MATHEMATICS OF COMPUTATION
Volume 78, Number 268, October 2009, Pages 2283–2305
S 0025-5718(09)02249-2
Article electronically published on April 3, 2009

COMPUTING PRIME HARMONIC SUMS

ERIC BACH, DOMINIC KLYVE, AND JONATHAN P. SORENSON

ABSTRACT. We discuss a method for computing $\sum_{p\le x} 1/p$, using time about $x^{2/3}$ and space about $x^{1/3}$. It is based on the Meissel-Lehmer algorithm for computing the prime-counting function $\pi(x)$, which was adapted and improved by Lagarias, Miller, and Odlyzko. We used this algorithm to determine the first point at which the prime harmonic sum first crosses 4.

1. INTRODUCTION

We consider $S(x) = \sum_{p\le x} 1/p$, which is a prime-number analog of the harmonic sum. Its values for prime x are

$$\frac{1}{2},\ \frac{5}{6},\ \frac{31}{30},\ \frac{247}{210},\ \frac{2927}{2310},\ \frac{40361}{30030},\ \frac{716167}{510510},\ \dots\ . \tag{1.1}$$

In this paper we discuss algorithms for computing this sum, and for solving the equation $\sum_{p\le x} 1/p = y$. More precisely, we seek the least integer x for which the sum equals or exceeds y.

Both of these problems are extremely hard by conventional standards of algorithm analysis. Indeed, by the prime number theorem the denominator of $\sum_{p\le x} 1/p$ has about $x/\log 2$ bits, so just to write down the sum exactly requires an effort that is exponential in the input length. This rapid growth is already apparent from (1.1). Even worse, since $\sum_{p\le x} 1/p \sim \log\log x$, the least solution x to $\sum_{p\le x} 1/p \ge y$ has length doubly exponential in the length of y.

It makes sense, therefore, to compute a floating-point approximation to the prime harmonic sum. We give an algorithm for this that uses about $x^{2/3}$ operations, and space about $x^{1/3}$. Our method is based on the Meissel-Lehmer algorithm [26, 22] for computing the prime-counting function $\pi(x)$, which was adapted and improved by Lagarias, Miller, and Odlyzko [8, 19, 20].

Here are the ideas behind the algorithm. If we start with the harmonic sum and sieve it, by removing terms indexed by multiples of small primes, the result approximates, in some sense, the sum we want. We sieve up to $x^{1/3}$. This removes terms $1/p$ with $p \le x^{1/3}$, which are easy to put back in, but leaves terms $1/(pq)$ where p and q are primes larger than $x^{1/3}$. These terms can be summed separately

Received by the editor June 19, 2008 and, in revised form, November 28, 2008.
2000 *Mathematics Subject Classification.* Primary 11Y16; Secondary 11Y35, 11N05, 68Q25.
E. Bach was supported by NSF grants CCR-0523680, CCF-0635355, and a Vilas Associate Award from the Wisconsin Alumni Research Foundation.
D. Klyve was supported by NSF grant DMS-0401422.
J. P. Sorenson was supported by a grant from the Holcomb Awards Committee.
A preliminary version of this work was presented as a poster at ANTS-VII in Berlin, Germany, July 2006.

and then subtracted to get what we want. The sieved harmonic sum obeys a recurrence relation. Used naively, this recurrence relation is no better than a straight evaluation of the sum. Judicious pruning of the recursion tree, however, leads to an algorithm that is.

We extend this to a method to ascertain when the sequence of prime harmonic sums first exceeds a given value. Our algorithm requires one evaluation of the sum and has comparable running time, assuming the Riemann hypothesis (RH). This hypothesis is only used for the running time and does not affect correctness. This method was used to determine when the prime harmonic sum first exceeds 4, which was an open problem of Neil Sloane. Specifically, we found that for the primes

$$x_4 = 1801241230056600523 \quad \text{and} \quad x_4 - 56 = 1801241230056600467,$$

we have

$$\sum_{p \le x_4 - 56} \frac{1}{p} \quad < \quad 4 \quad < \quad \sum_{p \le x_4} \frac{1}{p},$$

and that there are no primes in the interval $(x_4 - 56, x_4)$.

We are the first, we believe, to fully work out and test such a method for prime harmonic sums. Other authors, however, have discussed the possibility of this. D. H. Lehmer [22] has pointed out that the Meissel-Lehmer method could be used to sum multiplicative functions over primes. Lagarias, Miller, and Odlyzko [19] have also mentioned that their algorithm could be used for other sums over primes.

The algorithm we describe for $S(x)$ was developed independently by two groups. The first and third authors used it to answer Sloane's question, and we describe this computation here. The second author used it in his Ph.D. dissertation to compute explicit estimates for Brun's constant and the twin prime counting function [18]. This required an extension to prime harmonic sums over arithmetic progressions, and that work will be described in a separate paper.

The rest of this paper is organized as follows. In Section 2 we present the combinatorial dissection underlying our summation algorithm, which is given in full detail in Section 3. Since floating-point arithmetic is more expensive than integer arithmetic, we need more sophisticated data structures than the stacked arrays of [19]; Section 4 is devoted to these. In Section 5, we use analytic methods to independently estimate the key partial results that our algorithm computes. These estimates are of obvious utility for checking an implementation, but are of independent interest. Section 6 gives asymptotic theorems on the cost of our summation procedure. Section 7 presents our equation-solving algorithm. Finally, in Section 8 we discuss our computation for Sloane's problem, and in Section 9 present floating-point error analysis to argue for its validity.

If we were asked to boil our algorithm down to its essence, we would respond by giving the two formulas (2.7) and (3.2), which appear below. Our exposition is guided by these, in the sense that expressions will be discussed in the same order that they appear in these formulas.

2. Basic formulas

Let $p_1 < p_2 < p_3 < \cdots$ be the primes. Let $\ell(n)$ denote the least prime factor of n. The sieved harmonic sum is

$$\phi(x, a) = \sum_{\substack{n \le x \\ \ell(n) > p_a}} \frac{1}{n}.$$

Let S_k be the same sum, but with n restricted to numbers with exactly k prime factors. (Repeated primes count.) By convention, $S_0 = 1$, so that

$$\phi(x,a) = S_0 + S_1 + S_2 + \cdots . \tag{2.1}$$

For $x, a \geq 1$ we have

$$\phi(x, a-1) = \phi(x,a) + \frac{1}{p_a}\phi(x/p_a, a-1),$$

since every denominator in the left sum is either composed of primes $> p_a$, or is divisible by p_a. This implies the recurrence relation

$$\phi(x,a) = \phi(x,a-1) - \frac{1}{p_a}\phi(x/p_a, a-1). \tag{2.2}$$

The recurrence may be terminated as soon as the second argument is small enough to allow direct evaluation. For example, $\phi(x,0)$ is the harmonic sum $\sum_{n\leq x} 1/n$. Our algorithm uses $\phi(x,1)$, for which see (3.4).

We will now take $a = \pi(x^{1/3})$, which makes p_a the largest prime $\leq x^{1/3}$. Note that

$$S_1 = \sum_{p\leq x}\frac{1}{p} - \sum_{p\leq p_a}\frac{1}{p} \tag{2.3}$$

and

$$S_2 = \sum_{\substack{pq\leq x\\ p_a<p\leq q}}\frac{1}{pq} \tag{2.4}$$

$$= \sum_{p_a<p\leq\sqrt{x}}\frac{1}{p}\sum_{p\leq q\leq x/p}\frac{1}{q} \tag{2.5}$$

$$= \sum_{p_a<p\leq\sqrt{x}}\frac{1}{p}\left[\sum_{q\leq x/p}\frac{1}{q} - \sum_{q\leq p}\frac{1}{q} + \frac{1}{p}\right]. \tag{2.6}$$

By the choice of a, $S_k = 0$ for all further k.

Combining (2.1), (2.3), and (2.4) we see that

$$\sum_{p\leq x}\frac{1}{p} = \sum_{p\leq p_a}\frac{1}{p} - S_2 + \phi(x,a) - 1. \tag{2.7}$$

Our basic strategy will be to compute each term of this separately, but with one modification. We use the recurrence relation to break $\phi(x,a)$ into two sums, which we call ϕ_o and ϕ_s.

