

Recognizing Primes

R. A. Mollin

Department of Mathematics and Statistics
University of Calgary, Calgary, Alberta, Canada, T2N 1N4
<http://www.math.ucalgary.ca/~ramollin/>
ramollin@math.ucalgary.ca

Abstract

This is a survey article on primality testing and general methods for recognizing primes. We also look at methods for generating random primes since this is valuable in cryptographic methods such as the RSA cipher.

Mathematics Subject Classification: Primary: 11A41; Secondary: 11A51

Keywords: primality testing, factorization, cryptography

1 Introduction

Primality testing is now one of the most dynamic areas of number-theoretic research. The power of modern computers and development of sophisticated algorithms have made the recognition of primes and factoring of composite integers highly efficient to the point that would have been thought infeasible only a generation ago. These algorithms are important in the modern day for ensuring that sensitive data is protected in electronic transmissions for government, industry, and the military. Much of what follows is adapted from [4]

2 Primality and Compositeness Tests

In general, factorization methods are quite time consuming, whereas deciding whether a given n is composite or prime is much more efficient. Part of the reason is that a test for recognizing primes, which are not attempts to factor n , and which determine n to be composite, do not provide the factors of n . Two tests for recognizing primes are given as follows.

- (1) The test has a condition for compositeness. If n satisfies the condition, then n *must* be composite. If n fails the condition, it might still be composite (with low probability). In other words, a successful completion of the test — satisfying the condition — always guarantees that n is composite; whereas an unsuccessful completion of the test — failing to satisfy the condition — does *not* prove that n is prime.
- (2) The test has a condition for primality. If n satisfies the condition, then n *must* be prime. If it fails the condition, then n *must* be composite.

We call (1) a *compositeness test* and (2) a *primality test*.

Factorization methods may be used as compositeness tests, but quite expensive ones, as we discussed above. Thus, they may be used only as compositeness tests, and only to find very small factors.

Primality tests, as we have defined them, are sometimes called *primality proofs*, which are typically either complicated to apply or else are applicable only to special numbers such as Fermat numbers, those of the form $2^{2^n} + 1$. In other words, they sacrifice either speed or generality, but always provide a correct answer without failure. We look at one, which employs the converse of Fermat's little theorem. The following is attributable to D.H. Lehmer, M. Kraitchik, and others.

◇ Primality Test via the Converse of Fermat's Little Theorem

Suppose that $n \in \mathbb{N}$ with $n \geq 3$. Then n is prime if and only if there exists an $m \in \mathbb{N}$ such that $m^{n-1} \equiv 1 \pmod{n}$, but $m^{(n-1)/q} \not\equiv 1 \pmod{n}$ for any prime $q \mid (n-1)$.

A major pitfall with the above primality test is that we must have knowledge of a factorization of $n-1$, so it works well on special numbers such as Fermat numbers, for instance. However, the above is a general “proof” that n is prime since the test finds an element of order $n-1$ in $(\mathbb{Z}/n\mathbb{Z})^*$, namely a primitive root modulo n . Furthermore, it can be demonstrated that if we have a factorization of $n-1$ and n is prime, then the above primality test can be employed to prove that n is prime in polynomial time; but if n is composite the algorithm will run without bound, or *diverge*.

Primality proofs sacrifice one of speed or generality. For instance, there is the Lucas-Lehmer test for Mersenne numbers, Pocklington's theorem, Proth's theorem, and Pepin's theorem, all of which the reader may see in detail by consulting [5], for instance.

We will be concerned herein with tests that are used in practice. Usually, these are tests that are simple, generally applicable, and efficient, but unlike the aforementioned tests, they sometimes fail. These are the compositeness tests. Note that in a compositeness test, failure of the *test* means that n

does not satisfy the condition for compositeness *and* n is composite. In other words, failure results in a composite number being indicated as a prime, but never is a prime indicated as a composite number. The reason is that the condition is a “proof of compositeness” in the sense that if the condition is satisfied, n is forced to be composite. However, the converse is false. Composite numbers may fail to satisfy the condition. We may again employ the converse of Fermat’s little theorem as an illustration.

