Akhilanand Sirra (002197798)

Program Structures & Algorithms
Fall 2021
Assignment No. 2

Task

- (Part 1) Implement three methods of *Timer.java* & check implementation by running the unit tests in *BenchmarkTest.java* and *TimerTest.java*

- (Part 2) Implement *InsertionSort* (in the *InsertionSort* class) & check implementation by running the unit tests in *InsertionSortTest*

- (Part 3) Implement 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.

- Use the doubling method for choosing *n* and test for at least five values of *n*.

- Draw conclusions from the observations regarding the order of growth.

Relationship Conclusion:

The order of growth of the running time of Insertion Sort (Randomly ordered array of size N) is $\approx N^{2.033}$.

The order of growth of the running time of Insertion Sort (Ordered array of size N) is $\approx N^{1}$.

The order of growth of the running time of Insertion Sort (Partially ordered array of size N) is $\approx N^{2.027}$.

The order of growth of the running time of Insertion Sort (Reverse ordered array of size N) is $\approx N^{2.068}$.

In terms of order of growth, the running time of Insertion sort arranged in ascending order

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

Evidence to support the conclusion:

Let $T(N)$ be the running time of Insertion sort for N numbers

Using the doubling method, we can generate a sequence of random input arrays, doubling the array size at each step, and print the running times of Insertionsort() for each input size of different ordering situations.

Random Ordered Array

Here, we took the initial input size of the array as 1000, kept increasing it till 16000, and measured the running time of the insertion sort algorithm for a randomly ordered array.

Array Length	Time	lg(Array Length)	lg(Time)	Slope
1000	3.04	9.965784285	1.604071324	-
2000	7.2	10.96578428	2.847996907	-
4000	29.28	11.96578428	4.871843649	2.0238467
8000	110.36	12.96578428	6.786073552	1.9142299
16000	494.04	13.96578428	8.948484044	2.1624105
			Avg Slope=	2.0334957

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

The below diagrams show the result of plotting the above table data, both on a standard and a log-log scale, with the input size N on the $x-axis$ and the running time $T(N)$ on the $y-axis$.

Standard Plot: problem size N vs running time T(n)
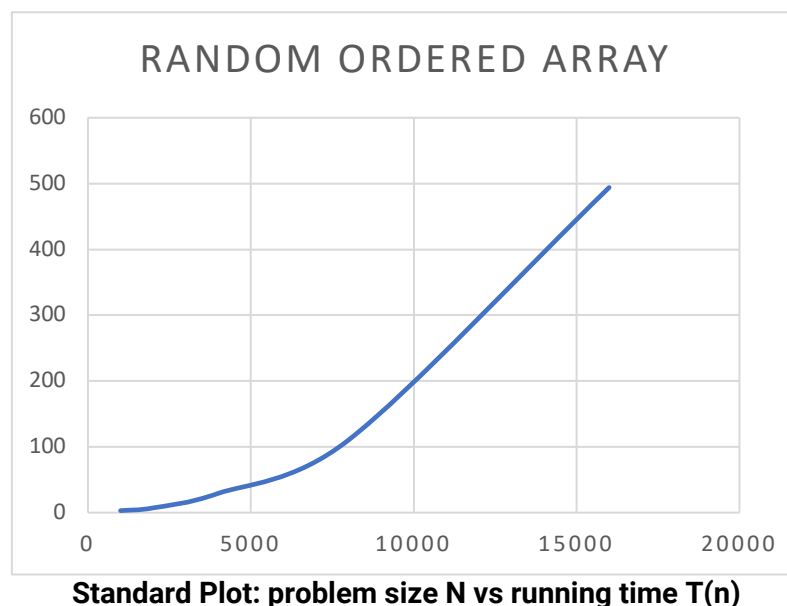
By looking at the plots we can lead to hypothesis that the function $T(n)$ is in form of

