MATHEMATICS OF COMPUTATION
Volume 00, Number 0, Pages 000–000
S 0025-5718(XX)0000-0

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 essentially linear time. These estimates improves over the usual versions, which take space about $O(\sqrt{N})$ and essentially linear time. We can also apply our sieve to any subinterval of $[1, N]$ of length $\Omega\left(N^{1/3}\right)$ in time essentially linear on the length of the interval. Before, this 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 essentially linear 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 roughly 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 in batches to a program that needs it and can handle it sequentially. Then the space used is certainly $O(\sqrt{N})$.

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

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 essentially linear time. Like the sieve in [AB04], on which it is based[1], Galway's sieve is specific to finding prime numbers.

We will show how to implement a sieve of Eratosthenes in space close to $O(N^{1/3})$ and essentially 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 $p \leq N$ 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 $n \leq N$ in

$$\text{space } O\left(N^{1/3}(\log N)^{2/3} \log\log N\right) \quad \text{and} \quad \text{time } O(N \log N). \tag{1.2}$$

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

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

[1] 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

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

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 $\tau(n)$ is 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]).[2] 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 essentially 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. My initial interest in the problem stemmed from the fact that I had to compute values of $\mu(n)$ so as to check inequalities of the form $\sum_{n\le x} \mu(n) \le \sqrt{x}$ (Mertens), $\sum_{n\le x} \mu(n)/n \le 2/\sqrt{x}$, 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.

[2]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.

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 $|f(x)| \leq Cg(x)$ for all large enough x. We write $f(x) = O^*(g(x))$ to mean that $|f(x)| \leq g(x)$ for all 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.

Acknowledgements. The author is currently supported by funds from his Humboldt Professorship. Thanks are due to Manuel Drehwald, for having written code implementing an earlier version of the algorithm, and to Lola Thompson, for helpful suggestions.

2. Analysis of the problem

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

2.1. **Reduction to an approximate linear equation** mod 1**.** Our motivation for asking this question is that we will be taking intervals I and sieving them by all integers, or all primes, $m \leq \sqrt{x}$. Going over all integers $\leq \sqrt{x}$ would take time at least $\sqrt{x}$, which could be much larger than Δ. (Going over all primes $\leq \sqrt{x}$ would take time at least $\sqrt{x}/\log\sqrt{x}$, in addition to begging the question of how one is supposed to store all such primes in space $O(\Delta)$.)

We will be able to sieve our intervals by $m \leq \sqrt{x}$ by producing a list of those m which might have multiples in I, without testing all $m \leq \sqrt{x}$ 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 $K \geq 2$ be a parameter to be set later. If $m \leq K\Delta$, 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

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

where the fractional part $\{\alpha\}$ of $\alpha \in \mathbb{R}$ is defined to lie in $\{-1/2, 1/2]$, say.

Say we have already dealt with all $m \leq M$, where $M \geq K\Delta$, 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 $\kappa > 0$ is a constant we will set later. (That is, we allow our error term to be not much larger than the interval we are trying to hit.) We do this 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 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 $\eta = (1+\kappa)\Delta/M$. In other words, we are being asked to find all approximate solutions to a linear equation in $\mathbb{R}/\mathbb{Z}$ efficiently.

2.2. **Solving an approximate linear equation** $\bmod 1$**.** Finding all approximate solutions to a linear equation in $\mathbb{R}/\mathbb{Z}$ is something we can do in general.

Lemma 2.1. *Let $P(r) = \alpha_1 r + \alpha_0$, where $\alpha_0 = a_0/q_0$, $\alpha_1 = a_1/q_1$. Let $R \ge 1$, $0 \le \eta \le 1/2$. Then the ℓ integers $-R \le r \le R$ such that*

$$\alpha_1 r + \alpha_0 \in [-\eta, \eta] \bmod \mathbb{Z}$$

can be listed in time $O(\log N) + O(\ell)$ and space $O(1)$, where $N = \max(q_0, q_1, R)$.

It is understood, of course that, the output is given sequentially, and need not be all stored; this is how space can be $O(1)$.

Proof. 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 5; 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, $p_n^{-1} = (-1)^{n+1} q_{n-1} \bmod q_n$.

For each $-k-1 \le j \le k+1$, let ρ_j be the integer r in $(-q/2, q/2]$ satisfying (2.6). We should now consider all $r \in [-R, R]$ congruent to $\rho_j \bmod q$ and see which satisfy $P(r) \in [-\eta, \eta]$.

If $-k+1 \le j \le k-1$, then all such r satisfy $P(r) \in [-\eta, \eta]$, since, by (2.4) and (2.6),

$$P(r) \equiv \frac{c + a\rho_j}{q} + O^*\left(\frac{1}{q}\right) = -\frac{j}{q} + O^*\left(\frac{1}{q}\right) \bmod q.$$

Listing all such r obviously takes time proportional to their number.

It remains to consider j with $|j| = k, k+1$. Write $r = sq + \rho_j$; compute ρ_j and $P(\rho_j)$. Let $\delta = q\alpha_1 - a$, so that $\alpha_1 = a/q + \delta/q$ and $|\delta| \le 1/Q$. Then

$$P(r) \equiv \delta s + (\rho_j \alpha_1 + \alpha_0) \bmod 1,$$

and $\rho_j \alpha_1 + \alpha_0 = P(\rho_j)$. The question is for which s within the range $I_j = ([-R, R] - \rho_j)/q$ the image $\phi(s)$ under the linear map

$$\phi : s \mapsto \delta s + P(\rho_j) \bmod 1$$

lies in $[-\eta, \eta]$. In other words, we have reduced our problem to that of solving a linear equation $\bmod 1$, with a small leading coefficient δ.

What do we mean by "small"? For $s \in I_j$, because $|\rho_j| \le q/2$,

$$|\delta s| \le \left| \delta \cdot \frac{R + q/2}{q} \right| \le \left| \delta \left(\frac{R}{q} + \frac{1}{2} \right) \right| \le \frac{1}{2q} + \frac{1}{2Q} \le \frac{1}{q}.$$

If $q = 1$, then $\rho_j = 0$, and we actually obtain that $|\delta s| \le 1/2q$. Hence, for all q, $|\delta s| \le 1/2$. This means that there is no "wrap-around", i.e., the image of I_j under $s \mapsto \delta s + P(\rho_j)$ is contained in an interval of length 1.

