may be used to establish an authenticated connection to be used conventionally. The intrinsic security requirements of a public-key authentication server are easier to meet than those of a conventional one, but a complete evaluation of the system problems in implementing such a server in a real system, and the need to retain a secure record of old public keys to guarantee future correct arbitration of old signatures may minimize this advantage. We conclude that the choice of technique should be based on the economy and cryptographic strength of the encryption techniques themselves, rather than for their effects on protocol complexity.

Finally, protocols such as those developed here are prone to extremely subtle errors that are unlikely to be detected in normal operation. The need for techniques to verify the correctness of such protocols is great, and we encourage those interested in such problems to consider this area.

Acknowledgments. We are indebted to a number of people who have read drafts of this paper and made careful and helpful comments, notably: Peter Denning, Stockton Gaines, Jim Gray, Steve Kent, Gerry Popek, Ron Rivest, Jerry Saltzer, and Robin Walker.

Received September 1977; revised April 1978; final revision May 1978

References

1. Branstad, D. Security aspects of computer networks, Proc. AIAA Comptr. Network Syst. Conf., April 1973, paper 73-427.
2. Branstad, D. Encryption protection in computer data communications. Proc. Fourth Data Communications Symp., Oct. 1975, pp. 8.1–8.7 (available from ACM, New York).
3. Diffie, W., and Hellman, M. Multiuser Cryptographic Techniques, Proc AFIPS 1976 NCC, AFIPS Press, Montvale, N.J., pp. 109–112.
4. Feistel, H. Cryptographic coding for data bank privacy. Res. Rep. RC2827, IBM T.J. Watson Res. Ctr., Yorktown Heights, N.Y., March 1970.
5. Kent, S. Encryption-based protection protocols for interactive user-computer communication, M.S. Th., EECS Dept., M.I.T., 1976; also available as Tech. Rep. 162, Lab. for Comptr. Sci., M.I.T., Cambridge, Mass., 1976.
6. Kent, S. Encryption-based protection for interactive user/computer communication. Proc. Fifth Data Communication Symp., Sept. 1977, pp. 5-7–5-13 (available from ACM, New York).
7. National Bureau of Standards. Data Encryption Standard. Fed. Inform. Processing Standards Pub. 46, NBS, Washington, D.C., Jan. 1977.
8. Pohlig, S. Algebraic and combinatoric aspects of cryptography. Tech. Rep. No. 6602-1, Stanford Electron. Labs., Stanford, Calif., Oct. 1977.
9. Rivest, R.L., et al. A method for obtaining digital signatures and public-key cryptosystems. *Comm. ACM 21*, 2 (Feb. 1978), 120–126.

Programming Techniques

S. L. Graham Editor

A Linear Sieve Algorithm for Finding Prime Numbers

David Gries
Cornell University

Jayadev Misra
University of Texas at Austin

A new algorithm is presented for finding all primes between 2 and *n*. The algorithm executes in time proportional to *n* (assuming that multiplication of integers not larger than *n* can be performed in unit time). The method has the same *arithmetic complexity* as the algorithm presented by Mairson [6]; however, our version is perhaps simpler and more elegant. It is also easily extended to find the prime factorization of *all* integers between 2 and *n* in time proportional to *n*.

Key Words and Phrases: primes, algorithms, data structures

CR Categories: 5.25, 5.24, 5.29

1. Introduction

An algorithm is presented for finding all primes between 2 and n, for $n \geq 4$, that executes in time proportional to n. Like the sieve of Eratosthenes, it works by removing nonprimes from the set $\{2, \ldots, n\}$. Unlike the sieve of Eratosthenes, no attempt is ever made to remove a nonprime that was removed earlier; this allows us to develop a linear algorithm.

The algorithm deals with sets S satisfying $S \subset \{2, \ldots, n\}$. Two operations will be required on such sets:

This research was partially supported by the National Science Foundation under Grants DCR75-09842 and MCS76-22360.

