1977 ACM Student Award
Paper: Second Place

Some New Upper Bounds on the Generation of Prime Numbers

Harry G. Mairson
Yale University

Given an integer N, what is the computational complexity of finding all the primes less than N? A modified sieve of Eratosthenes using doubly linked lists yields an algorithm of $O_A(N)$ arithmetic complexity. This upper bound is shown to be equivalent to the theoretical lower bound for sieve methods without preprocessing. Use of preprocessing techniques involving space-time and additive-multiplicative tradeoffs reduces this upper bound to $O_A(N/\log \log N)$ and the bit complexity to $O_B(N \log N \log \log \log N)$. A storage requirement is described using $O_B(N \log N/\log \log N)$ bits as well.

Key Words and Phrases: computational complexity, sieve, prime number generation, number theory, linked list, preprocessing, balancing

CR Categories: 5.25, 5.39

Author's address: Calhoun College, Yale University, New Haven, CT 06520.

1. Eratosthenes' Sieve

The sieve of Eratosthenes, named after the third-century B.C. Greek astronomer and geographer, consists of writing the odd integers from 3 up in succession up to N, and crossing off every third element after 3, every fifth element after 5, and so on until multiples of all odd integers less than $\sqrt{N}$ have been eliminated. The integers remaining from this sieve technique are exactly the prime numbers from 3 to N.

2. Complexity Analysis and Models of Computation

We evaluate the effectiveness of an algorithm such as the sieve of Eratosthenes by analyzing its *space complexity* (the size of the data structure needed to run the procedure) and its *time complexity* (the time required to run the procedure) as a function of the magnitude of the input parameters. The manner in which we measure time or space requirements is dependent on a *model of computation*. Storage is measured in either (1) pieces of data, such as number of items to be sorted, number of items in a list, or number of storage locations necessary, or (2) overall number of bits needed to contain the data structure. Time is measured in terms of either (1) number of arithmetic operations, such as addition or multiplication or logical arithmetic compare operations (equal, greater or less than, etc.), or (2) number of bit operations (and, or, not) required. Our model of computation is essentially a RAM (random access machine), so that data may be stored or retrieved from a location in memory in a constant amount of time. In both storage and time analyses we refer to (1) and (2) as the *arithmetic complexity* and the *bit complexity*, respectively. We simplify here to consider algorithms with one input parameter N, such as the sieve of Eratosthenes.

If an algorithm requires $g(n)$ time or storage to run, the algorithm is said to be *of order* $f(n)$, denoted $O(f(n))$ if and only if $g(n) \leq cf(n)$ for some constant c and most n, excepting a finite and possibly empty set of special cases of n. Arithmetic and bit complexities are denoted as $O_A(f(n))$ and $O_B(f(n))$, respectively. Note that the inclusion of a constant factor c allows us to essentially ignore lower-order terms. For example, an algorithm running in $n^3 + 4n^2$ time runs in less than $5n^3$ time and hence is $O(n^3)$. Our attention here will be devoted to the asymptotic arithmetic complexity and asymptotic bit complexity of fast algorithms that generate prime numbers, that is, their complexity of computation as N becomes very large.

The arithmetic complexity of computation for the sieve of Eratosthenes can be measured in terms of the number of integers that are sieved out. With the approximations that there are $O((\sqrt{N})/\log\sqrt{N})$ primes up to $\sqrt{N}$ and that the ith prime grows asymptotically as $i \log i$, the number of integers crossed out is roughly

$$C(N) = \sum_{i=2}^{(\sqrt{N})/\log\sqrt{N}} \frac{N/2}{i \log i} \simeq \frac{N}{2} \int_{x=2}^{(\sqrt{N})/\log\sqrt{N}} \frac{dx}{x \log x} \simeq \frac{N \log\log x}{2} \Big|_{x=2}^{x=(\sqrt{N})/\log\sqrt{N}},$$

so the order of arithmetic complexity is $O_A(N \log \log N)$.[1] Since each cross-out requires an addition of two integers less than N and addition is an $O_B(\log N)$ procedure, the bit complexity is $O_B(N \log N \log\log N)$. The storage requirement can be satisfied by an array of $N/2$ bits, initially set to 1 and zeroed as they are sieved out. It is evident that the sieve described above is far from an efficient algorithm since every odd composite number is crossed out by each of its prime factors.

