MATHEMATICS OF COMPUTATION
VOLUME 44, NUMBER 170
APRIL 1985, PAGES 537-560

Computing $\pi(x)$: The Meissel-Lehmer Method

By J. C. Lagarias, V. S. Miller and A. M. Odlyzko

Abstract. E. D. F. Meissel, a German astronomer, found in the 1870's a method for computing individual values of $\pi(x)$, the counting function for the number of primes $\leqslant x$. His method was based on recurrences for partial sieving functions, and he used it to compute $\pi(10^9)$. D. H. Lehmer simplified and extended Meissel's method. We present further refinements of the Meissel-Lehmer method which incorporate some new sieving techniques. We give an asymptotic running time analysis of the resulting algorithm, showing that for every $\varepsilon > 0$ it computes $\pi(x)$ using at most $O(x^{2/3+\varepsilon})$ arithmetic operations and using at most $O(x^{1/3+\varepsilon})$ storage locations on a Random Access Machine (RAM) using words of length $[\log_2 x] + 1$ bits. The algorithm can be further speeded up using parallel processors. We show that there is an algorithm which, when given M RAM parallel processors, computes $\pi(x)$ in time at most $O(M^{-1}x^{2/3+\varepsilon})$ using at most $O(x^{1/3+\varepsilon})$ storage locations on each parallel processor, provided $M \leqslant x^{1/3}$. A variant of the algorithm was implemented and used to compute $\pi(4 \times 10^{16})$.

1. Introduction. The problem of computing $\pi(x)$, the number of primes $p \leqslant x$, has been studied for a long time. The ancient Greeks had a method for locating all the prime numbers below a given bound, the sieve of Eratosthenes. Legendre (see [4]) was the first to suggest a method of calculating $\pi(x)$ without locating all the primes less than x. He observed that the principle of inclusion-exclusion implied that

$$(1.1)\quad \pi(x) - \pi(x^{1/2}) + 1 = [x] - \sum\left[\frac{x}{p_i}\right] + \sum\left[\frac{x}{p_i p_j}\right] - \sum\left[\frac{x}{p_i p_j p_k}\right] + \cdots,$$

where $[z]$ denotes the greatest integer $\leqslant z$ and the p_i run over all primes $\leqslant x^{1/2}$, and $p_i < p_j$ in the second sum, $p_i < p_j < p_k$ in the third sum, and so on. This formula is not of immediate utility in computing $\pi(x)$ because it involves computing about $6\pi^{-1}(1 - \log 2)x$ nonzero terms. Actual calculations of $\pi(x)$ by Gauss and others were based on factor tables made using sieve methods. The first efficient algorithm for computing $\pi(x)$ which does not involve locating all the primes below x is ascribable to the astronomer E. D. F. Meissel ([11]–[14]). His method is economical of space, and can be viewed as reducing the number of terms in "Legendre's sum" (1.1). Using his methods he calculated $\pi(10^8) = 5{,}761{,}455$ in 1871, and then after an enormous calculation announced in 1885 that $\pi(10^9)$ was 50,847,478. (His value of $\pi(10^9)$ was subsequently shown to be too small by 56, see [9].) Modifications of Meissel's method were subsequently suggested by several authors (see [4, pp. 429–434]) who did not, during the era of hand computation, attempt to explicitly

Received May 10, 1983; revised March 12, 1984.
1980 *Mathematics Subject Classification*. Primary 10A20; Secondary 10H15, 68C25.

compute larger values of $\pi(x)$. With the advent of digital computers, the problem of computing $\pi(x)$ was reconsidered by D. H. Lehmer [9], who extended and simplified Meissel's method. Lehmer used an IBM 701 to calculate $\pi(10^{10}) = 455{,}052{,}512$ (a value later shown [3] to be too large by 1). Following this, other authors ([3], [10]) calculated $\pi(x)$ for larger values of x using variants of the Meissel-Lehmer method, the current record being the calculation of $\pi(10^{13})$ by Bohman [3] in 1972. Bohman's value for $\pi(10^{13})$ turns out to be too small by 941, see Section 6.

In this paper we propose new algorithms of Meissel-Lehmer type for computing $\pi(x)$ and analyze their asymptotic computational complexity. Our interest in the asymptotic complexity of computing $\pi(x)$ was stimulated by a paper of H. S. Wilf [15], which cited $\pi(x)$ as an example of a function whose individual values are hard to compute. Indeed, in this connection, it appears that the existing methods for computing $\pi(x)$, which are based on variants of the Meissel-Lehmer method, have asymptotic running times at least $c_\varepsilon x^{1-\varepsilon}$ for any $\varepsilon > 0$ and some $c_\varepsilon > 0$. (It is hard to estimate the asymptotic computational complexity of Meissel's original method [11], since it is presented as a collection of rules to be applied according to the human calculator's judgment.) We analyze asymptotic running times using as a model of computation a *Random Access Machine* (*RAM*), which is a relatively realistic model of the addressible core storage area of a digital computer. (The RAM model is described in detail in Aho, Hopcroft and Ullman [1, Chapter 1].) An essential feature of the RAM model relevant to the complexity analysis is that on a RAM it is possible to jump between any two storage locations in a single operation. This feature is needed to quickly implement algorithms which use sieve methods. We prove the following result.

THEOREM A. *The Extended Meissel-Lehmer algorithm computes $\pi(x)$ on a Random Access Machine using at most $O(x^{2/3+\varepsilon})$ arithmetic operations and at most $O(x^{1/3+\varepsilon})$ storage locations, for any fixed $\varepsilon > 0$. All integers used in the course of the computation have at most $[\log_2 x] + 1$ bits in their binary expansions.*

By arithmetic operations we mean additions, subtractions, multiplications and divisions. Since any such operation on k bit integers takes $O(k^2)$ bit operations using the usual algorithms, the difference between counting arithmetic operations and bit operations is a multiplicative factor of $O((\log x)^2)$. Consequently, Theorem A implies that the Extended Meissel-Lehmer Method computes $\pi(x)$ in $O(x^{2/3+\varepsilon})$ bit operations using $O(x^{1/3+\varepsilon})$ bit locations of storage.

The constants implied by the O-symbols in Theorem A depend on $\varepsilon > 0$. The $O(x^\varepsilon)$ terms in Theorem A can be replaced by lower-order terms by a more detailed running time analysis than we give; for example, it can be shown that the algorithm described in Section 3 halts after at most $O(x^{2/3}(\log x)^4)$ bit operations using at most $O(x^{1/3}(\log x)^2)$ bit locations of space. At the end of Section 3 we explain why we are unable to improve the exponent of the running time bound below $2/3$ for an algorithm of Meissel-Lehmer type, even if more space locations are available. On the other hand, one can find variants of the Extended Meissel-Lehmer algorithm which use less space, provided a running time of more than $O(x^{2/3})$ is allowed.

In addition, one can further speed up the computation of $\pi(x)$ using parallel processing. We consider a model of parallel processing in which simultaneous transfers of information are allowed. We assume that we have M processors, which are arranged as the first M leaves of a complete balanced binary tree having 2^k leaves, where $2^{k-1} < M \leqslant 2^k$, and that the nonleaf nodes are occupied by switches. In a single time unit, a bit of information together with its destination (which will consist of $[\log_2 M]$ bits) can be transmitted over any of the edges in the tree. This model allows many messages to be transmitted simultaneously between different processors, provided the message paths do not intersect. We propose an algorithm called the *Extended Parallel Meissel-Lehmer Method*, and prove in Section 4 the following result.

THEOREM B. *For any M with $1 \leqslant M \leqslant x^{1/3}$ the Extended Parallel Meissel-Lehmer algorithm computes $\pi(x)$ using M Random Access Machine parallel processors using at most $O(M^{-1}x^{2/3+\varepsilon})$ arithmetic operations, and at most $O(x^{1/3+\varepsilon})$ storage locations for each parallel processor, i.e. $O(Mx^{1/3+\varepsilon})$ storage locations in all, for any $\varepsilon > 0$. All integers used in the course of the computation have length at most $[\log_2 x] + 1$ bits in their binary expansions.*

The constants implied by the O-symbols in Theorem B depend on $\varepsilon > 0$ and *not* on M in the range $1 \leqslant M \leqslant x^{1/3}$. The $O(x^{\varepsilon})$ terms in Theorem B can be replaced by lower-order terms like $O((\log x)^4)$ by a more detailed running time analysis of the algorithm described in the text.

Theorem B logically includes Theorem A. We prove Theorem A separately because a somewhat simplified algorithm description and running time analysis are possible in this case.

The algorithms described in Sections 3 and 4 were designed to facilitate their asymptotic running time analysis, and have inefficiences from the viewpoint of practical computation. In Section 5 we describe modifications of the Extended Meissel-Lehmer Method which improve its performance in practice. The second author implemented a version of the algorithm described in Section 5 on an IBM 3081, and used it to compute various values of $\pi(x)$, the largest being $x = 4 \times 10^{16}$. These results are described in Section 6.

Several authors [5], [9] have noted that algorithms of the Meissel-Lehmer type can be developed to compute the function $\pi(x; k, l)$ which counts the number of primes $p \leqslant x$ in the arithmetic progression $k \pmod l$, as well as summatory functions of other multiplicative integer-valued functions. The ideas underlying the Extended Meissel-Lehmer algorithm also carry over to these situations.

The first and third authors [7] have found an entirely different algorithm for computing $\pi(x)$ which runs in $O(x^{3/5+\varepsilon})$ time and uses $O(x^{\varepsilon})$ space, which can also be speeded up using parallel processors. It is based on ideas from analytic number theory related to the "explicit formulae" for $\pi(x)$. The algorithm of [7] seems considerably more difficult to program than the Extended Parallel Meissel-Lehmer Method, and the constants implied by the O-symbols in [7] are probably rather large. Consequently, despite its asymptotic superiority, the algorithm of [7] is very likely not competitive with the Extended Parallel Meissel-Lehmer Method for computing $\pi(x)$ for $x \leqslant 10^{17}$.