In particular, this means that the interval $L = \delta I_j + P(\rho_j)$ intersects at most two intervals of the form $n + [-\eta, \eta]$, $n \in \mathbb{Z}$. These n (call them n_-, n_+) are trivial to determine. The preimage of $n_\pm + [-\eta, \eta]$ under the map $s \mapsto \delta s + P(\rho_j)$ is just the interval $\delta^{-1} \cdot (n_\pm + [-\eta, \eta] - P(\rho_j))$ (if $\delta \neq 0$) or either $\mathbb{R}$ or $\emptyset$ (if $\delta = 0$). Hence, $\phi^{-1}([-\eta, \eta]) \cap I_j$ is either the empty set, an interval, or the union of two intervals.

Thus, s simply has to go over at most two runs of consecutive integers contained in I_j. Hence we obtain all $r \equiv \rho_j \bmod q$ such that $P(r) \in [-\eta, \eta]$, in time proportional to their number plus $O(\log N)$. □

If we do not mind "false positives" then we can simply find find a^{-1}, q and c, and work with all integers $-R \le r \le R$ obeying (2.6) for some $-k-1 \le j \le k+1$. We will later analyze the time complexity resulting from such a simplified procedure. "False positives" does not mean that the final results would be incorrect, as in supposed primes that are not prime; it simply means that we would waste some time on integers m that have no multiples in the interval $[n-\Delta, n+\Delta]$.

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– 5), with some commentary.

We also give a simplified version of the main algorithm as Algorithm 6. It is simpler in the sense that, while, like the main version, it uses Algorithms 2, 3 and 5, it does not use Algorithm 4. The running time of the simplified version will differ from that of the main one.

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

function NewSegSiev(n,Δ,K,κ)
Ensure: $S_j = 1$ if $n+j$ is prime, $S_j = 0$ otherwise, for $-\Delta \le j \le \Delta$
Require: $\Delta \ge \sqrt[3]{n/\kappa K^2}$, where $\kappa > 0$, $K \ge 2(1+\kappa)$ ▷ $\kappa = 1/2$, $K = 3$ will do
  $S' \leftarrow$ SubSegSiev($n - \lfloor \Delta \rfloor, 2\Delta, K\Delta$) ▷ Sieve by all $p \le K\Delta$
  $S_j \leftarrow S'_{j+\lfloor \Delta \rfloor}$ for all $-\Delta \le j \le \Delta$
  $M \leftarrow \lfloor K\Delta \rfloor + 1$
  while $M \le \sqrt{n+\Delta}$ **do**
    $R \leftarrow \lfloor M\sqrt{\kappa\Delta/n} \rfloor$, $m_0 \leftarrow M + R$
    $\alpha_1 \leftarrow -n/m_0^2$, $\alpha_0 \leftarrow n/m_0$, $\eta \leftarrow (1+\kappa)\Delta/M$
    $\mathbf{r} \leftarrow$ SolveModZ($\alpha_1, \alpha_0, R, \eta$)
    for $r \in \mathbf{r}$ **do**
      $m \leftarrow m_0 + r$ ▷ We will sieve by m
      $n' \leftarrow \lfloor (n + \lfloor \Delta \rfloor)/m \rfloor \cdot m$ ▷ n' is a multiple of m
      if $(n' \in [n-\Delta, n+\Delta]) \wedge (n > m)$ **then**
        $S_{n'-n} \leftarrow 0$ ▷ thus sieving out n'
    $M \leftarrow M + 2R + 1$
  return S

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, $\mathrm{den}(\alpha) > 0$).

Algorithm 2 A simple sieve of Eratosthenes

```
1: function SimpleSiev(N)
Ensure: for 1 ≤ n ≤ N, P_n = 1 if n is prime, P_n = 0 otherwise
2:     P_1 ← 0, P_2 ← 1, P_n ← 0 for n ≥ 2 even, P_n ← 1 for n ≥ 3 odd
3:     m ← 3, n ← m · m
4:     while n ≤ N do
5:         if P_m = 1 then
6:             while n ≤ N do                                   ▷ [sic]
7:                 P_n ← 0, n ← n + 2m    ▷ sieves odd multiples ≥ m^2 of m
8:         m ← m + 2, n ← m · m
9:     return P
```

Time: $O(N \log\log N)$. **Space:** $O(N)$.

In Algorithms 1 and 6 (as in Algorithm 8, to be discussed later), the assignments $\alpha_1 \leftarrow -n/m_0^2$, $\alpha_0 \leftarrow n/m_0$ may be changed to $\alpha_1 \leftarrow \{-n/m_0^2\}$, $\alpha_0 \leftarrow \{n/m_0\}$, where fractional parts may be taken either in the sense we give elsewhere to them (that is, $\{\alpha\}$ is the element of $(-1/2, 1/2]$ congruent to $\alpha \bmod \mathbb{Z}$) or in the other common sense (namely, $\{\alpha\}$ is the element of $[0, 1)$ congruent to $\alpha \bmod \mathbb{Z}$). The reason is that the output of SolveModZ depends on α_0, α_1 only $\bmod \mathbb{Z}$. The motivation would be to keep variable sizes small.

3.1. **Sieving for primes.** The main function is NewSegSiev (Algorithm 1). It takes as inputs n and Δ, as well as two parameters K and κ that affect time and space consumption, and will be discussed later. The basic procedure is easy to summarize. NewSegSiev uses SubSegSiev (Algorithm 3), a segmented sieve of traditional type, to sieve for primes $p \le K\Delta$. It then proceeds as detailed in §2.1 to find the integers m in intervals $[M, M + 2R]$ that divide at least one integer in the interval $[n - \Delta, n + \Delta]$. Then it sieves by them.

The way we find such integers m in $[M, M + 2R]$ is by solving a linear equation $\bmod 1$. The solution was explained in §2.2; here we give it again in full as Algorithm 4. The function we call is SolveModZ. It starts by calling 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 $a^{-1} \bmod q$. Then it uses that approximation to reduce the problem to that of solving a linear equation $\bmod 1$ with small leading coefficient δ. Function SmallSolveModZ solves this problem by reducing it to that of solving a linear equation in $\mathbb{R}$ – a trivial problem (function SolveInR).