Authors' addresses: D. Gries, Computer Science Department, Cornell University, Ithaca, NY 14853; J. Misra, Computer Science Department, University of Texas at Austin, Austin, TX 78712.

Table I. Execution of the Algorithm for $n = 27$.

p	q	S																								
2	2	1	2	3	④	5	6	7	⑧	9	10	11	12	13	14	15	⑯	17	18	19	20	21	22	23	24	25	26	27
2	3	1	2	3		5	⑥	7		9	10	11	⑫	13	14	15		17	18	19	20	21	22	23	㉔	25	26	27
2	5	1	2	3		5		7		9	⑩	11		13	14	15		17	18	19	⑳	21	22	23		25	26	27
2	7	1	2	3		5		7		9		11		13	⑭	15		17	18	19		21	22	23		25	26	27
2	9	1	2	3		5		7		9		11		13		15		17	⑱	19		21	22	23		25	26	27
2	11	1	2	3		5		7		9		11		13		15		17		19		21	㉒	23		25	26	27
2	13	1	2	3		5		7		9		11		13		15		17		19		21		23		25	㉖	27
3	3	1	2	3		5		7		⑨		11		13		15		17		19		21		23		25		㉗
3	5	1	2	3		5		7				11		13		⑮		17		19		21		23		25		
3	7	1	2	3		5		7				11		13				17		19		㉑		23		25		
5	5	1	2	3		5		7				11		13				17		19				23		㉕		

remove(S, i) is defined for $i \in S$ and implements $S := S - \{i\}$.
next(S, i) is a function defined only for $i \in S$ such that there is an integer larger than i in S; it yields the next larger integer in S.

In order to achieve linearity, the total time spent executing these operations must be no worse than proportional to n. Thus this algorithm provides an interesting context for a discussion of selection of data structures.

2. The Algorithm

For $i \geq 2$, denote by $lp(i)$ the lowest prime that divides i evenly. The algorithm is based on the following theorem.

THEOREM 2.1. *A nonprime x can be written uniquely as*

$$x = p^k \cdot q$$

where (1) p is prime, $p = lp(x)$; (2) $1 \leq k$; (3) $p = q$ or $p < lp(q)$.

PROOF. By the unique factorization theorem (see, for example, LeVeque [5]), x can be written uniquely as

$$x = p_1^{n_1} \cdot \ldots \cdot p_m^{n_m}$$

where $m \geq 1$, the p_i are primes, $p_i < p_{i+1}$ for $1 \leq i < m$, and $m = 1$ implies $n_1 > 1$. Hence the following yields the only choice for p, q, and k of the theorem:

If $m = 1$, let $p = p_1$, $q = p_1$, and $k = n_1 - 1$. If $m > 1$, let $p = p_1$, $q = p_2^{n_2} \cdot \ldots \cdot p_m^{n_m}$ and $k = n_1$. □

Subsequently, we write $x = \chi(p, q, k)$ to denote that x is nonprime and $x = p^k \cdot q$ where p, q, and k have the properties described in Theorem 2.1.

A prime cannot be written as described in the theorem, so the algorithm to delete nonprimes from S need only produce all combinations of (p, q, k) and delete the corresponding nonprimes $x = \chi(p, q, k)$. The trick is to produce each combination exactly once, and in such an order that the next combination can be efficiently calculated from the current one. For this purpose, we use the total ordering α on nonprimes $x = \chi(p, q, k)$ induced by the lexicographic ordering of the corresponding triples (p, q, k).

DEFINITION 1. *Let* $x = \chi(p, q, k)$ *and* $\bar{x} = \chi(\bar{p}, \bar{q}, \bar{k})$. *Then*

$x \alpha \bar{x} \Leftrightarrow p < \bar{p}$ **or**
$(p = \bar{p}$ **and** $q < \bar{q})$ **or**
$(p = \bar{p}$ **and** $q = \bar{q}$ **and** $k < \bar{k})$.