Some of the results of this paper are sketched in the announcement [6], which describes a simpler variant of the Extended Meissel-Lehmer algorithm which requires $O(x^{2/3+\varepsilon})$ storage space.

2. Algorithms of Meissel-Lehmer Type. We first describe the basic structure of algorithms of Meissel-Lehmer type, following the treatment of Lehmer [9]. Let $p_1, p_2, p_3, \ldots$ denote the primes $2, 3, 5, \ldots$ numbered in increasing order, and for $a \geqslant 1$ let

$$\phi(x, a) = |\{n \leqslant x : p | n \Rightarrow p > p_a\}|$$

denote the *partial sieve function* which counts numbers $\leqslant x$ with no prime factor less than or equal to p_a, and let

$$P_k(x, a) = \left|\left\{n \leqslant x : n = \prod_{j=1}^{k} p_{m_j}, m_j > a \text{ for } 1 \leqslant j \leqslant k\right\}\right|$$

denote the *kth partial sieve function*, which counts numbers $\leqslant x$ with exactly k prime factors, none smaller than or equal to p_a. We set $P_0(x, a) = 1$. We then have the identity

$$\phi(x, a) = P_0(a, x) + P_1(x, a) + P_2(x, a) + \cdots, \tag{2.1}$$

where the sum on the right has only finitely many nonzero terms. Now

$$P_1(x, a) = \pi(x) - a, \tag{2.2}$$

so that if one can compute $\phi(x, a)$, $P_2(x, a)$, $P_3(x, a)$, etc., one can obtain $\pi(x)$. Meissel-Lehmer methods split the computation of $\pi(x)$ into two parts, which are the computation of the $P_i(x, a)$ and that of $\phi(x, a)$.

The computation of the $P_i(x, a)$ is the simpler of the two parts. In the Extended Meissel-Lehmer Method, we will take $a = \pi(x^{1/3})$ in which case $P_3(x, a) = P_4(x, a) = \cdots = 0$, so we need only consider the computation of $P_2(x, a)$ here. The computation of $P_2(x, a)$ is based on the formula

$$\begin{aligned} P_2(x, a) &= |\{n : n \leqslant x, n = p_b p_c \text{ with } a < b \leqslant c\}| \\ &= \sum_{j=a+1}^{\pi(x^{1/2})} \left|\left\{n : n \leqslant x, n = p_j p_k \text{ with } j \leqslant k \leqslant \pi\left(\frac{x}{p_j}\right)\right\}\right| \\ &= \sum_{j=a+1}^{\pi(x^{1/2})} \left(\pi\left(\frac{x}{p_j}\right) - j + 1\right) = \binom{a}{2} - \binom{\pi(x^{1/2})}{2} + \sum_{j=a+1}^{\pi(x^{1/2})} \pi\left(\frac{x}{p_j}\right), \end{aligned} \tag{2.3}$$

which is valid whenever $a \leqslant \pi(x^{1/2})$. The sum on the right side of (2.3) is evaluated by completely sieving the interval $[1, [x/p_{a+1}]]$ to compute all the $\pi(x/p_j)$. For $a = \pi(x^{1/3})$ this interval is a subinterval of $[1, N]$, where $N = [x^{2/3}]$ and so the evaluation of $P_2(x, a)$ can be done in $O(x^{2/3+\varepsilon})$ arithmetic operations. There are more complicated identities for those $P_k(x, a)$ with $k \geqslant 3$ which are derived in a way similar to (2.3), see [9].

We now consider the computation of the term $\phi(x, a)$. Meissel-Lehmer methods accomplish this by repeated use of the recurrence

$$\phi(x, a) = \phi(x, a-1) - \phi\left(\frac{x}{p_a}, a-1\right). \tag{2.4}$$

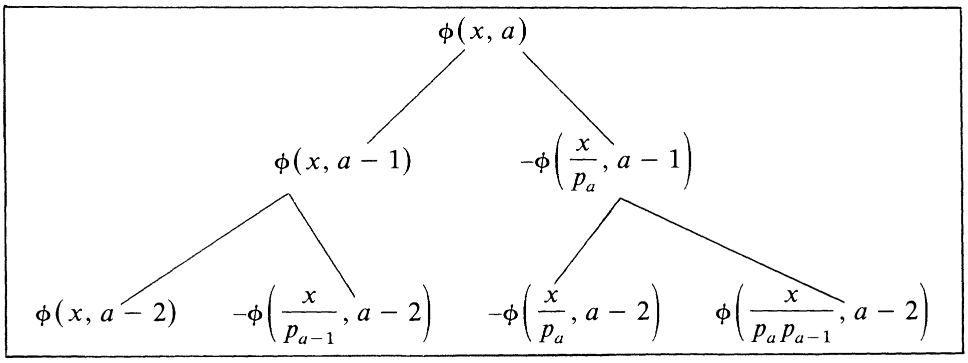
$\phi(x, a)$
$\phi(x, a-1)$
$-\phi\left(\frac{x}{p_a}, a-1\right)$
$\phi(x, a-2)$
$-\phi\left(\frac{x}{p_{a-1}}, a-2\right)$
$-\phi\left(\frac{x}{p_a}, a-2\right)$
$\phi\left(\frac{x}{p_a p_{a-1}}, a-2\right)$

FIGURE 1

Binary tree for the computation of $\phi(x, a)$.

One may think of the process of applying (2.4) repeatedly as creating a rooted binary tree (Figure 1) starting from the root node $\phi(x, a)$. Each application of the rule (2.4) splits a node into two leaves corresponding to the terms on the right side of (2.4) (See Figure 1). At all times in this process $\phi(x, a)$ is equal to the sum (with the signs indicated) of the terms associated with the leaves of the rooted tree. At some point a truncation rule is applied to terminate the growth of the tree, and the contributions of the leaves are added up to compute $\phi(x, a)$. Variants of the Meissel-Lehmer algorithm are obtained by using different truncation rules and different methods to evaluate the contributions of the leaves. We note that the nodes of the binary tree are uniquely identified by ordered pairs (n, b) where

$$(2.5) \qquad n = p_{a_1} \cdots p_{a_r} \quad \text{with } a \geqslant a_1 > \cdots > a_r \geqslant b+1.$$

The node (n, b) has the associated term $(-1)^r\phi(x/n, b)$. In practice it is not necessary to construct the tree explicitly; all that is needed is a way to evaluate the contribution of the leaves.

As an example, Lehmer [9] uses the following truncation rule.

TRUNCATION RULE L. *Do not split a node labelled $\pm\phi(x/n, b)$ if either of the following holds.*

(i) $x/n < p_b$.

(ii) $b = c(x)$, *where $c(x)$ is a very slowly growing function of x.*

Lehmer chose $c(x)$ equal to the constant 5 for his computation. He computed the contribution of the leaves using the formula

$$(2.6) \qquad \phi(y, b) = 1 \quad \text{if } y < p_b,$$

for leaves of type (i) and the formula

$$(2.7) \qquad \phi(y, c) = \left[\frac{y}{Q}\right]\phi(Q) + \phi\left(y - \left[\frac{y}{Q}\right]Q, c\right),$$

for leaves of type (ii), where $c = c(x)$, $Q = \prod_{i \leqslant c(x)} p_i$ and a precomputed table of $\{\phi(y, b): 1 \leqslant y \leqslant Q\}$ is available.

The defect of previous variants of the Meissel-Lehmer method that we have examined is that they have too many leaves. For example, in Lehmer's algorithm $a = \pi(x^{1/4})$, and the tree has nodes corresponding to $\phi(x/n, s-1)$ for all $n = p_s p_t p_u p_v$, where $6 \leqslant s < t < u < v \leqslant a$, and there are roughly $\frac{1}{24}a^4 = \Omega(x/(\log x)^4)$ of them.

The novel features of the Extended Meissel-Lehmer Method are the use of a new truncation rule and a new method for evaluating the contribution of the leaves to compute $\phi(x, a)$ where $a = \pi(x^{1/3})$. The truncation rule is as follows.

TRUNCATION RULE T. *Do not split a node labelled* $\pm\phi(x/n, b)$ *if either of the following holds.*

(i) $b = 0$ *and* $n \leqslant x^{1/3}$.

(ii) $n > x^{1/3}$.

We call leaves of type (i) *ordinary leaves* and leaves of type (ii) *special leaves*. The resulting binary tree does not have very many leaves, as we now show.

LEMMA 2.1. *The rooted binary tree resulting from applying Truncation Rule* T *to* $\phi(x, a)$ *with* $a = \pi(x^{1/3})$ *has at most* $x^{1/3}$ *ordinary leaves and at most* $x^{2/3}$ *special leaves.*

Proof. The first thing to observe is that no two leaves (n, b) have the same value of n. Indeed if (n, d) and (n, b) are nodes with $d \geqslant b$, then there is a descending path through the tree with nodes $(n, d-1), \ldots, (n, b+1)$, (n, b) so that (n, d) cannot be a leaf. It follows that there are at most $x^{1/3}$ ordinary leaves.

To bound the number of special leaves, we use the fact that a special leaf (n, b) has the father node $(n^*, b+1)$ where

$$n = n^* p_{b+1}. \tag{2.8}$$

Furthermore,

$$n > x^{1/3} \geqslant n^* \tag{2.9}$$

because $(n^*, b+1)$ is not a special leaf. Now (2.9) implies that there are at most $x^{1/3}$ choices for n^*, and there are at most $a = \pi(x^{1/3})$ choices for p_{b+1} in (2.8). Hence, there are at most $x^{2/3}$ choices for n in (2.8), and thus at most $x^{2/3}$ special leaves. □

The Extended Meissel-Lehmer Method computes the contributions of the leaves resulting from Truncation Rule T by a partial sieving process applied to the interval $[1, N]$ where $N = [x^{2/3}]$. The sieving is done on successive subintervals of length $x^{1/3}$ to reduce the storage requirements of the algorithm. A special data structure (the array $\{a(i, j)\}$ given by (3.9) below) is used to allow fast storage and retrieval of the intermediate partial sieving results as they are produced.

We now describe the Extended Meissel-Lehmer algorithm in detail and analyze its asymptotic running time.

