MATHEMATICS OF COMPUTATION
https://doi.org/10.1090/mcom/3438
Article electronically published on April 23, 2019

AN IMPROVED SIEVE OF ERATOSTHENES

HARALD ANDRÉS HELFGOTT

Abstract. We show how to carry out a sieve of Eratosthenes up to N in space $O\left(N^{1/3}(\log N)^{2/3}\right)$ and time $O(N \log N)$. In comparison, the usual versions of the sieve take space about $O(\sqrt{N})$ and time at least linear on N. We can also apply our sieve to any subinterval of $[1, N]$ of length $\Omega\left(N^{1/3}\right)$ in time close to linear on the length of the interval. Before, such a thing was possible only for subintervals of $[1, N]$ of length $\Omega(\sqrt{N})$.

Just as in (Galway, 2000), the approach here is related to Diophantine approximation, and also has close ties to Voronoï's work on the Dirichlet divisor problem. The advantage of the method here resides in the fact that, because the method we will give is based on the sieve of Eratosthenes, we will also be able to use it to factor integers, and not just to produce lists of consecutive primes.

1. Introduction

The sieve of Eratosthenes is a procedure for constructing all primes up to N. More generally, such a sieve can be used for factoring all integers up to N, or to compute the values $f(n)$, $n \leq N$, of arithmetical functions f that depend on the factorization of integers. For instance, one can take $f = \mu$, the Möbius function, or $f = \lambda$, the Liouville function.

The point of the sieve of Eratosthenes is that it can be carried out in time close to linear on N, even though determining whether an individual integer is prime, let alone factoring it, takes much more than constant time with current methods. Though a very naïve implementation of the sieve would take space proportional to N, it is not hard to see how to implement the sieve in space $O(\sqrt{N})$, simply by applying the sieve to intervals of length $O(\sqrt{N})$ one at a time; the time taken is still close to linear[1] on N. (One could take shorter intervals, but the algorithm would then become much less efficient.) This is called the "segmented sieve"; see [Sin69] for an early reference.

Of course, the output still takes space linear on N, but that is of less importance: we can store the output in slower memory (such as a hard drive), or give the output

Received by the editor April 25, 2018, and, in revised form, December 3, 2018, and February 22, 2019.

2010 *Mathematics Subject Classification.* Primary 11Y05, 11Y16; Secondary 11Y11.

The author was supported by funds from his Humboldt Professorship.

[1]In its standard form, the sieve of Eratosthenes (segmented or not) takes time $O(N \log \log N)$; when used for sieving out primes, its time consumption can be reduced to $O(N)$.

We follow the convention that arithmetic operations (e.g., adding or multiplying two numbers) take $O(1)$ time. This convention holds in the range in which using a sieve is realistic. (Outside that range, such operations do take time $O((\log N)^c)$, c close to 1.) We shall measure space in bits, again reflecting how matters work in practice.

in batches to a program that needs it and can handle it sequentially. Then the space used is certainly $O(\sqrt{N})$.

There has been a long series of improvements to the basic segmented sieve. Most of them improve the running time or space by a constant factor or by a factor in the order of $\log\log N$. Many work only when the sieve is used to construct primes, as opposed to computing μ, say. See [Sor98], which reviews the state of matters at the time before improving on it; see also [Wal] for further references and for a contemporary implementation combining some of the most useful existing techniques.

In practice, saving space sometimes amounts to saving time, even when it seems, at first, that there should be a trade-off. As of the time of writing, a good office computer can store about 10^{12} integers in very slow memory (a hard drive), about 10^9 integers in memory working at intermediate speed (RAM), and about 10^6 integers in fast memory (cache); a program becomes faster if it can run on cache, accessing RAM infrequently. Having enough RAM is also an issue; sieves have been used, for instance, to verify the binary Goldbach conjecture up to $4\cdot 10^{18}$ [OeSHP14], and so we are close to the point at which $O(\sqrt{N})$ space might not fit in RAM. Space constraints can become more severe if several processor cores work in parallel and share resources.

Moreover, finding all primes within a short interval can be useful in itself. For instance, there are applications in which we verify a conjecture on one interval at a time, and we neither need nor can store a very long interval in memory. See the verification of Goldbach's (binary) conjecture up to $4 \cdot 10^{18}$ in [OeSHP14], and, in particular [OeSHP14, §1.2].

Galway [Gal00] found a way to sieve using space $O(N^{1/3})$ and time $O(N)$. Like the sieve in [AB04], on which it is based,[2] Galway's sieve is specific to finding prime numbers. There is also the algorithm in [Sor06], specific, again, to finding primes: under the assumption of the Generalized Riemann Hypothesis, it finds all primes up to N in space $O((\log N)^3/\log\log N)$ and time $O(N(\log N)^2/\log\log N)$; unconditionally, it runs in space $O(N^{0.132})$ and time $O(N^{1.132})$. (It runs in time $O(N\log N)$ under the assumption of a more specialized conjecture.)

We will show how to implement a sieve of Eratosthenes in space close to $N^{1/3}$ and still close to linear time. Our method is not limited to finding prime numbers; it can be used to factor integers, or, of course, to compute $\mu(n)$, $\lambda(n)$, or other functions given by the factorization of n.

Main Theorem. *We can construct all primes $p \le N$ in*

$$\text{space } O\left(N^{1/3}(\log N)^{2/3}\right) \quad \text{and} \quad \text{time } O(N\log N). \tag{1.1}$$

We can also factor all integers $n \le N$ in

$$\text{space } O\left(N^{1/3}(\log N)^{5/3}\right) \quad \text{and} \quad \text{time } O(N\log N). \tag{1.2}$$

Moreover, for $N^{1/3}(\log N)^{2/3} \le \Delta \le N$, we can construct all primes in an interval $[N-\Delta, N+\Delta]$ in

$$\text{space } O(\Delta) \quad \text{and} \quad \text{time } O(\Delta\log N)$$

[2] Atkin and Bernstein's preprint was already available in 1999, as the bibliography in [Gal00] states.

and factor all integers in the same interval in

$$\textit{space } O(\Delta \log N) \qquad \textit{and} \qquad \textit{time } O(\Delta \log N).$$

Here we recall that *space* refers to the number of bits used, and *time* to the number of operations of words used (on integers of size $O(N)$).

The main ideas come from elementary number theory. In order for us to be able to apply the sieve to an interval I of length $O(N^{1/3})$ without large time inefficiencies, we need to be able to tell in advance which primes (or integers) d up to $\sqrt{N}$ divide at least one integer in I, without testing each d individually. We can do this by Diophantine approximation, followed by a local linear approximation to the function $x \mapsto n/x$ for n fixed, and then by solving what amounts to a linear equation $\bmod 1$.

The idea of using Diophantine approximation combined with a local linear approximation is already present in [TCH12], where it was used to compute $\sum_{n \le x} \tau(n)$ in time $O(x^{1/3}(\log x)^{O(1)})$. (We write $\tau(n)$ for the number of divisors of an integer n.) The basic underlying idea in [Gal00] may be said to be the same as the one here: we are speaking of a Diophantine idea that stems ultimately from Voronoï's work on the Dirichlet divisor problem [Vor03] (in our work) and Sierpinski's adaptation of the same method to the circle problem [Sie06] (in the case of [Gal00]).[3] For that matter, [Gal00, §5] already suggests that Voronoï's work on the Dirichlet divisor problem could be used to make the sieve of Eratosthenes in space about $O(N^{1/3})$ and close to linear time. We should also make clear that Galway can sieve out efficiently segments of length about $x^{1/3}$, just as we do.

One difference between this paper and Galway's is that the relation to Voronoï's and Sierpinski's work in Galway's paper may be said to be more direct, in that Galway literally dissects a region between two circles, much as Sierpinski does. In the case of the present paper, we can say that Voronoï's main idea originated in the context of giving elementary estimates (for $\sum_{n \le x} \tau(n)$) and is now used to carry out an exact computation.

