Edgar H. Sibley
Panel Editor

Based on the sieve of Eratosthenes, a faster and more compact algorithm is presented for finding all primes between 2 and N.

A PRACTICAL SIEVE ALGORITHM FOR FINDING PRIME NUMBERS

XUEDONG LUO

In the third century B.C., the Greek astronomer and geographer Eratosthenes described an algorithm for the problem of finding all primes between 2 and N by removing nonprimes from the set $\{2, \ldots, N\}$. Each nonprime is found once for each of its prime factors (not greater than $\sqrt{N}$). It is established at $O(N \bullet \log \log N)$ additions and proved to be one of the most effective sieves of algorithms until now [3].

In recent years a number of significant advances about sieve algorithms have been made. Mairson described a primal sieve that executes in linear time [4]. Gries and Misra discovered another linear multiplicative sieve [2].

Pritchard [5] introduced a surprising method for improving multiplicative sieve and modified Mairson's algorithm to achieve an $O(N/\log \log N)$ additive sieve basing it on a simple fact:

$$(p_1 \bullet p_2 \cdots p_i, \quad p_1 \bullet p_2 \cdots p_i + k) = 1 \quad \textbf{iff} \quad (k, p_j) = 1$$

where p_i denotes ith prime and $1 \le j \le i$ [5].

But all the above sieve algorithms have more theoretical significance than practical significance. In this article we will present a new sieve algorithm that has not only the same complexity as Eratosthenes' sieve, but also more practical significance.

A PRACTICAL ALGORITHM

Before we talk about this algorithm we first describe the sieve of Eratosthenes as follows [3]:

Algorithm 1.
{We may assume that $N = 2M$ is even}
  $j, k, p, q, S := 1, 1, 3, 4, \quad \{3, 5, \ldots, 2M - 1\}$;
(a) **If** $S[j] = 0$, **go to** (c).
  Otherwise $k \leftarrow q$.
(b) **If** $k \le M$,
  then $S[k] \leftarrow 0$, $k \leftarrow k + p$
  and *repeat this* **step**.
(c) $j \leftarrow j + 1$, $p \leftarrow p + 2$, $q \leftarrow q + 2p - 2$.
  If $q < M$, *return* **to** (a).
$S = \{x \mid x$ is zero **or** x is prime $\le N\}$.

Looking at Algorithm 1, we can see it suggests that array S avoids all even $\le N$, or it avoids every composite that has 2 as one of its prime factors. In the same way, we can avoid all composites that have 2 or 3 as one of their prime factors. Let $S = \{5, 7, 11, 13, \ldots 3i + 2, 3(i + 1) + 1, \ldots, N\}$ (where i is odd).

Tables Ia and Ib suggest that multiples of 5 increase regularly; the interval between two close multiples of 5 is 3 or 7, and we can therefore remove all the multiples of 5 by using additions only. The same thing applies to 7 and the interval is 9 or 5. But how can we find all the intervals?

Define several signs and operations as follows:

{We suppose that $N = 3M + 2$, where M is odd}
  ko: odd;
  ke: even;
  S: $\{5, 7, 11, \ldots, N\}$;
  C_k: $[S[k]/3]$
    {the position **of** square **of** kth element in S}
$I(i, j)$: $[S[i] \bullet S[j + 1]/3] - [S[i] \bullet S[j]/3]$
  {the interval between two close multiples **of** kth element in S}

TABLE Ia. Composites Appearing Regularly in S

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	[illegible]	[illegible]	[illegible]
$s[i]$	5	7	11	13	17	19	23	[25]	29	31	[35]	37	41	43	47	(49)	53	[55]	59

TABLE Ib. Composites Appearing Regularly in S

i	20	21	22	23	24	25	26	27	28	29	30	31	32	33	[illegible]	[illegible]	36
$s[i]$	61	[65]	67	71	73	(77)	79	83	[85]	89	(91)	[95]	97	101	103	107	109

To avoid the multiplication, we use a trick to produce the position of the square of the next element from the current one. It is clear that $S[ko] = 3ko + 2$ and $S[ke] = 3ke + 1$. So we obtain:

$$C_{ke+1} - C_{ke} = [(3(ke + 1) + 2)/3] - [(3ke + 1)/3]$$
$$= 8\ (ke + 1)$$
$$C_{ko+1} - C_{ko} = [(3(ko + 1) + 1)/3] - [(3ko + 2)/3]$$
$$= 4\ (ko + 1).$$

And hence:

$$C_{ke+1} = C_{ke} + 8(ke + 1)$$
$$C_{ko+1} = C_{ko} + 4(ko + 1)$$

Otherwise, we have the relation:

$$I(i, j) = [S[i] \bullet S[j + 1]/3] - [S[i] \bullet S[j]/3]$$
$$= \begin{cases} 2i + 1 & j \text{ is odd} \\ 4i + 1 & i, j \text{ is even} \\ 4i + 3 & i \text{ is odd, } j \text{ is even.} \end{cases}$$

Now, the idea of the algorithm is easy to understand. The new algorithm is given below; it uses an array S which is initially set to $\{5, 7, 11, \ldots, 3i + 2, 3(i + 1) + 1, \ldots, N\}$ (where i is odd) and from which nonprimes are set to zero as they are sieved out. It is written using guarded commands [1].