3. The Extended Meissel-Lehmer Algorithm.

The Extended Meissel-Lehmer algorithm computes $\pi(x)$ using the formula (2.1) with $a = \pi(x^{1/3})$, so that

$$\pi(x) = a - P_2(x, a) + \phi(x, a). \tag{3.1}$$

The algorithm splits into two parts, one to compute $P_2(x, a)$, the other to compute $\phi(x, a)$. We first describe the algorithm to compute $P_2(x, a)$.

Algorithm P_2. This algorithm computes $P_2(x, \pi(x^{1/3}))$ using the formula

$$P_2(x, \pi(x^{1/3})) = \binom{\pi(x^{1/3})}{2} - \binom{\pi(x^{1/2})}{2} + \sum_{x^{1/3} < p \leqslant x^{1/2}} \pi\left(\frac{x}{p}\right). \tag{3.2}$$

The main difficulty is the computation of the sum on the right side of (3.2). To compute this sum we will sieve the interval $[1, x^{2/3}]$ and use the results to determine all the values of $\pi(x/p)$ in the sum. To keep down the space requirements of the algorithm we divide the interval $[1, x^{2/3}]$ into at most $x^{1/3} + 1$ blocks of length $N = [x^{1/3}]$, and sieve these blocks sequentially, keeping exactly one block in storage at a time. The jth block is $B_j = [(j-1)N + 1, jN]$, except that the last block is a possibly short block with largest element $[x^{2/3}]$.

In what follows we use the phrase to *completely sieve* an interval $[x, y]$ to mean sieving out from the interval all proper multiples of all primes $p \leqslant y^{1/2}$, so that the sieved interval contains exactly the primes in $[x, y]$.

In an initial precomputation, we completely sieve the first block and create an ordered list **P** of all primes $p \leqslant x^{1/3}$, for use in subsequent sieving. We also determine $\pi(x^{1/3})$.

We now process the blocks B_j sequentially. During the jth pass we are given as input $\pi((j-1)N)$ saved from the last pass. We sieve out from the block B_j all proper multiples of all primes from our list **P**, which completely sieves B_j because it contains only integers $\leqslant x^{2/3}$. We now use the sieved list and $\pi((j-1)N)$ to compute the list $\{\pi(y): (j-1)N + 1 \leqslant y \leqslant jN\}$. We determine $\pi(x^{1/2})$ when the appropriate block B_j for which $(j-1)N + 1 \leqslant x^{1/2} < jN + 1$ is being processed. Our next step is to locate all primes p with $x^{1/3} < p \leqslant x^{1/2}$ such that

$$(j-1)N + 1 \leqslant \frac{x}{p} < jN + 1.$$

This will occur if and only if p is a prime lying in the half-open interval

$$I_j = \left(\frac{x}{jN+1}, \frac{x}{(j-1)N+1}\right] \cap \left(x^{1/3}, x^{1/2}\right].$$

Our next step is to sieve out from the interval I_j all proper multiples of all primes $\leqslant x^{1/4}$, which completely sieves it because I_j contains no elements $> x^{1/2}$. Also note at this point that all the intervals I_j are disjoint. Now we have located all the primes $\{p: p \in I_j\}$. Next we compute the set $\{y = [x/p]: p \in I_j\}$, and evaluate the sum

$$\Delta_j = \sum_{p \in I_j} \pi\left(\frac{x}{p}\right)$$

in one pass through the list $\{\pi(y): (j-1)N + 1 \leqslant y \leqslant jN\}$. Then we add Δ_j to an accumulator

$$S_j = S_{j-1} + \Delta_j = \sum_{i=1}^{j} \Delta_i.$$

Finally, we empty storage of everything except the list **P** of primes $\leqslant x^{1/3}$, $\pi(x^{1/3})$, $\pi(jN)$, S_j and $\pi(x^{1/2})$ (if known), and proceed on to process block B_{j+1}.

At the end of this process the accumulator contains

$$S = \sum_{x^{1/3} < p \leqslant x^{1/2}} \pi\left(\frac{x}{p}\right)$$

and we also know $\pi(x^{1/3})$ and $\pi(x^{1/2})$. We then compute $P_2(x, \pi(x^{1/3}))$ using (3.2). □

LEMMA 3.1. *Algorithm P_2 computes $P_2(x, \pi(x^{1/3}))$ using at most $O(x^{2/3+\varepsilon})$ arithmetic operations and at most $O(x^{1/3+\varepsilon})$ storage locations on a RAM. All integers stored have at most* $[\log_2 x] + 1$ *bits in their binary expansions.*

Proof. It is clear that Algorithm P_2 correctly computes $P_2(x, \pi(x^{1/3}))$, and that the integers stored during the course of the computation are of length at most $[\log_2 x] + 1$ bits in binary.

To bound the running time, we observe that the main contribution to it is the sieving of the intervals B_j, which requires $O(x^{2/3+\varepsilon})$ arithmetic operations. The auxiliary sieving of the intervals I_j requires only $O(x^{7/12+\varepsilon})$ arithmetic operations, since all the intervals I_j are disjoint and lie in $[1, x^{1/2}]$, and each is sieved by primes up to $x^{1/4}$ only. It is easily checked that the remainder of the algorithm requires at most $O(x^{2/3+\varepsilon})$ arithmetic operations.

To bound the space requirements, we note that the lists **P** and $\{\pi(y)\colon (j-1)N + 1 \leqslant y \leqslant jN\}$ require at most $O(x^{1/3})$ storage locations. Also, at most one block B_j and one block I_j are in storage at any time, and each of these uses at most $O(x^{1/3+\varepsilon})$ storage locations. To see this for I_j, note that we must have $x/(jN+1) < x^{1/2}$ for I_j to be nonempty, so that $jN + 1 > x^{1/2}$. Hence I_j contains at most

$$\frac{x}{(j-1)N+1} - \frac{x}{jN+1} + 1 \leqslant \frac{Nx}{((j-1)N+1)^2} + 1 \leqslant 2x^{1/3}$$

integers. The remaining space requirements are asymptotically negligible. □

Remark. It is easy to extend the method of Algorithm P_2 to compute $P_2(x, \pi(x^{\alpha}))$ in $O(x^{1-\alpha+\varepsilon})$ arithmetic operations using $O(x^{(1-\alpha)/2+\varepsilon})$ storage locations. To do so one sieves the interval $[1, x^{1-\alpha}]$ broken up into blocks of length $x^{(1-\alpha)/2}$.

Now we describe an algorithm to compute $\phi(x, \pi(x^{1/3}))$.

Algorithm ϕ. This algorithm computes $\phi(x, \pi(x^{1/3}))$ using the formula derived from (2.4) using Truncation Rule T. This is

$$(3.3)\qquad \phi(x, \pi(x^{1/3})) = \sum_{\substack{(n,0)\text{ an}\\ \text{ordinary leaf}}} \mu(n)\phi\left(\frac{x}{n}, 0\right) + \sum_{\substack{(n,b)\text{ a}\\ \text{special leaf}}} \mu(n)\phi\left(\frac{x}{n}, b\right),$$

where $\mu(n)$ is the Möbius function, which is given by

$$(3.4)\qquad \mu(n) = \begin{cases} (-1)^{\omega(n)} & \text{if } n \text{ is square-free, and } \omega(n) \text{ is the number of prime factors of } n, \\ 0 & \text{otherwise.} \end{cases}$$

Here:

(1) $(n, 0)$ is an ordinary leaf exactly when n is square-free and $n \leqslant x^{1/3}$.

(2) (n, b) is a special leaf exactly when $n = p_{a_1} \cdots p_{a_r}$, where $\pi(x^{1/3}) \geqslant a_1 > a_2 > \cdots > a_r = b + 1$, and

$$n \geqslant x^{1/3} \geqslant n/p_{b+1}.$$

Let S_1 and S_2 denote the sums in (3.3) over the ordinary and special leaves, respectively.

As a first step Algorithm ϕ constructs an ordered list of all primes $p \leqslant x^{1/3}$, which is stored and used throughout the remainder of the algorithm.

The contribution S_1 of the ordinary leaves is easy to compute. We obtain a list of all square-free integers n in the interval $[1, [x^{1/3}]]$ along with $\mu(n)$ by a simple sieving procedure. We start with each cell containing a 1. We sieve by all primes $p < x$, multiplying each element sieved by -1. Then we sieve by squares of primes p^2 for all $p < x^{1/6}$, and put a zero in each cell thus sieved. After this sieving the content of cell n is just $\mu(n)$. Then

$$S_1 = \sum_{n=1}^{[x^{1/3}]} \mu(n)\phi\left(\frac{x}{n}, 0\right).$$

We compute $\phi(x/n, 0)$ using the formula

$$\phi\left(\frac{x}{n}, 0\right) = \left[\frac{x}{n}\right]$$

and add up the terms $\mu(n)\phi(x/n, 0)$ successively in an accumulator to compute S_1.

The computation of the contribution S_2 of the special leaves is the most complicated part of Algorithm ϕ. We first note that special leaves (n, b) have the form

$$n = n^* p_{b+1}, \tag{3.5}$$

where $(n^*, b+1)$ is the father node of (n, b) in the binary tree, and that

$$n^* = p_{a_1} \cdots p_{a_j} \quad \text{with } a \geqslant a_1 > \cdots > a_j > b+1 \tag{3.6}$$

and necessarily

$$n > x^{1/3} \geqslant n^*.$$

In an initial preprocessing step, we prepare an ordered array

$$\mathbf{F} = \{(m, f(m), \mu(m)) \colon 1 \leqslant m \leqslant [x^{1/3}]\}, \tag{3.7}$$

where $f(m)$ is the least prime factor of m and $\mu(m)$ is given by (3.4). The array $\mathbf{F}$ is constructed by first sieving the interval $[1, x^{1/3}]$ by all primes less than $x^{1/3}$ in increasing order, setting $f(m) = p$ the first time m is sieved out by some prime p, and computing the values of $\mu(m)$ by the sieving process described earlier.

