RESEARCH REVIEW JOURNALS
Volume-03
Issue-12
December -2018

ISSN: 2455-3085 (Online)
RESEARCH REVIEW International Journal of Multidisciplinary
www.rrjournals.com [UGC Listed Journal]

Primality testing methods involve arithmetic in algebraic number theory

Rupen Chatterjee

Department of Mathematics, Nabagram Hiralal Paul College, Nabagram, Hooghly, West Bengal Pin:71224, India (Affiliated by Calcutta University)

ARTICLE DETAILS

Article History
Published Online: 10 December 2018

Keywords
Primality Testing, algebraic, theory.

ABSTRACT

Primality testing methods is an algorithm for determining whether an input number is prime. Among 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. Algebra is one of the broad parts of mathematics, together with number theory, geometry and analysis.As such, it includes everything from elementary equation solving to the study of abstractions such as groups, rings, and fields. The word algebra is also used in certain specialized ways. A special kind of mathematical object in abstract algebra is called an "algebra", and the word is used, for example, in the phrases linear algebra and algebraic topology. In this paper, we discuss primality testing methods involve arithmetic which are best understood in the context of algebraic number theory.

1. Introduction

We know that a prime number is a natural number $p \neq 1$ for which the only divisors are 1 and p. This essentially means that for a prime number p, gcd(a, p)=1 $\forall a \in N$ and $1 \leq a \leq (p-1)$. So the value of Euler's totient function for a prime p, $\varphi(p)$ equals $(p-1)$ as all the $(p-1)$ values of a in (1,2,...p − 1) satisfy gcd(a, p)=1. A primality test, the topic of this paper, is simply an algorithm that tests, either probabilistically or deterministically, whether or not a given input number is prime. A general primality test does not provide us with a prime factorisation of a number not found to be prime, but simply labels it as composite. In cryptography, for example, we often need the generation of large primes and one technique for this is to pick a random number of requisite size and detrmine if it's prime. The larger the number, the greater will be the time required to test this and this is what prompts us to search for efficient primality tests that are polynomial in complexity. Note that the desired complexity is logarithmic in the number itself and hence polynomial in its bit–size as a number n requires O(log n) bits for its binary representation.

Primality testing of large numbers is *very* important in many areas of mathematics, computer science and cryptography. For example, in public-key cryptography, if we can find two large primes p and q, each with 100 digits or more, then we can get a composite

$n= p * q$

with 200 digits or more. This composite n can be used to encode a message securely even when n is made public. The message cannot be decoded without knowledge of the prime factors of n. of course, we can try to use a modern integer factorization method such as the Elliptic Curve Method to factor n and to get its prime factors p and q, but it would take about 20 million years to complete the job even on a supercomputer. Thus, it is practically impossible to decode the message. Another good example is the searching for amicable numbers. In the following algebraic method for generating amicable numbers,if we can make sure thatthe following four integers p, q, r, s

$p=2^x * g-1$

$q = 2^y + (2^{n+l} - 1) * g$ where $0<x<n$

$r= 2^{n-y} * g * q-1$

$g=2^{n-x}+ 1$

$S =2^{n-y+x} * g2 * q - 1$ $0<y<n$

are all primes, then the pair(m, n) = $(2^n qpr, 2^n qs)$

is an amicable pair. Thus, searching for amicable numbers is often the same as the primalitytesting of some related integers.

Primality testing is one of the oldest problems as well as open problems in mathematics, which goes back to the ancient Greeks about 2000 years ago. The problem can be simply described as follows:

Input: n (n ∈ Natural Numbers and n > 1).

{ Yes, if n ∈ Primes,
{ Output: No, if n ∈ Composites.

