CSCI 2110- Data Structures and Algorithms Laboratory No. 2

Exercise 1

The sample output of Prime.java is:

number 11 prime is:31, Execution time is: 0 millisecs

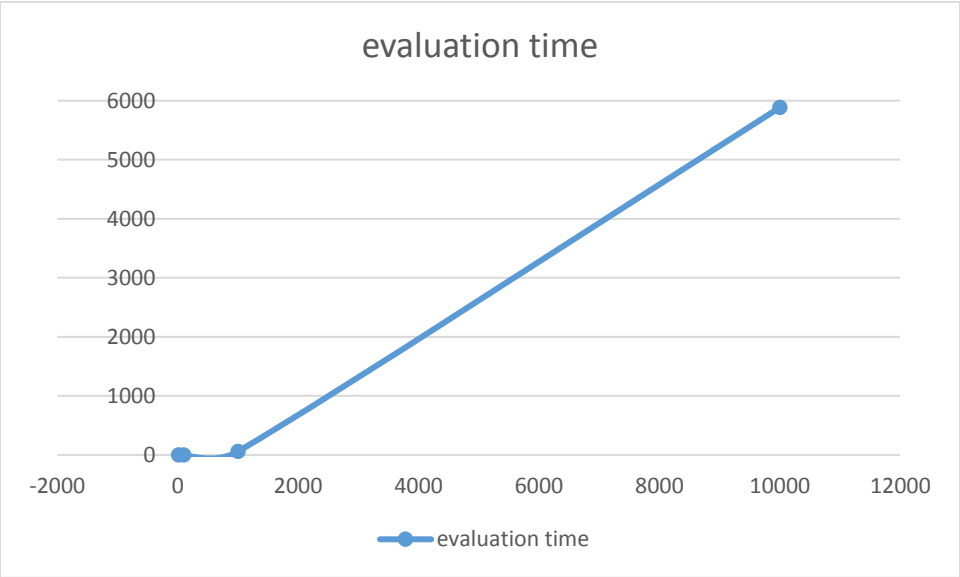
number 101 prime is:547, Execution time is: 1 millisecs

number 1001 prime is:7927, Execution time is: 59 millisecs

number 10001 prime is:104743, Execution time is: 5886 millisecs

Table 1: nth Prime Number

n	n^{th} prime number	Execution time (millisesc)
11	13	0
101	547	1
1001	7927	59
10001	104743	5886



Exercise 2

The sample output of CollatzSequence.java is:

N=5,number is:3,longest length is:8,execution time is: 0 millisecs

N=100,number is:97,longest length is:119,execution time is: 0 millisecs

N=1000,number is:871,longest length is:179,execution time is: 3 millisecs

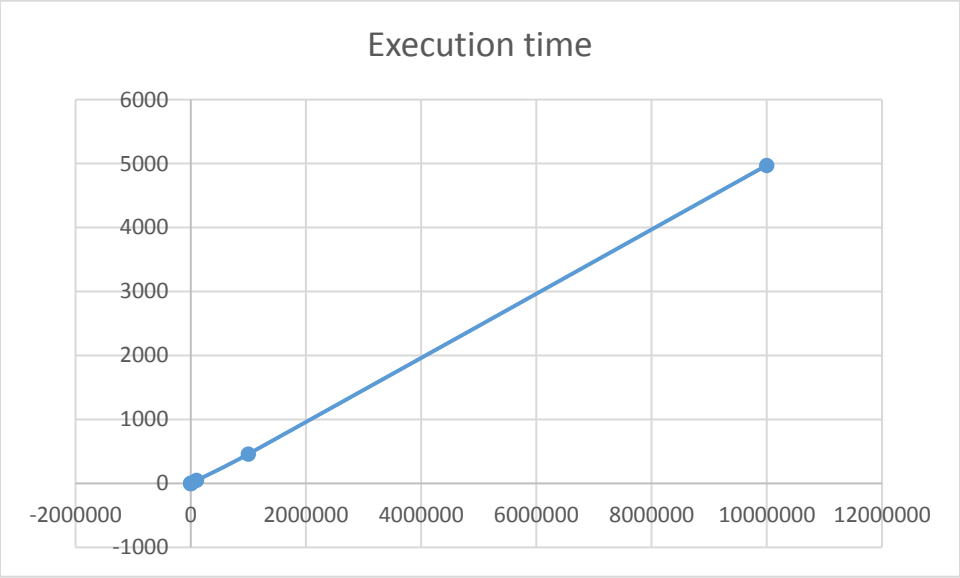
N=10000,number is:6171,longest length is:262,execution time is: 5 millisecs

N=100000,number is:77031,longest length is:351,execution time is: 47 millisecs

N=1000000,number is:837799,longest length is:525,execution time is: 459 millisecs

N=10000000,number is:8400511,longest length is:686,execution time is: 4974 millisecs