The simplified version of the main algorithm (SimpleNewSegSiev) is just like NewSegSieg, except that, instead of calling SolveModZ, we call DiophAppr to find a^{-1} and q, and then sieve by all m in $[M, M + 2R]$ in certain congruence classes $\bmod q$ given in terms of a^{-1}.

Algorithm 3 Segmented sieves of Eratosthenes, traditional version

function SIMPLESEGSIEV(n,Δ,M) ▷ sieves $[n, n+\Delta]$ by primes $p \le M$
Ensure: $S_j = \begin{cases} 0 & \text{if } n+j = 0, 1 \text{ or if } p \mid (n+j) \text{ for some } p \le M \\ 1 & \text{otherwise} \end{cases}$
  $S_j \leftarrow 1$ for all $0 \le j \le \Delta$
  $S_j \leftarrow 0$ for $0 \le j \le 1-n$
  $P \leftarrow$ SIMPLESIEV(M)
  for $m \le M$ **do**
    if $P_m = 1$ **then**
      $n' \leftarrow \max(m \cdot \lceil n/m \rceil, 2m)$
      while $n' \le n+\Delta$ **do** ▷ n' goes over mults. of m in $n + [0, \Delta]$
        $S_{n'-n} \leftarrow 0$, $n' \leftarrow n' + m$
  return S

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

function SUBSEGSIEV(n,Δ,M) ▷ sieves $[n, n+\Delta]$ by primes $p \le M$
Ensure: $S_j = \begin{cases} 0 & \text{if } n+j = 0, 1 \text{ or if } p \mid (n+j) \text{ for some } p \le M \\ 1 & \text{otherwise} \end{cases}$
  $S_j \leftarrow 1$ for all $\max(0, 2-n) \le j \le \Delta$
  $S_j \leftarrow 0$ for $0 \le j \le 1-n$
  $\Delta' \leftarrow \lfloor \sqrt{M} \rfloor$, $M' \leftarrow 1$
  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 m is a prime...
        $n' \leftarrow \max(p \cdot \lceil n/p \rceil, 2p)$
        while $n' \le n+\Delta$ **do**
          $S_{n'-n} \leftarrow 0$, $n' \leftarrow n' + p$
    $M' \leftarrow M' + \Delta' + 1$
  return S

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

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

We could try to improve on NEWSEGSIEVE or SIMPLENEWSEGSIEV by throwing out even values of m, say; as they are certainly not prime, we need not sieve by them.

Algorithm 4 Solving an approximate linear equation $\bmod \mathbb{Z}$

function SolveModZ(α_1,α_0,R,η)
Ensure: returns the set of all $-R \le r \le R$ s.t. $\alpha_1 r + \alpha_0 \in [-\eta, \eta] \bmod \mathbb{Z}$
Require: $0 \le \eta \le 1/2$
    $(a, a^{-1}, q) \leftarrow$ DiophAppr($\alpha_1, 2R$)
    $c \leftarrow \lfloor \alpha_0 q + 1/2 \rfloor$, $\delta \leftarrow q\alpha_1 - a$, $k \leftarrow \lfloor \eta q \rfloor$, $\mathbf{r} \leftarrow \emptyset$
    for $-k - 1 \le j \le k + 1$ **do**
        $\rho \leftarrow -a^{-1}(c + j) \bmod q$
        if $|j| \le k - 1$ **then**
            include in $\mathbf{r}$ every $r \in \{-R, -R+1, \dots, R\}$ s.t. $r \equiv \rho \bmod q$
        else
            $I \leftarrow ([-R, R] - \rho)/q$
            $\mathbf{s} \leftarrow$ SmallSolveModZ($\delta, \rho\alpha_1 + \alpha_0, I, \eta$)
            For every $s \in \mathbf{s}$, **include** $sq + \rho$ in $\mathbf{r}$
    return r

Time: $O(|\mathbf{r}| + \log \max(R, \mathrm{den}(\alpha))) + 1$, where $|\mathbf{r}|$ = size of $\mathbf{r}$
Space: $O(|\mathbf{r}| + 1)$ as written, but $O(1)$ if output on demand

function SmallSolveModZ(α_1,α_0,$I = [I_-, I_+]$, η)
Ensure: returns the set of all $r \in I \cap \mathbb{Z}$ s.t. $\alpha_1 r + \alpha_0 \in [-\eta, \eta] \bmod \mathbb{Z}$
Require: $\alpha_1 I \subset [-1/2, 1/2]$, $0 \le \eta \le 1/2$
    $[L_-, L_+] \leftarrow \alpha_1 [I_-, I_+] + \alpha_0$
    $n_- \leftarrow \lfloor L_- + \eta \rfloor$, $n_+ \leftarrow \lfloor L_+ + \eta \rfloor$
    $\mathbf{r} \leftarrow$ SolveInR($\alpha_1, \alpha_0, I, n_- + [-\eta, \eta]$)
    if $n_+ > n_-$ **then** ▷ $n_+ = n_- + 1$
        $\mathbf{r} \leftarrow \mathbf{r} \cup$ SolveInR($\alpha_1, \alpha_0, I, n_+ + [-\eta, \eta]$)
    return r

Time: $O(\ell + 1)$, where ℓ = size of output.
Space: $O(\ell + 1)$ as written, but $O(1)$ if output on demand

function SolveInR(α_1, α_0, $I = [I_-, I_+]$, $J = [J_-, J_+]$)
Ensure: returns the set of all $s \in I \cap \mathbb{Z}$ s.t. $\alpha_1 s + \alpha_0 \in J$, where $I, J \subset \mathbb{R}$
    if $\alpha_1 = 0$ **then** ▷ Checking whether $\alpha_1 = 0$ is trivial for $\alpha_1 \in \mathbb{Q}$
        if $\alpha_0 \in J$ **then**
            return $I \cap \mathbb{Z}$
        else
            return $\emptyset$
    else
        return $I \cap (\alpha_1^{-1} \cdot (J - \alpha_0)) \cap \mathbb{Z}$

Time: $O(\ell + 1)$, where ℓ = size of output.
Space: $O(\ell + 1)$ as written, but $O(1)$ if output on demand