Unfortunately, it is not a simple matter to determine whether or not a random integer *n* is prime,particularly when n is very large. An efficient algorithm for primality testing from the complexitypoint of view would have to run in $O(\log^k n)$ steps, for some fixed *k*. But unfortunately, no such *deterministic* algorithm exists for random integer n, although, for example, Miller [2] showed that n can be checked in $O(\log^5 n)$ steps, assuming the truth of the unproved Extended Riemann Hypothesis (ERH). Recently, many of the modern primality testing algorithms have been incorporated in Computer Algebra Systems (CAS) such as Axiom and Maple (see [3,4] for a reference) as a standard. In this paper, we shall discuss primality testing of large numbers in Maple.

For some forms of numbers, it is possible to find 'short-cuts' to a primality proof. This is the case for the Mersenne numbers. In fact, due to their special structure, which allows for easier verification of primality, the six largest known prime numbers are all Mersenne number. There has been a method in use for some time to verify primality of Mersenne numbers, known as the Lucas–Lehmer test. This test does not rely on elliptic curves. However we present a result where numbers of the form $N = 2^k n - 1$ where $k,n \in Z$, $K \geq 2$, n odd can be

proven prime (or composite) using elliptic curves. Of course this will also provide a method for proving primality of Mersenne numbers, which correspond to the case where n = 1. There is a method in place for testing this form of number without elliptic curves (with a limitation on the size of k) known as the Lucas–Lehmer–Riesel test. The following method is drawn from the paper Primality Test for $2^k n - 1$ using Elliptic Curves.

2. Methods for primality testing

The general strategy to test whether an integer n > 2 is prime or composite is to choose some property, say A, implied by primality, and to search for a counterexample a to this property for the number n, namely some a for which property A fails. We look for properties for which checking that a candidate a is indeed a countexample can be done quickly. Typically, together with the number n being tested for primality, some candidate counterexample a is supplied to an algorithm which runs a test to determine whether a is really a counterexample to property A for n. If the test says that a is a counterexample, also called a witness, then we know for sure that n is composite. If the algorithm reports that a is not a witness to the fact that n is composite, does this imply that n is not prime.

This is because, there may be some composite number n and some candidate counterexample a for which the test says that a is not a countexample. Such a number a is called a liar . The other reason is that we haven't tested all the candidate counterexamples a for n. The remedy is to make sure that we pick a property A such that if n is composite, then at least some candidate a is not a liar, and to test all potential countexamples a. The difficulty is that trying all candidate countexamples can be too expensive to be practical. The following analogy may be helpful to understand the nature of such a method. Suppose we have a population and we are interested in determining whether some individual is rich or not (we will say that someone who is not rich is poor). Every individual n has several bank accounts a, and there is a test to check whether a bank account a has a negative balance. The test has the property that if it is applied to an individual n and to one of its bank accounts a, and if it is positive (it says that account a has a negative balance), then the individual n is definitely poor. Note that we are assuming that a rich person is honest, namely that all bank accounts of a rich person have a nonnegative balance. This may be an unrealistic assumption. But if the test is negative (which means that account a has a nonnegative balance), this does not imply that n is rich. The problem is that the test may not be 100% reliable. It is possible that an individual n is poor, yet the test is negative for account a (account a has a nonnegative balance). We may also not have tested all the accounts of n. One way to deal with this problem is to use probabilities. If we know that the conditional probability that the test is positive for some account a given that n is poor is greater than $p \geq 1/2$, then we can apply the test to l accounts chosen independently at random. It is easy to show that the conditional probability that the test is negative l times given that an individual n is poor is less than $(1-p)^l$. For p close to 1 and l large enough, this probability is very small. Thus, if we have high confidence in the test (p is close to 1) and if an individual n is poor, it is very unlikely that the test will be negative l times. Actually, what we would really like to know is the conditional probability that the individual n is rich given that the test is negative l times. If the probability that an individual n is rich is known, then the above conditional probability can be computed using Bayes's rule. We will show how to do this later. A Monte Carlo algorithm does not give a definite answer. However, if l is large enough (say l = 100), then the conditional probability that the property of interest holds (here, n is rich), given that the test is negative l times, is very close to 1. In other words, if l is large enough and if the test is negative ` times, then we have high confidence that n is rich.

