BIT 17 (1977), 121–127

THE SEGMENTED SIEVE OF ERATOSTHENES AND PRIMES IN ARITHMETIC PROGRESSIONS TO 10^{12}

CARTER BAYS and RICHARD H. HUDSON

Abstract.

The sieve of Eratosthenes, a well known tool for finding primes, is presented in several algorithmic forms. The algorithms are analyzed, with theoretical and actual computation times given. The authors use the sieve in a refined form (the "dual sieve") to find the distribution of primes in twenty arithmetic progressions to 10^{12}. Tables of values are included.

1. Introduction.

The sieve of Eratosthenes is a well-known technique for finding all primes up to a given value, x. To apply the sieve, we first write down all the integers from 1 to x. We then cross out all multiples of 2, all multiples of 3, all multiples of 5, etc., until we have crossed out all multiples of primes not exceeding $\sqrt{x}$. The remaining integers are all the primes between $\sqrt{x}$ and x and, if we take care not to cross out 2, 3, 5, etc. (the primes $\leqq \sqrt{x}$) our sieve gives us all the primes up to x.

The sieve of Eratosthenes is computationally exceedingly fast; just how fast is easily shown below. Let $p_i =$ the ith prime, $\pi(x) =$ the number of primes not exceeding x. Then the time, t, in term of "cross out" operations we must make is given by

$$t = x(1/2+1/3+1/5+\ldots+1/p_k)$$

where $p_k =$ largest prime not exceeding $\sqrt{x}$. Thus

$$t = x \sum_{i=1}^{\pi(x^{\frac{1}{2}})} 1/p_i \sim x \log\log(x^{\frac{1}{2}}) .$$

Hence as x increases, the computation time for any interval of given length increases only as $\log\log(x^{\frac{1}{2}})$. From a practical standpoint, on a digital computer with a word size of, say, 32 bits, the computation time is essentially linear with x. (E.g. for $x = 2^{15}$, $\log\log(x^{\frac{1}{2}}) \doteq 1.75$ and for $x = 2^{31}-1$, $\log\log(x^{\frac{1}{2}}) \doteq 2.37$).

Received August 11, 1976. Revised Feb. 15, 1977.

2. Extending the Sieve.

Of course we have made the assumption that our sieve size is x. This assumption is obviously unreasonable, since it would require an absurd amount of computer memory for even "small" values of x such as 10^7 or 10^8. Fortunately, this situation can be resolved at a small expense in computer time. The idea is to create a predetermined fixed sieve size, Δ. Then since we observe that $x = x_0 + k\Delta$ we may calculate all the primes up to x by finding all the primes between x_0 and $x_0+\Delta$, between $x_0+\Delta$ and $x_0+2\Delta$, etc. Our modified sieve is complicated only by the fact that at the jth sifting, where we are concerned with numbers in the range

$$x_0+(j-1)\Delta \leqq y < x_0+j\Delta$$

we must find, for each $p_i \leqq (x_0+j\Delta)^{\frac{1}{2}}$ the starting point, y_{p_i}, which is the smallest y within the above range such that $p_i | y_{p_i}$. We then proceed by striking out $y_{p_i}, y_{p_i}+p_i, y_{p_i}+2p_i, \ldots$ until all multiples of p_i within the range

$$x_0+(j-1)\Delta \leqq y_{p_i}+kp_i < x_0+j\Delta, \quad k = 0,1,\ldots$$

have been crossed out.

It is advantageous, as will be shown later, to use as large a value for Δ as is practical; also, by dealing only with the odd integers, we may effectively multiply the size of Δ by 2. We are now ready to state a version of the segmenting algorithm.

3. Algorithm A: The Segmented Sieve.

Let $p_i =$ the ith prime, and let $\mathrm{mod}_a(b)$ denote $b(\mathrm{mod}\, a)$. Also, let array S be composed of elements $s_1, s_2, \ldots, s_\Delta$, initially zero, and let $x_0 =$ an odd integer $> \Delta^{\frac{1}{2}}$. Then to find all primes between x_0 and $x_0+2\Delta$, execute the following algorithm for $i = 2,3,4,\ldots \pi\left((x_0+2\Delta)^{\frac{1}{2}}\right)$

1) [calculate starting point]
   Set $y = \mathrm{mod}_{p_i}(p_i - \mathrm{mod}_{p_i}[(x_0-p_i)/2]) + 1$.
   [y is now the starting point for sifting out multiples of p_i]

2) [cross out multiples of p_i]
   For $j = y, y+p_i, y+2p_i \ldots$ up to $j \leqq \Delta$ set $s_j = 1$.