Another precedent that must be mentioned is Oliveira e Silva's technique for efficient cache usage [eS], [OeSHP14, Algorithm 1.2]. It seems that this technique can be combined with the algorithm here. Such a combination can be useful, for instance, when N is so large that $N^{1/3}$ bits fit in RAM but not in cache.

Additional motivation. Our initial interest in the problem stemmed from the fact that we had to compute values of $\mu(n)$ so as to check inequalities of the form[4] $\sum_{n \le x} \mu(n) \le \sqrt{x}$, $\sum_{n \le x} \mu(n)/n \le 2/\sqrt{x}$, etc., for all x less than some large finite N. The importance of such sums in number theory is clear. Explicit results on them for x bounded help complement analytic estimates, which are generally strong only when x is large.

While we can obviously determine values of $\mu(n)$ by first factorizing n and then using the definition of μ, it is also possible and rather simple to save some space

[3] To be precise, the immediate inspiration for [TCH12] came from Vinogradov's simplified version of Voronoï's method, as in [Vin54, Ch. III, exercises 3–6]. A bound of roughly the same quality as Voronoï's result was claimed long before Voronoï in [Pfe86]. While the proof there was apparently incomplete, Landau later showed [Lan12] that it could be made into an actual proof, and sharpened to give a result matching Voronoï's. Thanks are due to S. Patterson for pointing out Pfeiffer's result to the author.

[4] Mertens's conjecture states that the inequality $\sum_{n \le x} \mu(n) \le \sqrt{x}$ holds for all x. That conjecture has been disproved [OtR85], but the inequality is known to hold for all $x \le 10^{16}$ [Hur18], and may hold in a far wider range.

and time in practice by modifying the procedure we will give (Algorithm 6) so as to keep track of $\mu(n)$ instead of the list of factors. Time and space complexity remain the same.

Conventions and notation. Integer operations are assumed to take constant time. As is usual, we write $f(x) = O(g(x))$ to mean that there exists a constant $C > 0$ such that $|f(x)| \le Cg(x)$ for all large enough x. We write $f(x) = O^*(g(x))$ to mean that $|f(x)| \le g(x)$ for all x. We use either $f(x) \ll g(x)$ or $g(x) \gg f(x)$ to mean $f(x) = O(g(x))$.

For n a non-zero integer and p a prime, $v_p(n)$ denotes the largest k such that $p^k|n$. As is customary, we write (a, b) for the gcd (greatest common divisor) of a and b, provided that no confusion with the ordered pair (a, b) is possible.

Given $\alpha \in \mathbb{R}$, we denote by $\{\alpha\}$ the element of $[0, 1)$ congruent to α modulo 1, i.e., modulo $\mathbb{Z}$.

2. Analysis of the problem

Let $I = [n - \Delta, n + \Delta] \subset [0, x]$ be an interval. How can we tell which integers $m \le \sqrt{x}$ have at least one integer multiple in the interval I?

Our motivation for asking this question is that we will be taking intervals I and sieving them by all integers, or all primes, $m \le \sqrt{x}$. Going over all integers $\le \sqrt{x}$ would take time at least $\sqrt{x}$, which could be much larger than Δ.

We will be able to sieve our intervals by $m \le \sqrt{x}$ by producing a list of those m which might have multiples in I, without testing all $m \le \sqrt{x}$ in succession. A quick probabilistic heuristic hints that the number of such m should be proportional to $\Delta \log x$, which, for $\Delta \sim x^{1/3}$, is much smaller than $\sqrt{x}$.

Let $K \ge 2$ be a parameter to be set later. If $m \le K\Delta$, we simply sieve by m. Doing so takes time $O(1 + \Delta/m)$. We could, of course, test first whether m is prime; we will discuss this option later.

Assume henceforth that $m > K\Delta$. The interval I contains a multiple of m if and only if

$$\left\{\frac{n}{m}\right\} \in \left[-\frac{\Delta}{m}, \frac{\Delta}{m}\right] \bmod 1,$$

that is, $\{n/m\} \in [-\Delta/m, \Delta/m] + \mathbb{Z}$.

Say we have already dealt with all $m \le M$, where $M \ge K\Delta$, and that we want to examine m close to m_0, where $m_0 > M$. Write $m = m_0 + r$. The truncated Taylor expansion

$$f(m) = f(m_0) + f'(m_0)r + \frac{f''(m_0 + \theta r)}{2} r^2$$

(where θ is some element of $[0, 1]$) gives us, once we set $f(x) = n/x$,

$$\frac{n}{m} = \frac{n}{m_0} - \frac{n}{m_0^2} r + O^*\left(\frac{n}{m_-^3} r^2\right), \tag{2.1}$$

where $m_- = \min(m, m_0)$. We will make sure that r is small enough that $nr^2/m_-^3 \lesssim \kappa\Delta/m$, where $\kappa > 0$. (That is, we allow our error term to be not much larger than the interval we are trying to hit.) We ensure r satisfies this condition by letting

$$R = \left\lfloor \sqrt{\frac{\kappa\Delta}{n}} M \right\rfloor, \qquad m_0 = M + R. \tag{2.2}$$

Then $nr^2/m_-^3 \le nr^2/M^3 \le \kappa\Delta/M$ for all $m = m_0 + r$, $r \in [-R, R]$. We let $\kappa = 1/4$, since it is a value that is neither too large nor too small. We will also assume $\Delta \ge n^{1/3}$, and so $R \ge \lfloor K/2 \rfloor \ge 1$, since $K \ge 2$.

We have thus reduced our problem to that of quickly finding all r in an interval $[-R, R]$ such that $P(r) \in [-\eta, \eta] \bmod 1$, where $P(r)$ is the linear polynomial $-(n/m_0^2)r + (n/m_0)$ and $\eta = (1+\kappa)\Delta/M = 5\Delta/4M$. In other words, we are being asked to find all approximate solutions to a linear equation in $\mathbb{R}/\mathbb{Z}$ efficiently. Let us assume that $K \ge 5/2$, so that $\eta \le 1/2$; otherwise there is not much to do.

Let $\alpha_1 = -(n/m_0^2)$, $\alpha_0 = n/m_0$. Given a rational number α, we can find—by means of continued fractions—an approximation a/q to α with $q \le Q$ and

$$\left|\frac{a}{q} - \alpha\right| \le \frac{1}{qQ} \tag{2.3}$$

in time $O(\log Q)$. The procedure **DiophAppr**(α, Q) to obtain a/q is given in Algorithm 4; it runs in time $O(\log Q)$ and constant space. The fact that its output satisfies (2.3) follows from [HW79, Thm. 164] or [Khi97, Thm. 9]. (In the notation of both sources: since any two consecutive approximants p_n/q_n, p_{n+1}/q_{n+1} to α satisfy $|p_n/q_n - \alpha| \le 1/q_n q_{n+1}$, the last p_n/q_n with $q_n \le Q$ satisfies $|p_n/q_n - \alpha| < 1/q_n Q$.)

We invoke **DiophAppr**$(\alpha_1, 2R)$, and obtain a rational a/q with $(a, q) = 1$ and $q \le Q = 2R$ satisfying (2.3) for $\alpha = \alpha_1$. Hence, for $r \in [-R, R]$,

$$P(r) = \alpha_1 r + \alpha_0 = \alpha_0 + \frac{ar}{q} + O^*\left(\frac{1}{2q}\right) \equiv \frac{c + ar}{q} + O^*\left(\frac{1}{q}\right) \bmod 1 \tag{2.4}$$

for $c = \lfloor \alpha_0 q + 1/2 \rfloor$. We have thus reduced our problem to a problem in $\mathbb{Z}/q\mathbb{Z}$: if $P(r) \in [-\eta, \eta]$, then

$$c + ar \in \{-k-1, -k, \dots, k+1\} \bmod q, \tag{2.5}$$

where $k = \lfloor \eta q \rfloor$. We must find the values of r satisfying (2.5).

