Zixiao Wang 001058840

INFO 6205
Program Structures & Algorithms
Fall 2020
Assignment No. 2

The report format follows this document

Task

Assignment 2

- Part 1
- Part 2
- Part 3

Your task for this assignment is in three parts.

Part 1

(Part 1) You are to implement four 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. The APIs of these class are as follows:

```
public interface Benchmark<T> {
    default double run(T t, int m) {
        return runFromSupplier(() -> t, m);
    }

    double runFromSupplier(Supplier<T> supplier, int m);
}
```

```
public class Benchmark_Timer<T> implements Benchmark<T> {
public Benchmark_Timer(String description, UnaryOperator<T> fPre, Consumer<T>
fRun, Consumer<T> fPost)
public Benchmark_Timer(String description, UnaryOperator<T> fPre, Consumer<T>
fRun)
public Benchmark_Timer(String description, Consumer<T> fRun, Consumer<T> fPost)
public Benchmark_Timer(String description, Consumer<T> f)


public class Timer {
... // see below for methods to be implemented...
}



public <T, U> double repeat(int n, Supplier<T> supplier, Function<T, U> function,
UnaryOperator<T> preFunction, Consumer<U> postFunction) {
// TO BE IMPLEMENTED
}


private static long getClock() {
    // TO BE IMPLEMENTED
}


private static double toMillisecs(long ticks) {
    // TO BE IMPLEMENTED
}
```

The function to be timed, hereinafter the "target" function, is the Consumer function fRun (or just f) passed in to one or other of the constructors. For example, you might create a function which sorts an array with n elements.

The generic type T is that of the input to the target function.

The first parameter to the first run method signature is the parameter that will, in turn, be passed to target function. In the second signature, supplier will be invoked each time to get a t which is passed to the other run method.

The second parameter to the run function (m) is the number of times the target function will be called.

The return value from run is the average number of milliseconds taken for each run of the target function.

Don't forget to check your implementation by running the unit tests in BenchmarkTest.

Part 2

(Part 2) Implement InsertionSort (in the InsertionSort class) by simply looking up the insertion code used by Arrays.sort. You should use the helper.swap method although you could also just copy that from the same source code. In the main method of Benchmark, remove the reference to SelectionSort.

Part 3

(Part 3) 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. As usual, the submission will be your entire project (clean, i.e. without the target and project folders). There are stubs and unit tests in the repository.

Report on your observations and show screenshots of the runs and also the unit tests. Please note that you may have to adjust the required execution time for the insertion sort unit test(s) because your computer may not run at the same speed as mine.

Further notes: you should use the System.nanoTime method to get the clock time. This isn't guaranteed to be accurate which is one of the reasons you should run the experiment several times for each value of n. Also, for each invocation of run, run the given target function ten times to get the system "warmed up" before you start the timing properly.

The Sort interface takes care of copying the array when the sort(array) signature is called. It returns a new array as a result. The original array is unchanged. Therefore, you do not need to worry about the insertion-based sorts getting quicker because of the arrays getting more sorted (they don't).

If you need clarification, ask on Slack.

Implementation - Code

`Timer.java`

`InsertionSort.java`

`hw2Part3.java`

`CSVExport.java`

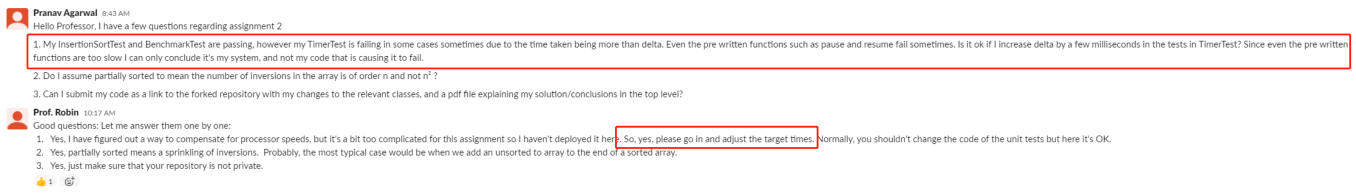
Python Code

For this part of code, you should run it in jupyter notebook! Or you need to modify some part of it

Screenshot of Unit Test Passing

Please note that I have make some adjustment of TEST CASE in order to adapt the running threodhold

According to the professor answer, it is allowed.



