Program Structures & Algorithms
Spring 2023
Assignment No. 3

Name: Anurag Nandre
(NUID): 002950342

Task

- (Part 1) Implementation of three methods in *Timer.java,* & check this implementation by running the unit tests in *BenchmarkTest.java* and *TimerTest.java*
- (Part 2) implementation of *InsertionSort* (in the *InsertionSort* class) & check this implementation by running the unit tests in *InsertionSortTest*
- (Part 3) Implementation of a main program to 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.
- Using doubling method for choosing *n* and test for at least five values of *n*
- Drawing conclusions from the observations regarding the order of growth

Relationship Conclusion

- Order of growth of the running time of Insertion Sort (Randomly ordered array of size *N*) is $\approx N^{1.758}$
- Order of growth of the running time of Insertion Sort (Ordered array of size *N*) is $\approx N^{0.81}$
- Order of growth of the running time of Insertion Sort (Partially ordered array of size *N*) is $\approx N^{1.26}$
- Order of growth of the running time of Insertion Sort (Reverse ordered array of size *N*) is $\approx N^{1.86}$
- In terms of order of growth, for the running time of Insertion sort:

$$Ordered < Partially\ Ordered < Randomly\ Ordered < Reverse\ Ordered$$

Evidence to the Conclusion

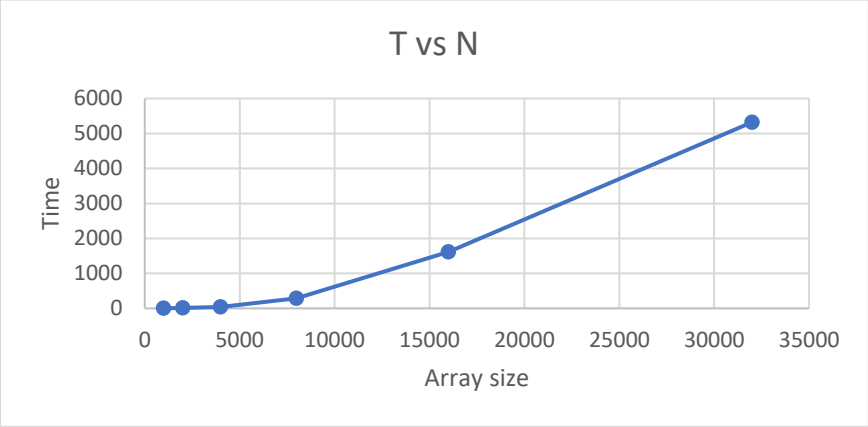
- Running time of the insertion sort for an array of 'n' numbers has been captured
- Each time the size of the array would be doubled and running time would be captured again (5 different sizes of array)
- Every time, we run the insertion sort algorithm, we make sure to test on four different states of the array (Ordered, Partially Ordered, Randomly Ordered, Reverse Ordered)

Random Ordered Array

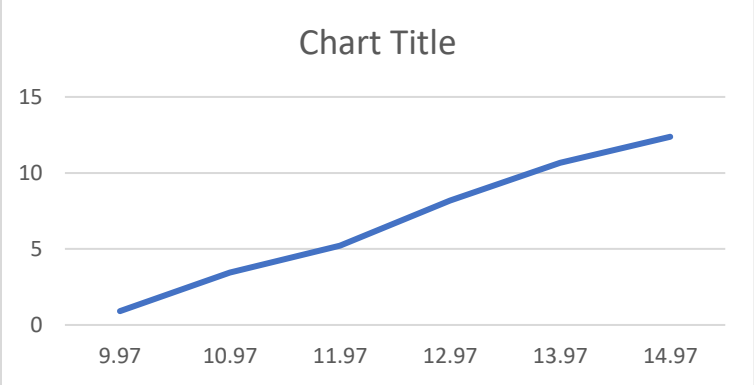
Various sizes of the Array and the running time of the Insertion sort