An obvious lower bound on any sieve method is clearly $O_B(N)$, as there are approximately $N - (N/\log N)$ composite numbers to cross out. The best sieve algorithm will cross out each composite only once and will require some constant number of bit operations to generate each of these composite integers.

3. Space-Time Tradeoff Strategies

The efficiency of the sieve of Eratosthenes, as well as most sieve methods, is due to the implied tradeoff between running time and storage space. A more complicated data structure is introduced (a sieve) to raise the speed of the algorithm. By contrast, let us consider the complexity of a more naive and familiar technique.

Suppose we test the primeness of each integer i by checking if it is divisible by any integer less than $\sqrt{i}$. To find the primes up to N, the arithmetic complexity would be

$$C(N) = \sum_{i=1}^{N} \sqrt{i} \simeq \int_1^N (\sqrt{x})\, dx = 2x^{3/2}/3 \Big|_1^N \simeq O_A(N^{1.5}).$$

With the fastest known division algorithm, the bit complexity would be $O_B(N^{1.5} \log N \log\log N \log\log\log N)$. This will run very slowly. Its strength, however, is that it can run in $O_B(\log N)$ storage since we only need a few registers to count up to N plus some space to carry out the division.

The sieve technique increases required storage in a trade for reduction in the running time. Ideally the two are balanced, so that the cost of neither is prohibitive. We suggest further ways of extending this tradeoff.

4. Chartres' Algorithm

In one earlier attempted speed-up of the sieve of Eratosthenes, B.A. Chartres [2] suggested a modified sieve in which sweeps over a large array of length $n/2$ are replaced by repeated sweeps over a much smaller array, where potential primes r are compared to odd multiples of primes less than $\sqrt{r}$. To compute the primes up to m, the Chartres algorithm uses two loops:

for $n := 5$ **step** 2 **until** m
⋮
for $i := 2$ **step** 1 **until** j

where j is the largest integer such that the jth prime p_j is less than $\sqrt{m}$. This allows us to prove the following theorem.

THEOREM 1. *The Chartres algorithm is of asymptotic complexity* $O_A(m^{1.5}/\log m) \le C(m) \le O_A(m^{1.5})$.

PROOF. Assume that operations in the inner loop are done in constant time. This is a convenient assumption to evaluate the arithmetic complexity. Since there are roughly $O((\sqrt{m})/\log\sqrt{m})$ primes less than $\sqrt{m}$, the cost of this algorithm is, for some constant k,

$$C(m) = \sum_{n=0}^{(m-5)/2} \sum_{i=2}^{[2\sqrt{(2n+5)}]/\log(2n+5)} k = 2k \sum_{n=0}^{(m-5)/2} \left(\frac{\sqrt{(2n+5)}}{\log(2n+5)} - 1 \right) \simeq 2k \int_0^{(m-5)/2} \frac{\sqrt{(2x+5)}\,dx}{\log(2x+5)}.$$

Substitution of $y = 2x + 5$ yields

$$C(m) = k \int_5^m (\sqrt{y})/\log y \, dy,$$

and substitution of $y = e^{2v}$ yields

$$C(m) = k \int_{(\log 5)/2}^{(\log m)/2} \frac{e^v (2e^{2v}) dv}{2v} = k \int \frac{e^{3v} dv}{v}.$$

Since $v \le (\log m)/2$ we know that

$$C(m) \ge k \int \frac{e^{3v} dv}{(\log m)/2} \quad \text{or} \quad C(m) \ge \frac{2k}{3 \log m} e^{3v} \Big|_{(\log 5)/2}^{(\log m)/2}$$

which gives the result that $C(m) \ge O_A(m^{1.5}/\log m)$. But we also know that $v \ge (\log 5)/2$; hence

$$C(m) \le \frac{2k}{\log 5} \int e^{3v} dv = \frac{2k}{3 \log 5} e^{3v} \Big|_{(\log 5)/2}^{(\log m)/2} \le O_A(m^{1.5}),$$

and therefore we know that $C(m)$ is bounded above by $O_A(m^{1.5})$. Hence there exist constants c_1, c_2 such that $c_1 m^{1.5}/\log m \le C(m) \le c_2 m^{1.5}$. Note that only one addition in the inner loop immediately raises the bit complexity to at least $O_B(m^{1.5})$. □