There are two classes of primality testing algorithms:

1.Algorithms that try all possible countexamples, and for which the test does not lie. These algorithms give a definite answer: n is prime or n is composite. Until 2002, no algorithms running in polynomial time, were known. The situation changed in 2002 when a paper with the title "primes is in *P*," by Agrawal, Kayal and Saxena, appeared on the website of the Indian Institute of Technology at Kanpur, India. In this paper, it was shown that testing for primality has a deterministic (nonrandomized) algorithm that runs in polynomial time.

2. Randomized algorithms. To avoid having problems with infinite events, we assume that we are testing numbers in some large finite interval L. Given any positive integer $m \in L$, some candidate witness a is chosen at random. We have a test which, given m and a potential witness a, determines whether or not a is indeed a witness to the fact that m is composite. Such an algorithm is a Monte Carlo algorithm, which means the following

a) If the test is positive, then $m \in L$ is composite. In terms of probabilities, this is expressed by saying that the conditional probability that $m \in L$ is composite given that the test is positive is equal to 1. If we denote the event that some positive integer $m \in L$ is composite by C, then we can express the above as

Pr(C | test is positive) = 1

b)If $m \in L$ is composite, then the test is positive for at least 50% of the choices for a. We can express the above as

Pr(test is positive | C) ≥ 1 /2

This gives us a degree of confidence in the test.

The contrapositive of (a) says that if $m \in L$ is prime, then the test is negative. If we denote by P the event that some positive integer $m \in L$ is prime, then this is expressed as

Pr(test is negative | P) = 1

If we repeat the test l times by picking independent potential witnesses, then the conditional probability that the test is negative l times given that n is composite, written

Pr(test is negative l times | C), is given by

$$\Pr(\text{test is negative } l \text{ times} \mid C) = \Pr(\text{test is negative} \mid C)^l$$
$$= (1 - \Pr(\text{test is positive} \mid C))^l$$
$$\leq (1 - 1/2)^l$$
$$= (1/2)^l$$

Primality Testing Using Randomized and Algorithms

The problem of distinguishing prime numbers from composite numbers and resolving the latter into their prime factors is known to be one of the most important and useful in

arithmetic. It has engaged the industry and wisdom of ancient and moderm geometers to such an extent that it would be superfluous to discuss the problem at length. Nevertherless we must confess that all methods that have been proposed thus far are either restricted to very special cases or are so laborious and difficult that even for numbers that do not exceed the limits of tables constructed by estimable men, they try the patience of even the practiced calculator. And these methods do not apply at all to larger numbers ... The techniques that were previously known would require intolerable labor even for the most indefatigable calculator."

The problem of determining whether a given integer is prime is one of the better known and most easily understood problems of pure mathematics. This problem has caught the interest of mathematicians again and again for centuries. However, it was not until the 20th century that questions about primality testing and factoring were recognized as problems of practical importance, and a central part of applied mathematics. The advent of cryptographic systems that use large primes, such as RSA, was the main driving force for the development of fast and reliable methods for primality testing. Indeed, as we saw in earlier sections of these notes, in order to create RSA keys, one needs to produce large prime numbers. How do we do that?

One method is to produce a random string of digits (say of 200 digits), and then to test whether this number is prime or not. As we explained earlier, by the Prime Number Theorem, among the natural numbers with 200 digits, roughly one in every 460 is a prime. Thus, it should take at most 460 trials (picking at random some natural number with 200 digits) before a prime shows up. Note that we need a mechanism to generate random numbers, an interesting and tricky problem, but for now, we postpone discussing random number generation. It remains to find methods for testing an integer for primality, and perhaps for factoring composite numbers. In 1903, at the meeting of the American Mathematical Society, F.N. Cole came to the blackboard and, without saying a word, wrote down

