log N).$$

Proof. Let $u = \prod_i q_i^{e_i}$ and $v = \prod_i q_i^{f_i}$ be the prime factorisations of u and v (temporarily allowing $e_i = 0$ and $f_i = 0$ so that we can use the same list of primes for both u and v). Set $\tilde{\alpha} \coloneqq \alpha^s$ and $\tilde{\beta} \coloneqq \beta^t$ where

$$s \coloneqq \prod_{e_i < f_i} q_i^{e_i}, \qquad t \coloneqq \prod_{e_i \geqslant f_i} q_i^{f_i}.$$

Then $\mathrm{ord}_N(\tilde{\alpha}) = \prod_{e_i \geqslant f_i} q_i^{e_i}$ and $\mathrm{ord}_N(\tilde{\beta}) = \prod_{e_i < f_i} q_i^{f_i}$. Note that $\mathrm{ord}_N(\tilde{\alpha})$ and $\mathrm{ord}_N(\tilde{\beta})$ are coprime and their product is equal to $w = \mathrm{lcm}(u, v)$. Hence $\gamma \coloneqq \tilde{\alpha}\tilde{\beta}$ meets our requirements.

Computing $\tilde{\alpha} = \alpha^s$ via the standard binary powering algorithm requires time

$$O(\log s \log N \log\log N) = O(\log u \log N \log\log N).$$

This bound also covers the cost of computing s itself, which requires $O(\log u)$ multiplications of bit size $O(\log N)$. The analysis for $\tilde{\beta}$ is similar. □

Lemma 2.3. *Let $N \geqslant 2$ and $\beta \in \mathbb{Z}_N^*$, and put $m \coloneqq \operatorname{ord}_N(\beta)$. Then*

$$\operatorname{ord}_p(\beta) = m \qquad \text{for all primes } p \text{ dividing } N$$

if and only if

$$\gcd(\beta^{m/r} - 1, N) = 1 \qquad \text{for all primes } r \text{ dividing } m.$$

Proof. This is [Hit18, Lem. 2.3]. □

Recall that a positive integer n is said to be *y-smooth* if every prime factor q of n satisfies $q \leqslant y$. For $x \geqslant 1$ we denote by $\Psi(x, y)$ the number of positive integers $n \leqslant x$ that are y-smooth. We will need the following lower bound for $\Psi(x, y)$.

Lemma 2.4. *For $x \geqslant y \geqslant 2$ and $x \geqslant 4$ we have*

$$\Psi(x, y) \geqslant \frac{x}{(\log x)^{\log x / \log y}}.$$

Proof. This is [KP97, §2, Thm. 1]. □

For any prime p, let $g(p)$ denote the least primitive root modulo p.

Lemma 2.5. *For any $\varepsilon > 0$ we have $g(p) \ll p^{1/4+\varepsilon}$.*

Proof. See [Bur62]. □

3. Proof of the main theorem

In this section we prove Theorem 1.1. Recall that the input consists of integers $N \geqslant 3$ and $D \geqslant 1$ with $D < N - 1$. Pseudocode for the algorithm is given in Algorithm 3.1. In Section 3.1 we prove that the output of the algorithm is correct, and in Section 3.2 we analyse its complexity.

In what follows, various arguments will require N or D to be "sufficiently large". We assume that N_0 and D_0 are absolute constants chosen so that these arguments are valid for all $N \geqslant N_0$ and $D \geqslant D_0$. Note that after line 1 we have $N \geqslant N_0$. Assuming we choose $N_0 \geqslant 2^{D_0}$, then after line 3 we also have $D \geqslant \log_2 N \geqslant D_0$.

