

Enumerating primes in nearly linear time, intuitively

24 October 2026

TL;DR

Harvey recently gave a deterministic algorithm that lists the primes below N in $N(\log \log N)^{1+o(1)}$ bit operations, but it may report failure instead of producing the list (it never outputs a wrong list, though). A conjecture about square-primes (m^2p with p prime) in short intervals guarantees that this failure never happens. In this note, we explain a new unconditional algorithm that always produces the list, with the same bound.

Harvey's core algorithm (which is distinct from the algorithm above, but is a part of it) essentially finds the square-primes (squares times a prime) block by block. It is correct on blocks that aren't too crowded, but not on the crowded ones - and it does not indicate which ones were crowded. Once these square-primes are found, they can be used to enumerate primes in linear time. Unfortunately, one version of the full algorithm depends on a conjecture on how many square-primes can be present in certain kinds of short intervals (which lets it avoid dealing with crowded blocks), and the other one requires an extra runtime penalty to detect and handle crowded blocks.

The new idea in our algorithm is to be able to tell (conservatively) which blocks aren't too crowded, by computing upper bounds on a proxy for each block's count of square-primes. These bounds are allowed to have some slack, as long as the slack is small when summed over blocks. Note that this is different from Harvey's proxy (counts of rough numbers), which can be computed exactly but is costly. Then, as usual, the few blocks that the algorithm can't say much about (i.e., whether they are crowded or not) are handled by directly checking for square-prime-ness.

We explain, intuitively and at a high level, how and why this algorithm works, without having to deal with the technicalities in the paper. We will be intentionally loose as far as details such as rounding/constants are concerned, so that we can focus only on the main points of the algorithm.

Contents

1	The basic setup	2
2	Harvey's algorithm	2
2.1	Square-primes instead of primes	3
2.2	Blocks and the core algorithm	3
2.3	The repair tool	3
2.4	The plan	3
3	Some wishful thinking and how it fails	4
3.1	The new idea	4
3.2	The pigeonhole split	5

4 Building the proxy: a sieve majorant	5
4.1 Inclusion-exclusion	6
4.2 What this weight does	6
4.3 From primes to square-primes	7
5 Job A: large proxy sums are rare	7
5.1 The way out: second moments	8
6 Job B: computing $U_r \geq W_r$ in linear time	10
6.1 Counting products in groups	11
6.2 From pair counts to block bounds	11
6.3 The total overshoot is small	12
6.4 Technicalities: approximate bins	13
6.5 Choosing η and T	13
7 Parameter bookkeeping	14
References	14

1 The basic setup

The problem is the following: given N , list all primes below N in increasing order, on a multitape Turing machine. We count bit operations, and reading/writing a number of size around N costs around $\log N$ of them. Note that the multitape Turing machine model is different from the usual RAM model, since there is no random access - any movement has to be done via sequential scans. So, for instance, the sieve of Eratosthenes would naively need about $\sqrt{N}/\log N$ full scans, one for each prime up to $\sqrt{N}$ (hence accessing about $N^{3/2}/\log N$ positions), which is nowhere near the cost we want.

What's the best possible cost? By the prime number theorem, there are roughly $N/\log N$ primes below N , so just writing the answer down takes time around N (each prime has about $\log N$ bits on average). So our goal of $N(\log \log N)^{1+o(1)}$ is very close to optimal.

In what follows, we use $L \approx \log_2 N$. Since this is an intuitive explanation, we will not be very careful about exact values (for those, the paper can be referred to), and we will mostly talk about what the approach looks like, the various issues we face, and how to fix them. Most parameters we see will be fixed powers of L (or very small powers of N), and the bulk of the proof is just verifying that things fit together.

2 Harvey's algorithm

We won't go into the internals of Harvey's algorithm. Instead, we will treat three of its important components as black boxes, and only use their input/output guarantees.

2.1 Square-primes instead of primes

A *square-prime* is a number of the form m^2p with p prime, such as $45 = 3^2 \cdot 5$ and $27 = 3^2 \cdot 3$ (primes correspond to $m = 1$). Why do we care about them? Harvey’s generating-function identity (after some handling) naturally produces the odd square-primes rather than the primes, and going from the list of odd square-primes to the list of primes costs only $O(N)$ (this is another part of Harvey’s algorithm). So from now on, our target is the odd square-primes. Note that these are about as dense as the primes - roughly c/L of all integers, for some constant c (for each fixed odd m , there are around $\pi(N/m^2) \approx N/(m^2 \log N)$ choices of p , and summing over m gives the same order of density as the density of primes).

2.2 Blocks and the core algorithm

Let’s partition $[0, N)$ into *blocks* of length T , and let w_r be the number of odd square-primes in block r . On average, $w_r \approx cT/L$.

Harvey’s *core algorithm* takes a threshold R and outputs a candidate set for every block, with the following guarantee - if $w_r \leq R$, the candidate is correct. If $w_r > R$, the candidate is wrong (it has at most R elements), and no error is reported. Most of what our algorithm does is detecting and handling these unreported errors.