3. The algorithm

We now describe this algorithm in more detail. For ease of description we will assume that x is a perfect cube. Since

$$x^3 - \lfloor x^{1/3}\rfloor^3 = O(x^{2/3}),$$

we can extend the algorithm to cover all cases without increasing its asymptotic complexity. (This assumption is only necessary for evaluating ϕ_s.)

3.1. **Sieves.** Our algorithm requires a list of primes up to $x^{1/3}$, and tables of $\mu(n)$ (the Möbius function) and $\ell(n)$ (the smallest prime factor of n), for $n \le x^{1/3}$. All of these can be computed in time $O(x^{1/3+\epsilon})$ using the sieve of Eratosthenes.

Primes larger than this bound are dealt with using segmentation. Accordingly, let the k-th segment Δ_k consist of all integers t satisfying

$$kx^{1/3} \le t < (k+1)x^{1/3}. \tag{3.1}$$

We will take $0 \le k \le x^{1/3}$. Note that by bringing segments into storage, and sieving them, it is possible to enumerate the primes $\le x^{2/3}$ in either increasing or decreasing order (see, for example, [31]).

3.2. **Evaluating the main term.** Our initial sieving determined p_a, which is the largest prime $\le x^{1/3}$. So we can evaluate the first term in (2.7), which is straightforward. To enhance numerical accuracy, however, we sum from larger to smaller p.

3.3. **Evaluating S_2.** Our next goal is to evaluate (2.4). To do this, we let p increase and update the inner sum in brackets for each new prime. This will require the two segments Δ_k (for increasing k) and Δ_l (for decreasing l). Since $p \le \sqrt{x}$, we need only $k \le x^{1/6}$. We now consider the incremental effect of including one more p. Let p' be the old value of p, i.e. the last prime used. We must decrease the first sum in brackets by the contribution of all q satisfying

$$\frac{x}{p} < q \le \frac{x}{p'}.$$

Therefore q decreases. Since $p' \ge x^{1/3}$ (roughly), the largest l we will need is also about $x^{1/3}$. By enumerating p in increasing order and q in decreasing order, we can get (2.4) efficiently. Note that each p and each q are considered once.

3.4. **Evaluating $\phi(x,a)$.** Imagine that the recurrence relation (2.2) is terminated whenever we reach $\phi(x/m,b)$ with either:

(a) $b = 1$ and $m \le x^{1/3}$ (an ordinary node), or
(b) $m > x^{1/3}$ (a special node).

From (2.2) it can be seen that each such node contributes

$$\frac{\mu(m)}{m}\phi(x/m,b)$$

to the total sum. Furthermore, a node is ordinary or special, but not both, and can be reached in only one way. Hence, we have

$$\phi(x,a) = \sum_{(m,b)\ \text{ordinary}} \frac{\mu(m)}{m}\phi(x/m,b) + \sum_{(m,b)\ \text{special}} \frac{\mu(m)}{m}\phi(x/m,b). \tag{3.2}$$

Call these sums ϕ_o and ϕ_s, respectively. We will evaluate each without generating the tree.

3.5. **Evaluating ϕ_o.** By (2.2), $\phi(x/m,1)$ is ordinary iff m is an odd square-free number $\le x^{1/3}$. Hence

$$\phi_o = \sum_{\substack{m\le x^{1/3}\\ m \text{ odd}}} \frac{\mu(m)}{m}\phi(x/m,1). \tag{3.3}$$

For this sum we use an analytic approximation to $\phi(y,1)$, obtained via Euler-Maclaurin summation. When y is an odd positive integer, $\phi(y,1) = \sum_{\substack{m\le y\\ m \text{ odd}}} 1/m$, so

$$\phi(y,1) = \frac{\log y}{2} + \frac{\gamma+\log 2}{2} + \frac{1}{2y} - \frac{1}{6y^2} + \frac{1}{15y^4} - \frac{8}{63y^6} + O(y^{-8}). \tag{3.4}$$

(See (25) of [23].) We can evaluate $\phi(z,1)$ for any z by noting that the largest odd integer $\le z$ is $2\lfloor (z-1)/2\rfloor + 1$.

For even $\nu \ge 2$, the coefficient c_ν of $y^{-\nu}$ in (3.4) is $(-1)^{\nu/2}2^{\nu-1}B_\nu/\nu$, where B_ν is the ν-th Bernoulli number. Thus, $|c_\nu|$ is asymptotic to $(\nu-1)!/\pi^\nu$. Therefore, (3.4) does not converge. If we truncate it, however, we incur an error with the same sign as, and less in absolute value than, the first term omitted. This can be proved in the same way as the corresponding result for harmonic sums [16, p. 114].

If sufficiently accurate special function code is available, we can use the following idea, in lieu of (3.4). Using [1, 6.3.4] with $y = 2n-1$, we see that

$$\phi(y,1) = \frac{\gamma}{2} + \log 2 + \frac{1}{2}\psi(y/2+1),$$

where $\psi = \Gamma'/\Gamma$ is the digamma function.

3.6. **Evaluating ϕ_s.** We first note that when $\phi(x/m,b)$ is a special node, we have $x/m \in \Delta_k$ iff

$$m = m'q \qquad \text{and} \qquad \frac{x^{2/3}}{(k+1)q} < m' \le \frac{x^{2/3}}{kq}, \tag{3.5}$$

where $q = p_{b+1}$. It can be seen that there are no special nodes with $k = 0$, so that (3.5) makes sense in all cases. Recall that x is a perfect cube, so that $x/m \in \Delta_k$ iff $\lfloor x/m\rfloor$ is. Our strategy will be to evaluate all special nodes in the k-th segment, and let k increase. This will require us to save the cumulative sums

$$C_b = \sum_{\substack{n\in\text{previous segments}\\ (n,2\cdot 3\cdots p_b)=1}} \frac{1}{n},$$

for $b = 1,\dots,a-2$. We now access special nodes by finding their parents. To process the special nodes in Δ_k, we do the following.

(1) For $i \in \Delta_k$, set $A_i = \begin{cases} 1/i, & \text{if } i \text{ is odd;}\\ 0, & \text{if } i \text{ is even.}\end{cases}$

(2) For $b = 1,2,\dots,a-2$:
   (a) Set $q = p_{b+1}$. [We can now compute $\phi(y,b)$, when $y \in \Delta_k$.]
   (b) For m' satisfying the second part of 3.5:
   $m = qm'$,
   $\phi(x/m,b) = C_b + \sum_{kx^{1/3}\le i\le x/m} A_i$,
   add $\frac{\mu(m)}{m}\phi(x/m,b)$ to the total for ϕ_s.
   (c) Update C_b by adding $\sum_{i\in\Delta_k} A_i$.
   (d) Set $A_i = 0$ for all $i \equiv 0 \bmod q$.

In the description above, A is an array with integer indices, implemented so as to allow rapid evaluation of sums such as $\sum_{u \le i \le v} A_i$.

Since the intervals defined by (3.5) can be comparable to $x^{2/3}$ in length, we need an economical way to run through the m'. This is provided by the following characterization of special nodes. For a given b, a node $\phi(x/m, b)$ is special iff $m = m' p_{b+1}$, with m' an odd square-free integer $\le x^{1/3}$, such that the smallest prime factor of m' exceeds p_{b+1}.

Using a sieve, we can make a list of all possible m', together with their smallest prime factors. For each b, we will have a pointer into this list, which decreases as more m' are tried. Initially, each pointer starts at $\lfloor x^{1/3} \rfloor$. When we process the k-th segment, we compute the lower bound in (3.5), which may make several new m' eligible. Each can be tested for suitability using $O(1)$ arithmetic operations.

It is interesting to consider the precise order in which special nodes are enumerated. Generally speaking, there is a tendency for earlier nodes to be associated with larger values of m and b, which is good numerically, since their contributions will be smaller. The distribution of special nodes is not uniform over segments, but seems to fall off as k increases.

This concludes the description of the algorithm.

4. Data structures

To evaluate ϕ_s, we require a data structure for maintaining values A_i, together with information to help in computing "range sums" $\sum_{u \le i \le v} A_i$. The classic solution is to use a binary tree in which nodes contain the sum of all array elements in their subtrees. Using the $2i, 2i+1$ trick for storing a full binary tree in a linear array, we can do this for N elements using about $2N$ cells. The time for each operation (set, clear, range sum) is $O(\log N)$ arithmetic operations.