3.1. Proof of correctness.

A few cases are easily dealt with:

- If the algorithm terminates in lines 1–3, the output is clearly correct. (In line 1, if N is composite, we return any nontrivial divisor, and if N is prime, we return a primitive root modulo N, which has order $N - 1 > D$.)
- For sufficiently large N and D we have $B < D^{1/3} + 1 < N^{1/3} + 1 < N$. Hence if the algorithm terminates in line 7, the output is correct.
- If the algorithm terminates in line 10, the output is clearly correct.
- The GCD in line 13 can never be equal to N, because the definition of m ensures that $\beta^{m/r} \not\equiv 1 \pmod{N}$. Thus if the algorithm terminates in this line, the output is correct.
- It is clear that the main for-loop maintains the invariant $M = \operatorname{ord}_N(\alpha)$ (thanks to lines 5, 9, 14 and 15), so if the algorithm terminates in 16, the output is correct.
- If N is prime and sufficiently large, and if $D > N^{9/10}$, then in some iteration the algorithm will terminate correctly in line 10. Indeed, Lemma 2.5 implies that the least primitive root modulo N satisfies $g(N) < N^{3/10}$ for sufficiently large N. If $D > N^{9/10}$ then $B > N^{3/10}$, so line 10 is guaranteed to find a primitive root, which has order $N - 1 > D$.

Algorithm 3.1 The main algorithm

Input: Integers $N \geqslant 3$ and $D \geqslant 1$ with $D < N - 1$.
Output: Either some $\alpha \in \mathbb{Z}_N^*$ with $\mathrm{ord}_N(\alpha) > D$, or a nontrivial divisor $d \mid N$.

– *See proof of Theorem 1.1 for the definition of the constant N_0.*
1: **if** $N < N_0$ **then** return suitable α or d via a lookup table.
2: **if** N is even **then** return $d := 2$.
3: **if** $2^D < N$ **then** return $\alpha := 2$.
4: Set $B := \lceil D^{1/3} \rceil$.
– *On entry to each iteration of the for-loop below, M is equal to $\mathrm{ord}_N(\alpha)$.*
– *The factorisation of M is always known and stored alongside M.*
5: Set $\alpha := 1$ and $M := 1$.
6: **for** $\beta = 2, 3, \ldots, B$ **do**
7: **if** $\beta \mid N$ **then** return $d := \beta$.
8: **if** $\beta^M \equiv 1 \pmod{N}$ **then** go to line 6 (proceed to next value of β).
9: Use Lemma 2.1 to search for $m := \mathrm{ord}_N(\beta)$ up to D.
10: **if** $m > D$ **then** return $\alpha := \beta$.
11: Compute the prime factorisation of m.
12: **for** each prime $r \mid m$ **do**
13: **if** $d := \gcd(\beta^{m/r} - 1, N) \neq 1$ **then** return d.
14: Apply Lemma 2.2 to α and β to compute α' of order $M' := \mathrm{lcm}(m, M)$.
15: Set $\alpha := \alpha'$ and $M := M'$.
16: **if** $M > D$ **then** return α.
17: Compute an integer Z such that $\tilde{Z} < Z < \tilde{Z} + 2$ where
$$\tilde{Z} := 2M \cdot (\log 2M)^{(\log 2M)/(-1+\log B)}.$$
18: **for** $k = 1, 2, \ldots, \lfloor Z/M \rfloor$ **do**
19: **if** $d := kM + 1 \mid N$ **then** return d
– *The proof of Theorem 1.1 shows that this line is never reached.*

Assume now that the algorithm has reached line 17. It remains to prove that the loop in lines 18–19 will succeed in finding a nontrivial divisor $d \mid N$. For the rest of this section, the symbol M refers to the value of M at the point that line 17 is reached. Note that $M \leqslant D$ due to line 16.

We claim that every prime divisor p of N satisfies

$$p \equiv 1 \pmod{M}, \tag{3.1}$$

$$\Psi(p, B) \leqslant M. \tag{3.2}$$

Since no factors were found in line 13 in any iteration of the for-loop, Lemma 2.3 implies that $\mathrm{ord}_p(\beta) = m$ at each iteration, and hence that $m \mid p - 1$. Since M is the LCM of these values of m, we have $M \mid p - 1$, i.e., (3.1) holds. Next, line 7 ensures that $p > B$, and lines 8, 9 and 14 ensure that $\beta^M \equiv 1 \pmod{N}$ for all positive integers $\beta \leqslant B$. It follows that $n^M \equiv 1 \pmod{p}$ for all positive integers $n \leqslant p$ that are B-smooth. The number of such integers n is by definition $\Psi(p, B)$. On the other hand, since p is prime there are at most M solutions to the congruence $n^M \equiv 1 \pmod{p}$. Thus (3.2) must hold.