One way to think about this silent failure is the following. Suppose I pick a set S of nonzero elements in $\mathbb{C}$, and only tell you the power sums $\sum_{x \in S} x^k$ for $k = 1, \dots, R$. If $|S| \leq R$, Newton’s identities let you rebuild a polynomial whose nonzero roots give you S . If $|S| > R$, the same procedure still outputs some polynomial of degree R (without complaining), and its nonzero roots form a plausible-looking but wrong answer. Harvey’s compression has a very similar flavor - each block is stored as a short “sketch”, and decoding works only when the block is sparse.

If T is at most a fixed power of L , the core algorithm costs about

$$\left(1 + \frac{RL}{T}\right) N(\log \log N)^{1+o(1)}.$$

To keep the bracket bounded, we want R to be $O(T/L)$. On the other hand, R can’t really be smaller than a constant times T/L , since the average block already has about cT/L odd square-primes. So R is a large constant times T/L , which means that every block that is more crowded than a constant times the average would make the algorithm fail without us knowing. This is the main issue we need to fix.

2.3 The repair tool

Harvey also has a *direct test*: given any list of integers, it finds the square-primes among them in

$$O(N) + (\text{list size}) \cdot L^{11.5+o(1)}$$

bit operations. This is expensive per number, but it is affordable as long as the list has at most N/L^{12} numbers, since $N/L^{12} \cdot L^{11.5} = o(N)$. So we want to somehow ensure that in the end, at most N/L^{12} numbers need this direct test.

2.4 The plan

Given these three components, the plan is fairly natural:

1. Run the core algorithm.

2. Keep its answers on the blocks we can *certify* have $w_r \leq R$.
3. *Mark* every other block, and repair it with the direct test.

For this, we need two things. The first is **safety**: every unmarked block really has $w_r \leq R$. The second is **cheapness**: the marked blocks have total length $O(N/L^{12})$, and deciding what to mark costs $O(N)$.

3 Some wishful thinking and how it fails

The first thing one might try is to just count w_r for each block. Unfortunately, this is likely circular, since we don't know how to count odd square-primes in every block without essentially solving the problem.

The second thing one might try is to prove that few blocks are overcrowded. This is the right kind of thing to prove, but even if it is true, it doesn't tell the algorithm *which* blocks are bad. So we need a way to find the bad blocks (or at least narrow them down to a small set).

Harvey's algorithm has a few versions. The conditional version relies on a conjecture that implies that these bad blocks don't exist in the first place (for a certain threshold that decides which blocks are bad). The (deterministic) unconditional version uses a different proxy for estimating w_r , namely odd *square-rough numbers* (a square times a number with no prime factor below some Y) instead of odd square-primes - it marks/rejects a block if it has more than R odd square-rough numbers or contains an odd square-prime that is not square-rough. This runs into a trade-off.

Since checking a number directly costs a power of L , rejected blocks must be rare. Harvey's sieve estimate bounds the fraction of blocks with more than R square-rough numbers by something roughly of the form $(T/\log Y)^R$ (ignoring constants and smaller factors). To make this bound smaller than a fixed negative power of L , we need $R \log(\log Y/T) \gtrsim \log L$, and since $\log x < x$, this forces $R/T \gtrsim \log L/\log Y$. So the core algorithm's factor $1 + RL/T$ is at least around $L \log L/\log Y$. On the other hand, making the rough-number array itself costs around $N \log Y$. Since the core also costs a factor of $(\log L)^{1+o(1)}$, the two costs to balance are roughly

$$N \log Y + \frac{NL(\log L)^{2+o(1)}}{\log Y}.$$

This gives $\log Y \approx \sqrt{L}(\log L)^{1+o(1)}$, which adds a factor of $\sqrt{\log N}$ (up to $\log \log N$ factors) to the complexity.

3.1 The new idea

Still, we would like something that behaves like rough numbers do in Harvey's setup, but without the trade-off that leads to the slowdown.

Concretely, we want a proxy $W_r \geq w_r$ (except on some blocks that we mark separately - see the next section) such that

- (a) Its density is only a constant times the prime density, so about $1/L$ (the same density desideratum as earlier).
- (b) We can *prove* that it is rarely large on blocks (to keep the number of marked blocks small).
- (c) We can *compute useful upper bounds* U_r for it in linear time (to avoid a cost like the square-rough one).

We then mark block r whenever $U_r > R$ (and also the few blocks containing an odd square-prime with a tiny prime part - see the next section).

On any unmarked block, we have

$$w_r \leq W_r \leq U_r \leq R$$

so the core algorithm's answer there is guaranteed to be correct. So once we have this proxy, safety is guaranteed, and only cheapness needs to be worked on.

3.2 The pigeonhole split

For cheapness, we need to come up with bounds U_r on W_r such that not too many blocks get marked due to our choice of U_r , and such that U_r can be computed efficiently. We focus on the former for now, and look at the latter later.

The following is how we will bound the number of marked blocks in the proof, and it will also help us come up with some desiderata for U_r and W_r .

