**COMP1201 Assignment 1**

**Q1:**

(a) To measure the running time of different sizes of arrays we must create, for example, three more arrays with modified sizes:

//Create more variables to store the lengths of the arrays.

int N = 1000;

int smallerN = 500;

int evenSmallerN = 100;

int biggerN = 2500;

//Create more arrays with the modified sizes

double[] smallerData = new double[smallerN];

double[] evenSmallerData = new double[evenSmallerN];

double[] biggerData = new double[biggerN];

Then we must populate the arrays, via for loops, using the *Math.random()* method.

Afterwards we must create 3 data containers (arrays) for each of the previously created arrays (data, smallerData, evenSmallerData and biggerData). The created sub-containers are copies of the arrays.

//Put all the data from the original array in these ones

double[] data1 = (double[])data.clone();

double[] data2 = (double[])data.clone();

double[] data3 = (double[])data.clone();

double[] smallerData1 = (double[])smallerData.clone();

double[] smallerData2 = (double[])smallerData.clone();

double[] smallerData3 = (double[])smallerData.clone();

double[] evenSmallerData1 = (double[])evenSmallerData.clone();

double[] evenSmallerData2 = (double[])evenSmallerData.clone();

double[] evenSmallerData3 = (double[])evenSmallerData.clone();

double[] biggerData1 = (double[])biggerData.clone();

double[] biggerData2 = (double[])biggerData.clone();

double[] biggerData3 = (double[])biggerData.clone();

Moving forward, we must call each of three algorithms (insertion sort, shell sort and quick sort) for each of the sub-data containers (for data those would be data1, data2, data3).

Example for smallerData:

//Calls the first method - InsertionSort(smallerData1);

time = (System.nanoTime()-time\_prev\_smaller)/1000000000.0;

System.out.println("Insertion Sort\nTime= " + time);

time\_prev\_smaller = System.nanoTime();

//Calls the second method - ShellSort(smallerData2);

time = (System.nanoTime()-time\_prev\_smaller)/1000000000.0;

System.out.println("Shell Sort\nTime= " + time);

time\_prev\_smaller = System.nanoTime();

//Calls the third method - Arrays.sort(smallerData3);

time = (System.nanoTime()-time\_prev\_smaller)/1000000000.0;

System.out.println("Quick Sort\nTime= " + time);

In the end, if we want to display all the items (numbers) in the array (smallerData) we could use the following:

//Display all the info System.out.println("SMALLER-DATA");

System.out.println("\tPresorted\tInsertion\t\t Shell\t\t Quick");

for (int i=0; i<smallerData.length; i++) System.out.println(smallerData[i] + " " + smallerData1[i] + " " + smallerData2[i] + " " + smallerData3[i]);

(b)

(c)

Average-case time complexity of insertion sort is O(n^2).

(d)

**Q2:**

(a)

The new class will loop through sizes 12 to 17 and create a new graph for each size.

The class will contain only one method, which is static. This way you don’t have to instantiate it.

public class GraphExtension {  
  
 public static void estimateTime() {  
  
 long time\_prev = System.nanoTime();  
 double time;  
  
 for (int i = 12; i < 18; i++) {  
 Graph graph = new Graph(i, 0.5);  
 Colouring colouring = graph.bestColouring(3);  
 graph.show(colouring);  
 time = (System.nanoTime() - time\_prev) / 1000000000.0;  
 System.out.println(i + " points time = " + time);  
 time\_prev = System.nanoTime();  
 }  
  
 }  
}  
  
  
//The main method only contains:

GraphExtension.estimateTime();

(b)

**Q3:**