CMP -5014Y Data Structures and Algorithm

Coursework
Assignment 1
100108964

Exercise 1: Frequency Distribution Histogram

Example

```
1-10  |***
11-20 |********
21-30 |*****
31-40 |********
41-50 |**
51-60 |*
61-70 |
71-80 |******
81-90 |*
91-100|******************
```

1. **Informal Algorithm:** Given an array of size n, where each element of the array is an integer in the range 1 to 100 inclusive, create and display a histogram to visually inspect frequency of distribution of collection of integers.

   Formal Algorithm:

Scan ***histogramArray[]*** of *n* size using for loop and add random values to from 1-100 ***max*** , then scan using nested for loop and ***print*** out values in formatted histogram.

```
1: n := 100 // or any n value
2: max := 100 //max value of random numbers
3: for i = 0 to n
4:         temp:=(int)(Math.random()*max) + 1
5:         histogramArray[temp % 10]++
6:
7: for i = 1 to i <=10
8:         print i + "-" + i * 10 + "| "
9:         for j = 0 to j < histogramArray[i-1]
10:                 print "*"
11:                 print " "
```

2. **Analyse run-time complexity:**
   a. Determine the fundamental operation(s)
      histogramArray[temp % 10]++
   b. Determine the worst case:
      All cases are the same
   c. Form runtime complexity function

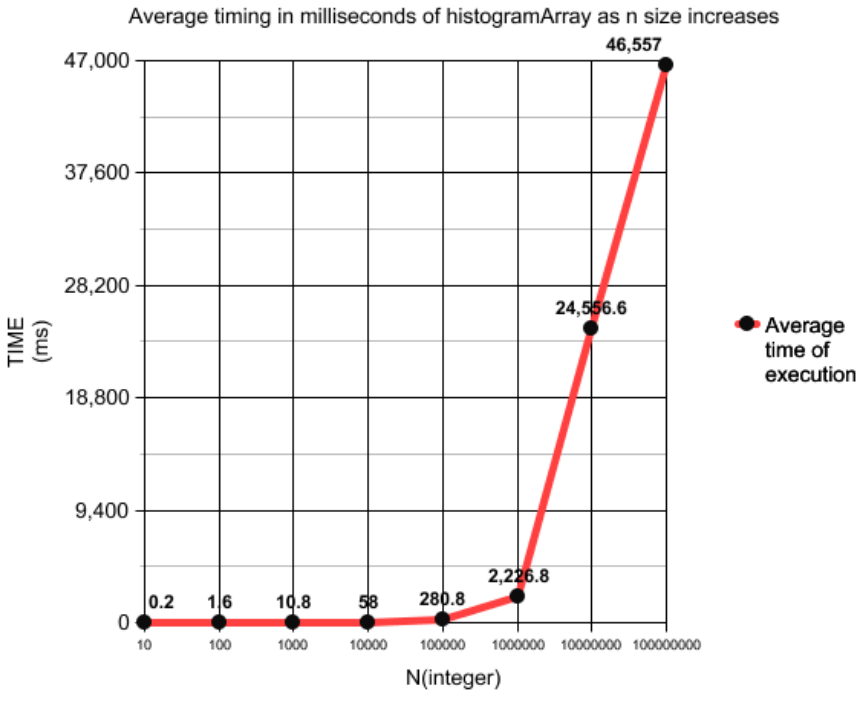
$$\sum_{i=1}^{n} \sum_{j=0}^{n} 1$$

   d. Characterise runtime complexity:
      $O(n^2)$

3. Algorithm of program

```
1.  public class Histogram {
2.
3.      public static void main(String[] args) {
4.
5.          int n = 100;
6.          int max = 100;
7.          int histogramArray[] = new int[n];
8.
9.          for (int i = 0; i < n; i++) {
10.             int temp = (int)(Math.random()*max) + 1;
11.             histogramArray [temp % 10]++;
12.         }
13.
14.         // print values in array
15.         for (int i = 1; i <= 10; i++)
16.         {
17.             System.out.print(i + " - " + i * 10 + "| ");
18.             for (int j = 0; j < histogramArray [i-1]; j++)
19.             {
20.                 System.out.print("*");
21.             }
22.                 System.out.println();
23.             }
24.     }
25. }
26.
```