Take $R = 2C_1T/L$, in line with our requirement that R be a large constant multiple of T/L (note that C_1 has a stronger requirement to fulfill - it should be large relative to the typical/average W_r , as we need in Job A, instead of just w_r - but it still ends up being a constant). The factor of 2 is there so that the following argument works.

We split $U_r = W_r + (U_r - W_r)$. So if $U_r > R$, then either $W_r > C_1T/L$ (i.e., the true proxy is large), or $U_r - W_r > C_1T/L$ (i.e., our bound on it overshoot by a lot). This splits the remaining work into two completely separate jobs:

- **Job A (number theory):** Show that few blocks have large W_r (the total length of these should be $O(N/L^{12})$).
- **Job B (algorithms):** Pick U_r fast (e.g., in $O(N)$ time), so that the total overshoot $\sum_r (U_r - W_r)$ is small (more precisely, $O(N/L^{13})$).

Note that Job B only needs a bound on the *total*. Since each overshoot is nonnegative, a Markov-type count shows that only a few blocks can have a big one:

$$\#\{r : U_r - W_r > C_1T/L\} \leq \frac{L}{C_1T} \sum_r (U_r - W_r)$$

This gives that the total length marked due to overshooting is $O(N/L^{12})$, as desired.

In particular, the bound on an individual block is allowed to be terrible, and that's fine because this inequality lets us get away with it.

4 Building the proxy: a sieve majorant

We want a weight W_r for the blocks. The easiest way to do so is to construct a weight $W(n)$ for each integer n and then sum it over the block. This would give us $W_r \geq w_r$ if we had pointwise inequalities such as $W(n) \geq 1$ for n being a square-prime that we care about (and $W(n)$ nonnegative in general).

The natural candidate for this W comes from sieve theory. Without going too much into the details for now, we want a W that is at least 1 on odd square-primes (with prime part $> y$ - the rest are handled separately), nonnegative everywhere, and not much bigger than the indicator of the square-primes on average.

Square-primes are not very natural to work with, so let's first try an easier problem - we try to construct a θ that is at least 1 on numbers that don't have prime factors below y , nonnegative everywhere, and not much bigger than the indicator of the primes on average. We will then use this θ to construct an appropriate W satisfying the conditions in the previous paragraph.

4.1 Inclusion-exclusion

Let's start with the exact version of the indicator function that θ corresponds to. The statement “ n has no prime factor below y ” has the indicator

$$\prod_{p < y} (1 - [p \mid n]) = \sum_{e \mid P} \mu(e) [e \mid n]$$

where P is the product of the primes below y , and $\mu(e) = (-1)^{\#\text{prime factors of } e}$ for squarefree e (this can also be seen as the identity $\sum_{d \mid m} \mu(d) = [m = 1]$ with $m = \gcd(n, P)$). In other words, we subtract the multiples of each prime, add back the multiples of each pair, and so on. The divisors e appearing here can be astronomically large, and we would like to keep only $e \leq Z$, with $Z = N^{1/1000}$. There are many reasons for keeping Z small - it affects our error bounds in various computations (more terms mean more accumulated error), and if we compute things down the line, it may lead to more terms, making the computation costly. Z also needs to be a small power of N for some technical reasons that we will explain later.

Roadblock: naive truncation breaks positivity

If we sieve by 2 and 3 and drop the term with $e = 6$, we get $1 - [2 \mid n] - [3 \mid n]$, which is -1 at $n = 6$. So the “upper bound” went negative.

Fix: the beta-sieve

The idea is to delete terms in groups whose total sign is safe (i.e., deleting them does not reduce the upper bound). Write the primes of e in decreasing order, $p_1 > p_2 > \dots$, apply a size test only at the *odd* positions j (roughly, whether $p_1 \cdots p_j p_j^\beta < Z$), and drop e if any test fails. If all tests pass, set $\lambda_e = \mu(e)$, and otherwise set $\lambda_e = 0$. Note that with a bit of work, one can show that the test implies $e \leq Z$ for nonzero λ_e . We then define $\theta(n) = \sum_{e \mid n} \lambda_e$.

To see why this works, fix n , and group the dropped terms (i.e., the dropped divisors e of n) by their first failed prefix $p_1 > \dots > p_j$, with j odd. Such a group consists of that prefix together with *every* subset of the smaller prime divisors of n , so its total sign is

$$(-1)^j \sum_C (-1)^{|C|}$$

where the sum runs over those subsets C . This sum is 1 if there are no smaller prime divisors and 0 otherwise, and since j is odd, each dropped group has total sign -1 or 0. Deleting non-positive chunks can only *increase* the sum, so

$$\theta(n) = \sum_{e \mid n} \lambda_e \geq [n \text{ has no prime factor } < y] \geq 0$$

4.2 What this weight does

For primes $p > y$, we have $\theta(p) = 1$ (the only divisor that participates nontrivially is 1). The weight also puts mass on some non-primes, such as pq with both factors large, but that's not an issue since we only need an upper bound. The one result we take from sieve theory without proof is the *fundamental lemma*: the average of θ is at most twice $\prod_{p < y} (1 - 1/p) \approx 1/\log y \approx 1/L$ (the last approximate equality holds for our choice $y = \lfloor Z^{1/s} \rfloor$, where s is a large constant, as needed for applying the fundamental lemma). So truncating costs only a constant factor in density.

