Northeastern University

PROGRAM STRUCTURES & ALGORITHMS
INFO – 6205

BENCHMARKING

Assignment 3

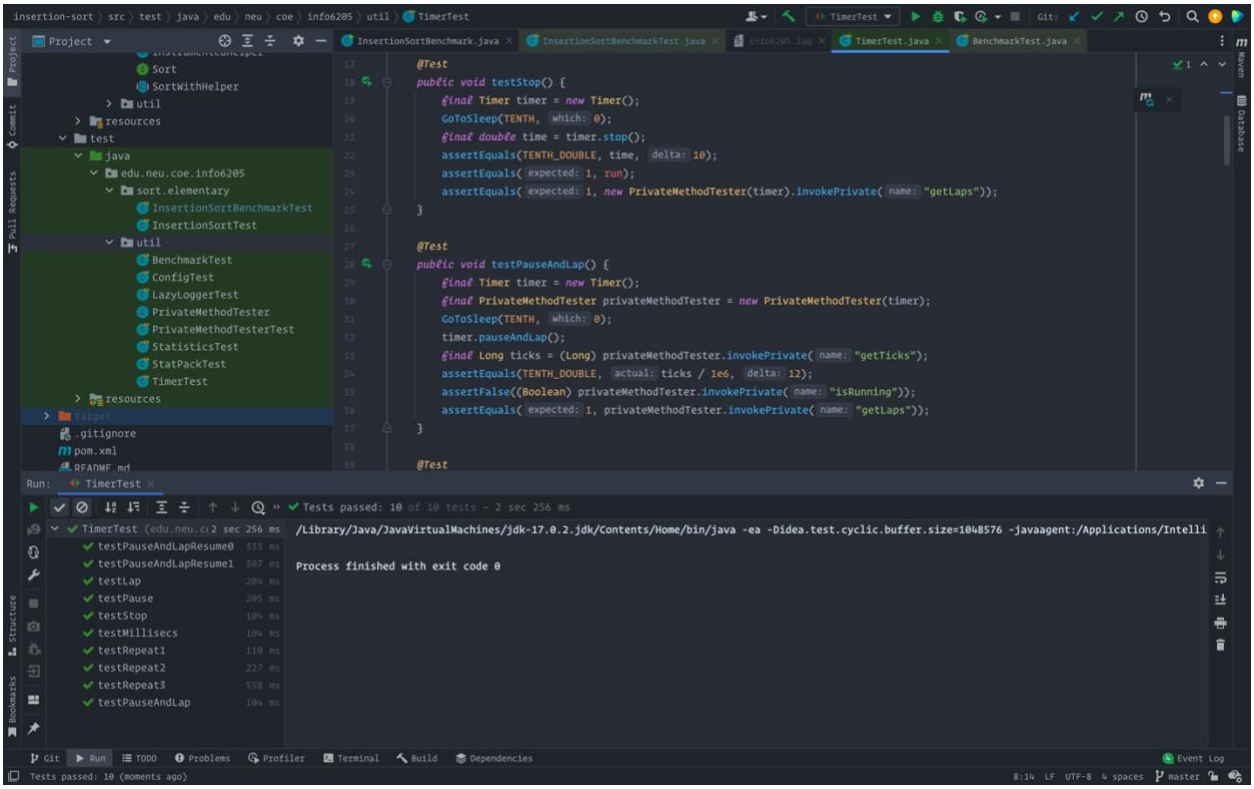
SIDDHARTH RAWAT
002963295

Table of Contents

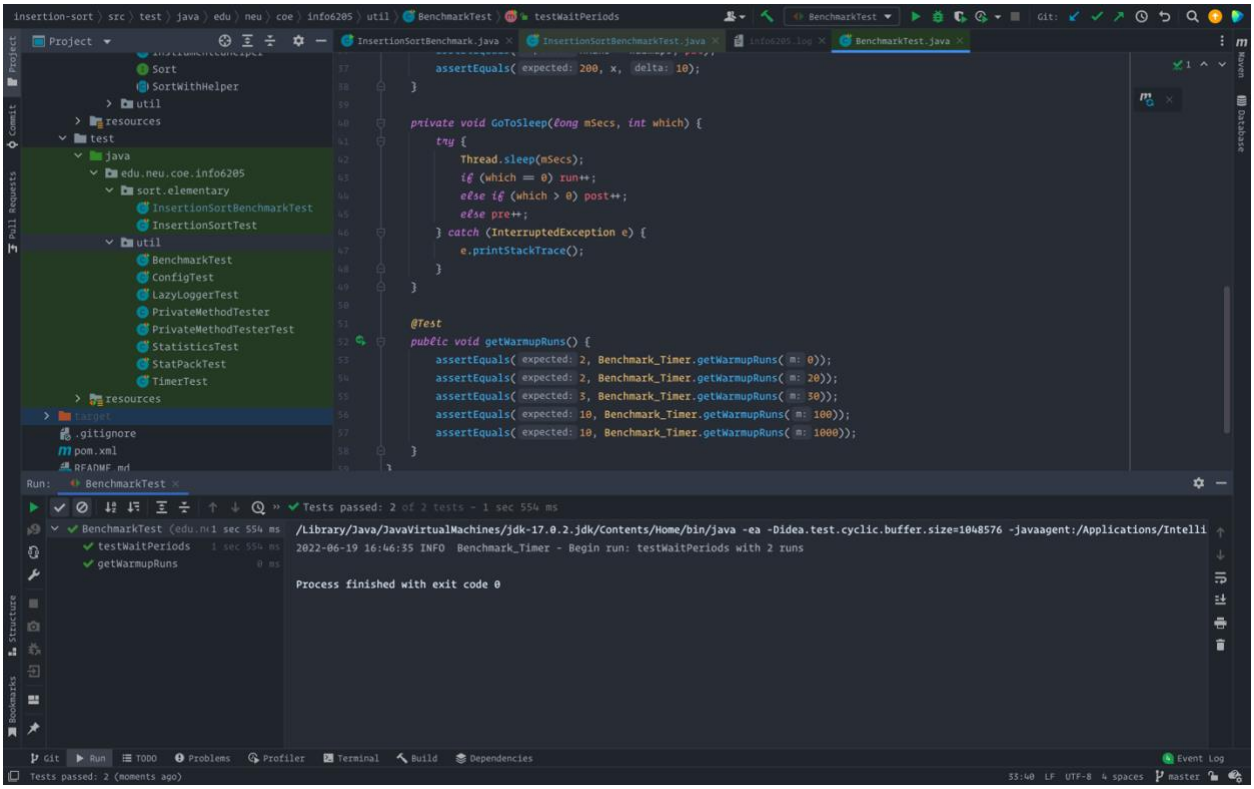
1. Unit Tests

Below is the screenshot of all the unit tests that have passed successfully.

