Exploring Factorization and Primality *

Hamman W.Samuel †

January 2009

Abstract

There are various algorithms for determining whether a given number is prime. When dealing with large numbers, some of these methods are inapplicable while others are time-consuming and inefficient. This paper explores some useful methods for primality testing. It is assumed that the reader has previous experience with mathematical notations and computer programming.

*This research was conducted under the supervision of Dr. Jeffrey Adler for the Summer Research Award from the Mathematics and Statistics Department, College of Arts and Sciences, American University, Washington D.C. E-mail: jadler@american.edu

†Hamman Samuel is a computer science major at the School of Information Technology and Communications, American University of Nigeria. E-mail: hamman.samuel@aaun.edu.ng, hs8818a@student.american.edu

Contents

1 Introduction

1.1 Factorization

Let $\mathbb{N}$ represent the natural numbers. Any number $n \in \mathbb{N}$ can be represented as a product of other numbers, viz:

$$n = \prod_{i=1}^{m} x_i, \qquad x_i \leq n$$

x_i is said to be a *factor* of n.

EXAMPLE The factors of 24 can be 2×12 or 3×8 or 4×6. We also have $2 \times 3 \times 4$ as factors.

1.2 Prime Numbers

Suppose n has only two factors, x_1 and x_2, $x_1 = 1$ and $x_2 = n$ such that x_2 cannot be factored any further. Then n is said to be *prime*. This special case of factorization exists when 1 and n itself are the only possible factors of n. If a number is not prime, then it is stated as a *composite* number.

EXAMPLE 23 is prime since it can only be factored into 1×23. On the other hand, 24 is composite.

1.3 Fundamental Theorem of Arithmetic

The Fundamental Theorem of Arithmetic states that every $n \in \mathbb{N}, n > 1$ can be *uniquely* factored into a product of prime numbers. This theorem establishes that primes are the building blocks of all numbers.

EXAMPLE If we consider the factors of 24 as 2×12, 2 is prime and cannot be factored further. Factoring 12 gives 2×6. Then, factoring 6 gives 2×3. Thus, the unique factorization of 24 is given as $2 \times 2 \times 2 \times 3$. To see that this is unique, we can deduce the same prime factors using 3×8, 4×6, or $2 \times 3 \times 4$. For a general proof, see[1]

1.4 Divisibility

Given $a, b \in \mathbb{Z}$, a is said to divide b if there is some number $c \in \mathbb{Z}$ such that $b = ac$. We denote this relationship as $a \mid b$. More commonly, b is divisible by a, and a is a divisor of b. Clearly, any $n \in \mathbb{N}$ has the property $1 \mid n$.

A prime number $n > 1$ can now be defined as any number whose only divisors are 1 and n.

EXAMPLE The divisors of 23 are 1 and 23 only. Since no other number is a divisor of 23, it is prime.

1.5 Primality Test

Primality denotes the attribute that a number is prime. To determine the primality of n, we determine if n is prime or not.

In order to find out if n is prime, we could divide n by all numbers between 1 and n. If any is a divisor of n, then n is not prime.

Suppose n is composite and $n = ab$. If $a > \sqrt{n}$ **and** $b > \sqrt{n}$, then $ab > n$, which is a contradiction. It is clear that $a, b \leq \sqrt{n}$. So to determine if n is composite, we look for divisors between 1 and $\sqrt{n}$ [1].

It follows that n is prime if it is **not** divisible by any prime p, $1 < p \leq \sqrt{n}$.

EXAMPLE We can determine if 223 is prime by dividing with prime numbers $\leq \sqrt{223}$. If any of these divide 223, then it is not prime. It is not hard to check that 223 is not divisible by any prime up to $\sqrt{223}$. We conclude that 223 is a prime number.

1.6 The Division Theorem

The Division Theorem states that given a and b with $a \neq 0$, there exist unique numbers $q, r \in \mathbb{N}$ such that

$$b = aq + r \qquad 0 \leq r <\mid a \mid$$

If $a \mid b$, then the number q of the theorem satisfes $b = aq$, and $r = 0$. More commonly, a number b is divisible by a if the *remainder* of b divided by a is 0.

EXAMPLE We can write $24 = 2 \times 12 + 0$. Thus, $2 \mid 24$. However, $23 = 2 \times 12 + 1$, so $2 \nmid 23$.

1.7 Modulo Notation

Remainders can also be represented and processed conveniently using modular arithmetic. The remainder, r when b is divided by a can be represented as

$$r = b \bmod a$$

Then, a is a factor of b if $b \bmod a = 0$.

Suppose the numbers $b_1, b_2, ..., b_k$ give the same remainder when divided by a. Then we say these numbers are *congruent*. Notationally, suppose b_1 is congruent $b_2 \pmod a$. We denote this as

$$b_1 \equiv b_2 \pmod a$$

The following observations can be made:

$$b_1 \bmod a = b_2 \bmod a = r$$
$$a \mid (b_1 - b_2)$$

b_2 is said to be a *residue* of $b_1 \pmod a$.

EXAMPLE $23 \bmod 2 = 1$ and $13 \bmod 2 = 1$. Then, we can write $23 \equiv 13 \pmod 2$. Also, suppose $23 \equiv 14 \bmod 9$. Clearly the remainder when 14 is divided by 9 is the same as when 23 is divided by 9.