The solutions to (2.5) are clearly given by

$$r \equiv -a^{-1}(c + j) \bmod q \tag{2.6}$$

for $-k-1 \le j \le k+1$. The multiplicative inverse $a^{-1} \bmod q$ of a can be computed in time $O(\log q)$ by the Euclidean algorithm. In fact, we compute it at the same time as the continued fraction that gives us a/q: by [Khi97, Thm. 2], two consecutive approximants p_n/q_n, p_{n+1}/q_{n+1} satisfy $p_n^{-1} = (-1)^{n+1} q_{n+1} \bmod q_n$; moreover, $p_n^{-1} = (-1)^{n+1} q_{n-1} \bmod q_n$.

Thus, we simply need to go over all m of the form $m_0 + r$, where r goes over integers in $[-R, R]$ satisfying (2.6). Since $m_0 - R = M > K\Delta \ge 2\Delta$, each such m has at most one multiple in $I = [n - \Delta, n + \Delta]$. We do not miss any $m \in [m_0 - R, m_0 + R] = [M, M + 2R]$ that has a multiple in I.

Due to the errors involved in the truncation of the Taylor expansion and in Diophantine approximation, we may obtain some m that have no multiples within I. We simply ignore them, after checking that that is the case.

3. Description of the algorithm

We have already given a nearly full description of the algorithm. It only remains to give the pseudocode (Algorithms 1–4), with some commentary.

Algorithm 1 Main algorithm: sieving $[n-\Delta, n+\Delta] \subset \mathbb{R}^+$

function NewSegSiev(n,Δ,K)
Ensure: $S_j = 1$ if $n+j$ is prime, $S_j = 0$ otherwise, for $-\Delta \le j \le \Delta$
Require: $n, \Delta \in \mathbb{Z}^+$, $\sqrt[3]{n} \le \Delta < n$, $K \ge 5/2$
  $S' \leftarrow$ SubSegSiev$(n-\Delta, 2\Delta, K\Delta)$ ▷ sieve by all $p \le K\Delta$
  $S_j \leftarrow S'_{j+\Delta}$ for all $-\Delta \le j \le \Delta$
  $M \leftarrow \lfloor K\Delta \rfloor + 1$
  while $M \le \sqrt{n+\Delta}$ **do**
    $R \leftarrow \lfloor M\sqrt{\Delta/4n} \rfloor$, $m_0 \leftarrow M + R$
    $\alpha_1 \leftarrow \{-n/m_0^2\}$, $\alpha_0 \leftarrow \{n/m_0\}$, $\eta \leftarrow 5\Delta/4M$
    $(a, a^{-1}, q) \leftarrow$ DiophAppr$(\alpha_1, 2R)$
    $c \leftarrow \lfloor \alpha_0 q + 1/2 \rfloor$, $k \leftarrow \lfloor \eta q \rfloor$
    for $-k-1 \le j \le k+1$ **do**
      $r_0 \leftarrow -a^{-1}(c+j) \bmod q$
      for $m \in (m_0 + r_0 + q\mathbb{Z}) \cap [M, M+2R]$ **do**
        $n' \leftarrow \lfloor (n+\Delta)/m \rfloor \cdot m$ ▷ n' is a multiple of m
        if $n' \in [n-\Delta, n+\Delta] \wedge (n' > m)$ **then**
          $S_{n'-n_0} \leftarrow 0$ ▷ thus sieving out n'
    $M \leftarrow M + 2R + 1$
  return S

Algorithm 2 A simple sieve of Eratosthenes

function SimpleSiev(N)
Ensure: for $1 \le n \le N$, $P_n = 1$ if n is prime, $P_n = 0$ otherwise
  $P_1 \leftarrow 0$, $P_2 \leftarrow 1$, $P_n \leftarrow 0$ for $n \ge 2$ even, $P_n \leftarrow 1$ for $n \ge 3$ odd
  $m \leftarrow 3$, $n \leftarrow m \cdot m$
  while $n \le N$ **do**
    if $P_m = 1$ **then**
      while $n \le N$ **do** ▷ [sic]
        $P_n \leftarrow 0$, $n \leftarrow n + 2m$ ▷ sieves odd multiples $\ge m^2$ of m
    $m \leftarrow m + 2$, $n \leftarrow m \cdot m$
  return P
Time: $O(N \log\log N)$. **Space:** $O(N)$.

All variables are integers, rationals, or tuples or sets with integer or rational entries. Thus, all arithmetic operations can be carried out exactly. We write $\mathrm{num}(\alpha)$ and $\mathrm{den}(\alpha)$ for the numerator and denominator of $\alpha \in \mathbb{Q}$ (written in minimal terms: $\mathrm{num}(\alpha)$, $\mathrm{den}(\alpha)$ coprime, $\mathrm{den}(\alpha) > 0$).

In Algorithm 6, we are defining $\alpha_1 \leftarrow \{-n/m_0^2\}$, $\alpha_0 \leftarrow \{n/m_0\}$, rather than $\alpha_1 \leftarrow -n/m_0^2$, $\alpha_0 \leftarrow n/m_0$, as in the exposition above. The motivation is simply to keep variable sizes small. It is easy to see that the values of α_0, α_1 matter only $\bmod \mathbb{Z}$.

3.1. **Sieving for primes.** The main function is NewSegSiev (Algorithm 1). It takes as inputs n and Δ, as well as the parameter K, which affects time and space consumption. The basic procedure is easy to summarize. NewSegSiev

Algorithm 3 Segmented sieves of Eratosthenes, traditional version

1: **function** SIMPLESEGSIEV(n,Δ,M) ▷ sieves $[n, n+\Delta]$ by primes $p \le M$

Ensure: $S_j = \begin{cases} 0 & \text{if } n+j = 0, 1 \text{ or if } p \mid (n+j) \text{ for some } p \le M \\ 1 & \text{otherwise} \end{cases}$

2: $S_j \leftarrow 1$ for all $0 \le j \le \Delta$
3: $S_j \leftarrow 0$ for $0 \le j \le 1-n$ ▷ [sic; excluding 0 and 1 from prime list]
4: $P \leftarrow$ SIMPLESIEV(M)
5: **for** $1 \le m \le M$ **do**
6: **if** $P_m = 1$ **then**
7: $n' \leftarrow \max(m \cdot \lceil n/m \rceil, 2m)$
8: **while** $n' \le n+\Delta$ **do** ▷ n' goes over mults. of m in $n + [0, \Delta]$
9: $S_{n'-n} \leftarrow 0$, $n' \leftarrow n' + m$
10: **return** S

Time: $O((M+\Delta)\log\log M)$. **Space:** $O(M+\Delta)$.

11: **function** SUBSEGSIEV(n,Δ,M) ▷ sieves $[n, n+\Delta]$ by primes $p \le M$

Ensure: $S_j = \begin{cases} 0 & \text{if } n+j = 0, 1 \text{ or if } p \mid (n+j) \text{ for some } p \le M \\ 1 & \text{otherwise} \end{cases}$

12: $S_j \leftarrow 1$ for all $\max(0, 2-n) \le j \le \Delta$
13: $S_j \leftarrow 0$ for $0 \le j \le 1-n$
14: $\Delta' \leftarrow \lfloor \sqrt{M} \rfloor$, $M' \leftarrow 1$
15: **while** $M' \le M$ **do**
16: $P \leftarrow$ SIMPLESEGSIEV($M', \Delta', \lfloor \sqrt{M'+\Delta'} \rfloor$)
17: **for** $M' \le p < \min(M, M'+\Delta')$ **do**
18: **if** $P_{p-M'} = 1$ **then** ▷ if m is a prime...
19: $n' \leftarrow \max(p \cdot \lceil n/p \rceil, 2p)$
20: **while** $n' \le n+\Delta$ **do**
21: $S_{n'-n} \leftarrow 0$, $n' \leftarrow n' + p$
22: $M' \leftarrow M' + \Delta' + 1$
23: **return** S

Time: $O((M+\Delta)\log\log M)$. **Space:** $O(\sqrt{M}+\Delta)$.

