Compiling

If compiling with gcc please add the *-lm* argument so that gcc correctly includes *math.h*.

Checking if a number is prime

In the file *prime.c* my *is_prime(int x)* function checks if a number is prime by testing if *x % i == 0*, where *i* is a number in the range [2, sqrt(x)]. The square root of *x* is used as the limit as there is no need to test beyond this point. Using the square root of *x* as the limit can greatly reduce the number of operations that need to be performed per call.

Sieve of Eratosthenes

In the file *prime_with_sieve.c* I have implemented the Sieve of Eratosthenes to find prime numbers. Before *is_prime(int x)* can be called the sieve must first be initialised. An array of ints, called *primes*, is filled with 1 or 0 depending on whether or not the index of the cell is a prime number. Example: *primes[2]* is 1 as 2 is prime, while *primes[4]* is 0 as 4 is not prime.

This approach is more efficient that my previous *is_prime(int x)* function, but it requires a possibly large array to be kept in memory.

Semiprimes

Definition: a semiprime is a number that can be written as *p x q*, where *p* and *q* are both primes (and *p = q* is permitted).

One key property of semiprimes is used in my *semiprime.c* file to determine if a number is semiprime: namely that a semiprime only has 1, *p*, *q* and itself as factors.

To take advantage of this fact my *is_semiprime(int x)* function does the following:

1. Starting from *i = 2* and going to *i = sqrt(x)* find *i* and *j* such that:
    a. *x % i == 0*
    b. *j = x / i*
    c. *x % j == 0*
2. Return 1 if *i* and *j* are prime, else return 0

Once a non-prime factor that is not 1 or *x* itself has been found the function returns 0 as *x* cannot be semiprime. Likewise as soon as two prime factors have been found the function returns 1. This saves time as it does not have to find all factors of a number.