Pranav Agarwal 8:43 AM
Hello Professor, I have a few questions regarding assignment 2

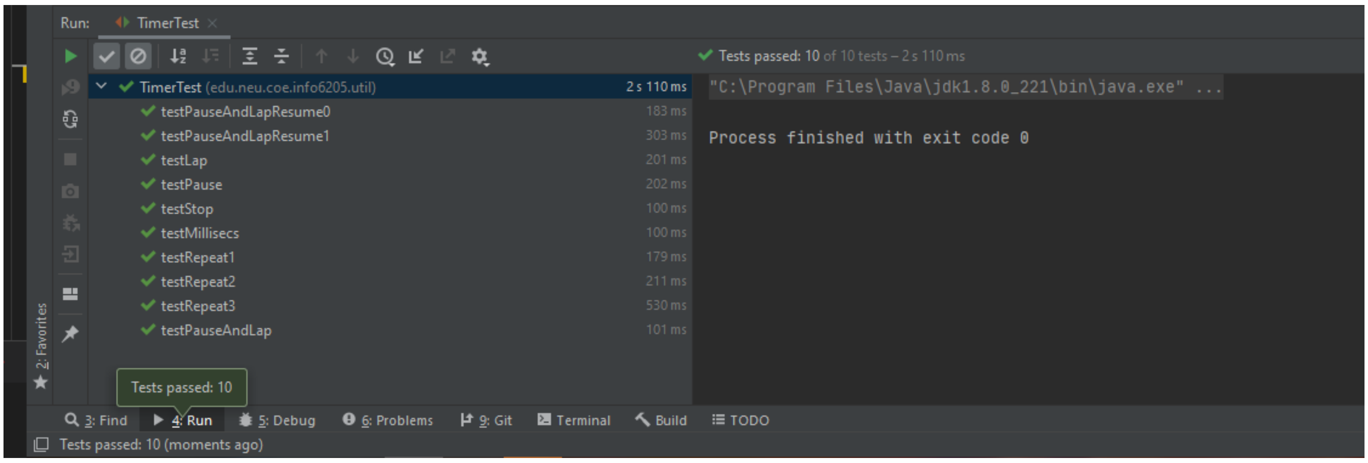
1. My InsertionSortTest and BenchmarkTest are passing, however my TimerTest is failing in some cases sometimes due to the time taken being more than delta. Even the pre written functions such as pause and resume fail sometimes. Is it ok if I increase delta by a few milliseconds in the tests in TimerTest? Since even the pre written functions are too slow I can only conclude it's my system, and not my code that is causing it to fail.

2. Do I assume partially sorted to mean the number of inversions in the array is of order n and not n^2 ?

3. Can I submit my code as a link to the forked repository with my changes to the relevant classes, and a pdf file explaining my solution/conclusions in the top level?

Prof. Robin 10:17 AM
Good questions: Let me answer them one by one:
1. Yes, I have figured out a way to compensate for processor speeds, but it's a bit too complicated for this assignment so I haven't deployed it here. So, yes, please go in and adjust the target times. Normally, you shouldn't change the code of the unit tests but here it's OK.
2. Yes, partially sorted means a sprinkling of inversions. Probably, the most typical case would be when we add an unsorted to array to the end of a sorted array.
3. Yes, just make sure that your repository is not private.

Part I



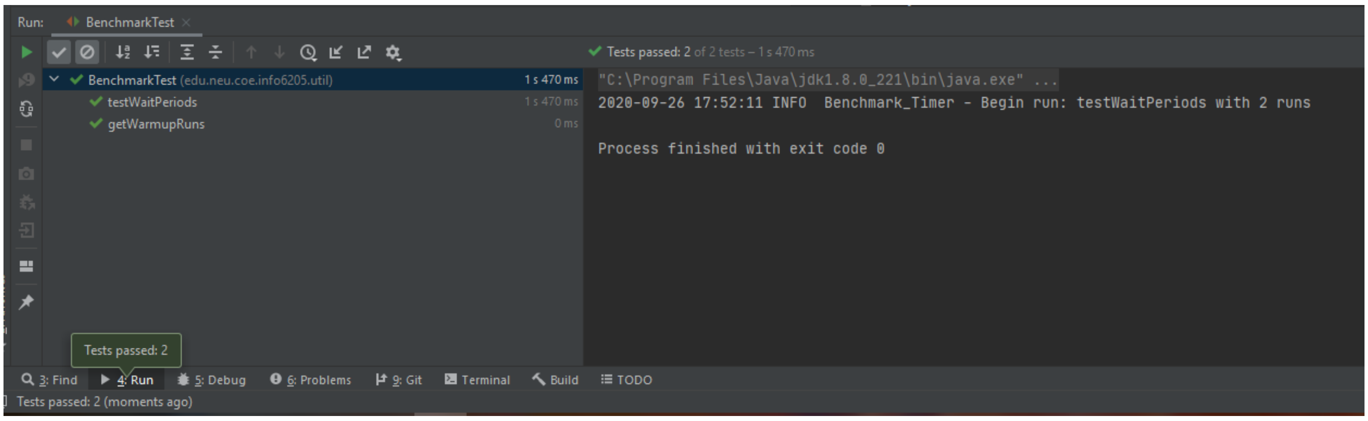
```
TimerTest (edu.neu.coe.info6205.util)   2 s 110 ms
  testPauseAndLapResume0                183 ms
  testPauseAndLapResume1                303 ms
  testLap                               201 ms
  testPause                             202 ms
  testStop                              100 ms
  testMillisecs                         100 ms
  testRepeat1                           179 ms
  testRepeat2                           211 ms
  testRepeat3                           530 ms
  testPauseAndLap                       101 ms
```

