Assignment 2 report

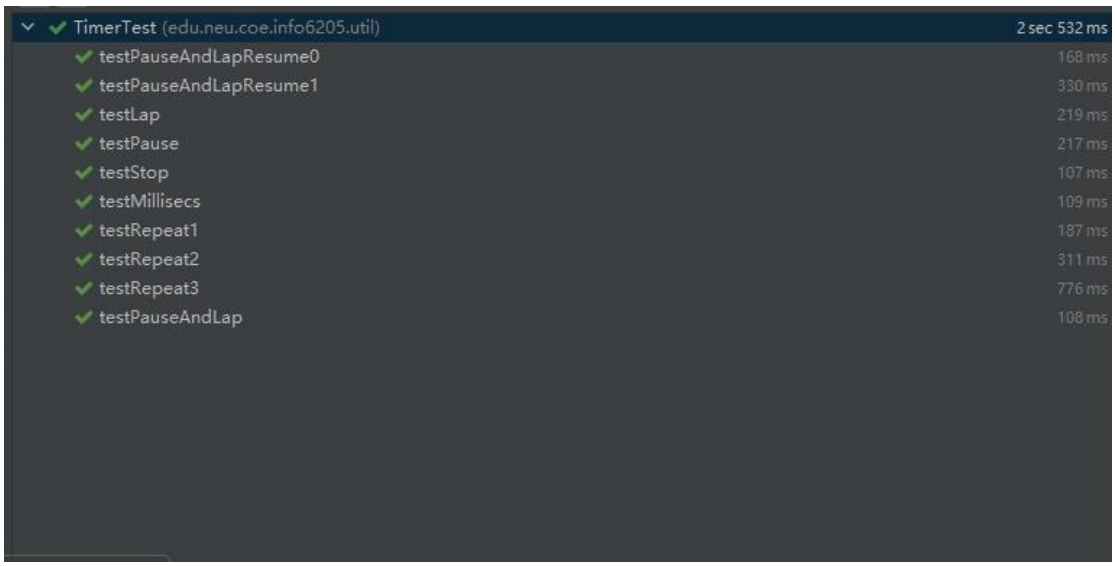
Tasks:

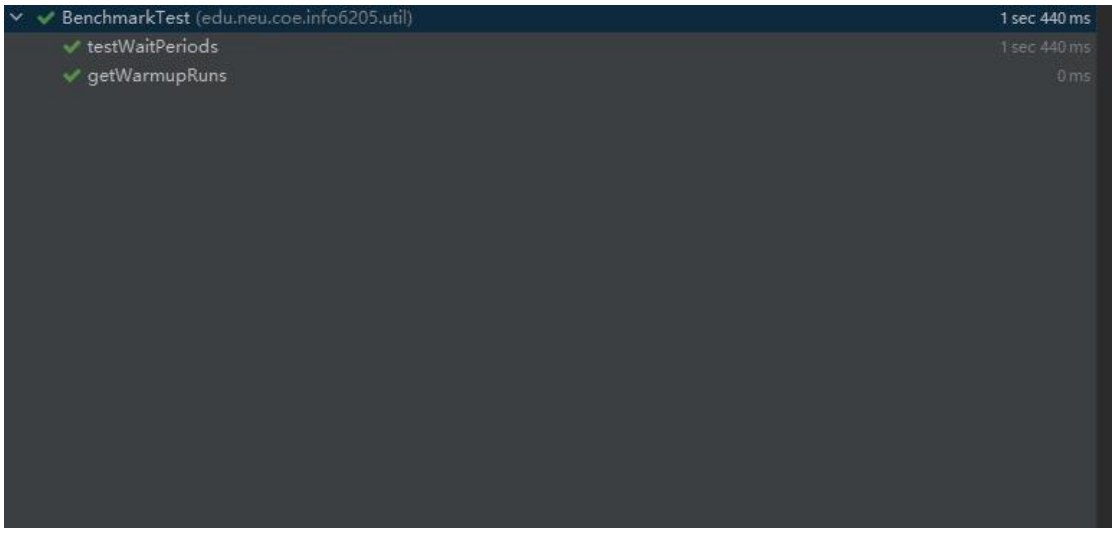
(Part 1) You are to implement three methods of a class called Timer. Please see the skeleton class that I created in the repository. Timer is invoked from a class called Benchmark_Timer which implements the Benchmark interface and pass unit test.

