# Primes and How to Recognize them

Satwant Rana
2012 MT 50618
Advised by, Prof. Amitabha Tripathi

## 1 Motivation

Prime numbers are central to Number Theory, acting as the atomic units around which all numbers are built. Therefore it is barely a surprise that *Primality Testing* is a problem with a rich history in Number Theory.

The motivation behind this project is to rediscover and implement the greatest and the latest in primality testing algorithms.

## 2 Primes

Primes are defined as natural numbers which are only divisible by 1 and themselves. Formally, given $p \in \mathbb{N}$ is a prime, if whenever $q \mid p$, then $q \in \{1, p\}$.

Any natural greater than 1 which is not a prime is called a *composite*.

## 3 A Naive Primality Test

The definition of primes presented above can be used to create the simplest primality testing algorithm.
Given a natural $n$, whose primality has to determined, we start with 2 and check for every number upto $n - 1$, whether it divides $n$ or not. If we find a divisor in this range, then the above definition tells us that $n$ is not a prime, otherwise it is.

We present the above discussion in pseudocode form in Algorithm 1.

The while loop runs over $O(n)$ values, making the runtime of the algorithm linear in $n$. Inside a computer numbers are stored in base 2 as sequence of bits, and the size of a number is thus determined by its number of bits which is $O(\log n)$. Therefore the above algorithm is exponential in the size of $n$.

**Algorithm 1** Naive Primality Test

```
procedure NaivePrimalityTest($n$)
    $d \leftarrow 2$
    while $d \leq n - 1$ do
        $r \leftarrow n \mod d$
        if $r = 0$ then
            return false                              ▷ $n$ is composite
        $d \leftarrow d + 1$
    return true                                       ▷ $n$ is prime
```

## 4 An Optimization

The definition of division in $\mathbb{Z}$ dictates that for $a, n \in \mathbb{Z}$, if $a \mid n$, then there exists $b \in \mathbb{Z}$ such that $n = ab$. Thus, if $a \geq \sqrt{n}$, then $b \leq \sqrt{n}$.

This reveals that a check for divisors of $n$ upto $\sqrt{n}$ is sufficient as other divisors would be paired up with one of these divisors. We can optimize the first algorithm we presented to $O(\sqrt{n})$ with this optimization, as seen in Algorithm 2.

**Algorithm 2** Optimized Naive Primality Test

```
procedure OptimizedNaivePrimalityTest($n$)
    $d \leftarrow 2$
    while $d \leq \min(n - 1, \sqrt{n})$ do
        $r \leftarrow n \mod d$
        if $r = 0$ then
            return false                              ▷ $n$ is composite
        $d \leftarrow d + 1$
    return true                                       ▷ $n$ is prime
```

## 5 Compositeness Tests

So far we have seen tests which check whether a given number is prime or not. But there exists tests which run much faster than the primality tests discussed above, but on some rare occasions indicate composites as primes. These composite numbers which the compositeness test labels as primes are called the *pseudoprimes* for the test.

More formally, a successful *primality test* proves that a given number is prime, whereas a successful *compositeness test* proves that a given number is composite.

If a compositeness test is not successful, then we can't comment on the primality of the given number. So how do we use a compositeness test to test primality? We will see in coming sections.

Before we proceed to present some compositeness tests, we need some number theoretic background.

## 6 Fermat's (Little) Theorem

An important and beautiful result in Number Theory, commonly referred to as *Fermat's Little Theorem*, is as presented as Theorem 6.1.

**Theorem 6.1** (Fermat's Theorem). *Given prime $p$, and $a \in \mathbb{Z}$, $(a, p) = 1$ we have,*

$$a^{p-1} \equiv 1 \mod p$$

*Proof.* If $p \mid a$, then $a^p \equiv a \equiv 0 \mod p$, and we are done. Otherwise if $(a, p) = 1$, then list all the first $p - 1$ multiples of $a$,

$$a, 2a, 3a, \ldots (p-1)a$$

All of these numbers are unique $\mod p$, for if $ia \equiv ja \mod p$ then since $(a, p) = 1$, we can cancel $a$ from both sides giving $i \equiv j \mod p$. So, the above numbers represent the following sequence in some order,

$$1, 2, 3, \ldots (p-1)$$

We can equate the products of the sequences to get,

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

Again since $((p-1)!, p) = 1$, we can cancel $(p-1)!$ to get

$$a^{p-1} \equiv 1 \mod p$$

which completes the proof. □

**Corollary 6.1.1.** *Given prime $p$, and $a \in \mathbb{Z}$ we have,*

$$a^p \equiv a \mod p$$

*Proof.* If $(a, p) = 1$, then we have $a^{p-1} \equiv 1 \mod p$ by Theorem 6.1. We can multiply on both sides by $a$ to get $a^p \equiv a \mod p$.

