Homework #1

In this assignment you will implement a program that inserts elements into an array, analyze the Big-O performance of this code, then profile the program to see if the actual performance matches the predicted performance.

All code implemented in this assignment should be in a class called `Homework1`.

a) **(2 points)** Implement a method named `insert`. This method should take an array of `int`s, 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. The following sections provide a detailed description of this function:

Method signature:

```
static int[] insert(int[] array, int index, int value)
```

Parameters:

`array`	The original array of ints.
`index`	The location where the value will be inserted.
`value`	The value to be inserted.

Return value:

A new array of `int`s containing the contents of the original `array` plus the new `value` inserted at the given `index`.

Pseudocode:

```
// Create new array one larger than original array
Let newArray = a newArray with array.length + 1 elements

// Copy elements up to insert point from original array to new array
Loop to copy array[0, index) to newArray[0, index)

// Place insert value into new array
Set newArray[index] to value

// Copy elements after insert point from original array to new array
Loop to copy array[index, length) to newArray[index + 1, length + 1)

Return newArray
```

Answer: See file “Homework1.java”.

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

Pseudocode:

```
main()
   /* Setting to allow fine-tuning the granularity of the readings */
   Let INSERTS_PER_READING = 1000

   /* Start with an empty array */
   Let array = empty array (i.e. NULL)
   Let length = 0

   /* Take 60 readings */
   Loop 60 times

      /* Each reading will be taken after INSERTS_PER_READING inserts */
      Let startTime = current time
      Loop INSERTS_PER_READING times
         Let index = random integer in range [0, length]
         Let value = random integer value
         Let array = insert(array, length, index, value)
         Let length = length + 1
      End Loop
      Let stopTime = current time
      Let timePerInsert = (stopTime - startTime) / INSERTS_PER_READING

      /* Output reading in tabular format */
      Output array length and timePerInsert

   End Loop

   /* Free the old array */
   Free array
```

Report format:

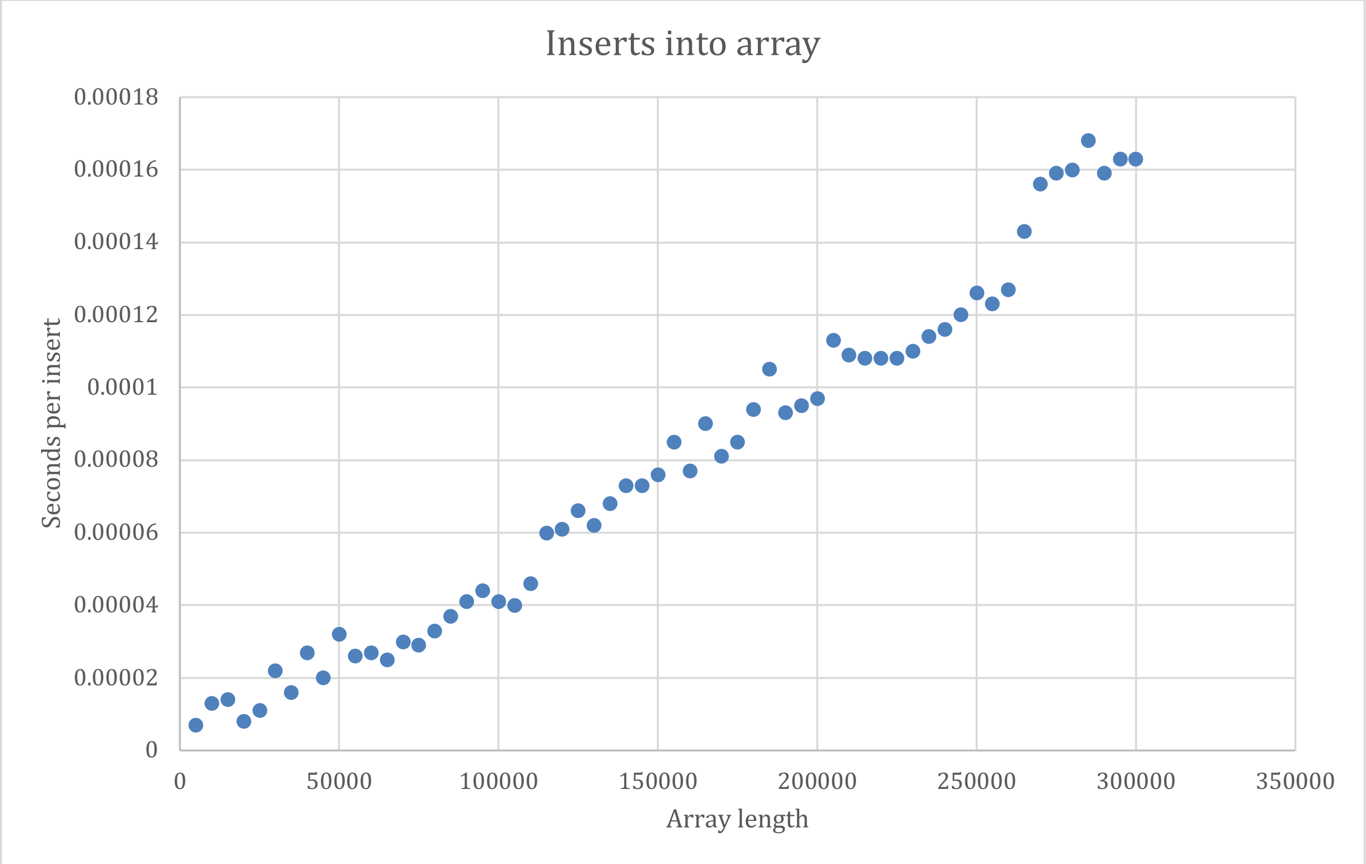
`main` should output a report similar to the format below (your values will be different). You should fine-tune the INSERTS_PER_READING constant so that none of the readings ("Seconds per insert") are zero:

```
Array length        Seconds per insert
        1000                  0.000024
        2000                  0.000028
        3000                  0.000041
        4000                  0.000036
         ...                       ...
       57000                  0.000262
       58000                  0.000318
       59000                  0.000324
       60000                  0.000328
```

Answer: See file "Homework1.java".

c) **(2 points)** Plot a scatter graph showing "Seconds per insert" (Y-axis) vs. "Array length" (X-axis) using the profiling data that was output by main.

Answer:



d) **(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?

Answer: See file "Homework1.java" for line-by-line analysis of implementation of insert.

The overall performance of insert is O(n). The two loops that copy the original array to the new array contribute most heavily to the overall Big-O. Both of these loops add up to O(n) which determines the overall performance. The rest of the lines in the method are O(1).

e) **(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?

Answer: The graph shows that the performance of insert degrades as the length of the array grows. My Big-O analysis matches the results (both show "linear" O(n) performance).
The fluctuations above & below a pure linear increase in insert time may be due to the load caused by other processes running or the Java garbage collector running.

f) **(1 point)** Make sure your source code is well-commented, consistently formatted, uses no magic numbers/values, follows programming best-practices, and is ANSI-compliant.

Answer: See file "Homework1.java".

Turn in all source code, program output, diagrams, and answers to questions in a single Word document.