24: **function** SEGSIEV(n,Δ) ▷ finds primes in $[n, n+\Delta]$

Ensure: for $0 \le j \le \Delta$, $S_j = 1$ if $n+j$ prime, $S_j = 0$ otherwise

25: **return** SUBSEGSIEV($n, \Delta, \lfloor \sqrt{n+\Delta} \rfloor$)

Time: $O((\sqrt{n}+\Delta)\log\log(n+\Delta))$. ▷ $\sqrt{n+\Delta} \le \sqrt{n} + \sqrt{\Delta} \le \sqrt{n} + \Delta$
Space: $O(n^{1/4}+\Delta)$. ▷ $(n+\Delta)^{1/4} \le n^{1/4} + \Delta$

uses SUBSEGSIEV (Algorithm 3), a segmented sieve of traditional type, to sieve for primes $p \le K\Delta$. It then proceeds as detailed in §2 to find the integers m in $[M, M+2R]$ that may divide an integer in the interval $[n-\Delta, n+\Delta]$. Then it sieves by them. The key step, in finding such m, is to call DIOPHAPPR, which uses continued fractions in a standard way to find (i) a rational approximation a/q to the input α_1, (ii) the inverse $a^{-1} \bmod q$.

Algorithm 4 Finding a Diophantine approximation via continued fractions

1: **function** DIOPHAPPR(α,Q)

Ensure: Returns (a, a^{-1}, q) s.t. $\left|\alpha - \frac{a}{q}\right| \leq \frac{1}{qQ}$, $(a,q) = 1$, $q \leq Q$, $aa^{-1} \equiv 1 \bmod q$

2: $b \leftarrow \lfloor \alpha \rfloor$, $p \leftarrow b$, $q \leftarrow 1$, $p_- \leftarrow 1$, $q_- \leftarrow 0$, $s \leftarrow 1$

3: **while** $q \leq Q$ **do**

4: **if** $\alpha = b$ **then return** $(p, -sq_-, q)$

5: $\alpha \leftarrow 1/(\alpha - b)$

6: $b \leftarrow \lfloor \alpha \rfloor$, $(p_+, q_+) \leftarrow b \cdot (p,q) + (p_-, q_-)$

7: $(p_-, q_-) \leftarrow (p,q)$, $(p,q) \leftarrow (p_+, q_+)$, $s \leftarrow -s$

8: **return** (p_-, sq, q_-)

Time: $O(\log \max(Q, \mathrm{den}(\alpha)))$. **Space:** $O(1)$.

We could try to improve on NEWSEGSIEVE by throwing out even values of m, say; as they are certainly not prime, we need not sieve by them. More details are given in "Side notes on wheels", below.

Preexistent sieves as subroutines. Going back to SUBSEGSIEV: we could avoid using it at all, just by sieving by all integers $m \leq K\Delta$, rather than by all primes $p \leq K\Delta$. That would give a running time of $O(K\Delta + \Delta \log M)$ rather than $O(K\Delta \log\log M)$. We take the slightly more complicated route in Algorithm 3, not just because it is better for $K = o((\log M)/\log\log M)$, but also for the sake of exposition, in that we get to see several existing forms of the sieve of Eratosthenes. Note, however, that none of this will decrease the order of magnitude of the time taken by our entire algorithm, since the total time will be at least in the order of $\Delta \log M$.

Function SIMPLESIEV is a relatively simple kind of sieve of Eratosthenes. It sieves the integers up to N by the primes up to $\sqrt{N}$, where these primes are found by this very same process. It is clear that we need to sieve only by the primes up to $\sqrt{N}$, since any composite number $n \leq N$ has at least one prime factor $p \leq \sqrt{N}$. Sieving only by the primes, rather than by all integers, is enough to take down the running time to $O(N \log\log N)$. We also use a very primitive "wheel" in that we sieve using only odd multiples of the primes. (Again, see the comments on wheels below.)

The basic segmented sieve is implemented as SIMPLESEGSIEV(n, Δ, M). It uses SIMPLESIEV so as to determine the primes up to M. Then it sieves the interval $[n, n+\Delta]$ by them.

We could use SIMPLESEGSIEV instead of SUBSEGSIEV. The point of SUBSEGSIEV is simply to reduce space consumption by one more iteration: SUBSEGSIEV determines the primes up to M by SIMPLESEGSIEV, taking only space $O(\sqrt{M})$ at a time; it sieves $[n, n+\Delta]$ by these primes as it goes along. The total time taken by calls on SIMPLESEGSIEV is $O(M \log\log M)$; to this we add the time $O(\Delta \log\log M)$ taken by sieving the interval $[n, n+\Delta]$ by the primes up to M.

Incidentally, it should be clear that all factors of $\log\log N$, $\log\log M$, and the like are coming from Mertens's classical asymptotic statement

$$\sum_{p \leq N} \frac{1}{p} = \log\log N + O(1).$$

For instance, the number of times the instructions $S_{n'-n} \leftarrow 0$, $n' \leftarrow n' + m$ are executed in SimpleSegSiev or SubSegSiev is at most

$$\sum_{p \le M} \left\lceil \frac{\Delta + 1}{p} \right\rceil \le \Delta \sum_{p \le M} \frac{1}{p} + \sum_{p \le M} 1 \\ = O(\Delta \log\log M + M).$$

Side note on wheels. In general, a "wheel" is just $(\mathbb{Z}/P\mathbb{Z})^*$, where $P = \prod_{p \le C} p$ for some constant C. We would use it by sieving only by multiples $m \cdot p'$ of the primes p', where m reduces $\bmod P$ to an element of the wheel. Obviously $m \bmod P$ should be constantly updated by shifting as m increases, rather than be determined by division each time; hence the "wheel". It is possible (and common, at least in theoretical analyses) to choose C of size $\delta \log N$ for a small constant δ (say, $\delta = 1/4$), with the consequence that $P \sim N^\delta$. Then only a proportion $\prod_{p \le C}(1 - 1/p) \sim \frac{e^{-\gamma}}{\log C} = O(1/\log\log N)$ of the elements of $\mathbb{Z}/P\mathbb{Z}$ (here γ is Euler's constant) are in $(\mathbb{Z}/P\mathbb{Z})^*$. With appropriate coding, this fact can be used to reduce the running time of a simple sieve or a segmented sieve for primes (such as SimpleSiev or SimpleSegSiev) by a factor of $\log\log N$ [Pri83]. We will not bother including wheels in our pseudocode, since they would reduce only a second-order term of the total time bound in this fashion; the time complexity of NewSegSiev would remain the same. They can obviously be added in implementation.

It is tempting to introduce a wheel in a different place, namely, to make sure that the variable m in NewSegSiev has no prime factors $p < \delta \log N$. The hope there would be to reduce the main term in the total time by a factor of $\log\log N$. However, as we will later see, q will be usually close to $2R$; thus, most of the time, it will not make sense to use a large wheel as m goes over integers in $(m_0 + r_0 + q\mathbb{Z}) \cap [M, M + 2R]$, as there will be few such integers. Using a wheel on q instead—something that would take time $\gg q$ to set up—would make no sense: we do not go over $\mathbb{Z}/q\mathbb{Z}$ several times, but rather go over a few elements of it once.

What can make sense is to introduce a very small wheel, of bounded size, to attempt to gain a constant factor. For instance, a wheel of size 2 would work as follows: if q is even, then, in the loop on j in NewSegSiev, we consider only values of j such that $c + j$ is odd; if q is odd, then we hop over every other value of m in a given congruence class mod q within $[M, M + 2R]$, considering only odd values m. This sort of modification will reduce total time consumption only by a constant factor, and so, for the sake of simplicity, we do not include it in the pseudocode, or use it further.

Variable size. We should bound the size of our variables in case we choose to implement our algorithm using fixed-size integers and rationals. It is easy to see that our integers will be of size $\le n + \Delta$. We should also bound the size of our rational variables. It is trivial to modify function NewSegSiev so that m_0 is always $\le \sqrt{n + \Delta}$. It is then easy to show that we work entirely with integers—in numerators, in denominators or on their own—of size $\le \min(n+\Delta, (2R)^2) \le n+\Delta$.

For n very large and Δ not as large, it may be helpful to store some variables (such as a, a^{-1}, q, k, j, and r_0) in smaller integer types (64 bits, say); these variables are all bounded by $2R \le M\sqrt{\Delta/n} \le \sqrt{\Delta(n + \Delta)/n}$.