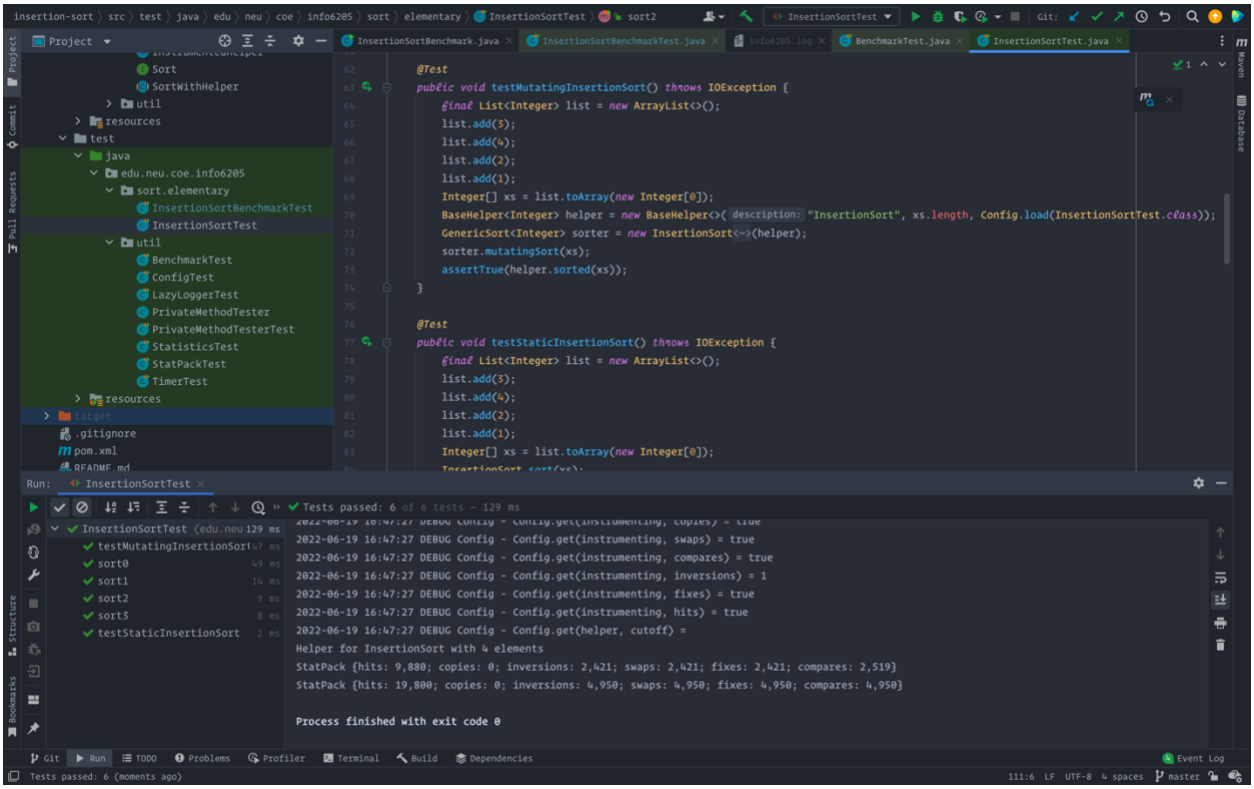
- Timer Test Cases:



- Benchmark Test Cases:

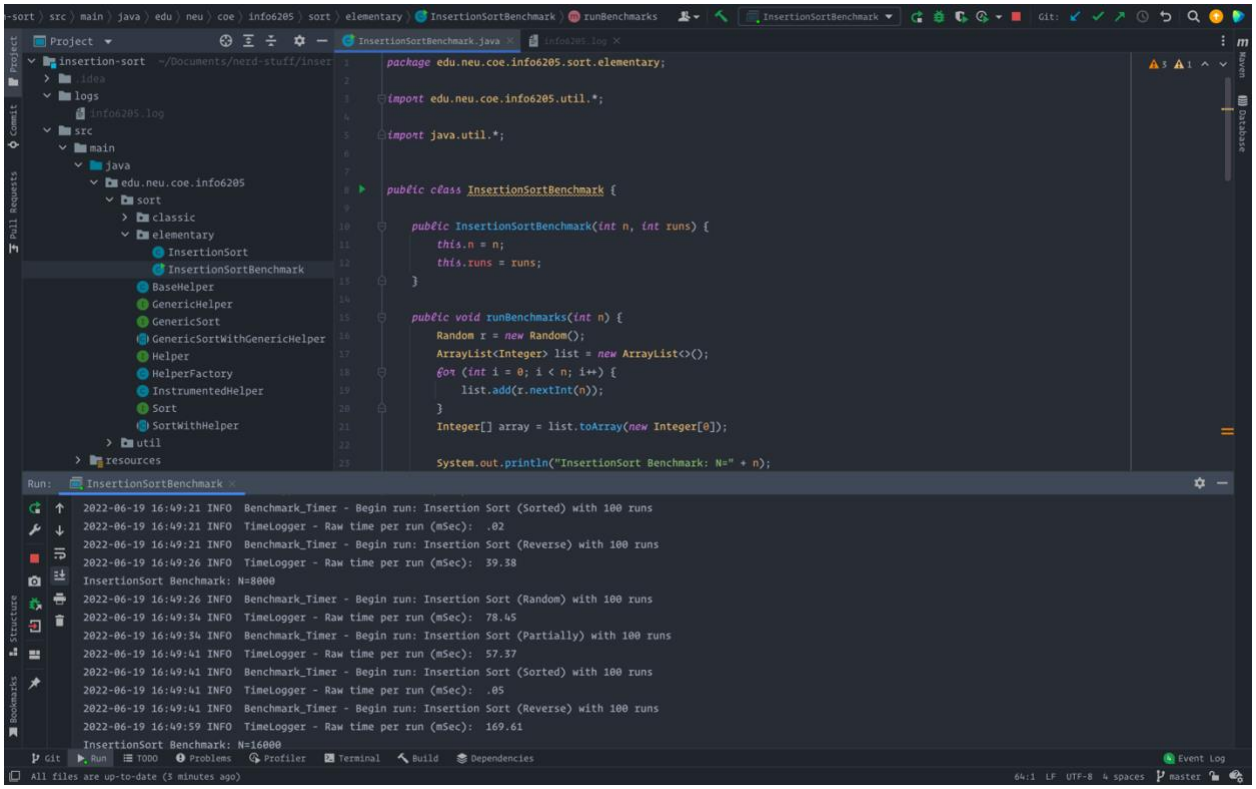


- Insertion Sort Test Cases:



2. Output

The below terminal output shows the result of the benchmarking that has been performed on the Insertion Sort algorithm for arrays that are sorted, partially sorted, randomly sorted and sorted in reverse.



Below is the output that shows the same:

```
InsertionSort Benchmark: N=250
2022-06-18 17:53:10 INFO  Benchmark_Timer - Begin run: Insertion Sort
(Random) with 100 runs
2022-06-18 17:53:10 INFO  TimeLogger - Raw time per run (mSec):  .22
2022-06-18 17:53:10 INFO  Benchmark_Timer - Begin run: Insertion Sort
(Partially) with 100 runs
2022-06-18 17:53:10 INFO  TimeLogger - Raw time per run (mSec):  .12
2022-06-18 17:53:10 INFO  Benchmark_Timer - Begin run: Insertion Sort
(Sorted) with 100 runs
2022-06-18 17:53:10 INFO  TimeLogger - Raw time per run (mSec):  .00
2022-06-18 17:53:10 INFO  Benchmark_Timer - Begin run: Insertion Sort
(Reverse) with 100 runs
2022-06-18 17:53:10 INFO  TimeLogger - Raw time per run (mSec):  .28
InsertionSort Benchmark: N=500
2022-06-18 17:53:10 INFO  Benchmark_Timer - Begin run: Insertion Sort
(Random) with 100 runs
```

```
2022-06-18 17:53:10 INFO  TimeLogger - Raw time per run (mSec):  .64
2022-06-18 17:53:10 INFO  Benchmark_Timer - Begin run: Insertion Sort
(Partially) with 100 runs
2022-06-18 17:53:10 INFO  TimeLogger - Raw time per run (mSec):  .49
2022-06-18 17:53:10 INFO  Benchmark_Timer - Begin run: Insertion Sort
(Sorted) with 100 runs
2022-06-18 17:53:10 INFO  TimeLogger - Raw time per run (mSec):  .00
2022-06-18 17:53:10 INFO  Benchmark_Timer - Begin run: Insertion Sort
(Reverse) with 100 runs
2022-06-18 17:53:10 INFO  TimeLogger - Raw time per run (mSec):  .76
InsertionSort Benchmark: N=1000
2022-06-18 17:53:10 INFO  Benchmark_Timer - Begin run: Insertion Sort
(Random) with 100 runs
2022-06-18 17:53:10 INFO  TimeLogger - Raw time per run (mSec):  1.25