Tests passed: 10 of 10 tests – 2 s 110 ms

```
"C:\Program Files\Java\jdk1.8.0_221\bin\java.exe" ...

Process finished with exit code 0
```

Tests passed: 10 (moments ago)

> Timer test result



```
BenchmarkTest (edu.neu.coe.info6205.util)   1 s 470 ms
  testWaitPeriods                           1 s 470 ms
  getWarmupRuns                             0 ms
```

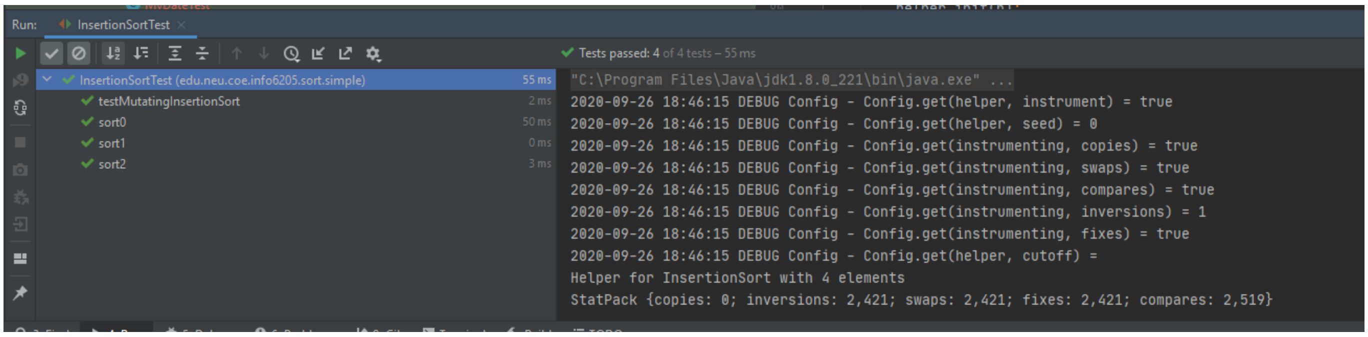
Tests passed: 2 of 2 tests – 1 s 470 ms

```
"C:\Program Files\Java\jdk1.8.0_221\bin\java.exe" ...
2020-09-26 17:52:11 INFO  Benchmark_Timer - Begin run: testWaitPeriods with 2 runs

Process finished with exit code 0
```

Tests passed: 2 (moments ago)