Algorithm 5 Finding a Diophantine approximation via continued fractions

function DiophAppr(α,Q)
Ensure: Returns (a, a^{-1}, q) s.t. $\left|\alpha - \frac{a}{q}\right| \le \frac{1}{qQ}$, $(a,q) = 1$, $q \le Q$, $aa^{-1} \equiv 1 \bmod q$
  $b \leftarrow \lfloor \alpha \rfloor$, $p \leftarrow b$, $q \leftarrow 1$, $p_- \leftarrow 1$, $q_- \leftarrow 0$, $s \leftarrow 1$
  while $q \le Q$ **do**
    if $\alpha = b$ **then return** $(p, -sq_-, q)$
    $\alpha \leftarrow 1/(\alpha - b)$
    $b \leftarrow \lfloor \alpha \rfloor$, $(p_+, q_+) \leftarrow b \cdot (p,q) + (p_-, q_-)$
    $(p_-, q_-) \leftarrow (p,q)$, $(p,q) \leftarrow (p_+, q_+)$, $s \leftarrow -s$
  return (p_-, sq, q_-)

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

Algorithm 6 Simpler version of main algorithm: sieving $[n-\Delta, n+\Delta] \subset \mathbb{R}^+$

function SimpleNewSegSiev(n,Δ,K,κ)
Ensure: $S_j = 1$ if $n+j$ is prime, $S_j = 0$ otherwise, for $-\Delta \le j \le \Delta$
Require: $\Delta \ge \sqrt[3]{n/\kappa K^2}$, where $\kappa > 0$, $K \ge 2(1+\kappa)$ ▷ $\kappa = 1/2$, $K = 3$ will do
  $S' \leftarrow$ SubSegSiev($n - \lfloor \Delta \rfloor, 2\Delta, K\Delta$) ▷ Sieve by all $p \le K\Delta$
  $S_j \leftarrow S'_{j+\lfloor \Delta \rfloor}$ for all $-\Delta \le j \le \Delta$
  $M \leftarrow \lfloor K\Delta \rfloor + 1$
  while $M \le \sqrt{n+\Delta}$ **do**
    $R \leftarrow \lfloor M\sqrt{\kappa\Delta/n} \rfloor$, $m_0 \leftarrow M + R$
    $\alpha_1 \leftarrow -n/m_0^2$, $\alpha_0 \leftarrow n/m_0$, $\eta \leftarrow (1+\kappa)\Delta/M$
    $(a, a^{-1}, q) \leftarrow$ 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**
        $m \leftarrow m_0 + r$ ▷ We will sieve by m
        $n' \leftarrow \lfloor (n + \lfloor \Delta \rfloor)/m \rfloor \cdot m$ ▷ n' is a multiple of m
        if $(n' \in [n-\Delta, n+\Delta]) \wedge (n > m)$ **then**
          $S_{n'-n} \leftarrow 0$ ▷ thus sieving out n'
    $M \leftarrow M + 2R + 1$
  return S

Preexistent sieves as subroutines. Going back to SubSegSiev: we could avoid using it at all, just by sieving by all integers $m \le K\Delta$, rather than by all primes $p \le K\Delta$. 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. (In general, a "wheel" is just $(\mathbb{Z}/P\mathbb{Z})^*$, where $P = \prod_{p \leq c} p$ 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".)

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 function 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.

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 $\leq n + \Delta$. We should also bound the size of our rational variables.

It is trivial to modify function NewSegSiev so that m_0 is always $\leq \sqrt{n + \Delta}$. Then it is simple to see that the denominators of all our rational variables will be bounded by

$$2Rm_0^2 \leq 2 \left(M \sqrt{\frac{\kappa \Delta}{n}} \right) (n + \Delta) \leq 2\sqrt{\kappa \Delta}(n + \Delta),$$

where $M \leq m_0$. Denominators of this size may arise when we approximate $-n/m_0^2$ by a rational with denominator $\leq 2R$, and then take the difference.

As for the absolute value of our rationals: we set α_1 to $\{-n/m_0^2\}$ rather than $-n/m_0^2$, and similarly for α_0, so that the size of the inputs to SolveModZ will be bounded. We can then arrange matters so that our rational variables are never larger than

$$\max(|\alpha_1|R + |\alpha_0|, 1) = R + 1 \leq \sqrt{\kappa \Delta} + 1.$$

A naïve implementation of SolveInR would divide $(J-\alpha_0)$ by a potentially very small $\alpha_1 = \delta$; in practice, we test first whether $\alpha_1 I_\pm + \alpha_0$ lies in J, and thus avoid this issue.

The situation with function SimpleNewSegSiev (Algorithm 6) is better. (It is understood that the same remarks on implementation apply as above: we ensure that $m_0 \le \sqrt{n}$, and we work with $\alpha_1 = \{-n/m_0^2\}$, $\alpha_0 = \{n/m_0\}$.) We can work entirely with integers of size $\le \min(n+\Delta, (2R)^2) \le n+\Delta$. It would seem at first sight that working with integers of size $\le \Delta q \le 2\Delta R$ would be needed to determine $\lfloor \eta q \rfloor$, and $2\Delta R$ could be more than $n+\Delta$. However, we can compute $\lfloor \lfloor \eta \rfloor q \rfloor$ and $\lfloor (\eta - \lfloor \eta \rfloor) q \rfloor$, and then add them.

Of course, we can also use arbitrary-size integers and rationals for variables outside the innermost loops, and fixed-size integers for variables in the innermost loops. In that way we reduce the running time of the steps that are executed most often.

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 7), 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.

The main idea, much as in SimpleSiev or SegSiev, is that we need to sieve only by primes up to $\sqrt{x}$, where $x = n+\Delta$. To see this, note that, for $n \le x$, the product $\prod_{p|n, p\le\sqrt{x}} p^{v_p(n)}$ equals either n (if n has no prime divisors $p > \sqrt{x}$) or n/p_0, where p_0 is the only prime divisor $> \sqrt{x}$ of n. Thus, taking that product, we can tell whether such a prime divisor p_0 exists, and which prime it is.

Our new sieve for factoring (Algorithm 8), designed for intervals around x of length $\Delta \gg x^{1/3}$, is almost identical to our sieve for primes (Algorithm 1). We use the preexistent sieve (Algorithm 7) for factoring by primes of size up to $K\Delta$, and then we locate primes in the interval $(K\Delta, \sqrt{x})$ that may divide integers in our interval around n. We finish, as in SegSievFac (Algorithm 7) by comparing $\prod_{p|n, p\le\sqrt{x}} p^{v_p(n)}$ to integers n, to see whether a factor p_0 is missing.