Randomly Ordered Array						
Array Size	Time	Ratio (Time/Previous Time)	lg(Array size)	lg(Time)	Log Ratio	Slope
1000	1.79	-	9.97	0.84	11.87	
2000	5.446	3.04	10.97	2.45	4.48	2.92
4000	25.66	4.71	11.97	4.68	2.56	1.91
8000	74.4	2.9	12.97	6.22	2.09	1.33
16000	309.73	4.16	13.97	8.27	1.69	1.33
32000	1670.72	5.39	14.97	10.71	1.4	1.3
					Avg Slope	1.758

Analysis of experimental data (the running time of insertion sort with random ordered input)



Standard Plot: Running time T(n) Vs Array size N



Log-Log Plot: lg(T(n)) Vs lg(N)

The equation of the log-log plot is

$$\lg(T(N)) = 1.758 \lg N + \lg a$$

Which is equivalent to,

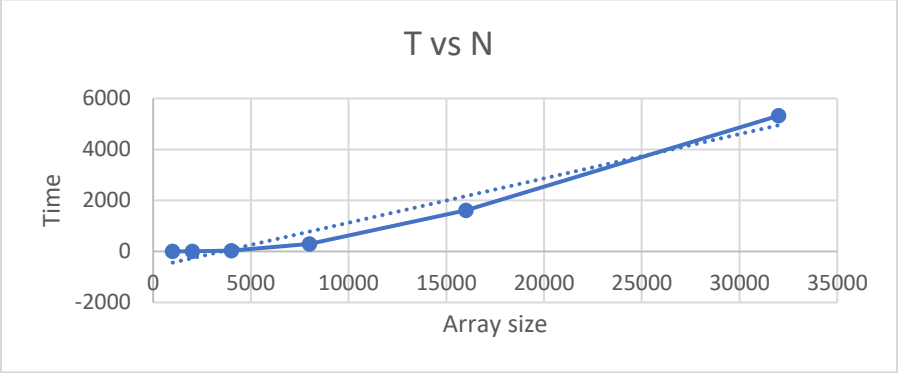
$$T(N) = aN^{1.758}$$

Ordered Array

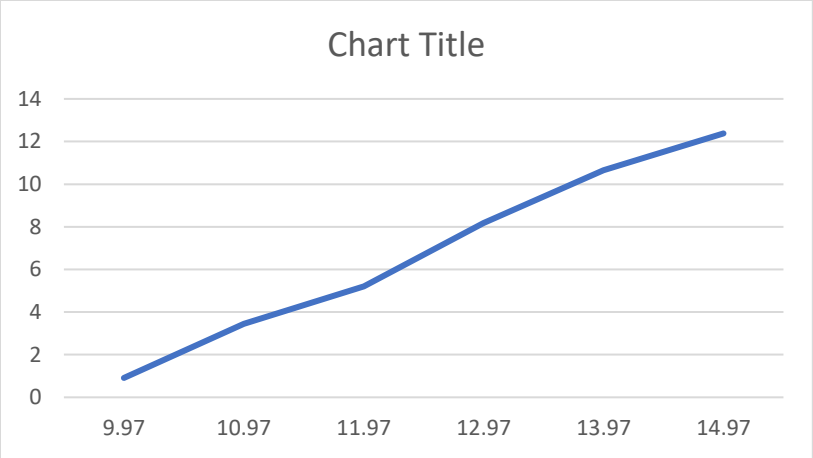
Various sizes of the Array and the running time of the Insertion sort

Ordered Array						
Array Size	Time	Ratio (Time/Previous Time)	lg(Array size)	lg(Time)	Log Ratio	Slope
1000	0.005	-	9.97	-7.64	-1.3	
2000	0.016	3.2	10.97	-5.97	-1.84	0.78
4000	0.036	2.25	11.97	-4.8	-2.49	0.8
8000	0.05225	1.45	12.97	-4.26	-3.04	0.89
16000	0.127	2.43	13.97	-2.98	-4.69	0.7
32000	0.164	1.29	14.97	-2.61	-5.74	0.88
					Avg Slope	0.81

Analysis of experimental data (the running time of insertion sort with random ordered input)



Standard Plot: Running time T(n) Vs Array size N



Log-Log Plot: lg(T(n)) Vs lg(N)