Algorithm 2.

```
{Supposing that N has the form of 3M + 2, where M is
 odd}; i, q, S := 1, √N/3,  {5, 7, 11, . . . , N};
do i ≤ q → put the position of square of ith element
           into C_i; set zero to all multiples of ith ele-
           ment beginning from C_i;
   i := i + 1
od
{S = {x | x is zero or prime ≤ N}}
```

C_i is defined as in the front of this section.

Let us implement the unrefined statement in the loop body.

```
"put . . . "
C_i = 0;
if i is odd → C_i := C_{i-1} + 8i
□i is even → C_i := C_{i-1} + 4i
fi
{C_i is the position of square of ith element in S}.
```

"set . . . " is implemented as follows:

```
j := C_i;
do j ≤ M → S[j] := 0;
  if j is odd → j := j + 2i + 1
  □i, j is even → j := j + 4i + 1
  □i is odd, j is even → j := j + 4i + 3
  fi
od
```

Algorithm 3 is essentially the same algorithm as Algorithm 2 but written more conventionally. Algorithm 3 uses several tricks for avoiding both multiplication and logical arithmetic comparison operations. It is, therefore, less readable than Algorithm 2. Note that variables k and t are used to shun the comparison phase so that the algorithm is more compact.

Algorithm 3.

```
c, k, t, q, M, S := 0, 1, 2, √N/3, N/3,  {5, 7, 11, . . . , N};
for i := 1 to q do begin
  k := 3 − k; c := c + 4k • i; j := c;
  ij := 2i • (3 − k) + 1; t := t + 4k;
  while j ≤ M do begin
    S[j] := 0; j := j + ij; ij := t − ij
  end
end
```

Algorithm 3 was tested on the VAX-11 computer. Table II presents a contrast between Eratosthenes' sieve and Algorithm 3.

DISCUSSION

Clearly, the arithmetic complexity of computation and the storage requirement for Algorithm 2 is $O(N \bullet \log\log N)$ and $O(N)$—the same as the sieve of Eratosthenes (note: $O(N \bullet \log\log N) - N/6 = O(N \bullet \log\log N)$). But the new algorithm is faster and more compact (needs $N/3$ auxiliary storage in fact) and is therefore more practical than Eratosthenes' sieve. All other algorithms [2, 4, 5] to which we refer have more theoretical significance. Otherwise, Algorithm 2 is simple and easy to

TABLE II. Contrast between Eratosthenes' Sieve and Algorithm 3 (Time in seconds)

N =	5000	50000	500000	1000000	2000000
E's sieve	0.75	0.92	3.55	6.83	64.78
Alg. 3	0.75	0.81	2.83	5.19	16.97

understand. The attempt to avoid all the multiples of 5, 7, . . . initially setting it to S may cause an increase of comparative operations that actually lowers the efficiency of the algorithm.

REFERENCES

1. Dijkstra, E.W. Guarded commands, nondeterminacy and formal derivation of programs. *Commun. ACM 18*, 8 (Aug. 1975), 453–457.
2. Gries, D., and Misra, J. A linear sieve algorithm for finding prime numbers. *Commun. ACM 21*, 12 (Dec. 1978), 999–1003.
3. Knuth, D. *The Art of Computer Programming*. Vol. 2, 2d ed. 1981, p. 394.
4. Mairson, H.G. Some new upper bounds on the generation of prime numbers. *Commun. ACM 20*, 9 (Sept. 1977), 664–669.
5. Pritchard, P. A sublinear additive sieve for finding prime numbers. *Commun. ACM 24*, 1 (Jan. 1981), 18–23.

CR Categories and Subject Descriptors: E.1 [**Data**]: Data Structures—*array, tables*; E.4 [**Data**]: Coding and Information Theory—*data compaction and compression*; G.4 [**Mathematics of Computing**]: Mathematical Software—*efficiency*

General Terms: Algorithms, Design

Additional Key Words and Phrases: Algorithms, prime numbers, sieve

ABOUT THE AUTHOR:

XUEDONG LUO is from Beijing in the People's Republic of China. He is currently a visiting scholar at the Institute of Mathematics and Computer Science at the University of Neuchatel. His research interests include algorithms and prime numbers. Author's Present Address: Route de Beaumont 1, 2068 Hauterive, Switzerland.

Sanden (*continued from p. 343*)

ABOUT THE AUTHOR:

BO SANDEN is an associate research professor in the Department of Information Systems and Systems Engineering, George Mason University. His current research interests are in the continued development of entity-life modeling, its application to various real-world examples, and its comparison with other approaches to real-time software design. Author's Present Address: Department of Information Systems and Systems Engineering, George Mason University, Fairfax, VA 22030-4444. BSANDEN@GMUVAX.GMU.EDU.