Homework #1

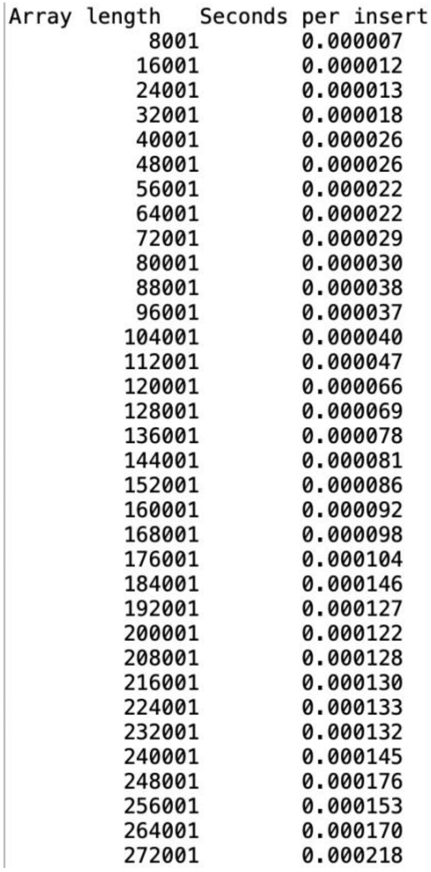
1. (2 points) Implement a method named insert. This method should take an array of ints, the index at which a new value should be inserted, and the new value that should be inserted. The function should return a new array populated with the contents of the original array with the given value inserted at the given index.

   See appendix 1 homework1.java

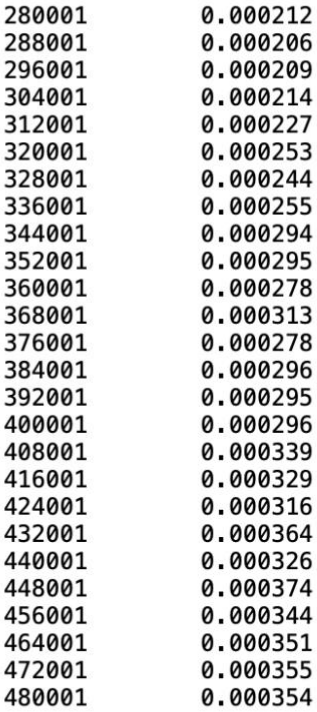
2. (2 points) Implement a main method that profiles the performance of insert and outputs a table showing the average time per insert as the length of the array increases.

   See appendix 1 homework1.java

   Output from the code:

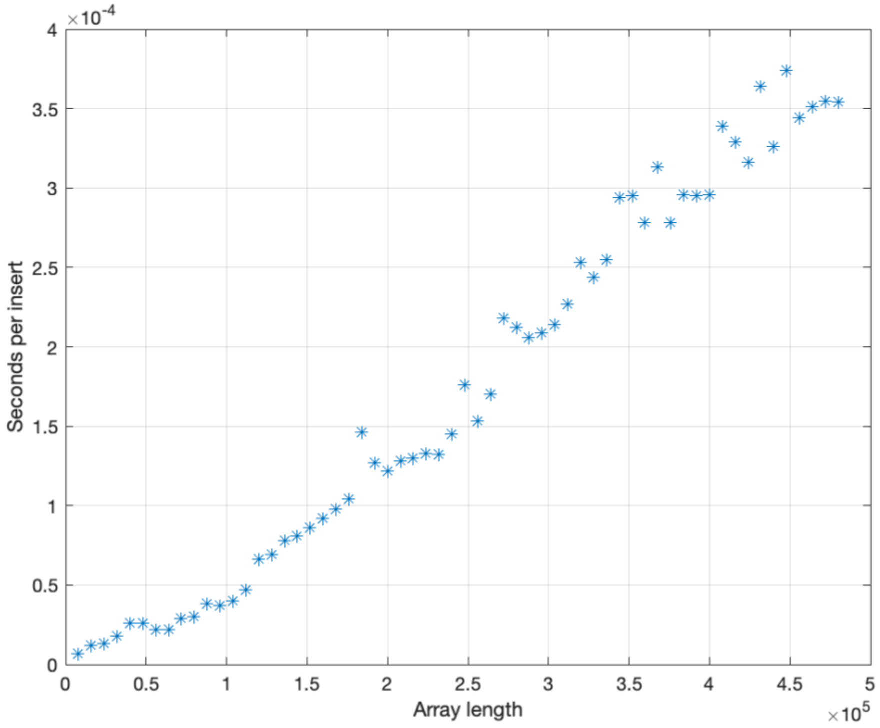


```
Array length   Seconds per insert
        8001        0.000007
       16001        0.000012
       24001        0.000013
       32001        0.000018
       40001        0.000026
       48001        0.000026
       56001        0.000022
       64001        0.000022
       72001        0.000029
       80001        0.000030
       88001        0.000038
       96001        0.000037
      104001        0.000040
      112001        0.000047
      120001        0.000066
      128001        0.000069
      136001        0.000078
      144001        0.000081
      152001        0.000086
      160001        0.000092
      168001        0.000098
      176001        0.000104
      184001        0.000146
      192001        0.000127
      200001        0.000122
      208001        0.000128
      216001        0.000130
      224001        0.000133
      232001        0.000132
      240001        0.000145
      248001        0.000176
      256001        0.000153
      264001        0.000170
      272001        0.000218
```



```
280001        0.000212
288001        0.000206
296001        0.000209
304001        0.000214
312001        0.000227
320001        0.000253
328001        0.000244
336001        0.000255
344001        0.000294
352001        0.000295
360001        0.000278
368001        0.000313
376001        0.000278
384001        0.000296
392001        0.000295
400001        0.000296
408001        0.000339
416001        0.000329
424001        0.000316
432001        0.000364
440001        0.000326
448001        0.000374
456001        0.000344
464001        0.000351
472001        0.000355
480001        0.000354
```