If $p \mid a$, then $a^p \equiv 0 \equiv a \mod p$. □

**Corollary 6.1.2.** *If $n \in \mathbb{N}$, $n \geq 2$ and $\exists a \in \mathbb{Z}$ such that,*

$$a^n \not\equiv a \mod n$$

*then $n$ is not a prime.*

Corollary 6.1.2 states the converse of Corollary 6.1.1.

## 7 Fermat's Theorem as a Compositeness Test

The Corollary 6.1.2 is a simple compositeness test using *Fermat's Theorem*. For instance, for $n = 9$, $2^9 \equiv 8 \not\equiv 2 \mod 9$, indicating the compositeness of 9.

However, there do exist combinations of $a$ and composite $n$ which satisfy the *Fermat's Theorem*. For instance $n = 341 = 11.31$ gives $2^{341} \equiv 2 \mod 341$. This makes 341 a pseudoprime to the Fermat's Compositeness Test, or a *Fermat Pseudoprime*. Although, in this case a change of base $a$ from 2 to 3 yields $3^{341} \equiv 168 \not\equiv 3 \mod 341$ which indicates that 341 is not a prime.

We arrive at this fallacy because the condition in Corollary 6.1.2 is only a sufficient condition for $n$ to be a composite, but not a necessary one. And that is what makes it a compositeness test and not a primality test.

As a matter of fact, there do exist composites $n$ which satisfy Corollary 6.1.2 for all $a$ coprime to $n$, and are called *Carmichael Numbers*. Formally defined, $n$ is a Carmichael Number if $a^{n-1} \equiv 1 \mod n, \forall a \in \mathbb{Z}, (a, n) = 1$. The smallest example of Carmichael Numbers is 561, and there exist infinitely many of them.

If we add an additional check of whether $(a, n) \neq 1$ in the beginning of the test, we can correctly label Carmichael Numbers as composites given a suitable base $a$ (which is not coprime to $n$).

Let's have some algorithmic buildup before we can present the pseudocode of Fermat's Compositeness Test.

## 8 Eucledian Algorithm for G.C.D.

In this section we present the well-known euclidean algorithm to calculate the greatest common divisor of two (non-negative) ingtegers. Since gcd is defined to be positive, the negative sign can be eliminated by taking absolute values at beginning.

We trivially have $(a, 0) = a$, $(a, b) = (a, b - a)$ and therefore $(a, b) = (a, b \mod a)$.

This set of identities can be leveraged to formulate a systematic way to calculate $(a, b)$, $\forall a, b \in \mathbb{Z}$. If we $b \geq a$, then it can be replaced by $b \mod a$ without affecting the gcd value. Since $b \mod a < a$, we have that the sequence of the minimum of the two numbers is a strictly decreasing one. This assures that the minimum value will finally become 0, and the algorithm will terminate.

Let's look at the pseudocode in Algorithm 3.

**Algorithm 3** Euclidean Algorithm

```
procedure EuclideanAlgorithm($a, b$)
    $a \leftarrow$ Abs($a$)
    $b \leftarrow$ Abs($b$)                    ▷ Eliminating negative signs
    if $a > b$ then
        Swap($a, b$)
    while $a \neq 0$ do
        $c \leftarrow b \mod a$
        $b \leftarrow a$
        $a \leftarrow c$
    return $b$
```

Let's define $a_n, b_n$ to be two numbers at the $i^{th}$ iteration, such that $0 \leq a_n \leq b_n$. We have $a_{n+1} = b_n \mod a_n$, $b_{n+1} = a_n$ and $a_{n+2} = a_n \mod (b_n \mod a_n)$, $b_{n+2} = b_n \mod a_n$. Thus either $a_{n+1} \leq \frac{a_n}{2}$, or $a_{n+2} = a_n - a_{n+1} \leq \frac{a_n}{2}$.

Therefore, the minimum of the two numbers gets halved in atmost two steps, making the algorithm take $O(\log \min(|a|, |b|))$ remainder calculation steps.

## 9 Logarithmic Exponentiation

An essential part of the Fermat's Compositeness Test is to be able to calculate the value $a^n$. We can calculate it by repeatedly multiplying $a$ to a running product $n$ times and calculating the remainder modulo $n$. This is an $O(n)$ algorithm to calculate $a^n$, thus making Fermat's Composteness Test $\Omega(n)$ in time - not an improvement over primality tests discussed so far.

Fortunately, there's structure to the problem in the form of the following recursion

$$a^n = \begin{cases} 1 & n = 0 \\ (a^{\frac{n}{2}})^2 & n \equiv 0 \mod 2 \\ a(a^{\frac{n-1}{2}})^2 & n \equiv 1 \mod 2 \end{cases}$$

The above can be rewritten in the form of Algorithm 4.

