Algorithm Design

LAB 12 : PRIME NUMBER DESIGN

Due Saturday at 5:00 PM MST

This week, we will create a design for an algorithm to compute prime numbers. Note that we are representing a very specific prime number finding algorithm; the exercise here is not to invent our own algorithm.

Program Description

A prime number is a number that is only divisible by itself and 1. Thus, 2 is a prime number (the factors are 1 and 2) but 4 is not (the factors are 1, 2, and 4). We are to find all the prime numbers at or below a certain number. For example, the prime numbers at or below 20 are:

```
{ 2, 3, 5, 7, 11, 13, 17, 19 }
```

Prime Number Algorithm

To find all the prime numbers below a certain value *n*, we will start with an array containing *n* elements. Each element in the array will correspond to a number. We will start by ruling out all the multiples of two greater than 2. Thus, we will cross out the values 4, 6, 8, 10, 12, ... *n*. We know these values are not prime because they have 2 as a factor (as well as 1 and the number itself).

We will then rule out all the multiples of three greater than 3. Thus we will "cross out" the values 6, 9, 12, 15, 18, ... *n*. Note that 6, 12, 18 were already crossed out. This shouldn't matter; crossing out a value twice keeps it crossed out.

Next we will rule out all the multiples of four. Notice that 4 is already crossed out! We did that when we went through all the multiples of 2. Thus we can be assured that 4 is not a prime (its factors are 1, 2, and 4).

Next, we will rule out the multiples of five. Since 5 is not crossed out already, we will iterate through all the multiples of five greater than 5: 10, 15, 20, 25, ... *n*.

This process continues until we attempt to rule out the square root of *n*. We can stop here because everything above the square root of *n* is already crossed out. Why is that? See if you can figure it out.

Every number in our array that is not crossed out is prime!

Hints

There are a few common mistakes people make when doing this assignment.

- The list of numbers does not change size. If you are appending to the list or removing elements, then you are probably using the wrong algorithm.
- We never use MODULOUS (%) in this algorithm. We count by 2's and 3's and 5's, but we never have to perform division.

Assignment

To submit this assignment, three things are needed: a pseudocode program, the algorithmic efficiency, and a program trace. This will be submitted through I-Learn as a **single-file PDF**.

As with two weeks ago, please use the "Comments..." field to answer the following questions:

- How long did it take for you to complete this assignment?
- What was the hardest part of the assignment?
- Was there anything unclear about the instructions or how you were to complete this lab?

Pseudocode Program

Your program must do the following:

- Prompt the user for a number *n*, from which we will find all the primes at or below that value.
- Compute the primes at or below *n*.
- Place all the primes in an array.

- Display the primes on the screen.

For this assignment, our job is to create a pseudocode design of the algorithm.

Algorithmic Efficiency

You are required to compute the algorithmic efficiency of this algorithm. Both name the efficiency (such as O(log *n*)) and give a rational as to why it is what you say it is. Note that, to simplify this problem, we will say that sqrt(*n*) is close to log(*n*).

Program Trace

Please also create a program trace of your algorithm. Your program trace is to include a single test case: the primes at or below 10.

Assessment

Your grade for this activity will be according to the following rubric:

	Exceptional 100%	Good 90%	Acceptable 70%	Developing 50%	Missing 0%
Efficiency 20%	It is unambiguous that the correct algorithmic efficiency was determined	Efficiency determination and rationale are correct	Insufficient rationale or incorrect efficiency	There exists an informal discussion of the algorithmic efficiency	Algorithmic efficiency was not computed for this problem
Trace 30%	The trace is correct	The trace is correct except for one or two minor errors	One major error occurred or the trace is not detailed enough to be helpful	An attempt was made to trace the output, but it did not follow any of the guidelines	No program trace exists
Design Quality 40%	The most elegant and correct solution is found	The design completely covers the problem definition	One aspect of the problem definition is missing or one aspect of the design will not work as expected	Elements of the solution are present	Pseudocode is missing or the provided solution does not resemble the problem definition
Professionalism 10%	Professional, beautiful, elegant, single-spaced, using a fixed-width font	Everything is clear and legible, pseudocode used correctly	Misspelling, smudge, incorrect pseudocode, or examples of unprofessional-ism	At least one aspect of the design is too messy to read or is not pseudocode	Difficult or impossible to read