



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
Català
Deutsch
★ Español
Esperanto
Français
Italiano
Magyar
Nederlands
日本語
Norsk bokmål
Polski
Português
Русский
Simple English
Svenska
Українська
Tiếng Việt
中文

Edit links

Create account Log in

Article Talk

Read Edit View history

Search

Primality test

From Wikipedia, the free encyclopedia

A **primality test** is an **algorithm** for determining whether an input number is **prime**. Amongst other fields of **mathematics**, it is used for **cryptography**. Unlike **integer factorization**, primality tests do not generally give **prime factors**, only stating whether the input number is prime or not. Factorization is thought to be a computationally difficult problem, whereas primality testing is comparatively easy (its **running time** is **polynomial** in the size of the input). Some primality tests *prove* that a number is prime, while others like **Miller–Rabin** prove that a number is **composite**. Therefore the latter might be called *compositeness tests* instead of primality tests.

Contents [hide]

- Simple methods
 - Pseudocode
- Heuristic tests
- Probabilistic tests
 - Fermat primality test
 - Miller-Rabin and Solovay-Strassen primality test
 - Frobenius primality test
 - Other tests
- Fast deterministic tests
- Complexity
- Number-theoretic methods
- References
- External links

Simple methods [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. *(August 2013)*

The simplest primality test is *trial division*: Given an input number *n*, check whether any integer *m* from 2 to *n* − 1 evenly **divides** *n* (the division leaves no **remainder**). If *n* is divisible by any *m* then *n* is **composite**, otherwise it is **prime**.^[1]

For example, to test whether 17 is prime, test whether 17 is divisible by 2, or 3, or 4, 5, 6, ..., 16. Since a prime is only divisible by 1 and itself, if we reach 16 without finding a **divisor**, then we have proven that 17 is prime. However, we don't actually have to check all numbers up to *n*. Let's look at another example: all the divisors of 100:

2, 4, 5, 10, 20, 25, 50

here we see that the largest factor is 100/2 = 50. This is true for all *n*: all divisors are less than or equal to *n*/2. We can do better though. If we take a closer look at the divisors, we will see that some of them are redundant. If we write the list differently:

100 = 2 × 50 = 4 × 25 = 5 × 20 = 10 × 10 = 20 × 5 = 25 × 4 = 50 × 2

it becomes obvious. Once we reach 10, which is $\sqrt{100}$, the divisors just flip around and repeat. Therefore we can further eliminate testing divisors greater than \sqrt{n} .^[1] We can also eliminate all the even numbers greater than 2, since if an even number can divide *n*, so can 2.

The algorithm can be improved further by observing that all primes are of the form $6k \pm 1$, with the exception of 2 and 3. This is because all integers can be expressed as $(6k + i)$ for some integer *k* and for *i* = −1, 0, 1, 2, 3, or 4; 2 divides $(6k + 0)$, $(6k + 2)$, $(6k + 4)$; and 3 divides $(6k + 3)$. So a more efficient method is to test if *n* is divisible by 2 or 3, then to check through all the numbers of form $6k \pm 1 \leq \sqrt{n}$. This is 3 times as fast as testing all *m*.

Generalising further, it can be seen that all primes are of the form $c\#k + i$ for *i* < *c*# where *i* represents the numbers that are **coprime** to *c*# and where *c* and *k* are integers. For example, let *c* = 6. Then *c*# = 2 . 3 . 5

= 30. All integers are of the form $30k + i$ for $i = 0, 1, 2, \dots, 29$ and k an integer. However, 2 divides 0, 2, 4, ..., 28 and 3 divides 0, 3, 6, ..., 27 and 5 divides 0, 5, 10, ..., 25. So all prime numbers are of the form $30k + i$ for $i = -1, 1, 7, 11, 13, 17, 19, 23, 29$ (i.e. for $i < 30$ such that $\gcd(i, 30) = 1$). Note that if i and 30 are not coprime, then $30k + i$ is divisible by a prime divisor of 30, namely 2, 3 or 5, and is therefore not prime.

