Computer Laboratory 5
CSCI 1913: Introduction to Algorithms, Data Structures, and Program Design
October 8–9, 2019

0. Introduction.

A *prime number* is a positive integer with exactly two distinct divisors: 1 and itself. The *Sieve of Eratosthenes* (pronounced *Era-TOSS-the-knees*) is an algorithm for finding prime numbers. It's named after a Greek mathematician of the Third Century BCE. In this lab assignment, you will implement the Sieve as a Java class, and use an instance of that class to print a list of prime numbers.

1. Theory.

The Sieve algorithm finds prime numbers between 2 and some positive integer *n.* Unlike other prime number algorithms, it doesn't need square roots, multiplication, or division. It works in the following way.

1. Make an empty set.

2. Add integers 2, 3, ..., *n* to the set.

3. Visit each integer in the set. After you visit an integer, remove all other integers from the set that are multiples of that integer. For example, after you visit the integer 2, you remove the integers 4, 6, 8, etc.

4. The integers left in the set are primes.

For example, suppose you want to find the prime numbers between 2 and 25. You start with the following set of numbers.

2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25

When you visit the integer 2 (it's underlined), you remove all other integers that are multiples of 2 (they turn gray), so the set looks like this.

2 3 4 **5** 6 **7** 8 **9** 10 **11** 12 **13** 14 **15** 16 **17** 18 **19** 20 **21** 22 **23** 24 **25**

Then you visit 3, removing integers that are multiples of 3.

2 3 4 **5** 6 **7** 8 9 10 **11** 12 **13** 14 15 16 **17** 18 **19** 20 21 22 **23** 24 **25**

And you visit 5, removing integers that are multiples of 5.

2 3 4 **5** 6 **7** 8 9 10 **11** 12 **13** 14 15 16 **17** 18 **19** 20 21 22 **23** 24 25

The Sieve algorithm continues in this way, visiting multiples of 7, 11, 13, etc., all the way up to 25. However, there are no more primes in this set after you visit 5, so you can stop the example here. The numbers that remain in the set are primes.

You can represent a set of integers that can range from 0 to $n-1$ as an array of n `boolean` values. Each `boolean` value is either `true` or `false`. An integer k is in the set if the element at index k is `true`. It is not in the set if the element at index k is `false`.

2. Implementation.

You must write a class called `Sieve` that has the following methods, with the same names that are shown here.

`Sieve(int max)`

This is the constructor. It must make an array called `numbers`. The array's length must be `max` and its elements must be `boolean`s. If `max` is less than `2`, then throw an `IllegalArgumentException`. Set the elements at indexes `0` and `1` of `numbers` to `false`. Set all the other elements of `numbers` to `true`.

The array `numbers` represents a set of integers that will be tested for primality. Initially, `0` and `1` are not in the set, because you know they cannot be prime. However, all the other numbers, from `2` to `max - 1`, are still in the set, because you do not know yet if they are prime.

`findPrimes()`

Perform the Sieve algorithm described in the theory section. You must visit each `boolean` element of `numbers`. If you visit an element that is `false`, then you must do nothing. However, suppose that you visit an element that is `true`, at index `i`. Then you must visit all the other elements of `numbers` whose indexes are multiples of `i`, setting those elements to `false`.

`toString()`

Start with an empty string. Visit each `boolean` element in `numbers`. If the element is `true`, then add its index to the string, followed by a blank. If the element is `false`, then do nothing. The indexes in the string are the prime numbers found by the Sieve algorithm. Return the string.

As a matter of style, never write *B* `==` `true` or *B* `==` `false` where *B* is a `boolean` expression. The expression *B* `==` `true` always has the same value as *B*. The expression *B* `==` `false` can be written better as `!`*B*.

3. Tests.

The file **`TossTheKnees.java`** contains Java source code for a driver class. The class contains a `main` method, which in turn contains code that tests your class `Sieve`. Put the Java code for your class `Sieve` in this file, and run it to obtain the results of the tests in `main`. If a test succeeds, then you will receive all the points for that test. If a test fails, then you will recieve no points for it. Your score for this lab will be the sum of the points you get on the tests.

3. Deliverables.

You must submit a copy of `TossTheKnees.java` with the code for your `Sieve` class in it. If you have questions about how or where to submit your file, then ask your lab TA. Your work will be due on **Wednesday, October 16, 2019** at **11:55 PM**.