Sieve of Eratosthenes

From Wikipedia, the free encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item
Cite this page

Print/export
Create a book
Download as PDF
Printable version

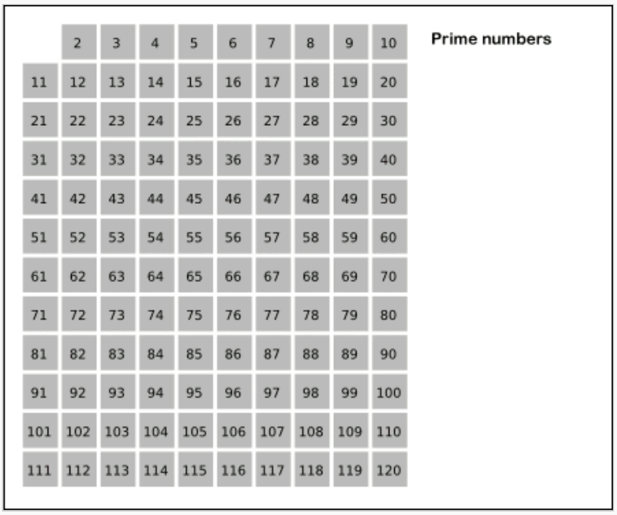
Languages
العربية
Български
Bosanski
Català
Čeština
Dansk
Deutsch
Ελληνικά
Español
Esperanto
فارسی
Français
한국어
Hrvatski
Bahasa Indonesia
Italiano
עברית
ქართული
Latina
Latviešu
Lietuvių
Magyar
Македонски
مازرونی
Bahasa Melayu
Nederlands
日本語
Norsk bokmål
Occitan
Oʻzbekcha/ўзбекча
Piemontèis
Polski
Português
Română
Русский
Shqip
Sicilianu
සිංහල
Simple English
Slovenčina
Slovenščina
Српски / srpski

In mathematics, the **sieve of Eratosthenes** (Ancient Greek: κόσκινον Ἐρατοσθένους, *kóskinon Eratosthénous*), one of a number of prime number sieves, is a simple, ancient algorithm for finding all prime numbers up to any given limit. It does so by iteratively marking as composite (i.e., not prime) the multiples of each prime, starting with the multiples of 2.[1]

The multiples of a given prime are generated as a sequence of numbers starting from that prime, with constant difference between them that is equal to that prime.[1] This is the sieve's key distinction from using trial division to sequentially test each candidate number for divisibility by each prime.[2]

The sieve of Eratosthenes is one of the most efficient ways to find all of the smaller primes. It is named after Eratosthenes of Cyrene, a Greek mathematician; although none of his works have survived, the sieve was described and attributed to Eratosthenes in the *Introduction to Arithmetic* by Nicomachus.[3]

The sieve may be used to find primes in arithmetic progressions.[4]



	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	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100
101	102	103	104	105	106	107	108	109	110
111	112	113	114	115	116	117	118	119	120

Sieve of Eratosthenes: algorithm steps for primes below 121 (including optimization of starting from prime's square).

Contents

Overview [edit]

Sift the Two's and Sift the Three's,
The Sieve of Eratosthenes.
When the multiples sublime,
The numbers that remain are Prime.

Anonymous[5]

A prime number is a natural number that has exactly two distinct natural number divisors: 1 and itself.

To find all the prime numbers less than or equal to a given integer n by Eratosthenes' method:

1. Create a list of consecutive integers from 2 through n: (2, 3, 4, ..., n).
2. Initially, let p equal 2, the first prime number.
3. Starting from p, enumerate its multiples by counting to n in increments of p, and mark them in the list (these will be $2p$, $3p$, $4p$, ... ; the p itself should not be marked).
4. Find the first number greater than p in the list that is not marked. If there was no such number, stop. Otherwise, let p now equal this new number (which is the next prime), and repeat from step 3.

When the algorithm terminates, the numbers remaining not marked in the list are all the primes below n.