2022-06-18 17:53:10 INFO  Benchmark_Timer - Begin run: Insertion Sort
(Partially) with 100 runs
2022-06-18 17:53:10 INFO  TimeLogger - Raw time per run (mSec):  .92
2022-06-18 17:53:10 INFO  Benchmark_Timer - Begin run: Insertion Sort
(Sorted) with 100 runs
2022-06-18 17:53:10 INFO  TimeLogger - Raw time per run (mSec):  .00
2022-06-18 17:53:10 INFO  Benchmark_Timer - Begin run: Insertion Sort
(Reverse) with 100 runs
2022-06-18 17:53:10 INFO  TimeLogger - Raw time per run (mSec):  3.16
InsertionSort Benchmark: N=2000
2022-06-18 17:53:10 INFO  Benchmark_Timer - Begin run: Insertion Sort
(Random) with 100 runs
2022-06-18 17:53:11 INFO  TimeLogger - Raw time per run (mSec):  5.93
2022-06-18 17:53:11 INFO  Benchmark_Timer - Begin run: Insertion Sort
(Partially) with 100 runs
2022-06-18 17:53:12 INFO  TimeLogger - Raw time per run (mSec):  3.44
2022-06-18 17:53:12 INFO  Benchmark_Timer - Begin run: Insertion Sort
(Sorted) with 100 runs
2022-06-18 17:53:12 INFO  TimeLogger - Raw time per run (mSec):  .01
2022-06-18 17:53:12 INFO  Benchmark_Timer - Begin run: Insertion Sort
(Reverse) with 100 runs
2022-06-18 17:53:13 INFO  TimeLogger - Raw time per run (mSec):  9.14
InsertionSort Benchmark: N=4000
2022-06-18 17:53:13 INFO  Benchmark_Timer - Begin run: Insertion Sort
(Random) with 100 runs
2022-06-18 17:53:14 INFO  TimeLogger - Raw time per run (mSec):  17.75
2022-06-18 17:53:14 INFO  Benchmark_Timer - Begin run: Insertion Sort
(Partially) with 100 runs
2022-06-18 17:53:16 INFO  TimeLogger - Raw time per run (mSec):  14.27
2022-06-18 17:53:16 INFO  Benchmark_Timer - Begin run: Insertion Sort
(Sorted) with 100 runs
2022-06-18 17:53:16 INFO  TimeLogger - Raw time per run (mSec):  .01
2022-06-18 17:53:16 INFO  Benchmark_Timer - Begin run: Insertion Sort
(Reverse) with 100 runs
2022-06-18 17:53:20 INFO  TimeLogger - Raw time per run (mSec):  37.28
```