4.3 From primes to square-primes

To handle odd square-primes, we iterate over odd squares:

$$W(n) = \sum_{m \text{ odd}, m^2 | n} \theta(n/m^2)$$

This works because if $n = m^2 p$ is odd with $p > y$, the term for that m is 1, and all the other terms are nonnegative, so $W(n) \geq 1$.

Square-primes with a tiny $p \leq y$ are an exception, but there are only $O(\sqrt{Ny})$ of them, so we can just list them and mark their blocks. Even if we account a whole block for each one, this marks only $O(T\sqrt{Ny}) = o(N/L^{12})$ positions, since T is polylogarithmic and $y \leq N^{1/1000}$.

Expanding everything gives

$$W(n) = \sum_{(d,\sigma) \in \mathcal{S}} \sigma[d | n]$$

where each divisor is $d = m^2 e$ and each sign is $\sigma = \pm 1$ (note that e is uniquely determined by d), and $\mathcal{S}$ is the set of pairs $(m^2 e, \lambda_e)$ with m odd, $e \leq Z$, $\lambda_e \neq 0$ and $m^2 e \leq N$. It is this *signed divisor-sum* structure that both jobs below use.

To check that the density of $W(n)$ is not too large (property (a) from Section 3), note that for each odd m , the integers n with $m^2 | n$ make up a $1/m^2$ fraction of all integers, and on those n , the term $\theta(n/m^2)$ averages like θ itself. More precisely, let ν be the average of θ . Counting the multiples of each divisor e gives $\sum_{q \leq x} \theta(q) = \nu x + O(Z)$, since each of the at most Z divisors contributes a rounding error of at most 1. Hence

$$\sum_{1 \leq n < N} W(n) = \sum_{\substack{m \text{ odd} \\ m^2 < N}} \sum_{1 \leq q < N/m^2} \theta(q) = \nu N \sum_{\substack{m \text{ odd} \\ m^2 < N}} \frac{1}{m^2} + O(Z\sqrt{N})$$

Since $Z\sqrt{N}$ is much smaller than N/L , the average of W is about $\frac{\pi^2}{8}\nu = O(1/L)$. As an aside, $\pi^2/8$ is the same factor by which odd square-primes outnumber primes, so passing from θ to W multiplies the average by the same factor as passing from primes to odd square-primes. So this construction of W has property (a).

5 Job A: large proxy sums are rare

Recall that W_r is the sum of the weights $W(n)$ over block r , and by property (a), it is on average a constant times T/L . Job A asks us to show that the blocks with $W_r > C_1 T/L$, i.e., blocks whose proxy sum is many times larger than average, have total length $O(N/L^{12})$. If we try to do this naively, two things go wrong.

Roadblock: short intervals are hard

Counting the multiples of e in an interval of length T gives T/e plus an error of up to 1. With Z divisors (already for θ - the situation is worse for W), the error per interval can be as large as $Z = N^{1/1000}$, which is vastly bigger than the main term T/L , since T is only polylogarithmic. So no single short interval can be certified by this kind of argument.

Roadblock: averaging once isn't enough

The total weight is about N/L , so Markov's inequality allows about N/T blocks above $C_1 T/L$, which is a total length of about N . But we need the total length to be about N/L^{12} .

5.1 The way out: second moments

It turns out that both roadblocks have the same way out. For the first one, note that Job A never needs to certify a single interval - it only has to show that bad intervals are few, and individual blocks are certified later by computation (Job B). So instead of bounding the error on each interval, we bound it on average over all positions of the interval, where the errors coming from different divisors largely cancel. For the second one, we average the *squared* deviation rather than the deviation itself, which makes large deviations much rarer (Chebyshev instead of Markov).

The argument has four steps:

1. a general lemma saying that sums over a sliding window rarely stray far from their average;
2. the lemma cannot be applied to all of W at once, so we split W into two parts, which turns Job A into two smaller goals;
3. for one part (the tail), Markov's inequality is enough after all;
4. for the other part, we use the lemma and Chebyshev's inequality.

So, roughly speaking, we will show that the total squared deviation of window sums from their average is small, so only a small total length of window positions can have a large deviation (Chebyshev). On the other hand, every bad block forces a large deviation at many positions, so there can only be a few bad blocks.

Step 1: sliding windows and the variance lemma. Take any function of the form $f(n) = \sum_{d|n} \gamma_d$, where the coefficients satisfy $|\gamma_d| \leq 1$ and $\gamma_d = 0$ for $d > Q$. Our weight W has exactly this form (Section 4), with coefficients $\sigma = \pm 1$. Since d divides a $1/d$ fraction of all integers, the average value of f is $\bar{f} = \sum_d \gamma_d/d$.