As $c \rightarrow \infty$, the number of values that $c\#k + i$ can take over a certain range decreases, and so the time to test n decreases. For this method, it is also necessary to check for divisibility by all primes that are less than c . Observations analogous to the preceding can be applied [recursively](#), giving the [Sieve of Eratosthenes](#).

A good way to speed up these methods (and all the others mentioned below) is to pre-compute and store a list of all primes up to a certain bound, say all primes up to 200. (Such a list can be computed with the [Sieve of Eratosthenes](#) or by an algorithm that tests each incremental m against all known primes $< \sqrt{m}$). Then, before testing n for primality with a serious method, n can first be checked for divisibility by any prime from the list. If it is divisible by any of those numbers then it is composite, and any further tests can be skipped.

A simple, but very inefficient primality test uses [Wilson's theorem](#), which states that p is prime if and only if:

$$(p-1)! \equiv -1 \pmod{p}$$

Although this method requires about p modular multiplications, rendering it impractical, theorems about primes and modular residues form the basis of many more practical methods.

Pseudocode [\[edit\]](#)

The following is a naive primality test in [pseudocode](#).

```
function is_prime(n : integer)
  if n ≤ 1
    return false
  else if n ≤ 3
    return true
  else if n mod 2 = 0 or n mod 3 = 0
    return false
```

```
let i ← 5
while i×i ≤ n
  if n mod i = 0 or n mod (i + 2) = 0
    return false
  i ← i + 6
return true
```

Heuristic tests [\[edit\]](#)

These are tests which seem to work well in practice, but are unproven and therefore are not, technically speaking, algorithms at all. The Fermat test and the Fibonacci test are simple examples, and they are very effective when combined. [John Selfridge](#) has conjectured that if p is an odd number, and $p \equiv \pm 2 \pmod{5}$, then p will be prime if both of the following hold:

- $2^{p-1} \equiv 1 \pmod{p}$
- $f_{p+1} \equiv 0 \pmod{p}$,

where f_k is the k th [Fibonacci number](#). Selfridge, [Carl Pomerance](#), and [Samuel Wagstaff](#) together offer \$620 for a counter example. The problem is still open as of August 23, 2015.^[2] The [Baillie-PSW primality test](#) is another excellent heuristic, using the [Lucas sequence](#) in place of the Fibonacci sequence.

Probabilistic tests [\[edit\]](#)

[Probabilistic tests](#) are more rigorous than heuristics in that they provide provable bounds on the probability of being fooled by a composite number. Many popular primality tests are probabilistic tests. These tests use, apart from the tested number n , some other numbers a which are chosen at random from some [sample space](#); the usual randomized primality tests never report a prime number as composite, but it is possible for a composite number to be reported as prime. The probability of error can be reduced by repeating the test with several independently chosen values of a ; for two commonly used tests, for *any* composite n at least half the a 's detect n 's compositeness, so k repetitions reduce the error probability to at most 2^{-k} , which can be made arbitrarily small by increasing k .

The basic structure of randomized primality tests is as follows:

1. Randomly pick a number a .
2. Check some equality (corresponding to the chosen test) involving a and the given number n . If the equality fails to hold true, then n is a composite number, a is known as a *witness* for the compositeness, and the test stops.
3. Repeat from step 1 until the required accuracy is achieved.

After one or more iterations, if n is not found to be a composite number, then it can be declared [probably prime](#).

Fermat primality test [\[edit\]](#)

The simplest probabilistic primality test is the [Fermat primality test](#) (actually a compositeness test). It works as follows:

Given an integer n , choose some integer a coprime to n and calculate a^{n-1} modulo n . If the result is different from 1, then n is composite. If it is 1, then n may or may not be prime.

If a^{n-1} (modulo n) is 1 but n is not prime, then n is called a [pseudoprime](#) to base a . In practice, we observe that, if a^{n-1} (modulo n) is 1, then n is usually prime. But here is a counterexample: if $n = 341$ and $a = 2$, then

$$2^{340} \equiv 1 \pmod{341}$$

even though $341 = 11 \cdot 31$ is composite. In fact, 341 is the smallest pseudoprime base 2 (see Figure 1 of [\[3\]](#)).