1.8 Motivation

According to Euclid[1], there are infinitely many primes. Also, according to Kumanduri and Romero[1], a prime number is defined as a negative assertion. Consequenlty, determining whether very large numbers are primes is not trivial. Because of these properties, prime numbers are ideal for cryptography and data security. The RSA scheme is a popular example in which data is encoded using very large primes. Decoding data is then equivalent to factoring these primes. For a detailed study, see [1].

2 Elementary Factoring Methods

The following methods are sufficient for prime factorization of numbers of magnitude 10^6[1].

2.1 Trial Division

The method of trial division is based on the definition of primes in terms of divisibility. Given a list of primes, P, we divide n by $p_i = \{2, 3, 5, ..., \sqrt{n}\}, p_i \in P$. The first divisor, p_k, of n is a factor of n, while the quotient of n divided by p_i is sent back to the algorithm to be factored further. The algorithm terminates if the quotient is prime (or n itself is prime). Since $1 < p \leq \sqrt{n}$, a supply of prime numbers up to $\lfloor\sqrt{n}\rfloor$ is required by the algorithm.

ALGORITHM FactorTrialDiv (n, P)
// Prints the factors of n
// Input: Positive integer n; Array of prime numbers up to $\sqrt{n}$
// Output: Unique factors of n

IF $n < 2$ THEN EXIT

$primeSuspect = true$
FOR $i = 1$ TO SIZE(P)
    IF n mod $P[i] = 0$ THEN
        $primeSuspect = false$
        PRINT $P[i]$
        FactorTrialDiv($n/P[i]$, P)
        EXIT LOOP
    END IF
LOOP

IF $primeSuspect = true$ THEN PRINT n
EXIT

RUN Given $n = 20$, $P = \{2, 3\}$, the first factor printed will be 2 since $2 \mid 20$. The second factor is $20/2 = 10$, but this is sent back to the algorithm as a recursive call to factor further. The next factor printed is 2, since $2 \mid 10$, while $10/2 = 5$ is sent back to the algorithm. Then, 5 is the next factor since $5 \mid 5$, and $5/5 = 1$ is passed back, completing the algorihtm. The algorithm gives the factors of 20 as 2, 2, and 5.

NOTE As its name implies, the boolean variable *primeSuspect* is used to mark all numbers coming into the algorithm as "suspected" prime numbers. Using the reverse-principle of "guilty until proven innocent", these numbers are printed if their status does not change due to divisibility.

2.2 Sieve of Eratosthenes

Trial division requires a list of primes up to a given number. In order to supply such a list, the sieve of Eratosthenes is useful. The sieve for primes up to s is generated by first making an array L of all positive integers from 2 up to s. Then all multiples of numbers $l_i, i = 2, 3, ..., s$ in the list are removed. All the remaining numbers are primes. For ease of computation, we use the subscripts of L to refer to the positive integers. We cross out multiples by setting the corresponding element to 1. On completion, elements of L that are not set to 1 are returned[2].

ALGORITHM SievePrime (s)
// Prints primes up to s
// Input: Positive integer s
// Output: List of primes up to s

SIZE(P) = s
FOR $i = 2$ TO $\sqrt{s}$
    FOR $j = i^2$ TO s STEP i
        IF $P[j] <> 1$ THEN $P[j] = 1$
    LOOP
LOOP

FOR $i = 2$ TO s
    IF $P[i] <> 1$ THEN PRINT $P[i]$
LOOP

EXIT

RUN Given $s = 10$, the sieve begins with the array subscript 2. For each subscript of P that is a multiple of 2, the corresponding element is set to 1. For instance, the elements of subscripts 4, 6, 8 are 1. For the array subscript 3, the element of subscript 6 is already 1, and that of subscript 9 is set to 1. On conclusion, the sieve prints out all the remaining subscripts whose elements have not been set to 1, which are 2, 3, 5, 7

NOTE Optimizations have been used. The outer loop concludes after $\sqrt{s}$ iterations instead of s. Also, elements already set as 1 do not need to be marked again[1]. In the example run, 6 is a multiple of both 2 and 3, but will be marked once. The algorithm prints out a list of primes, and can be modified to store this list for use in the trial division algorithm.

2.3 Fermat's Factoring Method

Suppose a positive number, n can be written in the form of difference of squares using some integers a and b.

$$n = a^2 - b^2$$

If $a - b \neq 1$, the factorization of n is given as:

$$n = (a + b)(a - b)$$

Fermat's method proceeds by choosing some number a such that $a^2 \geq n$. If $a^2 - n$ gives a perfect square, b^2, then the difference of squares can be computed as a factorization of n. Otherwise, the process is repeated by selecting another value for a[1].

A limitation of Fermat's method arises because numbers of the form $4k + 2$ cannot be expressed as a difference of squares, i.e. if $n \bmod 4 = 2$, the algorithm gives no results [1]. This can be handled by dividing numbers of the form $4k+2$ by 2, since $2 \mid 4k+2$. Then, the number 2 is treated as a special case factorization of n.

The algorithm can be made to give unique factorizations by passing back the factors $(a - b)$ and $(a + b)$ to the algorithm recursively.