N	Starting Number with longest Collatz Sequence between 1 and N	Length of the longest sequence	Execution time (millisecs)
5	3	8	0
100	97	119	0
1000	871	179	3
10000	6171	262	5
100000	77031	351	47
1000000	837799	525	459
10000000	8400511	686	4974



Exercise 3

The sample output of CollatzSequence.java is:

Enter the size of each matrix: 100

Enter the matrix element

All elements of the matrices are assumed to be the same: 3.1

100

Execution time: 20 millisecs

Enter the size of each matrix: 200

Enter the matrix element

All elements of the matrices are assumed to be the same: 3.1

200

Execution time: 48 millisecs

Enter the size of each matrix: 300

Enter the matrix element

All elements of the matrices are assumed to be the same: 3.1

300

Execution time: 96 millisecs

Enter the size of each matrix: 500

Enter the matrix element

All elements of the matrices are assumed to be the same: 3.1

500

Execution time: 347 millisecs

Enter the size of each matrix: 1000

Enter the matrix element

All elements of the matrices are assumed to be the same: 3.1

1000

Execution time: 4362 millisecs

Enter the size of each matrix: 20000

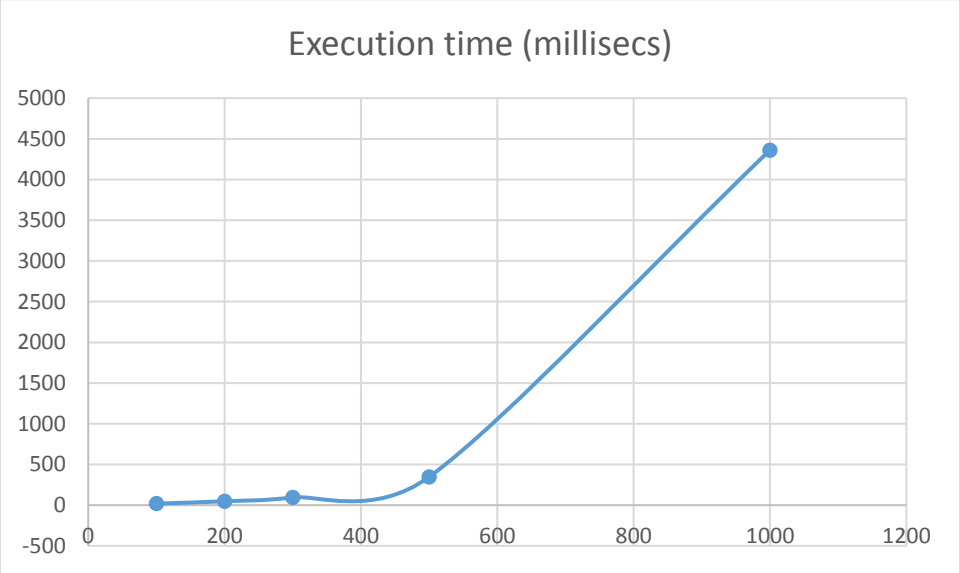
Enter the matrix element

All elements of the matrices are assumed to be the same: 3.1

Exception in thread "main" java.lang.OutOfMemoryError: Java heap space

at MatrixMult.main(MatrixMult.java:21)

Size of matrix (n)	Execution time (millisecs)
100	20
200	48
300	96
500	347
1000	4362
20000	error



Explore why this happens (you can use web resources for this) and write a brief answer.

Answers:

Firstly, it's important to know what's the error about OutOfMemoryError, there is a detailed description: https://docs.oracle.com/javase/8/docs/technotes/guides/troubleshoot/memleaks002.html

In this case, the temporary variable double[][] m will store the result to multiply m1 and m2. In order to get the cij, there are three inner FOR loops:

```
for(int i=0;i<m1.length;i++){
    for(int j=0;j<m1.length;j++){
        double sum=0.0;
        for(int k=0;k<m1.length;k++){
```

```
            sum=sum+m1[i][k]*m2[k][j];
          }
          m[i][j]=sum;
        }
      }
```

It means that there will have m1.length * m1.length * m1.length times in order to get cij. Each time, the JVM needs to store the variable sum, if the N=20000, so the JVM needs 2000*2000*2000 memory space in order to store sum. This exceeds the physical memory in my computer.

In conclusion, the method multiplyMatrix use a couple of memory which is more than the operation system's capacity. As a result, there is an error message.

In order to solve this issue, some ways are:

1. Adding a System.gc() in the second inner loop, but this way maybe does not work as expected because System.gc() is just to make a suggestion for garbage collection for JVM, if the JVM is very busy, JVM will not perform this.
2. Reducing the number of calculation times. For example, m1[0][0] * m2[0][0] was performed for a couple of times in these three FOR loops, we can store this value after the first time to calculate it. But this needs to improve the algorithm.
3. Adding the heap size configuration, such as java -Xms<initial heap size> -Xmx<maximum heap size>.