Table I illustrates this ordering and at the same time depicts how the algorithm works. The rows give successive values for pairs (p, q), together with the contents of the set S before nonprimes with this p and q are deleted. In each row, the nonprimes $x = \chi(p, q, k)$ to be deleted for $k = 1, 2, 3$ have been circled.

The algorithm uses a variable S, which initially contains the set $\{2, \ldots, n\}$ and from which nonprimes are to be deleted, and integer variables p, q, k, and x used to generate nonprimes in the order defined by α. The invariant relation P used in the loop of the algorithm is given in Definition 2. It is not difficult to follow; conditions (1)–(3) describe properties of p, q, and k, condition (4) describes the properties of the value x under consideration for deletion, and condition (5) describes the current contents of set S.

DEFINITION 2.

$P \equiv$ (1) p prime, $4 \leq p^2 \leq n$;
(2) $p = q$ **or** $p < lp(q)$; $p \cdot q \leq n$;
(3) $1 \leq k$;
(4) $x = \chi(p, q, k)$;
(5) $S = \{2, \ldots, n\} - \{y | y$ nonprime **and** $y \alpha x\}$.

The goal of the loop of the algorithm is to have S contain only the primes in $\{2, \ldots, n\}$. The object of each iteration of the loop is to get us closer to this goal, while keeping P invariantly true. We now investigate operations with this property.

If $x = \chi(p, q, k) \leq n$, then clearly x is to be deleted from S, and P can be restored by executing $k, x := k + 1, p \cdot x$.[1] If $x > n$, we need to determine the next nonprime y (say), according to ordering α, to be deleted from S. Remarkably enough, Lemmas 1 and 2 indicate that under suitable conditions $y = p \cdot next(S, q)$, so that one can change p, q, k, x to denote the next nonprime to

[1] A concurrent assignment $x1, x2, \cdots := e1, e2, \cdots$ calls for concurrent evaluation of the ei, followed by simultaneous assignment of the values ei to the variables xi (which must be all different). See [3].

delete by executing $q := next(S, q)$; $k, x := 1, p \cdot q$. Similarly, Lemmas 3 and 4 state the conditions under which $y = next(S, p)$[2]. We shall prove these lemmas in Section 3.

LEMMA 1. *Invariant P implies that next(S, q) is defined, next(S, q) < n, and p < lp(next(S, q)). Writing y = χ(p, next(S, q), 1), we have x α y.*

LEMMA 2. *Suppose (P* **and** *x > n). Write y = χ(p, next(S, q), 1). Then no nonprime z in S satisfies x α z α y.*

LEMMA 3. *Invariant P implies that next(S, p) is defined, next(S, p) < n, and next(S, p) is prime. Writing y = χ(next(S, p), next(S, p), 1), we have x α y.*

LEMMA 4. *Suppose (P* **and** *x > n* **and** *p·next(S, q) > n). Write y = next(S, p)[2]. Then no nonprime z in S satisfies x α z α y.*

We write Algorithm 1 using guarded commands [1]. The arguments necessary to ascertain correctness will be discussed in Section 3. Note that variable k is used only in assignments to itself, so that all references to it may be deleted. It has been included only to clarify the relationship between p, q, and x.

ALGORITHM 1.

```
{n ≥ 4}
p, q, k, x, S := 2, 2, 1, 4, {2, ... , n};
do x ≤ n                              → remove(S, x);
                                        k, x := k + 1, p·x
[] x > n and p·next(S, q) ≤ n         → q := next(S, q);
                                        k, x := 1, p·q
[] x > n and p·next(S, q) > n         and next(S, p)² ≤ n
                                      → p := next(S, p);
                                        q, k, x := p, 1, p·p
od
{S = {y | 2 ≤ y ≤ n and y prime}
```

Algorithm 2 is essentially the same algorithm as Algorithm 1 but written more conventionally. We feel that Algorithm 1 is easier to understand and prove correct; one loop with one invariant is easier to understand in this instance than three nested loops with three invariants.