Now let's slide a window $(x - H, x]$ of length H along the integers, where the endpoint x is a real number.

Since H is an integer, the window contains the same integers for every x between two consecutive integers, so the window sum is constant there. So averaging over real endpoints x is the same as averaging over integer endpoints - the real-number version is just a convenient way to write it.

On average, the window sum should be about $H\bar{f}$, and we call the difference the *gap*:

$$G(x) = \sum_{x-H < n \leq x} f(n) - H\bar{f} = \sum_d \gamma_d E_d(x), \quad E_d(x) = (\#\text{multiples of } d \text{ in } (x - H, x]) - \frac{H}{d}.$$

So the gap is a combination of the counting errors E_d , one for each divisor. Each E_d is a two-valued step function with period d and mean 0, taking values in $[-1, 1]$. For a single window, bounding every $|E_d|$ by 1 only shows that $|G(x)| \leq Q$, which is just the first roadblock again.

Instead, we look at the *average of $G(x)^2$* over all endpoints x in a range of length X . This is the right quantity to look at because of Chebyshev's inequality: if this average is V , then the endpoints where $|G(x)| > \lambda$ make up a total length of at most XV/λ^2 (in Step 4, we will see that each bad block produces such endpoints). So a small average square means that large gaps are rare.

The key lemma (informal)

Over all window endpoints x in a range of length X , the average of $G(x)^2$ is at most a constant times

$$H(\log Q)^3 + \frac{Q^4}{X},$$

as opposed to the Q^2 that the single-window bound gives.

There are two facts behind the main term.

- **Each E_d is small on average when d is large.** If d is much larger than H , a window of length H usually doesn't contain any multiple of d , so $E_d(x)$ is usually just the tiny constant $-H/d$. Its average square is at most about H/d (and at most $1/4$ for every d). Summing over $d \leq Q$ already gives a bound of $O(H \log Q)$.
- **Different E_d -s are nearly uncorrelated.**

If $\gcd(d, e) = 1$, the Chinese Remainder Theorem says that over a complete period, the integer endpoints visit every pair of residues mod d and mod e exactly once. Since H is an integer, averaging over these endpoints is the same as averaging over real endpoints, as above. So E_d and E_e behave like independent mean-zero random variables, and their average product over the period is exactly 0.

In general, writing $g = \gcd(d, e)$, the average of $E_d E_e$ is $g^2/(de)$ times the average square of E_g (the latter is $\alpha(1 - \alpha)$ for α being the fractional part of H/g), which is at most $H \gcd(d, e)/(de)$. Each such term is small, but there are many of them - summing $\gcd(d, e)/(de)$ over all pairs gives $(\log Q)^3$, with the three logs coming from g , d/g and e/g .

The error term Q^4/X comes from incomplete periods: the product $E_d E_e$ has period $\text{lcm}(d, e) \leq Q^2$, the leftover part of a period in the range can contribute up to about Q^2 , and there are about Q^2 pairs (d, e) .

Step 2: splitting W into two parts. We would like to apply the lemma to W itself, averaging over all endpoints in $[0, N]$, so that $X \approx N$.

Roadblock: the Q^4 error term

With $X \approx N$, the error term is Q^4/N . But the divisors $m^2 e$ of W go up to N (since m can be as large as $\sqrt{N}$), so this error term is about N^3 , which is useless.

Fix: split at $m \leq D = L^{12}$

Let's split $W(n)$ into the terms with $m > D$ (the *tail*) and the terms with $m \leq D$ (the *truncated part*), and write $W_r = B_r + W_r^{\leq D}$ for the corresponding block sums. If $W_r > C_1 T/L$, then one of the two parts exceeds $C_1 T/(2L)$ (this is the same pigeonhole idea as in Section 3). So Job A follows from two goals:

- **(A1)** the blocks with $B_r > C_1 T/(2L)$ have total length $O(N/L^{12})$;
- **(A2)** the blocks with $W_r^{\leq D} > C_1 T/(2L)$ have total length $O(N/L^{12})$.

The point of the split is that the tail has very little total weight, and the truncated part has only small divisors.

Step 3: the tail (goal A1), by Markov. Since $\sum_{m>D} 1/m^2 \leq 1/D$, the total weight of the tail over all blocks is only about $N/(DL)$. Every block with $B_r > C_1 T/(2L)$ uses up at least $C_1 T/(2L)$ of this total, so there are at most about $(N/(DL))/(T/L) = N/(DT)$ such blocks, with total length about $N/D = N/L^{12}$. This proves A1.

Note that Markov works here even though it failed in the second roadblock, because the tail's total weight is only $O(N/(DL))$, which saves the factor $1/D = L^{-12}$ that was missing. This is why $D = L^{12}$, matching the repair exponent.