$$T(N) = aN^b$$

Where,

$$N = \text{input size}$$

$$a = \text{machine dependent constant}$$

$$b = \text{slope of the log-log scale graph}$$
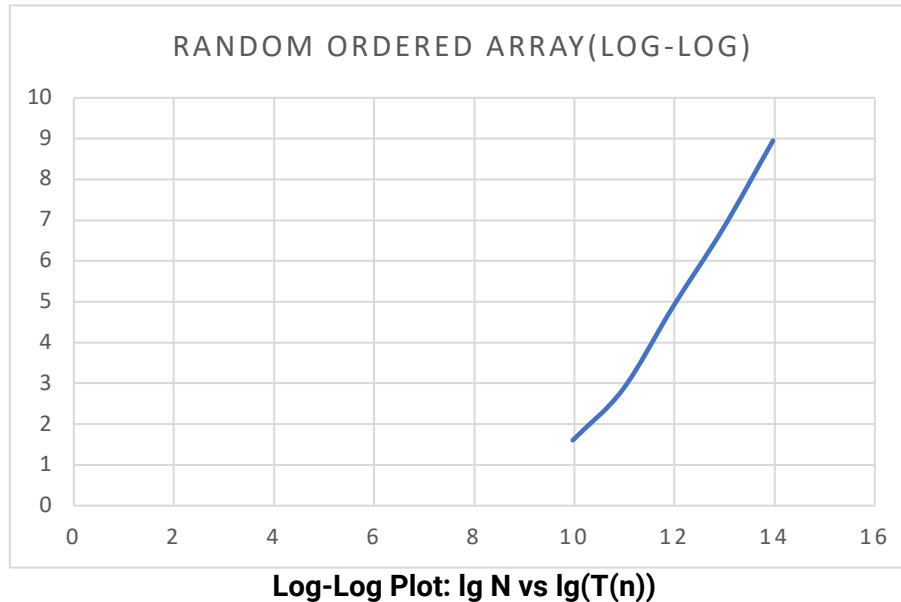
RANDOM ORDERED ARRAY(LOG-LOG)

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

The log-log plot of the above computed data leads to a hypothesis that it fits a straight line of slope 2 on the log-log plot. The data above in the table indicates that the average slope was found to be ~2.0334.

The equation of such a line is

$$lg(T(N)) = 2.0334 \, lg \, N + lg \, a$$

Which is equivalent to,

$$T(N) = aN^{2.0334}$$

Ordered Array

Array Length	Time	lg(Array Length)	lg(Time)	Slope
1000	0	9.965784285	-	-
2000	0	10.96578428	-	-
4000	0	11.96578428	-	-
8000	0.04	12.96578428	-4.64385619	-
16000	0.08	13.96578428	-3.64385619	1
			Avg Slope=	1

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

Here, we took the initial input size of the array as 1000, kept increasing it till 16000, and measured the running time of the insertion sort algorithm for an ordered array.
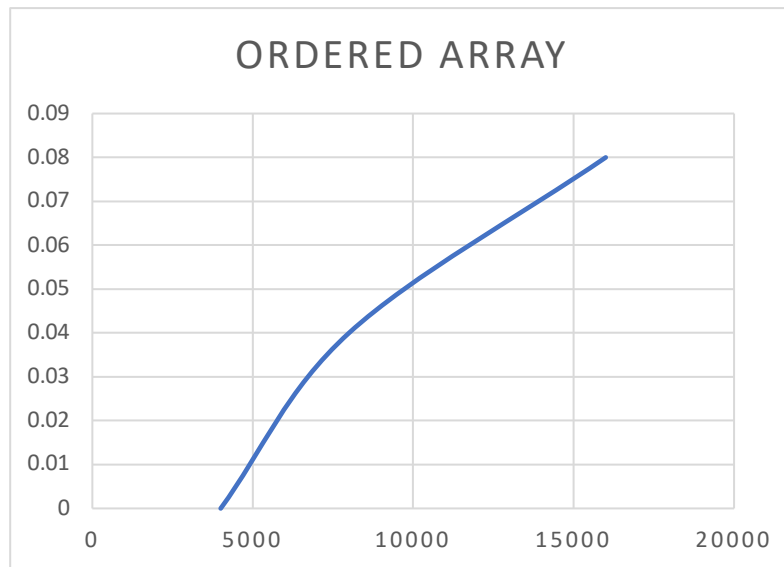
Standard Plot: problem size N vs running time T(n)
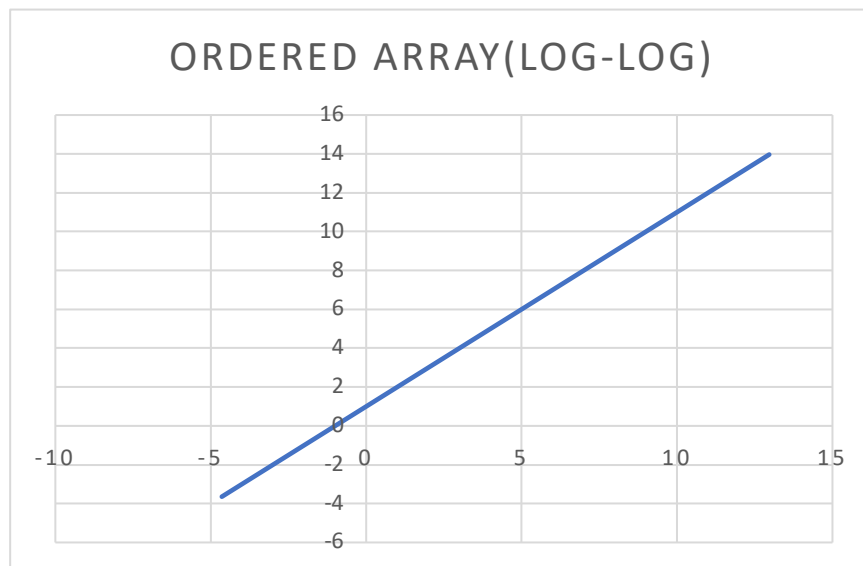
Log-Log Plot: lg N vs lg(T(n))

