Notes for Miller-Rabin Primality Test

Prime Numbers
A prime number is an integer 2 or greater that is divisible by only 1 and itself, and no other positive integers. Prime numbers are very important to public key cryptography.

Basic Primality Test
Just using the definition of primality, we can try dividing a value, n, which we are testing for primality by each integer in between 2 and n-1. If any divided in evenly (leave a remainder of 0), then n isn’t prime.

One quick observation speeds up this algorithm. Consider any factorization of a composite integer n, for example, n = 108 = 9 x 12. One number is less than or equal to the other number. This smaller number has to be less than or equal to the square root of n. If it weren’t then, there would be two numbers greater than the square root of n, multiplying exactly to n, but this is impossible. (In the example above, the square root of 108 is in between 10 and 11. It would be impossible to take two integers, 11 or greater and multiply them to get 108. The product would have to be greater than 108.)

Thus, we can speed up our basic primality test as follows:

```java
public static boolean isPrime(int n) {
    if (n < 2) return false;
    for (int i=2; i*i<=n; i++)
        if (n%i == 0)
            return false;
    return true;
}
```

Even if a modulus operation happened in constant time (which for large integers this isn’t the case), the run time of this algorithm is $O(\sqrt{n})$. For large values of n (anything much beyond $10^{18}$), this isn’t feasible in a short amount of time.
**Fermat's Theorem**

One really neat property of prime numbers is as follows:

For all prime numbers $p$ and positive integers $a$ such that $\gcd(a, p) = 1$,

$$a^{p-1} \equiv 1 \pmod{p}.$$ 

The proof is as follows:

Let $S = \{1, 2, 3, \ldots, p-1\}$

Now, let's create a set $S'$ where each value is a value multiplied in $S$ times an integer $a$, such that $\gcd(a, p) = 1$. So $S'$ looks like this:

$$S' = \{a, 2a, 3a, \ldots, (p-1)a\}$$

It turns out that the remainders, when each value in $S'$ is divided by $p$ form the set $S$.

To prove this, we must show the two following things about the set $S'$:

(a) No value in $S'$ is divisible by $p$.
(b) For all distinct items $x$ and $y$ in $S'$, $x$ and $y$ leave different remainders when divided by $p$.

The first is fairly easy to see. There's a theorem that if a prime number $p$ divides into a product of integers $ab$, then either $p$ divides evenly into $a$ or $p$ divides evenly into $b$. But if we take a look at the set of values in $S'$, each is the product of a and an integer $i$, where $i$ is in between 1 and $p-1$. It's clear that $p$ does not divide any of these components. Thus, it follows that $p$ can not divide any of the separate terms. These means that every item in $S'$, when divided by $p$, leaves a remainder that is not 0, so the possible remainders are \{1, 2, 3, \ldots, p-1\}.

To see (b), let's do a proof by contradiction. Assume the opposite, that two distinct items in $S'$ are equivalent mod $p$. It follows that there are integers $i$ and $j$ $(1 \leq i < j \leq p-1)$ such that

$$aj \equiv ai \pmod{p}$$

Now, let’s do some algebra:

$$aj - ai \equiv 0 \pmod{p}$$

$$a(j - i) \equiv 0 \pmod{p}$$

By definition of divisibility, we have that $p \mid (a(j-i))$. Since $p$ is prime, it follows that either $p \mid a$ or $p \mid (j - i)$. The first is not possible because we are given that $\gcd(a, p) = 1$. Thus, it follows that $p \mid (j - i)$, but this contradicts the fact that $j - i > 0$ and $j - i < p-2$, since $p$ does not divide any of these integers. This is a contradiction. It follows that $ai$ and $aj$ can not be equivalent mod $p$ and that no pair of values in the set $S'$ are equivalent mod $p$. 
Since none of the values of $S'$ are equivalent to 0 mod $p$ and there are $p-1$ values in $S'$, it follows that the values of $S'$ are equivalent to 1, 2, 3, ..., $p-1$ mod $p$. Thus, the sets $S$ and $S'$ are equivalent mod $p$.

Since the sets are equivalent mod $p$, their products are equivalent mod $p$. This gives us:

$$\prod_{i=1}^{p-1} a_i \equiv \prod_{i=1}^{p-1} i \pmod{p}$$

Subtract the term on the right over to the left:

$$\prod_{i=1}^{p-1} a_i - \prod_{i=1}^{p-1} i \equiv 0 \pmod{p}$$

Factor out $(n-1)!$ from both of the products:

$$\prod_{i=1}^{p-1} i \left( \prod_{i=1}^{p-1} a - 1 \right) \equiv 0 \pmod{p}$$

Applying the definition of product, we get:

$$(p - 1)! \left( a^{p-1} - 1 \right) \equiv 0 \pmod{p}$$

By definition of divisibility, we have $p \mid [(p - 1)! \left( a^{p-1} - 1 \right)]$. Since $p$ is prime, it follows that either $p \mid (p - 1)!$ or $p \mid (a^{p-1} - 1)$. The former isn't true since $(p-1)!$ only has divisors in between 1 and $p-1$, inclusive. It follows that the latter must be true. Writing this in its equivalent mod form we get:

$$a^{p-1} - 1 \equiv 0 \pmod{p}$$

Adding 1 to both sides we get:

$$a^{p-1} \equiv 1 \pmod{p}$$
Probabilistic Primality Testing

The difficulty with the standard method for testing for primality is that it’s a time-consuming task. The standard method of trial division would take thousands of years with some of the numbers we want to deal with. Recently, there has been the discovery of a polynomial-time algorithm to test for primality, but for common everyday procedures, it is also too time-consuming.