Upon completion, each $s_j = 0$ corresponds to a prime $p = x_0 + 2(j-1)$.

Hence, to complete the algorithm and set up the sieve for the next iteration, execute the following for $j=1,2,3,\dots \Delta$,

3) if $s_j=1$, then set $s_j=0$; otherwise
4) $x_0+2(j-1)$ is a prime.

Now, we may set $x_0=x_0+2\Delta$ and repeat the algorithm.

3.1. *Execution Time on a Digital Computer.*

We shall now obtain an approximation for the computation time necessary to calculate all of the primes between x_0 and $x_0+2\Delta$. The four steps of the algorithm comprise four distinct processes taking place during the sifting. Associate with each step a constant, $k_i (i=1,2,3,4)$, which represents the computation time necessary to effect one iteration of that step. Assume that $x_0 \gg \Delta$. Then the time t_1 spent in step (1) is given by

$$t_1 = k_1\pi(x_0^{\frac{1}{2}}) \sim k_1x_0^{\frac{1}{2}}/\log(x_0^{\frac{1}{2}}) .$$

Our expression for t_2 is similar to the expression derived in section 1.0 except we are not looking at multiples of 2 (i.e. p_1 is excluded from the sum). Hence,

$$t_2 = k_2\Delta \sum_{i=2}^{\pi((x_0+2\Delta)^{\frac{1}{2}})} 1/p_i \sim k_2\Delta(\log\log(x_0^{\frac{1}{2}})-\tfrac{1}{2}) .$$

Here we ignore the fact that for i approaching $\pi((x_0+2\Delta)^{\frac{1}{2}})$ our starting point, y_{p_i} might be greater than Δ. This is particularly likely when $p_i \gg \Delta$, but is overshadowed by the fact that when $p_i \gg \Delta, t_1$ becomes noticeable. Completing our evaluation, t_3 represents the time to "query" and reset the sieve and is given by

$$t_3 = k_3\Delta ,$$

while t_4 is the time required to "process" each prime (determine membership in an arithmetic progression, etc.). Since at a given x_0 the density of the primes is $\sim 1/\log(x_0)$ we observe that

$$t_4 \sim k_4\Delta/\log x_0 .$$

Thus, the total computation time, T, is simply given by

$$T = t_1+t_2+t_3+t_4 .$$

4. Refinements to the Segmented Sieve.

Several modifications may be made to algorithm A which will speed up the sieve operation considerably. For example, note that step 1 requires at least one division, which on most computers is a very slow operation — slowed down further when a double precision divide is used. Thus, on the IBM 370/158 when x_0 exceeds $2^{31}-1$ ($\approx 2.147\times 10^9$) a double precision floating point division is required, along with a double precision subtract. The execution time for these two operations totals 25.5 μsec. Altering step 1 to require only a subtract and a store operation will speed the execution to 1.5 μsec.

This refinement is not as difficult as one might suspect. We need merely maintain a value, L_i, along with the ith prime. Then modify algorithm A as shown below.

Algorithm B

Initially set $s_j=0$, $j=1,2,3,\ldots\Delta$ and set $m=1$.

1) [Initialize the necessary L_i]
   For $m<i\leqq\pi((x_0+2\Delta)^{\frac{1}{2}})$
   Set $L_i=\mathrm{mod}_{p_i}(p_i-\mathrm{mod}_{p_i}[(x_0-p_i)/2]+1)$. Then
   set $m=\pi((x_0+2\Delta)^{\frac{1}{2}})$.

Now, to locate all primes between x_0 and $x_0+2\Delta$ perform steps 2-4 for $i=2,3,4,\ldots m$.

2) Set $y=L_i$.
3) Set $s_j=1$ for $j=y,y+p_i,y+2p_i$ up to $j\leqq\Delta$. Exit this step as soon as $j>\Delta$.
4) Set $L_i=j-\Delta$. [This resets L_i for the next sieve operation].

Upon completion of steps 2-4, we query and re-initialize the sieve as in Algorithm A. For $j=1,2,3,\ldots\Delta$:

5) if $s_j=1$, set $s_j=0$; otherwise
6) $x_0+2(j-1)$ is prime.

We may now set $x_0=x_0+2\Delta$ and return to step 1.