4. Time analysis. Parameter choice.

The space and time consumption of Algorithms 2–5 and 7 is clearly as stated: we are using nothing sophisticated than the fact that the expected number of prime divisors of a random number of size about x is $O(\log\log x)$.

Algorithm 7 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 $n' < x$ **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: $O(M + \Delta \log\log(n + \Delta))$.

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($n, \Delta, \lfloor \sqrt{x} \rfloor$)
  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: $O(n^{1/4} + \Delta \log\log(n + \Delta))$.

(A moment's thought shows that this is so even if we are looking at a short interval – meaning an interval of the form $[x, x + x^\delta]$, $\delta > 0$.)

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($n, 2\Delta, K\Delta$), taking 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

Algorithm 8 Main algorithm: factoring integers in $[n-\Delta, n+\Delta]$

function CopList(m,F)
Ensure: Finds whether $p \nmid m$ for every $(p,k) \in F$
Require: F is a list of pairs (p,k)
    for $(p,k) \in F$ **do**
        if $p|m$ **then**
            return false
    return true

Time, Space: $O(\mathrm{len}(F))$

function NewSegSievFac(n,Δ,K,κ)
Ensure: for $-\Delta \le j \le \Delta$, F_j is the list of pairs $(p, v_p(n+j))$ for $p|n+j$
Require: $\Delta \ge \sqrt[3]{n/\kappa K^2}$, where $K \ge 2(1+\kappa)$ ▷ $\kappa = 1/2$, $K = 3$ will do
    $F' \leftarrow$ SubSegSievFac$(n - \lfloor \Delta \rfloor, 2\Delta, K\Delta)$ ▷ Sieve by all $p \le K\Delta$
    $F_j \leftarrow F'_{j+\lfloor \Delta \rfloor}$ for all $-\Delta \le j \le \Delta$
    $M \leftarrow \lfloor K\Delta \rfloor + 1$
    while $M \le \sqrt{n+\Delta}$ **do**
        $R \leftarrow \lfloor M\sqrt{\kappa\Delta/n} \rfloor$, $m_0 \leftarrow M + R$
        $\alpha_1 \leftarrow -n/m_0^2$, $\alpha_0 \leftarrow n/m_0$, $\eta \leftarrow (1+\kappa)\Delta/M$
        $\mathbf{r} \leftarrow$ SolveModZ$(\alpha_1, \alpha_0, R, \eta)$
        for $r \in \mathbf{r}$ **do**
            $m \leftarrow m_0 + r$ ▷ We will sieve by m
            $n' \leftarrow \lfloor (n + \lceil \Delta \rceil)/m \rfloor \cdot m$ ▷ n' is a multiple of m
            if $n' \in [n-\Delta, n+\Delta]$ **then**
                if CopList(m, F) **then** ▷ m is a new factor of n'
                    if $m^2|n'$ **then**
                        append $(m,2)$ to $F_{n'-n}$
                    else
                        append $(m,1)$ to $F_{n'-n}$
        $M \leftarrow M + 2R + 1$
    return S

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.)

The main question is how many "false positives" the function SolveModZ gives, i.e., integers r such that $m = m_0 + r$ does not divide any integer in the interval $[n-\Delta, n+\Delta]$. The parameter η we give to SolveModZ is $(1+\kappa)\Delta/M$; here $\kappa\Delta/M$ is, so to speak, a built-in tolerance that accounts for the error term in (2.1). We can look at matters backwards: every integer r returned by SolveModZ satisfies $n/m_0 - (n/m_0^2)r \in [-\eta, \eta] \bmod \mathbb{Z}$, and

so, by (2.1),

$$\frac{n}{m} \in \left[-\eta - \frac{\kappa\Delta}{M}, \eta + \frac{\kappa\Delta}{M}\right] = \left[-\frac{(1+2\kappa)\Delta}{M}, \frac{(1+2\kappa)\Delta}{M}\right] \bmod \mathbb{Z}.$$

for $m = m_0 + r$. Hence, n is congruent $\bmod m$ to an integer between $-((1+2\kappa)m/M)\Delta$ to $((1+2\kappa)m/M)\Delta$. Recall that

$$m \le M + 2R \le \left(1 + 2\sqrt{\frac{\kappa\Delta}{n}}\right) M.$$

We conclude that there is an integer divisible by m in the interval $[n - \Delta', n + \Delta']$, where

$$\Delta' = (1+2\kappa)\left(1 + 2\sqrt{\frac{\kappa\Delta}{n}}\right)\Delta.$$

In other words, even our "false positives" correspond to actual divisors of integers in an interval around n slightly wider than $[n - \Delta, n + \Delta]$.

The time taken by the inner loop in Algorithm 1 (and part of the time taken by function SolveModZ, in Algorithm 4) is proportional to the total number of values of $m = m_0 + r$ found. We need to bound the total number of divisors m of integers in $[n - \Delta', n + \Delta']$ in the range $K\Delta < m \le \sqrt{n + \Delta}$.

Let us take a shortcut here. This number is obviously bounded by the total number of divisors of integers in $[n - \Delta', n + \Delta']$, i.e., $S(n + \Delta') - S(n - \Delta')$, where $S(x) = \sum_{m \le x} \tau(m)$ and $\tau(m)$ is the number of divisors of m. As was proved by Voronoï [Vor03],

$$S(x) = x \log x + (2\gamma - 1)x + O\left(x^{1/3} \log x\right). \tag{4.1}$$

Hence, by the first few terms of the Taylor series of $x \log x$,

$$S\left(n + \Delta'\right) - S\left(n - \Delta'\right) = 2\Delta'(\log n + 2\gamma) + O\left(n^{1/3} \log n\right),$$

since we can assume that $(\Delta')^2/n \ll n^{1/3} \log n$. (We can certainly assume that $\Delta \le \sqrt{n}/2$, as otherwise the inner loop in NewSegSiev is never entered. The bound $(\Delta')^2/n \ll 1$ follows immediately.)