$2^{67} - 1 = 147573952589676412927 = 193707721 \times 761838257287$

and then used long multiplication to multiply the two numbers on the right-hand side to prove that he was indeed correct. Afterwards, he said that figuring this out had taken him "three years of Sundays." Too bad laptops did not exist in 1903. The moral of this tale is that checking that a number is composite can be done quickly (that is, in polynomial time), but finding a factorization is hard. In general, it requires an exhaustive search. Another important observation is that most efficient tests for compositeness do not produce a factorization. For example, Lucas had already shown that $2^{67} - 1$ is composite, but without finding a factor. In fact, although this has not been proved, factoring appears to be a much harder problem than primality testing, which is a good thing since the safety of many cryptographic systems depends on the assumption that factoring is hard! As we explained in the introduction, most algorithms for testing whether an integer n is prime actually test for compositeness. This is because tests for compositeness usually try to find a counterexample to some property, say A, implied by primality. If such a counterexample can be guessed, then it is cheap to check that property A fails, and then we know for sure that n is composite. We also have a witness (or certificate) that n is composite. If the algorithm fails to show that n is composite, does this imply that n is prime. Unfortunately, no. This is because, in general, the algorithm has not tested all potential countexamples. So, we fix the algorithm.One possibility is to try systematically all potential countexamples. If the algorithm fails on all counterexamples, then the number n has to be prime. The problem with this approach is that the number of counterexamples is generally too big, and this method is not practical. Methods of this kind are presented in Crandall and Pomerance and Ribenboim. Another approach is to use a randomized algorithm. Typically, a counterexample is some number a randomly chosen from the set {2, . . . , n -2}, and the algorithm performs a test on a and n to determine whether a is a counterexample. If the test is positive, then for sure n is composite, and a is a witness to the fact that n is composite. If the test is negative, then the algorithm does not find n to be composite, and we can call it again several times, each time picking (independently from previous trials) another random number a. If the algorithm ever reports a positive test, then for sure n is a composite. But what if we call thealgorithm say 20 times, and every time the test is negative (which means that the algorithm does not find n to be composite 20 times).

Not necessarily, because the test performed by the algorithm may not be 100% reliable. If n is prime, the test performed by the algorithm on every a is negative (as it should), but there may be some composite n and some a for which the test is negative. Such a number a is called a liar , because it fools the test. Even though n is composite, a does not trigger the test to be positive, to indicate that n is indeed composite. But if the conditional probability that the test performed by the algorithm is positive given that n is composite is large enough, say at least 1/2, then it can be shown that the conditional probability that n is composite, given that the test performed by the algorithm is negative 20 times, is less than $\ln(n) \cdot (1/2)^{20}$.1 In summary, if we run the algorithm l times (for l large enough, say l = 100) on some number n, and if each time the test performed by the algorithm is negative, then we can be very confident that n is prime. Such kind of randomized algorithm is called a Monte Carlo algorithm. Several randomized algorithms for primality testing have been designed, including the Miller–Rabin and the Solovay–Strassen tests.

Strong Pseudoprime

A positive integer n with $n - 1 = d * 2^j$ and *d* odd, is called a strong probableprime to the base *a* if it passes the strong pseudoprimality test described above (i.e., the lastterm in the Miller-Rabin sequence is 1, and the first occurrence of 1either is the first term or ispreceded by -1). A strong probable prime to the base a is called a strong pseudoprime to thebase *a* if it is a composite.

Although very few composites can pass the strong pseudoprimality test, the test itself is notdeterministic, but probabilistic. For example, the composite n = 2047 23 * 89 can pass the strong pseudoprimality test, because $n - 1 = 2^1 * 1023$, d = 1023 and the Miller-Rabin sequence is $2^{1023} \equiv$ l(mod2047), $2^{2046} \equiv$ l(mod2047). So n = 2047 is a *strong pseudoprime to the base 2.*Thus, we cannot conclude that n is prime just by a strong primality test, we will need some other tests as well. One of the other tests is the Lucas (pseudoprimality) test.

Noted that there is a special Lucas test (often called *Lucas-Lehmer test)* for Mersenne primes, based on the following theorem.