The equation of the log-log plot is

$$\lg(T(N)) = 0.81 \lg N + \lg a$$

Which is equivalent to,

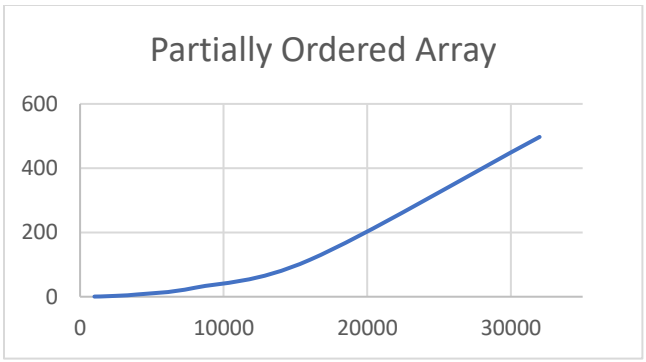
$$T(N) = aN^{0.81}$$

Partially Ordered Array

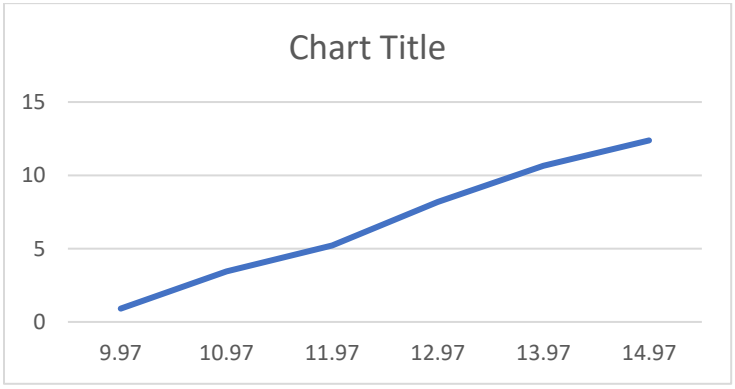
Various sizes of the Array and the running time of the Insertion sort

Partially Ordered Array						
Array Size	**Time**	**Ratio (Time/Previous Time)**	**lg(Array size)**	**lg(Time)**	**Log Ratio**	**Slope**
1000	0.49	-	9.97	-1.03	-9.68	
2000	2.09	4.27	10.97	1.06	10.35	-1.03
4000	9.475	4.53	11.97	3.24	3.69	3.06
8000	31.4	3.31	12.97	4.97	2.61	1.53
16000	138.22	4.4	13.97	7.11	1.96	1.43
32000	629.86	4.56	14.97	9.3	1.61	1.31
					Avg Slope	1.26

Analysis of experimental data (the running time of insertion sort with random ordered input)



Standard Plot: Running time T(n) Vs Array size N



Log-Log Plot: lg(T(n)) Vs lg(N)