Our main task is now to use (3.2) to show that

$$p \leqslant Z, \tag{3.3}$$

where Z is defined as in line 17. Note that the definition of $\tilde{Z}$ makes sense, as both M and B are bounded well away from zero; for example $M \geqslant \Psi(p, B) \geqslant B \geqslant 4$ for sufficiently large D.

To prove (3.3), it is enough to show that

$$\Psi(\tilde{Z}, B) > M. \tag{3.4}$$

For if instead we had $p > Z$, then since $\Psi(x, B)$ is non-decreasing in x, we would have $\Psi(p, B) \geqslant \Psi(Z, B) \geqslant \Psi(\tilde{Z}, B) > M$, but this contradicts (3.2).

Applying Lemma 2.4 to $\Psi(\tilde{Z}, B)$ (this is valid as $\tilde{Z} > 2M > B \geqslant 4$) yields

$$\Psi(\tilde{Z}, B) \geqslant \frac{\tilde{Z}}{(\log \tilde{Z})^{\log \tilde{Z}/\log B}}.$$

Thus to prove (3.4) it suffices to show that

$$\frac{\tilde{Z}}{(\log \tilde{Z})^{\log \tilde{Z}/\log B}} > M.$$

In fact, we will prove the more precise statement that

$$M < \frac{\tilde{Z}}{(\log \tilde{Z})^{\log \tilde{Z}/\log B}} < 4M. \tag{3.5}$$

To prove (3.5), let

$$z \coloneqq \log \tilde{Z} = \left(1 + \frac{\log\log 2M}{-1 + \log B}\right) \log 2M. \tag{3.6}$$

Since $M \leqslant D$ and $\log B \asymp \log D$ we have

$$\frac{\log\log 2M}{-1 + \log B} \ll \frac{\log\log D}{\log D}. \tag{3.7}$$

So by using the estimate $\log(1 + x) = x + O(x^2)$, for sufficiently large D we obtain

$$\begin{aligned} \log z &= \frac{\log\log 2M}{-1 + \log B} + \log\log 2M + O\left(\frac{(\log\log D)^2}{\log^2 D}\right) \\ &= \frac{\log B}{-1 + \log B} \cdot \log\log 2M + O\left(\frac{(\log\log D)^2}{\log^2 D}\right). \end{aligned}$$

Thus

$$1 - \frac{\log z}{\log B} = 1 - \frac{\log\log 2M}{-1 + \log B} + O\left(\frac{(\log\log D)^2}{\log^3 D}\right),$$

and it follows that

$$\begin{aligned} z\left(1 - \frac{\log z}{\log B}\right) &= \left(1 + \frac{\log\log 2M}{-1 + \log B}\right)\left(1 - \frac{\log\log 2M}{-1 + \log B} + O\left(\frac{(\log\log D)^2}{\log^3 D}\right)\right) \log 2M \\ &= \left(1 - \frac{(\log\log 2M)^2}{(-1 + \log B)^2} + O\left(\frac{(\log\log D)^2}{\log^3 D}\right)\right) \log 2M \\ &= \left(1 + O\left(\frac{(\log\log D)^2}{\log^2 D}\right)\right) \log 2M \end{aligned}$$

$$= \log 2M + O\left(\frac{(\log\log D)^2}{\log D}\right).$$

For sufficiently large D, this implies that

$$\log M < z\left(1 - \frac{\log z}{\log B}\right) < \log 4M,$$

which is equivalent to (3.5). This completes the proof of (3.3).

At this point we know that every prime p dividing N satisfies (3.1) and (3.3). The final loop in lines 18–19 simply tests every possible candidate. If N is composite, the loop will correctly terminate when it reaches the smallest prime divisor of N. (If we let it run to completion, it will actually find all prime divisors of N.)