For our implementation, we modified this in two ways. First, we used d-ary trees and varied d. The justification for this is empirical. Early on in the algorithm, there are many special nodes per block, whereas later blocks have very few special nodes. So it is good to use small d earlier, and large d later. Second, to minimize cache effects, it is wise to make the structure small. So we do not store the actual value of A_i, but merely a bit indicating whether it is zero or not. The partial sums for non-leaf nodes are stored in full.

In these tree-based algorithms, partial sums are stored for nested subsets of indices. Fredman [10] has studied a more general class of algorithms, in which the subsets can properly overlap. This leads to a gain in efficiency, but the precise rules for forming the subsets are complicated, so we did not attempt to implement these.

4.1. Array representations for d-ary trees.

Our structure is based on generalizing the $2i$, $2i+1$ trick for binary trees, so we review this first. Knuth [16, p. 401] gives this for arrays starting at 1 but we want to start at 0. The parent of i is at location $\lfloor i/d \rfloor - 1$. Therefore, the children of node i are at locations $di+1, \ldots, di+d$. All nodes but possibly one have d or 0 children.

Suppose there are n nodes. Then the least leaf is the smallest i such that $di + 1 > n - 1$. This location is

$$\left\lfloor \frac{n-2}{d} \right\rfloor + 1.$$

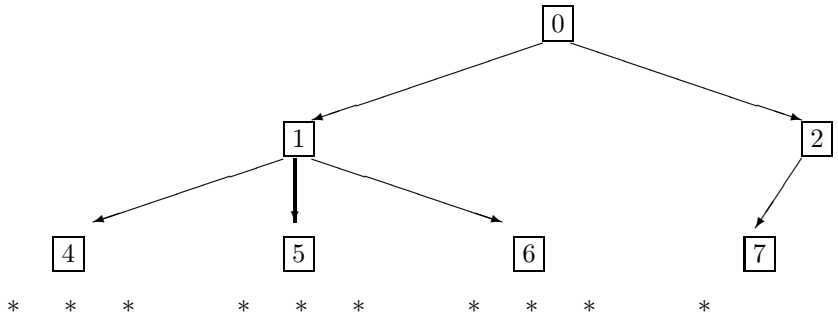

FIGURE 1. Trinary tree for $s = 10$ bits

Conversely, suppose there are ℓ leaves. How many nodes are there? Let the node locations be $0, \ldots, n-1$. Then

$$\ell = (n-1) - [\text{least leaf}] + 1 = n - 1 - \left\lfloor \frac{n-2}{d} \right\rfloor.$$

We can eliminate the floor by writing two inequalities. When this is done, and we solve for n, we find we must have

$$\frac{d\ell - 2}{d-1} < n \le \frac{d\ell - 2}{d-1} + 1 + \frac{1}{d-1}.$$

We want the minimum n of course, so we can discard the last term to find that

$$n = \left\lfloor \frac{d\ell - 2}{d-1} \right\rfloor + 1.$$

So the array is indexed over $0, \ldots, \left\lfloor \frac{d\ell-2}{d-1} \right\rfloor$.

To compute range sums, we also need to know the "older" siblings of a node i. If we agree that the eldest child has rank 0, then the rank of node i is

$$r = i \bmod d.$$

Therefore, node i's older siblings are at locations

$$i - (r-1), \ldots, i - 1.$$

4.2. **Our range sum structure.** We now describe how we store A_i, $i \in \Delta_k$. Using an offset, we may as well assume that indices start at 0. That is, we work with $A_0, \ldots, A_{s-1}$. We keep an array of s bits indicating whether A_i is zero, and a d-ary tree of floating-point values for parents, grandparents, etc. (Since we always start from the bottom, any A_i can be computed as needed.) In our algorithm the ranges for sums always start at 0, so we can avoid special cases by using a little trick. Imagine that $A_0, \ldots, A_{s-1}$ are all the way to the left, at the beginning of the first row in which they will fit. The leaves of the actual tree are their parents, of which there are $\left\lfloor \frac{s}{d} \right\rfloor + 1$.

In Figure 1 is an example, with $d = 3$ and $s = 10$. We indicate the bits using $*$.

To determine the number of floating-point nodes, we first compute lc ("left corner"), which is the location A_0 would have were it in the tree. (In the example above, $lc = 13$.) This is easily done with a loop, since the sequence of left corners is

$0, 1, 1+d, 1+d+d^2, \ldots$. Then the last floating-point node has the index of A_{s-1}'s parent, so the size must be

$$\left\lfloor \frac{lc + (s-1) - 1}{d} \right\rfloor + 1.$$

The parent of A_i is at location

$$\left\lfloor \frac{lc + i - 1}{d} \right\rfloor.$$

To clear location i, we walk up the tree and subtract. To compute the prefix sum out to location i, we walk up the tree and add each node's elder siblings to the total.

It is interesting to compare the size of our structure to an optimal tree, in which the parents of the A_i are put into the last nodes (and may be split between rows). Let $\ell = \lfloor \frac{s}{d} \rfloor + 1$. Our tree stores the parents of the A_i in the first ℓ leaves on row λ, where

$$\lambda = \lfloor \log_d \ell \rfloor.$$

Hence, it uses

$$1 + d + \cdots + d^{\lambda-1} + \lambda = \frac{d^\lambda - 1}{d-1} + \lambda$$

floating-point nodes. For large d this is about

$$\frac{s}{d} + \frac{s}{d^2}.$$

On the other hand, the optimal tree has, for large d, about

$$\frac{d\ell - 2}{d-1} \approx \frac{s}{d}$$

nodes. So the sizes are about the same.

5. Analytic results

In this section we collect various estimates useful for either analyzing or checking the algorithm. We use asymptotic notation (O, Θ, o, etc.) with the usual meaning, but emphasize that we write $f \sim g$ when the ratio f/g has the limit 1.

5.1. **A theorem of Mertens.**

As $x \to \infty$, we have

$$\sum_{p \le x} 1/p = \log\log x + B + O\left(\frac{1}{\log x}\right),$$

where

$$B = \gamma + \sum_p \left(\log(1 - p^{-1}) + p^{-1}\right) = 0.261497212847643\ldots,$$

with

$$\gamma = 0.577215664901532\ldots.$$

These constants can be computed in polynomial time [4].

Let us use Mertens's theorem to approximate the main term. Since p_a is the largest prime less than or equal to $x^{1/3}$, the main term is

$$\sum_{p \le x^{1/3}} 1/p \sim \log\log x.$$

As we will see, the other terms are $\Theta(1)$.

5.2. **Asymptotic value of S_2.** By our choice of p_a we have

$$S_2 = \sum_{\substack{pq \le x \\ x^{1/3} < p \le q}} \frac{1}{pq}. \tag{5.1}$$

Our goal is to show that this has a limit as $x \to \infty$ and compute it numerically.

Let us call q "small" if $q \le x^{1/2}$, and "large" otherwise.

The sum over small q is

$$\frac{1}{2}\left(\sum_{x^{1/3} < p \le x^{1/2}} \frac{1}{p}\right)^2 + o(1).$$

(Note that the contribution to (5.1) of terms with $p = q$ is $o(1)$ as $x \to \infty$.) By the prime number theorem, this has the limit

$$\frac{1}{2}(\log 3 - \log 2)^2 = 0.082201\ldots. \tag{5.2}$$

We now write the sum over large q as

$$\sum_{x^{1/3} < p \le x^{1/2}} \frac{1}{p} \sum_{x^{1/2} < q \le x/p} \frac{1}{q}.$$

By the prime number theorem, we have for the inner sum (since the additive constant will cancel)

$$\begin{aligned} \sum_{x^{1/2} < q \le x/p} \frac{1}{q} &= \log\log(x/p) - \log\log(x^{1/2}) + o(1) \\ &= \log 2 + \log\left(1 - \frac{\log p}{\log x}\right) + o(1). \end{aligned}$$

So

$$\sum_{x^{1/3} < p \le x^{1/2}} \frac{1}{p} \sum_{x^{1/2} < q \le x/p} \frac{1}{q} = (\log 2)(\log 3 - \log 2) - \int_{x^{1/3}}^{x^{1/2}} \log\left(1 - \frac{\log t}{\log x}\right) \frac{d\theta(t)}{t \log t} + o(1).$$

Here $\theta(t) = \sum_{p \le t} \log p$. Numerically we have

$$(\log 2)(\log 3 - \log 2) = 0.281047\ldots. \tag{5.3}$$

Put $\theta(t) = t + \epsilon(t)$ in the integral above. The main term becomes, after substituting $u = (\log t)/(\log x)$,

$$\int_{1/3}^{1/2} \frac{\log(1-u)}{u} du = -0.216027\ldots. \tag{5.4}$$

(This integral can be expressed using the dilogarithm function; see [3, p. 102]). The error term becomes, after integration by parts, the sum of three terms, which

are

$$A := \frac{\epsilon(t)}{t \log t} \log\left(1 - \frac{\log t}{\log x}\right) \Bigg|_{x^{1/3}}^{x^{1/2}};$$

$$B := \int_{x^{1/3}}^{x^{1/2}} \epsilon(t) \log\left(1 - \frac{\log t}{\log x}\right) \frac{1 + \log t}{t^2 \log^2 t} dt;$$

$$C := \int_{x^{1/3}}^{x^{1/2}} \epsilon(t) \frac{dt}{t^2 \log t \log x \left(1 - \frac{\log t}{\log x}\right)}.$$

We can estimate these easily if we first observe that $\log t / \log x$ is bounded away from 0 and 1 on the interval of integration. Each is $o(1)$ by the prime number theorem.