The equation of the log-log plot is

$$\lg(T(N)) = 1.26 \lg N + \lg a$$

Which is equivalent to,

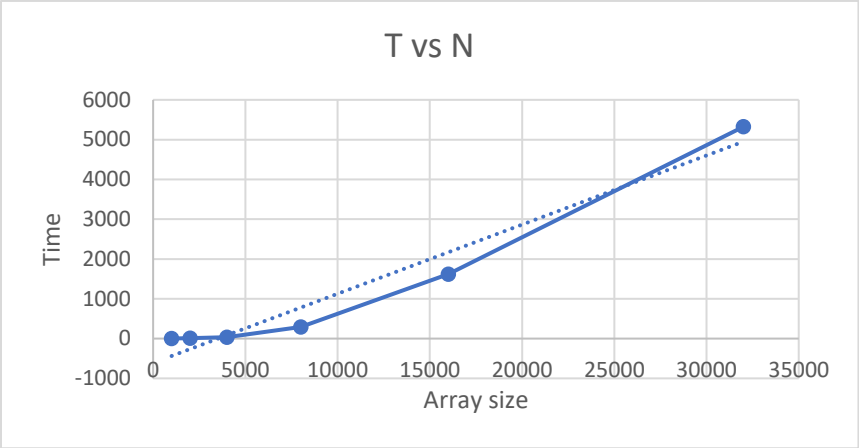
$$T(N) = aN^{1.26}$$

Reverse Ordered Array

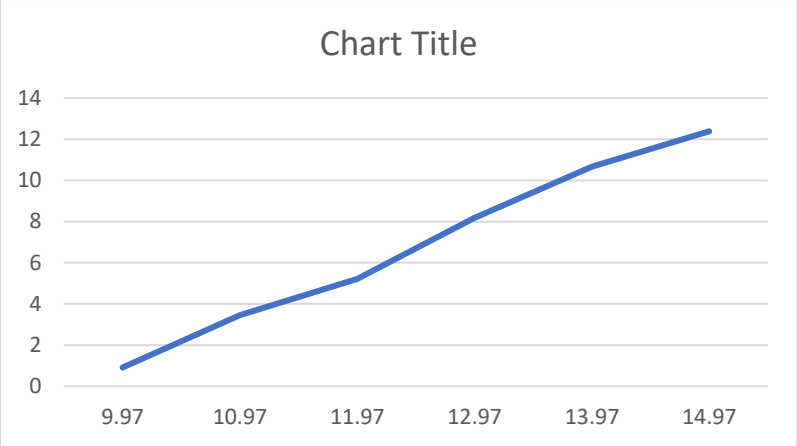
Various sizes of the Array and the running time of the Insertion sort

Reverse Ordered Array						
Array Size	**Time**	**Ratio (Time/Previous Time)**	**lg(Array size)**	**lg(Time)**	**Log Ratio**	**Slope**
1000	1.88	-	9.97	0.91	10.96	
2000	10.9	5.8	10.97	3.45	3.18	3.79
4000	37.04	3.4	11.97	5.21	2.3	1.51
8000	290.13	7.83	12.97	8.18	1.59	1.57
16000	1615.62	5.57	13.97	10.66	1.31	1.3
32000	5323	3.29	14.97	12.38	1.21	1.16
					Avg Slope	1.866

Analysis of experimental data (the running time of insertion sort with random ordered input)



Standard Plot: Running time T(n) Vs Array size N



Log-Log Plot: lg(T(n)) Vs lg(N)