The main idea here is that every value for p is prime, because we have already marked all the multiples of the numbers less than p. Note that some of the numbers being marked may have already been marked earlier (e.g., 15 will be marked both for 3 and 5).

As a refinement, it is sufficient to mark the numbers in step 3 starting from p^2, as all the smaller multiples of p will have already been marked at that point. This means that the algorithm is allowed to terminate in step 4 when p^2 is greater than n.[1]

Another refinement is to initially list odd numbers only, (3, 5, ..., n), and count in increments of $2p$ in step 3, thus marking only odd multiples of p. This actually appears in the original algorithm.[1] This can be generalized with wheel factorization, forming the initial list only from numbers coprime with the first few primes and not just from odds (i.e., numbers coprime with 2), and counting in the correspondingly adjusted increments so that only such multiples of p are generated that are coprime with those small primes, in the first place.[6]

Example [edit]

To find all the prime numbers less than or equal to 30, proceed as follows.

Srpskohrvatski / српскохрватски
Suomi
Svenska
Türkçe
Українська
Tiếng Việt
中文
Edit links

First generate a list of integers from 2 to 30:

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 28 29 30

First number in the list is 2; cross out every 2nd number in the list after it by counting up from 2 in increments of 2 (these will be all the multiples of 2 in the list):

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 ~~28~~ 29 ~~30~~

Next number in the list after 2 is 3; cross out every 3rd number in the list after it by counting up from 3 in increments of 3 (these will be all the multiples of 3 in the list):

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 28~~ 29 ~~30~~

Next number not yet crossed out in the list after 3 is 5; cross out every 5th number in the list after it by counting up from 5 in increments of 5 (i.e. all the multiples of 5):

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 28~~ 29 ~~30~~

Next number not yet crossed out in the list after 5 is 7; the next step would be to cross out every 7th number in the list after 7, but they are all already crossed out at this point, as these numbers (14, 21, 28) are also multiples of smaller primes because 7*7 is greater than 30. The numbers left not crossed out in the list at this point are all the prime numbers below 30:

```
2  3     5     7           11    13          17    19          23                29
```

Algorithm and variants [edit]

The sieve of Eratosthenes can be expressed in pseudocode, as follows:[7][8]

```
Input: an integer $n > 1$

Let $A$ be an array of Boolean values, indexed by integers 2 to $n$,
initially all set to true.

 for $i$ = 2, 3, 4, ..., not exceeding $\sqrt{n}$:
  if $A[i]$ is true:
    for $j = i^2, i^2+i, i^2+2i, i^2+3i$, ..., not exceeding $n$:
      $A[j]$ := false

Output: all $i$ such that $A[i]$ is true.
```

This algorithm produces all primes not greater than n. It includes a common optimization, which is to start looking for multiples of each prime i at i^2. The complexity of this algorithm is $\mathrm{O}(n \log \log n)$.[8]

Segmented sieve [edit]

As Sorenson notes, the problem with the sieve of Eratosthenes is not the number of operations it performs, but rather its memory requirements.[8] For large n, the range of primes may not fit in memory; worse, even for moderate n, its cache use is highly suboptimal: the algorithm walks through the entire array A, exhibiting almost no locality of reference.

A solution to these problems is offered by *segmented* sieves, where only portions of the range are sieved at a time.[9] These have been known since the 1970s, and work as follows:[8][10]

1. Divide the range 2 through n into segments of some size $\Delta \le \sqrt{n}$.
2. Find the primes up to $\sqrt{n}$, using the "regular" sieve.
3. For each Δ-sized block from $\sqrt{n}+1$ to n, set up a Boolean array of size Δ. Eliminate the multiples of each $p \le \sqrt{\mathrm{n}}$ found in step 1.

