**INFO 6205 PROGRAM STRUCTURES AND ALGORITHMS**

**ASSIGNMENT 2(BENCHMARK)**

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.

**Screenshots of all the testcases passed(part1) :**

TimerTest.java :

**A screenshot of a computer

Description automatically generated**

BenchmarkTest.java:

**A screenshot of a computer

Description automatically generated**

**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*.

**Screenshots of all the testcases passed(part2) :**

InsertionSortTest.java:

A screenshot of a computer

Description automatically generated

**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.

**Conclusion(part 3) :**

Based the values that came by using the doubling method, my observations resulted in conclusion that when the length of the array doubles, sort(Insert) time increased by four times.

**Equation: t=n2**

**Graph:**

**A screenshot of a computer

Description automatically generated**

**Screenshots:A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated with medium confidenceA screenshot of a computer

Description automatically generated with medium confidenceA screenshot of a computer

Description automatically generated with medium confidence**