# Program Structures and Algorithms

## Spring 2023(SEC-01)

### Assignment 3

Name: Raja Shekar Reddy Siriganagari

NUID: 002762145

**Task:**

Your task for this assignment is in three parts.

* (Part 1) You are to implement three (3) methods (*repeat*, *getClock*, and *toMillisecs*) of a class called *Timer*. Please see the skeleton class that I created in the repository. *Timer* is invoked from a class called *Benchmark\_Timer* which implements the *Benchmark* interface.
* (Part 2) Implement *InsertionSort*(in the *InsertionSort* class) by simply looking up the insertion code used by*Arrays.sort.* If you have the *instrument = true* setting in *test/resources/config.ini*, then you will need to use the *helper* methods for comparing and swapping (so that they properly count the number of swaps/compares). The easiest is to use the *helper.swapStableConditional* method, continuing if it returns true, otherwise breaking the loop. Alternatively, if you are not using instrumenting, then you can write (or copy) your own compare/swap code. Either way, you must run the unit tests in *InsertionSortTest*.
* (Part 3) Implement a main program (or you could do it via your own unit tests) to actually run the following benchmarks: measure the running times of this sort, using four different initial array ordering situations: random, ordered, partially-ordered and reverse-ordered. I suggest that your arrays to be sorted are of type *Integer*. Use the doubling method for choosing *n*and test for at least five values of *n.*Draw any conclusions from your observations regarding the order of growth.

**Benchmarks for differently ordered arrays:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **N** | **Random** | **Pre-ordered** | **Partially-ordered** | **Reverse-ordered** |
| 250 | 0.34 | 0.00 | 0.07 | 0.25 |
| 500 | 0.63 | 0.00 | 0.27 | 0.69 |
| 1000 | 1.23 | 0.01 | 0.9 | 2.79 |
| 2000 | 3.29 | 0.01 | 3.07 | 9.73 |
| 4000 | 14.04 | 0.03 | 13.7 | 45.09 |
| 8000 | 57.59 | 0.04 | 37.04 | 92.11 |
| 16000 | 181.52 | 0.07 | 211.73 | 407.99 |

It is clear from this data that the worst case scenario is when the array is sorted in decreasing order and the best case scenario occurs when the array is sorted in increasing order.

**Total number of comparisons when the array is sorted:**

*Increasing order*:

*Decreasing order*:

**Graphs:**

**Chart, line chart

Description automatically generated**

**Chart, line chart

Description automatically generated**

**Chart, line chart

Description automatically generated**

**Chart, line chart

Description automatically generated**

**Unit test cases:**

**Text

Description automatically generated**

**Text

Description automatically generated**

**Text

Description automatically generated**

**Text

Description automatically generated**

**A screenshot of a computer

Description automatically generated with medium confidence**

**Graphical user interface, text

Description automatically generated**