ALGORITHM FactorFermat (n)
// Prints the factors of n
// Input: Positive integer n
// Output: Unique factors of n

IF $n < 2$ THEN EXIT

IF $n \bmod 4 = 2$ THEN
  PRINT 2
  FactorFermat$(n/2)$
  EXIT
END IF

$primeSuspect = true$
FOR $a = \lceil \sqrt{n} \rceil$ TO $\lceil (n + 1)/2 \rceil$
    $b = \sqrt{a^2 - n}$
    IF $\left\lceil \sqrt{b} \right\rceil = b$ AND $(a - b) \neq 1$ THEN
      $primeSuspect = false$
      FactorFermat$(a - b)$
      FactorFermat$(a + b)$
      EXIT LOOP
    END IF
LOOP

IF $primeSuspect = true$ THEN PRINT n
EXIT

RUN Given $n = 20$, the initial value of a is $\lceil \sqrt{20} \rceil = 5$. The value for b is set to $\sqrt{5^2 - 20} = \sqrt{5}$. Since this is not a perfect square, the next value for a is 6, and the value of b is $\sqrt{6^2 - 20} = \sqrt{16} = 4$. The factors of 20 are $6 - 4 = 2$

and $6 + 4 = 10$, which are passed back to the algorithm. Since 2 is treated as a special case, it is one of the factors of 20.

Since 10 mod 4 = 2, 2 is reported as a factor and $10/2 = 5$ is passed back to the algorithm. 5 is further factored by setting a = $\lceil\sqrt{5}\rceil = 3$. The value of b is $\sqrt{3^2 - 5} = 2$. The algorithm concludes by reporting all unique factors of 20 as 2, 2, and 5.

NOTE Unlike trial division, Fermat's method does not require a list of primes. It can be shown that the loop in Fermat's algorithm needs to iterate up to $(n+1)/2$ instead of n by considering the case when n is prime. Note that when n is prime, no factors are found and the loop iterates to the end. The factors of n are $(a - b) = 1$ and $(a + b) = n$.

$$\begin{aligned} a^2 - b^2 &= n \\ b^2 &= a^2 - n \\ (a-1)^2 &= a^2 - n \\ a^2 - 2a + 1 &= a^2 - n \\ a &= (n+1)/2 \end{aligned}$$

Thus, the maximum number of iterations required are (n + 1)/2.

3 Simple Primality Tests

The following methods are sufficient for determining the primality of 7-digits numbers[1]. For 100-digit numbers, the fastest computers would take 10^t years, where t is a 2-digit number[3]. The algorithms are derived from the factoring methods covered above.

3.1 Trial Division

The trial division algorithm above for determining prime factors can be modified to determine whether a number is prime.

The algorithm checks for divisors of n up to $\sqrt{n}$. If any are found, the algorithm reports that n is not prime. If, after completing iterations up to $\sqrt{n}$, no divisors are found, it is reported that n is prime. The algorithm requires that primes up to $\sqrt{n}$ are known.

ALGORITHM PrimeTrialDiv (n, P)
// Reports whether n is prime or not
// Input: Positive integer n; Array of prime numbers up to $\sqrt{n}$
// Output: Primality of n

```
primeSuspect = true
FOR i = 1 TO SIZE(P)
      IF n mod P[i] = 0 THEN
         primeSuspect = false
         EXIT LOOP
      END IF
LOOP

PRINT primeSuspect
EXIT
```

RUN If $n = 221$, the algorithm requires a supply of primes up to $\sqrt{221}$, that is 2,3,5,7,11,13. 221 is divided by each of the elements in the list of primes. Since $13 \mid 221$, 221 is not prime.

NOTE This test for primality uses divisions to verify that a given number is prime[3]. Since the algorithm is based on the method of trail division for factorization, another approach for determining primality could be to check the factors of a given number:

```
IF FactorTrialDiv(n) = n THEN
   PRINT true
ELSE
   PRINT false
END IF
```

3.2 Sieve of Eratosthenes

The sieve of Eratosthenes above can be adapted for testing primality by marking off multiples and then checking the status of the number in question. This algorithm uses additions instead of divisions[3].

ALGORITHM PrimeSieve (n)
// Determines whether n is prime or not
// Input: Positive integer n
// Output: Primality of n

SIZE(P) = n
FOR $i = 2$ TO $\sqrt{n}$
    FOR $j = i^2$ TO n STEP i
        IF $P[j]$ <> 1 THEN $P[j] = 1$
    LOOP
LOOP

IF $P[n]$ <> 1 THEN
  PRINT *true*
ELSE
  PRINT *false*
END IF

EXIT

RUN Given n = 10, the sieve sets the elements of subscripts 2, 3, 5, and 7 to the value 1. Since the element of subscript 10 is not set to 1, the algorithm concludes by reporting that 10 is not prime.

NOTE An alternative approach would be to generate primes up to n using the sieve and then search for n within the list of primes. However, the algorithm presented here completes relatively faster.

3.3 Fermat's Method

When Fermat's factoring method is used to factor a prime number n, the difference of squares yields $(a + b) = n$. This property can be used to identify primes.

To handle numbers of the form $4k + 2$, which cannot be expressed as a difference of squares, we digress to investigate the primality of even numbers. Beacuse even numbers $2k$ are divisible by 2, then all even numbers are composite. Also, since $2 \mid 4k + 2$, then all numbers of the form $4k + 2$ can be reported as composite.