Now suppose that N is prime. As shown earlier, if $D > N^{9/10}$, then the algorithm would have already terminated in line 10. Assume therefore that $D \leqslant N^{9/10}$. From (3.7) we see that for large enough D we have $z \leqslant \frac{19}{18}\log 2M \leqslant \frac{19}{18}\log 2D$, and then

$$Z < \tilde{Z} + 2 \leqslant (2D)^{19/18} + 2 \ll D^{19/18} \leqslant (N^{9/10})^{19/18} = N^{19/20},$$

so $Z < N$ for sufficiently large N. But the only prime divisor of N is $p = N$, which manifestly cannot satisfy (3.3) if $Z < N$. We have reached a contradiction; in other words, if N is prime, the algorithm can never reach line 17, and must have already terminated (with a correct output) during the main for-loop.

3.2. **Complexity analysis.** Lines 1–3 run in time $O(\log N)$. After line 3 we have $D \geqslant \log_2 N$, so we may henceforth assume that $\log N \ll D$. The computation of B in line 4 requires time $O(\log N \log\log N) = O(\log N \log D)$.

Consider a single iteration of the for-loop. If we have $\beta^M \equiv 1 \pmod N$ in line 8, the iteration is aborted and its complexity is $O(\log M \log N \log\log N)$. Since $M \leqslant D$ for all of these iterations, their total cost is $O(B \log D \log N \log\log N) = O(D^{1/3}\log^2 D \log N)$, which is negligible compared to (1.2).

We next consider the complexity of an iteration that reaches the search for m in line 9. If $m > D$, Lemma 2.1 implies that the algorithm terminates in line 10 within the cost given by (1.2). We hence assume that $m \leqslant D$. The complexity of line 9 is then

$$O\left(\frac{m^{1/2}}{(\log\log m)^{1/2}} \log D \log N\right). \tag{3.8}$$

In line 11, we find all divisors of m via trial division in time $O(m^{1/2}\log m \log\log m)$, which is negligible compared to (3.8). In line 13, the cost of computing each power $\beta^{m/r} \pmod N$ is $O(\log m \log N \log\log N)$, and each GCD requires time $O(\log N\,(\log\log N)^2)$. The number of primes r dividing m is $O(\log m)$, so the cost of lines 12–13 is at most $O(\log^2 m \log N\,(\log\log N)^2) = O(\log^4 D \log N)$. The total over all iterations is $O(B\log^4 D \log N) = O(D^{1/3}\log^4 D \log N)$ which is negligible compared to (1.2). According to Lemma 2.2, line 14 runs in time $O((\log m + \log M)\log N \log\log N) = O((\log m + \log M)\log D \log N)$. The first summand is bounded by (3.8). Since $M \leqslant D$, the combined cost of the second summand over all iterations is $O(B\log^2 D \log N) = O(D^{1/3}\log^2 D \log N)$, which is negligible compared to (1.2). We have now shown that in any iteration of the for-loop that reaches line 9, the cost of each step is either bounded by (3.8), or is negligible when summed over all iterations. It thus remains to estimate the sum of (3.8) over all iterations.

Since we have assumed that $m \leqslant D$, for each such iteration we must have

$$\frac{D}{2^l} < m \leqslant \frac{D}{2^{l-1}} \tag{3.9}$$

for some $l \in \{1, 2, \ldots, \lceil \log_2 D \rceil\}$. We claim that for each possible l, the number of iterations for which (3.9) holds with the given l is at most $l+1$. To see this, first observe that line 8 ensures that $m \nmid M$ and hence that M increases by a factor of at least 2 in each iteration that reaches line 9. On the first iteration that (3.9) occurs for a given l, we certainly have $M > D/2^l$ after line 15. After this point, the inequality $M \leqslant D$ can hold for at most l more iterations, because line 16 terminates the algorithm as soon as $M > D$ occurs. Therefore (3.9) can occur at most $l+1$ times as claimed.