4.1. *Further Refinements.*

It is possible to modify the sieve further to obtain another improvement in speed and a more precise picture of how the algorithm operates. Let $p_b\leqq\Delta<p_{b+1}$ and let $x_0\geqq p^2_{b+1}$. Now when we apply algorithm B note that for $2\leqq i\leqq b$ there is at least one element of S which will be accessed (i.e. there exists at least one j_i such that $1\leqq j_i\leqq\Delta$.) This means that for all values of i up to b we need not check to see if $L_i\leqq\Delta$ since we know this to be true. Similarly, for all $i>b$ we must check to see if $L_i\leqq\Delta$; if not then we do not access S for this p_i at this sifting. On the other hand when $L_i \leqq \Delta, i > b$ then we know that we access S exactly once. This suggests the possibility of breaking up the sieve operation into two phases, which are given in the algorithm below.

Algorithm C (The Dual Sieve).

(Assume $x_0 \geqq p^2_{b+1}$ and $p_b \leqq \Delta < p_{b+1}$; for example, $\Delta = 250{,}000$, $p_b = 249989, b = 22044$ and $x_0 = 10^{11}$.) Initially, set $s_j = 0$, $j = 1, 2, 3, \ldots \Delta$ and set $m = 1$.

1) [Initialize the necessary L_i] For $m < i \leqq \pi((x_0 + 2\Delta)^{\frac{1}{2}})$ set $L_i = \operatorname{mod}_{p_i}(p_i - \operatorname{mod}_{p_i}[(x_0 - p_i)/2]) + 1$. Then set $m = \pi((x_0 + 2\Delta)^{\frac{1}{2}})$.

Now, to locate all primes between x_0 and $x_0 + 2\Delta$ first perform steps 2-4 for $i = 2, 3, 4, \ldots b$.

2) Set $j = L_i$
3) Set $s_j = 1$, set $j = j + p_i$ and if $j \leqq \Delta$ repeat step 3
4) Set $L_i = j - \Delta$.

Next, perform steps 5-7 for $i = b+1,\ b+2,\ b+3, \ldots m$.

5) Set $j = L_i$
6) If $j > \Delta$, set $L_i = j - \Delta$, otherwise
7) Set $s_j = 1$, set $L_i = j + p_i - \Delta$.

[We are now ready to query the sieve] Perform steps 8-9 for $j = 1, 2, 3, \ldots \Delta$.

8) If $s_j = 1$ set $s_j = 0$, otherwise
9) $x_0 + 2(j-1)$ is a prime.

Now, set $x_0 = x_0 + 2\Delta$ and go to step 1.

We can save a little time in steps 8-9 by introducing a binary variable, F, which will alternate between one and zero each sifting. Initially set each element of $S = (1 - F)$. Then at steps 3 and 7 set $s_j = F$. Steps 8-9 can now be combined:

8) For $j = 1, 2, 3, \ldots \Delta$
if $s_j = (1 - F)$ then set $s_j = F$ and $x_0 + 2(j-1)$ is prime.

Then set $x_0 = x_0 + 2\Delta$, $F = 1 - F$ and go to step 1. By using the variable F, we eliminate the need to reset each non-prime member of S back to zero.

4.2. *Further Analysis.*

When we analyze the performance of algorithm C we find that steps 2, 3, 4, 8, 9 may be treated in the same manner as steps 1, 2, 3, 4 of algorithm A, namely

$$t_1 \doteq k_1 b, t_2 \doteq k_2 \Delta \sum_{i=2}^{b} 1/p_i, t_3 \doteq k_3 \Delta, \quad \text{and} \quad t_4 \doteq k_4 \Delta / \log x_0 .$$

We still need to find the contribution of steps 5, 6, 7; this contribution is easily arrived at if we make the (not unreasonable) assumption that the length of time required to execute step 6 equals the time required to execute step 7. Then,

$$t_5 = k_5\left(\pi\left((x_0+2\Delta)^{\frac{1}{2}}\right)-b\right) \doteq k_5 x_0^{\frac{1}{2}}/\log(x_0^{\frac{1}{2}}) - k_5 b$$

and the total execution time, $T = \sum_{i=1}^{5} t_i$.
Note that $T_\Delta = t_1 + t_2 + t_3$ is constant for a given Δ. Hence $T = T_\Delta + t_4 + t_5$.

