ALGORITHM 35
SIEVE
T. C. Wood
RCA Digital Computation and Simulation Group, Moores-
town, New Jersey

procedure Sieve (Nmax) Primes: (p);
    integer Nmax; integer array p;
comment Sieve uses the Sieve of Eratosthenes to find all prime
    numbers not greater than a stated integer Nmax
    and stores them in array p. This array should be
    of dimension 1 by entry (2 x Nmax/f(n(Nmax)))
begin n, i, j := 1;
for n := 3 step 2 until Nmax do
    i := 3;
    L1: go to if p[i] <= sqrt(n) then a1 else a2;
    a1: go to if n/p[i] = n + p[i] then b1 else b2;
    b2: i := i + 1; go to L1;
    a2: p[i] := n; j := j + 1;
    b1: end end

CERTIFICATION OF ALGORITHM 35
SIEVE (T. C. Wood, Comm. ACM, March 1961)
P. J. BROWN
University of North Carolina, Chapel Hill, N. C.

SIEVE was transliterated into GAT for the UNIVAC 1105
and successfully run for a number of cases.

The statement:
    go to if n/p[i] = n + p[i] then b1 else b2;
was changed to the statement:
    go to if n/p[i] - n + p[i] < .5/Nmax then b1 else b2;
Roundoff error might lead to the former giving undesired results.

CERTIFICATION OF ALGORITHM 35
J. S. HILLMORE
Elliott Bros. (London) Ltd., Borehamwood, Herts.,
England

The statement:
    go to if n/p[i] = n + p[i] then b1 else b2;
was changed to the statement:
    go to if (n + p[i]) x p[i] = n then b1 else b2;
This avoids any inaccuracy that might result from introducing
real arithmetic into the evaluation of the relation.

The modified algorithm was successfully run using the Elliott
Algol translator on the National-Elliott 805.

REMARKS ON:

ALGORITHM 35 [A1]

ALGORITHM 310 [A1]
PRIME NUMBER GENERATOR 1 [B. A. Chartres,
Comm. ACM. 10 (Sept. 1967), 509]

ALGORITHM 311 [A1]
PRIME NUMBER GENERATOR 2 [B. A. Chartres,
Comm. ACM. 10 (Sept. 1967), 570]

B. A. CHARTRES (Reed, 13 Apr. 1967)
Computer Science Center, University of Virginia,
Charlottesville, Virginia

The three procedures SIEVE(m,p), SIEVE1(m,p), and SIEVE2(m,p),
which all perform the same operation of putting the primes less
than or equal to m into the array p, were tested and compared for
speed on the Burroughs B5500 at the University of Virginia.
The modification of SIEVE suggested by J. S. Hillmore [Comm. ACM 5
(Aug. 1962), 438] was used. It was also found that SIEVE could be
speeded up by a factor of 1.95 by avoiding the repeated evaluation
of sqrt(n). The modification required consisted of declaring an
integer variable s, inserting the statement s := sqrt(n) immediately
after i := 3, and replacing p[i] <= sqrt(n) by p[i] <= s.

The running times for the computation of the first 10,000 primes
were:

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIEVE (Algorithm 35)</td>
<td>845 sec</td>
</tr>
<tr>
<td>SIEVE (modified)</td>
<td>431 sec</td>
</tr>
<tr>
<td>SIEVE1</td>
<td>220 sec</td>
</tr>
<tr>
<td>SIEVE2</td>
<td>91 sec</td>
</tr>
</tbody>
</table>

The time required to compute the first k primes was found to be,
for each algorithm, remarkably accurate represented by a power
law throughout the range 500 ≤ k ≤ 50,000. The running time of
SIEVE varied as k^{1/n}, that of SIEVE1 as k^{1.8}, and that of SIEVE2 as
k^{1.9}. Thus the speed advantage of SIEVE2 over the other algorithms
increases with increasing k. However, it should be noted that
SIEVE2 took approximately 33 minutes to find the first 100,000
primes, and, if the power law can be trusted for extrapolation past
this point (there is no reason known why it should be), it would
take about 12 hours to find the first million primes.