Step 4: the truncated part (goal A2), by Chebyshev. First, note that the lemma now applies. The truncated part is again of the form $\sum_{d|n} \gamma_d$ with coefficients ± 1 or 0 (since e is determined by d , we can use $\gamma_d = \lambda_e$ when $d = m^2 e$ for some odd $m \leq D$, and 0 otherwise), and its divisors are at most $Q = D^2 Z$, which is a tiny power of N times a power of L . So the error term Q^4/N is about $N^{4/1000-1}$ times a power of L , which is negligible. Since $\log Q = O(L)$, the lemma gives, for windows of length $H = 3T$ (we'll see in a moment why $3T$):

the average of $G(x)^2$ over all endpoints is at most a constant times TL^3 .

To turn this into a statement about fixed blocks with $W_r^{\leq D} > C_1 T/(2L)$, we need two more observations.

- **From fixed blocks to moving windows.** A window of length T coincides with a given block only for endpoints x in an interval of length 1 (window sums only change when x crosses an integer), which is far too short to matter in an average over all endpoints. If we instead use windows of length $3T$, then for every endpoint x in $[(r+1)T, (r+2)T]$, an interval of length T , the window $(x-3T, x]$ contains block r , and since the weights are nonnegative, the window sum is at least the block sum. Different blocks give non-overlapping intervals of endpoints. (Near the start of the range, a window can include zero or negative integers. We use the divisor-sum formula there too - the paper checks that these weights are also nonnegative.)
- **From a large sum to a large gap.** For windows of length $3T$, the gap is measured from $3T\bar{f}$, where $\bar{f}$ is now the average truncated weight. The same density calculation as for W gives $\bar{f} = O(1/L)$, so $3T\bar{f} = O(T/L)$. We choose C_1 large enough that $3T\bar{f} \leq C_1 T/(4L)$ (this is the requirement on C_1 from Section 3). Then for every endpoint x in the interval of a bad block, the gap is $G(x) > C_1 T/(2L) - C_1 T/(4L) = C_1 T/(4L)$.

Putting the three ingredients together, we get

$$T \cdot \#(\text{bad blocks}) \leq \left| \left\{ x \in [T, N+T] : G(x) > \frac{C_1 T}{4L} \right\} \right| \leq \frac{\int_T^{N+T} G(x)^2 dx}{(C_1 T/(4L))^2} \lesssim \frac{N \cdot TL^3}{(T/L)^2} = \frac{NL^5}{T}$$

(where $|\cdot|$ denotes total length). The first inequality is the two observations above (each bad block gives an interval of length T of positions with a large gap, and these intervals don't overlap), the second is Chebyshev's inequality (each such position adds more than $(C_1 T/(4L))^2$ to the integral), and the third is the lemma (the average of $G(x)^2$ is $O(TL^3)$ over a range of length N).

In other words, the fraction of bad positions is at most $O(L^5/T)$, so longer blocks help, and with $T \geq L^{17}$, the bad length is $O(N/L^{12})$. This proves A2.

As an aside, this is also the technical reason (promised in Section 4) why Z must be a small power of N : since Q is Z times a power of L , the error term Q^4/N is negligible only if Z^4 is well below N , i.e., $Z < N^{1/4}$. Any fixed smaller power works, and $1/1000$ is just a convenient choice.

Result of Job A

The blocks with $W_r > C_1 T/L$ have total length at most about

$$\frac{N}{D} + \frac{NL^5}{T} \lesssim \frac{N}{L^{12}}$$

(up to a negligible term from Q^4), using $D = L^{12}$ and $T \geq L^{17}$.

6 Job B: computing $U_r \geq W_r$ in linear time

Recall from Section 3 what Job B has to deliver: numbers U_r with $U_r \geq W_r$ for every block, computed in $O(N)$ bit operations in total, with total overshoot $\sum_r (U_r - W_r) = O(N/L^{13})$.

Summing $W(n) = \sum_{(d,\sigma) \in \mathcal{S}} \sigma[d \mid n]$ (Section 4) over the integers n in block r counts, for each d , the multiples of d in the block:

$$W_r = \sum_{(d,\sigma) \in \mathcal{S}} \sigma \cdot \#\{k \geq 1 : dk \in \text{block } r\}.$$

So W_r is a signed count of pairs (d, k) - a divisor d from the list and a *multiplier* k - whose product lands in block r .

Roadblock: we can't afford to look at products one by one

Listing the products is already too expensive for $d = 1$ alone: there are about N products, most with about $\log N$ bits, so it costs about $N \log N$ just to write them down. So we need to count products *in groups*, without locating individual products.

6.1 Counting products in groups

One range of multipliers. For a fixed d , only $k < N/d$ matters. But this range depends on d , and we want to count divisors and multipliers independently. So we allow every pair (d, k) with $1 \leq k \leq N$, for every d - the extra products beyond N will be thrown away later, depending on where they lie.

Logarithmic bins and convolution. Since $\log_2(dk) = \log_2 d + \log_2 k$, let's sort factors by the size of their logarithm. Fix an integer K and let bin j be the interval $[2^{j/K}, 2^{(j+1)/K})$. If d lies in bin i and k in bin j , then

$$2^{(i+j)/K} \leq dk < 2^{(i+j+2)/K},$$

so the sum of the bin indices locates the product up to a factor of $2^{2/K}$. Near N , that is an uncertainty of a constant times N/K . We set $K = \lceil N/\eta \rceil$, so that near N , each product is located up to about η (we will choose $\eta = L^4$ at the end). There are only about $KL \approx NL/\eta$ bins.