4.3. *Observed Speed of the Dual Sieve.*

These theoretical results have been corroborated by the observed execution speed of the dual sieve program written for the IBM 370/168. For our particular program we can specify the k_i in terms of the number of machine instructions at each algorithm step, obtaining $k_1 = 18$, $k_2 = 2$, $k_3 = 6$, $k_4 = 18$, $k_5 = 11$.

Unfortunately, the execution time of instructions on the 370/168 varies greatly; depending upon the size of the loop, observed execution rates were found to be 3.5 instructions/10^{-6} second for a 2 instruction loop and 4.0 instructions/10^{-6} second for a 10 instruction loop. Using 3.75 as an average rate, the observed and theoretical rates were within 5% of each other for $10^8 < x < 10^{12}, \Delta = 250000$. (The execution speed was essentially linear, with the sieve slowing down by about 15% over the range of x from 10^9 to 10^{12}).

The greater speed of instructions which are executed in larger loops suggests a further refinement to the dual sieve whenever a computer like the 370/168 is employed. Since step 3 of algorithm C is only two instructions long (and therefore less efficient than a longer loop would be) we could construct a third sieve which would combine several (probably 2, 3, or 4) p_i into one loop. This would be most effective only for the smaller p_i; for example we could pick d, where $1 < d < b$ and $p_d^2 \approx \Delta$. Then use this third sieve for $2 \leq i \leq d$, steps 2-4 for $d < i \leq b$, and steps 5-7 for the remaining i. Of course, the value chosen for d would depend upon the computer used, as well as the value for Δ.

Since the initialization of the sieve for large x_0 is the most expensive part of the algorithm it pays off to make the removals of the found multiples in bits instead of computer words. The time t_1 will then decrease by a factor of 32 while the other times will increase by a small factor if the removal is done by one of the 32 prestored masks.

5. Primes in Arithmetic Progressions for the Moduli 3, 4, 8, 12 and 24.

Let $\pi_{b,c}(x)$ denote the number of primes not exceeding x that are contained in the arithmetic progression $bn+c$, $n=0,1,2,\ldots$. Table 1 gives counts for primes in all the arithmetic progressions for $b=24$. The values for $\pi_{24,1}(x)$ are given directly. To obtain values for $\pi_{24,c}(x)$, $c=5,7,11$ etc., simply add the number in the proper column to the value for $\pi_{24,1}(x)$. For example, $\pi_{24,7}(5\cdot 10^{11})=2,413,491,259+24948$.

A similar procedure may be used to find prime counts for all the progressions for $b=3,4,8,12$. For example, $\pi_{8,3}(x)=\pi_{24,11}(x)+\pi_{24,19}(x)+1$ (The prime 3 is not counted in the progressions of 24). For $x=2\cdot 10^{11}$, we get $2\cdot 1,000,872,637+17200+18075+1=2,001,780,550$.

Table 1.

X ($\times 10^{11}$)	$24n+1$	$+5$	$+7$	$+11$	$+13$	$+17$	$+19$	$+23$	$\pi(x)$	$\pi(x^{\frac{1}{2}})$
1	514,742,404	17670	20329	12688	14818	18176	13905	17993	4,118,054,813	27293
2	1,000,872,637	12750	13725	17200	23829	21365	18075	17017	8,007,105,059	37499
3	1,477,282,891	19210	26003	22786	30200	27006	25929	24871	11,818,439,135	45147
4	1,947,603,834	26067	27716	20818	25707	26293	22819	25563	15,581,005,657	51526
5	2,413,491,259	35623	24948	20541	27245	30227	32877	34607	19,308,136,142	57084
6	2,875,911,324	28788	24232	24264	36745	25183	39596	32384	23,007,501,786	62074
7	3,335,479,816	42952	40137	35218	27841	23289	32079	34264	26,684,074,310	66650
8	3,792,644,848	38878	32891	44463	31304	22381	26334	28490	30,341,383,527	70882
9	4,247,727,818	30456	19116	44043	17340	16052	21370	16663	33,981,987,586	74812
10	4,700,968,265	30413	33433	36463	5547	15320	21480	23240	37,607,912,018	78498

REFERENCES

1. D. E. Knuth, *The Art of Computer Programming* Vol. II *Seminumerical Algorithms*, Addison Wesley, Reading, Mass. (1971).

DEPARTMENT OF MATHEMATICS AND COMPUTER SCIENCE
UNIVERSITY OF SOUTH CAROLINA
COLUMBIA, S.C. 29208
USA