The log-log plot of the above computed data leads to a hypothesis that it fits a straight line of slope 1 on the log-log plot. The data above in the table indicates that the average slope was found to be ~1.

The equation of such a line is

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

Which is equivalent to,

$$T(N) = aN$$

When the ordering situation of the array is sorted, insertion sort takes the linear time to run. As it takes $N - 1$ compares and 0 exchanges.

Partially Ordered Array

Array Length	Time	lg(Array Length)	lg(Time)	Slope
1000	0.88	9.965784285	-0.184424571	-
2000	3.4	10.96578428	1.765534746	1.9499593
4000	14.16	11.96578428	3.82374936	2.0582146
8000	55.76	12.96578428	5.801158656	1.9774093
16000	243	13.96578428	7.924812504	2.1236538
			Avg Slope=	2.0273093

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

Here, we took initial input size of array as 1000 and kept increasing it till 16000 and measured the running time of the insertion sort algorithm for a partially ordered array.
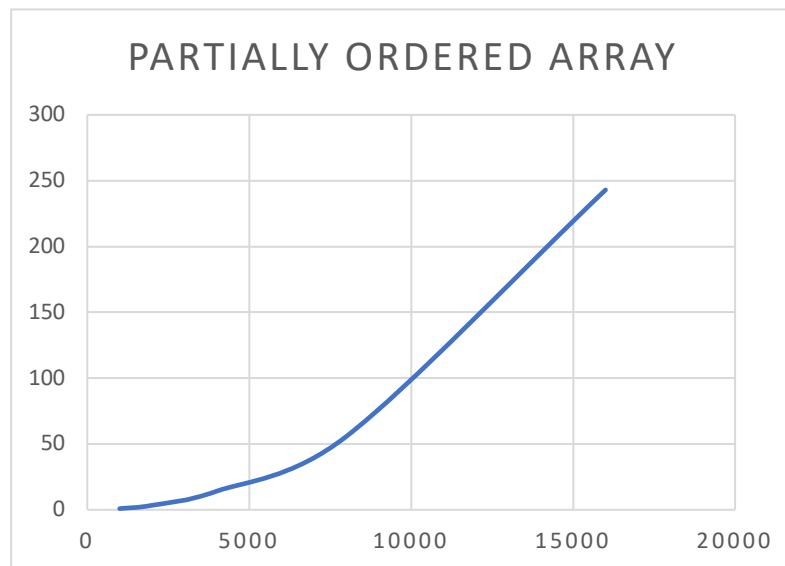
Standard Plot: problem size N vs running time T(n)
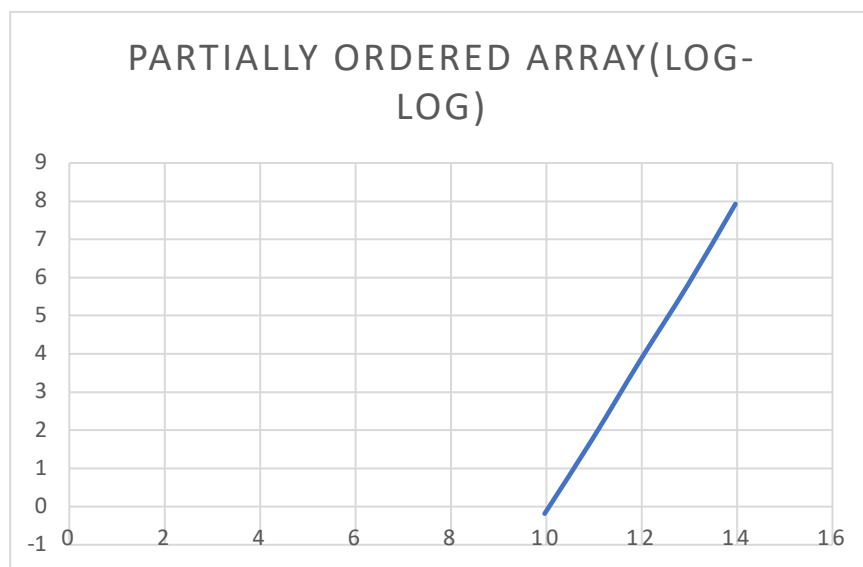
Log-Log Plot: lg N vs lg(T(n))

The log-log plot of the above computed data leads to a hypothesis that it fits a straight line of slope 2 on the log-log plot. The data above in the table indicates that the average slope was found to be ~2.0273.

The equation of such a line is

$$lg(T(N)) = 2.0273 \, lg \, N + lg \, a$$

Which is equivalent to,

$$T(N) = aN^{2.0273}$$

Reverse Ordered Array

Array Length	Time	lg(Array Length)	lg(Time)	Slope
1000	3.44	9.965784285	1.782408565	-
2000	14.04	10.96578428	3.811471031	2.0290625
4000	59.4	11.96578428	5.892391026	2.08092
8000	250.68	12.96578428	7.969703088	2.0773121
16000	1064.08	13.96578428	10.0553909	2.0856878
			Avg Slope=	2.0682456