**Algorithm 4** Recursive Logarithmic Exponentiation

```
procedure RecursiveLogarithmicExponentiation($a, n, m$)
    $result \leftarrow 1 \mod m$                                  ▷ Calculates $a^n \mod m$
    if $n > 0$ & $n \equiv 0 \mod 2$ then
        $result \leftarrow$ RecursiveLogarithmicExponentiation($a, \frac{n}{2}, m$)
        $result \leftarrow result * result \mod m$                ▷ Watch for overflow here
    else if $n > 0$ & $n \equiv 1 \mod 2$ then
        $result \leftarrow$ RecursiveLogarithmicExponentiation($a, \frac{n-1}{2}, m$)
        $result \leftarrow result * result \mod m$
        $result \leftarrow result * a \mod m$
    return $result$
```

At every step of recursion, the value of $n$ gets halved thus the algorithm takes $O(\log n)$ multiplication steps.

We can improve on Algorithm 4, by eliminating the recursion. Computers store numbers in binary. Suppose $n = b_{d-1}b_{d-2}\dots b_0 = \sum_{i=0}^{d-1} b_i 2^i$, where $d$ is the number of bits of $n$ and $b_i$ is the $i^{th}$ bit from right. So $a^n$ can be calculated as

$$a^n = a^{\sum_{i=0}^{d-1} b_i 2^i} = \prod_{i=0}^{d-1} a^{b_i 2^i}$$

The above equation can be represented as Algorithm 5.

Thus we can write Fermat's Compositeness Test as Algorithm 6.

It's easy to see that the algorithm takes $O(\log n)$ time.

## 10 Fermat's Theorem as a Probabilistic Primality Test

Once we have a compositeness test, we can run a it for a fixed number of times on the same number (with changing parameters). If we don't get indication of compositeness even a single time, then the probability of the given number being a composite is too low, and we have a *Probabilistic Primality*

**Algorithm 5** Iterative Logarithmic Exponentiation

```
procedure IterativeLogarithmicExponentiation($a, n, m$)
    $result \leftarrow 1 \mod m$                         ▷ Calculates $a^n \mod m$
    $b \leftarrow a$
    while $n > 0$ do
        if $n \mod 2 = 1$ then                         ▷ If rightmost bit is 1
            $result \leftarrow result * b \mod m$              ▷ Multiply by $b$
        $b \leftarrow b * b \mod m$                     ▷ $b$ stores $a^{2^i}$ on $i^{th}$ step
        $n \leftarrow \frac{n}{2}$                          ▷ Remove rightmost bit
    return result
```

**Algorithm 6** Fermat's Compositeness Test

```
procedure FermatCompositenessTest($a, n$)
    $gcd \leftarrow$ EuclediAnAlgorithm($a, n$).
    if $gcd > 1$ & $gcd < n$ then
        return $false$
    $left \leftarrow$ IterativeLogarithmicExponentiation($a, n, n$)
    $right \leftarrow a \mod m$
    return $left \neq right$
```

*Test.* A probabilistic primality test thus indicates whether a number is composite or a *probable prime*, i.e. a number which is prime with some (high) probability.

This technique can be applied to Fermat's Compositeness Test, by changing the base $a$ on successive iterations, as seen in Algorithm 7.

A small value of $iter = 20$ works well in practice, and thus we have an $O(\log n)$ algorithm.

## 11 Fermat's theorem as a Primality Test

When trying to convert a compositeness test into a primality test, the first obvious solution that comes to mind is to (somehow) maintain a table of all possible pseudoprimes within the range we wish to work on. Before running the actual test, one can lookup the table to see if the input number is a pseudoprime for the test, and label it as a composite right at the beginning of the test.

The pseudoprimes for Fermat's Test also depend upon the base $a$ other than input $n$. D.H. Lehmer prepared a table of all Fermat pseudoprimes

**Algorithm 7** Fermat's Probabilistic Primality Test

```
procedure FermatProbabilisticPrimalityTest($n, iter$)
    while $iter > 0$ do                                   ▷ $iter$ is number of iterations
        $a \leftarrow$ Random($0, n-1$)            ▷ Random number in the range $[0, n-1]$
        $check \leftarrow$ FermatCompositenessTest($a, n$)
        if check then
            return $false$                                        ▷ Composite found
        $iter \leftarrow iter - 1$
    return $true$                                             ▷ Probable prime found
```

below $2.10^8$ for the base 2 with no factor $< 317$. Thus a primality test to check primality for $n < 2.10^8$ can be formulated as Algorithm 8.

**Algorithm 8** Fermat's Primality Test

```
procedure FermatPrimalityTest($n$)
    if $n \geq 2.10^8$ then
        return $false$                                        ▷ Fail if out of range
    for $i = 2$, $i \leq \min(313, n-1)$, $i \leftarrow i + 1$ do
        if $i \mid n$ then
            return $false$                                            ▷ Factor $\leq 313$
    if IterativeLogarithmicExponentiation($2, p-1, p$) $\not\equiv 1 \mod 2$
then
        return $false$                              ▷ Composite by Fermat's Theorem
    return !IsLehmerPseudoprime($n$)                          ▷ Check Lehmer's Table
```