3.2. **Sieving for factorization.** We will now see how to modify our algorithm so that it factorizes all integers in the interval $[n-\Delta, n+\Delta]$, rather than simply finding all primes in that interval. Time and space usage will not be much greater than when we are just finding primes. It goes without saying that this makes it possible to compute various arithmetic functions (the Möbius function $\mu(m)$, the Liouville function $\Lambda(m)$, etc.) for all $m \in [n-\Delta, n+\Delta]$.

For the sake of clarity, we first give a well-known procedure for factoring all integers in an interval $[n, n+\Delta]$ by means of the sieve of Eratosthenes (Algorithm 5), just as we went over a traditional segmented sieve (Algorithm 3) before describing our sieve for primes. We will later reuse most of the subroutines.

Algorithm 5 Segmented sieve of Eratosthenes for factorization (traditional)

function SubSegSievFac(n,Δ,M) ▷ finds prime factors $p \le M$
Ensure: for $0 \le j \le \Delta$, $F_j = \{(p, v_p(n+j))\}_{p \le M, p|n+j}$
Ensure: for $0 \le j \le \Delta$, $\Pi_j = \prod_{p \le M, p|(n+j)} p^{v_p(n+j)}$.
  $F_j \leftarrow \emptyset$, $\Pi_j \leftarrow 1$ for all $0 \le j \le \Delta$
  $\Delta' \leftarrow \lfloor \sqrt{M} \rfloor$, $M' \leftarrow 1$
  while $M' \le M$ **do**
    $P \leftarrow$ SimpleSegSiev($M', \Delta', \lfloor \sqrt{M'+\Delta'} \rfloor$)
    for $M' \le p < M' + \Delta'$ **do**
      if $P_{p-M'} = 1$ **then** ▷ if p is a prime...
        $k \leftarrow 1$, $d \leftarrow p$ ▷ d will go over the powers p^k of p
        while $d \le n + \Delta$ **do**
          $n' \leftarrow d \cdot \lceil n/d \rceil$
          while $n' \le n + \Delta$ **do**
            if $k = 1$ **then**
              append $(p, 1)$ to $F_{n'-n}$
            else
              replace $(p, k-1)$ by (p, k) in $F_{n'-n}$
            $\Pi_{n'-n} \leftarrow p \cdot \Pi_{n'-n}$, $n' \leftarrow n' + d$
          $k \leftarrow k+1$, $d \leftarrow p \cdot d$
    $M' \leftarrow M' + \Delta'$
  return (F, Π)

Time: $O((M+\Delta)\log\log(n+\Delta))$.
Space: $O(M + \Delta \log(n+\Delta))$.

function SegSievFac(n,Δ) ▷ factorizes all $n' \in [n, n+\Delta]$
Ensure: for $0 \le j \le \Delta$, F_j is the list of pairs $(p, v_p(n+j))$ for $p|n+j$
  $(F, \Pi) \leftarrow$ SubSegSievFac($n, \Delta, \lfloor \sqrt{n+\Delta} \rfloor$)
  for $n \le n' \le n + \Delta$ **do**
    if $\Pi_{n'-n} \ne n'$ **then**
      $p_0 \leftarrow n'/\Pi_{n'-n}$, **append** $(p_0, 1)$ to $F_{n'-n}$
  return F

Time: $O((\sqrt{n}+\Delta)\log\log(n+\Delta))$.
Space: $O(n^{1/4} + \Delta \log(n+\Delta))$.

Our new sieve for factoring (NewSegSievFac, Algorithm 6), designed for intervals around x of length $\Delta \gg x^{1/3}$, is very similar to our sieve for primes (Algorithm 1). We use a classical segmented sieve (SubSegSievFac, Algorithm 5) to find all factors $p \le K\Delta$ of $n+j$ for $-\Delta \le j \le \Delta$. After the call to SubSegSievFac, the variable Π_j contains $\prod_{p \le K\Delta} p^{v_p(n+j)}$. Since $K\Delta > (2n)^{1/3} > (n+\Delta)^{1/3}$, we see that $(n+j)/\Pi_j$ is either 1 or the product of at most two primes $> K\Delta$ (not necessarily distinct). In the innermost loop of SubSegSievFac, when we come across an $m > K\Delta$ that divides not just $n+j$, but $(n+j)/\Pi_j$, we multiply Π_j by m, or by its square, if $m^2 | (n+j)/\Pi_j$, and include m (or its square) in the factorization. Note that m has to be a prime, or else Π_j would have already been multiplied by some factor $p^{v_p(n+j)}$, $p|m$, $p < m$, either earlier in the loop or in SubSegSievFac, thus making $m|(n+j)/\Pi_j$ impossible.

We should also explain the purpose of the final loop in NewSegSievFac. (The loop in the classical procedure SegSievFac is identical, and plays the same role.) Once we take care of all $m \le \sqrt{n+\Delta}$ (and possibly some beyond), what we have, for each $-\Delta \le j \le \Delta$, is either that $\Pi_j = 1$ and F_j contains a full factorization of $n+j$, or $\Pi_j \ne 1$ and F_j is missing a single large prime factor p, which has to be equal to n/Π_j. We include that prime factor in the factorization and are done.

Algorithm 6 Main algorithm: factoring integers in $[n-\Delta, n+\Delta]$

function NewSegSievFac(n,Δ,K)
Ensure: for $-\Delta \le j \le \Delta$, F_j is the list of pairs $(p, v_p(n+j))$ for $p|n+j$
Require: $n, \Delta \in \mathbb{Z}^+$, $\sqrt[3]{n} \le \Delta < n$, $K \ge 5/2$
  $(F', \Pi') \leftarrow$ SubSegSievFac$(n-\Delta, 2\Delta, K\Delta)$ ▷ find factors $p \le K\Delta$
  $F_j \leftarrow F'_{j+\Delta}$, $\Pi_j \leftarrow \Pi'_{j+\Delta}$ for all $-\Delta \le j \le \Delta$
  $M \leftarrow \lfloor K\Delta \rfloor + 1$
  while $M \le \sqrt{n+\Delta}$ **do**
    $R \leftarrow \lfloor M\sqrt{\Delta/4n} \rfloor$, $m_0 \leftarrow M + R$
    $\alpha_1 \leftarrow \{-n/m_0^2\}$, $\alpha_0 \leftarrow \{n/m_0\}$, $\eta \leftarrow 5\Delta/4M$
    $(a, a^{-1}, q) \leftarrow$ DiophAppr$(\alpha_1, 2R)$
    $c \leftarrow \lfloor \alpha_0 q + 1/2 \rfloor$, $k \leftarrow \lfloor \eta q \rfloor$
    for $-k-1 \le j \le k+1$ **do**
      $r_0 \leftarrow -a^{-1}(c+j) \bmod q$
      for $m \in (m_0 + r_0 + q\mathbb{Z}) \cap [M, M+2R]$ **do**
        $n' \leftarrow \lfloor (n+\Delta)/m \rfloor \cdot m$ ▷ n' is a multiple of m
        if $n' \in [n-\Delta, n+\Delta]$ **then**
          if $m | (n'/\Pi_{n'-n_0})$ **then** ▷ m is a new factor of n'
            if $m^2 | n'$ **then**
              $\Pi_{n'-n_0} \leftarrow m^2 \cdot \Pi_{n'-n_0}$, **append** $(m, 2)$ to $F_{n'-n}$
            else
              $\Pi_{n'-n_0} \leftarrow m \cdot \Pi_{n'-n_0}$, **append** $(m, 1)$ to $F_{n'-n}$
    $M \leftarrow M + 2R + 1$
  for $n_0 \le n' \le n_0 + 2\Delta$ **do**
    if $\Pi_{n'-n_0} \ne n'$ **then**
      $p_0 \leftarrow n'/\Pi_{n'-n_0}$, **append** $(p_0, 1)$ to $F_{n'-n_0}$
  return F

4. Time analysis. Parameter choice

The space and time consumption of Algorithms 2–5 is clearly as stated.