The main computation of the contribution S_2 is performed via a partial sieving of the interval $[1, x^{2/3}]$ by all primes $p \leqslant x^{1/3}$. To reduce the space requirements of the algorithm, this interval is divided up into at most $N + 2$ blocks of length $N = [x^{1/3}]$, which are sieved in order one at a time. The kth block B_k is given by

$$B_k = [(k-1)N + 1, kN]$$

and the last block may be short.

We now describe the sieving of the kth block B_k. The algorithm has available from the sieving of the previous blocks the table

$$\{\phi((k-1)N, j) \colon 1 \leqslant j \leqslant a\}. \tag{3.8}$$

We sieve the block successively by the primes $p_1, p_2, \ldots, p_a$, where $a = \pi(x^{1/3})$. To keep track of the intermediate results after sieving by all primes up to a given prime p_b, we employ an array

$$\left\{a(i, j) \colon 0 \leqslant i \leqslant [\log_2 N] \text{ and } 1 \leqslant j \leqslant \left[\frac{N}{2^i}\right] + 1\right\}, \tag{3.9}$$

where $a(i, j)$ counts the number of currently unsieved elements in the subinterval

$$I_{i,j} = [(k-1)N + (j-1)2^i + 1, (k-1)N + j2^i],$$

where the subintervals corresponding to $j = [N/2^i] + 1$ may be shorter. In particular,

$$a(0, j) = \begin{cases} 0 & \text{if } (k-1)N + j \text{ is currently sieved out,} \\ 1 & \text{if } (k-1)N + j \text{ is currently unsieved.} \end{cases}$$

To update this array, when we are sieving by the prime p_k and we have just sieved out the element $(k-1)N + l$ for the first time, we decrement by 1 the array elements $a(0, l)$, $a(1, [(l+1)/2])$, $a(2, [(l+3)/4]), \ldots$ whose associated intervals contain $(k-1)N + l$.

Now suppose we have finished sieving the block B_k by the prime p_b, having already sieved B_k by $p_1, \ldots, p_{b-1}$. Given an integer $y = (k-1)N + l$ in the interval B_k, we can easily compute the value $\phi(y, b)$ using the formula

$$\phi(y, b) = \phi((k-1)N, b) + \sum_{r=1}^{m} a\left(e_r, 1 + \sum_{s<r} 2^{e_s - e_r}\right), \tag{3.10}$$

where the $e_1 > e_2 > \cdots > e_m \geqslant 0$ are the exponents in the binary expansion of l, i.e.,

$$l = \sum_{r=1}^{m} 2^{e_r},$$

and $\phi((k-1)N, b)$ is available in the table (3.8). We now calculate the contribution to the sum S_2 of all special leaves (n, b) for which

$$(k-1)N + 1 \leqslant \frac{x}{n} < kN + 1, \tag{3.11}$$

as follows. We have $n = n^* p_{b+1}$ using (3.5), and we note that (3.11) implies that n^* satisfies

$$\frac{x}{(kN+1)p_{b+1}} < n^* \leqslant \frac{x}{((k-1)N+1)p_{b+1}}. \tag{3.12}$$

Conversely, any $n^* \leqslant x^{1/3}$ which satisfies (3.6) corresponds to a unique special leaf $(n^* p_{b+1}, b)$. Consequently, we are looking for exactly those n^* which lie in the interval

$$J_{k,b} = \left(\frac{x}{(kN+1)p_{b+1}}, \frac{x}{((k-1)N+1)p_{b+1}} \right] \cap [1, N]. \tag{3.13}$$

We locate the least integer L and the largest integer U in the interval $J_{k,b}$ and then examine the array $\mathbf{F}$ of all m for which $L \leqslant m \leqslant U$. Those elements m in this interval with

$$f(m) > p_{b+1} \quad \text{and} \quad \mu(m) \neq 0$$

are exactly those n^* for which (n, b) is a special leaf with $n = n^* p_{b+1}$ satisfying (3.11). For each such m we compute

$$y = \left[\frac{x}{m p_{b+1}} \right] = (k-1)N + l,$$

where by (3.10), $1 \leqslant l \leqslant N$. We then calculate

$$\mu(n)\phi\left(\frac{x}{n}, b\right) = -\mu(m)\phi(y, b) \tag{3.14}$$

using (3.10), and add it to the accumulator computing S_2. In this way we calculate the contribution of all the special leaves (n, b) with $[x/n]$ in B_k. At this point we compute $\phi(kN, b)$ using (3.10), and use it to update the table (3.8). We now proceed to the prime p_{b+1}. After all the primes up to $x^{1/3}$ have been processed, the table (3.8) is completely updated and we proceed to block B_{k+1}.

After all the blocks B_k are sieved, the accumulator contains the value of S_2.

Algorithm ϕ adds S_1 and S_2 and halts. □

LEMMA 3.2. *Algorithm ϕ computes $\phi(x, \pi(x^{1/3}))$ using at most $O(x^{2/3+\epsilon})$ arithmetic operations and using at most $O(x^{1/3+\epsilon})$ storage locations. All integers stored during the computation are of length at most* $[\log_2 x] + 1$ *bits.*

Proof. The correctness of Algorithm ϕ is easy to check, as is the length of integers stored during the computation.

To obtain the time and space bounds, we observe first that the evaluation of the sum S_1 over the ordinary leaves requires at most $O(x^{1/3+\epsilon})$ arithmetic operations and uses at most $O(x^{1/3+\epsilon})$ storage locations. Also the evaluation of the sum S_2 over the special leaves uses at most $O(x^{1/3+\epsilon})$ storage locations, since at most one block B_k is in storage at any time, along with the table (3.8), the array (3.9), the list of primes $p \leqslant x^{1/3}$ and the list **F** of (3.7). To obtain the time bound $O(x^{2/3+\epsilon})$ we examine the procedure of sieving by blocks. The sieving operations for a given block B_k require at most $O(x^{1/3}\log\log x)$ arithmetic operations, and the number of updating operations of the array $\{a(i, j)\}$ in (3.9) during each block B_k takes $O(x^{1/3}\log x)$ arithmetic operations. This adds up to $O(x^{2/3+\epsilon})$ arithmetic operations when summed over all blocks. Also, it takes $O(x^{\epsilon})$ bit operations to compute each value $\mu(n)\phi(x/n, b)$ using (3.10) and (3.14) and since there are at most $x^{2/3}$ special leaves by Lemma 2.1, this takes at most $O(x^{2/3+\epsilon})$ arithmetic operations in all. Finally, we must bound the number of operations involved in determining the sets

$$V_{k,b} = \left\{ n^*\colon (n^*p_{b+1}, b) \text{ is a special leaf and } \left[\frac{x}{n^*p_{b+1}}\right] \text{ is in the interval } B_k \right\},$$

which is done just after the block B_k has been sieved by $p_1, \ldots, p_b$. The sets $V_{k,b}$ are determined by checking all m in the interval $J_{k,b}$ given by (3.13) against the array **F** given by (3.7), and this takes $O(|J_{k,b}|x^{\epsilon})$ arithmetic operations, where $|J_{k,b}|$ denotes the number of integers in $J_{k,b}$. Consequently, it will suffice to show that

$$T = \sum_{b=1}^{a-1} \sum_{k=1}^{N+1} |J_{k,b}| \leqslant 7x^{2/3}. \tag{3.15}$$

To do this, we first note that for $k = 1$ the sets $J_{1,b}$ are all empty since $B_1 = [1, N]$ where $N = [x^{1/3}]$, while all special leaves (n, b) have $x/n \geqslant N + 1$. (Indeed the largest n in a special leaf has $n \leqslant N(N-2)$ and then $x/n \geqslant N + 1$.) So it suffices to treat the sum in (3.15) when $k \geqslant 2$. If $k \geqslant 2$, then (3.12) gives

$$|J_{k,b}| \leqslant 1 + \frac{x}{k(k-1)Np_{b+1}} \leqslant \frac{2x^{2/3}}{k(k-1)b} + 1,$$

while

$$|J_{k,b}| \leqslant x^{1/3}$$

by (3.13). Combining these inequalities, we have

$$T \leqslant \sum_{b=1}^{[2x^{1/3}]} \sum_{k=2}^{\infty} \min\left(x^{1/3}, \frac{2x^{2/3}}{k(k-1)b} \right) + x^{2/3}. \tag{3.16}$$

Now

$$\sum_{k=2}^{\infty} \min\left(x^{1/3}, \frac{2x^{2/3}}{k(k-1)b} \right) \tag{3.17}$$

$$\leqslant x^{1/3}\left(\frac{x^{1/6}}{b^{1/2}} \right) + \frac{2x^{2/3}}{b} \sum_{k=[x^{1/6}b^{-1/2}]+1}^{\infty} \frac{1}{k(k-1)} \leqslant 3\frac{x^{1/2}}{b^{1/2}}.$$

Substituting (3.17) in (3.16) yields

$$T \leqslant 3x^{1/2}\left(\sum_{b=1}^{[2x^{1/3}]} \frac{1}{b^{1/2}} \right) + x^{2/3} \leqslant 7x^{2/3},$$

the desired result. □

Proof of Theorem A. This follows from Lemma 3.1 and Lemma 3.2, using the formula (3.1). □

We do not see how to reduce the exponent of the running time bound in Theorem A below 2/3. This conclusion is explained by the following heuristic argument. Suppose there existed a Meissel-Lehmer type algorithm with an asymptotic running time exponent β smaller than 2/3, which computed $\pi(x)$ using the formula

$$\pi(x) = a - 1 + - \sum_{j=2}^{[\log_2 x]} P_j(x, a) + \phi(x, a),$$

for some sieving limit p_a. By the remark after Lemma 3.1, it seems that we must have

$$a \geqslant \pi(x^{1-\beta})$$

to compute $P_2(x, a)$ in $O(x^\beta)$ arithmetic operations. Now consider computing $\phi(x, a)$. In order to keep the time bound $O(x^\beta)$ we can only afford to sieve the interval $[1, x^\beta]$. Hence, it appears that we cannot evaluate nodes (n, b) in the binary tree for $\phi(x, a)$ having $n < x^{1-\beta}$ by a direct sieving. We know of no other way than sieving to quickly evaluate many such values $\phi(x/n, b)$ when b is large. Thus it appears that we must split all such nodes in the binary tree, consequently the tree must contain all nodes corresponding to $(p_r p_s, s-1)$, where

$$a \geqslant p_r \geqslant p_s \geqslant x^{1-\beta}/2,$$

and there are roughly $x^{2(1-\beta)}$ such nodes in the tree. Since the number of leaves exceeds the number of nodes, there are at least $x^{2(1-\beta)}$ leaves. Computing the contribution of these leaves one at a time requires a running time exponent at least $2(1-\beta)$, which exceeds 2/3, a contradiction.