This is close to Chartres' original measurement that computing the first k primes took $O(k^{1.57})$ time. If the Chartres method is indeed faster for small m than the simple sieve, it cannot be so when n gets very large, as is the case with any polynomial-ordered algorithm $O(m^t)$ where $t > 0$. Hence its asymptotic bit complexity is actually slower than the sieve of Eratosthenes.

A second version of Chartres' algorithm improved the order of complexity to $O(k^{1.35})$. R.C. Singleton [5] increased the efficiency of the Chartres method by

[1] "log" means $\log_e$ throughout the paper.

partially sorting the multiple list to lower the number of comparisons. Stopwatch timing of this method of finding the first m primes less than n, with a prohibitive storage requirement of $10m$, yielded an $O(n^{1.094})$ algorithm.

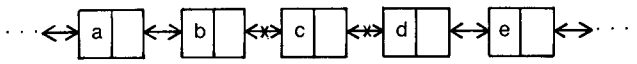
5. A Multiplicative Algorithm

The new method suggested in this paper consists of repeated sweeps over parts of a doubly linked list, where atoms in the list represent potential primes. Each composite number represented in the list is crossed out once as a multiple of its smallest prime factor. To find the primes up to N, the process is iterated for each prime $p \leq \sqrt{N}$.

For each prime $p_i \leq \sqrt{N}$, consider the factor set F_i where

$F_i = \{m \mid p_i \leq m \leq \lfloor N/p_i \rfloor$ and m is not divisible by any prime $p < p_i\}$.

Each atom in the list represented by an element of the product set $p_i \times F_i$ is then crossed out by switching pointers in the list. For example, to cross out atom c:



If $RLINK[i]$ and $LLINK[i]$ refer to the forward (right) and backward (left) pointers for the ith atom in the list, deletion would be effected through the assignments

$RLINK[LLINK[i]] \leftarrow RLINK[i]$,
$LLINK[RLINK[i]] \leftarrow LLINK[i]$.

A possible realization of the algorithm is given below.

```
integer arrays RLINK, LLINK, DELETE
procedure SIEVE (N):
  comment create the doubly linked list
  (RLINK[i] ← i + 1) for i ← 1 to N − 1 step 1;
  (LLINK[i] ← i − 1) for i ← 2 to N step 1;
  comment execute the sieve
  PRIME ← 2; FACTOR ← 2;
  while PRIME ≤ √N do (
    POINTER ← 0;
    while PRIME*FACTOR ≤ N do (
      K ← PRIME*FACTOR;
      POINTER ← POINTER + 1;
      DELETE[POINTER] ← K;
      FACTOR ← RLINK[FACTOR]);
    CROSSOFF(DELETE[i]) for i ← 1 to POINTER step 1;
    PRIME ← RLINK[PRIME];
    FACTOR ← PRIME);
  comment output the primes
  P ← RLINK[1];
  while P ≠ 0 do (
    output(P);
    P ← RLINK[P]);
end
subprocedure CROSSOFF (A):
  RLINK[LLINK[A]] ← RLINK[A];
  LLINK[RLINK[A]] ← LLINK[A];
end
```

The doubly linked structure allows the multiples of p_i not divisible by any prime $p < p_i$ to be crossed out in a linear $O_A(|F_i|)$ time. Theorem 2 expands upon this observation.

THEOREM 2. *The arithmetic complexity of the above algorithm is* $C(N) = O_A(N)$.