ALGORITHM PrimeFermat (n)
// Determines whether n is prime or not
// Input: Positive integer n

// Output: Primality of n

IF n mod 4 = 2 THEN
  PRINT *false*
  EXIT
END IF

$primeSuspect = false$
FOR $a = \lceil\sqrt{n}\rceil$ TO $\lceil(n+1)/2\rceil$
    $b = \sqrt{a^2 - n}$
    IF $a + b = n$ THEN
      $primeSuspect = true$
      EXIT LOOP
    END IF
LOOP

PRINT *primeSuspect*
EXIT

RUN Given $n = 23$, the initial value of a is $\lceil\sqrt{20}\rceil = 5$. The value for b is set to $\sqrt{5^2 - 23} = \sqrt{2}$. Since this is not a perfect square, the next value for a is 6, and the value of b is $\sqrt{6^2 - 23} = \sqrt{13}$. The sequence continues till $a = 12$ and $\sqrt{12^2 - 23} = \sqrt{121} = 11$. The factors of 23 are $12 - 11 = 1$ and $12 + 11 = \mathbf{23}$. Thus, 23 is reported as prime.

NOTE It can be observed that deciding whether a number is prime is easier than factorization[4]. Accordingly, the factorization algorithms presented here can be modified to test for primality before attempting factorization. However, factorization algorithms were presented first because of the definition of primes as a negative assertion. That is, numbers that are **not** composite are primes.

4 Review of Modular Arithmetic

Modular arithmetic deals with properties of remainders. Arithmetic notations and operations, as well as theorems are covered in this section.

4.1 Properties of Congruences

Recall that congruences are a convenient way to represent remainders and perform arithmetic operations on them. Given a, b, c and d are integers, the following can be deduced. For proofs, see [1].

4.1.1 Basic Properties

1. $a \equiv a \pmod{m}, \forall a$
2. $a \equiv b \pmod{m} \Leftrightarrow b \equiv a \pmod{m}$
3. $a \equiv b \pmod{m}$ and $b \equiv c \pmod{m} \Rightarrow a \equiv c \pmod{m}$

4.1.2 Arithmetic Properties

1. $a \equiv b \pmod{m} \Rightarrow ac \equiv bc \pmod{m}, \forall c$
2. $a \equiv b \pmod{m}$ and $c \equiv d \pmod{m} \Rightarrow a + c \equiv b + d \pmod{m}$
3. $a \equiv b \pmod{m}$ and $c \equiv d \pmod{m} \Rightarrow ac \equiv bd \pmod{m}$
4. $a \equiv b \pmod{m} \Rightarrow a^k \equiv b^k \pmod{m}, \forall k > 0$

EXAMPLE We determine 16 mod 17 by considering that $16 + 1 \equiv 0 \pmod{17}$. Thus, $16 \equiv (-1 + 0) \pmod{17}$. This can be re-written as $2^4 \equiv -1 \pmod{13}$. Also, $(2^4)^2 \equiv (-1)^2 \pmod{13}$. Thus, $2^8 \equiv 1 \pmod{17}$.

We can find the remainder when 2^{16} is divided by 17 using the preceding result. Note that $16 = 2 \times 8$, thus $2^{16} \pmod{17} \equiv (2^8)^2 \pmod{17} \equiv (-1)^2 \pmod{17} \equiv 1 \pmod{17}$. Thus, we have computed the remainder as 1 without actually computing 2^{16} and dividing by 17.

4.2 Fermat's Theorem

Fermat's theorem states the following. Note that statement 2 is also called Fermat's Little Theorem[4] (see [1] for proofs).

1. If p is a prime and $a \in \mathbb{Z}$ then $a^p \equiv a \pmod{p}$.
2. Also, given p is prime, if $p \nmid a$ then we have $a^{p-1} \equiv 1 \pmod{p}$.

EXAMPLE By Fermat's theorem, we can deduce the remainder when 2^{16} is divided by 17. Since $17 \nmid 2$, then $2^{17-1} \equiv 1 \pmod{17}$. Thus, $2^{16} \equiv 1 \pmod{17}$.

4.3 Euler's ϕ-Function

Before presenting Euler's ϕ-Function, the following definitions need to be stated.

4.3.1 Greatest Common Divisor (GCD)

Let a and b be two numbers. The largest integer g, such that $g \mid a$ and $g \mid b$ is referred to as the greatest common divisor of a and b, where g is denoted as (a, b).

EXAMPLE (12,9) = 3 because 3 is the largest number that divides both 12 and 9.

The following algorithm, called Euclid's algorithm, is useful for finding (a, b).

ALGORITHM EuclidGCD(a, b)
// Returns the greatest common divisor of a and b
// Input: Numbers a, b
// Output: Greatest common divisor of a and b

IF $b = 0$ THEN
  PRINT a
ELSE
  $r = a \bmod b$
  $a = b$
  $b = r$
  EuclidGCD(a, b)
END IF

EXIT

RUN Given a = 12 and b = 9, since b $\neq$ 0, the algorithm finds the remainder of $a/b = 3$. The recursive call sets a = 9 and b = 3. The next iteration sets a = 3 and b = 0, and concludes by printing out 3 as the GCD of 12 and 9.