This heuristic argument suggests that to improve the running time exponent below 2/3 one needs at least one of:

(i) A faster way to compute $P_2(x, a)$ than that given after Lemma 3.1.

(ii) A new method to evaluate the contribution of many leaves (n, b) when n is small and b is large.

4. The Extended Meissel-Lehmer Algorithm. We now adapt the Meissel-Lehmer algorithm to the case where there are M parallel processors, with $M \leqslant x^{1/3}$. We compute $\pi(x)$ via formula (3.1), using separate algorithms for computing $P_2(x, a)$ and $\phi(x, a)$, where $a = \pi(x^{1/3})$. Both algorithms have three phases, an *initialization phase* in which the interval $[1, [x^{2/3}]]$ to be sieved is divided into at most M subintervals, a *sieving phase* and a final *accumulation phase* during which the results of the processors are combined to compute the final answer.

The parallel processing algorithm for computing $P_2(x, a)$ is the simpler of the two algorithms.

Algorithm Parallel P_2. In the *initialization phase*, the interval $[1, [x^{2/3}]]$ is subdivided into M subintervals. We let $L = [x^{2/3}/M] + 1$ and set subinterval $B_j = [(j-1)L + 1, jL]$ for $1 \leqslant j \leqslant M - 1$ and set $B_M = [(M-1)L, [x^{2/3}]]$. Also, one processor sieves the interval $[1, x^{1/3}]$ and computes $\pi(x^{1/3})$ and a list $\mathbf{P}$ of all primes up to $x^{1/3}$. The list $\mathbf{P}$ is available to all processors until the end of the algorithm. (Note that the space bounds of Theorem B permit each processor to store the list $\mathbf{P}$ separately throughout the algorithm.)

To describe the *sieving phase*, we introduce some notation. Let $I = [a+1, b+1)$ be a half-open interval with integer endpoints, and let $S(I)$ denote the following set of values associated to the interval I:

(i) The number of primes $\bar{\pi}(I)$ in the interval, i.e.

$$\bar{\pi}(I) = \pi(b) - \pi(a).$$

(ii) The number $n(I)$ of primes p in the set

$$A(I) = \left\{p \colon \frac{x}{b+1} < p \leqslant \frac{x}{a+1} \text{ and } x^{1/3} < p \leqslant x^{1/2}\right\}.$$

(iii) The sum

$$t(I) = \sum_{p \in A(I)} \left[\pi\left(\frac{x}{p}\right) - \pi(a)\right] = \sum_{p \in A(I)} \pi\left(\frac{x}{p}\right) - n(I)\pi(a). \tag{4.1}$$

An important observation is that given the values $S(I_1)$ and $S(I_2)$ for two contiguous intervals $I_1 = [a+1, b+1)$ and $I_2 = [b+1, c+1)$, it is easy to compute the values $S(I)$ for the concatenated interval $I = I_1 \cup I_2 = [a+1, c+1)$ using the formulae

$$\bar{\pi}(I) = \bar{\pi}(I_1) + \bar{\pi}(I_2), \tag{4.2}$$

$$n(I) = n(I_1) + n(I_2), \tag{4.3}$$

and

$$t(I) = t(I_1) + t(I_2) + \bar{\pi}(I_1)n(I_2). \tag{4.4}$$

Now we can describe the sieving phase of the algorithm. Each processor j computes the information $S(B_j)$. This is done by breaking the interval B_j down into subintervals C_{ij} of length $N = [x^{1/3}]$ and sieving each of these subintervals sequentially with only one subinterval in storage at any time. There are at most $[x^{1/3}/M] + 2$ such subintervals. For each subinterval C_{ij}, the information $S(C_{ij})$ is computed during the sieving of that subinterval by a method exactly like that of Algorithm P_2 described in Section 2. By sieving the intervals C_{ij} in increasing order, we may calculate $S(C_{1j} \cup C_{2j} \cup \cdots \cup C_{ij})$ after the ith block is sieved, using the previously calculated data $S(C_{1j} \cup \cdots \cup C_{(i-1)j})$ and the just computed $S(C_{ij})$ using (4.2)–(4.4), then discarding $S(C_{1j} \cup \cdots \cup C_{(i-1)j})$. After the last interval C_{ij} is processed, the calculated data is $S(B_j)$.

In the *accumulation phase*, the information $S(B_j)$ for $1 \leqslant j \leqslant M$ is combined to yield $S(I)$ where $I = [1, [x^{2/3}] + 1)$. A simple method to do this, which is sufficient to obtain the required time and space bounds, is to move all the information $S(B_j)$ for $1 \leqslant j \leqslant M$ to the first processor and then sequentially calculate $S(B_j \cup \cdots \cup B_i)$ for $1 \leqslant i \leqslant M$ using (4.2)–(4.4). At the final step $S(I)$ is calculated. This gives

$$n(I) = \pi(x^{1/2}), \qquad t(I) = \sum_{x^{1/3} < p \leqslant x^{2/3}} \pi\left(\frac{x}{p}\right),$$

since $\pi(0) = 0$. Then we calculate

$$P_2(x, \pi(x^{1/3})) = \binom{n(I)}{2} - \binom{\pi(x^{1/3})}{2} + t(I). \quad \square$$

LEMMA 4.1. *Algorithm Parallel P_2 using M parallel RAM processors with $M \leqslant x^{1/3}$ computes $P_2(x, \pi(x^{1/3}))$ using $O(M^{-1}x^{2/3+\varepsilon})$ arithmetic operations on each processor and $O(x^{1/3+\varepsilon})$ storage locations on each processor, using words of length* $[\log_2 x] + 1$ *bits.*

Proof. This is straightforward. The initialization and accumulation phases clearly use $O(x^{1/3+\varepsilon})$ arithmetic operations and $O(x^{1/3+\varepsilon})$ storage locations. In the sieving phase the sieving of each interval C_{ij} is seen to require $O(x^{1/3+\varepsilon})$ arithmetic operations at $O(x^{1/3+\varepsilon})$ storage locations as in Lemma 3.1. There are at most $O(x^{1/3}/M)$ such intervals sequentially sieved on each processor, yielding a count of $O(x^{2/3+\varepsilon}/M)$ arithmetic operations. $\square$

We now describe our parallel processing algorithm for computing $\phi(x, \pi(x^{1/3}))$.

Algorithm Parallel ϕ. In the *initialization phase*, the interval $[[x^{1/3}], [x^{2/3}] + 1)$ is subdivided into at most M subintervals in such a way as to approximately equalize the running times of all processors in the sieving phase. The two major contributions to the running time in the sieving phase arise from the sieving process itself and from computing the contribution of the special leaves. For any subinterval $I = [y_1, y_2)$ with $y_1 \leqslant [x^{1/3}]$, we define the *sieving cost* by

$$C_S(I) = y_2 - y_1 \tag{4.5}$$

and the *special leaf cost* by

$$C_L(I) = x\left(\frac{1}{y_1} - \frac{1}{y_2}\right). \tag{4.6}$$

Note that both of these functions are additive on disjoint subintervals of $J = [[x^{1/3}], [x^{2/3}] + 1)$, and that

$$C_S(J) \leqslant x^{2/3}, \qquad C_L(J) \leqslant x^{2/3}. \tag{4.7}$$

We subdivide J into at most M subintervals $\{B_i^*: 1 \leqslant i \leqslant M\}$ with $B_i^* = [z_{i-1}, z_i)$ by setting $z_0 = [x^{1/3}]$ and then, after z_{i-1} is chosen, selecting z_i so that z_i is minimal with respect to the property that

$$\text{(4.8)} \qquad \operatorname{Max}(C_S([z_{i-1}, z_i)), C_L([z_{i-1}, z_i))) > \frac{2x^{2/3}}{M}.$$

If no such $z_i \leqslant [x^{2/3}]$ exists, set $z_i = [x^{2/3}]$. To compute z_i explicitly, we observe that (4.5) implies that $C_S([z_{i-1}, z_i)) > 2x^{2/3}/M$ exactly when

$$\text{(4.9)} \qquad z_i > \frac{2x^{2/3}}{M} - z_{i-1},$$

and (4.6) implies that $C_L([z_{i-1}, z_i)) > 2x^{2/3}/M$ exactly when

$$z_i > \frac{Mx^{1/3}z_{i-1}}{Mx^{1/3} - 2z_{i-1}} \quad \text{provided } 2z_{i-1} < Mx^{1/3},$$

and that no such z_i exists if $2z_{i-1} \geqslant Mx^{1/3}$. Hence, z_i is easily computed using

$$z_i = \begin{cases} \operatorname{Min}\left\{[x^{2/3}], 1 + \left[\dfrac{2x^{2/3}}{M} - z_{i-1}\right], 1 + \left[\dfrac{Mx^{1/3}z_{i-1}}{Mx^{1/3} - 2z_{i-1}}\right]\right\} & \text{(4.10a)} \\ \qquad\qquad \text{if } 2z_i - 1 < Mx^{1/3}. & \\ \operatorname{Min}\left\{[x^{2/3}], 1 + \left[\dfrac{2x^{2/3}}{M} - z_{i-1}\right]\right\} & \text{(4.10b)} \\ \qquad\qquad \text{if } 2z_i - 1 \geqslant Mx^{1/3}. & \end{cases}$$

(We verify in Lemma 4.2 below that this method divides $([x^{1/3}], [x^{2/3}] + 1)$ into at most M subintervals.)