ALGORITHM 2.

```
p, S := 2, {2, ... , n};
while p·p ≤ n do begin
  q := p;
  while p·q ≤ n do begin
    x := p·q;
    while x ≤ n do begin
      remove(S, x); x := p·x
      end;
    q := next(S, q)
    end;
  p := next(S, p)
  end
```

3. Showing Correctness and Linearity

In this section the proofs of Lemmas 1–4 are given. The axiomatic proof method with respect to guarded commands [1] is also discussed and some of the details of the proof are given.

In preparation for proving Lemmas 1 and 2, we first prove the following.

LEMMA 5. *Consider any nonprime z = χ(p̄, q̄, k̄). We have (P* **and** *x α z* **and** *q̄ ≤ n) implies q̄ ∈ S.*

PROOF. If $\bar{q}$ ($\leq n$) is prime it is in S. Suppose $\bar{q}$ is nonprime. From $x \alpha z$ and the decompositions of x and z we deduce $lp(x) = p \leq \bar{p} < lp(\bar{q})$ so that $x \alpha \bar{q}$. From the definition of S and $x \alpha \bar{q}$ we deduce $\bar{q} \in S$. □

Our proofs of Lemmas 1 and 2 rest on the remarkable fact that for any positive integer $i > 1$ there is a prime v satisfying $i < v < 2i$.[2]

PROOF OF LEMMA 1. Let v be a prime satisfying $q < v < 2 \cdot q$. From P we conclude

$$p \leq q < next(S, q) \leq v < 2 \cdot q \leq p \cdot q \leq n$$

and hence $next(S, q) < n$. Secondly, no nonprime in S has a divisor less than p, so that $p \leq lp(next(S, q))$. To show that $p \neq lp(next(S, q))$, consider the fact that any nonprime $z = \chi(p, \bar{q}, \bar{k})$ in S must have $q \leq \bar{q}$. Hence the smallest such nonprime z that may be in S is $p \cdot q$. Since $next(S, q) < p \cdot q$, $next(S, q)$ cannot be a nonprime with $p = lp(next(S, q))$.

Hence $p < lp(next(S, q))$. The relation $x \alpha y = \chi(p, next(S, q), 1)$ follows immediately.

PROOF OF LEMMA 2. From $x = \chi(p, q, k) > n$ and $y = \chi(p, next(S, q), 1)$, we see that a z in S satisfying $x \alpha z \alpha y$ would have a decomposition $z = \chi(p, \bar{q}, \bar{k})$ with $q < \bar{q} < next(S, q)$. But $\bar{q}$ would not be in S, contradicting Lemma 5.

PROOF OF LEMMA 3. Let v be a prime satisfying $p < v < 2 \cdot p$. From P we have

$$p < next(S, p) \leq v < 2 \cdot p \leq p^2 \leq n$$

and $next(S, p) < n$. Secondly, no nonprime in S has a divisor less than p, so that for all nonprimes $z \in S$ we have $p^2 \leq z$. Since $next(S, p) < p^2$, $next(S, p)$ must be prime. The fact $x \alpha y = \chi(next(S, p), next(S, p), 1)$ follows immediately.

PROOF OF LEMMA 4. This is similar to the proof of Lemma 2 and is left to the reader.