If Δ is chosen to be $\sqrt{n}$, the space complexity of the algorithm is $\mathrm{O}(\sqrt{n})$, while the time complexity is the same as that of the regular sieve.[8]

For ranges with upper limit n so large that the sieving primes below $\sqrt{n}$ as required by the page segmented sieve of Eratosthenes cannot fit in memory, a slower but much more space-efficient sieve like that sieve of Sorenson can be used instead.[11]

Incremental sieve [edit]

Trial division can be used to produce primes by filtering out the composites found by testing each candidate number for divisibility by its preceding primes. It is not the sieve of Eratosthenes but is often confused with it, even though the sieve of Eratosthenes directly generates the composites instead of testing for them. Trial division has worse theoretical complexity than that of the sieve of Eratosthenes in generating ranges of primes.[2]

When testing each candidate number, the optimal trial division algorithm uses just those prime numbers not exceeding its square root. The widely known 1975 functional code by David Turner[12] is often presented as an example of the sieve of Eratosthenes[6] but is actually a sub-optimal trial division algorithm.[2]

An incremental formulation of the sieve[2] generates primes indefinitely (i.e., without an upper bound) by interleaving the generation of primes with the generation of their multiples (so that primes can be found in gaps between the multiples), where the multiples of each prime *p* are generated directly, by counting up from the square of the prime in increments of *p* (or *2p* for odd primes). The generation must be initiated only when the prime's square is reached, to avoid adverse effects on efficiency.

Computational analysis [edit]

This section **needs additional citations for verification**. Please help improve this article by adding citations to reliable sources. Unsourced material may be challenged and removed. *(June 2015)*

The work performed by this algorithm is almost entirely the operations to cull the composite number representations which for the basic non-optimized version is the sum of the range divided by each of the primes up to that range or $n\sum_{p\leq n}\frac{1}{p}$, where n is the sieving range in this and all further analysis.

By rearranging Mertens' 2nd theorem, this is equal to $\ln\ln n + M$ as n approaches infinity, where M is the Meissel–Mertens constant of about 0.26149721284764278375542683860869585905156...

The optimization of starting at the square of each prime and only culling for primes less than the square root changes the "n" in the above expression to $\sqrt{n}$ (or $n^{\frac{1}{2}}$) and not culling until the square means that the sum of the base primes each minus two is subtracted from the operations. As the sum of the first x primes is $\frac{1}{2}x^2\ln x$[13] and the Prime number theorem says that x is approximately $\frac{x}{\ln x}$ then the sum of primes to n is $\frac{1}{2}\frac{n^2}{\ln n}$ and therefore the sum of base primes to $\sqrt{n}$ is $\frac{1}{\ln n}$ expressed as a factor of n. The extra offset of two per base prime is $2\pi(n^{\frac{1}{2}})$ where π is the Prime-counting function in this case or $\frac{4n^{\frac{1}{2}}}{\ln n}$; expressing this as a factor of n as are the other terms, this is $\frac{4}{\sqrt{n}\ln n}$. Combining all of this, the expression for the number of optimized operations without wheel factorization is $\ln\ln n - \frac{1}{\ln n}\left(1-\frac{4}{\sqrt{n}}\right)+M-\ln 2$.

For the wheel factorization cases, there is a further offset of the operations not done of $\sum_{p\leq x}\frac{1}{p}$ where x is the highest wheel prime and a constant factor of the whole expression is applied which is the fraction of remaining prime candidates as compared to the repeating wheel circumference. The wheel circumference is $\prod_{p\leq x}p$ and it can easily be determined that this wheel factor is $\prod_{p\leq x}\frac{p-1}{p}$ as $\frac{p-1}{p}$ is the fraction of remaining candidates for the highest wheel prime, x, and each succeeding smaller prime leaves its corresponding fraction of the previous combined fraction.

