

Program Structures & Algorithms

Spring 2022

Assignment No. 2

Benchmark

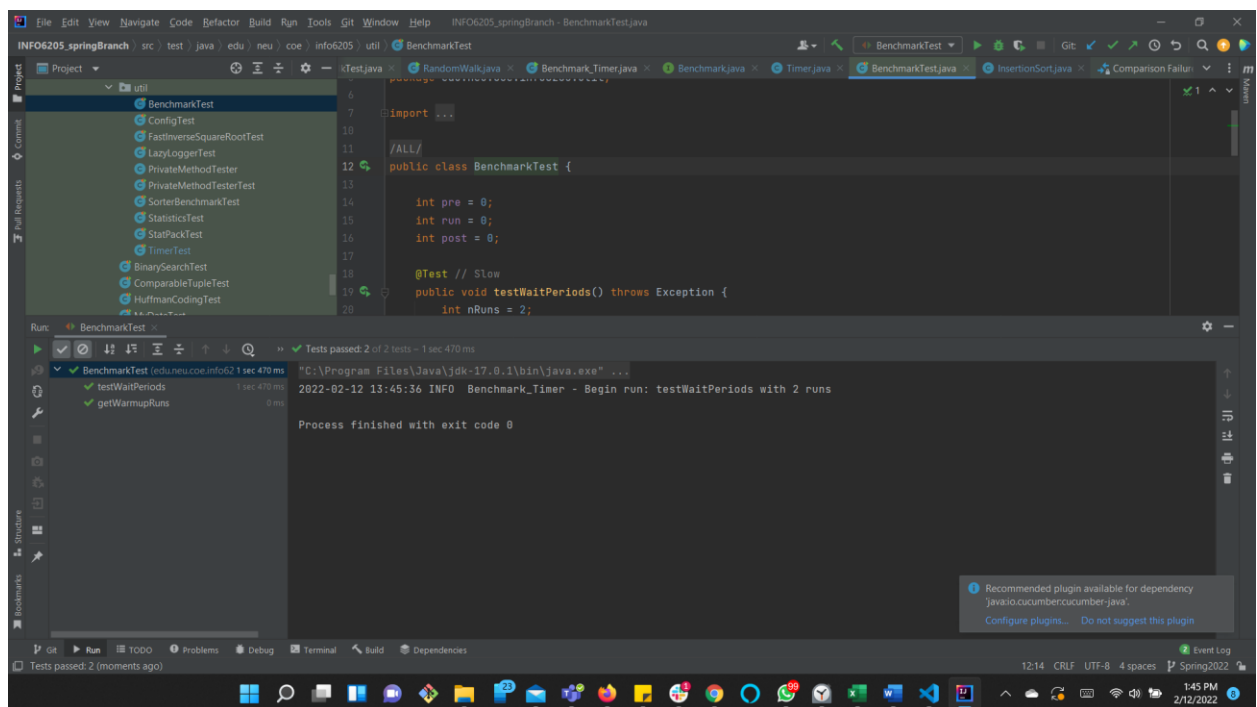
Name: Aishwarya Surve

(NUID): 002123768

TASK 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.

- Output Screenshot
 1. BenchmarkTest



The screenshot displays an IDE window for a project named 'INFO6205_springBranch'. The 'Project' view on the left shows a list of test classes under the 'util' package, including 'BenchmarkTest', 'ConfigTest', 'FastInverseSquareRootTest', 'LazyLoggerTest', 'PrivateMethodTester', 'PrivateMethodTesterTest', 'SorterBenchmarkTest', 'StatisticalTest', 'StatPackTest', 'TimerTest', 'BinarySearchTest', 'ComparableTupleTest', and 'HuffmanCodingTest'. The 'BenchmarkTest' class is selected and open in the editor. The code in the editor is as follows:

```
package edu.neu.coe.info6205.util;

import java.util.*;

//ALL
public class BenchmarkTest {

    int pre = 0;
    int run = 0;
    int post = 0;

    @Test // Slow
    public void testWaitPeriods() throws Exception {
        int nRuns = 2;
    }
}
```

The 'Run' view at the bottom shows the test results for 'BenchmarkTest'. It indicates that 2 tests passed in 1 second and 470 milliseconds. The test results are:

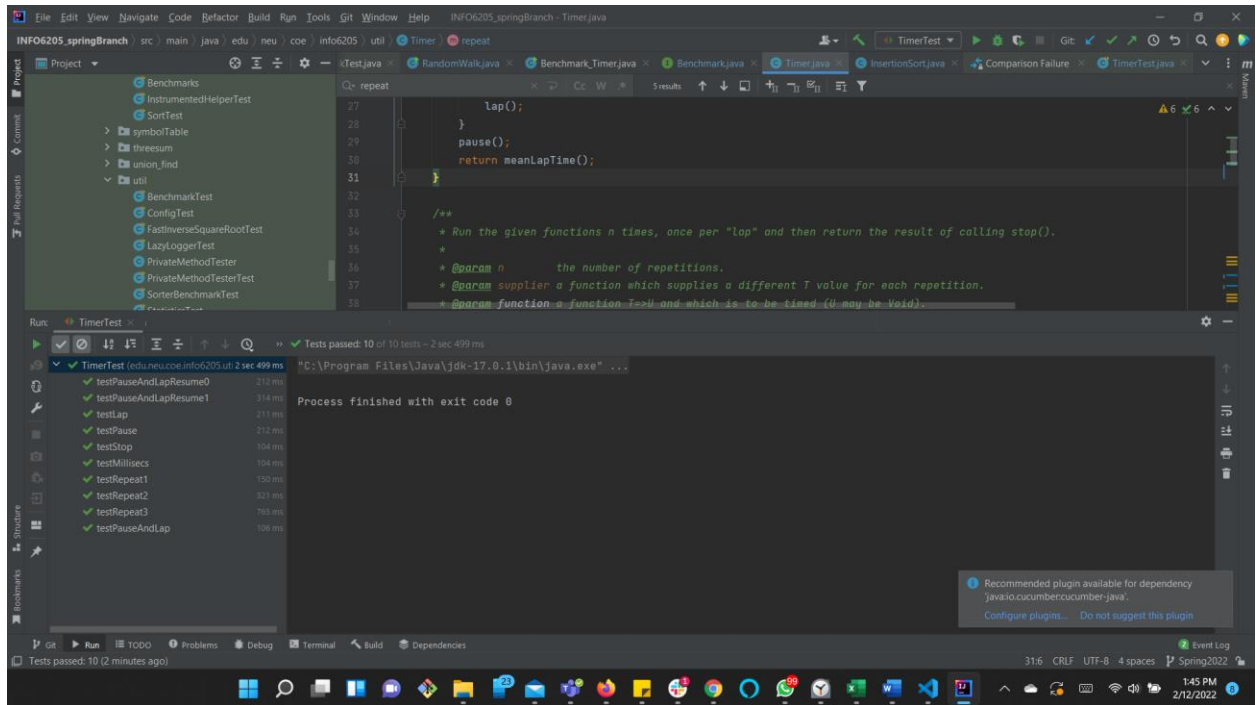
- testWaitPeriods (edu.neu.coe.info6205.util.BenchmarkTest) 1 sec 470 ms
- getWarmupRuns (edu.neu.coe.info6205.util.BenchmarkTest) 0 ms

The console output shows the following messages:

```
"C:\Program Files\Java\jdk-17.0.1\bin\java.exe" ...
2022-02-12 13:45:36 INFO Benchmark_Timer - Begin run: testWaitPeriods with 2 runs
Process finished with exit code 0
```

A notification at the bottom right suggests installing the 'javaio.cucumber.cucumber-java' plugin.

