Prime Numbers, Fermat's Theorem

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. We have see these numbers and referenced them in the classical cryptography portion of the course.

For this lecture, before we get into Fermat's Theorem, we'll go over a few things:

1) Standard way to determine if a positive integer is prime or not.
2) Fundamental Theorem of Arithmetic
3) How to Prime Factorize an Integer (by hand and with code)
4) How to list all prime numbers upto a target integer n efficiently, in code.

Standard Primality Test

Note that if an integer n is composite, then there exist two integers a and b such that $a > 1$, $b > 1$, and $n = ab$.

Notice that it's impossible for both $a > \sqrt{n}$ and $b > \sqrt{n}$ because if both of these statements were true, then $ab > \sqrt{n}\sqrt{n} = n$. It follows that, all composite integers, n, have at least one divisor greater than 1 that is also less than or equal to the square root of n.

Thus, our most simple test of primality is checking if each integer less than or equal to the square root of n divides evenly into n or not. Here is a function that accomplishes that in C:

```
// Pre-condition: n >= 2.
// Post-condition: Returns 1 iff n is prime.
int isPrime(int n) {
    for (int i=2; i*i<=n; i++)
        if (n%i == 0)
            return 0;
    return 1;
}
```

We just try dividing n by each possible candidate. Note how we avoid floating point comparisons in the for. The loop runs roughly $\sqrt{n}$ times. Unfortunately, for very large, n, this is too slow.

Fundamental Theorem of Arithmetic

We state but don't prove this here: All positive integer have a unique prime factorization. Here are three examples:

$75 = 3^1 5^2$
$210 = 2^1 3^1 5^1 7^1$
$96 = 2^5 3^1$

Formally, each positive integer, n, can uniquely be expressed as follows:

$$n = \prod_{i=1}^{\infty} {p_i}^{a_i}$$

Where each p_i is a unique prime number and a_i is the corresponding non-negative integer exponent.

Prime Factorizing by Hand

To prime factorize by hand, simply try dividing a given integer n by 2, 3, 5 etc. or try factorizing into smaller multiplicative factors until each is broken down into prime numbers.

Here is one example:

$$\begin{aligned} 225{,}000{,}000 &= 225 \times 10^6 \\ &= 15^2 \times (2 \times 5)^6 \\ &= (3 \times 5)^2 \times 2^6 \times 5^6 \\ &= 3^2 \times 5^2 \times 2^6 \times 5^6 \\ &= 2^6 \times 3^2 \times 5^8 \end{aligned}$$

Code to Prime Factorize

To do so by a program, all you have to do is edit the prime test…every time you find a prime factor, divide it out of n and keep it in your tally. Here is some code that prints out each prime factor of an integer n. In this version, a prime p raised to the power k gets represented by printing p k times.

```
// Pre-condition: n >= 2.
// Post-condition: Prints prime factorization of n.
void printPrimeFact(int n) {
    for (int i=2; i*i<=n; i++) {
        while (n%i == 0) {
            printf("%d ", i);
            n /= i;
        }
    }
    if (n>1) printf("%d", n);
}
```

A more common form of this is to store a listing of the prime factorization and return that. This is left as an exercise to the reader.

The key idea here is that you try each prime divisor until it doesn't divide out evenly any more. The run time of this code is similar to that of the original prime factorization code.

Prime Sieve

A simple, efficient way to mark all the prime numbers upto n is as follows:

```
int* isprime = malloc((n+1)*sizeof(int));
for (int i=2; i<=n; i++) isprime[i] = 1;

for (int i=2; i*i<=n; i++)
    for (int j=2*i; j<=n; j+=i)
        isprime[j] = 0;
```

The key idea is as follows: We know that multiples of any prime (except itself) are not prime. So, 4, 6, 8, 10, etc. are not prime. We first assume all integers from 2 to n are prime. We do this in the first for loop where we set each index of the array isprime to 1, from index 2 to index n. Next, we'll go ahead and start marking off composite numbers. When i = 2, notice that j starts at 4, and then we add 2 each time, so j = 4, 6, 8, 10, etc. and we mark each of these as composite by setting isprime of these indexes to 0 (which means not prime).

When the code is done, if isprime[i] == 1, then i is prime, and if isprime[i] == 0, then i is not prime, for all integers i, in between 2 and n, inclusive.

The run time of this is quite fast (try it yourself)! This can run on a regular laptop for upto n = 60 million in about a second or so!

Now we can move onto a more advanced idea in number theory: Fermat's Theorem!

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, …, 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, …, (p-1)a}

It turns out that the remainders, when each value in S' is divided by p the remainders 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, …, 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 ($1 \le i < j \le p-1$) 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 | (a(j-i)). Since p is prime, it follows that either p | a or p | (j - i). The first is not possible because we are given that gcd(a, p) = 1. Thus, it follows that p | (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:

$$\prod_{i=1}^{p-1} ai \equiv \prod_{i=1}^{p-1} i \ (mod\ p)$$

Subtract the term on the right over to the left:

$$\prod_{i=1}^{p-1} ai - \prod_{i=1}^{p-1} i \equiv 0 \ (mod\ p)$$

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

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

Applying the definition of product, we get:

$$(p-1)!\,(a^{p-1} - 1) \equiv 0 \ (mod\ p)$$

By definition of divisibility, we have $p|\ [(p-1)!\,(a^{p-1}-1)]$. Since p is prime, it follows that either $p|(p-1)!$ or $p|(a^{p-1}-1)$. 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:

$$(a^{p-1} - 1) \equiv 0 \ (mod\ p)$$

Adding 1 to both sides we get:

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