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

Here, we took the initial input size of the array as 1000, kept increasing it till 16000, and measured the running time of the insertion sort algorithm for a reverse ordered array.
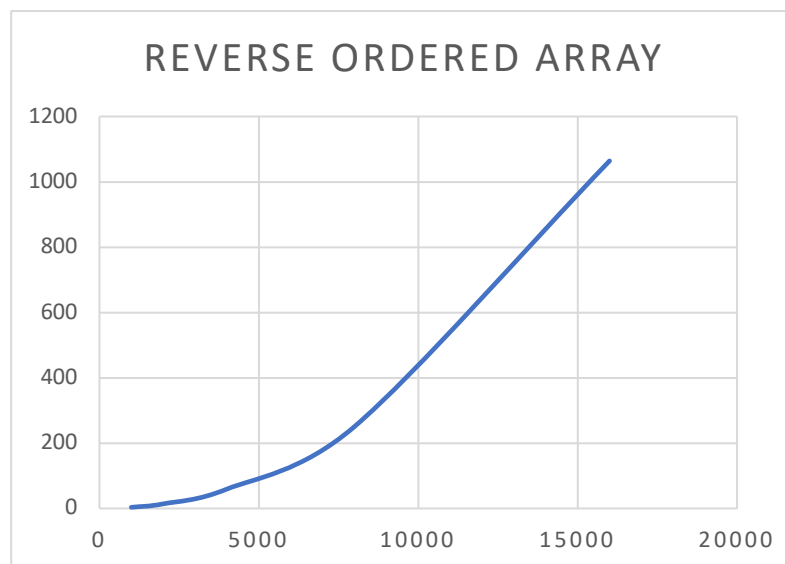
Standard Plot: problem size N vs running time T(n)
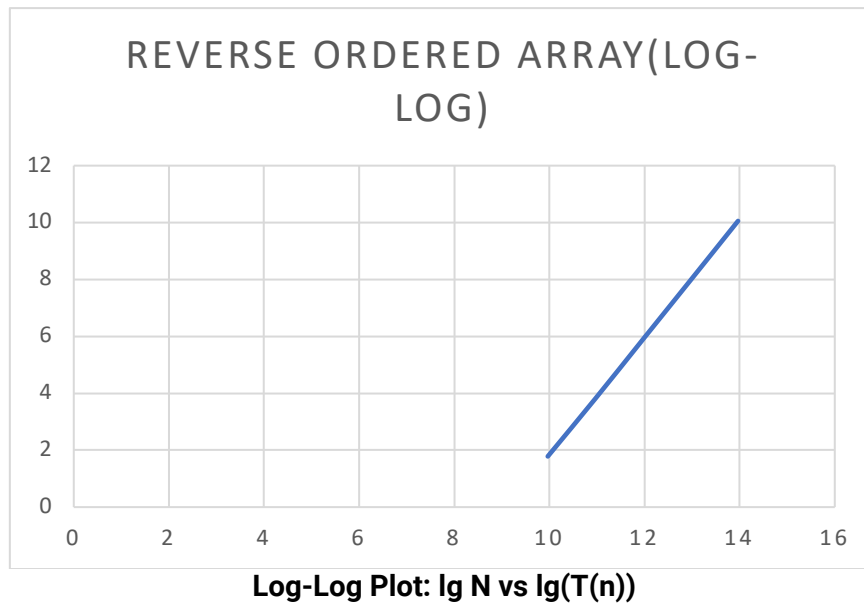