Time consumption of main algorithm. Let us analyze the time consumption of Algorithm 1. (Its space consumption, namely, $O(\Delta + \sqrt{K\Delta})$, will be clear.) Sieving by primes $p \le K\Delta$ gets done by the traditional segmented-procedure SubSegSieve$(n, 2\Delta, K\Delta)$, which takes time $O(K\Delta \log\log K\Delta)$ and space $O(\sqrt{K\Delta} + \Delta)$. We must analyze now how much time it takes to sieve by integers $K\Delta < m \le \sqrt{n+\Delta}$. (Our algorithm sieves by integers $K\Delta < m \le \sqrt{n+\Delta}$, not just by primes, simply because the Diophantine-approximation algorithm cannot tell in advance which of the integers it outputs will be prime. As we discussed before, we could apply a small wheel in the hope of saving a constant factor in time.)

We will use the main ideas of Vinogradov's proof of the bound

$$\sum_{n \le x} \tau(n) = x \log x + O(x^{1/3} (\log x)^{5/3}), \tag{4.1}$$

as given in [Vin54, Ch. III, exer. 3–6],[5] although we will not use the bound (4.1) itself. Our treatment will be self-contained.

Algorithm 1 does not correspond extremely closely to Vinogradov's approach—we use our approximations n/m_0, $-n/m_0^2$ on the relatively broad intervals on which they are useful, whereas Vinogradov's procedure changes approximations constantly. Nevertheless, we will be able to use the basic approach that he took to bound an error term, although we of course will use it to bound time consumption.

Let us look at an interval $[M, M+2R]$. Finding a, a^{-1}, q by Diophantine approximation (function DiophAppr) takes, as we know, time $O(\log \max(Q, m_0^2)) = O(\log n)$. The number of iterations of the outer loop in NewSegSieve is

$$\ll \sqrt{\frac{4n}{\Delta}} \cdot \log \frac{\sqrt{n}}{K\Delta} \le \sqrt{\frac{n}{\Delta}} \log n, \tag{4.2}$$

and thus the total time taken by the instructions inside the outer loop but outside the inner loop is $O(\sqrt{n/\Delta} (\log n)^2)$.

The time it takes to go over and sieve by all $m \in [M, M+2R]$ congruent to $m_0 - a^{-1}(c+j) \bmod q$ for all $-k-1 \le j \le k+1$ (where $k = \lfloor \eta q \rfloor$) is at most

$$\begin{aligned} \left\lceil \frac{2R+1}{q} \right\rceil \cdot (\lfloor \eta q \rfloor + 3) &\le \left(\frac{2R}{q} + 2 \right) (\eta q + 3) \ll (R+q)\eta + \frac{R}{q} + 1 \\ &\ll \frac{\Delta^{3/2}}{\sqrt{n}} + \frac{R}{q} + 1, \end{aligned} \tag{4.3}$$

since $R = \lfloor \sqrt{\Delta/4n} \cdot M \rfloor$, $\eta = 5\Delta/4M$, and $q \le R$.

Since the number of times the main loop is executed—that is, the number of intervals $[M, M+2R]$ we consider—is given by (4.2), the total contribution of the

[5] Actually, in the given reference, Vinogradov gives a bound of $O(x^{1/3}(\log x)^2)$ on the error term in (4.1). The reason is simply that he did not choose his parameters optimally: if the parameter τ in his proof is set to the value of τ in the $(A \log A)^{1/3}$ rather than A, the resulting bound is indeed as in (4.1).

first and last terms in the last line of (4.3) is

$$\ll \left(\frac{\Delta^{3/2}}{\sqrt{n}} + 1\right)\sqrt{\frac{n}{\Delta}}\log n \ll \Delta \log n,$$

since $\Delta \geq n^{1/3}$. It remains to account for the contribution of R/q. We may assume M is large enough for R to be ≥ 3, as otherwise the contribution of R/q is bounded by thrice the contribution of the last term 1.

Now we proceed much as in [Vin54, Ch. III, exer. 3–6]. We will examine how $\alpha_1 = \{-n/m_0^2\}$ changes as m_0 increases; we will then be able to tell how often Diophantine approximations a/q to α_1 with given q can occur. To be precise: we will see by how much m_0 has to increase for $-n/m_0^2$ to increase by 1 or more, and we also want to know for how long $-n/m_0^2$ can have a given, fixed Diophantine approximation a/q as m_0 increases.

Consider two intervals $[m_0 - R, m_0 + R]$, $[m_0' - R', m_0' + R']$, where $m_0 < m_0'$ and $R \leq R'$. Then

$$\left(-\frac{n}{(m_0')^2}\right) - \left(-\frac{n}{m_0^2}\right) = n\frac{(m_0')^2 - m_0^2}{m_0^2(m_0')^2}. \tag{4.4}$$

If $\alpha_1 = -n/m_0^2$ is $a/q + O^*(1/qR)$ and $\alpha_1' = -n/(m_0')^2$ is $a/q + O^*(1/qR')$, it follows that $n/m_0^2 - n/(m_0')^2$ is $O^*(1/qR + 1/qR') = O^*(2/qR)$. Suppose that this is the case.

Since $\Delta < n$, we know that $M' := m_0' - R' \leq \sqrt{n+\Delta} < \sqrt{2n}$, $R' \leq M'\sqrt{\Delta/4n} < M'/2$ and $m_0' = M' + R' < (\sqrt{2}+1/2)\sqrt{n} < 2\sqrt{n}$; in the same way, $m_0 = M + R < 3M/2$. Clearly, $m_0' \leq 2m_0$, as otherwise $n/m_0^2 - n/(m_0')^2 \geq 3n/4m_0^2 \geq 3n/(m_0')^2 > 3/4$, giving us a contradiction to $2/qR \leq 2/R \leq 2/3$. Hence

$$\frac{2}{qR} \geq n\frac{(m_0')^2 - m_0^2}{m_0^2(m_0')^2} = n \cdot \frac{m_0' - m_0}{m_0^2} \cdot \frac{m_0' + m_0}{(m_0')^2} \geq \frac{3n}{4} \cdot \frac{m_0' - m_0}{m_0^3}.$$

In other words, when the same approximant a/q is valid at m_0 and m_0',

$$m_0' - m_0 \leq \frac{8}{3}\frac{m_0^3}{qRn}.$$

The number of intervals for which $\alpha_1 = -n/m_0^2$ has a given approximation a/q is thus at most

$$\begin{aligned}\frac{8}{3}\frac{m_0^3}{qR(2R+1)n} + 1 \ll \frac{m_0^3}{q(R+1)^2 n} + 1 &\leq \frac{m_0^3}{qM^2\Delta/4} + 1 \\ &< (3/2)^2\frac{4m_0}{q\Delta} + 1 \ll \frac{m_0}{q\Delta} + 1.\end{aligned}$$

We should also see when $n/m_0^2 - n/(m_0')^2 \geq 1$, or rather when $n/m_0^2 - n/(m_0')^2 \geq 1 - 2/R$, as then n/m_0^2 and $n/(m_0')^2$ may have Diophantine approximations that, while distinct, are congruent $\bmod 1$, i.e., differ by an integer. By (4.4), the inequality $n/m_0^2 - n/(m_0')^2 \geq 1 - 2/R$ is fulfilled exactly when

$$n((m_0')^2 - m_0^2) \geq m_0^2(m_0')^2(1 - 2/R),$$

and, since $R \geq 3$, that implies

$$6n(m_0' - m_0) > m_0^3.$$

Hence, for given m_0, it makes sense to consider all following intervals with $m_0' \le m_0 + m_0^3/6n$. In that range, n/m_0^2 and $n/(m_0')^2$ can have the same Diophantine approximation $\bmod 1$ only if they in fact have the same Diophantine approximation.