Combining all of the above analysis, the total number of operations for a sieving range up to n including wheel factorization for primes up to x is approximately:

$$\prod_{p\leq x}\frac{p-1}{p}\left(\ln\ln n-\frac{1}{\ln n}\left(1-\frac{4}{\sqrt{n}}\right)+M-\ln 2-\sum_{p\leq x}\frac{1}{p}\right)$$

To show that the above expression is a good approximation to the number of composite number cull operations performed by the algorithm, following is a table showing the actually measured number of operations for a practical implementation of the sieve of Eratosthenes as compared to the number of operations predicted from the above expression with both expressed as a fraction of the range (rounded to four decimal places) for different sieve ranges and wheel factorizations (Note that the last column is a maximum practical wheel as to the size of the wheel gaps Look Up Table - almost 10 million values):

n	no wheel	odds	2/3/5 wheel	2/3/5/7 wheel	2/3/5/7/11/13/17/19 wheel
10^3	1.4090/1.3745	0.4510/0.4372	0.1000/0.0909	0.0580/0.0453	0.0060/------
10^4	1.6962/1.6844	0.5972/0.5922	0.1764/0.1736	0.1176/0.1161	0.0473/0.0391
10^5	1.9299/1.9261	0.7148/0.7130	0.2388/0.2381	0.1719/0.1714	0.0799/0.0805
10^6	2.1218/2.1220	0.8109/0.8110	0.2902/0.2903	0.2161/0.2162	0.1134/0.1140
10^7	2.2850/2.2863	0.8925/0.8932	0.3337/0.3341	0.2534/0.2538	0.1419/0.1421
10^8	2.4257/2.4276	0.9628/0.9638	0.3713/0.3718	0.2856/0.2860	0.1660/0.1662

The above table shows that the above expression is a very good approximation to the total number of culling operations for sieve ranges of about a hundred thousand (10^5) and above.

Algorithm complexity [edit]

As can be seen from the above by removing all constant offsets and constant factors and ignoring terms that tend to zero as n approaches infinity, the time complexity of calculating all primes below *n* in the random access machine model is $O(n\log\log n)$ operations, a direct consequence of the fact that the prime harmonic series asymptotically approaches $\log\log n$. It has an exponential time complexity with regard to input size, though, which makes it a pseudo-polynomial algorithm. The basic algorithm requires $O(n)$ of memory.

The bit complexity of the algorithm is $O(n(\log n)(\log\log n))$ bit operations with a memory requirement of $O(n)$.[14]

The normally implemented page segmented version has the same operational complexity of $O(n\log\log n)$ as the non-segmented version but reduces the space requirements to the very minimal size of the segment page plus the memory required to store the base primes less than the square root of the range used to cull composites from successive page segments of size $O(\frac{\sqrt{n}}{\log n})$.

A special rarely if ever implemented segmented version of the sieve of Eratosthenes, with basic optimizations, uses $O(n)$ operations and $O(n^{1/2}\log\log n/\log n)$ bits of memory.[15][16][17]

To show that the above approximation in complexity is not very accurate even for about as large as practical a range, the following is a table of the estimated number of operations as a fraction of the range rounded to four places, the calculated ratio for a factor of ten change in range based on this estimate, and the factor based on the log log n estimate for various ranges and wheel factorizations (the combo column uses a frequently practically used pre-cull by the maximum wheel factorization but only the 2/3/5/7 wheel for the wheel factor as the full factorization is difficult to implement efficiently for page segmentation):