> This is benchTest

Part II



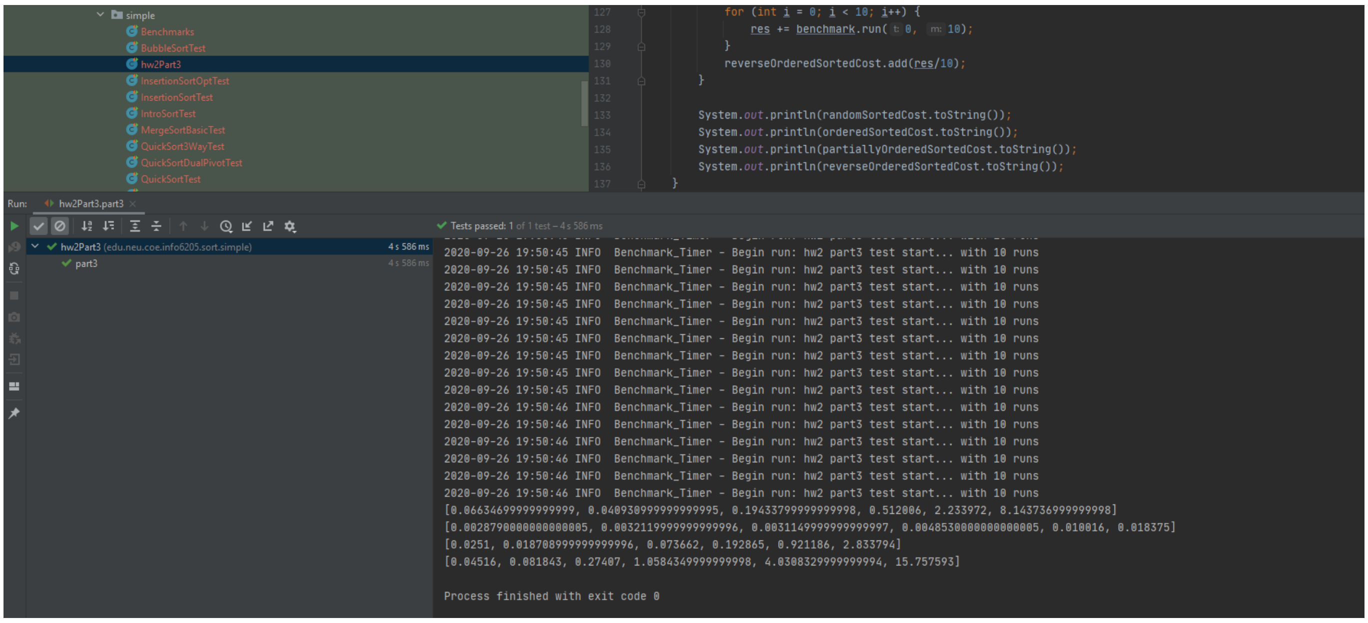
```
InsertionSortTest (edu.neu.coe.info6205.sort.simple)   55 ms
  testMutatingInsertionSort                            2 ms
  sort0                                                50 ms
  sort1                                                0 ms
  sort2                                                3 ms
```

Tests passed: 4 of 4 tests – 55 ms

```
"C:\Program Files\Java\jdk1.8.0_221\bin\java.exe" ...
2020-09-26 18:46:15 DEBUG Config - Config.get(helper, instrument) = true
2020-09-26 18:46:15 DEBUG Config - Config.get(helper, seed) = 0
2020-09-26 18:46:15 DEBUG Config - Config.get(instrumenting, copies) = true
2020-09-26 18:46:15 DEBUG Config - Config.get(instrumenting, swaps) = true
2020-09-26 18:46:15 DEBUG Config - Config.get(instrumenting, compares) = true
2020-09-26 18:46:15 DEBUG Config - Config.get(instrumenting, inversions) = 1
2020-09-26 18:46:15 DEBUG Config - Config.get(instrumenting, fixes) = true
2020-09-26 18:46:15 DEBUG Config - Config.get(helper, cutoff) =
Helper for InsertionSort with 4 elements
StatPack {copies: 0; inversions: 2,421; swaps: 2,421; fixes: 2,421; compares: 2,519}
```

> InsertionSortTest

Part III



```java
                for (int i = 0; i < 10; i++) {
                    res += benchmark.run( t: 0,  m: 10);
                }
                reverseOrderedSortedCost.add(res/10);
            }

            System.out.println(randomSortedCost.toString());
            System.out.println(orderedSortedCost.toString());
            System.out.println(partiallyOrderedSortedCost.toString());
            System.out.println(reverseOrderedSortedCost.toString());
        }
```

