Program Structures & Algorithms

Spring 2022

Assignment No. 2

Benchmark

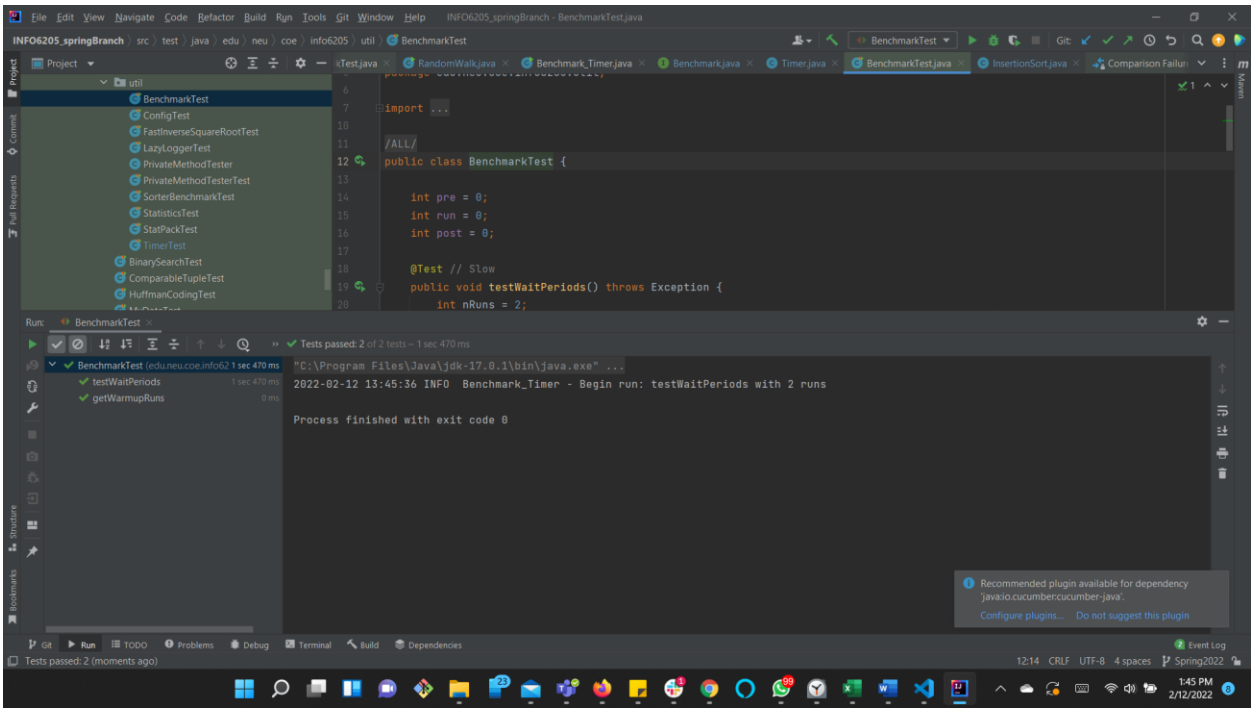
Name: Aishwarya Surve

(NUID): 002123768

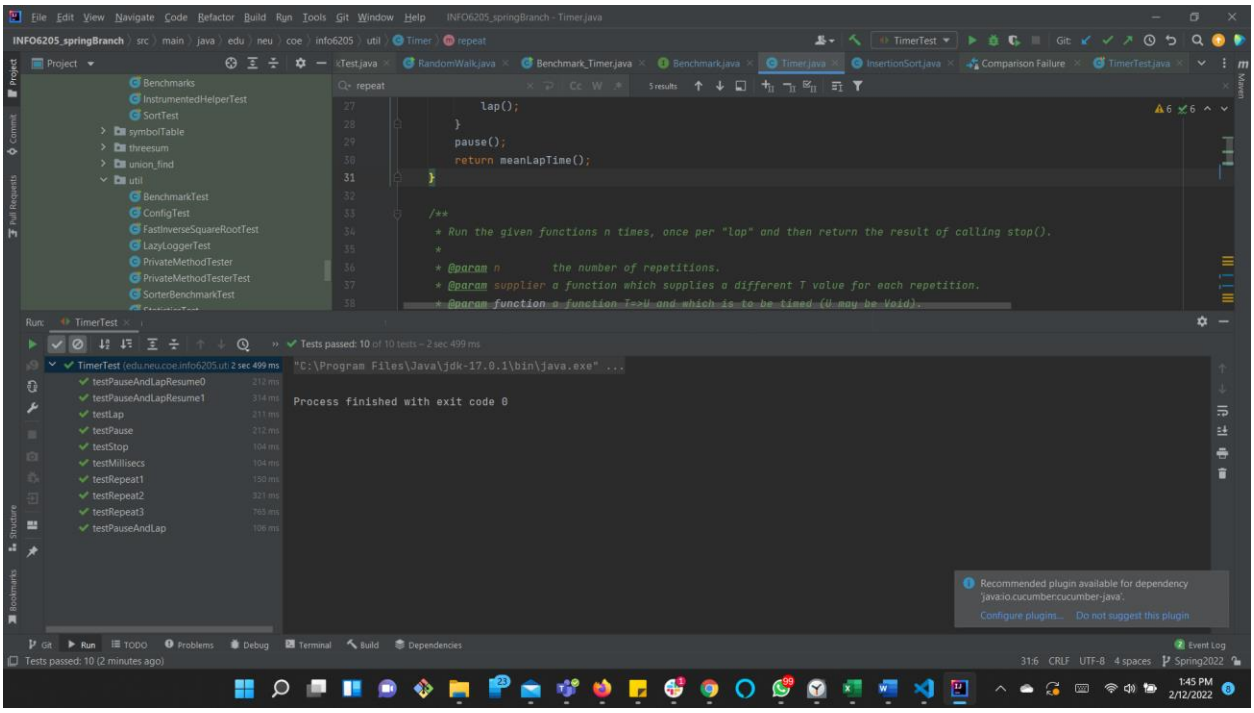
TASK 1

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**