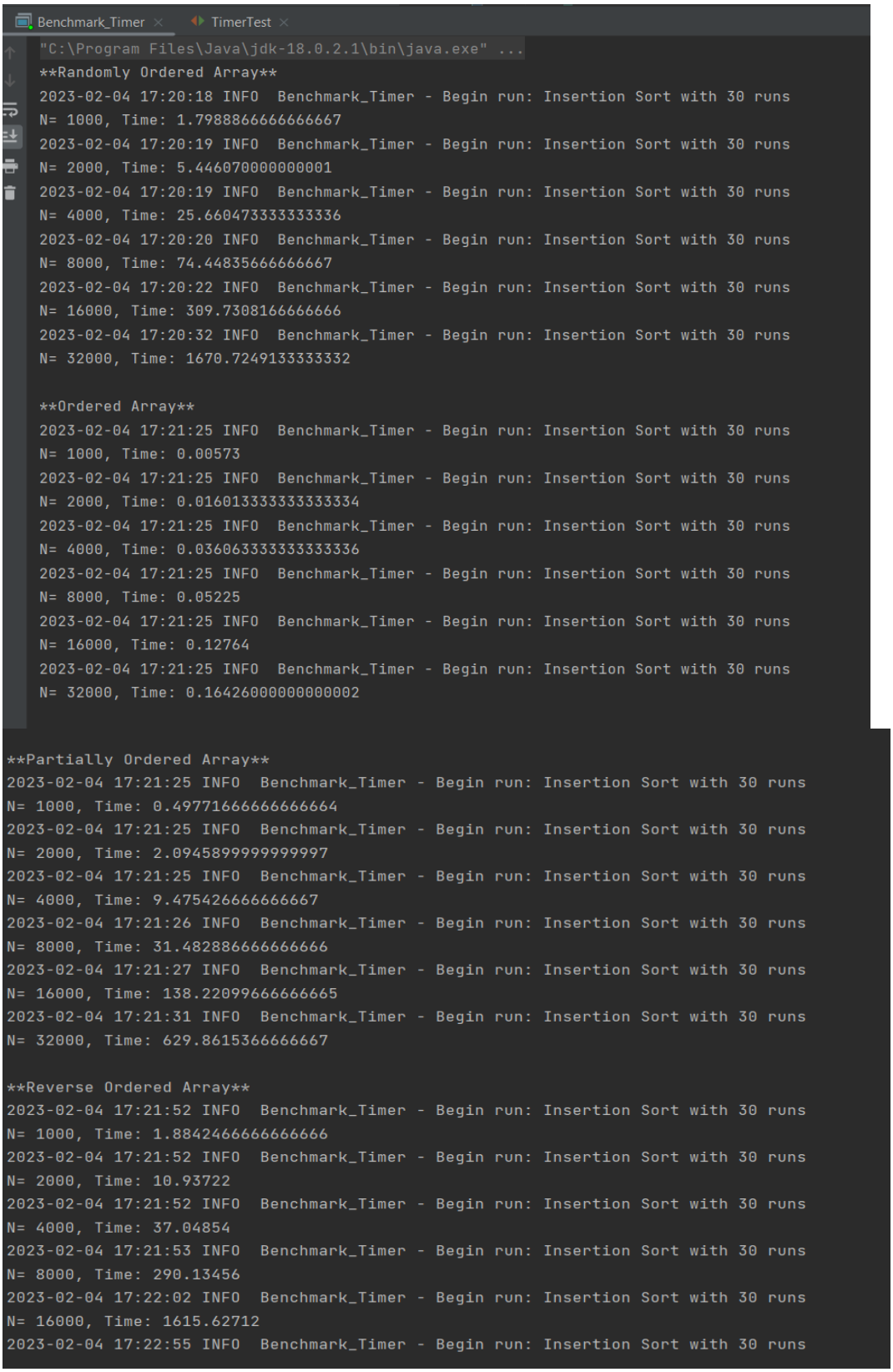
The equation of the log-log plot is

$$\lg(T(N)) = 1.86 \lg N + \lg a$$

Which is equivalent to,

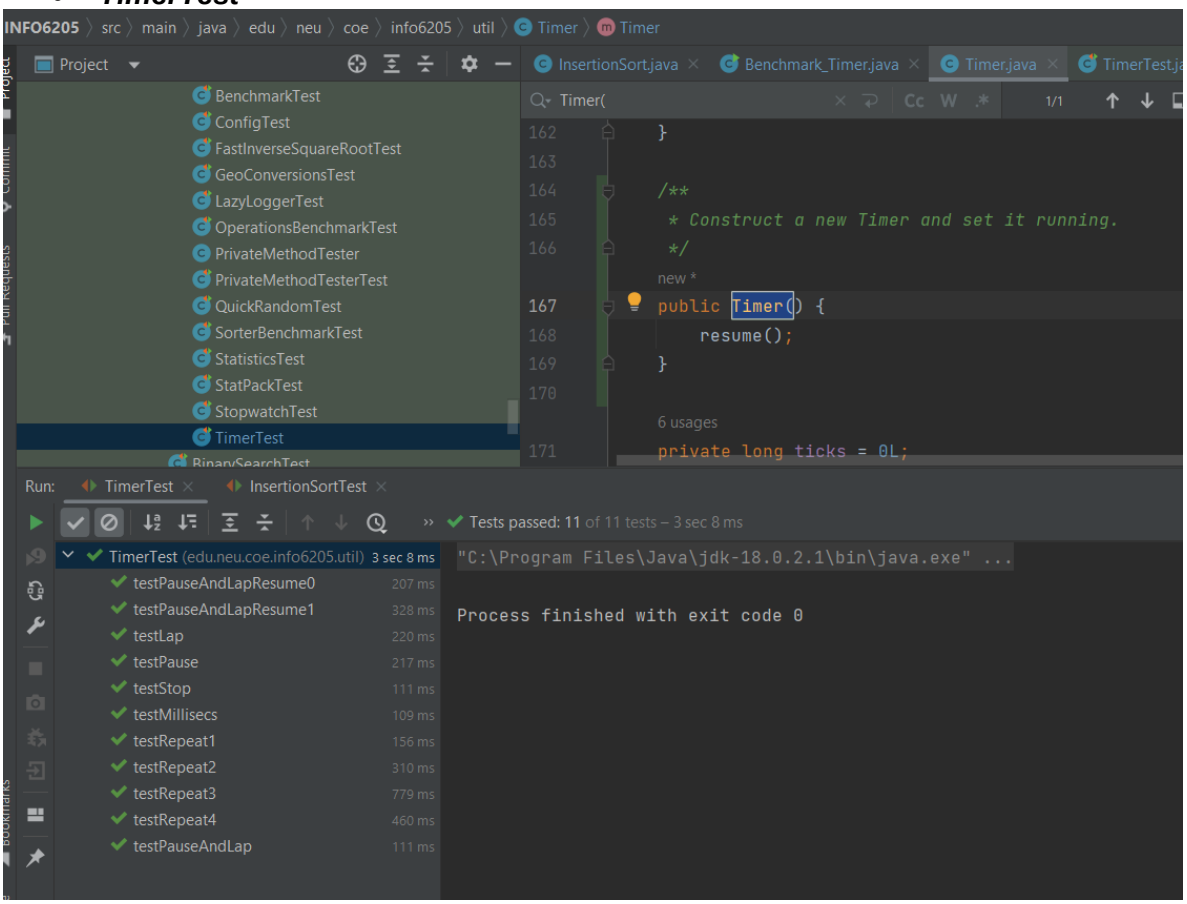
$$T(N) = aN^{1.86}$$

Output Screenshot

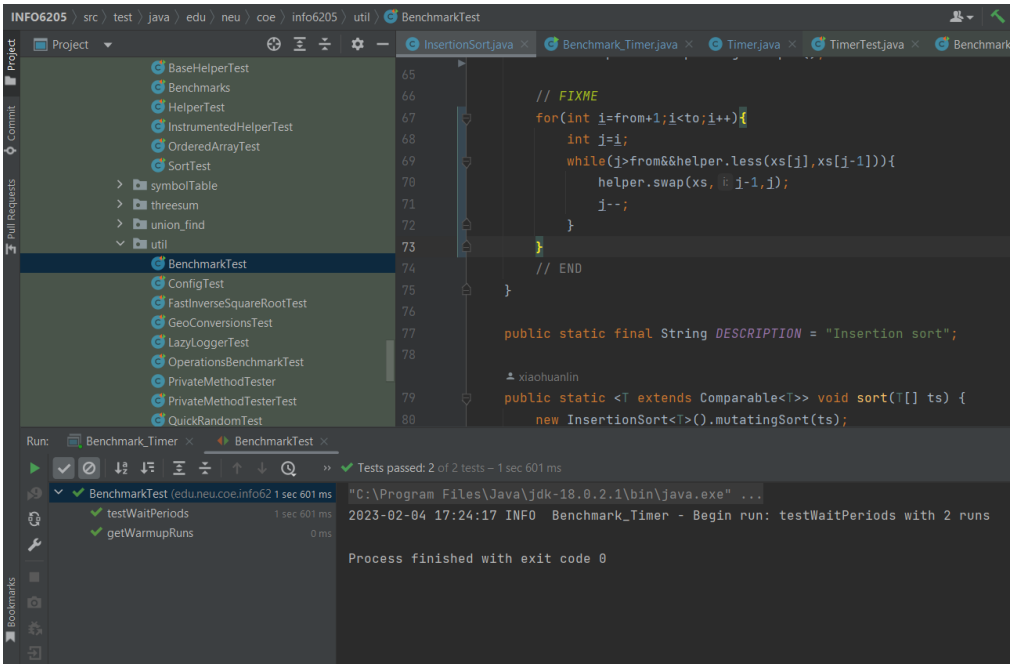

Benchmark_Timer
TimerTest
"C:\Program Files\Java\jdk-18.0.2.1\bin\java.exe" ...
Randomly Ordered Array
2023-02-04 17:20:18 INFO Benchmark_Timer - Begin run: Insertion Sort with 30 runs
N= 1000, Time: 1.7988866666666667
2023-02-04 17:20:19 INFO Benchmark_Timer - Begin run: Insertion Sort with 30 runs
N= 2000, Time: 5.446070000000001
2023-02-04 17:20:19 INFO Benchmark_Timer - Begin run: Insertion Sort with 30 runs
N= 4000, Time: 25.660473333333336
2023-02-04 17:20:20 INFO Benchmark_Timer - Begin run: Insertion Sort with 30 runs
N= 8000, Time: 74.44835666666667
2023-02-04 17:20:22 INFO Benchmark_Timer - Begin run: Insertion Sort with 30 runs
N= 16000, Time: 309.7308166666666
2023-02-04 17:20:32 INFO Benchmark_Timer - Begin run: Insertion Sort with 30 runs
N= 32000, Time: 1670.7249133333332