◇ **Fermat’s Little Theorem as a Compositeness Test**

If $n \in \mathbb{N}$, $a \in \mathbb{Z}$ with $\gcd(a, n) = 1$, and

$$a^{n-1} \not\equiv 1 \pmod{n}, \quad (2.1)$$

then n is composite.

Note that any n satisfying condition (2.1) *must* be composite by Fermat’s little theorem. An application of the above is an interpretation of Lucas’ test as a compositeness test by letting n be odd and $a = 2$. See [2] for a recent article by John Brillhart on this famous test by Lucas. He argues that a primality test is an algorithm “whose steps verify the hypothesis of a theorem whose conclusion is “ n is prime.” which is consistent with our definition.

If n fails condition (2.1), then this is *not* a proof that n is prime. For instance, $2^{340} \equiv 1 \pmod{341}$, yet $341 = 11 \cdot 31$. Such numbers that fail condition (2.1), and are composite are called *pseudoprimes*. In fact, there are composite numbers for which the choice of the base a is irrelevant in the sense that they will *always* fail the test. For instance, $a^{541} \equiv a \pmod{541}$ for any $a \in \mathbb{Z}$, yet $541 = 3 \cdot 11 \cdot 17$. This is an example of a *Carmichael number*. Moreover, there are known to be infinitely many Carmichael numbers (see [1]). This shows, in the extreme, that Fermat’s little theorem may *not be used as a primality test*. However, the following is a well-known and utilized primality test.

◇ **The Miller-Selfridge-Rabin Primality Test**

Let $n - 1 = 2^t m$ where $m \in \mathbb{N}$ is odd and $t \in \mathbb{N}$. The value n is the input to be tested by executing the following steps.

- (1) Choose a random integer a with $2 \leq a \leq n - 2$.
- (2) Compute

$$x \equiv a^m \pmod{n}.$$

If

$$x \equiv \pm 1 \pmod{n},$$

then terminate the algorithm with

“ n is probably prime”.

If $t = 1$, terminate the algorithm with

“ n is definitely composite.”

Otherwise, set $j = 1$ and go to step (3).

(3) Compute

$$x \equiv a^{2^j m} \pmod{n}.$$

If $x \equiv 1 \pmod{n}$, then terminate the algorithm with

“ n is definitely composite.”

If $x \equiv -1 \pmod{n}$, terminate the algorithm with

“ n is probably prime.”

Otherwise set $j = j + 1$ and go to step (4).

(4) If $j = t - 1$, then go to step (5). Otherwise, go to step (3).

(5) Compute

$$x \equiv a^{2^{t-1} m} \pmod{n}.$$

If $x \not\equiv -1 \pmod{n}$, then terminate the algorithm with

“ n is definitely composite.”

If $x \equiv -1 \pmod{n}$, then terminate the algorithm with

“ n is probably prime.”

If n is declared to be “probably prime” with base a by the Miller-Selfridge-Rabin test, then

n is said to be a *strong pseudoprime to base a* .

Thus, the above test is often called the *strong pseudoprime test* in the literature. The set of all pseudoprimes to base a is denoted by $\text{spsp}(a)$.

Let us look a little closer at the above test to see why it is possible to declare that “ n is definitely composite” in step (3). If $x \equiv 1 \pmod{n}$ in step (3), then for some j with $1 \leq j < t - 1$:

$$a^{2^j m} \equiv 1 \pmod{n}, \text{ but } a^{2^{j-1} m} \not\equiv \pm 1 \pmod{n}.$$

Thus, it can be shown that $\gcd(a^{2^{j-1} m} - 1, n)$ is a nontrivial factor of n . Hence, if the Miller-Selfridge-Rabin test declares in step (3) that “ n is definitely composite”, then indeed it is. Another way of saying this is that if n is prime, then

Miller-Selfridge-Rabin will declare it to be so. However, if n is composite, then it can be shown that the test fails to recognize n as composite with probability at most $(1/4)$. This is why the most we can say is that “ n is probably prime”. However, if we perform the test r times for r large enough, this probability $(1/4)^r$ can be brought arbitrarily close to zero. Moreover, at least in practice, using the test with a single choice of a base a is usually sufficient.