Also in the initialization phase, we completely sieve the interval $[1, [x^{1/3}]]$ and compute:

(i) *A* list **P** of all primes $p \leqslant x^{1/3}$.

(ii) An array $\{\phi(z_0, i): 1 \leqslant i \leqslant \pi(x^{1/3})\}$ of the partial sieve function values, where $z_0 = [x^{1/3}]$.

(iii) The array $\mathbf{F} = \{(m, f(m), \mu(m)): 1 \leqslant m \leqslant [x^{1/3}]\}$ given by (3.7).

(iv) The contribution S_1 of the *ordinary leaves* computed as in Section 3.

To describe the sieving phase, we introduce some notation. Let $I = [a + 1, b + 1)$ be a half-open interval with integer endpoints, and let $S^*(I)$ denote the following set of values associated to the interval I.

(i) An array $T(I)$ of partial sieving information for the interval I. Here

$$T = \{\psi_i(I): 1 \leqslant i \leqslant \pi(x^{1/3})\},$$

where

$$\text{(4.11)} \qquad \psi_i(I) = \phi(b, i) - \phi(a, i)$$

denotes the number of elements in the interval I not sieved out by the first i primes.

(ii) An array $R(I)$ counting special leaves. Here

$$R(I) = \{r_i(I): 1 \leqslant i \leqslant \pi(x^{1/3})\},$$

where

$$r_i(I) = \sum{}' \mu(n)$$

where the prime denotes the sum run over all special leaves of the form (n, i) such that x/n lies in the interval I.

(iii) The sum

$$u(I) = \sum_{i=1}^{\pi(x^{1/3})} \sum_{(n,i)}{}' \mu(n)\left[\phi\left(\frac{x}{n}, i\right) - \phi(a, i)\right], \tag{4.12}$$

where the prime means that the sum is over all special leaves (n, i) for which x/n lies in the interval I.

Given the values $S^*(I_1)$ and $S^*(I_2)$ for two contiguous subintervals $I_1 = [a + 1, b + 1)$ and $I_2 = [b + 1, c + 1)$, we can directly compute the values $S^*(I)$ for the concatenated interval $I = I_1 \cup I_2 = [a + 1, c + 1)$ using the formulae

$$\psi_i(I) = \psi_i(I_1) + \psi_i(I_2), \tag{4.13}$$

$$r_i(I) = r_i(I_1) + r_i(I_2), \tag{4.14}$$

for $1 \leqslant i \leqslant \pi(x^{1/3})$, and

$$u(I) = u(I_1) + u(I_2) + \sum_{i=1}^{\pi(x^{1/3})} \psi_i(I_1) r_i(I_2). \tag{4.15}$$

Also, for the interval $J = [[x^{1/3}], [x^{2/3}] + 1)$ the sum $u(J)$ gives the total contribution of the special leaves.

Now we describe the sieving phase of the algorithm. Processor j computes the information $S^*(B_j^*)$. If

$$\text{length}\left(B_j^*\right) = z_j - z_{j-1} \leqslant [x^{1/3}],$$

then B_j^* is sieved in one piece. If length $(B_j^*) > [x^{1/3}]$, then B_j^* is subdivided into subintervals C_{ij}^* of length $[x^{1/3}]$ (with one possibly short subinterval) and these are sieved to compute the information $S^*(C_{ij}^*)$. The information $S^*(C_{ij}^*)$ is computed by a method like that of Algorithm ϕ described in Section 3; the modifications needed are described below. By sieving the intervals C_{ij}^* in increasing order, we may calculate $S^*(C_{1j}^* \cup \cdots \cup C_{ij}^*)$ after the ith subinterval is sieved, using the previously calculated data $S^*(C_{1j}^* \cup \cdots \cup C_{(i-1)j}^*)$ and the just computed $S^*(C_{ij}^*)$ using (4.13)–(4.15). After the last subinterval C_{ij} is sieved, the calculated data is $S^*(B_j^*)$.

To compute the information $S^*(I)$ for an interval $I = [a + 1, b + 1)$, we use the same data structure and formulae (3.4)–(3.14) as in Algorithm ϕ, with the following modifications. We create an array of accumulators $\{\nu_i\colon 1 \leqslant i \leqslant \pi(x^{1/3})\}$. Just after the interval I is sieved by the $(i + 1)$st prime p_{i+1}, we search for special leaves (n, i) having $x/n \in I$, where $n = n^* p_{i+1}$, and for each leaf found we add $\mu(n) = -\mu(n^*)$ to the accumulator ν_i. We locate all such special leaves by finding exactly those n^* in the interval

$$J_i(I) = \left[\frac{x}{(b+1)p_{i+1}}, \frac{x}{(a+1)p_{i+1}}\right] \cap [1, [x^{1/3}]] \tag{4.16}$$

which have $f(n^*) > p_{i+1}$ and $\mu(n^*) \neq 0$. After all these special leaves are found, the value of ν_i is $r_i(I)$. To compute $u(I)$ we use an accumulator to which, after sieving by p_{i+1}, we add the sum

$$\Delta_i = \sum_{(n,i)}{}' \mu(n)\left[\phi\left(\frac{x}{n}, i\right) - \phi(a, i)\right], \tag{4.17}$$

where the prime indicates this sum is over all special leaves (n, i) with $x/n \in I$. (Here i is held fixed.) To compute the inner sum in (4.17) we use a modified form of (3.10), which is

$$\phi(y, i) - \phi(a, i) = \sum_{r=1}^{M} a\left(e_r, 1 + \sum_{s>r} 2^{e_s - e_r}\right), \tag{4.18}$$

where $e_1 > \cdots > e_m \geqslant 0$ are the exponents in the binary expansion of $y - a$.

In the *accumulation phase*, the information $S^*(B_j^*)$ for $1 \leqslant j \leqslant M$ is combined to compute $S^*(J)$ where $J = [[x^{1/3}], [x^{2/3}] + 1)$. Since combining the information $S^*(I)$ from two contiguous intervals requires on the order of $x^{1/3}$ arithmetic operations (see (4.15)), we cannot use the sequential method of Algorithm Parallel P_2. Instead, we combine intervals using a binary tree structure. We define

$$I_i^{(0)} = B_i^*, \qquad 1 \leqslant i \leqslant M,$$
$$I_i^{(j)} = \bigcup_{(i-1)2^j + 1 \leqslant k \leqslant i2^j} B_k^*, \quad 1 \leqslant i \leqslant \left\lceil \frac{M}{2^j} \right\rceil.$$

At step 0 we have $S^*(I_i^{(0)})$ stored in processor i. At step j the values in processors $2i - 1$ and $2i$ are combined and sent to processor i, for $1 \leqslant i \leqslant M/2^{j+1}$. If $m_j = \lceil M/2^j \rceil$ is odd, then the information in processor m_j is sent unchanged to processor $(m_j + 1)/2$. After $\lceil \log_2 M \rceil + 1$ such steps we have computed $S^*(J)$ in processor 1. The contribution S_2 of the special leaves is given in terms of $S^*(J)$ by

$$S_2 = u(J).$$

Finally, we compute

$$\phi(x, [x^{1/3}]) = S_1 + S_2. \quad \square$$

LEMMA 4.2. *Algorithm Parallel ϕ using M parallel RAM processors with $M \leqslant x^{1/3}$ computes $\phi(x, [x^{1/3}])$ using $O(M^{-1}x^{2/3+\epsilon})$ arithmetic operations on each processor and $O(x^{1/3+\epsilon})$ storage locations on each processor, using words of length at most* $[\log_2 x] + 1$ *bits.*

Proof. We first verify that Algorithm Parallel ϕ is correct. The main thing that requires checking is that the initialization phase divides $J = [[x^{1/3}], [x^{2/3}] + 1)$ into at most M subintervals. To see this, let K be the number of such subintervals. According to (4.8), at least $K - 1$ of these subintervals I satisfy

$$C_S(I) + C_L(I) > \frac{2x^{2/3}}{M}.$$

Now using (4.7) and the additivity of the functions C_S and C_L on disjoint intervals, we have

$$\begin{aligned} 2x^{2/3} &\geqslant C_S(J) + C_L(J) \\ &= \sum_{j=1}^{K} \left[C_S(B_j^*) + C_L(B_j^*)\right] > 2x^{2/3}\left(\frac{K-1}{M}\right). \end{aligned}$$

Hence, $K - 1 \leqslant M - 1$ and $K \leqslant M$ follows.

It is easily checked that the running time of Algorithm Parallel ϕ requires $O(x^{1/3+\epsilon})$ arithmetic operations using at most $O(x^{1/3+\epsilon})$ locations on a single processor to complete the Initialization and Accumulation phases of the algorithm. Note here that since there are $\leqslant x^{1/3}$ processors, the z_t can be computed sequentially using (4.10) in $O(x^{1/3+\varepsilon})$ arithmetic operations in the Initialization Phase. Also note that the assumption concerning the simultaneous transfer of information between processors (described just before the statement of Theorem B) is used in obtaining the $O(x^{1/3+\varepsilon})$ arithmetic operations bound for the Accumulation phase.

It remains to examine the sieving phase of the algorithm. We first observe that for any interval B_j,

$$\operatorname{Max}\big(C_S(B_j), C_L(B_j)\big) < \frac{2x^{2/3}}{M} + 2x^{1/3}. \tag{4.19}$$

This inequality follows from the fact that z_j was chosen to be the smallest integer satisfying (4.8), and that C_S and C_L are additive on disjoint intervals, and that for any interval $I = [z, z+1)$ with $z \geqslant [x^{1/3}]$ one has

$$\operatorname{Max}\big(C_S(I), C_L(I)\big) \leqslant 2x^{1/3}.$$

Second, we observe that the main contributions to the running time of each processor in the sieving phase come from the sieving operations and the special leaf computations. The sieving operations on an interval I require $O(C_S(I)x^{\varepsilon} + x^{1/3+\varepsilon})$ arithmetic operations and the special leaf operations require $O(C_L(I)x^{\varepsilon} + x^{1/3+\varepsilon})$ arithmetic operations. For the special leaf operations case, this bound arises from the total length of the intervals $J_t(I)$ searched in (4.16). The time spent computing $S^*(B_j)$ from the data $S^*(C_{ij}^*)$ is also at most $O(C_L(I)x^{\varepsilon})$ arithmetic operations. Finally, the bound of $O(x^{1/3+\varepsilon})$ storage locations follows by observing that all intervals sieved are of length $\leqslant [x^{1/3}]$. □

Proof of Theorem B. This is immediate from Lemma 4.1 and Lemma 4.2. □

5. Practical Modifications of the Extended Meissel-Lehmer Method.

Efficient implementation of the Extended Meissel-Lehmer method requires some modifications to the algorithm presented in Section 3. These modifications are in the area of changing the sieving limit slightly to gain a factor of $\log^2 x$ in speed and of changing the method of finding the special leaves contained in a given interval, to gain an additional factor of $\log x$.