4. Graph of timing experiment



Size of N	Results x5	AvgTime(ms)
10	1,0,0,0,0	0.2
100	1,2,2,2,1	1.6
1000	9,14,11,10,10	10.8
10000	58,57,55,57,63	58
100000	266,278,282,286,292	280.8
1000000	2256,2277,2163,2267,2171	2226.8
10000000	23480,23286,21804,21536,32677	24556.6
100000000	317207,339413,376804,325257,336700	46557

Conclusion:

From the graph it is apparent that as ***n*** increases so does the time it takes for the program to execute. The results seem to correspond to the analysis in part 2, increasing n by times 10 each runtime shows a clear steady increase. The first six points increase equally, after that there is an exponential increase in time it takes for the program to complete and the last two points increase equally. Repeating the program five times there is a consistent result each time – again corresponding to the analysis of $O(n^2)$ – performance which is proportionate to the square of the size of input data with the use of nested for loops.

Exercise 2: Minimum travel times

	Norwich	Cambridge	Bristol	Cardiff	Edinburgh	Glasgow
Norwich	0	58	184	271	378	379
Cambridge	58	0	167	199	351	382
Bristol	184	167	0	43	374	370
Cardiff	271	199	43	0	394	390
Edinburgh	378	351	374	394	0	47
Glasgow	379	382	370	390	47	0

1. **Informal description**: Find the distance of the closest city for all n cities in a given matrix M

   Input: Matrix M of symmetric square distances, the number of cities n.

   Output: distance of the closest city to city i. Example <58,58,43,43,47,47>

Find distance of the closest city to city **i** in Matrix **M**

```
1: n:=6 //or any n value
2: m[][]:= new int[n][n]
3://input for matrix
4: for i to n do
5:         for j to n do
6:                 m[i][j] := sc.nextInt()
7: result[] := new int[m.length]
8: min:=0
9: for i to m.length do
10:        for j to m[i].length
11:                if m[i][j]<m[i][min] && m[i][j] != 0
12:                        min := j
13:                result[i]:=m[i][min]
14: for i to result.length do
15:        print result[i]
```

2. Analyse the run-time complexity:

a. Determine fundamental operation(s):

```
if m[i][j] < m[i][min] && m[i][j] != 0
        min := j
result[i]:=m[i][min]
```

b. identify worst case:

All cases are the same

c. Form runtime complexity function

$$\sum_{i=0}^{n} \sum_{j=0}^{n} 1$$

d. Characterise run time complexity function:

