**Program Structures and Algorithms**

**Assignment-3**

**Summer-2022**

**Dimpleben Kanjibhai Patel – 002965372**

**Task-1**: You are to implement three (3) methods (repeat, getClock, and toMillisecs) of a class called Timer. Please see the skeleton class that is created in the repository. Timer is invoked from a class called Benchmark\_Timer which implements the Benchmark interface. Don't forget to check your implementation by running the unit tests in BenchmarkTest and TimerTest. If you have trouble with the exact timings in the unit tests, it's quite OK (in this assignment only) to change parameters until the tests run. Different machine architectures will result in different behavior.

**Output:**

TimerTest Output:

Graphical user interface, text, application

Description automatically generated

BenchmarkTest Output:

Graphical user interface, text, application, email

Description automatically generated

**Task-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*.

**Output:**

InsertionSortTest

Graphical user interface, text, application, table

Description automatically generated

**Task-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.

**Ordered Array Output:**

Text

Description automatically generated with low confidence

**Partially Ordered Array Output:**

Table

Description automatically generated

**Random Array Output:**

Text, table

Description automatically generated with medium confidence

**Revese Ordered Array Output:**

Graphical user interface, table

Description automatically generated

**Comparison Table:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **N** | **Ordered Array** | **Partially Ordered Array** | **Random Array** | **Reversed Array** |
| 100 | 0.015 | 0.011 | 0.010 | 0.019 |
| 200 | 0.014 | 0.017 | 0.015 | 0.014 |
| 400 | 0.016 | 0.026 | 0.018 | 0.073 |
| 800 | 0.022 | 0.028 | 0.023 | 0.145 |
| 1600 | 0.025 | 0.045 | 0.049 | 0.184 |
| 3200 | 0.048 | 0.060 | 0.187 | 0.362 |

**Comparison Graph:**

Chart, line chart

Description automatically generated

**Conclusion :**

From the above graph and analysis, we observe that the order of time taken by each different arrangement of elements in an array is -

Ordered array < partially ordered array < random array < reversed array