Exercise 4

The sample output of CollatzSequence.java is:

Enter the value of n: 10

Execution time to generate 2^10 binary numbers:1 millisecs

Enter the value of n: 15

Execution time to generate 2^15 binary numbers:6 millisecs

Enter the value of n: 17

Execution time to generate 2^17 binary numbers:17 millisecs

Enter the value of n: 20

Execution time to generate 2^20 binary numbers:94 millisecs

Enter the value of n: 22

Execution time to generate 2^22 binary numbers:245 millisecs

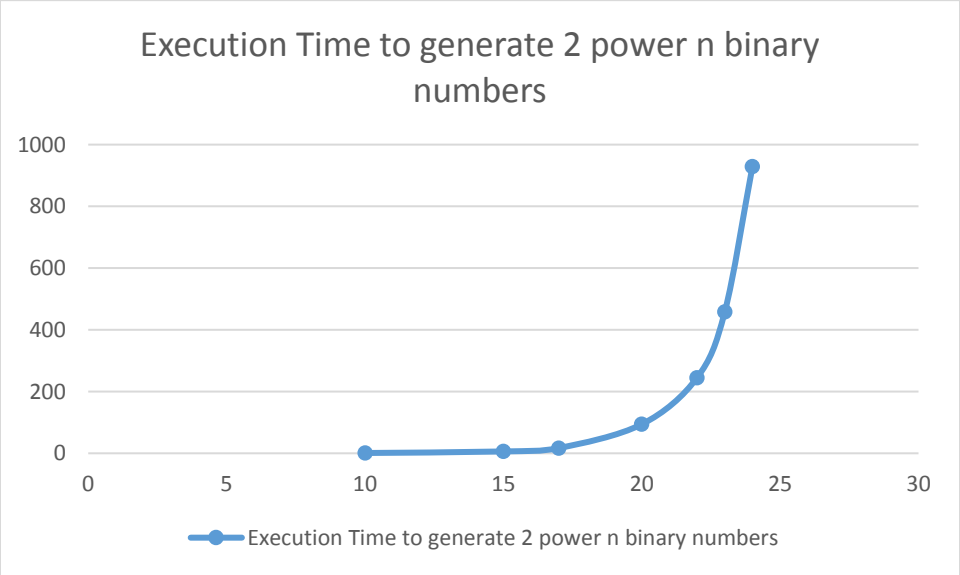
Enter the value of n: 23

Execution time to generate 2^23 binary numbers:458 millisecs

Enter the value of n: 24

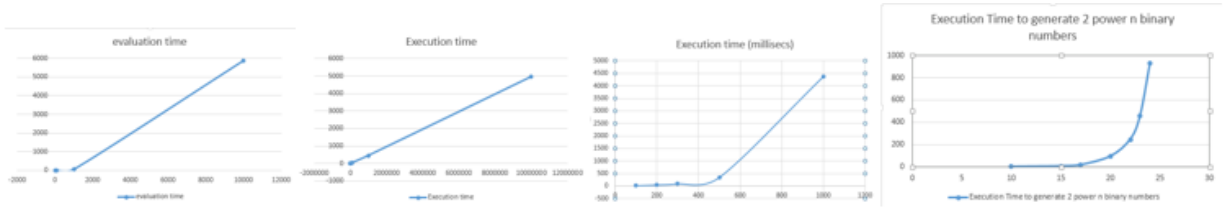
Execution time to generate 2^24 binary numbers:929 millisecs

Value of n	Execution Time to generate 2 power n binary numbers (millisecs)
10	1
15	6
17	17
20	94
22	245
23	458
24	929
30	hangs



Exercise 5: Analyze your codes using the rules of thumb discussed in the lecture and determine the orders of complexity for each of the programs in exercises 1 to 4. (Write this in a simple text document or handwritten and scanned as pdf and include it to your zip file).

Firstly, compare the graph for 4 exercises.



The curve shape of each graph is not the same. It means that they have different complexity of algorithms, as the result the Big O notation is not the same.

Secondly, calculating the basic operation for the above four exercises:

Exercise 1:

Each isPrime method calling, the basic operation is N-1; in order to find the xth prime, how to count the basic operation is not easy. At least, needs N times to get the nth prime. So the Big O is almost n*n.

Exercise 2:

In order to get the longest sequence among 2 to N, at least needs N-1. In order to get the length of Collatz sequence of each number from 2 to N, how to count the basic operation is not easy, according to the experiment result, it seems that it's like N nor x*N. so, the Big O is almost n*n.

Exercise 3:

There are three FOR loops, so the Big O is almost n*n*.

Exercise 4:

There are pow(2,n) times in the FOR loop, so the Big O is almost 2^n.

In conclusion:

	Exercise 1	Exercise 2	Exercise 3	Exercise 4
Big O notation	n^2	n^2	n^3	2^n

The result can be verified from the graph of each exercise as well.