4.3.2 Relatively Prime Numbers

Two numbers a and b are relatively prime or coprime if $(a, b) = 1$. That is, if the GCD of a and b is 1, then a and b are relatively prime. An additional property of congruences can be stated, given $(m, n) = 1$.
$a \equiv b \pmod{m}$ and $a \equiv b \pmod{n} \Leftrightarrow a \equiv b \pmod{mn}$

EXAMPLE Since (25, 13) = 1, 25 and 13 are relatively prime. More generally, given two numbers a and b, if the algorithm EuclidGCD(a, b) returns 1, this concludes that a and b are relatively prime.

4.3.3 Statement of Euler's ϕ-Function

Euler's phi-function, also called Euler's Totient function, $\phi(m)$ gives the number of elements in $S = \{0, 1, ..., m - 1\}$, such that $(a, m) = 1$, where $a \in S$.

EXAMPLE $\phi(10)$ is determined by counting those elements in 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 that are relatively prime to 10. These are 1, 3, 7, and 9. Thus, $\phi(10) = 4$.

Note that $\phi(p) = p - 1$ if p is a prime. This can be shown by considering that given numbers $0, 1, ..., p - 1$, the only number that is not relatively prime to p is 0, since 0 is a multiple of p, $(0, p) = p$.

Generally, for p as prime, $\phi(p^k) = p^k - p^{k-1}$, which lends to the previous statement when $k = 1$, as $\phi(p^1) = p^1 - p^0 = p - 1$. This can be shown by considering that the numbers that are not coprime to p^k are multiples q of p, since $q \mid p \rightarrow (q, p^k) = p$. Given p^k, there are p^{k-1} multiples of p in the range $0, 1, 2, ..., p^k - 1$.

A useful property of the ϕ function is that if $(a, b) = 1$ then $\phi(ab) = \phi(a)\phi(b)$. The following algorithm can be used to compute $\phi(m)$.

ALGORITHM EulerPhi(m)
// Returns the value of $\phi(m)$
// Input: Number m
// Output: $\phi(m)$

count = 0
FOR $a = 0$ TO $m - 1$
  IF EuclidGCD(a, m) = 1 THEN
    count = *count* + 1
  END IF
LOOP

RETURN *count*
EXIT

RUN To compute $\phi(10)$, the algorithm begins by setting the counter variable to 0. If any of the elements 0,1, ..., 9 are relatively prime to 10, the counter is incremented. Since 1, 3, 7, and 9 are relatively prime to 10, the algorithm returns the value of counter as 4.

The algorithm can be optimized for handling large numbers by considering that the value of $\phi(m)$ can be trivially calculated if m is prime or a power of a prime. Also, the algorithm can handle composite numbers by factorizing m and using the Totient function on these factors.

For example, suppose m is prime. Then, $\phi(m^k) = m^k - m^{k-1}$. Also, suppose m is composite, such that $m = m_1 \times m_2$, where $(m_1, m_2) = 1$. Then, $\phi(m) = \phi(m_1 m_2) = \phi(m_1)\phi(am_2)$, where m_1 and m_2 are smaller than m, and can in turn be factored further.

EXAMPLE The value of $\phi(1000)$ can be calculated by use of the above optimizations. We can factor 1000 into 8 and 125, and $(8, 125) = 1$. Thus, $\phi(1000) = \phi(8 \times 125) = \phi(8)\phi(125)$. Also, we note that $125 = 5^3$, and $\phi(8) = 4$.

Thus, $\phi(1000) = 4 \times \phi(5^3)$. Since 5 is prime, we compute $\phi(5^3) = 5^3 - 5^2 = 125 - 25 = 100$. In conclusion, $\phi(1000) = 4 \times 100 = 400$.

4.4 Euler's Theorem

Euler's theorem generalizes Fermat's theorem presented above[4]. Whereas Fermat's theorem deals with primes, Euler's theorem applies to composite numbers. It states that if a and m are integers such that $(a, m) = 1$, then $a^{\phi(m)} \equiv 1 \pmod{m}$. For a proof, see [1].

EXAMPLE Suppose we wish to find the last digit of 2009^{2009}. That is, we solve $2009^{2009} \pmod{10}$. Note that 2009 and 10 are relatively prime, and $\phi(100) = 4$. Thus, we use Euler's theorem to state that $2009^4 = 1 \pmod{10}$. We can manipulate the original problem, viz $2009^{2009} \equiv 2009^{4\times502+1} \equiv (2009^4)^{502}2009^1 \equiv 1^{502}2009^1 \equiv 2009^1 \equiv 9 \pmod{10}$. Thus, the last digit is 9.

5 Probabilistic Primality Tests

Probabilistic methods test primality in terms of the likelihood of a given number being prime. If a number passes these tests, it is likely that the number is prime. Probabilistic methods classify a number as a probable primes, but do not prove the number is prime.

Some numbers "masquerade" as prime numbers when in fact they are not. As an example, Fermat's theorem would declare 341 as prime since $2^{341} \equiv 2 \pmod{341}$, but $341 = 11 \times 31$. Also, 561 could be "prime" since $2^{560} \equiv 1 \bmod (561)$, but $561 = 3 \times 11 \times 17$.