ECPP

Elliptic curve primality testing techniques, or elliptic curve primality proving (ECPP), are among the quickest and most widely used methods in primality proving. It is an idea put forward by Shafi Goldwasser and Joe Kilian in 1986 and turned into an algorithm by A. O. L. Atkin the same year. The algorithm was altered and improved by several collaborators subsequently, and notably by Atkin and François Morain [de], in 1993. The concept of using elliptic curves in factorization had been developed by H. W. Lenstra in 1985, and the implications for its use in primality testing (and proving) followed quickly.

Primality testing is a field that has been around since the time of Fermat, in whose time most algorithms were based on factoring, which become unwieldy with large input; modern algorithms treat the problems of determining whether a number is prime and what its factors are separately. It became of practical importance with the advent of modern cryptography. Although many current tests result in a probabilistic output (*N* is either shown composite, or probably prime, such as with the Baillie–PSW primality test or the Miller–Rabin test), the elliptic curve test proves primality (or compositeness) with a quickly verifiable certificate.

3. Primality Test and Results

I Primality test and Result

Strong Pseudoprimality test and Lucas tests , we introduce some basic concepts and ideas of probable primes, pseudo primesand pseudoprimality tests, which will be used throughout the paper.

THEOREM. (Fermats's Theorem)

Ifp is prime and gcd(a,p) = 1, *then*

$a^{p-1} \equiv 1 (\mathrm{mod} p)$.

Most modern primality testing algorithms depend in some way on the converse (an immediate corollary) of Fermat's Theorem.

COROLLARY. (Converte of Fermats's Theorem -Fermat test)

Let n be an odd positive integer. If gcd(a, n) = 1 and

$a^{n-1} \neq 1 \pmod n$,

then n is composite.

By Corollary, we know that if there exists on a with 1 < a < n, gcd (a, n) = 1 and $a^{n-1} \neq 1 (\mathrm{mod} n)$, then n must be composite. What happens if we find a number *n* such that $a^{n-1} \equiv (\mathrm{mod}^{n-1})$. Can we conclude that n is certainly a prime. The answer is unfortunately not, because n sometimes is indeed a prime, but sometimes is not! This leads to the following important concepts of probable primes and pseudo primes.

If $a^{n-1} \equiv 1 (\mathrm{mod} n)$, then we call n a (Fermat) probable prime to the base a. A(Fermat) probable prime n to the base a is called a (Fermat) pseudoprime to the base a if n is composite.

For example, $2^{1387-1} \equiv 1 (\mathrm{mod}\ 1387)$. Thus, 1387 is a Fermat probable prime to the base 2. But since 1387 = 19 * 73 is composite, then it is Fermat pseudoprime to the base 2. A further and immediate improvement over the Fermat test is the strong pseudoprimality test (often called the Miller-Rabin test, or just the strong test). We describe it in the following algorithmic form.

ALGORITHM (Strong Pseudoprimality Test)

[S1] Let n be an odd number, and the base a be a random number in the range 1 < a < n. Find

j and d with d odd, so that $n - 1 = 2^j d$.

[S2]Compute a d (modn). Ira a (modn) - ±1 (modn), then n is a strong probable prime andoutput "Yes"; stop.

[S3]Square a^d to compute $a^{2d} (\mathrm{mod}\ n)$. If $a^{2d} (\mathrm{mod}\ n) \equiv 1 (\mathrm{mod}\ n)$, then n is composite andoutput "No"; stop. If $a^{2d} (\mathrm{mod} n) = -1 (\mathrm{mod} n)$, then n is a strong probable prime and output "Yes"; stop.

[S4]Repeat step S3 with a^{2d} replaced by

$a^{4d}, a^{8d}, \ldots\ldots, a^{2i-1d}$

(Note that the sequence

$a^d, a^{2d}, a^{4d}, a^{8d}, \ldots\ldots, a^{2i-1d}$

is often called the Miller-Rabin sequence.)

THEOREM (Lucas - Lehmer test for Mersenne primes)

Let p be an odd prime. Definethe Lucas sequence $\{U_k\}$ by

$$\begin{cases} U_0 = 4, \\ U_{k+1} \equiv (U_k^2 - 2) (\mathrm{mod}\, 2^p - 1). \end{cases}$$

Then $2^p - 1$ is prime if and only if $U_{p-2} \equiv 0 (\mathrm{mod} 2^p - 1)$.

