CSCE 111
Midterm
Fall 2022

Q0 Please make sure that you understand all the material outlined in the *Lecture Notes* document available on the Canvas home page for the class, and that you can solve all the examples and have a good idea of how to solve practice problems mentioned therein. Ask questions during recitation and/or on Piazza to get help with them. You don't need to submit anything for it.

Q1 In order to calculate whether a positive number n is prime, you divide n by *prime* numbers in the range $2 \ldots \lfloor\sqrt{n}\rfloor$. You can assume that 2 and 3 are prime. That's the method of the sieve of Eratosthenes[1]. If any of these prime numbers divides n, then n is not a prime number. But if none of these prime numbers dives n, then n is a prime. One consequence of this procedure is that you can generate all *primes* upto $\lfloor\sqrt{n}\rfloor$.

Write a program that when given a command line argument of an integer, prints the following data depending on whether it's a prime or not.

1▶ For 11

```
11 is a prime.
Primes that do not divide 11: 2, 3.
```

2▶ For 121

```
121 is not a prime.
Primes that do not divide 121: 2, 3, 5, 7.
First prime that divides 121: 11.
```

Hint: Here's a strawman idea to get you started. You know how to determine whether an integer p divides another integer q i.e., $p \mid q$. As you loop through integers from $2 \ldots \lfloor\sqrt{n}\rfloor$, at each step determine whether the current number (in the loop) is prime, and if it is, add it to another array (or ArrayList) that you maintain. Testing whether the current number i in the loop is prime is easy because you've already added all primes $< i$ to your list. If i is prime, then you test whether $i \mid n$. If it does, then n is not a prime. If you finish the entire loop, then n is a prime.

Q2 The repeating k-mer problem is adapted from *Computational Thinking for Life Scientists* by Benny Chor and Amir Rubinstein.

Suppose we are given a string S and an integer k, and we ask whether S contains a substring of length k (k-mer) that repeats itself (appears at least twice). For example, for $S =$ CCTAGTCCA

[1] See https://en.wikipedia.org/wiki/Sieve_of_Eratosthenes.

and $k = 2$ there is a repeating 2-mer CC. But there is no repeating 3-mer (check it out). In this problem we do not consider overlapping repeats. That is, two repeating sub-strings of length k are valid duplicates only if they don't overlap each other. For e.g., the string GATATACA has a duplicate 3-mer ATA but the two instances overlap each other.

Write a Java program that takes a filename and k as arguments on the command line, reads all the lines in the specified file to concatenate them into one (long) string, and finds all non overlapping k-mers for the specified k. In particular, validate your program by running it on the Mycobacterium tuberculosis genome data (provided in Mycobacterium_tuberculosis.txt) to find repeating k-mers of length 70. Does your execution terminate within 2 minutes?

Hint: No need to find a clever solution for this assignment. Just the brute force algorithm will suffice. The idea is to convey how efficiency matters in algorithms. And there's an interplay between data structures and algorithms. Simple minded solutions often work poorly in practice and with large data sizes the difference between the asymptotic complexity of two algorithms matters a lot. In this case the difference between $O(n \log n)$ versus $O(n^2)$ can be huge as n grows. $\log n$ is asymptotically smaller than any polynomial in n.

Programming Assignment #2

Magic Squares. A *magic square* of order n is an arrangement of $n \times n$ numbers, usually distinct integers, in a square, such that the n numbers in all rows, all columns, and both diagonals sum to the same value. See the Wikipedia page for Magic Square at https://en.wikipedia.org/wiki/Magic_square.

For this assignment, you will read data from a file and the format of the file will look like this.

Format of the input file

```
8 ← the size of the magic square
8,58, 59,  5 ,4,62,63,1 ← numbers surrounded by optional WS, separated by ','
49, 15  , 14 ,52,53,11,10,56
41,23,22,44, 45 ,19,18,48
32, 34, 35, 29, 28,    38,39  ,25
40,26,27,37,36,30,31,33
17,47,46,20,21,43,42,24
9,55,54,12,13,51,50,16
64,2,3,61,60,6,7,57
```

Consider the following 7×7 magic square. All rows, columns, and diagonals in the magic square add up to 175. Notice the labeling of the rows and columns. Rows are represented as $r\#$, columns are represented as $c\#$ and diagonals are represented as $d1$ and $d2$.

$d1$	$c1$	$c2$	$c3$	$c4$	$c5$	$c6$	$c7$	$d2$
$r1$	20	12	4	45	37	29	28	$r1$
$r2$	11	3	44	36	35	27	19	$r2$
$r3$	2	43	42	34	26	18	10	$r3$
$r4$	49	41	33	25	17	9	1	$r4$
$r5$	40	32	24	16	8	7	48	$r5$
$r6$	31	23	15	14	6	47	39	$r6$
$r7$	22	21	13	5	46	38	30	$r7$

Your program will take a filename as an argument on the command line, read the data from the file and print whether the data forms a magic square or not. You may assume that the data in the files is specified correctly. When you extract the integers for a row, you should trim any surrounding whitespace before you convert it to an integer.

Your output will be either

```
The data in msquare.txt is a magic square.
```

or

```
The data in msquare.txt is not a magic square. r3 and d1 are not equal.
```

where the filename argument supplied to your program is *msquare.txt*. When the data does not form a magic square, you will specify any two vectors (rows, columns, diagonals) that add up to different sum.