3. (2 points) Plot a scatter graph showing “Seconds per insert” (Y-axis) vs. “Array length” (X-axis)



(Did it with MATLAB)

4. (2 points) Provide a line-by-line Big-O analysis of your implementation of insert. You can do this by adding a comment next to each line in your source code. What is the overall Big-O performance of insert? What parts of the algorithm contribute most heavily to the overall Big-O performance?

The overall Big-O performance of ‘insert’ is O(n).
The two parts of copying elements before insert point and after insert point, contribute most heavily to the overall Big-O performance, which are both O(n).

5. (1 point) Based on the graph does the performance of improve, degrade, or stay the same as the length of the array grows? Does your Big-O analysis of match the results of running the program?

Based on the graph, as the length of array growth, the performance has degraded, since it takes more time for one insertion.
My big-O analysis, which is O(n), matches the results of running, which is also O(n) as the graph is in a linear increasing trend.

Appendix

// file homework1.java

```
package com;

import java.util.Random;



public class Homework1 {

        public static void main(String[] args) {

                // Setting to allow fine-tuning the granularity of the readings

                int NUM_READINGS = 60;

                int INSERTS_PER_READING = 8000;


                // Start with an array containing 1 element

                int[] array = new int[1];

                array[0]=0;


                System.out.format("%-15s%-15s\n", "Array length","Seconds per insert");


                // Take NUM_READINGS readings

                for (int t=0; t < NUM_READINGS; t++) {


                        // Each reading will be taken after INSERTS_PER_READING inserts

                  long startTime = System.currentTimeMillis();


                        for (int p=0; p < INSERTS_PER_READING; p++) {

                                Random rn = new Random();

                                int index = rn.nextInt(array.length);

                                int value = rn.nextInt();

                                array = Homework1.insert(array, index, value);

                        }


                        long stopTime = System.currentTimeMillis();


                        System.out.println(String.format("%15d\t%15f", array.length, (stopTime - startTime) / (1000. * INSERTS_PER_READING)));

                }


        }


        private static int[] insert(int[] array, int index, int value) {

                // create new array one larger than original array

                int[] newArray;                         // O(1)

                newArray = new int[(array.length + 1)];         // O(1)
        //copy elements up to insert point from original array to new array
        for (int i=0; i<index; i++) {                          // O(n)
            newArray[i] = array[i];                        // O(1)
        }

        //place insert value into new array
        newArray[index] = value;                               // O(1)

        //copy elements after insert point from original array to new array
        for(int i=index; i<=array.length-1; i++) {            // O(N)
            newArray[i+1] = array[i];                      // O(1)
        }

        return newArray;                                       // O(1)
    }


}
```