Let A_i^+ and A_i^- count the divisors with sign $+1$ and -1 in bin i , and let B_j count the multipliers $k \leq N$ in bin j . Since every divisor is paired with every multiplier, the number of positive pairs whose bin indices add up to ℓ is the *convolution*

$$C_\ell^+ = \sum_{i+j=\ell} A_i^+ B_j,$$

and similarly for C_ℓ^- .

One multiplication computes a convolution. The numbers C_ℓ^+ are the coefficients of the polynomial product $(\sum_i A_i^+ x^i)(\sum_j B_j x^j)$. Substituting $x = b = 2^{4L+4}$ turns each histogram into a single integer, with its counts written one after the other in slots of $4L + 4$ bits, and their product is $\sum_\ell C_\ell^+ b^\ell$. Each C_ℓ^+ is at most the total number of pairs, $|\mathcal{S}| \cdot N \leq N^3 < b$, so every coefficient fits in its own slot, and no carry spills into the next one. So one integer multiplication gives all the C_ℓ^+ (this is *Kronecker substitution*), and a second one gives the C_ℓ^- . The integers have about $(NL/\eta) \cdot 4L$ bits, i.e., about N/L^2 for $\eta = L^4$, and fast integer multiplication handles them in fewer than N bit operations.

6.2 From pair counts to block bounds

For each index sum ℓ , the products counted in $C_\ell^\pm$ lie in a known interval, which we call the enclosure of ℓ (Section 6.4 makes this precise). Only enclosures that meet $[0, N)$ can contribute to our bounds, and these have length $O(\eta)$. Blocks have length T , which is much larger than η , so each such enclosure

meets at most two blocks (to meet three, it would have to cover the whole middle block). If it lies inside one block, we know where its pairs go.

Roadblock: enclosures crossing a boundary, plus signs

If an enclosure crosses a block boundary, we don't know how its pairs are split between the two blocks. And a wrong guess for a negative pair would make U_r *smaller* than W_r .

Fix: one-sided rounding

If the enclosure lies inside one block, add $C_\ell^+ - C_\ell^-$ to that block. If it crosses a boundary, add C_ℓ^+ to *every* block it meets, and subtract C_ℓ^- from *none*. Enclosures entirely at or beyond N are ignored, and the ends 0 and N count as boundaries (only the positive count is kept, for the first or last block).

This can only overestimate: a positive pair is always counted in its own block (and maybe one more), and a negative pair is subtracted only when its whole enclosure (and hence its product) lies in that block. So $U_r \geq W_r$.

For example, if an enclosure crossing a boundary holds 7 positive and 3 negative pairs, both blocks get $+7$ - subtracting 3 from either block could be wrong, since all 3 negative products might lie in the other block. The true contribution is 4, and the bounds receive 14, which is an overshoot of 10, but a safe one.

6.3 The total overshoot is small

Overshoot only comes from enclosures crossing a boundary, i.e., a multiple of T (including 0 and N). A product in such an enclosure is within about η of that multiple of T , and is less than $2N$. Each such pair adds at most 2 - a positive pair is counted in at most two blocks, and a negative pair is left out of at most one. So the total overshoot is at most twice the number of pairs (d, k) with $dk < 2N$ and dk within about η of a multiple of T .

Roadblock: what if many products sit near multiples of T ?

For some divisors, they all do - if $T \mid d$, every product dk is a multiple of T .

Fix: equidistribution mod T

Let $g = \gcd(d, T)$. As k runs through any T/g consecutive values, $dk \bmod T$ takes each multiple of g exactly once. About $\eta/g + 1$ of these residues (up to a constant) are within η of 0, so the fraction of products near a boundary is about $(\eta + g)/T$. This is about η/T , *provided g is small*. This is the reason behind two design choices. The core algorithm allows any block length T dividing N that is at most a fixed power of L , and we take T to be a *power of 2*; also, W only uses *odd m* . Since $d = m^2e$ with m odd and e squarefree, d has at most one factor of 2, so $g \leq 2$.

For a single divisor d , there are about $2N/d$ multipliers with $dk < 2N$. Each complete run of T/g multipliers puts only an $O(\eta/T)$ fraction near a boundary. There may also be one incomplete run at the end, but it contributes at most $O(\eta)$ near-boundary products, since no residue repeats within that run. So the total overshoot is bounded by

$$\sum_r (U_r - W_r) \lesssim \frac{N\eta}{T} \sum_{(d,\sigma) \in \mathcal{S}} \frac{1}{d} + \eta|\mathcal{S}|.$$

The sum of $1/d$ over the list is at most a constant times L : writing $d = m^2e$, it is at most $\sum_{e \leq Z} 1/e \approx$

$\log Z$ times $\sum_m 1/m^2$, which is a constant. Also, for each e , there are at most $\sqrt{N/e}$ choices of m , so

$$|\mathcal{S}| \leq \sqrt{N} \sum_{e \leq Z} e^{-1/2} \lesssim \sqrt{NZ}.$$

Thus

$$\sum_r (U_r - W_r) \lesssim \frac{N\eta L}{T} + \eta\sqrt{NZ}.$$

For $\eta = L^4$, the second term is $o(N/L^{13})$, which is small enough for Job B. It remains to make $N\eta L/T$ small enough too.