There are only 21853 pseudoprimes base 2 that are less than $25 \cdot 10^9$ (see page 1005 of [\[3\]](#)). This means that, for n up to $25 \cdot 10^9$, if 2^{n-1} (modulo n) equals 1, then n is prime, unless n is one of these 21853 pseudoprimes.

The Fermat primality test is only a heuristic test; some composite numbers ([Carmichael numbers](#)) will be declared "probably prime" no matter what witness is chosen. Nevertheless, it is often used if a rapid screening of numbers is needed, for instance in the key generation phase of the [RSA public key cryptographic algorithm](#).

Miller-Rabin and Solovay-Strassen primality test [\[edit\]](#)

The [Miller-Rabin primality test](#) and [Solovay-Strassen primality test](#) are more sophisticated variants which detect all composites (once again, this means: for every composite number n , at least 3/4 (Miller-Rabin) or 1/2 (Solovay-Strassen) of numbers a are witnesses of compositeness of n). These are also compositeness tests.

The Miller-Rabin primality test works as follows: Given an integer n , choose some positive integer $a < n$. Let $2^s d = n - 1$ where d is odd. If

$$a^d \not\equiv 1 \pmod{n}$$

and

$$a^{2^r d} \not\equiv -1 \pmod{n} \text{ for all } 0 \leq r \leq s-1$$

then n is composite and a is a witness for the compositeness. Otherwise, n may or may not be prime. The Miller-Rabin test is a [strong pseudoprime](#) test (see, [\[3\]](#) page 1004).

The Solovay-Strassen primality test uses another equality: Given an odd number n , choose some integer $a < n$, if

$$a^{(n-1)/2} \not\equiv \left(\frac{a}{n}\right) \pmod{n}, \text{ where } \left(\frac{a}{n}\right) \text{ is the } \text{Jacobi symbol},$$

then n is composite and a is a witness for the compositeness. Otherwise, n may or may not be prime. The Solovay-Strassen test is an [Euler pseudoprime](#) test (see, [\[3\]](#) page 1003).

For each individual value of a , the Solovay-Strassen test is weaker than the Miller-Rabin test. For example, if $n = 1905$ and $a = 2$, then the Miller-Rabin test shows that n is composite, but the Solovay-Strassen test does not. This is because 1905 is an Euler pseudoprime base 2 but not a strong pseudoprime base 2; this is illustrated in Figure 1 of [\[3\]](#).

Frobenius primality test [\[edit\]](#)

The Miller-Rabin and the Solovay-Strassen primality tests are simple and are much faster than other general primality tests. One method of improving efficiency further in some cases is the [Frobenius pseudoprimitivity test](#); a round of this test takes about three times as long as a round of Miller-Rabin, but achieves a probability bound comparable to seven rounds of Miller-Rabin.

The Frobenius test is a generalization of the [Lucas pseudoprime](#) test. One can also combine a Miller-Rabin type test with a Lucas pseudoprime test to get a primality test that has no known counterexamples. That is, this combined test has no known composite n for which the test reports that n is probably prime. One such test is

the [Baillie-PSW primality test](#), several variations of which are described in.^[4]

Other tests [\[edit\]](#)

[Leonard Adleman](#) and Ming-Deh Huang presented an errorless (but expected polynomial-time) variant of the [elliptic curve primality test](#). Unlike the other probabilistic tests, this algorithm produces a [primality certificate](#), and thus can be used to prove that a number is prime.^[5] The algorithm is prohibitively slow in practice.

If [quantum computers](#) were available, primality could be tested [asymptotically faster](#) than by using classical computers. A combination of [Shor's algorithm](#), an integer factorization method, with the [Pocklington primality test](#) could solve the problem in $O(\log^3 n \log \log n \log \log \log n)$.^[6]

Fast deterministic tests [\[edit\]](#)