Combining (5.2)–(5.4) we find that

$$S_2 \sim 0.147219\ldots\ . \tag{5.5}$$

This can serve as a check on a computation of S_2. For example, for $x \doteq 1.8 \times 10^{18}$, we found that $S_2(x) = 0.147174\ldots$. Note that $x^{1/3}$ is about 10^6, so we should, granting the Riemann hypothesis, expect the two numbers to match to about 3 digits.

5.3. **Counting nodes.** The number of ordinary nodes is

$$\sum_{\substack{m \le x^{1/3} \\ m \ odd}} \mu(m)^2 \sim \frac{1}{2} \prod_{p \ge 3} \left(1 - \frac{1}{p^2}\right) x^{1/3} = \frac{4}{\pi^2} x^{1/3} \tag{5.6}$$

[21, p. 634]. (Prachar [28] estimates the error in this approximation.) We found this approximation to be surprisingly accurate. For example, for the perfect cube x of Section 8 (about 1.8×10^{18}), there were 493116 ordinary nodes, whereas the right side of (5.6) gives 493118.

The number of special nodes is

$$\frac{\pi(x^{1/3})^2}{2} + O(x^{1/2}) \sim \frac{9}{2} \frac{x^{2/3}}{\log^2 x} \tag{5.7}$$

[19, Lemma 5.1]. For the same x, we observed 4430940725 special nodes, whereas the first term in (5.7) gives $4.428\ldots \times 10^9$.

Ironically, the special nodes far outnumber the ordinary nodes. We also note that for most special nodes, m is a product of two primes.

5.4. **Truncation error for $\phi(y, 1)$.** Suppose we use (3.4) up to, but not including, the term $c_\nu y^{-\nu}$. If the terms decrease rapidly, the truncation error should be about

$$\frac{c_\nu}{x^\nu} \sum_{\substack{m \le x^{1/3} \\ m \ \text{odd}}} \mu(m) m^{\nu - 1}. \tag{5.8}$$

It will be useful to estimate this rigorously. By the enveloping property of (3.4), the truncation error has absolute value no more than

$$|c_\nu| \sum_{\substack{m \le x^{1/3} \\ m \ \text{odd}}} \frac{1}{m y^\nu}. \tag{5.9}$$

Assume $x \geq 8$, so that $y \geq x/m - 2 \geq x/(2m)$. Then, by comparing the sum to an integral, we see that the truncation error will not exceed

$$2^\nu |c_\nu| x^{-2\nu/3} \left[\frac{1}{\nu} + x^{-1/3} \right]. \tag{5.10}$$

This estimate is good enough for our purposes, but could be improved by taking the cancellation in (5.8) into account.

The relevant sum involves powers twisted by the Möbius function, and interesting work on such sums has been done by Grosswald [13].

Since (3.4) does not converge, we must estimate the maximum precision it can deliver. Since $\nu, x \geq 1$, the bound (5.10) is at most $2^{\nu+1}|c_\nu|x^{-2\nu/3}$, and, using Stirling's formula, the logarithm of this quantity is asymptotically

$$-\frac{2}{3}\nu \log x + \nu \log \nu - \nu \log(e\pi/2) - 1/2 \log \nu - 1/2 \log(8\pi). \tag{5.11}$$

Treating ν as a continuous variable, this is minimized near $\nu = (\pi/2)x^{2/3}$. (Experiments, using the actual bound (5.10), confirm this.) Substituting this value into (5.11) and simplifying the result, we see that the maximum precision available is $\Theta(x^{2/3})$ bits.

5.5. **Asympotic value of ϕ_o.** Recall that

$$\sum_{m \geq 1} \frac{\mu(m)}{m^s} = \frac{1}{\zeta(s)}. \tag{5.12}$$

If we write E and O for the sum over even and odd m, respectively, we find easily that $E = -2^{-s}O$, which gives us

$$\sum_{\substack{m \geq 1 \\ m \text{ odd}}} \frac{\mu(m)}{m^s} = \left(1 - \frac{1}{2^s}\right)^{-1} \frac{1}{\zeta(s)} \tag{5.13}$$

and

$$\sum_{\substack{m \geq 1 \\ m \text{ odd}}} \frac{\mu(m) \log m}{m^s} = -\frac{d}{ds}\left(\left(1 - \frac{1}{2^s}\right)^{-1} \frac{1}{\zeta(s)} \right). \tag{5.14}$$

These are absolutely convergent for $\Re(s) > 1$ and conditionally convergent on $\Re(s) = 1$ [21, p. 625]. By the Abelian theorem for Dirichlet series [33, cor. 1c, p. 183], we can let $s \to 1$ and obtain

$$\sum_{\substack{m \geq 1 \\ m \text{ odd}}} \frac{\mu(m)}{m} = 0 \tag{5.15}$$

and

$$\sum_{\substack{m \geq 1 \\ m \text{ odd}}} \frac{\mu(m) \log m}{m} = -2. \tag{5.16}$$

We need to know that the sum in (5.15) does not converge too slowly. This follows from a result of Landau [21, p. 632]. He proved that if k and ℓ are positive integers,

and q and t are real, then

$$\sum_{\substack{m\geq z+1\\ m\equiv \ell \pmod k}} \frac{\mu(m)(\log m)^q}{m^{1+it}} = O\left(e^{-(1/5)\sqrt{\log z}}\right).$$

In this, we take $k=2$, $\ell=1$, and $q=t=0$. Since the sum without the bounds on m vanishes, we conclude that

$$\sum_{\substack{m\leq z\\ m \text{ odd}}} \frac{\mu(m)}{m} = O\left(e^{-(1/5)\sqrt{\log z}}\right) = o\left(\frac{1}{\log z}\right). \tag{5.17}$$

By the enveloping property of (3.4), for odd integral $y\geq 1$ we have

$$\phi(y,1) = \frac{\log y}{2} + \frac{\gamma+\log 2}{2} + \epsilon\frac{1}{2y}, \qquad 0\leq\epsilon\leq 1. \tag{5.18}$$

Taking y to be the largest odd integer below x/m, we can substitute this in (3.3) and find that ϕ_o equals

$$-\frac{1}{2}\sum_{\substack{m\leq x^{1/3}\\ m \text{ odd}}} \frac{\mu(m)\log m}{m} + \frac{\log x+\gamma+\log 2}{2}\sum_{\substack{m\leq x^{1/3}\\ m \text{ odd}}} \frac{\mu(m)}{m} + \sum_{\substack{m\leq x^{1/3}\\ m \text{ odd}}} \mu(m)O(x^{-2/3}).$$

Here, we observed that $\log y = \log(x/m)+O(m/x)$ and $y\geq x/m-2\geq x^{2/3}-2$.

The first term has the limit 1 by (5.16). Using (5.17) with $z=x^{1/3}$, the second term is $o(1)$. Finally, the third term is $O(x^{-1/3})$, so it is also $o(1)$. To summarize, as $x\to\infty$ we have

$$\phi_o \sim 1. \tag{5.19}$$

As a check, for the x of Section 8 (about 1.8×10^{18}), we found $\phi_o \doteq 0.9979826\ldots$.

5.6. **Bounds for node values.** For any node (m,b), we have

$$1\leq \phi(x/m,b)\leq \log x - \log m + 1.$$

The lower bound holds because sieving never removes 1 from a harmonic sum, and the upper bound holds by comparing the harmonic sum to an integral.