Log-Log Plot: lg N vs lg(T(n))

The log-log plot of the above computed data leads to a hypothesis that it fits a straight line of slope 2 on the log-log plot. The data above in the table indicates that the average slope was found to be ~2.0682.

The equation of such a line is

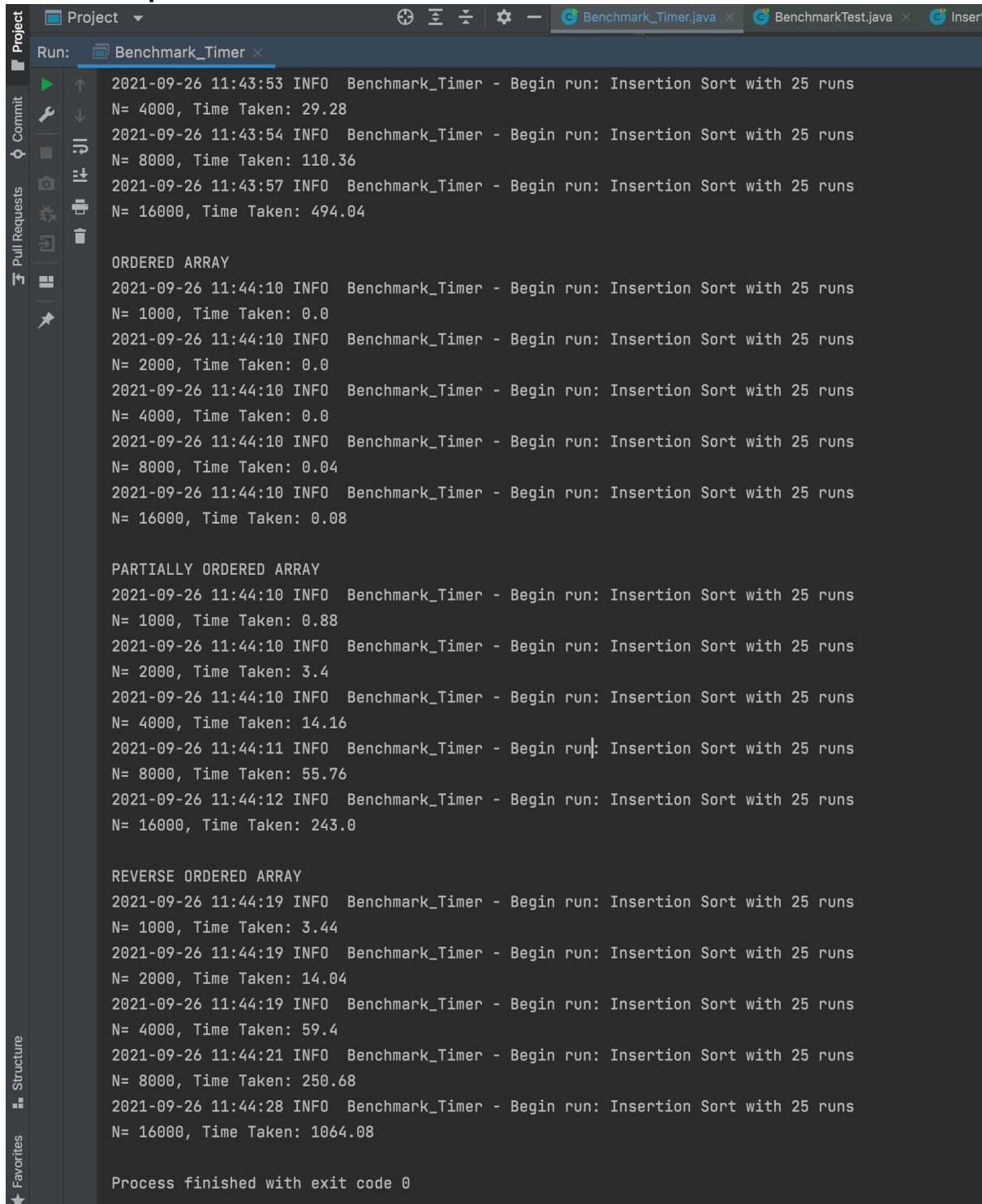$$lg(T(N)) = 2.0682 \, lg \, N + lg \, a$$