For example, suppose we wish to test the primality of 2^7-1. We first compute the Lucassequence $\{U_k\}$ for 2^7-1 (k = 0, 1 . . . . , p-2 = 5):

Since $U_{p-2} \equiv 0 (\mathrm{mod}\ 2^p-1)$, then $2^{7-}1$ is a prime.

The Lucas test we are interested in here is a more general one. It is an analog of Fermat's theorem for Lucas sequences (see [5] or [6] for a reference).

[**S5**] If the procedure has not alreadyterminated, then n is composite and output "No".

II Primality test and Result

Now the simplest primality tests from high school. Given any number n, we can check all numbers less than or equal to √ n and see if any of them divides n. If at least one of them does, the algorithm outputs composite and otherwise, it outputs prime. This follows from the fact that if n is indeed composite, it must be possible for us to factor it into at least two factors and at least one of them must be less than or equal to √ n. However, this test, involving √ n operations, is Ω(√ n) and hence the algorithm is essentially exponential in bit-size. The algorithm can be made more efficient, however, by skipping all even numbers below √ n except 2, because non-divisibility by 2 guarantees non-divisibility by all other even even numbers. Another better modification is obtained by noting that all primes exceeding 3 are of the form 6k ± 1 and this restriction further narrows down the list of possible primes,

as now only 2 of any 6 consequtive integers are candidates for being a prime. This algorithm can now be presented in pseudocode as follows:

1.Input n. If n = 2 or n = 3, output PRIME. If n ≠ 2 and n ≡ 0 (mod 2), output COMPOSITE.

2.Elif n 6≡ 1 (mod 6) or if n 6≡ 5 (mod 6), output COMPOSITE.

3. Else : for i in range (2, b √ nc+1) and i 6≡ 0 (mod 2), if n ≡ 0 (mod i), output COMPOSITE.

4. Else : i = i + 1. If i=(int√ n)+1, output PRIME.

The time-complexity of the algorithm is readily seen to be Ω(√ n). A method of finding prime numbers from antiquity is the Sieve of Eratosthenes.

III Primality test and Result

we know that even though 391 ≡ 3 (mod 91) and 2341 ≡ 2 (mod 341), 91 and 341 are not primes. In fact, these numbers are called pseudoprimes to the chosen base, i.e., the value of a. So we conclude that 91 and 341 are pseudoprimes to base 3 and base 2 respectively. However, since 2341 6≡ 2 (mod 341) and 391 6≡ 3 (mod 91), we may correctly pronounce these two numbers as composite by testing out both the bases 2 and 3 for them and, in general, 4 we may hope that if we try out all values of a with 1 ≤ a ≤ (n − 1), n being the input number, we will always be accurate with our conclusions from this test. This will be the case if there is no number n that is a pseudoprime or a Fermat pseudoprime to all values of a with a ≤ (n−1) and gcd(a, n)=1. However, there do exist such numbers and, for these numbers, we will not be able to predict their compositeness even by carrying out Fermat's test for all bases. Examples of such numbers were first found by Carmichael and hence, these numbers are called Carmichael numbers.

Definition : A Carmichael number or an absolute Fermat pseudoprime is a composite natural number n satisfying the congruence a n−1 ≡ 1 (mod n) for all a with gcd(a, n)=1. Definitely, for all a with 1 ≤ a ≤ (n − 1), it satisfies a n ≡ a (mod n). Korselt's criterion is one which imparts a classifying characteristic to all Carmichael numbers, the first of which is 561. Fact (Korselt's criterion) : If n is a Carmichael number, i.e., if for all a with gcd(a,n)=1, a n−1 ≡ 1 (mod n), then it satisfies the following two conditions:

(a) n must be square-free, i.e., for any prime p, if. n is ***divisible*** by but n is not divisible by p^2

(b) If p is a prime with p|n, then (p − 1)|(n − 1). The first Carmichael number is 561=3×11 × 17 and it's easy to see that it indeed satisfies Korselt's criterion. There is also a result which says that any Carmichael number is a product of at least 3 odd primes, which, according to Korselt's criterion (a) above, must be distinct.