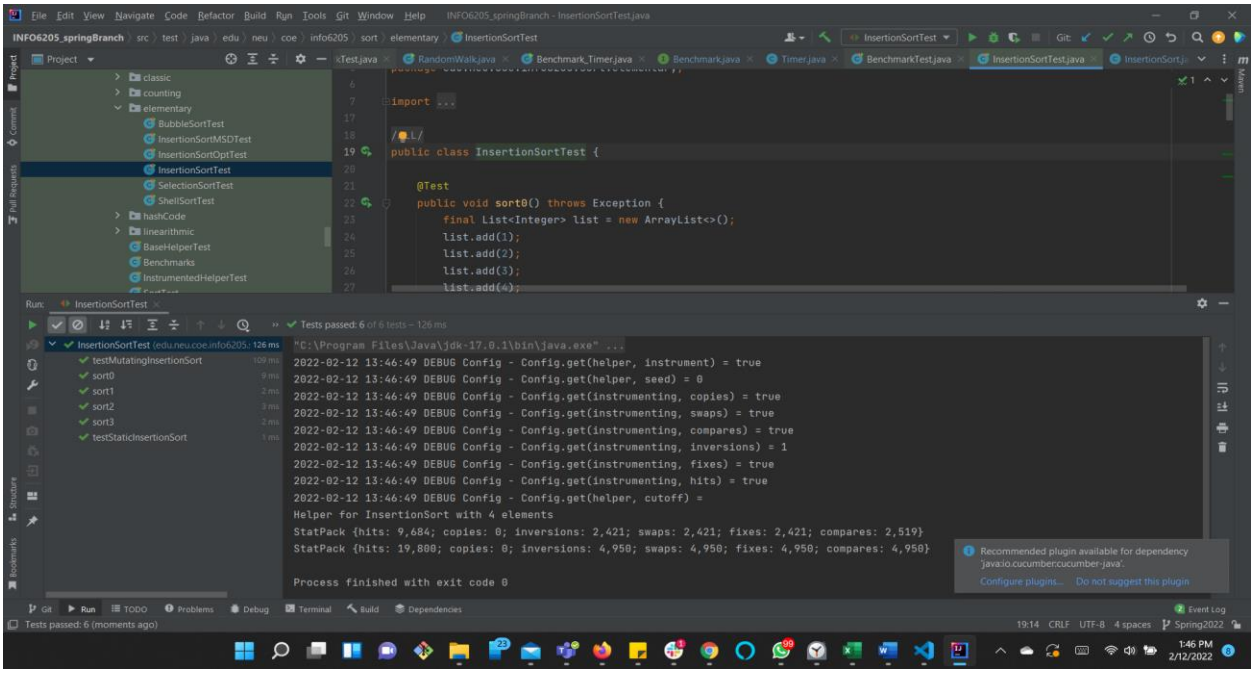
2. **TimerTest**



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*.

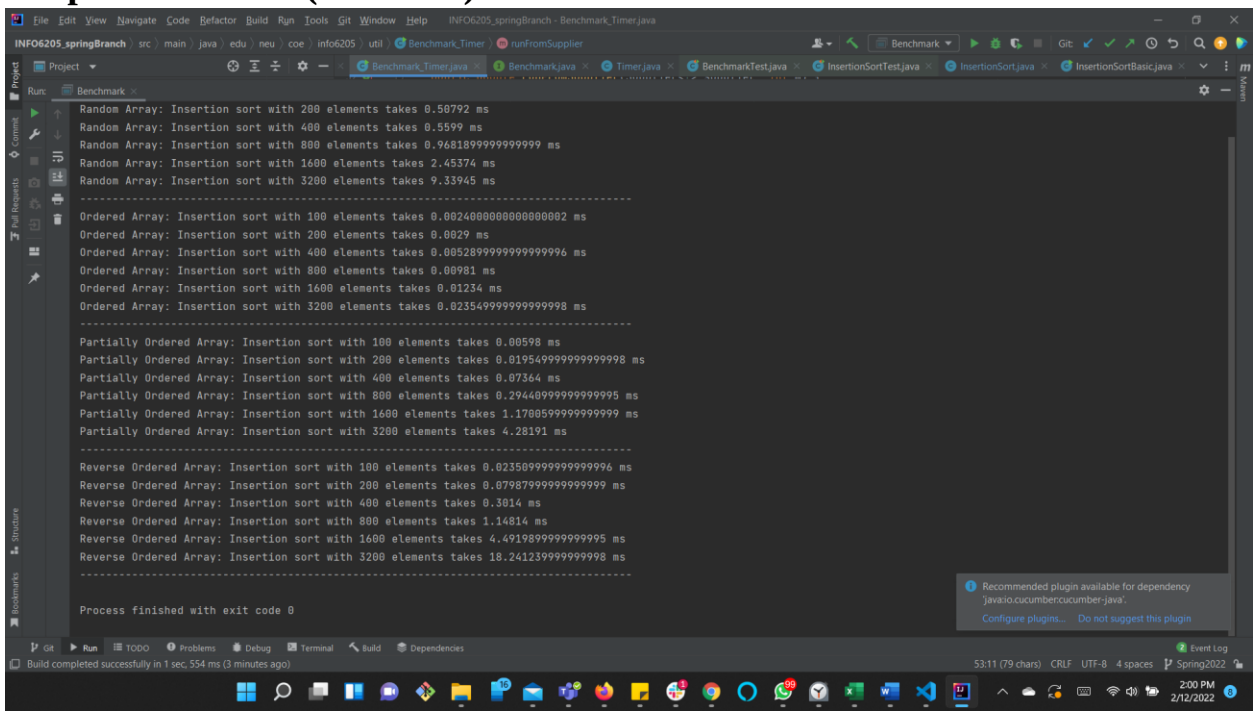
- **Output Screenshot(testcases)**



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	0.0056300000000000005	0.00874999999999999999	0.01416	0.03132
Partially ordered	0.008069999999999999	0.0298300000000000002	0.15209999999999999	0.3319999999999999996	1.12668	4.80423
Reversed	0.02555	0.09627	0.35373	1.45217	5.79073	18.17541