We now discuss the proof method and give some details. The main part of Algorithm 1 is a loop of the form

do $B1 \rightarrow SL1$ [] $B2 \rightarrow SL2$ [] $B3 \rightarrow SL3$ **od**

Showing correctness involves exhibiting an invariant P (ours is given in Definition 2) and an integer function t, and showing that the following hold:

[2] As might be imagined, this is difficult to prove. J. Bertrand conjectured this fact in 1845, after showing empirically that it was true for $i \leq 10^6$. Chebyshev proved the conjecture in 1850 (see [5] for details). Our first draft of a proof and algorithm did not rely on this fact at all. It used weaker lemmas with more complicated proofs and required the additional element $n + 1$ to be in S so that $next(S, i)$ would be sure to be defined. For example, the original Lemma 1 read: *Let y = p·next(S, q). Suppose P* **and** *x > n. Then either next(S, q) = n + 1; or y = χ(p, next (S, q), 1)* **and** *x α y.*

(1) P is true before execution of the loop;
(2) P **and not** ($B1$ **or** $B2$ **or** $B3$) implies the desired result;
(3) $\{P$ **and** $Bi\}$ SLi $\{P\}$ for $i = 1, 2, 3$;
(4) Execution of the loop terminates:
  (a) (P **and** ($B1$ **or** $B2$ **or** $B3$)) $\Rightarrow t \geq 0$
  (b) Execution of SLi reduces t (for $1 \leq i \leq 3$). Using an extra variable T, this means that
    $\{P$ **and** $Bi\}\, T := t;\ SLi\ \{t \leq T - 1\}$.

Point (1) is obvious; point (2) we leave to the reader, since it can be shown quite easily with the help of Lemma 4. Point (3) concerns the invariance of P under execution of each guarded command SLi. The only difficulty concerns the generation of new values for q, p, and x to satisfy P. Lemmas 1–4 yield the necessary facts.

To see this a bit more formally in at least one case, consider determining the precondition Q in $\{Q\}$ $SL3$ $\{P\}$ where $SL3$ is the third guarded command list of the loop of Algorithm 1 and P is in Definition 2. $SL3$ is a sequence of assignments, so we apply the normal assignment and concatenation rules to arrive at:

$Q \equiv$ (1) $next(S, p)$ prime, $4 \leq next(S, p)^2 \leq n$;
(2) $next(S, p) = next(S, p)$ *or* $(next(S, p) < lp(next(S, p))$; $next(S, p)^2 \leq n$;
(3) $1 \leq 1$;
(4) $next(S, p)^2 = \chi(next(S, p), next(S, p), 1)$;
(5) $S = \{2, \ldots, n\} - \{y | y$ nonprime **and** $y\ \alpha\ next(S, p)^2\}$.

It is then a simple matter to prove that (P **and** $B3$) implies Q, with the help of Lemmas 3 and 4. To show termination, we use the function t: the number of nonprimes $z \in S$ satisfying ($x = z$ **or** $x\ \alpha\ z$) plus the number of nonprimes $z \in S$ satisfying $x\ \alpha\ z$.

Note that $t \geq 0$. Execution of the first guarded command $SL1$ reduces the first term of t by at least one, since it removes x from S. Execution of the second or third guarded command begins with $x = x0$ (say) and $x \notin S$ and finishes with $x0\ \alpha\ x$ and $x \in S$; hence they reduce the second term by one. Hence we conclude that the algorithm terminates.

The initial value for t is a bound on the number of times the loop will iterate. The initial value is bounded by $2 \cdot$(number of nonprimes in S) $< 2 \cdot n$. Hence the algorithm is linear if we can satisfactorily implement S and the operations on it. The implementation is the subject of Section 4.

4. Implementing the Set S

We discuss three approaches to implement $S \subset \{2, \ldots, n\}$, all dealing with forms of linked lists. We will actually implement sets $S \cup \{n + 1\}$. The purpose is to provide an "anchor" for one end of the linked list. The integer 2 serves the same purpose at the other end of the list since it is never deleted.

Approach 1. If we implement S as a doubly linked list, then $remove(S, i)$ and $next(S, i)$ each can be performed in constant time. Thus we use

var s: **array** $(2{:}n + 1)$ **of record** ($pred$, $succ{:}integer$)

where the various parts of s are used as follows:

$s(i).succ = next(S \cup \{n + 1\}, i)$ for $i \in S$;
$s(i).pred \equiv$ unique integer j such that $next(s, j) = i$, for $i \in S \cup \{n + 1\}$, $i \neq 2$.