IV Primality test and Result

Maple is a very powerful computer algebra system developed by the Symbolic ComputationGroup at the University of Waterloo and the Institute for Scientific Computing at ETH Zurich. It can manipulate mathematical formulas following the rules of number theory, algebra, geometry, trigonometry, calculus and combinatory. In this section, we are only concerned with the primality test facility in the number theory package of Maple.For example, Pinch at Cambridge [4] tested the numbers in the followinglist y_0 (the first two are Fermat pseudoprimes to the base 2 and the other three axe Carmichaelnumbers):

2152302898747 = 6763 * 10627 * 29947
3474749660383 = 1303 *16927*157543
10710604680091 = 3739 *18691 * 153259
4498414682539051 = 46411 * 232051 * 417691
6830509209595831 = 21319 * 106591 * 3005839

and found that they all can pass the isprime test. That is, Maple declares the above five composites to be prime. But starting from Maple V Release 3, the isprime test uses a combination of a strong pseudoprimality test and a Lucas test. It is much more powerful and reliable than that in Maple. We tested the five numbers in $y_0$by Maple V Release 3, and found that they cannot pass the isprime test. That is, Maple this time declares the five numbers in y_0 to be composite. As we can see, these numbers are indeed composites, so Maple V Release 3 provides a powerful and reliable approach to the primality test of large numbers.

As mentioned previously; primality testing is a very important operation in searching for amicablenumbers. In a research project on algebraic methods for generating amicable numbers, we have tested three other lists of integers by using the isprime tes

List y_1: Four integers
9288811670405087
145135534866431
313887523966328699903,
455562336787531090452868968512225 27
of p, q, r, s in [1], which generate a new 65-digit amicable pair.

List y_2: 204 integers (see Table 1 in the Appendix) of sixty-eight q, r, s in [11], which generate
68 new large amicable pairs in the 101-122 digit range.

List y_3: Two large 520-digit numbers in [1]

6632285536963621091599720517630946848785159900258273537949139058912332902956 50_
9271649269780780600900085257209710528441948321598665854807136654409025661374 27_
0667658688272835179399068804314447608183257016726016120240820634875491616977 74_
3110984363557517151927286379149643480217362783804583033068892992150693096268 16_
8952017200647384662428772848776389139741063330922157771133640130874834678356 95_
*807181754057979471754499144424268957636699060565069202221342263517860285324742_*
9237872442965593215267325943862952255228142339076351
6327402753789224887018153257141755147949648160405944047993578483512642365264 06_
8452885320900204676028183611011210983041652561601556675790530882506715903223 09_
0575538247864438005782032162148726159096760992069458850601848793572743226449 97_

254248956714784431538486182141543538783008629621566451550927419407740660304484_
945027709696773434687534485995266004134857036148105086354503584538997217621468_
327457083003403742747130571751051829100781214766777504855678715294348177033954_
745814707897573279413323799432427031198196540702719 9

which generates the largest amicable pair with 1041 digits found by HolgerWiethaus inWest Germany.

All the 210 numbers in lists y1, y2, and y3 are found to be prime on Maple. Thetesting only takes about half hour on a parallel Silicon Graphics R4D/340S computer in the University of York Computing Centre. Notice that I have also tested these 210 numbers to be prime in Maple V Release 2 in 1993. As suggested by Bradley Lucier at Purdue University, I have confirmed that all the numbers in y1 and y2 are indeed prime on a Silicon Graphics R4D/340Scomputer in the University of York Computing Centre, by using a deterministic *elliptic curvetest* algorithm ECPP (Elliptic Curve Primality Proving) developed by Atkin and Morain [9]. Asfor the primality of the two large 520-digit numbers in list y3, the confirmation was actuallycompleted by F. Morain in France by using a new version of his ECPP program.