Since R increases and the intervals are of width $2R$, there will be at most $m_0^3/12Rn + 1 \ll m_0^2/\sqrt{\Delta n} + 1 \ll m_0^2/\sqrt{\Delta n}$ intervals with $m_0' \le m_0 + m_0^3/6n$. (Note that $m_0^2/\sqrt{\Delta n} \ge K^2\Delta^2/\sqrt{\Delta n} > 1$.) Among those intervals, $\ll m_0'/q\Delta + 1 \ll m_0/q\Delta + 1$ will have a given approximation $a/q \bmod 1$.

As we said before, we have to account for the total contribution of R/q. Since $1/q$ and $1/q^2$ decrease as q increases, the worst-case scenario is for there to be as many m_0's as possible for which the approximation has q as small as possible. We would have $\lfloor O(m_0/q\Delta + 1)\phi(q) \rfloor = O(m_0/\Delta + q)$ values of a/q with $q \le Q$, and none for $q > Q$, where $Q \ll m_0/(\Delta n)^{1/4}$. (To bound Q, we apply $\sum_{q \le Q} \phi(q) = |\{1 \le q_1, q_2 \le Q : q_1,\ q_2 \text{ coprime}\}| \gg Q^2$.) The contribution of R/q for all intervals with $m_0' \le m_0 + m_0^3/6n$ is thus

$$\ll R \sum_{q \le Q} \left(\frac{m_0}{\Delta q} + 1 \right) \ll R \cdot \left(\frac{m_0}{\Delta} \log Q + Q \right).$$

Now split $[M_0, 2M_0]$ into $O(n/M_0^2)$ chunks of the form $[m_0, m_0(1 + m_0^2/6n)]$. We obtain that the contribution for all intervals with m_0 inside $[M_0, 2M_0]$ is

$$\begin{aligned} &\ll \frac{n}{M_0^2} \cdot 2M_0 \sqrt{\frac{\Delta}{4n}} \cdot \left(\frac{M_0}{\Delta} \log n + \frac{M_0}{(\Delta n)^{1/4}} \right) \\ &\ll \frac{\sqrt{n}}{\sqrt{\Delta}} \log n + (n\Delta)^{1/4}, \end{aligned}$$

and thus the total contribution will be

$$\ll \frac{\sqrt{n}}{\sqrt{\Delta}} (\log n)^2 + (n\Delta)^{1/4} \log n.$$

We conclude that the total time consumption of Algorithm 1 is

$$\begin{aligned} &\ll \Delta \log\log K\Delta + \Delta \log n + \frac{\sqrt{n}}{\sqrt{\Delta}} (\log n)^2 + (n\Delta)^{1/4} \log n \\ &\ll \Delta \log n + \sqrt{\frac{n}{\Delta}} (\log n)^2, \end{aligned}$$

since $\Delta \ge n^{1/3}$. Under the stronger assumption $\Delta \ge n^{1/3} \log^{2/3} n$, we obtain that the total time consumption is

$$O(\Delta \log n).$$

Time consumption of Algorithm 6. We must now analyze Algorithm 6, which factors integers in the interval $[n - \Delta, n + \Delta]$. Everything is much the same as for Algorithm 1, except in one respect. The total space taken will be

$$O(\Delta \log n)$$

rather than $O(\Delta)$, simply because storing the list of prime factors of an integer in $[n - \Delta, n + \Delta]$ takes $O(\log n)$ bits. (To wit: the number of bits taken to store a factor p^α is $O(\log p) + O(\log(\alpha + 1))$, which is $O(\alpha \log p)$. Hence, storing all the factors $p_i^{\alpha_i}$ of an integer $n = p_1^{\alpha_1} \cdots p_k^{\alpha_k}$ takes $O\left(\sum_i \alpha_i \log p_i\right) = O(\log n)$ bits.)

The total time estimate is still

$$O\left(\Delta \log n\right).$$

The claims in (1.1) and (1.2) follow immediately once one splits the interval $[1, N]$ into intervals of length 2Δ. The main theorem is thus proved.

5. Further perspectives

It is tempting to try to improve on this algorithm by taking a longer truncated Taylor expansion in (2.1). However, this would require us to find the solutions $x \in \{-R, -R+1, \dots, R\}$ to a $P(x) \in [-\eta, \eta] \bmod \mathbb{Z}$, where P is a polynomial with $\deg P \geq 2$ and $\eta \ll 1/R$ or thereabouts.

Solving a quadratic modular equation $a_2x^2 + a_1x + a_0 \equiv 0 \bmod q$ is not the main difficulty in our context. We can obviously reduce such a problem to taking square roots $\bmod q$. Now, the problem of finding square roots modulo q (for q arbitrary) is well known to be equivalent to factoring q [Rab79]. (There is a classical algorithm (Tonelli-Shanks) for finding square roots to prime modulus.) This is not a problem, since we can factor all integers $q \leq x^{1/4}$ (say) in advance, before we start sieving.

The problem is that it is not at all clear how to reduce finding solutions to $P(x) \in [-\eta, \eta] \bmod \mathbb{Z}$, $\deg P = 2$, to finding solutions to quadratic equations $\bmod q$. We can try to find rational approximations with the same denominator q (*simultaneous Diophantine approximation*) to the non-constant coefficients of P, but such approximations will be generally worse than when we approximate a single real number, and so matters do not work out: we need a more, not less, precise approximation than before to the leading coefficient, since it is now the coefficient of a quadratic term.

Concretely, for $P(x) = \alpha_2x^2 + \alpha_1x + \alpha_0$, in order to reduce the problem of finding solutions to $P(r) \in [-\eta, \eta] \bmod \mathbb{Z}$ with $r \in [R, R] \cap \mathbb{Z}$ to the problem of solving a quadratic modular equation, we would need a_1, a_2, q with $|\alpha_2 - a_2/q| \leq 1/qR^2$, $|\alpha_1 - a_1/q| \leq 1/qR$. In general, we can do such a thing only for $q \gg R^3$, and that is much too large. For one thing, for $\epsilon \sim 1/R$, we would have to solve $\epsilon q \sim R^2$ distinct equations, and that is obviously too many.

5.1. **Geometric interpretation.** As we have seen, sieving integers in the interval $[n-\Delta, n+\Delta]$ reduces to finding integers m such that $\{n/m\}$ is close to 0 modulo 1. This task is equivalent to finding integer points close to a hyperbola $x \mapsto n/x$. Seen from this perspective, our approach consists simply in approximating a hyperbola locally by linear functions. Such a geometric perspective is already present (and dominant) in [Vor03].

What we did in §2 then amounts to finding points close to a line, starting with an approximation of the slope by a rational, and then proceeding by modular arithmetic.

Taking one more term in the Taylor expansion would be the same as approximating a hyperbola by segments of parabolas, rather than by segments of lines. We could then take longer segments, and thus hope to capture points near the hyperbola efficiently even when Δ is considerably smaller than $n^{1/3}$. (We can hope to capture them efficiently because the segments are long enough that we expect at least one point in each segment.) The problem of how to find the points remains. It seems difficult to do so without reducing a non-linear problem $\bmod 1$ to a problem $\bmod q$, and we do not yet know how to carry out such a reduction well.

6. A few words on the implementation

We have written and tested a simple implementation as proof-of-purpose, that is, mainly so as to check that the algorithms work as described. On the higher end of what can be done in 64-bit computer arithmetic (say: $n = 5 \cdot 10^{18}$ and $\Delta = 2 \cdot 10^7$ or $\Delta = 4 \cdot 10^7$), our implementation of algorithm NewSegSiev runs substantially faster than our own implementation of the traditional algorithm SegSiev. Still, on those same inputs, our implementation of NewSegSiev is clearly slower (by a factor between 2 and 2.5) than a publicly available, highly optimized implementation [Wal] of the traditional algorithm (basically SegSiev, but improved in all the ways detailed below, and some other ones) on the same interval. Diophantine approximation (Algorithm 4) turns out to take less than 2 percent of the running time of NewSegSiev, so the fact that [Wal] runs faster is due to better coding, and not to the overhead of Diophantine approximation. For the values of n and Δ above, we are talking about total running times of only a few seconds in all cases.

The advantage of NewSegSieve over existing methods should become clearer as n grows larger—meaning substantially larger than 2^{64}. However, on the range $n > 2^{64}$, it seems harder to find highly optimized implementations of the traditional algorithm for comparison.