In order to save space in the tables computed, as well as time in the actual sieving, *Truncation rule* T is modified to:

Truncation rule T′. *Do not split a node labelled* $\pm\,\phi(x/n, b)$ *if either of the following holds.*

(i) $b = k$ *and* $n \leqslant y$.

(ii) $n > y$.

Here y is some parameter, $x^{2/5} \geqslant y \geqslant x^{1/3}$, and k is some small integer (taken in the actual implementation to be 5).

In order to find a good choice of y, it is necessary to get a good asymptotic formula for the number of special leaves. More precisely, if the sieving limit is taken to be of the form x/y, then the number of special leaves is the cardinality of the set

$$\left\{n \colon n > y,\ \frac{n}{\delta(n)} \leqslant y\right\},$$

where n is square-free, not prime, and not divisible by primes $< p_k$, and $\delta(n)$ denotes the smallest prime factor of n. A simple upper bound for the cardinality of this set is $y\pi(y)$, which is asymptotic to $y^2/\log y$. However, we have the following:

LEMMA 5.1. *The number of special leaves is*

$$\frac{1}{2}\pi(y)^2 + O\left(\frac{y^{3/2}}{\log y}\right).$$

Proof. We can divide the special leaves x/n into two classes: Those for which $\delta(n) < \sqrt{y}$ and those for which $\delta(n) \geqslant \sqrt{y}$. The cardinality of the former set is $\leqslant y\pi(\sqrt{y})$ which is $O(y^{3/2}/\log y)$ because there are only $\pi(\sqrt{y})$ possible choices for $\delta(n)$. The cardinality of the latter set is exactly

$$\tfrac{1}{2}\left(\pi(y) - \pi(\sqrt{y})\right)\left(\pi(y) - \pi(\sqrt{y}) - 1\right)$$

because in that case we must have $n = pq$ where $\sqrt{y} \leqslant p < q$. These two facts give the result. □

Finding the sieving limit. We use the above formula to find the best sieving limit. The special leaves x/n may be divided into four classes:

(1) Those leaves for which $x^{1/3} < \delta(n) \leqslant y$.

(2) Those leaves for which $\sqrt{x/y} < \delta(n) \leqslant x^{1/3}$.

(3) Those leaves for which $x/y^2 < \delta(n) \leqslant \sqrt{x/y}$.

(4) Those leaves for which $\delta(n) \leqslant x/y^2$.

The contribution of each of the leaves in class (1) is 1, because in that case we have $n = pq$, where $q > p > x^{1/3}$ and so $x/(pq) < x^{1/3} < p$. The exact number of leaves in class (1) is

$$\frac{\left(\pi(y) - \pi(x^{1/3})\right)\left(\pi(y) - \pi(x^{1/3}) - 1\right)}{2},$$

which is also their total contribution. It takes constant time to calculate this. Each of the leaves in class (2) is also of the form $x/(pq)$. If $q > x/p^2$, then $x/(pq) < p$, so the contribution of such a leaf is 1. We call such a leaf *trivial.* The number of such leaves is

$$\sum_p \left\{\pi(y) - \pi\left(\frac{x}{p^2}\right)\right\},$$

where the sum is over primes p, $\sqrt{x/y} < p < x^{1/3}$. This number is also their contribution. This sum takes time $O(x^{1/3})$ to calculate, once we have tabulated $\pi(m)$ for $m \leqslant y$. All of the other leaves in class (2) satisfy $\sqrt{y} \leqslant p \leqslant x/(pq) \leqslant p^2$, so we have

$$\phi(x/(pq), \pi(p) - 1) = 1 + \pi(x/(pq)) - \pi(p) + 1.$$

We now derive a bound for the number of such leaves.

If $e^{m-1}\sqrt{x/y} < p \leqslant e^{m}\sqrt{x/y}$, then the number of q such that $p < q < x/p^2$ is $O(y/(e^{2m-2}\log x))$, so the total number of pairs (p, q) is $O(\sqrt{xy}/(e^{m-2}\log^2 x))$. Summing over all m gives $O(\sqrt{xy}/\log^2 x)$ nontrivial leaves in class (2). We may evaluate each of the leaves in class (2) in constant time after doing $O(y \log\log x)$ preprocessing required to tabulate the values of $\pi(m)$ for $m \leqslant y$.

A leaf in class (3) is of the form $x/(pq)$ because $p > x/y^2 \geqslant \sqrt{y}$. If a leaf $x/(pq)$ in class (3) satisfies $q \geqslant x/(yp)$, then $x/(pq) \leqslant y \leqslant p^2$. We call such a leaf *easy*. A leaf in class (3) which is not easy is *hard*. Each easy leaf may be evaluated in constant time as can each of those in class (2). The number of such leaves is $O(\sqrt{xy}/\log^2 x)$. We derive a bound for the number of hard leaves. If $e^{m-1}x/y^2 < p \leqslant e^{m}x/y^2$, then the number of q such that $p < q < x/(yp)$ is $O(y/(e^{m-1}\log x))$. So the total number of pairs (p, q) is $O(x/(y\log^2 x))$. For leaves of class (3) we have that $m \leqslant \frac{1}{2}\log(y^3/x)$. So the total number of leaves in question is $O((x\log(y^3/x))/(y\log^2 x))$.

The number of leaves in class (4) is $O(x/(y\log^2 x))$.

The total amount of work done in the algorithm is

$$O\left(\frac{x}{y}\log\frac{x}{y}\right),$$

for the sieving, $O(\sqrt{x}\log\log x)$ for finding the primes $\leqslant \sqrt{x}$ in algorithm P_2, and

$$O\left(x\log\log x + \left(\log(y^3/x)\right)(x/y)/\log x + \sqrt{xy}/\log^2 x\right)$$

for the calculation of the leaf sums. The first term is negligible, and the second is always dominated by the total sieving cost. We choose $y = cx^{1/3}\log^2 x$ which balances the sieving and calculating special leaves, giving time $O(x^{2/3}/\log x)$. (In the actual implementation, a good value of the constant c was determined empirically.) It is also necessary to account for the time spent in actually finding the special leaves. We show below that using the proper data structure, this is

$$O\left(\sqrt{xy}/\log^2 x + Iy/\log x\right),$$

where I is the number of intervals sieved. Thus, with the above choice of y we must have $I = O(x^{1/3}/\log^2 x)$. This is satisfied if we take equal-sized intervals of length $x^{1/3}$ or greater.

Finding the special leaves. In order to find the special leaves in the above quoted time, we calculate for each prime $p_k \leqslant \sqrt{y}$ two parallel tables A_k and M_k. The value of $A_k(j)$ is the jth square-free $n \leqslant y$ such that $\delta(n) = p_k$. The value of $M_k(j)$ is $\mu(A_k(j))$. For each prime $p_k \leqslant \sqrt{y}$ we store a table called N_k such that $l = N_k(j)$ satisfies $A_k(l-1) < jp \leqslant A_k(l)$. Clearly, these tables may be computed in time $O(y\log x)$ and take up space $O(y\log\log x)$.

To find all special $n \in [a, b)$ we use these two tables together with the procedure in Figure 2. By using these tables one can find each special leaf $x/n \in [a, b)$ in a constant time, plus a constant time for each interval and each pair (p, q), $p < q < \sqrt{y}$, which gives time $O(\pi(\sqrt{y})^2)$, plus the time to find the special leaves of the form $x/(pq)$, which takes $O(\pi(y))$ time per interval, in overhead.

```
{Special leaves with $\delta(n) \leqslant \sqrt{y}$}
for $i := k$ to $\pi(\sqrt{y})$ do begin
    for $j := i + 1$ to $\pi(\sqrt{y})$ do begin
        $m := A_j([(a-1)/p_j + 1])$
        while $N_j(m) \leqslant b/p_i$ do begin
            process $(p_i N_j(m))$;
            {it is a special leaf}
            $m := m + 1$
            end {while}
        end {for}
    end {for $i$};
{------------------------------}
{special leaves which are the product of two primes}
for $i := k$ to $\pi(y)$ do begin
    $m :=$ nextprime $([a/p_i])$;
        while $p_m \leqslant b/p_i$ do begin
            process $(p_i p_m)$;
            {it is a special leaf}
            $m := m + 1$
        end {while}
    end {for $i$};
```

FIGURE 2
Computing special leaves

Calculating $P_2(x, a)$. The sieving done in calculating the contributions for the special leaves can also be used to compute $P_2(x, a)$. The algorithm P_2 may be interleaved with the running of algorithm ϕ to avoid doing the sieving twice.

Remarks. The dominant time in the sieving is contributed by the time that it takes to delete the nonprimes. Thus, if we choose $k > 0$ we can get a speedup of a factor of r_k by storing only the residue classes mod P_k which are not divisible by the first k primes. This, in essence, precomputes the first k stages of the sieving. It does, however, slow down the accesses to the individual elements of the sieve. In the balance, though, it seems to be a big gain. In the program described below, changing k from 0 to 5 resulted in a speedup of 25 percent.

6. Computational Results. The second author implemented a version of the Extended Meissel-Lehmer method on an IBM/370 3081 Model K, using the modified algorithm described in Section 5.