The biggest number we have tested on Maple V Release 3 is a 564-digit prime factor of the11th Fermat number F_{11}:

173462447179147555430258970864309778377421844723664084649347019061363579192879_
108857591038330408837177983810868451546421940712978306134189864280826014542758_
708589243873685563973118948869399158545506611147420216132557017260564139394366_
945793220968665108959685482705388072645828554151936401912464931182546092879815_
733057795573358504982279280090942872567591518912118622751714319229788100979251_
036035496917279912663527358783236647193154777091427745377038294584918917590325_
110939381322486044298573971650711059244462177542540706913047034664643603491382_
441723306598834177

The test only takes about 6 minutes of CPU time on a Silicon Graphics R4D/340S machine.Since no counterexample has been found for the isprime test in Maple V Release 3, we can have the following much stronger definition and result for probable primes.

Let n be a positive integer and n > 1. If n passes the isprimetest in Maple V Release 3, then n is called a Maple probable prime.

4. Conclusion

In this paper, we discussed primality testing methods involve arithmetic which are best understood in the context of algebraic number theory. Since the primality testing facility is prime in Maple V Release 3 is based on a combined use of a strong pseudoprimality test and a Lucas test, it is a very efficient and reliable test for large numbers. Number theoretic methods exist for testing whether a number is prime, such as the Lucas test ,Maple test, ECPP test, Pseudoprimality 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. These 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. No composite has been found that can pass the is prime test in Maple V Release 3. Our computation experience shows that the Maple primality test results are exactly the same as that obtained by the deterministic elliptic curve test in ECPP. This proves, at least from a practical point of view, that the is prime test in Maple V Release 3 is reliable. Our experience also shows that the Maple primality test is always far more efficient than ECPP, particularly for large numbers. For example, to test the two large 520-digit numbers in *y*3, Maple only needs a few minutes of CPU time on a Silicon Graphics R4D/340S computer, but ECPP will need several hours. But of course, we usually need to use an elliptic curve test or some other deterministic tests to confirm the results obtained by the Maple test. So we are finally approaching to a more practical and realistic primality test for large numbers (assume n is the integer to be tested):

MAPLE TEST { n is composite (100% correct)
==> { n is prime (error «10^{-15})

ECPP TEST { n is composite (100% correct)
== > { n is prime (100% correct)

References

1. S.Y. Yan and T.H. Jackson, A new large amicable pair, Computers Math. Applic. 27 (6), 1 3 (1994).
2. G.L. Miller, Riemann's hypothesis and tests for primality, Journal of Computer and System Science 13, 300-317 (1976).
3. J.H. Davenport, Primality testing revisited, In Proceedings of International Symposium of Symbolic and Algebraic Computations, ACM Press, 123-129, (1992).
4. R.G.E. Pinch, Some primality testing algorithms, Department of Pure Mathematics and Mathematical Statistics, University of Cambridge, (June 24, 1993).
5. I. Niven, H.S. Zuckerman and H.L. Montgomery, An Introduction to the Theory of Numbers, Fifth edition,John Wiley & Sons, (1991).
6. H. Reisel, Prime Numbers and Computer Methods for Factorization, Birkh~user, (1990).
7. R.J. Baillie and S.S. Wagstatf, Jr., Lucas pseudoprimes, Mathematics of Computation 35, 1391-1417 (1980).
8. C. Pomerance, J.L. Selfridge and S.S. Wagstaff, Jr., The pseudoprimes to 25.109, Mathematics of Computation 35, 1003-1026 (1980).
9. A.O.L. Atkin and F. Morain, Elliptic curves and primality proving, Department of Mathematics, University of Illinois at Chicago, (1991).
10. D.E. Knuth, The Art of Computer Programming: Seminumerical Algorithms, 2ndedition, Addison-Wesley, (1981).
11. S.Y. Yan, 68 new large amicable pairs, Computers Math. Applic. 28 (5), 71-74 (1994).