At any time, the elements of S can be found by following the successor chain beginning at $s(2)$ and ending just before $s(n + 1)$. This approach requires roughly $2n$ locations (each of ln n bits).[3] The three operations on S are:

```
S := {2, ..., n} :: i := 1;
                     do i < n → i := i + 1; s(i).succ := i + 1;
                                            s(i + 1).pred := i od;
remove(S, i)     :: s(s(i).pred).succ := s(i).succ;
                    s(s(i).succ).pred := s(i).pred;
next(S, i)       :: s(i).succ
```

Approach 2.[4] Under the assumptions that 2 is not removed from S and $remove(S, i)$ is only executed if i is in S, a singly linked list can be used to implement $S \in \{2, \ldots, n\}$. We use an array s:

var s: **array** $(2..n)$ **of** integer.

If $i \in S$, then $s(i)$ will contain its successor $next(S \cup \{n + 1\}, i)$. It will also be necessary to determine i's predecessor. Clearly, if $i \in S$ and $(i - 1) \in S$, then $i - 1$ is i's predecessor and $s(i - 1) = i$. On the other hand, if $i \in S$ and $(i - 1) \notin S$, the element $s(i - 1)$ can be used to contain i's predecessor. Note then that for $i > 2$, $i \in S$, i's predecessor is given by min $(i - 1, s(i - 1))$. Thus the array s above is used to describe S as follows:

$2 \in S$;
for $i \in S$, $s(i) = next(S \cup \{n + 1\}, i)$;
for $i \in S$, $i \neq 2$, the predecessor i is min $(i - 1, s(i - 1))$.

At any time S is the set of values 2, $s(2)$, $s(s(2))$, ... , up to but not including the value $n + 1$.

The three operations $S := \{2, \cdots, n\}$, $remove$ (S, i), and $next$ (S, i) are then

```
S := {2, ..., n} :: i := 1;
                    do i < n → i := i + 1; s(i) := i + 1 od;
remove(S, i)     :: pred := min (i − 1, s(i − 1)); [pred is local]
                    s(pred) := s(i);
                    s(s(i) − 1) := pred;
next(S, i)       :: s(i)
```

Approach 3. In Approach 2, the value $s(i)$ always contains an integer representing a successor or predecessor j of value i in the set. Instead, one could store the *increment* needed to get from i to j; that is, the value $i - j$. Since the increment will in general be much smaller

[3] Throughout, ln refers to he base 2 logarithm.

[4] In our initial formulation of Approach 2, we used an extra bit vector in, where $in(i)$ denoted whether or not $i \in S$. $Remove(S, i)$ simply set $in(i)$ to false. However, $next(S, i)$ was a complicated operation that in order to achieve a linear algorithm, had the "benevolent side effect" of changing the representation of S without changing the values of $next(S, i)$ for *any* i. During a presentation of the algorithm at Harvard, Norman Cohen proposed a technique that was further refined by the authors to yield the present Approach 2. The development of $next(S, i)$ in the case that $remove(S, i)$ simply sets $in(i)$ to false might be an interesting exercise for the reader.

than n, we may be able to reduce the number of bits needed for each $s(i)$. Asymptotically speaking, there are $n/\ln n$ or more primes in $\{2, \cdots, n\}$, and if they were evenly distributed the increment to get from one to another would not be greater than $\ln n$. Hence we would need only *ceil*(ln ln n) bits for each $s(i)$ instead of *ceil*(ln n).

Unfortunately, the primes are not evenly distributed, so we cannot assume $s(i)$ will be so small. Knuth [4, p. 402] gives a table of "record breaking" gaps between prime numbers. For primes less than or equal to 20831533 we see that the largest gap is 210, so that eight bits will suffice for $n \leq 20831533$.

