Miller Rabin Primality Test

Many public key cryptography schemes need a method to generate large prime numbers. The standard way to do this would be to generate a random odd integer and then test to see if it's prime or not.

The difficulty with this method is that testing for primality is 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 probablistic 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,

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

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 $a^{n-1} \bmod n$.
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:

```
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:

$2^1 = 2 \bmod 7$	$3^1 = 3 \bmod 7$	
$2^2 = 4 \bmod 7$	$3^2 = 2 \bmod 7$	
$2^3 = 1 \bmod 7$	$3^3 = 6 \bmod 7$	
$2^4 = 2 \bmod 7$	$3^4 = 4 \bmod 7$	
$2^5 = 4 \bmod 7$	$3^5 = 5 \bmod 7$	(order of 2 mod 7 is 3,
$2^6 = 1 \bmod 7$	$3^6 = 1 \bmod 7$	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 $a^{k/2} = -1 \bmod p = (p-1) \bmod p$, 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 $b = a^m \bmod n$.
4. if b == 1, answer probably prime and quit.
5. for (i=0; i<k; i++)
6.    if ($b = -1 \bmod n$)
          answer probably prime and quit.
      else
          $b = b^2 \bmod n$
7. if you get here, answer "composite"
```

(taken from Cryptography: Theory and Practice by Stinson)

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 $1 – (.25)^{50}$ that the number is indeed prime.