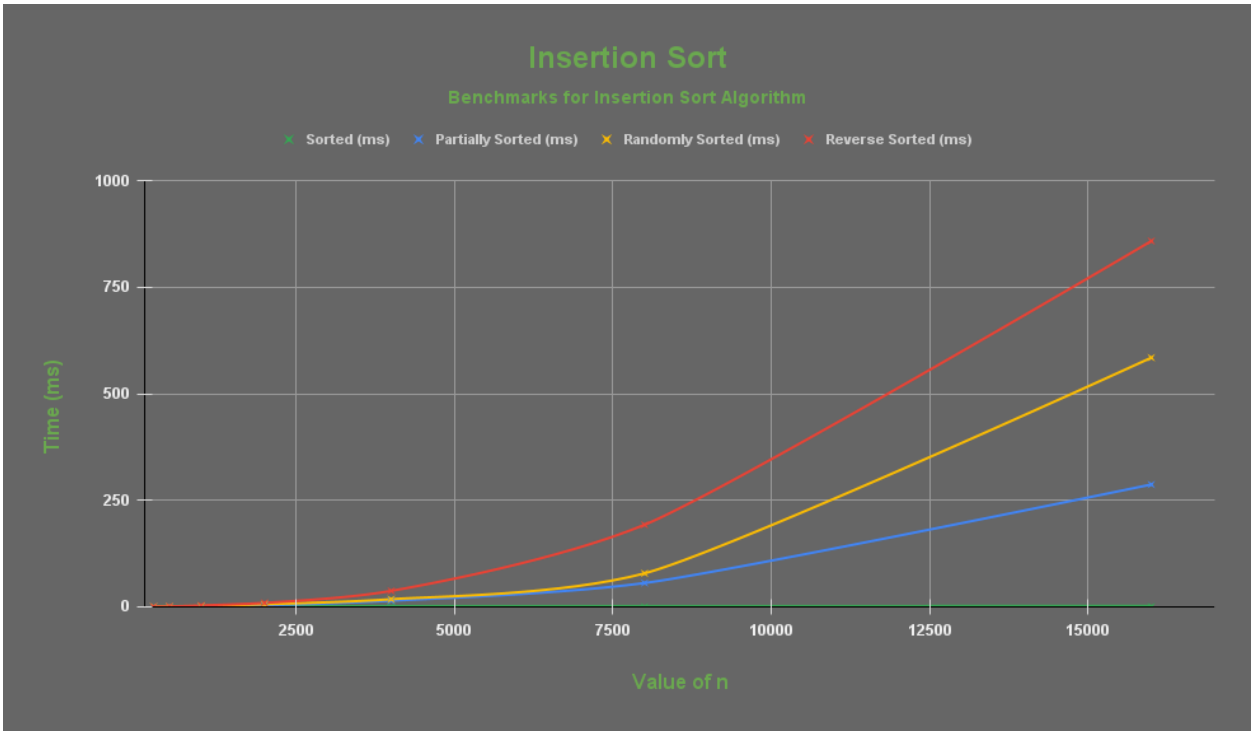
```
InsertionSort Benchmark: N=8000
2022-06-18 17:53:20 INFO  Benchmark_Timer - Begin run: Insertion Sort
(Random) with 100 runs
2022-06-18 17:53:29 INFO  TimeLogger - Raw time per run (mSec):  78.50
2022-06-18 17:53:29 INFO  Benchmark_Timer - Begin run: Insertion Sort
(Partially) with 100 runs
2022-06-18 17:53:35 INFO  TimeLogger - Raw time per run (mSec):  55.76
2022-06-18 17:53:35 INFO  Benchmark_Timer - Begin run: Insertion Sort
(Sorted) with 100 runs
2022-06-18 17:53:35 INFO  TimeLogger - Raw time per run (mSec):  .05
2022-06-18 17:53:35 INFO  Benchmark_Timer - Begin run: Insertion Sort
(Reverse) with 100 runs
2022-06-18 17:53:56 INFO  TimeLogger - Raw time per run (mSec):  192.52
InsertionSort Benchmark: N=16000
2022-06-18 17:53:56 INFO  Benchmark_Timer - Begin run: Insertion Sort
(Random) with 100 runs
2022-06-18 17:54:59 INFO  TimeLogger - Raw time per run (mSec):  584.80
2022-06-18 17:54:59 INFO  Benchmark_Timer - Begin run: Insertion Sort
(Partially) with 100 runs
2022-06-18 17:55:31 INFO  TimeLogger - Raw time per run (mSec):  287.05
2022-06-18 17:55:31 INFO  Benchmark_Timer - Begin run: Insertion Sort
(Sorted) with 100 runs
2022-06-18 17:55:31 INFO  TimeLogger - Raw time per run (mSec):  .45
2022-06-18 17:55:31 INFO  Benchmark_Timer - Begin run: Insertion Sort
(Reverse) with 100 runs
2022-06-18 17:57:05 INFO  TimeLogger - Raw time per run (mSec):  858.66

Process finished with exit code 0
```