PROOF I. (1) *Claim: The algorithm is correct, and no composite is crossed off twice.* Every composite integer m represented in the list has a lowest prime factor p_a, where $2 \leq p_a \leq \sqrt{m}$ and $\sqrt{m} \leq \sqrt{N}$, and hence will be sieved out by one element of the factor set $p_a \times F_a$. No composite is crossed off twice because the fundamental theorem of arithmetic states that p_a is unique since m has a unique prime factorization. (2) *Claim: Each set* F_i *can be created in* $O_A(|F_i|)$ *time.* We create F_i by beginning a search of the doubly linked list at the atom representing p_i. We then search forward through the list, adding new elements to F_i, until an element greater than $\lfloor N/p_i \rfloor$ is found. There is no discarding of elements excepting the last one scanned, which terminates the procedure creating F_i; this is because the elements in the list are naturally sorted in ascending order, and the doubly linked structure of the list only allows examination of elements relatively prime with respect to primes less than p_i.

The number of composites that are sieved out is then

$$\left| \bigcup_i p_i \times F_i \right| \simeq N - (N/\log N)$$

since the algorithm is correct. But each composite is sieved out in a constant number of arithmetic operations since each F_i was created in linear time. Hence $C(N) = O_A(N)$. □

PROOF II. An alternate proof of the order of this algorithm results from the observation that

$$|F_i| = \lfloor N\,\phi(p_1 p_2 \cdots p_i)/p_1 p_2 \cdots p_i \rfloor$$

where $\phi(n)$, the Euler ϕ function, equals the number of integers between 0 and n which are relatively prime to n. Using the well-known expression for the ϕ function (see [4]),[2]

$$\phi(p_1 p_2 \cdots p_i) = (p_1 - 1)(p_2 - 1)\cdots(p_i - 1),$$

we obtain[3]

$$|F_i| = \left\lfloor N \prod_{j=1}^{i} (p_j - 1)/p_j \right\rfloor,$$

and hence the cost is

$$C(N) = \sum_{i=1}^{(2\sqrt{N})/\log N} Nk \prod_{j=1}^{i} (p_j - 1)/p_j.$$

If p_k is the largest prime less than $\sqrt{N}$, we note that since the following is true:

[2] Proof available from the author.
[3] This was pointed out to me by David Dobkin.

$$\prod_{j=1}^{i} (p_j - 1)/p_j \le ((p_k - 1)/p_k)^i \le ((\sqrt{N} - 1)/\sqrt{N})^i,$$

it is clear that

$$C(N) \le Nk \sum_{i=1}^{(2\sqrt{N})/\log N} \left(\frac{\sqrt{N} - 1}{\sqrt{N}}\right)^i \simeq Nk \left[\frac{((\sqrt{N} - 1)/\sqrt{N})^{(2\sqrt{N})/\log N} - 1}{((\sqrt{N} - 1)/\sqrt{N}) - 1}\right]$$

or

$$C(N) \le Nk\,(1 - 1/\sqrt{N})^{(2\sqrt{N}/\log N}.$$

As $(\sqrt{N})/\log\sqrt{N}$ is an increasing function for large N, this converges to

$$C(N) \le Nk\,(1 - 1/\sqrt{N})^{\sqrt{N}} \le Nk \text{ or } C(N) \le O_A(N).$$

Since we have already shown an arithmetic lower bound of $O_A(N)$ without preprocessing, $C(N) = O_A(N)$. □

A bit complexity analysis of the above algorithm, however, yields less impressive results. This is because each cross-out necessitates one multiplication of p_i by some element of F_i, which is not done in constant time. Two integers $q, r \le N$ can be multiplied by using the Schönhage-Strassen algorithm in $O_B(\log N \log\log N \log\log\log N)$ time. Thus the bit complexity of our algorithm is $O_B(N \log N \log\log N \log\log\log N)$, which is actually slower than the sieve of Eratosthenes. This is because the sieve of Eratosthenes does not use multiplication, but rather a simple indexing procedure that requires only addition.

The storage requirement of the algorithm described above is also greater than the simple sieve. While the simple sieve uses $O_B(N)$ bits of storage, the multiplicative algorithm must store a doubly linked list with N atoms at a cost of $2N\log N$ bits and a list of atoms marked for deletion, which has a maximum length of $(N/2)\log N$ bits. Hence storage is of order $O_B(N\log N)$.