We should also take into account the number of iterations in the outer loop in NewSegSiev. At each step, M is replaced by $M + 2R + 1 = M + 2\lfloor M\sqrt{\kappa\Delta/n}\rfloor + 1$; in other words, M is multiplied approximately by $1 + \sqrt{\kappa\Delta/n}$. (Note that M is always $\ge K\Delta \ge 2(1+\kappa)\Delta$, and so

$$M\sqrt{\kappa\Delta/n} \ge 2\sqrt{(\kappa\Delta)^3/n} \ge 1$$

provided that $\Delta \ge n^{1/3}/2^{2/3}\kappa$. Thus the approximation here is always acceptable.) It follows that the loop is iterated approximately

$$\frac{\log \frac{\sqrt{n}}{K\Delta}}{\log\left(1 + \sqrt{\frac{\kappa\Delta}{n}}\right)} \sim \sqrt{\frac{n}{\kappa\Delta}} \cdot \log \frac{\sqrt{n}}{K\Delta} \tag{4.2}$$

times. This amount – multiplied by a factor of $O(\log n)$, coming from SolveModZ – needs to be added to the total time taken, as each iteration of the outer loop takes time, even when the inner loop is not entered.

Our total time expenditure is thus

$$\begin{aligned} &O\left(K\Delta \log\log K\Delta\right) + O(2(1+2\kappa)\Delta(\log n + 2\gamma)) \\ &+ O\left(\sqrt{\frac{n}{\kappa\Delta}} \log \frac{\sqrt{n}}{K\Delta} \log n\right) + O\left(n^{1/3} \log n\right). \end{aligned} \tag{4.3}$$

In order to keep the total time bounded by $O(\Delta \log n)$, we require that $\Delta \geq n^{1/3} \log^{2/3} n$. Then $\sqrt{n/\kappa\Delta} \log n \leq \Delta/\sqrt{\kappa}$. Taking derivatives of the expressions in (4.3) with respect to K, we see that, at least in principle, we ought to set K as low as possible, i.e., $K = 2(1+\kappa)$. (In practice, it can be better to set K to a somewhat larger constant.) As for κ, it makes sense to set it very low when Δ is large, but, for the sake of simplicity, we may set it to a constant, such as (say) $\kappa = 1/4$. We then obtain that the total time taken by the function NewSegSiev is

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

It remains to estimate the time and space taken by NewSegSievFac, which factors integers in the interval $[n-\Delta, n+\Delta]$. Everything is much the same, except in two respects. The total space taken will be

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

rather than $O(\Delta)$, simply because storing the list of prime factors of an integer in $[n-\Delta, n+\Delta]$ takes space $O(\log\log n)$ on average. (Showing that this is the case is actually much easier than showing the number of divisors of integers in the same interval is $O(\log\log n)$ on average: an integer $\leq n+\Delta$ can have at most two prime divisors of size $> \sqrt[3]{n+\Delta}$.)

The other matter to take into account is that, when we find integers $K\Delta < m \leq \sqrt{n+\Delta}$ dividing some integer n' in $[n-\Delta, n+\Delta]$, we have to test whether they are divisible by any the prime divisors of n' that have already been found. Let $n' = p_1^{\alpha_1} p_2^{\alpha_2} \dots p_k^{\alpha_k}$, where the prime factors p_i are written in the order in which they are found. Every divisor of n' gets tested for divisibility by p_1 (except for p_1 itself, which gets stored, a process that itself takes $O(1)$ time). Hence, testing the divisors of n' for divisibility by p_1 takes time proportional to $\tau(n') = (\alpha_1+1)\cdots(\alpha_k+1)$. Every divisor of n' not divisible by p_1 gets tested for divisibility by p_2 (except for p_2 itself). Hence, testing for divisibility by p_2 takes time $O((\alpha_2+1)\cdots(\alpha_k+1))$. Proceeding this way, we see that the total time spent in testing the divisibility of divisors

of n' is

$$O\left(\sum_{j=1}^{k}(\alpha_j+1)(\alpha_{j+1}+1)\cdots(\alpha_k+1)\right)$$
$$=O\left(\tau(n')\cdot\sum_{j=1}^{k}\frac{1}{(\alpha_1+1)\dots(\alpha_{j-1}+1)}\right)=O\left(\tau(n')\sum_{j=1}^{k}\frac{1}{2^{j-1}}\right)=O(\tau(n')).$$

As a consequence, we just have an additional term of the form $O(S(n+\Delta)-S(n-\Delta))$ in the time estimate. We already had such a term, or in fact a slightly larger one, namely, $O(S(n+\Delta')-S(n-\Delta'))$. Thus, the total time estimate is still

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

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

Remark. We might ask ourselves why we took a shortcut and used (4.1). Why not bound the number of divisors m of integers in the range $K\Delta<m\le\sqrt{n+\Delta}$, using a method based on the algorithm given here? In other words, why not reprove Voronoï's result?

What happens is that the algorithm here does not hew too closely to either Voronoï or later work, in that it has been shaped by the demand to compute rather than estimate. We can, however, simplify our algorithm (see Algorithm 6) and then do a time analysis that follows Vinogradov's take on the matter ([Vin54, Ch. III, exer. 3–6]), in effect reproving his version of the result in the process. We are about to do as much. It will be easy to see that the time taken by the algorithm analyzed above is bounded by a constant times the time taken by the simplified algorithm. (Indeed, one would expect the simplified algorithm to be slower, since it allows for more "false positives".)

In [Vin54], the bound Vinogradov gives for the error term of his variant of Voronoï's work is $O(n^{1/3}(\log n)^2)$, that is, it is worse than Voronoï's bound by a factor of $\log n$. As it happens, for whatever reason, Vinogradov did not choose his parameters optimally; if the value of τ in the solution to [Vin54, Ch. III, exer. 3–6]) is set to be $(A\log A)^{1/3}$ rather than A, Vinogradov's proof gives a bound of $O(n^{1/3}(\log n)^{5/3})$ on the error term, rather than $O(n^{1/3}(\log n)^2)$. If we use that bound in time analysis above, we get qualitatively the same result that we got, that is, we obtain a bound $O(\Delta\log n)$ on the running time whenever $\Delta\gg n^{1/3}(\log n)^{2/3}$.