For special nodes, the upper bound can be improved. Since $m > x^{1/3}$ in this case, the upper bound becomes

$$\phi(x/m,b)\leq (2/3)\log x + 1.$$

As a consequence, we can afford some error when evaluating these, because

$$\phi(x/m,b)/m \leq (2/3)x^{-1/3}(\log x+1).$$

5.7. **Asymptotic value of ϕ_s.** If we apply the Mertens theorem to both sums in (2.7), we see that

$$\log 3 + \phi_o + \phi_s - 1 - S_2 = o(1).$$

Using $\phi_o\sim 1$, and the asymptotic value of S_2, given by (5.5), we find that

$$\phi_s \sim \log 3 + 0.147219\ldots = 1.245831\ldots\ .$$

For the x of Section 8, our computed ϕ_s was $1.247731\ldots$.

6. Bounds on time and space

In this section we provide theoretical results on the time and space required by our procedure to compute $S(x)$ with an absolute error bounded by ρ. It is interesting to ask about the precise nature of the optimal procedure, as both x and ρ vary, but we will not answer that question fully here. Rather, we show that our procedure works well for delivering a moderately precise value of the sum.

We will distinguish between bit operations and arithmetic operations, with the latter referring to one of the four basic operations on a floating-point number or an integer. For simplicity we assume that all floating-point numbers have the same precision in their fractions. We emphasize that this precision can be set by a user of the algorithm, and for this reason, our arithmetic operations are not the same as the "flops" traditionally counted in numerical analysis.

Suppose our goal is to compute the prime harmonic sum with absolute error no more than ρ. It will suffice to use floating-point numbers with precision

$$\ell = \log(1/\rho) + O(\log x).$$

We need $O(\log x)$ guard bits for three reasons. First, we are accumulating sums with $O(x^{2/3})$ terms, which will incur round-off error. Second, in computing ϕ_o and ϕ_s, the largest terms are around $\log x$, whereas the sums are $\Theta(1)$. This cancellation must be rendered harmless. Finally, we deliver a result close to $\log\log x$, so relative error is not the same as absolute error.

Our complexity bounds are summarized in Table 1. We give growth rates only. Hence, for example, the space required for the main term is $O(x^{1/3}\ell)$.

Table 1. Cost to compute $S(x)$ with ℓ-bit arithmetic

	Time (bit operations)	Space (bits)
main term	$x^{1/3}\ell^2$	$x^{1/3}\ell$
S_2	$x^{2/3}\ell^2 \log\log x$	$x^{1/3}\ell$
ϕ_o	$x^{1/3}\ell^3$	ℓ^2
ϕ_s	$x^{2/3}\ell^2(\log x)(\log\log x)^{-1}$	$x^{1/3}\ell$

This analysis assumes that naive (quadratic time) arithmetic is used. We also assume $\log_2 x \le \ell = O(x^{2/3})$. The lower bound allows any integers to be represented exactly, and the upper bound reflects the limited accuracy of Euler-Maclaurin summation.

When $\ell \ge x^{2/3}$, we have $x^{1/3}\ell^3 \ge x\ell^2$. There is no point, therefore, in doing our analysis for ℓ larger than we have, since the naive algorithm will be better anyway.

For the most part, entries in Table 1 follow from bounds in the literature and estimates we have already given. We will, however, have to carefully analyze the floating-point computation for ϕ_o.

6.1. **Cost for main term.** By the prime number theorem, the main term requires us to invert and accumulate

$$\pi(x^{1/3}) \sim 3\frac{x^{1/3}}{\log x}$$

floating-point numbers. Along with $O(1)$ floating-point registers, we need space for the sieves.

6.2. **Cost for S_2.** Using the expansion of (2.4) as a double sum, we can compute S_2 using $O(x^{2/3}\log\log x)$ arithmetic operations, each having cost $O(\ell^2)$.

We can bound the space requirements by noting that the list of stored primes, and the lists Δ_k and A each require at most $O(x^{1/3})$ storage locations, of size $O(\ell)$. Since at most one block Δ_k is in storage at any time, we find that S_2 can be found using space $O(x^{1/3}\ell)$.

6.3. **Cost for ϕ_o.** We must do four things: (i) compute $\log 2$ to ℓ bits; (ii) compute γ to ℓ bits; (iii) compute $\le x^{1/3}$ logarithms of integers $y \le x$; (iv) evaluate (3.4) $\le x^{1/3}$ times.

For (i), if $1/2 \le f \le 1$, the Taylor expansion for $\log(1-z)$ will find ℓ bits of $\log f$, using

$$O(\ell^3) \tag{6.1}$$

bit operations; then $\log 2 = -\log(1/2)$. For (ii), the number of bit operations is given in [4] as

$$O(\ell^3 \log \ell). \tag{6.2}$$

For (iii) we use $\log(2^e \cdot f) = e\log 2 + \log f$ to reduce to logarithms on $[1/2, 1]$. By (6.1), the cost for all logarithms is

$$O(x^{1/3}\ell^3). \tag{6.3}$$

We now consider (iv). Let

$$\ell \le \frac{\pi e}{4}x^{2/3}. \tag{6.4}$$

If terms up to and including $y^{-\ell}$ are used, the truncation error is bounded by $2^{-\ell}$. (From (5.11) with $\nu = \ell$, and the enveloping property of Stirling's series [34, p. 253], we find that this bound holds provided that

$$\ell\left[\frac{2}{3}\log_2 x - \log_2 \ell + \log_2\left(\frac{\pi e}{4}\right)\right] + \frac{1}{2}\log_2 \ell + \frac{1}{2}\log_2(8\pi) - \frac{1}{12\log 2}\ell^{-1} \ge 0.$$

The condition (6.4) makes the expression in brackets positive, and the remaining terms make a positive contribution when $\ell \ge 1$.)

Since the terms in (3.4) ultimately alternate, we must also check that they decrease rapidly enough to prevent cancellation from reducing precision. The term ratio has absolute value

$$-\frac{c_{\nu+2}/y^{\nu+2}}{c_\nu/y^\nu} \sim \frac{\nu}{\pi^2}y^{-2} \le \frac{e}{4\pi}x^{-2/3}$$

(using $y \ge x^{2/3}$ and (6.4)). Therefore, when each number is aligned with the next one, there is an "overhang" of $\Theta(\log x)$ bits.

With the method given by Knuth and Buckholtz [17], we can compute the first ν Bernoulli numbers to precision ℓ using $O(\nu^2 \ell \log \ell + \nu \ell^2)$ bit operations [4]. (Furthermore, the space is $O(\nu\ell)$.) Taking $\nu = \ell$, this is

$$O(\ell^3 \log \ell). \tag{6.5}$$

Now we can consider the sums. There are at most $x^{1/3}$ series to evaluate, each with $O(\ell)$ terms. The total work for these sums is

$$O(x^{1/3}\ell^3). \tag{6.6}$$

If we note that $\log \ell \le \log x = O(x^{1/3})$, then combining (6.1)–(6.6) gives the bound of the table.

6.4. **Cost for ϕ_s.** Although they are interleaved in the algorithm, it will be convenient to imagine three separate tasks: finding special nodes, computing prefix sums, and sieving.

Recall that our data structure for range sums is based on d-ary trees, where $d \ge 2$. To simplify the analysis, we will assume that d is fixed throughout the computation; a good value for d will be chosen shortly.

By (5.7), there are $O(x^{2/3}/(\log x)^2)$ special nodes. To identify all the special nodes, we must drag $\pi(x^{1/3})$ pointers through an array of size $x^{1/3}$, each pointer decrement being associated with $O(1)$ arithmetic operations. By the prime number theorem, this uses $O(x^{2/3}(\log x)^{-1}\ell^2)$ bit operations.

We also have a prefix sum for each special node. The path from this node to the root has length $O(\log_d x^{1/3})$, and at each level, contributions from up to d sibling nodes must be accumulated. Therefore, we do about $d \log x^{1/3} (\log d)^{-1}$ arithmetic operations per special node.

To sieve each segment, we must clear about $\sum_{p \le x^{1/3}} \frac{x^{1/3}}{p}$ entries of A. For each p, we clear every p-th entry as usual, but only update nodes on the lowest $\log_d p$ levels. (This is the level on which we expect to hit every node.) Before going on to the next p, we update the nodes further up the tree. Since about $\log_d x^{1/3} - \log_d p$ levels remain, we need to recompute $x^{1/3}/p$ entries. This is easily done by processing array elements from right to left.