Also, in step (5), notice that we have not mentioned the possibility that

$$a^{2^{t-1}m} \equiv 1 \pmod{n}$$

specifically. However, if this did occur, then that means that in step (3), we would have determined that

$$a^{2^{t-2}m} \not\equiv \pm 1 \pmod{n},$$

from which it follows that n cannot be prime. Furthermore, by the above method, we can factor n since $\gcd(a^{2^{t-2}m} - 1, n)$ is a nontrivial factor. This final step (4) is required since, if we get to $j = t - 1$, with $x \not\equiv \pm 1 \pmod{n}$ for any $j < t - 1$, then simply invoking step (3) again would dismiss those values of $x \not\equiv \pm 1 \pmod{n}$, and this would not allow us to claim that n is composite in those cases. Hence, it allows for more values of n to be deemed composite, with certainty, than if we merely performed step (3) as with previous values of j .

3 Generating Primes

Selecting random primes for, say, the RSA cipher is important since bad choices can be made that compromise the algorithm. Safe primes are those primes p for which $(p - 1)/2$ is also prime. Such primes are important in selecting an RSA modulus $n = pq$ since, if p and q are safe primes, then RSA is not vulnerable to $p - 1$ factoring method discovered by Pollard in [6] or the $p + 1$ factoring method found by Williams in [7]. In general, having safe primes in the modulus makes it more difficult to factor. However, finding such primes is also more difficult.

◇ Algorithm for Generating (Probable) Safe Primes

Let b be the input bitlength of the required prime. Execute the following steps.

- (1) Select a $(b - 1)$ -bit odd random $n \in \mathbb{N}$ and a smoothness bound B (determined experimentally).
- (2) Trial divide n by primes $p \leq B$. If n is divisible by any such p , go to step (1). Otherwise, go to step (3).

- (3) Use the Miller-Selfridge-Rabin test to test n for primality. If it declares that “ n is probably prime”, then go to step (4). Otherwise, go to step (1).
- (4) Compute $2n + 1 = q$ and use the Miller-Selfridge-Rabin test on q . If it declares q to be a probable prime, terminate the algorithm with q as a “probable safe prime”. Otherwise go to step (1).

There are primes that have even more constraints to ensure security of the RSA modulus. They are given as follows.

Definition 3.1 (Strong Primes)

A prime p is called a strong prime if each of the following hold.

- (1) $p - 1$ has a large prime factor q .
- (2) $p + 1$ has a large prime factor r .
- (3) $q - 1$ has a large prime factor s .

The following algorithm was initiated in [3].

◊ **Gordon’s Algorithm for Generating (Probable) Strong Primes**

- (1) Generate two large (probable) primes $r \neq s$ of roughly equal bitlength using the Miller-Selfridge-Rabin test.
- (2) Select the first prime in the sequence $\{2js + 1\}_{j \in \mathbb{N}}$, and let

$$q = 2js + 1$$

be that prime.

- (3) Compute $p_0 \equiv r^{q-1} - q^{r-1} \pmod{rq}$.
- (4) Find the first prime in the sequence $\{p_0 + 2iqr\}_{i \in \mathbb{N}}$, and let

$$p = p_0 + 2iqr$$

be that prime, which is a strong prime.

Although it is possible to generate primes that are both safe and strong, the algorithms are not as efficient as Gordon’s algorithm. Furthermore, choosing random primes large enough will generally thwart direct factoring attacks. The following provides a mechanism for generating large random primes.