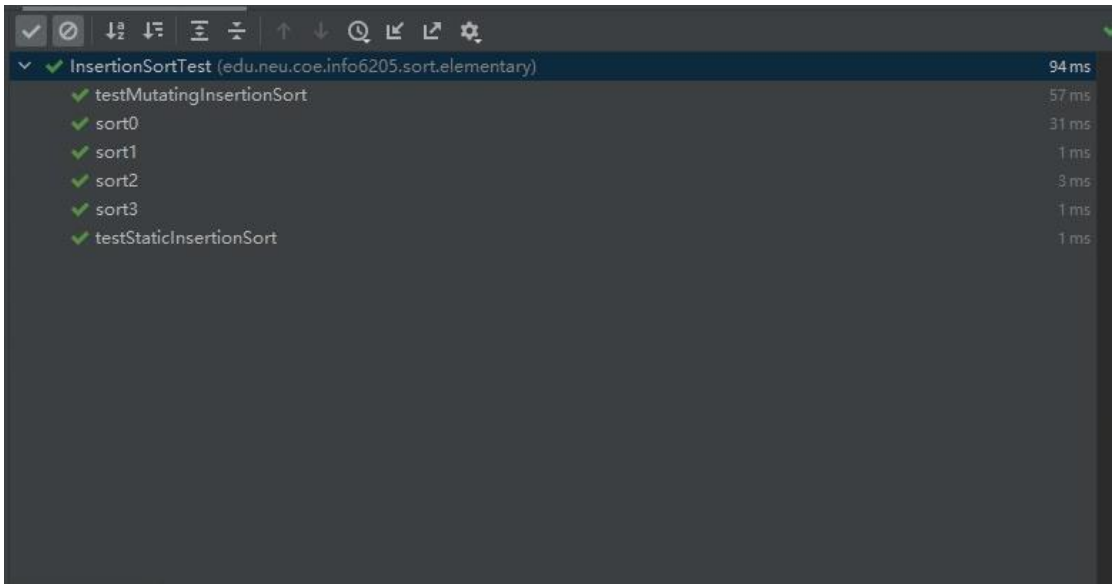
(Part 2) Implement InsertionSort (in the InsertionSort class) by simply looking up the insertion code used by Arrays.sort. If you have the instrument = true setting in test/resources/config.ini, then you will need to use the helper methods for comparing and swapping (so that they properly count the number 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.

Unite Tests:







Experiment result using doubling method:

```
time for sorted array = 0 milliseconds with n = 4000
time for reversed array = 20 milliseconds with n = 4000
time for random array = 10 milliseconds with n = 4000
time for partially random array = 4 milliseconds with n = 4000
```

```
time for sorted array = 0 milliseconds with n = 8000
time for reversed array = 68 milliseconds with n = 8000
time for random array = 35 milliseconds with n = 8000
time for partially random array = 18 milliseconds with n = 8000
```

```
time for sorted array = 1 milliseconds with n = 16000
time for reversed array = 345 milliseconds with n = 16000
time for random array = 108 milliseconds with n = 16000
time for partially random array = 78 milliseconds with n = 16000
```

```
time for sorted array = 1 milliseconds with n = 32000
time for reversed array = 1356 milliseconds with n = 32000
time for random array = 493 milliseconds with n = 32000
time for partially random array = 330 milliseconds with n = 32000
```

```
time for sorted array = 3 milliseconds with n = 64000
time for reversed array = 5686 milliseconds with n = 64000
time for random array = 3441 milliseconds with n = 64000
time for partially random array = 1391 milliseconds with n = 64000
```

Conclusion:

For any given n, the sorted array took least time, partially ordered array took second least time, random array took third least time, reversed array took longest time.

When doubling n, the time for sorted array does not increase significantly, which is close to doubling. Which means for a sorted array, the relationship between time t and n is linear,

For other 3 arrays, when doubling n, it takes 4 times longer to sort, which mean for other 3 arrays, the relationship is quadratic.