3. Observations

Below is the table that displays the timing observations in milliseconds for the arrays that are **sorted, partially sorted, randomly sorted and sorted in reverse** for the Insertion Sort algorithm. The tests are performed using the **doubling method** and for **100 runs**.

Value of n	Sorted (ms)	Partially Sorted (ms)	Randomly Sorted (ms)	Reverse Sorted (ms)
250	0	0.12	0.22	0.28
500	0	0.49	0.64	0.76
1000	0	0.92	1.25	3.16
2000	0.01	3.44	5.93	9.14
4000	0.01	14.27	17.75	37.28
8000	0.05	55.76	78.5	192.52
16000	0.45	287.05	584.8	858.66

Below is the graph of the **raw timing observations** from the above table.



The insertion sort algorithm is both **stable** and **adaptive**. The average number of **comparisons** required are $\frac{1}{4}$ **N(N-1)** and the average number of **swaps** required are $\frac{1}{4}$ **(N (N-1)** as well. So as the **number of swaps is none for the sorted array**, it takes the **least amount of time** to run the insertion sort algorithm on it. In contrast, an array that is **sorted in reverse order, will take the most amount of time** since all the numbers are to be swapped.

This can be related to the output table above. As seen from the benchmarking output in the observation table, the array sorted in reverse takes the most amount of time to run the insertion sort algorithm, the randomly sorted array runs faster than the reverse sorted array, the partially sorted array runs even faster than the randomly sorted array, and the sorted array runs the fastest with insertion sort.

An interactive version of this graph is **available here**.

In conclusion, the following can be deduced about the run time of insertion sort for different order situations in arrays:

Ordered < Partially Ordered < Randomly Ordered < Reverse Ordered

4. Code

The code for this assignment is available on **my GitHub repository**. The excel sheet containing the graph and timing observations can be found **here**.