n	*no wheel*	*odds*	*2/3/5 wheel*	*2/3/5/7 wheel*	*combo wheel*	*2/3/5/7/11/13/17/19 wheel*
10^6	2.122/1.102/1.075	0.811/1.137/1.075	0.2903/1.22/1.075	0.2162/1.261/1.075	0.1524/1.416/1.075	0.114/1.416/1.075
10^7	2.2863/1.077/1.059	0.8932/1.101/1.059	0.3341/1.151/1.059	0.2537/1.174/1.059	0.1899/1.246/1.059	0.1421/1.246/1.059
10^8	2.4276/1.062/1.048	0.9638/1.079/1.048	0.3718/1.113/1.048	0.286/1.127/1.048	0.2222/1.17/1.048	0.1662/1.17/1.048
10^9	2.5514/1.051/1.04	1.0257/1.064/1.04	0.4048/1.089/1.04	0.3143/1.099/1.04	0.2505/1.127/1.04	0.1874/1.127/1.04
10^{10}	2.6615/1.043/1.035	1.0808/1.054/1.035	0.4342/1.073/1.035	0.3395/1.08/1.035	0.2757/1.101/1.035	0.2063/1.101/1.035
10^{11}	2.7608/1.037/1.03	1.1304/1.046/1.03	0.4607/1.061/1.03	0.3622/1.067/1.03	0.2984/1.082/1.03	0.2232/1.082/1.03
10^{12}	2.8511/1.033/1.027	1.1755/1.04/1.027	0.4847/1.052/1.027	0.3828/1.057/1.027	0.319/1.069/1.027	0.2387/1.069/1.027
10^{13}	2.9339/1.029/1.024	1.217/1.035/1.024	0.5068/1.046/1.024	0.4018/1.049/1.024	0.3379/1.059/1.024	0.2528/1.059/1.024
10^{14}	3.0104/1.026/1.022	1.2552/1.031/1.022	0.5272/1.04/1.022	0.4193/1.044/1.022	0.3554/1.052/1.022	0.2659/1.052/1.022
10^{15}	3.0815/1.024/1.02	1.2907/1.028/1.02	0.5462/1.036/1.02	0.4355/1.039/1.02	0.3717/1.046/1.02	0.2781/1.046/1.02
10^{16}	3.1478/1.022/1.018	1.3239/1.026/1.018	0.5639/1.032/1.018	0.4507/1.035/1.018	0.3868/1.041/1.018	0.2894/1.041/1.018

The above shows that the log log n estimate is not very accurate even for maximum practical ranges of about 10^{16}. One can see why it does not match by looking at the computational analysis above and seeing that within these practical sieving range limits, there are very significant constant offset terms such that the very slowly growing log log n term doesn't get large enough so as to make these terms insignificant until the sieving range approaches infinity - well beyond any practical sieving range. Within these practical ranges, these significant constant offsets mean that the performance of the Sieve of Eratosthenes is much better than one would expect just using the asymptotic time complexity estimates by a significant amount. bit that also means that the slope of the performance with increasing range is steeper than predicted as the benefit of the constant offsets becomes slightly less significant.

One should also note that in using the calculated operation ratios to the sieve range, it must be less than about 0.2587 in order to be faster than the often compared sieve of Atkin **if the operations take approximately the same time each in CPU clock cycles**, which is a reasonable assumption for the one huge bit array algorithm. Using that assumption, the sieve of Atkin is only faster than the maximally wheel factorized sieve of Eratosthenes for ranges of over 10^{13} at which point the huge sieve buffer array would need about a quarter Terabyte (about 250 Gigabytes) of RAM memory even if bit packing were used - i.e., not very practical! An analysis of the page segmented versions will show that the assumption that the time per operation stays the same between the two algorithms does not hold for page segmentation and that the sieve of Atkin operations get slower much faster than the sieve of Eratosthenes with increasing range. Thus for practical purposes, the maximally wheel factorized Sieve of Eratosthenes is faster than the Sieve of Atkin although the Sieve of Atkin is faster for lesser amounts of wheel factorization.

Using Big O Notation is also not the correct way to compare practical performance of even variations of the Sieve of Eratosthenes as it ignores constant factors and offsets that may be very significant for practical ranges: The Sieve of Eratosthenes variation known as the Pritchard Wheel Sieve[15][16][17] has an O(n) performance, but its basic implementation requires either a "one large array" algorithm which limits its usable range to the amount of available memory else it needs to be page segmented to reduce memory use. When implemented with page segmentation in order to save memory, the basic algorithm still requires about $O(\frac{n}{\log n})$ bits of memory (much more than the requirement of the basic page segmented Sieve of Eratosthenes using $O(\frac{\sqrt{n}}{\log n})$ bits of memory). Pritchard's work reduced the memory requirement to the limit as described above the table, but the cost is a fairly large constant factor of about three in execution time to about 0.75 times the sieve range due to the complex computations required to do so. As can be seen from the above table for the basic Sieve of Eratosthenes, even though the resulting wheel sieve has O(n) performance and an acceptable memory requirement, it will never be faster than a reasonably Wheel Factorized basic Sieve of Eratosthenes for any practical sieving range by a factor of about two. Other than that it is quite complex to implement, it is rarely practically implemented because it still uses more memory than the basic Sieve of Eratosthenes implementations described here as well as being slower for practical ranges. It is thus more of an intellectual curiosity than something practical.