Summing the total complexity over powers of two, we obtain

$$\sum_{l=1}^{\lceil \log_2 D \rceil} \frac{(l+1)(D/2^{l-1})^{1/2}}{(\log\log(D/2^{l-1}))^{1/2}} \log D \log N \ll D^{1/2} \log D \log N \sum_{l=1}^{\lceil \log_2 D \rceil} \frac{l/2^{l/2}}{(\log\log(D/2^{l-1}))^{1/2}}.$$

To estimate the inner sum, observe that

$$\begin{aligned}
&\sum_{l=1}^{\lceil \log_2 D \rceil} \frac{l/2^{l/2}}{(\log\log(D/2^{l-1}))^{1/2}} \\
&\quad \ll \sum_{l=1}^{\lceil (\log_2 D)/2 \rceil - 1} \frac{l/2^{l/2}}{(\log\log(D/2^{l-1}))^{1/2}} + \sum_{l=\lceil (\log_2 D)/2 \rceil}^{\lceil \log_2 D \rceil} l/2^{l/2} \\
&\quad \ll \sum_{l=1}^{\lceil (\log_2 D)/2 \rceil - 1} \frac{l/2^{l/2}}{(\log\log(D^{1/2}))^{1/2}} + D^{-1/4+o(1)} \ll (\log\log D)^{-1/2}.
\end{aligned}$$

Thus the total contribution of (3.8) over all iterations is bounded by (1.2).

Finally we consider the complexity of lines 17–19. From (3.7) we have $z \ll \log M \leqslant \log D \leqslant \log N$ and hence $\tilde{Z} < N^{O(1)}$. Using the formula (3.6), we may approximate z and then $\tilde{Z} = e^z$ with $O(\log N)$ bits of precision in time $O(\log N (\log\log N)^2) = O(\log^2 D \log N)$. Thus the cost of computing Z in line 17 is $O(\log^2 D \log N)$. The cost of the loop in lines 18–19 is $O((Z/M) \log N \log\log N) = O((Z/M) \log D \log N)$. We have

$$\frac{Z}{M} \ll \frac{\tilde{Z}}{2M} = (\log 2M)^{(\log 2M)/(-1+\log B)}.$$

Taking logarithms,

$$\log(Z/M) < \frac{\log 2M \log\log 2M}{-1+\log B} + O(1) \ll \frac{\log D \log\log D}{\log D} = \log\log D.$$

This implies that $Z/M < (\log D)^{O(1)}$, so the cost of the loop is

$$(\log D)^{O(1)} \log N,$$

which is negligible compared to (1.2).

References

[AHU75] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley Series in Computer Science and Information Processing, Addison-Wesley Publishing Co., Reading, Mass.-London-Amsterdam, 1975, Second printing. MR 413592

[AKS04] M. Agrawal, N. Kayal, and N. Saxena, *PRIMES is in P*, Ann. of Math. (2) **160** (2004), no. 2, 781–793. MR 2123939

[BM84] M. Blum and S. Micali, *How to generate cryptographically strong sequences of pseudorandom bits*, SIAM Journal on Computing **13** (1984), no. 4, 850–864.

[Bur62] D. A. Burgess, *On character sums and primitive roots*, Proc. London Math. Soc. (3) **12** (1962), 179–192. MR 132732

[BZ11] R. P. Brent and P. Zimmermann, *Modern Computer Arithmetic*, Cambridge Monographs on Applied and Computational Mathematics, vol. 18, Cambridge University Press, Cambridge, 2011. MR 2760886

[Cop97] D. Coppersmith, *Small solutions to polynomial equations, and low exponent RSA vulnerabilities*, J. Cryptology **10** (1997), no. 4, 233–260. MR 1476612

[DH76] W. Diffie and M. E. Hellman, *New directions in cryptography*, IEEE Transactions on Information Theory **22** (1976), no. 6, 644–654.

[Elg85] T. Elgamal, *A public key cryptosystem and a signature scheme based on discrete logarithms*, IEEE Transactions on Information Theory **31** (1985), no. 4, 469–472.

[GFHP25] Y. Gao, Y. Feng, H. Hu, and Y. Pan, *On factoring and power divisor problems via rank-3 lattices and the second vector*, Cryptology ePrint Archive, `https://eprint.iacr.org/2025/1004`, to appear in Mathematics of Computation, `https://doi.org/10.1090/mcom/4188`, 2025.