Here are a few hints for the reader who would like to write a more serious program on his or her own. Most of these tricks are standard, but are scattered here and there in the literature (and in code).

(1) Obviously, we can save on space by storing the sieve as a bit array. Saving on space leads to better cache usage, and hence often to time savings as well. Our very conventions for space usage (expressed in terms of bits, rather than words) reflect this fact. Of course, in some ranges, it can save time to be a little more wasteful and store some integer data as words (e.g., prime factors, in the case of the sieve applied to factorization).

(2) We can first apply a simple sieve to the integers between 1 and $M = \prod_{p \le p_0} p$ (where $p_0 = 17$, say), taking only the effect of primes $p \le p_0$ into account (whether we are sieving for primality, or computing the Möbius function μ, or factoring numbers: in the nth entry, we would store whether n is coprime to P, or, instead, store $\mu(\gcd(n, M^2))$, if we are computing μ, or the set $\{p \le p_0 : p|n\}$, if we are factoring numbers). We then initialize our sieve by repeating that block of length M, and so we do not need to sieve by the primes $p \le p_0$ ever again.

(3) We can of course implement a sieve in parallel in a trivial sense, by letting different processors look at different intervals (of length at least $n^3(\log n)^{2/3}$, in our case). There seems to be a small literature on parallel implementations of sieves; see, e.g., [SP94].

(4) At the end of §2, we went briefly over the issue of "false alarms", that is, values of m that lie in a valid congruence class $m \equiv m_0 + r_0 \bmod q$ but do not actually have multiples within the interval I we are sieving. Such false alarms do not change the outcome of the algorithm, but they do waste some time. It is possible to avoid them; the details are given in the first version of the present paper, available on arXiv.org. There, in fact, the algorithm without "false alarms" is called NewSegSiev, and presented as the main version of the algorithm. Unfortunately, at least in my implementation, the process of eliminating false alarms takes more time than it saves.

(5) As we said in the introduction, Oliveira e Silva has shown how to use cache efficiently when implementing a sieve of Eratosthenes [eS], [OeSHP14, Algorithm 1.2]. In effect, when sieving an interval of length L, he needs not much more than $\sqrt{L}$ units of memory in cache. There seems to be no reason why Oliveira e Silva's technique cannot be combined with the algorithms put forward here. Thus we could hope to sieve intervals of the form $[n-\Delta, n+\Delta]$, $\Delta \sim n^{1/3}(\log n)^{2/3}$, while using no more than $O(n^{1/6}(\log n)^{1/3})$ units of memory in cache at a time (and $O(n^{1/3}(\log n)^{2/3})$ units of memory in total). The range up to about $n \sim 10^{36}$ could then become accessible, at least for short or medium-sized intervals. Note that we can still use 128-bit-integer arithmetic in that range.

Acknowledgments

Thanks are due to Manuel Drehwald, for having written code implementing an earlier version of the algorithm, to Lola Thompson, for helpful suggestions, and to an anonymous referee, for several useful remarks.

References

[AB04] A. O. L. Atkin and D. J. Bernstein, *Prime sieves using binary quadratic forms*, Math. Comp. **73** (2004), no. 246, 1023–1030, DOI 10.1090/S0025-5718-03-01501-1. MR2031423

[eS] T. Oliveira e Silva, *Fast implementation of the segmented sieve of Eratosthenes*, `http://sweet.ua.pt/tos/software/prime_sieve.html`. Accessed: 2016-6-22.

[Gal00] W. F. Galway, *Dissecting a sieve to cut its need for space*, Algorithmic Number Theory (Leiden, 2000), Lecture Notes in Comput. Sci., vol. 1838, Springer, Berlin, 2000, pp. 297–312, DOI 10.1007/10722028_17. MR1850613

[Hur18] G. Hurst, *Computations of the Mertens function and improved bounds on the Mertens conjecture*, Math. Comp. **87** (2018), no. 310, 1013–1028, DOI 10.1090/mcom/3275. MR3739227

[HW79] G. H. Hardy and E. M. Wright, *An introduction to the theory of numbers*, 5th ed., The Clarendon Press, Oxford University Press, New York, 1979. MR568909

[Khi97] A. Ya. Khinchin, *Continued Fractions*, Translated from the third (1961) Russian edition, Dover Publications, Inc., Mineola, NY, 1997. With a preface by B. V. Gnedenko; Reprint of the 1964 translation. MR1451873

[Lan12] E. Landau, *Die Bedeutung der Pfeifferschen Methode für die analytische Zahlentheorie*, Wien. Ber. **121** (1912), 2196–2332.

[OeSHP14] T. Oliveira e Silva, S. Herzog, and S. Pardi, *Empirical verification of the even Goldbach conjecture and computation of prime gaps up to* $4 \cdot 10^{18}$, Math. Comp. **83** (2014), no. 288, 2033–2060, DOI 10.1090/S0025-5718-2013-02787-1. MR3194140

[OtR85] A. M. Odlyzko and H. J. J. te Riele, *Disproof of the Mertens conjecture*, J. Reine Angew. Math. **357** (1985), 138–160, DOI 10.1515/crll.1985.357.138. MR783538

[Pfe86] E. Pfeiffer, *Über die Periodicität in der Teilbarkeit der Zahlen und über die Verteilung der Klassen positiver quadratischer Formen auf ihre Determinanten*, Jahresbericht der Pfeiffer'schen Lehr- und Erziehungs-Anstalt zu Jena, 1886, pages 1–21.

[Pri83] P. Pritchard, *Fast compact prime number sieves (among others)*, J. Algorithms **4** (1983), no. 4, 332–344, DOI 10.1016/0196-6774(83)90014-7. MR729229

[Rab79] M. O. Rabin. Digitalized signatures and public-key functions as intractable as factorization. Technical report, Cambridge, MA, USA, 1979.

[Sie06] W. Sierpiński, *O pewnem zagadnieniu z rachunku funkcyj asymptotycznych*, Prace matematyczno-fizyczne **1** (1906), no. 17, 77–118.

[Sin69] R. C. Singleton, *Algorithm 357: an efficient prime number generator*, Comm. ACM **12** (1969), 563–564.

[Sor98] J. P. Sorenson, *Trading time for space in prime number sieves*, Algorithmic Number Theory (Portland, OR, 1998), Lecture Notes in Comput. Sci., vol. 1423, Springer, Berlin, 1998, pp. 179–195, DOI 10.1007/BFb0054861. MR1726070

[Sor06] J. P. Sorenson, *The pseudosquares prime sieve*, Algorithmic Number Theory, Lecture Notes in Comput. Sci., vol. 4076, Springer, Berlin, 2006, pp. 193–207, DOI 10.1007/11792086_15. MR2282925

[SP94] J. Sorenson and I. Parberry, *Two fast parallel prime number sieves*, Inform. and Comput. **114** (1994), no. 1, 115–130, DOI 10.1006/inco.1994.1082. MR1294306

[TCH12] T. Tao, E. Croot III, and H. Helfgott, *Deterministic methods to find primes*, Math. Comp. **81** (2012), no. 278, 1233–1246, DOI 10.1090/S0025-5718-2011-02542-1. MR2869058

[Vin54] I. M. Vinogradov, *Elements of number theory*, Dover Publications, Inc., New York, 1954. Translated by S. Kravetz. MR0062138

[Vor03] G. Voronoï, *Sur un problème du calcul des fonctions asymptotiques* (French), J. Reine Angew. Math. **126** (1903), 241–282, DOI 10.1515/crll.1903.126.241. MR1580627

[Wal] K. Walisch, *primesieve: fast* C/C++ *prime number generator*, `http://primesieve.org`. Accessed: 2016-6-21.

Mathematisches Institut, Georg-August Universität Göttingen, Bunsenstrasse 3-5, D-37073 Göttingen, Germany; and IMJ-PRG, UMR 7586, 58 avenue de France, Bâtiment S. Germain, case 7012, 75013 Paris CEDEX 13, France

Email address: `harald.helfgott@gmail.com`