The programs implementing the algorithm were all written in PL.8 [2], a high-level systems programming language with a highly optimizing compiler written at IBM Research. There are tremendous benefits involved in implementing this algorithm in a higher-level language. This allowed different strategies to be tried in the algorithm with relative ease, compared to an assembly language program. Optimizing various program parameters led to a five-fold speedup of the original version of the program.

The analysis of Section 5 assumed that all arithmetic operations take constant time. For the actual program this is a reasonable approximation. All arithmetic was done using 32-bit integers when possible, otherwise it was done in floating point, which on the IBM/370 is capable of representing integers up to $2^{56} - 1$ exactly. We show below that all intermediate results for $x \leqslant 4 \times 10^{16}$ satisfy this bound.

Fix a parameter k and define $P_k = p_1 p_2 \cdots p_k$ and

$$r_k = (1 - 1/p_1)(1 - 1/p_2) \cdots (1 - 1/p_k),$$
$$s_k = (6/\pi^2)/((1 + 1/p_1) \cdots (1 + 1/p_k)).$$

Note that $s_k \leqslant r_k$.

The ordinary leaves. The contribution of the ordinary leaves is $N(y)$, where

$$N(z) = \sum_n \mu(n)\phi\left(\frac{x}{n}, k\right)$$

and n ranges over all square-free integers $\leqslant z$ not divisible by the first k primes. We have

$$N(z) = x\phi(P_k) \sum_{n \leqslant z} \frac{\mu(n)}{P_k n} + \sum_{n \leqslant z} \mu(n)\left\{\frac{x}{P_k n}\right\} + \sum_{n \leqslant z} \mu(n)\phi\left(\frac{x}{n} \bmod P_k, k\right).$$

If we calculate the sum in order then any intermediate result is bounded by $r_k x + (1 + \phi(P_k))y$ (cf. [8, p. 582]). When $x = 4 \times 10^{16}$ and $k = 5$ this is bounded by 0.85×10^{16}.

The special leaves. The contribution of the special leaves is

$$\sum_n \mu(n)\phi\left(\frac{x}{n}, \pi(\delta(n)) - 1\right),$$

where the sum is taken over all special n. Now we have

$$\phi\left(\frac{x}{n}, r\right) = \left[\frac{x}{P_r n}\right]\phi(P_r) + \phi\left(\left[\frac{x}{n}\right] \bmod P_r, r\right),$$

where $r = \pi(\delta(n)) - 1$. So if $\delta(n) > p_{k+1}$, then the contribution of this n is $\leqslant (x/n)r_k + 2\phi(P_k)$. Hence, any intermediate sum is bounded by

$$\sum_n \left(r_k \frac{x}{n} + 2\phi(P_k)\right), \tag{6.1}$$

where the sum is taken over all square-free $n \in [x^{1/3}, x^{2/3})$ not divisible by primes $\leqslant p_k$. If we denote by $Q_k(x)$ the number of such integers $\leqslant x$, then, by Landau [8, pp. 633–636], we have

$$Q_k(x) = s_k x + R(x),$$

and $|R(x)| \leqslant \phi(P_k)\sqrt{x}$. The sum may be written as a Stieltjes integral

$$\int_{x^{1/3}}^{x^{2/3}} \left(r_k \frac{x}{t} + 2\phi(P_k)\right) dQ_k(t).$$

Using the above expression for Q_k and integration by parts we find that the sum in (6.1) is bounded by $(1/3)r_k^2 x \log x + 2\phi(P_k)x^{2/3} + r_k\phi(P_k)x^{5/6}$. When $x = 4 \times 10^{16}$ and $k = 5$ this is bounded by 2.2×10^{16}.

Numerical results. The program was run on selected values of x up to 4×10^{16}. Some of the results are presented in Table 1. That table also includes the discrepancies between $\pi(x)$ and $Li(x) = \int_0^x dt/\ln t$, where the integral is to be regarded as the

Cauchy principal value, and between $\pi(x)$ and Riemann's approximation to $\pi(x)$,

$$R(x) = \sum_{n=1}^{\infty} \frac{\mu(n)}{n} Li(x^{1/n}),$$

as well as timing information.

A major surprise is that J. Bohman's value [3] for $\pi(10^{13})$ turns out to be too small by 941. We checked our computations of $\pi(10^{13})$ in several ways. First, we checked that the program computed values of $\pi(x)$ that agreed with existing tables for values of x smaller than 10^{13}. Second, the value of $\pi(10^{13})$ was computed several times using different sieving limits, in which case the intermediate terms summing to $\pi(10^{13})$ are different for each different sieving limit. Third, we computed $\pi(10^{13} + 10^5)$ and sieved the interval $[10^{13}, 10^{13} + 10^5]$ to locate all primes in it, using this to get a check on the computation for $\pi(10^{13})$. Similarly, we checked all the other values of $\pi(x)$ in the table by also computing $\pi(x + 10^5)$ and sieving the interval $[x, x + 10^5]$. (The other values in Bohman's [3] table agree with ours.) Our computation of $\pi(10^{13})$ took approximately 9 minutes, which is about 30 times less than Bohman's computation [3]. Since the IBM 3081 Model K computer is generally thought to be 5 to 7 times faster than Bohman's machine, our algorithm seems to be about 5 times faster than Bohman's at $x = 10^{13}$, and we expect the disparity to grow rapidly for larger x.

TABLE 1

Values of $\pi(x)$

x	$\pi(x)$	$Li(x) - \pi(x)$	$R(x) - \pi(x)$	Time (minutes)
10^{12}	37,607,912,018	38,263	-1,476	2
2×10^{12}	73,301,896,139	48,195	-6,432	3
3×10^{12}	108,340,298,703	80,927	15,096	4
4×10^{12}	142,966,208,126	61,848	-13,314	5
5×10^{12}	177,291,661,649	72,126	-11,182	6
6×10^{12}	211,381,427,039	99,289	8,669	7
7×10^{12}	245,277,688,804	110,790	13,484	7
8×10^{12}	279,010,070,811	79,478	-24,020	8
9×10^{12}	312,600,354,108	127,831	18,542	8
10^{13}	346,065,536,839	108,971	-5,773	9
2×10^{13}	675,895,909,271	170,356	12,194	14
3×10^{13}	1,000,121,668,853	157,353	-33,533	18
4×10^{13}	1,320,811,971,702	221,646	3,483	22
5×10^{13}	1,638,923,764,567	253,033	11,037	25
6×10^{13}	1,955,010,428,258	323,908	60,505	28
7×10^{13}	2,269,432,871,304	197,552	-85,431	31
8×10^{13}	2,582,444,113,487	327,644	26,520	34
9×10^{13}	2,894,232,250,783	348,266	30,170	36
10^{14}	3,204,941,750,802	314,890	-19,200	39
2×10^{14}	6,270,424,651,315	531,925	70,408	59
3×10^{14}	9,287,441,600,280	434,926	-122,759	77
4×10^{14}	12,273,824,155,491	567,492	-70,423	93
5×10^{14}	15,237,833,654,620	812,601	104,541	107
6×10^{14}	18,184,255,291,570	530,687	-240,406	120
7×10^{14}	21,116,208,911,023	874,392	45,623	132
8×10^{14}	24,035,890,368,161	1,084,477	202,253	143
9×10^{14}	26,944,926,466,221	1,179,734	247,489	155
10^{15}	29,844,570,422,669	1,052,619	73,218	165
10^{16}	279,238,341,033,925	3,214,632	327,052	718
2×10^{16}	547,863,431,950,008	3,776,488	-225,875	1125
4×10^{16}	1,075,292,778,753,150	5,538,861	-10,980	1730

AT & T Bell Laboratories
Murray Hill, New Jersey 07974

IBM T. J. Watson Research Center
Yorktown Heights, New York 10598

AT & T Bell Laboratories
Murray Hill, New Jersey 07974

1. A. Aho, J. E. Hopcroft & J. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, Mass., 1974.

2. M. A. Auslander & M. E. Hopkins, "An overview of PL.8 compiler," *Proceeding of the SIGPLAN 82 Symposium on Compiler Construction*, Boston, 1982.

3. J. Bohman, "On the number of primes less than a given limit," *BIT*, v. 12, 1972, pp. 576–577.

4. L. E. Dickson, *History of the Theory of Numbers*, Volume 1, Chapter XVIII, Chelsea, New York, 1971. (Reprint.)

5. R. A. Hudson & A. Brauer, "On the exact number of primes in the arithmetic progressions $4n \pm 1$ and $6n \pm 1$," *J. Reine Angew. Math.*, v. 291, 1977, pp. 23–29.

6. J. C. Lagarias & A. M. Odlyzko, "New algorithms for computing $\pi(x)$," in *Number Theory* (D. V. Chudnovsky et al., Eds.), Lecture Notes in Math., Vol. 1052, Springer-Verlag, New York, 1984, pp. 176–193.

7. J. C. Lagarias & A. M. Odlyzko, "Computing $\pi(x)$: An analytic method." (Preprint.)

8. E. Landau, *Primzahlen*, reprint, Chelsea, New York, 1958.

9. D. H. Lehmer, "On the exact number of primes less than a given limit," *Illinois J. Math.*, v. 3, 1959, pp. 381–388.

10. D. C. Mapes, "Fast method for computing the number of primes less than a given limit," *Math. Comp.* v. 17, 1963, pp. 179–185.

11. E. D. F. Meissel, "Über die Bestimmung der Primzahlmenge innerhalb gegebener Grenzen," *Math. Ann.*, v. 2, 1870, pp. 636–642.

12. E. D. F. Meissel, "Berechnung der Menge von Primzahlen, welche innerhalb der ersten Hundert Millionen natürlicher Zahlen vorkommen," *Math. Ann.*, v. 3, 1871, pp. 523–525.

13. E. D. F. Meissel, "Über Primzahlmengen," *Math. Ann.*, v. 21, 1883, p. 304.

14. E. D. F. Meissel, "Berechnung der Menge von Primzahlen, welche innerhalb der ersten Milliarde natürlicher Zahlen vorkommen," *Math. Ann.*, v. 25, 1885, pp. 251–257.

15. H. S. Wilf, "What is an answer?," *Amer. Math. Monthly*, v. 89, 1982, pp. 289–292.