[Har21] D. Harvey, *An exponent one-fifth algorithm for deterministic integer factorisation*, Mathematics of Computation **90** (2021), no. 332, 2937–2950.

[HH22] D. Harvey and M. Hittmeir, *A log-log speedup for exponent one-fifth deterministic integer factorisation*, Mathematics of Computation **91** (2022), no. 335, 1367–1379.

[HH24] J. Hales and G. Hiary, *A generalization of Lehman's method*, The Ramanujan Journal **65** (2024), no. 4, 1773–1790.

[Hit18] M. Hittmeir, *A babystep-giantstep method for faster deterministic integer factorization*, Math. Comp. **87** (2018), no. 314, 2915–2935. MR 3834692

[Hit21] ———, *A time-space tradeoff for Lehman's deterministic integer factorization method*, Mathematics of Computation **90** (2021), no. 330, 1999–2010.

[HvdH21] D. Harvey and J. van der Hoeven, *Integer multiplication in time $O(n \log n)$*, Ann. of Math. (2) **193** (2021), no. 2, 563–617. MR 4224716

[KP97] S. Konyagin and C. Pomerance, *On primes recognizable in deterministic polynomial time*, The mathematics of Paul Erdős, I, Algorithms Combin., vol. 13, Springer, Berlin, 1997, pp. 176–198. MR 1425185

[Nir26] I. Nir, *Deterministically finding an element of large order in $\mathbb{Z}_n^*$*, preprint at `https://arxiv.org/abs/2605.09592`, 2026.

[OV26] Z. Oznovich and B. L. Volk, *On deterministically finding an element of high order modulo a composite*, Proceedings of the 2026 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), Society for Industrial and Applied Mathematics, 2026, pp. 6379–6391.

[Pap94] C. H. Papadimitriou, *Computational Complexity*, Addison-Wesley Publishing Company, Reading, MA, 1994. MR 1251285 (95f:68082)

[Pol74] J. M. Pollard, *Theorems on factorization and primality testing*, Mathematical Proceedings of the Cambridge Philosophical Society **76** (1974), no. 3, 521–528.

[Rie94] H. Riesel, *Prime Numbers and Computer Methods for Factorization*, second ed., Progress in Mathematics, vol. 126, Birkhäuser Boston, Inc., Boston, MA, 1994. MR 1292250

[Sha71] D. Shanks, *Class number, a theory of factorization, and genera*, 1969 Number Theory Institute (Proc. Sympos. Pure Math., Vol. XX, State Univ. New York, Stony Brook, N.Y., 1969), 1971, pp. 415–440. MR 0316385

[Sho90] V. Shoup, *Searching for primitive roots in finite fields*, Proceedings of the Twenty-Second Annual ACM Symposium on Theory of Computing (New York, NY, USA), STOC '90, Association for Computing Machinery, 1990, pp. 546–554.

[Shp96] I. Shparlinski, *On finding primitive roots in finite fields*, Theoret. Comput. Sci. **157** (1996), no. 2, 273–275. MR 1389773

[Str77] V. Strassen, *Einige Resultate über Berechnungskomplexität*, Jber. Deutsch. Math.-Verein. **78** (1976/77), no. 1, 1–8. MR 0438807 (55 #11713)

[Sut07] A.V. Sutherland, *Order computations in generic groups*, Thesis (Ph.D.)–Massachusetts Institute of Technology, ProQuest LLC, Ann Arbor, 2007.

[vzGG13] J. von zur Gathen and J. Gerhard, *Modern Computer Algebra*, 3rd ed., Cambridge University Press, Cambridge, 2013. MR 3087522

[Wag13] S. S. Wagstaff, Jr., *The Joy of Factoring*, Student Mathematical Library, vol. 68, American Mathematical Society, Providence, RI, 2013. MR 3135977

[Wan61] Y. Wang, *On the least primitive root of a prime*, Scientia Sinica **10** (1961), 1–14.

School of Mathematics and Statistics, University of New South Wales, Sydney NSW 2052, Australia

Email address: d.harvey@unsw.edu.au

NORCE Research, Nygårdsgaten 112, 5008 Bergen, Norway

Email address: mahi@norceresearch.no