Ordered Array
2023-02-04 17:21:25 INFO Benchmark_Timer - Begin run: Insertion Sort with 30 runs
N= 1000, Time: 0.00573
2023-02-04 17:21:25 INFO Benchmark_Timer - Begin run: Insertion Sort with 30 runs
N= 2000, Time: 0.016013333333333334
2023-02-04 17:21:25 INFO Benchmark_Timer - Begin run: Insertion Sort with 30 runs
N= 4000, Time: 0.036063333333333336
2023-02-04 17:21:25 INFO Benchmark_Timer - Begin run: Insertion Sort with 30 runs
N= 8000, Time: 0.05225
2023-02-04 17:21:25 INFO Benchmark_Timer - Begin run: Insertion Sort with 30 runs
N= 16000, Time: 0.12764
2023-02-04 17:21:25 INFO Benchmark_Timer - Begin run: Insertion Sort with 30 runs
N= 32000, Time: 0.16426000000000002
Partially Ordered Array
2023-02-04 17:21:25 INFO Benchmark_Timer - Begin run: Insertion Sort with 30 runs
N= 1000, Time: 0.49771666666666664
2023-02-04 17:21:25 INFO Benchmark_Timer - Begin run: Insertion Sort with 30 runs
N= 2000, Time: 2.0945899999999997
2023-02-04 17:21:25 INFO Benchmark_Timer - Begin run: Insertion Sort with 30 runs
N= 4000, Time: 9.475426666666667
