**Lab Three: BigIntegers**

In Lesson 3, we learned about the `java.math.BigInteger` class and studied as a m p l e p r o g r a m t o c o m p u t e p o w e r s:

```java
/*
June 9, 2020 Walter Carlip Compute large powers.
*/
import java.math.BigInteger;
public class BigPowers {
    public static BigInteger power(String m, String n){
        BigInteger bigbase = new BigInteger(m);
        BigInteger bigpower = new BigInteger(n);
        BigInteger biganswer = new BigInteger(m);
        long stop = Long.parseLong(n);
        for(long i = 2; i <= stop ; i++){
            biganswer = biganswer.multiply(bigbase);
        }
        return(biganswer);
    }
    public static void main(String[] args) {
        System.out.println(power(args[0],args[1]));
    }
}
```

1. Enter and run this program as well as its `int` version. Do both programs work? Do you see why `BigInteger` is useful?

**Some Problems**

**Exercise.** Write a method that uses `BigIntegers` to compute huge factorials. [Hint: You could start with a copy of your old factorial program and modify it.]

**Exercise.** Modify the program `Euclid`, that uses the Euclidean Algorithm to compute greatest common divisors, to use `BigIntegers`.

**Exercise.** Modify the program `FibonacciArray`, to use `BigIntegers` to compute gigantic Fibonacci numbers.
Exercise. Modify the prime testing and divisor counting programs in Lab 2 to use BigIntegers.

Exercise. Write a program that uses the Sieve of Eratosthenes to compute a list of prime numbers less than 5000. Here are some hints to get you started:

1. Use an array of type boolean declared with a statement like this:

   ```java
   boolean[] mySieve = new boolean[5000];
   ```

   in your `main` method. This statement tells Java to allocate a new array of 5000 boolean variables with the name “mySieve”.

2. Write a method called `initialize` that loops through the array and sets every entry to the value `true`. Such a method would begin with the line

   ```java
   public void initialize(boolean[] input){ . . .
   ```

   and can be called from your `main` method by writing

   ```java
   initialize(mySieve);
   ```

3. Write a method called `sieve`, that runs through the array (following Eratosthenes’ algorithm) and changes the entry `mySieve[n]` to `false` if the number `n` is not prime. Begin the method with the line

   ```java
   public void sieve(boolean[] input){ . . .
   ```

   and invoke it in your `main` method by

   ```java
   sieve(mySieve);
   ```

4. Write a method called `printSieve` that runs through the sieve and prints out the value `n` if `mySieve[n]` is still true after `sieve` has been run.
Exercise. Modify your program that factors Mersenne numbers to use BigInteger. Can you find any additional Mersenne primes? (It is unknown whether there are infinitely many Mersenne primes. Circa 2017, the largest known Mersenne prime is $2^{77,232,917} - 1$. Approximately how many decimal digits does this number have?)

Exercise. The function $\pi(n)$ is defined to be the number of prime integers less than $n$. Revise your implementation of the Sieve of Eratosthenes (in the previous exercise) to compute the function $\pi(n)$.

Exercise. Use the idea of a number sieve (as in the previous problem) to write a program that types out a list of Ulam numbers.

Exercise. A perfect number $n$ is a positive integer with the property that the sum of its divisors is equal to $2n$. Examples are the number 6, since $2 \cdot 6 = 12 = 1 + 2 + 3 + 6$, and 28, since $2 \cdot 28 = 56 = 1 + 2 + 4 + 7 + 14 + 28$.

Write a program that finds perfect numbers.

1. In your first version, write a method that checks whether the input argument is perfect.

2. In your second version, use two methods: one method that checks whether it’s argument is perfect; one main method that loops through the integers from 1 up through its input argument, and prints out that integer if it is perfect.

Exercise. The Euler phi-function $\phi(n)$ is equal to the number of integers $k$ such that $k < n$ and $\gcd(n, k) = 1$, that is, the number of integers less than $n$ and relatively prime to $n$. Write a program that computes $\phi(n)$, i.e., that evaluates the Euler phi-function on an integer $n$ given as the input argument.