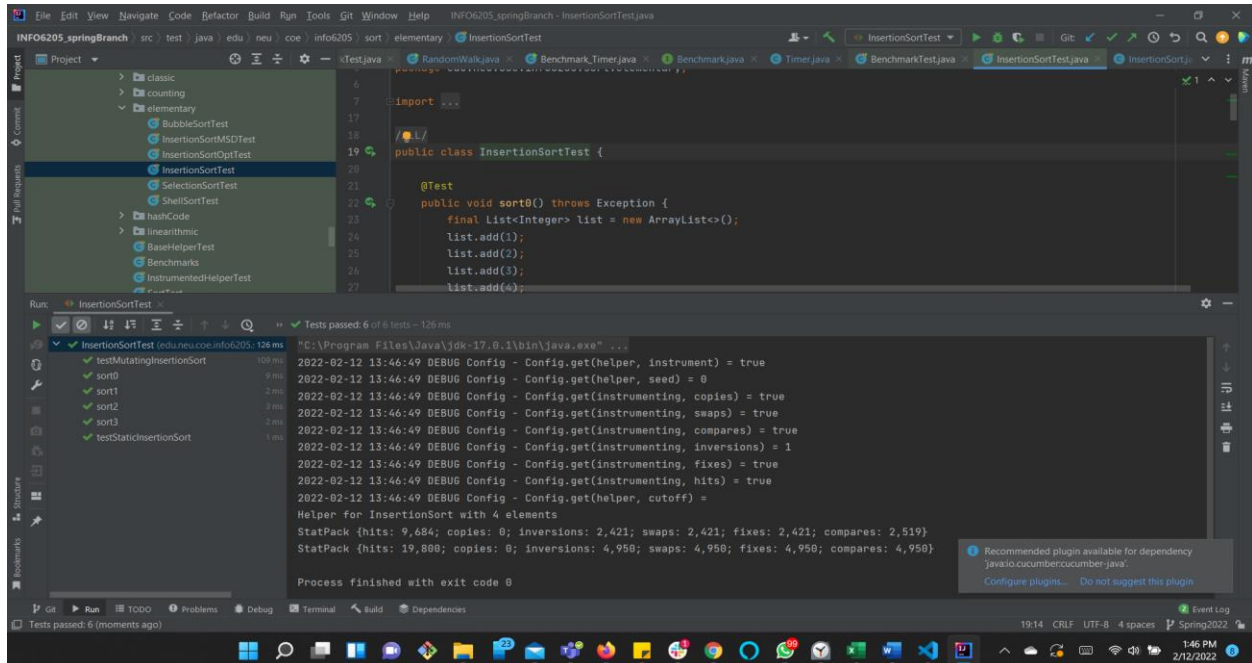
2. TimerTest



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*.

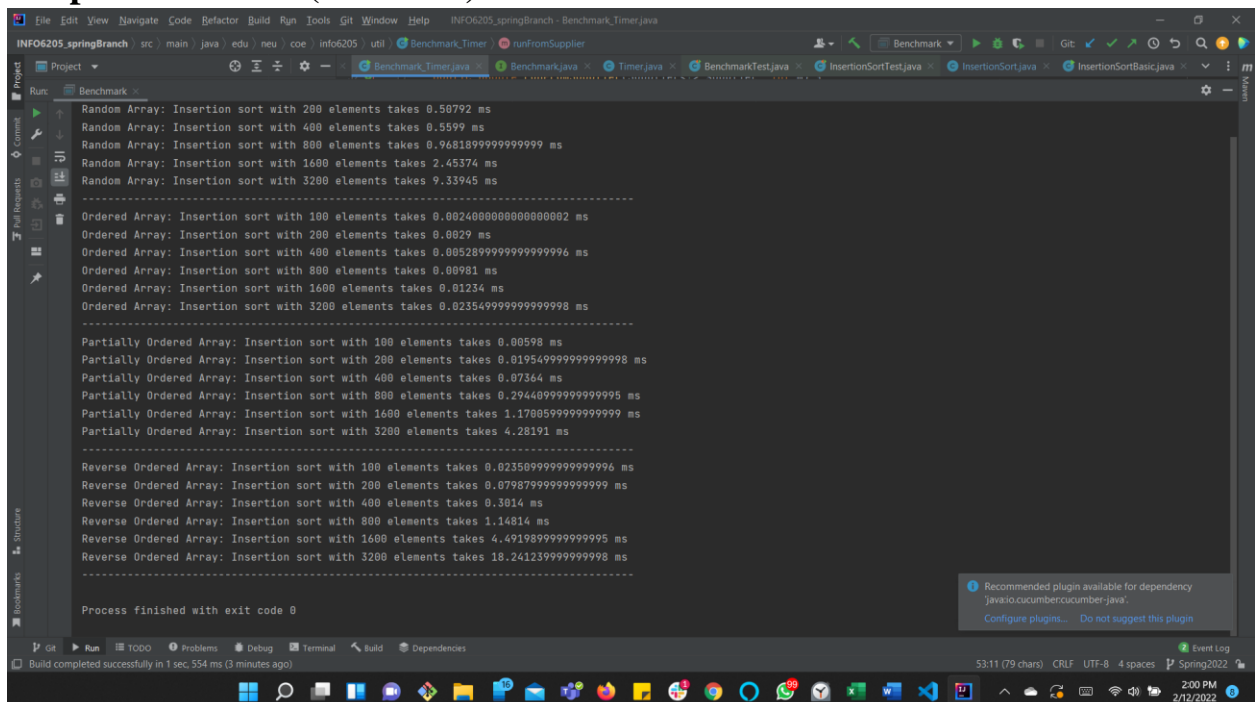
- **Output Screenshot(testcases)**



TASK 3

Implement the 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.

- **Output Screenshot(test cases)**



CONCLUSION

- The ordered array took a minimum time amount of time to sort the array using insertion sort among all types of arrays.
- A partially ordered array took triple time to sort the array using insertion sort when the array of the size was doubled
- Reversed array and random array took maximum time to sort the array using insertion sort among all types of arrays.

Evidence to support the conclusion

Array ordering	100	200	400	800	1600	3200
Random	0.18438	0.50792	0.5599	0.9681899999999999	2.45374	9.33945
Ordered	0.00312	0.00392	0.005630000000000005	0.008749999999999999	0.01416	0.03132
Partially ordered	0.0080699999999999999	0.029830000000000002	0.15209999999999999	0.33199999999999996	1.12668	4.80423
Reversed	0.02555	0.09627	0.35373	1.45217	5.79073	18.17541