6. Preprocessing and Additive-Multiplicative Tradeoff Strategies

What can be done to further reduce these upper bounds? So far, we have interpreted our list so that the ith atom represents the ith integer between 1 and N. Suppose we eliminate multiples of primes $p_1, \ldots, p_k$ beforehand and compact the linked list so that it is stored in a smaller array. If our algorithm remains substantially the same, its complexity will be noticeably reduced because it will be sieving over a reduced data structure. We note that this preprocessing will have two basic effects:

(1) The number of atoms in the doubly linked list will be reduced to

$$N\prod_{i=1}^{k} (p_i - 1)/p_i.$$

Furthermore, the maximum length of the delete list will be reduced to

$$(N/p_{k+1})\prod_{i=1}^{k} (p_i - 1)/p_i.$$

(2) Since the ith atom no longer represents i, we must have an interpretation function from the atoms of the list onto a subset of integers $1 \le i \le N$ such that i is relatively prime with respect to $p_1, \ldots, p_k$. The distribution pattern of nonmultiples of these primes repeats modulo $\Pi_1^k p_i$, and there are $\Pi_1^k (p_i - 1)$ integers in each cycle. A lookup table T of size $\Pi_1^k (p_i - 1)$ is then necessary to find such a relatively prime integer.

Given this reduced data structure, the atom representing n can be determined as

$$ATOM(n) = \left\lfloor n \prod_{1}^{k} (p_i - 1)/p_i \right\rfloor + 1,$$

and the inverse transformation from atom to value is

$$VALUE(a) = \left(\left\lfloor a \Big/ \left(\prod_{1}^{k} p_i - 1\right)\right\rfloor \times \prod_{1}^{k} p_i\right) + T\left[a \bmod \prod_{1}^{k} (p_i - 1)\right].$$

The total storage requirement will then be reduced to

$$S(N, k) = N(1 + 1/p_{k+1}) \prod_{i=1}^{k} (p_i - 1)/p_i + \prod_{j=1}^{k} p_j - 1.$$

Given a fixed N, what is a good choice for k to minimize this storage function?

THEOREM 3. *$S(N, k)$ is minimized at some $k \ge t$, where $\Pi_1^t p_j = N$.*

PROOF. We first note that, for large N and k,

$$S(N, k) \simeq N\prod_{1}^{k} (p_i - 1)/p_i + \prod_{1}^{k} p_i - 1$$

since the $1/p_{k+1}$ term tends to zero. Let $N = \Pi_1^t p_j$. Then assuming that $k \le t$,

$$\begin{aligned} S(N, k) &\simeq \prod_{1}^{t} p_j \prod_{1}^{k} (p_i - 1)/p_i + \prod_{1}^{k} p_i - 1 \\ &\simeq \left(\prod_{k+1}^{t} p_j \prod_{1}^{k} p_j\right) \prod_{1}^{k} (p_i - 1)/p_i + \prod_{1}^{k} p_i - 1 \\ &\simeq \prod_{k+1}^{t} p_j \prod_{1}^{k} (p_i - 1) + \text{lower-order terms}. \end{aligned}$$

This is certainly minimized at some $k \ge t$ since trading a p_j term for a $p_j - 1$ term will always reduce the size of $S(N, k)$. □

However, what is an approximation for t in terms of N, or what is a reasonably close choice greater than t to produce a suitable balancing? We know that $\Pi_1^t p_i > t^t$ since

$$\prod_{1}^{t} p_i \simeq \prod_{2}^{t} i \log i \simeq t! \prod_{2}^{t} \log i \quad \text{and}$$

$$\prod_{2}^{t} \log i = e^{\Sigma_2^t \log i} \simeq e^{\int_1^t \log x\,dx} \simeq e^{x\log x - x|^t} \simeq t^t/e^t.$$

Since by Stirling's formula $t! \simeq (\sqrt{2\pi t})(t/e)^t$, it is clear that $(\sqrt{2\pi t})(t/e)^{2t} > t^t$. Then choosing t such that $t^t = N$ will be a reasonable high approximation of a solution to $\Pi_1^t p_i = N$. If $t^t = N$, then $t\log t = \log N$. Let $t \simeq \log N/\log\log N$. Then

$$t\log t \simeq \frac{\log N}{\log\log N}\log\left(\frac{\log N}{\log\log N}\right)$$
$$\simeq \frac{\log N}{\log\log N}(\log\log N - \log\log\log N) \simeq \log N,$$

which allows us to state (choosing $k = t$):