6.4 Technicalities: approximate bins

So far, we pretended that bins can be computed exactly. Instead, each factor is put into a *nearby* bin, using *certified* logarithms and powers of 2 (more specifically, $K \log_2 u$ and $2^{j/k}$) - values within $1/4$ of the true value, with a proof of the error bound, costing about L^3 bit operations each.

- **Divisors** (there are at most about $\sqrt{NZ}$ of them) get one logarithm each, which gives an index within 1 of $K \log_2 d$.
- **Multipliers** (N of them) are too many to compute one logarithm each. Instead, we compute the about NL/η bin boundaries $2^{j/K}$, rounded to integers, and count the integers between consecutive boundaries by subtraction. This gives an index within 2 of $K \log_2 k$. The first $4K$ multipliers are binned individually, since for small numbers, moving a boundary by 1 shifts the logarithmic index by too much.

Note that every factor is still put into exactly one bin, so the counts $C_\ell^\pm$ are exact for the bins we use. Only the location guarantee weakens: the index sum is within 3 of $K \log_2(dk)$, so the enclosure is taken slightly wider and rounded outwards, and it still contains every product counted in $C_\ell^\pm$. Every enclosure reaching into $[0, N)$ then has length at most 64η , which is the precise version of “about η ” from above. The numerical work is about $(NL/\eta + |\mathcal{S}|) \cdot L^3 = NL^4/\eta + o(N)$ bit operations.

We also need to account for moving things around on tapes. To build a histogram, we write down bin indices and counts, sort them by index, and add counts with the same index in a single scan. We collect the contributions to blocks in the same way. There are $O(NL/\eta + |\mathcal{S}|)$ records, each with $O(L)$ bits, so sorting costs

$$O((NL/\eta + |\mathcal{S}|)L^2).$$

For $\eta = L^4$, this is $o(N)$, so these operations fit within our time budget too.

6.5 Choosing η and T

Two requirements pull η in opposite directions:

- the numerics cost about NL^4/η , which is $O(N)$ only if $\eta \geq L^4$;
- the overshoot $N\eta L/T$ must be $O(N/L^{13})$, which needs $T \geq \eta L^{14}$ (a larger η means wider enclosures, more products near boundaries, and hence longer blocks).

Taking $\eta = L^4$ gives $T \geq L^{18}$. **This is where the 18 comes from.** It also satisfies Job A’s requirement $T \geq L^{17}$, and is still a fixed power of L , as the core algorithm needs.

Result of Job B

With $\eta = L^4$ and $T \geq L^{18}$, we compute $U_r \geq W_r$ for all blocks in $O(N)$ bit operations, with total overshoot at most about $N\eta L/T \leq N/L^{13}$.

7 Parameter bookkeeping

For the sake of completeness, here is how all the parameters fit together.

Parameter	Value	Why
Repair budget	N/L^{12}	the per-number test costs $L^{11.5}$, and $12 > 11.5$
D	L^{12}	the tail beyond D marks length N/D
η	L^4	the numerics cost NL^4/η
T	$\approx L^{18}$	the excess requires $T \geq \eta L^{14}$ (the variance only needs L^{17})
R	$\approx C_1 T/L$	keeps the core's factor $1 + RL/T$ bounded
Z	$N^{1/1000}$	small enough that every error term involving Z is a negligible power of N

Since T is polylogarithmic, $\log T = O(\log \log N)$, which the core needs. Certification, repair and the conversion to primes each cost $O(N)$, and the core costs $N(\log \log N)^{1+o(1)}$, which dominates.

References

- D. Harvey, *Faster enumeration of primes*, [arXiv:2606.22851](https://arxiv.org/abs/2606.22851), 2026.
- Deterministic enumeration of primes in $N(\log \log N)^{1+o(1)}$ bit operations*, manuscript, 24 October 2026.
- J. Friedlander and H. Iwaniec, *Opera de Cribro*, AMS Colloquium Publications 57, 2010.
- K. Matomäki, *Almost primes in almost all very short intervals*, J. London Math. Soc. 106 (2022), 1061–1097.
- J. von zur Gathen and J. Gerhard, *Modern Computer Algebra*, 3rd ed., Cambridge University Press, 2013.
- A. Schönhage and V. Strassen, *Schnelle Multiplikation großer Zahlen*, Computing 7 (1971), 281–292.
- D. Harvey and J. van der Hoeven, *Integer multiplication in time $O(n \log n)$* , Annals of Math. 193 (2021), 563–617.