What we are about to do is to apply Vinogradov's ideas, rather than his result (optimized or not), to the time analysis, for the sake of giving a self-contained treatment. We will obtain a result that is just as good as the one we have just proved: we will bound the running time by $O(\Delta\log n)$ whenever $\Delta\gg n^{1/3}(\log n)^{2/3}$.

Time consumption of simplified variant. Algorithm 6 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, though 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 $O(\log Q) = O(\log n)$, or, in total,

$$\ll \sqrt{\frac{n}{\kappa\Delta}} \cdot \log \frac{\sqrt{n}}{K\Delta} \log n \leq \sqrt{\frac{n}{\Delta}} (\log n)^2,$$

as in (4.2). As before, we set κ to a positive constant – say, $1/4$.

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 $-k-1 \leq j \leq k+1$ (where $k = \lfloor \eta q \rfloor$) is at most

$$\begin{aligned} \left\lceil \frac{2R+1}{q} \right\rceil \cdot (\lfloor \eta q \rfloor + 3) &\leq \left(\frac{2R}{q} + 2 \right) (\eta q + 3) \ll (R+q)\eta + \frac{R}{q} + 1 \\ &\ll \frac{\Delta^{3/2}}{\sqrt{n}} + \frac{R}{q} + 1, \end{aligned} \tag{4.4}$$

since $R \leq \sqrt{\kappa\Delta/n} \cdot M$, $\eta = (1+\kappa)\Delta/M \leq (1+\kappa)/K \leq 1/2$ (because $K \geq 2(1+\kappa)$) and $q \leq R$. We set K to a constant, as before. Let us say $K \geq 3$.

As we were saying, the number of times the main loop is executed – that is, the number of intervals $[M, M+2R]$ we consider – is still given by (4.2). Hence, the total contribution of the first and last terms in the last line of (4.4) is

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

provided that $\Delta \geq n^{1/3}$. It remains to account for the contribution of R/q.

Now we proceed much as in [Vin54, Ch. III, exer. 3–6]. We will examine how $\alpha_1 = -n/m_0^2 \bmod 1$ 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 $m_0 < m_0'$. Then

$$\left(-\frac{n}{(m_0')^2} \right) - \left(-\frac{n}{m_0^2} \right) = n \frac{(m_0')^2 - m_0^2}{m_0^2 (m_0')^2}. \tag{4.5}$$

Now, 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.

We can assume without loss of generality that $\Delta \le n/100$, and so $M' := m_0' - R \le \sqrt{n+\Delta} \le (1+1/200)\sqrt{n}$, $R' \le M'\sqrt{\Delta/4n} \le M'/20$ and $m_0' = M' + R' < 1.1\sqrt{n}$; in the same way, $m_0 < 1.1\sqrt{n}$ and $m_0 \le (M+R) \le (1+1/20)M$. Clearly, $m_0' \le 2m_0$, as otherwise $n/m_0^2 - n/(m_0')^2 \ge 3n/4m_0^2 \ge 3n/(m_0')^2 \ge 3/1.1^2 > 2$, giving us a contradiction to $2/qR \le 2/R \le 2$. Hence

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

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

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

We recall that $R = \lfloor M\sqrt{\kappa\Delta/n} \rfloor \ge \lfloor K\Delta\sqrt{\kappa\Delta/n} \rfloor \ge \lfloor K\sqrt{\kappa} \rfloor \ge \lfloor 3\sqrt{4} \rfloor = 6$, and so $R(2R+1) > (3/2)(R+1)^2$. Hence, the number of intervals for which $\alpha_1 = -n/m_0^2$ has a given approximation a/q is at most

$$\begin{aligned}&\frac{8}{3}\frac{m_0^3}{qR(2R+1)n} + 1 \le \frac{16}{9}\frac{m_0^3}{q(R+1)^2 n} + 1 \le \frac{16}{9}\frac{m_0^3}{qM^2\kappa\Delta} + 1\\ &\le (1+1/20)^2 \frac{16}{9}\frac{4m_0}{q\Delta} + 1 \ll \frac{m_0}{q\Delta} + 1.\end{aligned}$$

We should also see when $n/m_0^2 - n/(m_0')^2 \ge 1$, or rather when $n/m_0^2 - n/(m_0')^2 \ge 1 - 2/R$. By (4.5), the latter inequality is fulfilled when

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

and, since $R \ge 6$, that implies

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

Hence, for given m_0, it makes sense to consider all following intervals with $m_0' \le m_0 + m_0^3/3n$. 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 $m_0^3/4Rn + 1 \ll m_0^2/\sqrt{\Delta n} + 1 \ll m_0^2/\sqrt{\Delta n}$ intervals with $m_0' \le m_0 + m_0^3/3n$. (Note that $m_0^2/\sqrt{\Delta n} \ge K^2\Delta^2/\sqrt{\Delta n} > 1$.) Among those intervals, $\ll m_0'/q\Delta + 1 \ll m_0/q + 1$ 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 $\lceil O(m_0/q\Delta + 1)\phi(q) \rceil = O(m_0/\Delta + q)$ values of a/q with $q \le Q$, and none for $q > Q$, where $Q \ll m_0/(\Delta n)^{1/4}$.

The contribution of R/q for all intervals with $m_0' \le m_0 + m_0^3/(2n)$ is thus

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

Hence, the contribution for all intervals with m_0 in an interval $[M_0, 2M_0]$ is

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

and 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 the simplified algorithm is

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

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

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

In other words, we obtain the same bound, qualitatively speaking, as before, under the same conditions. We have thus given a second proof of the main theorem, at least in regard to finding primes. We can also adapt the simplified algorithm to factorize numbers just as we adapted the original algorithm to do the same. The time analysis is as above, and thus we can obtain a second proof of the main theorem in its entirety.

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 $\deg P \ge 2$ and $\eta \ll 1/R$ or thereabouts.

Solving a quadratic modular equation $a_2 x^2 + a_1 x + a_0 \equiv 0 \bmod q$ 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 $q \le x^{1/4}$ (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 $P(x) = \alpha_2 x^2 + \alpha_1 x + \alpha_0$, 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 $|\alpha_2 - a_2/q| \le 1/qR^2$, $|\alpha_1 - a_1/q| \le 1/qR$. 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 small. This is the same as 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 §2.2 then amounts to finding points close to a line, starting by approximating 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 matters to a problem $\bmod q$, and we do not yet know how to carry out such a reduction well.