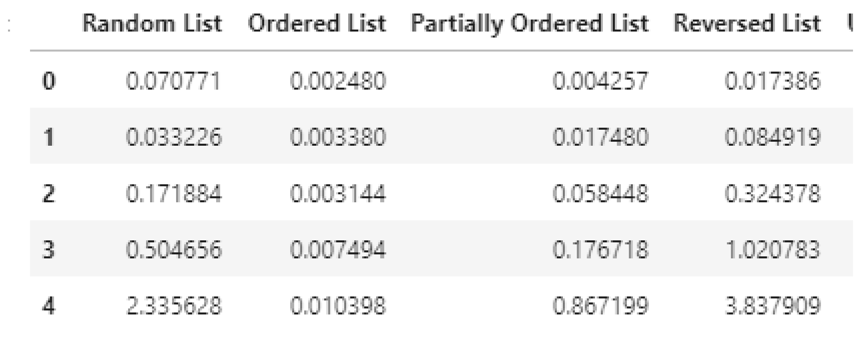
```
2020-09-26 19:50:45 INFO  Benchmark_Timer - Begin run: hw2 part3 test start... with 10 runs
2020-09-26 19:50:45 INFO  Benchmark_Timer - Begin run: hw2 part3 test start... with 10 runs
2020-09-26 19:50:45 INFO  Benchmark_Timer - Begin run: hw2 part3 test start... with 10 runs
2020-09-26 19:50:45 INFO  Benchmark_Timer - Begin run: hw2 part3 test start... with 10 runs
2020-09-26 19:50:45 INFO  Benchmark_Timer - Begin run: hw2 part3 test start... with 10 runs
2020-09-26 19:50:45 INFO  Benchmark_Timer - Begin run: hw2 part3 test start... with 10 runs
2020-09-26 19:50:45 INFO  Benchmark_Timer - Begin run: hw2 part3 test start... with 10 runs
2020-09-26 19:50:45 INFO  Benchmark_Timer - Begin run: hw2 part3 test start... with 10 runs
2020-09-26 19:50:45 INFO  Benchmark_Timer - Begin run: hw2 part3 test start... with 10 runs
2020-09-26 19:50:46 INFO  Benchmark_Timer - Begin run: hw2 part3 test start... with 10 runs
2020-09-26 19:50:46 INFO  Benchmark_Timer - Begin run: hw2 part3 test start... with 10 runs
2020-09-26 19:50:46 INFO  Benchmark_Timer - Begin run: hw2 part3 test start... with 10 runs
2020-09-26 19:50:46 INFO  Benchmark_Timer - Begin run: hw2 part3 test start... with 10 runs
2020-09-26 19:50:46 INFO  Benchmark_Timer - Begin run: hw2 part3 test start... with 10 runs
2020-09-26 19:50:46 INFO  Benchmark_Timer - Begin run: hw2 part3 test start... with 10 runs
[0.06634699999999999, 0.040930999999999995, 0.19433799999999998, 0.512006, 2.233972, 8.143736999999998]
[0.0028790000000000005, 0.0032119999999999996, 0.0031149999999999997, 0.0048530000000000005, 0.010016, 0.018375]
[0.0251, 0.018708999999999996, 0.073662, 0.192865, 0.921186, 2.833794]
[0.04516, 0.081843, 0.27407, 1.0584349999999998, 4.0308329999999994, 15.757593]

Process finished with exit code 0
```

> hw2Part3.java result

So here, I used `hw2Part3.java` for the part III

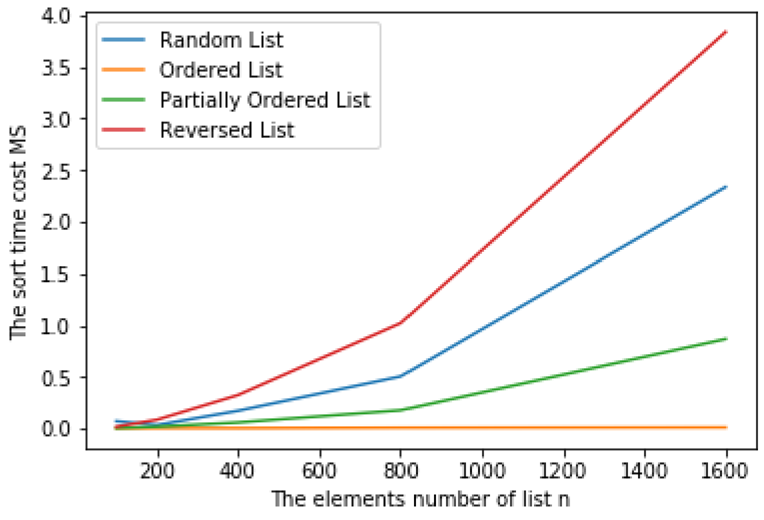
First, measure the running times of this sort, using four different initial array ordering situations: random, ordered, partially-ordered and reverse-ordered.

The n started from 100 and mutliply 2 for each time. This precess continue 5 times. Then I print the result following the random, ordered, partially-ordered and reverse-ordered one by one.



	Random List	Ordered List	Partially Ordered List	Reversed List
0	0.070771	0.002480	0.004257	0.017386
1	0.033226	0.003380	0.017480	0.084919
2	0.171884	0.003144	0.058448	0.324378
3	0.504656	0.007494	0.176718	1.020783
4	2.335628	0.010398	0.867199	3.837909

> Time cost recorded by program. For each row, the n times 2 (n start with 100, which indicate that the array has n elements)



1. We can easily find the the reverse order list spend the most time, which is $O(N^2)$
2. Random list has normal performance
3. Partially ordered list takes less time than random list.
4. Ordered list almost don't cost time for sorting.