5.1 Pseudoprimes

A composite number n such that $a^n \equiv a \pmod{n}$ is called a pseudoprime to base a, or a-pseudoprime. As stated above, 341 is a 2-pseudoprime.

In general, Fermat's theorem states that given p as prime, we have $a^p \equiv a \pmod{p}$. The converse of this statement would be to assume that if $a^p \equiv a \pmod{p}$, then p is prime. However, this is not always the case.

EXAMPLE It can be shown that 91 is a 3-pseudoprime. Note that $3^6 = 1 \pmod{7}$ and also $3^3 = 1 \pmod{13}$. Thus, we have:

$3^{91} \equiv 3^{90}3^1 \pmod{7} \equiv (3^6)^{15}3^1 \equiv 3 \pmod{7}$, and
$3^{91} \equiv 3^{90}3^1 \pmod{13} \equiv (3^3)^{30}3^1 \equiv 3 \pmod{13}$
So $3^{91} \equiv 3 \pmod{7 \times 13} \equiv 3 \pmod{91}$.

Thus, 91 is classified as a 3-pseudoprime. It can be verified that 91 is actually a composite number since $91 = 7 \times 13$.

5.2 Carmichael Numbers

A composite number n such that $a^{n-1} \equiv 1 \pmod{n}$ and $(a, n) = 1$, is called a Carmichael number. Note that Fermat's Little Theorem can be used to qualify a Carmichael number n as prime, even though this may not be true. Recall that if $p \nmid a$ and p is prime, then $a^{p-1} \equiv 1 \pmod{p}$. However, given $a^{p-1} \equiv 1 \pmod{p}$ we cannot conclude that p is prime.

A Carmichael number can also be defined in terms of divisibility. n is a Carmichael number if and only if for every $p \mid n$, we have that $p - 1 \mid n - 1$[1].

EXAMPLE 561 can be shown as Carmichael by selecting $p = 11$. Since $11 \mid 561$ and $10 \mid 560$, 561 is a Carmichael number.

5.3 Rabin-Miller Probabilistic Primality Test

It should be noted that primes also pass the requirements for pseudoprimes and Carmichael numbers, as do composite numbers. However, an additional property can be used to identify strong pseudoprimes, numbers that are more

likely prime. Suppose $n-1=2^r s$, then n is said be a strong pseudoprime to base a if:

1. $a^s \equiv 1 \pmod{n}$, or
2. $a^{s2^i} \equiv -1 \pmod{n}$ for some $0 \leq i < r$

The algorithm below tests for strong pseudoprimes. To determine if n is a strong pseudoprime, we first state that $n-1=2^r s$. Next, we compute the sequence $x_0, x_1, ..., x_r$, such that:

$$x_0 = a^s \bmod n,\ x_1 = a^{2s} \bmod n,\ x_2 = a^{4s} \bmod n,\ ...,\ x_r = a^{2^r s} \bmod n$$

where a is some arbitrary base. Finally, if at any point in generating the above sequence one of the properties of a strong pseudoprime is satisfied, it is more likely that n is prime [1].

ALGORITHM StrongPPrime (n, a)
// Outputs whether n is a probable prime
// Input: Number n, Arbitrary base a
// Output: *true* - Probable prime, *false* - Composite

$s = n-1$
$r = 0$
$c = 0$
WHILE ($s \bmod 2 = 0$)
  $s = s/2$
  $r = r+1$
LOOP

$a = a^s \bmod n$

IF $a = 1$ THEN RETURN *true*

$s = 2$
WHILE (*true*)
  IF $a = -1$ THEN
    RETURN *true*
  ELSE IF $a = 1$ OR $c > r-2$ THEN
    RETURN *false*
  END IF

  $a = a^s \bmod n$
  $c = c+1$
LOOP
EXIT

RUN We can verify whether 341 is a strong pseudoprime to base 2. The

algorithm begins by setting $s = 340$, $r = 0$, $c = 0$. 340 is continuously divided by 2 until $s = 85$ and $r = 2$, where $340 = 2^2 85$. The base is used to compute $a = 2^{85} \bmod 341 = 32$. Since a does not compute to 1, the algorithm further calculates the value of a using $s = 2$ and increments c, so that $a = a^2 \bmod 341 = 1$ and $c = 1$. Since a is computed as 1, the algorithm terminates and reports 341 as composite.

NOTE The algorithm can be modified to try out different values for a each time it reports a number as a pseudoprime. This increases the likelihood that n is not just a pseudoprime, but an actual prime[1].

6 Factoring Large Numbers

This section presents methods of factoring that are suitable for large numbers which do not yield to the elementary factoring methods due to the execution time of the algorithms.

6.1 Pollard's $p-1$ Method

Pollard's $p-1$ method does not work for all numbers, but is still useful for finding factors of some very large numbers. To factor n, the method determines the value of $a^{k!}-1 \equiv b \pmod{n}$ and then $(b-1,n)$, where $k = 1, 2, ...$ After successive trials, if $(b-1,n)$ has some value other than 1, then a nontrivial factor of n is $(b-1,n)$. For a proof, see [1].