It turns out that the most practical solution to this problem is utilizing a probabilistic algorithm. A probabilistic algorithm is one that doesn’t ALWAYS return the correct answer, but does so with some probability. This may sound unacceptable, but amazingly enough, we can utilize a test that is 75% accurate to create an overall algorithm that is accurate nearly every time.

Our key goal is to differentiate/categorize integers into one of two categories: prime or composite. If we can find a property that one group has that the other group doesn't (most of the time), then that can be our litmus test for categorizing integers. This isn't the most intuitive litmus test, but it turns to work out quite well. Earlier in these notes we found out that for all $1 < a < p$, where $p$ is prime,

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

This is a statement true for all primes, but as it turns out, is false for most values of $a$, for most composites. (Remember for composite numbers, $n$, the correct exponent is $\phi(n)$, which is strictly less than $n-1$.)

Thus, the intuitive idea for our algorithm for testing if $n$ is prime or not is as follows:

1) Pick a random value $a$, $1 < a < n$.
2) Calculate $a^{p-1} \mod n$.
3) If the answer is not 1, answer composite.
4) If the answer is 1, answer "probably prime."

We know that if this calculation does not yield one, then the number we are testing is DEFINITIVELY composite, since ALL primes would yield one.

But, if we get one, we can't be positive that the number is prime, since there are some composites paired with some values of $a$ that result in an answer of one as well.

It turns out that the probability this algorithm is incorrect when it answers "probably prime" is no more than 1/2 (unless we are dealing with Carmichael numbers).
Thus, if we want to increase our confidence in the "probably prime" answer, we can simply repeat the test multiple times. Here is some pseudocode with the idea:

```java
boolean isPrime(int n) {
    for (int i=0; i<50; i++) {
        int a = rand()%n;
        if (pow(a,n-1)%n != 1)
            return false;
    }
    return true;
}
```

Basically, all we do is test 50 random values of a. If any of them triggers a value other than one, we can be sure the number is composite. The probability a composite number gives the answer of one 50 straight times is less than \((.5)^{50}\).

The only exception to this is a special set of (infrequent) numbers known as Carmichael numbers. These are composite numbers for which each value of a that is relatively prime to n always yields 1 in this computation.

To thwart Carmichael numbers, the Miller-Rabin test utilizes a further property of Fermat's theorem. In particular, each possible value a has an "order" mod p. The order is simply the smallest exponent that a must be raised to, to obtain 1 mod p. From this point on, the modular exponentiation values cycle. Here are a couple examples:

\[
\begin{align*}
2^1 &= 2 \mod 7 & 3^1 &= 3 \mod 7 \\
2^2 &= 4 \mod 7 & 3^2 &= 2 \mod 7 \\
2^3 &= 1 \mod 7 & 3^3 &= 6 \mod 7 \\
2^4 &= 2 \mod 7 & 3^4 &= 4 \mod 7 \\
2^5 &= 4 \mod 7 & 3^5 &= 5 \mod 7 \\
2^6 &= 1 \mod 7 & 3^6 &= 1 \mod 7 \\
\end{align*}
\]

(order of 2 mod 7 is 3, order of 3 mod 7 is 6.)

Let k be the order of a mod p, for some prime p. What often happens is that \(a^{k/2} = -1 \mod p = (p-1) \mod p\), if k is even. (This is true for the second example above.) But, for Carmichael numbers, this property doesn't typically hold.
The Miller-Rabin test utilizes this fact. Here is the algorithm for testing if \( n \) is prime:

1. write \( n - 1 = 2^k m \), where \( m \) is odd.
2. choose a random \( a \), \( 1 < a < n \).
3. Compute \( b = a^m \mod n \).
4. if \( b == 1 \), answer probably prime and quit.
5. for (\( i=0; i<k; i++ \))
6. if (\( b = -1 \mod n \))
   answer probably prime and quit.
   else
   \( b = b^2 \mod n \) (taken from *Cryptography: Theory and Practice* by Stinson)
7. if you get here, answer "composite"

The basic rationale here is that if we look at the following list of numbers mod \( n \):

\[
a^m, a^{2m}, a^{4m}, \ldots, a^{(n-1)/2}
\]

for a prime number, either the first one will be 1, or one of the values on the list will be -1. If this isn't true, the number is definitively composite. Furthermore, these restrictions are more stringent than the original, so fewer composite numbers will be able to pass this test. In particular, this test thwarts Carmichael numbers.

The error of this algorithm is at most 25% (better than the previously stated 50%). Thus, if we run this algorithm 50 times and it reports "probably prime", we can be sure with probability \( 1 - (0.25)^{50} \) that the number is indeed prime.