Euler's Sieve [edit]

Euler's proof of the zeta product formula contains a version of the sieve of Eratosthenes in which each composite number is eliminated exactly once.[8] It, too, starts with a list of numbers from *2* to *n* in order. On each step the first element is identified as the next prime and the results of multiplying this prime with each element of the list are marked in the list for subsequent deletion. The initial element and the marked elements are then removed from the working sequence, and the process is repeated:

```
[2] (3) 5  7  9  11  13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49 51 53 55 57 59 61 63 65 67 69 71 73 75
77 79  ...
[3]    (5) 7     11  13     17 19     23 25     29 31     35 37     41 43     47 49     53 55     59 61     65 67     71 73
77 79  ...
[4]       (7)    11  13     17 19     23        29 31        37     41 43     47 49     53        59 61        67     71 73
77 79  ...
[5]             (11) 13     17 19     23        29 31        37     41 43     47        53        59 61        67     71 73
79  ...
[...]
```

Here the example is shown starting from odds, after the first step of the algorithm. Thus, on the *k*th step all the remaining multiples of the *k*th prime are removed from the list, which will thereafter contain only numbers coprime with the first *k* primes (cf., wheel factorization), so that the list will start with the next prime, and all the numbers in it below the square of its first element will be prime too.

Thus, when generating a bounded sequence of primes, when the next identified prime exceeds the square root of the upper limit, all the remaining numbers in the list are prime.[8] In the example given above that is achieved on identifying 11 as next prime, giving a list of all primes less than or equal to 80.

Note that numbers that will be discarded by a step are still used while marking the multiples in that step, e.g., for the multiples of 3 it is $3 \cdot 3 = 9, 3 \cdot 5 = 15, 3 \cdot 7 = 21, 3 \cdot \boldsymbol{9} = 27$, ..., $3 \cdot \boldsymbol{15} = 45$, ..., so care must be taken dealing with this.[8]

See also [edit]

- Sieve of Sundaram
- Sieve of Atkin
- Sieve theory

References [edit]