ALGORITHM pMethod (n)
// Possibly outputs a nontrivial factor of n
// Input: Number n
// Output: Nontrivial factor of n

$a = 1$
$i = 1$
$t = 10$ // Tolerance level
WHILE ($i < t$ AND $a = 1$)
  a = EuclidGCD($(2^{i!} \bmod n) - 1$, n)
  $i = i + 1$
LOOP

PRINT a
EXIT

RUN It is again emphasized that the algorithm does not guarantee a result for all numbers. To find a factor of 1073, we compute the following sequence:

$$\begin{array}{lcrlcrcr}
2^{1!} & \equiv & 2 & \pmod{1073} & , & (2-1,341) & = & 1 \\
2^{2!} & \equiv & 4 & \pmod{1073} & , & (4-1,341) & = & 1 \\
2^{3!} & \equiv & 64 & \pmod{1073} & , & (64-1,341) & = & 1 \\
2^{4!} & \equiv & 861 & \pmod{1073} & , & (861-1,341) & = & 1 \\
2^{5!} & \equiv & 285 & \pmod{1073} & , & (285-1,341) & = & 1 \\
2^{6!} & \equiv & 371 & \pmod{1073} & , & (371-1,341) & = & 37
\end{array}$$

Thus, a nontrivial factor of 1073 is 37.

6.2 Pollard's ρ-Method

The theoretical concepts of Pollard's ρ-method are beyond the scope of this paper. However, the method itself can be used mechanically as follows. To find

a factor of n we construct a sequence of integers $x_1, x_2, ...$ where $x_{k+1} = f(x_k)$ mod n and $f(x_k)$ is a polynomial with integer constants, such as $f(x) = x^2 + 1$. Next, we construct the sequence $y_k = f(f(y_{k-1}))$ mod n, and set $x_0 = y_0 = \alpha$, where α denotes some initial value. If $(y_k - x_k, n) \neq 1$, then $(y_k - x_k, n)$ is a nontrivial factor of n [1].

ALGORITHM rhoMethod (n)
// Outputs a nontrivial factor of n, using $f(x) = x^2 + 1$
// Input: Number n
// Output: Nontrivial factor of n

$x = 2$
$y = 2$
$a = 1$
$i = 0$
$t = 10$ // Tolerance level
WHILE ($i < t$ AND $a = 1$)
  $x = f(x)$ mod n
  $y = f(f(y))$ mod n
  $a =$ EuclidGCD$(y - x,\ n)$
  $i = i + 1$
LOOP

PRINT a
EXIT

RUN To find a factor of 1073, we take $f(x) = x^2 + 1$ and construct the sequences $x_{k+1} = x_k^2 + 1$ and $y_k = f(f(y_{k-1}))$ as follows [1]:

k	x_k	y_k	$y_k - x_k$	$(y_k - x_k, n)$
0	2	2	0	-
1	5	26	21	1
2	26	159	133	1
3	677	936	259	37

Thus, a nontrivial factor of 1073 is gotten in three steps as 37.

7 Proofs of Primality

Given a large number, determining whether it is prime is not a trivial task. Simple methods such as trial division are not computationally feasible while probabilistic methods still require rigorous computations. This section presents methods that provide short and simple proofs of primality.

7.1 Lucas-Lehmer Test

Suppose there exists a number a, such that $a^{n-1} \equiv 1 \pmod{n}$. Also suppose p is a prime such that $p \mid n-1$ and $a^{n-1/p} \equiv b \pmod{n}$. If $b \neq 1$, then n is prime.

ALGORITHM LucasLehmer (n, a)
// Outputs whether n is prime or not
// Input: Number n, Base a
// Output: Primality of n

p = FactorTrialDiv($n-1$)
IF a^{n-1} mod $n \neq 1$ THEN RETURN *false*
FOR $i = 1$ TO SIZE(p)
  IF $a^{n-1/p[i]}$ mod $n = 1$ THEN RETURN *false*
LOOP

RETURN *true*
EXIT

RUN We test whether 1321 is prime by taking $a = 2$. Since $2^{1321-1} \equiv 1 \pmod{1321}$, we can proceed. We next check that for each prime factor of $n - 1 = 1320$, the condition $2^{1320/p}$ mod 1321 always gives a value that is not 1. Values for p can be gotten via trial division as 2,3,5 and 11. Note that only unique values are selected, as $1320 = 2 \times 2 \times 2 \times 3 \times 5 \times 11$.

$$\begin{aligned} 2^{1320/2} &\equiv -1 \pmod{1321} \\ 2^{1320/3} &\equiv 1023 \pmod{1321} \\ 2^{1320/5} &\equiv 516 \pmod{1321} \\ 2^{1320/11} &\equiv -1 \pmod{1321} \end{aligned}$$

Since all the requirements for the Lucas-Lehmer test are satisfied, 1321 can be declared as a prime.

NOTE This test does not work if the condition $a^{n-1} \equiv 1 \pmod{n}$ is not satisfied. In the case this condition fails, a new arbitrary value for a can be tried out [1].

7.2 Primality Certificates

Using the Lucas-Lehmer test, a proof of primality of n involves computation of the factors of $n-1$. Note that an additional condition stated in the test is that

p needs to be a prime number. In other words, in order to determine whether n is prime, we also need to verify that the factors of $n-1$ are primes.