Near the beginning of the 20th century, it was shown that a corollary of [Fermat's little theorem](#) could be used to test for primality.^[7] This resulted in the [Pocklington primality test](#).^[8] However, as this test requires a partial factorization of $n - 1$ the running time was still quite slow in the worst case. The first [deterministic](#) primality test significantly faster than the naive methods was the [cyclotomy test](#); its runtime can be proven to be $O((\log n)^c \log \log n)$, where n is the number to test for primality and c is a constant independent of n . Many further improvements were made, but none could be proven to have polynomial running time. (Note that running time is measured in terms of the size of the input, which in this case is $\sim \log n$, that being the number of bits needed to represent the number n .) The [elliptic curve primality test](#) can be proven to run in $O((\log n)^6)$, but only if some still unproven (but widely assumed to be true) statements of [analytic number theory](#) are used.^[which?] Similarly, under the [generalized Riemann hypothesis](#), the Miller–Rabin test can be turned into a deterministic version (called [Miller's test](#)) with runtime $\tilde{O}((\log n)^4)$.^[9] In practice, this algorithm is slower than the other two for sizes of numbers that can be dealt with at all. Because the implementation of these methods is rather difficult and creates a risk of programming errors, the slower but simpler tests are often preferred.

In 2002 the first provably polynomial time test for primality was invented by [Manindra Agrawal](#), [Neeraj Kayal](#), and [Nitin Saxena](#). Their [AKS primality test](#) runs in $\tilde{O}((\log n)^{12})$ (improved to $\tilde{O}((\log n)^{7.5})$ in the published revision of their paper), which can be further reduced to $\tilde{O}((\log n)^6)$ if the [Sophie Germain conjecture](#) is true.^[10] Subsequently, Lenstra and Pomerance presented a version of the test which runs in time $\tilde{O}((\log n)^6)$ unconditionally.^[11]

Complexity [\[edit\]](#)

In [computational complexity theory](#), the formal language corresponding to the prime numbers is denoted as PRIMES. It is easy to show that PRIMES is in **Co-NP**: its complement COMPOSITES is in **NP** because one can decide compositeness by nondeterministically guessing a factor.

In 1975, [Vaughan Pratt](#) showed that there existed a certificate for primality that was checkable in polynomial time, and thus that PRIMES was in **NP**, and therefore in $\mathbf{NP} \cap \mathbf{coNP}$. See [primality certificate](#) for details.

The subsequent discovery of the Solovay–Strassen and Miller–Rabin algorithms put PRIMES in **coRP**. In 1992, the Adleman–Huang algorithm^[5] reduced the complexity to $\mathbf{ZPP} = \mathbf{RP} \cap \mathbf{coRP}$, which superseded Pratt's result.

The cyclotomy test of Adleman, [Pomerance](#), and Rumely from 1983 put PRIMES in **QP** ([quasi-polynomial time](#)), which is not known to be comparable with the classes mentioned above.

Because of its tractability in practice, polynomial-time algorithms assuming the Riemann hypothesis, and other similar evidence, it was long suspected but not proven that primality could be solved in polynomial time. The existence of the [AKS primality test](#) finally settled this long-standing question and placed PRIMES in **P**. However, PRIMES is not known to be **P-complete**, and it is not known whether it lies in classes lying inside **P** such as **NC** or **L**. It is known that PRIMES is not in **AC**⁰.^[12]

Number-theoretic methods [\[edit\]](#)

Certain number-theoretic methods exist for testing whether a number is prime, such as the [Lucas test](#) and [Proth's test](#). These tests typically require factorization of $n + 1$, $n - 1$, or a similar quantity, which means that they are not useful for general-purpose primality testing, but they are often quite powerful when the tested number n is known to have a special form.







The Lucas test relies on the fact that the [multiplicative order](#) of a number a modulo n is $n - 1$ for a prime n when a is a [primitive root modulo n](#). If we can show a is primitive for n , we can show n is prime.

References [\[edit\]](#)