◊ **Large (Probable) Prime Generation**

We let b be the input bitlength of the desired prime and let B be the input smoothness bound (empirically determined). Execute the following steps.

- (1) Randomly generate an odd b -bit integer n .
- (2) Use trial division to test for divisibility of n by all odd primes no bigger than B . If n is so divisible, go to step (1). Otherwise go to step (3).
- (3) Use the Miller-Selfridge-Rabin to test n for primality. If it is declared to be a probable prime, then output n as such. Otherwise, go to step (1).

◇ Large (Provable) Prime Generation

Begin with a prime p_1 , and execute the following steps until you have a prime of the desired size. Initialize the variable counter $j = 1$.

- (1) Randomly generate a small odd integer m and form $n = 2mp_j + 1$.
- (2) If $2^{n-1} \not\equiv 1 \pmod{n}$, then go to step (1). Otherwise, go to step (3).
- (3) Using the primality test on page 470, with prime bases $2 \leq a \leq 23$, if for any such a ,

$$a^{(n-1)/p} \not\equiv 1 \pmod{n}$$

for any prime p dividing $n - 1$, then n is prime. If n is large enough, terminate the algorithm with output n as the provable prime. Otherwise, set $n = p_{j+1}$, $j = j + 1$, and go to step (1). If the test fails go to step (1).

Note that since we have a known factorization of $n - 1$ in the above algorithm, and a small value of m to check, then the test is simple and efficient.

4 Decision Problem or Primality Test?

Earlier we discussed Lucas' test as an application of Fermat's compositeness criterion (2.1) (contrapositively speaking). There is, however, a brand of opinion that Lucas' test is really a decision problem. Here is the reasoning.

Lucas' test tells us to compute $2^{n-1} \pmod{n}$. If

$$2^{n-1} \not\equiv 1 \pmod{n},$$

then we know that n is not prime and send it off to some factoring routine.

If

$$2^{n-1} \equiv 1 \pmod{n},$$

then we send it off for primality testing. Hence, the Lucas test may be viewed as a decision problem on whether to send n for primality testing or factoring. Since decision problems are "yes-no" issues, we phrase the question as

Do we send n for primality testing?

If the answer is yes, we do so, and if the answer is no, we send it to a factoring algorithm.

There is merit to the above argument. Attendant with the above viewpoint is the opinion that assigning probabilities or improving such estimates has nothing to do with primality testing. Thus, this school of thought would not consider the Miller-Selfridge-Rabin algorithm to be a test that in any way assists with practical primality testing. Given that MSR does not satisfy our criterion (2) given above this viewpoint also has some merit. That said, this does not prevent the use of such algorithms in practice.

The aforementioned point of view is presented for mental fodder and general interest in what is often a contentious topic.

Acknowledgements: The author's research is supported by NSERC Canada grant # A8484.

References

- [1] W.R. Alford, A. Granville, and C. Pomerance, *There are infinitely many Carmichael numbers*, Ann. Math. **140** (1994), 703–722.
- [2] J. Brillhart, *Commentary on Lucas' Test*, Fields Inst. Comm. **41** (2004), 103–109.
- [3] J. Gordon, *Strong primes are easy to find*, in **Advances in Cryptology**, EUROCRYPT '84, Springer-Verlag, Berlin, LNCS **209** (1985), 216–223.
- [4] R.A. Mollin, **Codes – The Guide to Secrecy from Ancient to Modern Times**, Chapman and Hall/CRC Press, Boca Raton, New York, London, Tokyo (2005).
- [5] R.A. Mollin, **An Introduction to Cryptography, Second Edition**, Chapman and Hall/CRC Press, Boca Raton, New York, London, Tokyo (2006).
- [6] J.M. Pollard, *An algorithm for testing the primality of any integer*, Bull. London Math. Soc. **3** (1971), 337–340.
- [7] H.C. Williams, *A $p + 1$ method of factoring*, Math. Comp. **39** (1982), 225–234.

Received July 23, 2007