THEOREM 4. (1) *The storage function $S(N)$, the storage complexity of the algorithm, is $O_A(N/\log\log N)$ and $O_B(N\log N/\log\log N)$.* (2) *The computational complexity of the algorithm is $O_A(N/\log\log N)$ and $O_B(N\log N\log\log\log N)$.*

PROOF. Since $k = t$ and $\Pi_1^t p_i \simeq N$, we can re-evaluate the storage function as

$$S(N, t) \simeq N\prod_{1}^{t}(1 - 1/p_i) + N\left(\prod_{1}^{t} p_i - 1\right)\Big/N \simeq 2N\prod_{1}^{t}(1 - 1/p_i).$$

Now

$$\prod_{1}^{t}(1 - 1/p_i) = e^{\Sigma_1^t \log(1-1/p_i)},$$

and the infinite series expansion for $\log(1 - x)$ for $-1 \le x < 1$ is

$$\log(1 - x) = -x + x^2/2 - x^3/3 + x^4/4 - \cdots.$$

We then approximate $\log(1 - 1/p_i)$ as $-1/p_i$, and hence

$$\prod_{1}^{t}(1 - 1/p_i) \simeq e^{\Sigma_1^t -1/p_i} \simeq e^{-\Sigma_1^t 1/i\log i} \simeq e^{-\log\log t} \simeq 1/\log t.$$

Since $t \simeq \log N/\log\log N$, we then find

$$S(N, t) = 2N\prod_{1}^{t}(1 - 1/p_i) \simeq 2N/\log(\log N/\log\log N) \simeq 2N/\log\log N$$

or

$$S(N) = O_A(N/\log\log N) = O_B(N\log N/\log\log N),$$

which proves claim (1).

Since the storage compression results in a sieve over $O(N/\log\log N)$ integers, there are roughly $(N/\log\log N) - (N/\log N)$ composites, and therefore the arithmetic time complexity is also $O_A(N/\log\log N)$. The analysis of bit time complexity finally yields a result more efficient than the sieve of Eratosthenes as

$$C(N) = O_B((N/\log\log N)\log N\log\log N\log\log\log N) = O_B(N\log N\log\log\log N). \quad \square$$

We are still left with the problem of how to do this preprocessing most efficiently, so that the cost of doing the preprocessing is no greater than that of the algorithm. One solution is to preprocess with the sieve of Eratosthenes up to the $(\log N/\log\log N)$-th prime and then use the algorithm described above. In this case the preprocessing has a time bit complexity of

$$P(N) = \sum_{i=1}^{\log N/\log\log N} N/i\log i \simeq N\log\log x\Big|^{x=\log N/\log\log N}$$
$$\simeq O_A(N\log\log\log N) \simeq O_B(N\log N\log\log\log N),$$

which is the bit complexity of the main algorithm. The sieve for preprocessing runs in $O(N)$ bits and the table T can easily be created in linear $O_B(|T|) = O_B(N\log N/\log\log N)$ time.

An important characteristic of this preprocessing strategy is that it represents a merging of additive and multiplicative sieve methods. While a multiplicative sieve is certainly more efficient from an arithmetic point of view (where addition and multiplication are weighted equally), it is not as desirable as the additive sieve when multiples of small primes are being crossed out. The "wasted motion" of the additive sieve (i.e. repeated cross-outs of the same composite) is still less costly for the smaller primes than introducing the more powerful machinery of multiplication.

7. Further Improvements

An additional reduction in the storage requirement can be made through a particularly neat improvement.[4] The present linked-list structure wastes bits as atoms are deleted. A more efficient use of this storage would be to have it shared by the remaining atoms. This can be done if the *RLINK* and *LLINK* pointers are relative rather than absolute, indicating only how many bits there are until the next atom. Clearly, if atoms a and b are separated by $k - 1$ bits, then this can be expressed in $\log k$ bits. This suggests the following data structure: Let each atom be made up of 3 bits, namely a *MARK* bit, and *RLINK* and *LLINK* bits. $MARK[a] = 0$ if atom a has been deleted; otherwise $MARK[a] = 1$. The list structure is initialized as

```
          a b c
MARK . . . 1 1 1 1 1 1 1 . . .
RLINK . . . 1 1 1 1 1 1 1 . . .
LLINK . . . 1 1 1 1 1 1 1 . . .
```

which indicates that nothing has been deleted, and it is one bit in either direction to adjacent atoms. Deletion of atoms b and c would have the following effect:

```
          a b c
MARK . . . 1 1 1 0 0 1 1 . . .
RLINK . . . 1 1 1 1 0 1 1 . . .
LLINK . . . 1 1 1 0 1 1 1 . . .
```