A certificate of primality for n consists of the special number 2 and all factors of $n-1$ and can be used to verify whether n is prime [3]. Using factors $p_i \mid (n-1)$, a certificate for n would be of the form:

$$(2 : p_1, p_2, ...)$$

Thus, proving n is a prime can be done in two steps [1]:

1. Generate a certificate for n
2. Verify the certificate of n

Generating the certificate involves use of the Lucas-Lehmer method to produce $p_1, p_2, ...$, while verification is to prove that each number $p_1, p_2, ...$ in the certificate is a prime. In verification, to show that p_i is prime, a certificate for p_i could be generated, and so on. This recursive approach allows presentation of a proof of primality of any number fairly quickly because computation reduces to the fact that 2 is prime [3].

ALGORITHM CertPrime(n)
// Outputs a certificate of n
// Input: Number n
// Output: Primality certificate of n

RETURN FactorTrialDiv($n-1$)
EXIT

ALGORITHM CertVerify(n, a)
// Verifies a certificate of n
// Input: Number n, Base a
// Output: Primality of n

$nCert$ = CertPrime(n)
primeSuspect = *false*
IF a^{n-1} mod $n \neq 1$ THEN RETURN *false*
FOR $i = 1$ TO SIZE($nCert$)
  IF $a^{n-1/nCert[i]}$ mod $n = 1$ THEN RETURN *false*
  IF $nCert[i] = 2$ THEN
    primeSuspect = *true*
  ELSE
    primeSuspect = CertVerify($nCert[i]$, a) // Get certificates of certificate
  END IF
LOOP
RETURN *primeSuspect*
EXIT

RUN A certificate for 1321 would be (2:2,3,5,11). For each number, we generate a certificate and verify that each is prime. To prove 11 is prime, its

certificate is generated as (2:2,5). For 5, we generate the certificate (2:2), for 3 we have (2:2). Since all the certificates have been reduced to the condition that 2 is prime, we can conclude that 1321 is prime.

1321	(2:2,3,5,11)
11	(2:2,5)
5	(2:2)
3	(2:2)

7.3 Pocklington-Lehmer Test

The factorization of $n-1$ becomes time-consuming for large numbers. The Pocklington-Lehmer test is an alternative approach, in which only a partial factorization of $n-1$ is needed [1].

Suppose $n-1 = st, s > t$, where all the prime factors of a are known and $(s,t) = 1$. Also suppose there exists a number a, such that $a^{n-1} \equiv 1 \pmod{n}$. If q is a prime factor of s, $q \mid s$ and $(a^{n-1/q} - 1, n) = 1$, then n is a prime [1].

ALGORITHM PocklingtonLehmer (s, t, a)
// Outputs whether n is prime or not, where $n = st + 1$
// Input: Numbers s and t where $s > t$, Base a
// Output: Primality of n

$n = (s \times t) + 1$
q = FactorTrialDiv(s)
IF a^{n-1} mod $n \neq 1$ THEN RETURN *false*
FOR $i = 1$ TO SIZE(q)
  IF EuclidGCD($a^{n-1/q[i]}$, n) $\neq 1$ THEN RETURN *false*
LOOP

RETURN *true*
EXIT

RUN To find whether 1321 is prime, we may begin by finding a nontrivial factor of $1321-1 = 120 \times 11$. Note that $(120, 11) = 1$ and $2^{1320} \equiv 1 \pmod{1321}$. We then find factors of 120 as 8, 3, 5. Since $(2^{1320/2}, 1321) = 1$, $(2^{1320/3}, 1321) = 1$, and also $(2^{1320/5}, 1321) = 1$, we conclude that 1321 is prime.

NOTE Nontrivial factors for $n-1$ can be determined by using trial division or Pollard's ρ-method, leading to s and $t = (n-1)/s$. The larger of s or t can then be determined and fed to the above algorithm appropriately.

Summary

A quick Internet search reveals that as of 2008, the largest prime is $2^{43112609} - 1$, having about 13 million digits. In determining primality, methods such as trial division can be used for small numbers. For humongous numbers like the largest prime, simpler methods are computationally tedious.

Probabilistic methods exist that give a hint of the likelihood that a number is prime, but are still not definitive. Ideally, we would like to succinctly prove or disprove the primality of considerably large numbers. Primality certificates provide this relative speed and ease for proving the primality of very large numbers. Indeed, just as a factor of any number constitutes a quick proof that it is composite, there are algorithms to give a quick proof that a number is prime.

References

[1] R. Kumanduri and C. Romero. *Number Theory with Computer Applications*. 1998. New Jersey: Prentice Hall.

[2] H. M. Deitel and P. J. Deitel. *C How to Program, 3rd ed.* 2001. New Jersey: Prentice Hall.

[3] J. Higgins, D. Campbell, "Mathematical Certificates", *The Mathematics Magazine* (1), **67** (1994): 21-28, [Online–Subscription only], Available: <http://www.jstor.org/stable/2690549 >, JSTOR, All Mathematics Journals, 15 Aug. 2008.

[4] R. A. Mollin, "A Brief History of Factoring and Primality Testing B. C.(Before Computers)", *The Mathematics Magazine* (1), **75** (2002): 18-29, [Online–Subscription only], Available: <http://www.jstor.org/stable/3219180>, JSTOR, All Mathematics Journals, 15 Aug. 2008.