1. ^ *a b c d* Horsley, Rev. Samuel, F. R. S., "*Κόσκινον Ερατοσθένους* or, The Sieve of Eratosthenes. Being an account of his method of finding all the Prime Numbers," *Philosophical Transactions* (1683–1775), Vol. 62. (1772), pp. 327–347.
2. ^ *a b c d* O'Neill, Melissa E., "The Genuine Sieve of Eratosthenes", Journal of Functional Programming, Published online by Cambridge University Press 9 October 2008 doi:10.1017/S0956796808007004, pp. 10, 11 (contains two incremental sieves in Haskell: a priority-queue–based one by O'Neill and a list–based, by Richard Bird).
3. ^ Nicomachus, *Introduction to Arithmetic*, I, 13. [1]
4. ^ J. C. Morehead, "Extension of the Sieve of Eratosthenes to arithmetical progressions and applications", Annals of Mathematics, Second Series **10**:2 (1909), pp. 88–104.
5. ^ Clocksin, William F., Christopher S. Mellish, Programming in Prolog, 1984, p. 170. ISBN 3-540-11046-1.
6. ^ *a b* Runciman, Colin (1997). "Functional Pearl: Lazy wheel sieves and spirals of primes" (PDF). *J. Functional Programming* **7** (2): 219–225. doi:10.1017/S0956796897002670.
7. ^ Sedgewick, Robert (1992). *Algorithms in C++*. Addison-Wesley. ISBN 0-201-51059-6. , p. 16.
8. ^ *a b c d e f g h* Jonathan Sorenson, An Introduction to Prime Number Sieves, Computer Sciences Technical Report #909, Department of Computer Sciences University of Wisconsin-Madison, January 2, 1990 (the use of optimization of starting from squares, and thus using only the numbers whose square is below the upper limit, is shown).
9. ^ Crandall & Pomerance, *Prime Numbers: A Computational Perspective*, second edition, Springer: 2005, pp. 121–24.
10. ^ Bays, Carter; Hudson, Richard H. (1977). "The segmented sieve of Eratosthenes and primes in arithmetic progressions to 10^{12}". *BIT* **17** (2): 121–127. doi:10.1007/BF01932283.
11. ^ J. Sorenson, The pseudosquares prime sieve, Proceedings of the 7th International Symposium on Algorithmic Number Theory. (ANTS-VII, 2006).
12. ^ Turner, David A. SASL language manual. Tech. rept. CS/75/1. Department of Computational Science, University of St. Andrews 1975. (`sieve (p:xs) = p : sieve [x | x <- xs, rem x p > 0]; primes = sieve [2..]`)
13. ^ E. Bach and J. Shallit, §2.7 in Algorithmic Number Theory, Vol. 1: Efficient Algorithms, MIT Press, Cambridge, MA, 1996.
14. ^ Pritchard, Paul, "Linear prime-number sieves: a family tree," *Sci. Comput. Programming* **9**:1 (1987), pp. 17–35.
15. ^ *a b* Paul Pritchard, A sublinear additive sieve for finding prime numbers, Communications of the ACM 24 (1981), 18–23. MR 82c:10011
16. ^ *a b* Paul Pritchard, Explaining the wheel sieve, Acta Informatica 17 (1982), 477–485. MR 84g:10015
17. ^ *a b* Paul Pritchard, Fast compact prime number sieves (among others), Journal of Algorithms 4 (1983), 332–344. MR 85h:11080

External links [edit]

- *Eratosthenes, sieve of* at Encyclopaedia of Mathematics
- Interactive JavaScript Page
- Sieve of Eratosthenes by George Beck, Wolfram Demonstrations Project.
- Sieve of Eratosthenes in Haskell
- Sieve of Eratosthenes algorithm illustrated and explained. Java and C++ implementations.
- A related sieve written in x86 assembly language
- A highly optimized Sieve of Eratosthenes in C
- A parallel implementation in C#
- SieveOfEratosthenesInManyProgrammingLanguages c2 wiki page
- The Art of Prime Sieving Sieve of Eratosthenes in C from 1998 with nice features and algorithmic tricks explained.

v · t · e	**Number-theoretic algorithms**
Primality tests	AKS test · APR test · Baillie–PSW · ECPP test · Elliptic curve · Pocklington · Fermat · Lucas · *Lucas–Lehmer* · *Lucas–Lehmer–Riesel* · *Proth's theorem* · *Pépin's* · Quadratic Frobenius test · Solovay–Strassen · Miller–Rabin

Prime-generating	Sieve of Atkin · **Sieve of Eratosthenes** · Sieve of Sundaram · Wheel factorization
Integer factorization	Continued fraction (CFRAC) · Dixon's · Lenstra elliptic curve (ECM) · Euler's · Pollard's rho · *p* − 1 · *p* + 1 · Quadratic sieve (QS) · General number field sieve (GNFS) · *Special number field sieve (SNFS)* · Rational sieve · Fermat's · Shanks' square forms · Trial division · Shor's
Multiplication	Ancient Egyptian · Long · Karatsuba · Toom–Cook · Schönhage–Strassen · Fürer's
Discrete logarithm	Baby-step giant-step · Pollard rho · Pollard kangaroo · Pohlig–Hellman · Index calculus · Function field sieve
Greatest common divisor	Binary · Euclidean · Extended Euclidean · Lehmer's
Modular square root	Cipolla · Pocklington's · Tonelli–Shanks
Other algorithms	Chakravala · Cornacchia · Integer relation · Integer square root · Modular exponentiation · Schoof's
Italics indicate that algorithm is for numbers of special forms · Smallcaps indicate a deterministic algorithm	

Categories: Primality tests | Sieve theory | Algorithms