[4] This resulted from a suggestion by Ronald Rivest.

The relevant bits for *RLINK* (and *LLINK* similarly) can be retrieved by the following algorithm:

```
procedure RIGHTLINK (a):
  H ← 0;
  R ← a;
  while MARK[H + a] = 0 or a = R do (
    H ← H + H + RLINK[R];
    R ← R + 1);
  return(H + a);
end
```

The *RIGHTLINK* algorithm reads the *RLINK* pointer bit by bit, keeping track of the significance of each bit (i.e. what power of 2 it represents). While the pointer has not been fully read, it points to an intermediate atom that has already been sieved out since its *MARK* bit is 0. The pointer is fully retrieved when it points to an atom where the *MARK* bit is 1.

RIGHTLINK is an $O_B(\log a)$ procedure; so it does not raise the bit order of complexity of the sieve algorithm as it is less costly than multiplication. The size of the doubly linked list can be then reduced to $3N/\log \log N$ bits. We note that a list of atoms to be deleted can be stored in $O_B(N/\log \log N)$ bits also. The largest delete list will be generated from multiples of the $(\log N/\log \log N)$-th prime, and

$$p_{\log N/\log \log N} \simeq (\log N/\log \log N)\log(\log N/\log \log N) \simeq \log N$$

Therefore the length of the delete list is at most $N/\log N \log \log N$ atoms. Since a pointer to each atom can be stored in $\log N$ bits, the delete list can be at most $N/\log \log N$ bits.

An immediate consequence of this improvement is that storage for the doubly linked list becomes negligible, as it is of lower order than the size of the table T of residues. The storage result in Theorem 4 is not improved because T cannot be compressed by this method. However, use of this compression technique without preprocessing improves the storage result of the original multiplicative algorithm described in Section 5 to $O_B(N)$.

8. Final Remarks

In summary, an algorithm using $O_B(N\log N \log \log N)$ bit operations and $O_B(N\log N/\log \log N)$ bit storage has been presented where the constant factors for both are small. The algorithm connects an arithmetic upper bound with the theoretical arithmetic lower bound for sieves without preprocessing. The improvements in time and space complexity of the modified final algorithm result from a crossover between additive and multiplicative sieve techniques, as well as further time-space tradeoffs. It would be of interest to see whether some better lower bounds can be obtained in the hope of bringing the upper and lower bounds closer together.

Acknowledgments. This problem was originally proposed to me by David Dobkin in a course on computational complexity given at Yale University during the spring of 1976. I am particularly indebted to him for his enthusiasm and guidance while I was working on this research problem. I also wish to thank Richard Lipton of Yale University and Ronald Rivest and Vaughn Pratt of MIT for their interest and assistance in the extremely useful discussions I had with them.

References

[(Note. References 1, 3, and 6 are not cited in the text.)]

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

2. Chartres, B.A. Algorithms 310-311: Prime number generators 1 and 2. *Comm. ACM 10,* 9 (Sept. 1967), 569-570.

3. Knuth, D.E. *The Art of Computer Programming, Vol. 1: Fundamental Algorithms.* Addison-Wesley, Reading, Mass., 1973.

4. Niven, I., and Zuckerman, H. *An Introduction to the Theory of Numbers.* Wiley, New York, 1960, chap. 2.

5. Singleton, R.C. Algorithms 356-357: Prime number generation using the treesort principle. *Comm. ACM 12,* 10 (Oct. 1969), 563-564.

6. Wood, T.C. Algorithm 35: Sieve. *Comm. ACM 4,* 3 (March 1961), 151.