Appendix A. A few words on the implementation

I have written and tested a simple proof-of-purpose implementation, mainly to check that the algorithms work as described. On the higher-end of what can fit in ordinary 64-bit computer arithmetic (say: $n = 5 \cdot 10^{18}$, $\Delta = 2 \cdot 10^7$ or $\Delta = 4 \cdot 10^7$), my implementation of algorithm SimpleNewSegSieve runs substantially faster than my own implementations of either NewSegSiev or SegSiev. Still, on those same inputs, my implementation of SimpleNewSegSieve is clearly slower (by a factor between 2 and 2.5) than a publicly available, highly optimized implementation [Wal] of the traditional

algorithm (essentially SegSiev, but improved in all the ways detailed below, and some other ones) on the same interval. Diophantine approximation (Algorithm 5) turns out to take less than 2 percent of the running time of SimpleNewSegSiev, 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 algorithms we have seen in this paper can be easily implemented with arbitrary-precision arithmetic; indeed; the author's implementation uses arbitrary-precision arithmetic (GNU MP) for Algorithm 4 (used by NewSegSieve, not by SimpleNewSegSiege). Naturally, the advantage of SimpleNewSegSieve and NewSegSieve over existing methods should become clearer as n grows larger – meaning much 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, if we are sieving for primes or computing values of μ, 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.
(2) We can first apply a simple sieve to the integers between 1 and $M = \prod_{p \leq p_0} p$ (where $p_0 = 17$, say), taking only the effect of primes $p \leq p_0$ 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 $\{p \leq p_0 : p|n\}$, 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 $p \leq p_0$ ever again.
(3) If we are sieving for primality, it makes sense to use a *wheel* – meaning that, instead of taking out all multiples or all odd multiples of a number m, we take out only multiples pm with p prime, or multiples dm such that $d \mod M'$ lies in $(\mathbb{Z}/M'\mathbb{Z})^*$ for $M' = \prod_{p \leq p_1} p$, p small. See, e.g., [Pri83].
(4) If we are sieving so as to compute μ, then, instead of keeping track of $\prod_j = \prod_{p|(n+j)} p^{v_p(n+j)}$ as in Algorithm 7, we can keep track of $\Sigma_j = \sum_{p|(n+j)} \lceil \log_4 p \rceil$. (The function $\lceil \log_{2^k} p \rceil$ can be computed very rapidly in practice, thanks to the fact that computers work in binary.) We thus save space, and, more importantly, replace some multiplications and divisions by additions. Divisions are particularly time-consuming in practice, and thus should be avoided when at all possible.

Let us show how this trick works. If the value of Σ_j at the end (namely, $\sum_{p\le\sqrt{n+\Delta}:p|(n+j)}\lceil\log_4 p\rceil$) is less than $\lceil\log_4(n+j)\rceil$, then, evidently, $n+j$ has a prime factor that was unaccounted for, and so μ_j must be flipped in order for it to hold the correct value of $\mu(j)$. Conversely, if $\sum_{p\le\sqrt{n+\Delta}:p|n+j}\lceil\log_4 p\rceil \ge \lceil\log_4(n+j)\rceil$, then

$$\sum_{\substack{p\le\sqrt{n+\Delta}\\ p|im+j}} \log_4 p \ge \frac{1}{2}\sum_{p\le\sqrt{n+\Delta}} \lceil\log_4 p\rceil \ge \frac{1}{2}\log_4(n+\Delta)\rceil,$$

and so $n+j$ cannot have a prime factor larger than $\sqrt{n+\Delta}$; meaning that μ_j must not be flipped. The same idea can be found in Hurst [Hur18, §2]. The idea of flipping μ_j at the end is much older.

(5) 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].

(6) As we mentioned 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 can't 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. If we use SimpleNewSegSieve, 128-bit-integer arithmetic should be enough in that range.

References

[AB04] A. O. L. Atkin and D. J. Bernstein. Prime sieves using binary quadratic forms. *Math. Comp.*, 73(246):1023–1030 (electronic), 2004.

[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. In *Algorithmic number theory (Leiden, 2000)*, volume 1838 of *Lecture Notes in Comput. Sci.*, pages 297–312. Springer, Berlin, 2000.

[Hur18] G. Hurst. Computations of the Mertens function and improved bounds on the Mertens conjecture. *Math. Comp.*, 87:1013–1028, 2018.

[HW79] G. H. Hardy and E. M. Wright. An introduction to the theory of numbers. 5th ed. Oxford etc.: Oxford at the Clarendon Press. XVI, 426 p. hbk., 1979.

[Khi97] A. Ya. Khinchin. *Continued fractions*. Dover Publications, Inc., Mineola, NY, 1997. With a preface by B. V. Gnedenko, Reprint of the 1964 translation.

[Lan12] E. Landau. Die Bedeutung der *Pfeiffer*schen Methode für die analytische Zahlentheorie. *Wien. Ber.*, 121:2196–2332, 1912.

[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.*, 83:2033–2060, 2014.

[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*, pages 1–21, 1886.

[Pri83] P. Pritchard. Fast compact prime number sieves (among others). *Journal of algorithms*, 4(4):332–344, 1983.

[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(17):77–118, 1906.

[Sin69] R. C. Singleton. Algorithm 357: an efficient prime number generator. *Communications of the ACM*, 12:563–564, 1969. URL: `http://cr.yp.to/bib/entries.html#1969/singleton-357`.

[Sor98] J. P. Sorenson. Trading time for space in prime number sieves. In *Algorithmic number theory (Portland, OR, 1998)*, volume 1423 of *Lecture Notes in Comput. Sci.*, pages 179–195. Springer, Berlin, 1998.

[SP94] J. Sorenson and I. Parberry. Two fast parallel prime number sieves. *Inform. and Comput.*, 114(1):115–130, 1994.

[TCH12] T. Tao, E. Croot, III, and H. Helfgott. Deterministic methods to find primes. *Math. Comp.*, 81(278):1233–1246, 2012.

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

[Vor03] G. Voronoï. Sur un problème du calcul des fonctions asymptotiques. *J. Reine Angew. Math.*, 126:241–282, 1903.

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

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

E-mail address: `harald.helfgott@@gmail.com`