2023-02-04 17:21:26 INFO Benchmark_Timer - Begin run: Insertion Sort with 30 runs
N= 8000, Time: 31.482886666666666
2023-02-04 17:21:27 INFO Benchmark_Timer - Begin run: Insertion Sort with 30 runs
N= 16000, Time: 138.22099666666665
2023-02-04 17:21:31 INFO Benchmark_Timer - Begin run: Insertion Sort with 30 runs
N= 32000, Time: 629.8615366666667
Reverse Ordered Array
2023-02-04 17:21:52 INFO Benchmark_Timer - Begin run: Insertion Sort with 30 runs
N= 1000, Time: 1.8842466666666666
2023-02-04 17:21:52 INFO Benchmark_Timer - Begin run: Insertion Sort with 30 runs
N= 2000, Time: 10.93722
2023-02-04 17:21:52 INFO Benchmark_Timer - Begin run: Insertion Sort with 30 runs
N= 4000, Time: 37.04854
2023-02-04 17:21:53 INFO Benchmark_Timer - Begin run: Insertion Sort with 30 runs
N= 8000, Time: 290.13456
2023-02-04 17:22:02 INFO Benchmark_Timer - Begin run: Insertion Sort with 30 runs
N= 16000, Time: 1615.62712
2023-02-04 17:22:55 INFO Benchmark_Timer - Begin run: Insertion Sort with 30 runs

Unit Tests

- ***TimerTest***



- ***BenchmarkTest***



- ***InsertionSortTest***