5. Discussion

Mairson's [6] paper, which tied for second place in the annual George E. Forsythe Student Paper Competition, also presents a linear sieve algorithm. To delete from S all composite integers whose lowest prime factor is p, Mairson's algorithm first uses the set S to compile a list of these integers, then sequences through this list to delete them from S. The use of an auxiliary list is unnecessary in our algorithm because of Theorem 2.1. Mairson does an excellent job in analyzing the efficiency of his algorithm, and most of his analyses will carry over to our algorithm.

Algorithm 1 also works for $n = 2$ and $n = 3$; the condition $n \geq 4$ is used only to simplify the proof.

Dexter Kozen has discovered an easy way to extend the algorithm to find the complete factorization of an integer n in no worse than linear time. This extension is not surprising, since Shank's [8] algorithm finds factors of n in time $0(n^{(1/4+\epsilon)})$ for $\epsilon > 0$. However, Kozen's technique actually can be used to build a table in time n that yields the complete factorization of *all* integers between 2 and n! It is quite simple. Assume a singly linked list is used to implement set S, say using Approach 2, and use three new arrays xp, xk, $xq(2, \ldots, n)$. Initially, let $xp(x) = x$, for all x. When a nonprime $x = p^k \cdot q$ is about to be deleted in Algorithm 1 the values p, k, and q are available, so just record them in $xp(x)$, $xk(x)$, and $xq(x)$. Upon termination, for each i, $2 \leq i \leq n$, if $xp(i) < i$ then i is not prime, i's lowest prime factor is in $xp(i)$ and its multiplicity in $xk(i)$, while the other factors can be determined from $xq(i)$ in a similar fashion.

The development of this algorithm emphasizes several points. First, it could not have been developed without recognition of an important property of nonprimes—their unique decomposition given in Theorem 2.1. Efficient algorithms come less from clever tricks than from a good understanding of properties of the values being manipulated. Secondly, the correctness of the algorithm rests on some nontrivial mathematical theorems (Lemmas 1–4). Once these theorems are understood, the algorithm itself seems quite simple. We see here a distinction between the complexity of an algorithm and the complexity of its mathematical underpinnings, two quite different things. Finally, the algorithm provides a good basis for a discussion of control structures and programming style—we showed two ways of writing the algorithm—and for a discussion of the selection of data structures.

Acknowledgments. We note that Gale and Pratt [2] have discovered a different linear sieve algorithm. The authors would like to thank Jim Donahue, Don Knuth, and Gary Levin for carefully reading earlier drafts of this paper and for providing many constructive criticisms; and Norman Cohen and Dexter Kozen for their suggestions.

Received June 1977; revised April 1978

References

1. Dijkstra, E.W. Guarded commands, nondeterminacy and formal derivation of programs. *Comm. ACM 18*, 8 (Aug. 1975), 453–457.
2. Gale, R., and V. Pratt. CGOL—an algebraic notation for MACLISP users. Working paper, M.I.T. AI Lab., Cambridge, Mass., January 1977.
3. Gries, D. The multiple assignment statement. *IEEE Trans. Software Eng. SE-2* (March 1978), 89–93.
4. Knuth, D. *The Art of Computer Programming, Vol. 3: Sorting and Searching.* Addison-Wesley, Reading, Mass., 1973.
5. LeVeque, W.J. *Topics in Number Theory, Volume I.* Addison-Wesley, Reading, Mass., 1956.
6. Mairson, H.G. Some new upper bounds on the generation of prime numbers. *Comm. ACM 20*, 9 (Sept. 1977), 664–669.
7. Miller, G.L. Riemann's hypothesis and tests for primality. Proc. Seventh Annual ACM Symp. Theory of Computing, 1975, 234–239.
8. Shank, D. Class number, a theory of factorization and Genera. Proc. Symp. in Pure Mathematics 20, 1969, Amer. Math. Soc., Providence, R.I., 1971, 415–440.