- ^a ^b Riesel (1994) pp.2-3
- ^a John Selfridge#Selfridge's Conjecture about Primality Testing
- ^a ^b ^c ^d ^e Carl Pomerance; John L. Selfridge, Samuel S. Wagstaff, Jr. (July 1980). "The pseudoprimes to 25·10⁹"  (PDF). *Mathematics of Computation* **35** (151): 1003–1026. doi:10.1090/S0025-5718-1980-0572872-7 .
- ^a Robert Baillie; Samuel S. Wagstaff, Jr. (October 1980). "Lucas Pseudoprimes"  (PDF). *Mathematics of Computation* **35** (152): 1391–1417. doi:10.1090/S0025-5718-1980-0583518-6 . MR 583518 .
- ^a ^b Adleman, Leonard M.; Huang, Ming-Deh (1992). *Primality testing and Abelian varieties over finite field*. Lecture notes in mathematics **1512**. Springer-Verlag. ISBN 3-540-55308-8.
- ^a Chau, H. F.; Lo, H.-K. (1995). "Primality Test Via Quantum Factorization". [arXiv:quant-ph/9508005](https://arxiv.org/abs/quant-ph/9508005)  [quant-ph .
- ^a Pocklington, H. C. (1914). "The determination of the prime or composite nature of large numbers by Fermat's theorem". *Cambr. Phil. Soc. Proc.* **18**: 29–30. JFM 45.1250.02 .
- ^a Weisstein, Eric W., "Pocklington's Theorem" , *MathWorld*.
- ^a Gary L. Miller (1976). "Riemann's Hypothesis and Tests for Primality". *Journal of Computer and System Sciences* **13** (3): 300–317. doi:10.1016/S0022-0000(76)80043-8 .
- ^a Manindra Agrawal, Neeraj Kayal, Nitin Saxena, "PRIMES is in P" , *Annals of Mathematics* 160 (2004), no. 2, pp. 781–793.
- ^a Carl Pomerance and Hendrik W. Lenstra. [Primality testing with Gaussian periods](#) .
- ^a E. Allender, M. Saks, and I.E. Shparlinski, A lower bound for primality, *J. Comp. Syst. Sci.* **62** (2001), pp. 356–366.

- Richard Crandall and Carl Pomerance (2005). *Prime Numbers: A Computational Perspective* (2nd ed.). Springer. ISBN 0-387-25282-7. Chapter 3: Recognizing Primes and Composites, pp. 109–158. Chapter 4: Primality Proving, pp. 159–190. Section 7.6: Elliptic curve primality proving (ECP), pp. 334–340.
- Knuth, Donald (1997). "section 4.5.4". *The Art of Computer Programming*. Volume 2: *Seminumerical Algorithms* (Third ed.). Addison–Wesley. pp. 391–396. ISBN 0-201-89684-2.
- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein (2001). "Section 31.8: Primality testing". *Introduction to Algorithms* (Second ed.). MIT Press and McGraw–Hill. pp. 887–896. ISBN 0-262-03293-7.
- Papadimitriou, Christos H. (1993). "Section 10.2: Primality". *Computational Complexity* (1st ed.). Addison Wesley. pp. 222–227. ISBN 0-201-53082-1. Zbl 0833.68049 .
- Riesel, Hans (1994). *Prime Numbers and Computer Methods for Factorization*. Progress in Mathematics **126** (second ed.). Boston, MA: Birkhäuser. ISBN 0-8176-3743-5. Zbl 0821.11001 .

External links [\[edit\]](#)

- Excel Formula  ExcelExchange
- Solovay-Strassen  Implementation of the Solovay-Strassen primality test in Maple
- Distinguishing prime numbers from composite numbers , D.J. Bernstein
- The Prime Pages 
- Lucas Primality Test with Factored $N - 1$  at MathPages
- PRIMABOINCA  is a research project that uses Internet-connected computers to search for a counterexample to some conjectures. The first conjecture ([Agrawal's conjecture](#)) was the basis for the formulation of the first deterministic prime test algorithm in polynomial time ([AKS algorithm](#)).

v · t · e	Number-theoretic algorithms	[hide]
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 · <i>p</i> − 1 · <i>p</i> + 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 · Mbdular exponentiation · Schoof's	
<i>Italics</i> indicate that algorithm is for numbers of special forms · Smallcaps indicate a deterministic algorithm		

This page was last modified on 29 August 2015, at 13:52.

Text is available under the [Creative Commons Attribution-ShareAlike License](#); additional terms may apply. By using this site, you agree to the [Terms of Use](#) and [Privacy Policy](#). Wikipedia® is a registered trademark of the [Wikimedia Foundation, Inc.](#), a non-profit organization.

[Privacy policy](#) [About Wikipedia](#) [Disclaimers](#) [Contact Wikipedia](#) [Developers](#) [Mobile view](#)

