Primality Testing

Brute force primality testing – try dividing a number by each integer from 2 to the square root of the number:

Example 143… (note square root is 11.something)

143 % 2 No

143 % 3 No

143 % 4 No

143 % 5 No

143 % 6 No

143 % 7 No

143 % 8 No

143 % 9 No

143 % 10 No

143 % 11 = 0 Yes, so 143 = (143/11) x 11 = 13 x 11. (143 is NOT prime)

Example 67 (sqrt of 67 is 8.someting)

67% 2 No

67 % 3 No

67 % 4 No

67 % 5 No

67 % 6 No

67 % 7 No

67 % 8 No

67 is prime

67 = a x b, then if a and b are equal then a = sqrt(67), otherwise, if a < b then a < sqrt(67) and b > sqrt(67).

Formal proof: n = ab, assume the opposite that both a and b are greater the sqrt(n), but if that were the case, then n = a x b > sqrt(n) * sqrt(n) = n, contradiction because this says that n > n…so it's impossible for all non-trivial factors of an integer n to greater than the square root of n.

Run time is at least O(sqrt(n))…in reality, this is probably O(sqrt(n)*log(n))

Can we do better?

Yes, in a weird sort of way…

What we want is a quality of prime numbers such that all prime number satisfy this quality and most other numbers don't.

This is where Fermat's comes in…

All prime numbers satisfy this requiement:

If gcd(a p) = 1, then $a^{p-1} = 1 \pmod{p}$

Generally speaking, for most composite numbers, this will not hold…

So here's the idea….to prime test for n

1) Pick a random value for a. (1 < a < n)
2) If gcd(a, n) != 1, answer that n is composite.
3) Calculate a^{n-1} mod n.
4) If the answer in step 3 is NOT 1, we know for a fact the number is composite.
5) Otherwise, we can answer "it's probably prime"

If the n is really prime this algorithm guarantees to answer "it's probably prime."

If n is composite, there is a small chance, that it accidentally answers "it's probably prime"

Probabilistic Algorithms – algorithms that might be wrong, but we can carefully bound how they can be wrong and how often that occurs.

How often is this algorithm wrong when n is composite?

Turns out that for vast majority of composite numbers, this algorithm is correct at least 50% of the time, or even 75% of the time.

What we can do is repeat the test 100 times with 100 different values of a. The probability it's wrong once if n is composite is 50%. The probability it's wrong 100 times in a row is $.5^{100}$ which is a really small number.

THERE IS ONE CATCH!!!

There are a tiny set of composite numbers for which the 50% estimate is wrong. These numbers are called Carmichael numbers. For these numbers, the regular Fermat test I have above always passes.

First Carmichael # is 561, 1105, 1729, … (Ramanujan…movie The Man Who Knew Infinity) 1729 = 1 + 1728 ($1^3 + 12^3$) = 1000 + 729 ($10^3 + 9^3$)

Miller-Rabin Primality Test

For most primes, if they aren't generators, then you'll see a 1 on the exponentiation chart of a^x, at some power x where x is (p-1)/2 or (p-1)/4 or (p-1)/8 etc.

MillerRabin(n)

---

1. Calculate $n - 1 = 2^k n'$, where n' is odd (factor out as many 2s as you can)
2. Pick your random value a. (1 < a < n)
3. If gcd(a, n) != 1, output composite.
4. Calculate $Y = a^{n'} \bmod n$. If this is 1, answer is probably prime.
5. Repeatedly square Y, k-1 times. If you ever get n-1, answer is probably prime
6. If you never get n-1 in this sequence, answer composite.

Even though this could be wrong, it's wrong so infrequently and so much faster than the other method, this is what is accepted and used for primality testing.

FACTORING

Fermat Factoring Method

We assume that our integer is the product of two odd primes

N = xy, where x, y are odd primes

Goal: given n, figure x and y.

$13 \times 19 = (16 - 3)(16 + 3) = 16^2 - 3^2$

$N = a^2 - b^2 = (a+b)(a-b)$

Well, a > sqrt(N).

N = 247, sqrt(n) > 15

Try a = 16

$247 = 16^2 - b^2 \rightarrow b^2 = 256 - 247 = 9$, b = 3 done

N = 20453, sqrt(n) > 143

A	$A^2 - 20453$	
144	283	Not a perfect square
145	572	Not a perfect sqare
146	863	Not a perfect square
147	$1156 = 34^2$	

20453 = (147 – 34)(147+34) = 113 x 181

Next time: Pollard-Rho Factoring, Fast Modular Exponentiation Code