If this strategy is adopted, the number of arithmetic operations used for finding special nodes and sieving is within a constant factor of

$$\begin{aligned}\left(\frac{9}{2}\frac{x^{2/3}}{\log^2 x}\right)\frac{\log x^{1/3}}{\log d}d \quad &+ \quad x^{1/3}\sum_{p \le x^{1/3}}\left(\frac{x^{1/3}}{p}\frac{\log p}{\log d} + \frac{x^{1/3}}{p}\right) \\ &\sim \quad \frac{x^{2/3}}{\log d}\left(\frac{\log x}{3} + \frac{3}{2}\frac{d}{\log x}\right).\end{aligned}$$

The two terms in parentheses are equal when $d = (2/9)\log^2 x$, so for this choice the total number of arithmetic operations is

$$O\left(\frac{x^{2/3}\log x}{\log\log x}\right).$$

Multiplying this by ℓ^2 gives the number of bit operations in the table.

As for the space bound, inspection of the algorithm reveals that it uses $O(x^{1/3})$ storage locations, each with $O(\ell)$ bits.

6.5. **Remarks on the cost bounds.** In computing ϕ_o when $\ell = o(x^{1/3})$, it will suffice to use $O(\ell/(\log x))$ terms of (3.4).

Our recurrence for $\phi(x, a)$ used the simple termination rule given in Section 3.4. By stopping sooner (that is, allowing nodes with larger b to be ordinary), the asymptotic time for computing $\pi(x)$ can be improved [19, 8]. We did not use this idea, in part because the analog to (3.4) for arithmetic progressions mod k has coefficients that grow exponentially with the base k. (See [23, p. 141].)

When ℓ is very large, it will pay to use asymptotically efficient arithmetic algorithms. Brent [6] and Karatsuba [15] have analyzed their performance in computing logarithms and the Euler constant γ. The paper [4] gives a similar analysis for the Buhler-Crandall algorithm, which computes the first ℓ Bernoulli numbers using the FFT. If we incorporate these improvements, the number of bit operations shrinks to

$$x^{1/3}\ell^2 \log \ell \log\log \ell + x^{2/3}\ell \log x \log \ell \frac{\log\log \ell}{\log\log x}.$$

(The first term is for ϕ_o, and the second for everything else.) There is still a quadratic dependence on ℓ, because the first ℓ Benoulli numbers, however computed, consume $\ell^{2+o(1)}$ bits of storage.

7. Targeted sums

In this section, we consider solving $\sum_{p \le x} 1/p = y$ for a given y. Let us agree that the solution is the first x, necessarily prime, for which the sum is $\ge y$. Since the sum is monotonic in x, we could find the solution using a binary search. This would, however, involve multiple evaluations of the sum, an expense we would like to avoid.

We can get a rough idea of the crossing point using the theorem of Mertens. Assuming the Riemann hypothesis, Schoenfeld [30] made this theorem explicit:

$$\left| \sum_{p \le x} 1/p - \log\log x - B \right| < \frac{3\log x + 4}{8\pi\sqrt{x}}, \tag{7.1}$$

for $x \ge 13.5$. Let $L(x)$ and $U(x)$ be the lower and upper bounds on the prime harmonic sum that are implied by (7.1), and define numbers x^- and x^+ by the equations

$$U(x^-) = y, \qquad L(x^+) = y.$$

Since U and L are smooth increasing functions, we know that the solution x satisfies $x^- \le x \le x^+$.

We now observe that the Schoenfeld interval (x^-, x^+) has length $O(x^{1/2}\log^2 x)$. To prove this, denote the right-hand side of (7.1) by $\eta(x)$, so that $U - L = 2\eta$. For all sufficiently large y,

$$\begin{aligned} U(2x^-) &= U(x^+) + \log(1 + \frac{\log 2}{\log x^-}) + \eta(2x^-) - \eta(x^-) - 2\eta(x^+) \\ &\ge U(x^+) + \log(1 + \frac{\log 2}{\log x^-}) + \eta(2x^-) - 3\eta(x^-), \end{aligned}$$

since η ultimately decreases. Eventually, then, $U(2x^-) \ge U(x^+)$, so $x^+ \le 2x^-$, putting the three numbers x^-, x, x^+ within constant factors of each other. Define a fourth number x^* by the relation $2\eta(x^+)/(x^+ - x^*) = U'(x^+)$. (This is basically

the first step of Newton's method for $U^{-1}(y)$.) Since U'' is ultimately negative, we have

$$x^+ - x^- \le x^+ - x^* = \frac{2\eta(x^+)}{U'(x^+)} \sim \frac{3\log(x^+)}{4\pi\sqrt{x^+}} \cdot x^+ \log(x^+) = O(x^{1/2}\log^2 x).$$

This suggests the following method to solve $\sum_{p\le x} 1/p = y$. Evaluate $S(x)$ inside the Schoenfeld interval (x^-, x^+), say at the center. Depending on whether the result is less than or greater than y, add or subtract further values of $1/p$ until y is reached.

To analyze this algorithm, we must first ascertain the precision it requires. To distinguish $S(x)$ from the next value of the sum (i.e. $S(x')$ where x' is the next prime after x), we will need the absolute error $\rho < 1/x$. By the Mertens theorem, it will be sufficient to deliver $\sim \log_2 x$ bits of precision in the result. As explained in Section 6, we will need $O(\log x)$ guard bits, so the fractions in our floating-point numbers will have length $\ell = O(\log x)$.

Assuming RH, our algorithm solves $S(x) = y$ using $x^{2/3}(\log x)^{O(1)}$ bit operations. By the results of Section 6, this is certainly true for the initial sum. We will also do prime tests on $O(x^{1/2}(\log x)^2)$ numbers near x, and by the result of [2], this can be done with $x^{1/2}(\log x)^{O(1)}$ bit operations. We emphasize that RH was used only to bound the running time; the result is unconditionally correct.

As a variation on this idea, we can use a sieve to identify the primes in the Schoenfeld interval; see [12, 31] for methods to do this efficiently while staying within our space bounds.

Additionally, we could obtain a rigorous bound on the run time of our algorithm by using an unconditional version of (7.1). For example, Dusart [9] has shown that that for $x > 10372$,

$$\left| \sum_{p\le x} 1/p - \log\log x - B \right| < \frac{1}{10\log^2 x} + \frac{4}{15\log^3 x}.$$

This gives rise to an interval of width about $x/(5\log x)$. This is large enough that using a binary search (or possibly an interpolation search or a Newton iteration) would be more efficient.

8. Computational experience

For integers $n \ge 1$, let x_n denote the first x for which $\sum_{p\le x} 1/p \ge n$. The first three values of this sequence are $5, 277, 5195977$. We tested our algorithms by finding

$$x_4 = 1801241230056600523.$$

Computing this was an open problem of Neil Sloane, and in this section we explain how it was done. We note that x_5 is about 4.2×10^{49}, so its precise value may remain unknown for all eternity.

Applying Schoenfeld's estimate, we have

$$1.8012409393\ldots \times 10^{18} \le x_4 \le 1.8012415234\ldots \times 10^{18}. \tag{8.1}$$

This interval contains the perfect cube

$$x = 1801241484456448000 = 1216720^3.$$

Accordingly, we computed the prime harmonic sum to high precision for this x. We note that the width of the interval is about 5.84×10^{11}, which is close to the first order approximation $(3/4\pi)\sqrt{x}(\log x)^2 \doteq 5.659\ldots \times 10^{11}$.

Next, we used the segmented sieve of Eratosthenes to identify all primes $\leq x^{1/2} \doteq 1.34 \times 10^9$. Halved differences between odd primes then fit in a table of single-byte (8 bit) numbers [5].

Using our list of prime gaps, we sieved segments of the Schoenfeld interval. After experimenting with various sieve implementations, we decided to use 8-bit bytes to store residue classes mod 30 (since $\varphi(30) = 8$), and work with segments of size 9.6×10^8, which is a multiple of 30. This was fast enough for our purposes, so we did not explore any of the more exotic cache-compatible sieve algorithms [11, 27].