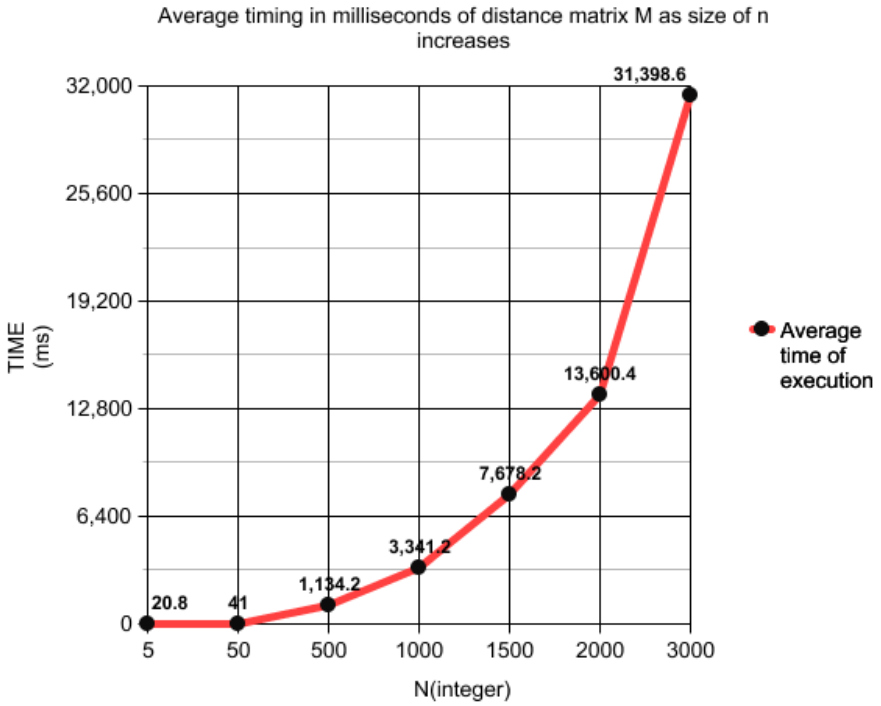
$O(n^2)$

3. Algorithm of program

```
1.
2.  package histogram;
3.
4.  import java.util.Scanner;
5.
6.  public class Distance {
7.
8.   public static void main(String[] args) {
9.
10.  int n = 10;
11.
12.  // used to test matrix table 3
13.  int mat[][] = {
14.   {0,58,184,271,378,379},
15.   {58,0,167,199,351,382},
16.   {184,167,0,43,374,370},
17.   {271,199,43,0,394,390},
18.   {378,351,374,394,0,47},
19.   {379,382,370,390,47,0}
20.  };
21.
22.  Scanner sc = new Scanner(System.in);
23.
24.  int m[][] = new int[n][n];
25.
```

```
26.  /*
27.  *example of input if n = 3 3x3 matrix:
28.  *  0 9 8  then enter
29.     9 9 0  then enter
30.     8 9 0  then enter
31.
32.  */
33.  System.out.println("enter numbers into matrix i.e add numbers to amount of rows
     then enter for each column    ");
34.
35.  for (int i = 0; i < n; i++) {
36.   for (int j = 0; j < n; j++) {
37.    m[i][j] = sc.nextInt();
38.   }
39.  }
40.
41.  // hold results of loop
42.  int[] result = new int[m.length];
43.
44.  //min value
45.  int min = 0;
46.
47.  for (int i = 0; i < m.length; i++) {
48.   for (int j = 0; j < m[i].length; j++) {
49.    System.out.print(m[i][j] + "\t");
50.
51.    if (m[i][j] < m[i][min] && m[i][j] != 0) {
52.
53.     //min becomes value in j as long as its larger than 0
54.     min = j;
55.    }
56.    result[i] = m[i][min];
57.   }
58.   System.out.println();
59.  }
60.
61.  System.out.println();
62.
63.  for (int i = 0; i < result.length; i++) {
64.
65.   System.out.print(result[i] + ",");
66.  }
67.
68. }
69.
70. }
```

4. Graph of timing experiment



Size of N	Result x5	Avg Time(ms)
5	20,21,22,20,21	20.8
50	44,40,40,40,41	41
500	1116,1099,1169,1091,1196	1134.2
1000	3249,3410,3461,3327,3259	3341.2
1500	7447,7920,8047,7375,7602	7678.2
2000	14697,13165,13156,13122,13871	13600.4
3000	31383,31216,31714,31332,31348	31398.6

Conclusion:

From the graph it can be seen that the time increases as the size of n increases. The values used in n tried from 5 to 50 to 500 to 5000, however 5000 caused the program to hang, so changing it to lower size to 1000 was possible then increment it by 500 then to the last value of 3000. This still shows the corresponding increase of $O(n^2)$ from the analysis in task 2 as the program uses multiple for loops and conditional statements.