We covered the Schoenfeld interval with 635 such segments (a bit more than strictly necessary). For each segment, we recorded the sum of $1/p$ for the primes therein. These values allowed us, with a linear search, to identify which segment contained the solution. Resieving it, we discovered that

$$\sum_{p \leq 18012412300566004 67} \frac{1}{p} = 3.99999\ 99999\ 99999\ 99966,$$

and

$$\sum_{p \leq 18012412300566005 23} \frac{1}{p} = 4.00000\ 00000\ 00000\ 00021.$$

Furthermore, we found no primes between the upper limits of these sums.

All programs were written in C++, except for some final "cleanup" code which used Maple. For floating-point computation, we relied on a method advocated by Dekker [7], for simulating quadruple precision using a pair of double precision numbers. With IEEE standard arithmetic, this gives about 30 digits of precision. We used the implementation in Shoup's package NTL [32], which (on the machines we used), can do quad float operations in about 10^{-7} seconds. C++ also has a 64-bit integer type (long long), which allowed all integers appearing in our algorithms to be stored exactly.

When computing the sums ϕ_o and ϕ_s, we accumulated positive and negative contributions separately. Visual inspection at the end of the computation allowed us to ascertain that the effects of subtractive cancellation were modest.

In processing the Schoenfeld interval, we were able to avoid floating-point computations with the following trick. Suppose a segment begins at the location o (for "offset"). Then if $p = o + \delta_i$ denotes the i-th prime in the segment Σ, we have, by Taylor's theorem,

$$\frac{1}{p} = \frac{1}{o} - \frac{\delta_i}{o^2} + \cdots . \tag{8.2}$$

By accumulating c (the number of primes) and s (the total δ_i) for a segment, we can compute the approximation

$$\sum_{p \in \Sigma} \frac{1}{p} \doteq \frac{c}{o} - \frac{s}{o^2}.$$

Furthermore, since (8.2) has alternating signs, we can easily (and rigorously!) bound the error in this approximation by estimating the next term.

The actual prime sum was done with four separate programs: one each for the main term, the sum over ordinary nodes, the sum over special nodes, and the sum S_2. Since the last two were the most time-consuming, we devoted a desktop workstation to each. These computations took about 6.5 days and 4 days, respectively. The code for sieving the Schoenfeld interval took about 16 hours, which is about 90 seconds per block.

9. Concrete error analysis

In this section we give various arguments, stopping just short of rigorous proof, that the computational results in the last section are correct. The most difficult computation is the sum over special nodes, so we will concentrate on that.

One time-honored quick and dirty trick is to run the same computation with increasing precision. Our code was written to make this easy to do, by changing one type definition. For $x = 10^{15}$, we computed ϕ_s to be

$$\begin{array}{ll} 1.24862902556037133904243183442 & \text{(quad float)}, \\ 1.248629025560361 & \text{(double)}, \\ 1.248066 & \text{(float)}. \end{array}$$

Smaller values of x were tried, using other arithmetics (including 40D Maple, and NTL's 45D RR data type), with similar results. Apparently, finite precision causes a loss of 2-3 decimal digits. If we conservatively assume that this loss is doubled for our target $x \doteq 1.8 \times 10^{18}$, our code is accurate enough, since $30 - 6 = 24$, and we only need 20 figure accuracy.

We next give a heuristic analysis, following Hamming [14]. Dividing the computation time by 10^{-7} (the time for one quad float operation), we see that we used 6×10^{12} quad float operations. We would expect the effects of round-off to grow like a random walk, i.e. about $\sqrt{n}$ units in the last place for n operations. This would leave us with an error around 10^{-24}, again well within our requirements.

We now attempt a more precise worst-case analysis. We will do this in three steps.

(1) Ascertain the error in C_b and values of ϕ.
(2) Bound the total error in the sum incurred by replacing each ϕ/m by its approximation $[\phi/m]$.
(3) Estimate the round-off error incurred in adding up these approximations.

For the quad float algorithms used in NTL, estimates on relative error can be found in the literature. In particular,

$$[\text{ computed } a \pm b\ ] = (a \pm b)(1 + \delta), \qquad |\delta| \le 2^{-105} := \epsilon_{\pm},$$

(as claimed in Section 3.3.2 of [24]), and for $b \neq 0$,

$$[\text{ computed } a/b\ ] = (a/b)(1 + \delta), \qquad |\delta| \le 3 \times 2^{-104} := \epsilon_{\div}$$

(take $p = 53$, $h = 0$ in the first column of Table 1 in [25]). The second actually uses a first-order Taylor approximation, so the arguments below will not be 100% rigorous.

For reference purposes we note that

$$\begin{aligned} x &= 1801241484456448000, \\ x^{2/3} &= 1480407558400 \le 1.5 \times 10^{12}, \\ x^{1/3} &= 1216720 \le 1.25 \times 10^{6}. \end{aligned}$$

9.1. **Error in C_b and ϕ.** Each ϕ is just a partial sum of the terms making up some C_b, so it is enough to estimate the error in the C_b's.

If we replace each reciprocal $1/t$ by its floating-point approximation, no value of C_b can change by more than

$$\epsilon_{\div} \sum_{t \le x^{2/3}} 1/t \le 5 \times 10^{-30}. \tag{9.1}$$

Each approximate reciprocal $[1/t]$ then participates in two more operations per node (one add, one subtract).

Consider the first segment Δ_0. The sum of $1/t$ for all odd $t \le x^{1/3}$ is $14.646\ldots$. Hence, for this segment we may pretend that the values of C_b are accumulated in fixed-point registers of the shape

$$\underbrace{\text{XXXX}}_{4\text{ bits}}.\underbrace{\text{YYYYYY}\ldots\text{YYYYYYY}}_{102\text{ bits}}$$

Reasoning this way will give an upper bound on the round-off error. The total number of operations is bounded by $x^{1/3}$ (half of the numbers, each used twice). Hence the round-off error is at most

$$x^{1/3} \cdot 16\epsilon_{\pm} \le 5 \times 10^{-25}. \tag{9.2}$$

(Experiments indicate it is much smaller. For example, if we sieve out all primes $\le x^{1/3}$, the result, printed to 30 places, is $1 + 3 \cdot 10^{-29}$.)

Now, consider a segment Δ_k with $k \ge 1$. By (3.1),

$$\sum_{t \in \Delta_k} \frac{1}{t} \le \frac{x^{1/3}}{2} \cdot \frac{1}{kx^{1/3}} = \frac{1}{2k}.$$

Reasoning similarly, the total round-off error for this segment's contribution to the C_b's will be at most

$$\frac{x^{1/3}\epsilon_{\pm}}{2k} \le \frac{3.1 \times 10^{-26}}{k}.$$

If we sum over $1 \le k \le x^{1/3} = n$, and replace H_n by $\log n + \gamma$, we get at most

$$5 \times 10^{-25}. \tag{9.3}$$

Adding (9.1)–(9.3), we estimate the error in any computed C_b to be at most 10^{-24}. Clearly the same holds for any computed ϕ. Call this number E.

9.2. **Error from approximating ϕ/m.** We will first need to estimate m from below. For any m found by sieving segment k, we have

$$m \ge \frac{x^{2/3}}{(k+1)^2}.$$

(Proof: From (5.16), $q > x^{2/3}/((k+1)m')$, but $m' \le x^{1/3}$, so $q > x^{1/3}/(k+1)$. Also, $q \le x^{1/3}$, so using (5.16) again, $m' > x^{2/3}/((k+1)q) \ge x^{1/3}/(k+1)$. Apply these two bounds to $m = m'q$.) We also know, from the definition of a special node, that

$$m \ge x^{1/3}.$$

The first bound is better when $k + 1 \le x^{1/6}$, and the second is better after that. Let us call k small if $k \le x^{1/6}$, and large otherwise.

Let there be ν_k nodes in segment k. We counted the nodes in each segment, and from these counts we found

$$\sum_{k \text{ small}} (k+1)^2 \nu_k = 3.4 \times 10^{14}$$

and

$$\sum_{k \text{ large}} \nu_k = 6.0 \times 10^7.$$

These will be useful later.

There are two sources of error in computed sums of ϕ/m: errors incurred by division, and errors arising from replacing an exact ϕ by its computed value. Since $\phi \le 16$, we can estimate the total contribution of the first to be

$$16\epsilon_{\div} \left[x^{-2/3} \sum_{k \text{ small}} (k+1)^2 \nu_k + x^{-1/3} \sum_{k \text{ large}} \nu_k \right] \le 7 \times 10^{-28}.$$

(This could be improved by choosing a better small/large breakpoint, a fine point we will ignore.) For the second, we have the estimate

$$E \left[x^{-2/3} \sum_{k \text{ small}} (k+1)^2 \nu_k + x^{-1/3} \sum_{k \text{ large}} \nu_k \right] \le 3.8 \times 10^{-22}.$$

Therefore, the total error incurred in replacing the ϕ/m's by their computed approximations is bounded by

$$3.9 \times 10^{-22}. \tag{9.4}$$

9.3. **Accumulation of final sum.** Since $\phi/m \le 1$, the total round-off error is no more than

$$\epsilon_{\pm} \cdot [\text{ number of special nodes }].$$

We used $\le 4.44 \times 10^9$ special nodes. Hence, the round-off error incurred in adding up the approximate ϕ/m's is at most

$$1.1 \times 10^{-22}. \tag{9.5}$$

Combining (9.4) and (9.5), we estimate the error in the final sum to be at most 5×10^{-22}. Of course, this is a worst-case estimate, so the true error is likely smaller.

9.4. **Other sums.** The other sums involve less computation, so we limit ourselves to a few remarks about them. To evaluate an ordinary node, we used (3.4) out to and including the y^{-2} term. With $\nu = 4$ and $c_\nu = 1/15$, our bound (5.10) on the truncation error is

$$\frac{16}{15} x^{-8/3} [1/4 + x^{-1/3}] < 10^{-49}.$$

Logarithms were evaluated by argument reduction followed by numerical solution, using Newton's method, of the transcendental equation $e^z = y$. Here, exponentials were evaluated by a Padé approximation. For the precision required, each logarithm needed a fixed number of arithmetic operations.

References

1. M. Abramowitz and I. Stegun. Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables. Dover Publications, 9th printing, 1972. MR1225604 (94b:00012)
2. M. Agrawal, N. Kayal, and N. Saxena. PRIMES is in P. *Ann. of Math.* (2)160:781-793, 2004. MR2123939 (2006a:11170)
3. G. E. Andrews, R. Askey, and R. Roy. Special Functions. Cambridge Univ. Press, 1999. MR1688958 (2000g:33001)
4. E. Bach. The complexity of number-theoretic constants, *Inform. Proc. Lett.*, 62:145-152, 1997. MR1453698 (98g:11148)
5. R. P. Brent. The first occurrence of large gaps between successive primes. *Math. Comp.*, 27(124):959–963, 1973. MR0330021 (48:8360)
6. R. P. Brent. Fast multiple-precision evaluation of elementary functions. *J. Assoc. Comput. Mach.*, 23(2):242–251, 1976. MR0395314 (52:16111)
7. T. J. Dekker. A floating-point technique for extending the available precision. *Numer. Math.*, 18, 224-242, 1971. MR0299007 (45:8056)
8. M. Deléglise and J. Rivat. Computing $\pi(x)$: The Meissel, Lehmer, Lagarias, Miller, Odlyzko method. *Math. Comp.*, 65(213):235–245, 1996. MR1322888 (96d:11139)
9. P. Dusart. Sharper bounds for ψ, θ, π, p_k. *Rapport de recherche #1998-06, Laboratoire d'Arithmétique de Calcul Formel et d'Optimisation*, Univ. Limoges, 1998.
10. M. Fredman. The complexity of maintaining an array and computing its partial sums. *J. Assoc. Comput. Mach.*, 29(1):250–260, 1982. MR662621 (83i:68068)
11. W. F. Galway. Robert Bennion's "Hopping Sieve." In *Algorithmic Number Theory (Portland, 1998)*, volume 1423 of *Lecture Notes in Comput. Sci.*, pages 169-178. Springer, Berlin, 2000. MR1726069 (2000m:11124)
12. 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. MR1850613 (2002g:11176)
13. E. Grosswald. Arithmetical functions with periodic zeros. *Acta Arith.*, 28(1):1–21, 1975/76. MR0404172 (53:7975)
14. R. W. Hamming. Numerical Methods for Scientists and Engineers. Dover, 1986. MR951632 (89h:65002)
15. E. A. Karatsuba, Fast evaluation of transcendental functions. *Prob. Inform. Trans.* 27(4):339-360, 1991. MR1156939 (93c:65027)
16. D. E. Knuth. The Art of Computer Programming: Volume 1, Fundamental Algorithms, 3rd edition, Addison-Wesley, 1997. MR0378456 (51:14624)
17. D. E. Knuth and T. J. Buckholtz, Computation of tangent, Euler, and Bernoulli numbers. *Math. Comp.* 21:663-688, 1967. MR0221735 (36:4787)
18. D. Klyve. *Explicit Bounds on Twin Primes and Brun's Constant*. Dissertation, Dartmouth College, 2007.
19. J. C. Lagarias, V. S. Miller, and A. M. Odlyzko. Computing $\pi(x)$: The Meissel-Lehmer method. *Math. Comp.*, 44(170):537–560, 1985. MR777285 (86h:11111)
20. J. C. Lagarias and A. M. Odlyzko. Computing $\pi(x)$: An analytic method. *J. Algorithms*, 8(2):173–191, 1987. MR890871 (88k:11095)
21. E. Landau. *Handbuch der Lehre von der Verteilung der Primzahlen. 2 Bände*. Chelsea Publishing Co., New York, 1953. 2d ed, With an appendix by Paul T. Bateman. MR0068565 (16:904d)
22. D. H. Lehmer. The exact number of primes less than a given limit. *Illinois J. Math.* 3:381–388, 1959. MR0106883 (21:5613)
23. D. H. Lehmer. Euler constants for arithmetical progressions. *Acta Arithmetica*, 27:125–142, 1975. MR0369233 (51:5468)
24. X. S. Li, J. W. Demmel, D. H. Bailey, G. Henry, Y. Hida, J. Iskandar, W. Kahan, S. Y. Kang, A. Kapur, M. C. Martin, B. J. Thompson, T. Tung, and D. J. Yoo. Design, implementation, and testing of extended and mixed precision BLAS. *ACM Trans. Math. Soft.*, 28:152–205, 2002.
25. S. Linnainmaa. Software for doubled-precision floating-point computations. *ACM Trans. Math. Soft.* 7:272–283, 1981. MR630437 (82h:68041)

26. E. D. F. Meissel. Ueber die Bestimmung der Primzahlenmenge innerhalb gegebener Grenzen. *Math. Ann.*, 2:636–642, 1870. MR1509683
27. T. Oliveira y Silva. Fast implementation of the segmented sieve of Eratosthenes. Manuscript, 2005.
28. K. Prachar. Über die kleinste quadratfreie Zahl einer arithmetischen Reihe. *Monat. Math.* 62:173-176, 1958. MR0092806 (19:1160g)
29. D. M. Priest, Algorithms for arbitrary precision floating point arithmetic. *Proc. 10th IEEE Symp. Computer Arithmetic*, IEEE Press, 1991.
30. L. Schoenfeld. Sharper bounds for the Chebyshev functions $\theta(x)$ and $\psi(x)$. II. *Math. Comp.*, 30(134):337–360, 1976. MR0457374 (56:15581b)
31. J. P. Sorenson. The pseudosquares prime sieve. In Florian Hess, Sebastian Pauli, and Michael Pohst, editors, *Proceedings of the 7th International Symposium on Algorithmic Number Theory (ANTS-VII)*, pages 193–207, Berlin, Germany, July 2006. Springer. Lecture Notes in Comput. Sci., 4076, ISBN 3-540-36075-1. MR2282925 (2007m:11168)
32. V. Shoup. NTL: A library for doing number theory. Software and documentation available on the author's home page, CS Dept., New York University.
33. D. V. Widder, The Laplace Transform. Princeton Univ. Press, 1941. MR0005923 (3:232d)
34. E. T. Whittaker and G. N. Watson. A Course of Modern Analysis. Cambridge Univ. Press, 1927. MR1424469 (97k:01072)

Computer Sciences Department, University of Wisconsin-Madison, 1210 W. Dayton Street, Madison, Wisconsin 53706
E-mail address: bach@cs.wisc.edu

Department of Mathematics, Carthage College, 2001 Alford Drive, Kenosha, Wisconsin 53140
E-mail address: dklyve@carthage.edu

Computer Science and Software Engineering, Butler University, Indianapolis, Indiana 46208
E-mail address: sorenson@butler.edu
URL: http://www.butler.edu/~sorenson