Which is equivalent to,

$$T(N) = aN^{2.0628}$$

When the ordering situation of the array is reverse sorted, insertion sort takes the longest time to run. As it takes $\sim N^2/2$ compares and $\sim N^2/2$ exchanges.

Console Output:

```
2021-09-26 11:43:53 INFO  Benchmark_Timer - Begin run: Insertion Sort with 25 runs
N= 4000, Time Taken: 29.28
2021-09-26 11:43:54 INFO  Benchmark_Timer - Begin run: Insertion Sort with 25 runs
N= 8000, Time Taken: 110.36
2021-09-26 11:43:57 INFO  Benchmark_Timer - Begin run: Insertion Sort with 25 runs
N= 16000, Time Taken: 494.04

ORDERED ARRAY
2021-09-26 11:44:10 INFO  Benchmark_Timer - Begin run: Insertion Sort with 25 runs
N= 1000, Time Taken: 0.0
2021-09-26 11:44:10 INFO  Benchmark_Timer - Begin run: Insertion Sort with 25 runs
N= 2000, Time Taken: 0.0
2021-09-26 11:44:10 INFO  Benchmark_Timer - Begin run: Insertion Sort with 25 runs
N= 4000, Time Taken: 0.0
2021-09-26 11:44:10 INFO  Benchmark_Timer - Begin run: Insertion Sort with 25 runs
N= 8000, Time Taken: 0.04
2021-09-26 11:44:10 INFO  Benchmark_Timer - Begin run: Insertion Sort with 25 runs
N= 16000, Time Taken: 0.08

PARTIALLY ORDERED ARRAY
2021-09-26 11:44:10 INFO  Benchmark_Timer - Begin run: Insertion Sort with 25 runs
N= 1000, Time Taken: 0.88
2021-09-26 11:44:10 INFO  Benchmark_Timer - Begin run: Insertion Sort with 25 runs
N= 2000, Time Taken: 3.4
2021-09-26 11:44:10 INFO  Benchmark_Timer - Begin run: Insertion Sort with 25 runs
N= 4000, Time Taken: 14.16
2021-09-26 11:44:11 INFO  Benchmark_Timer - Begin run: Insertion Sort with 25 runs
N= 8000, Time Taken: 55.76
2021-09-26 11:44:12 INFO  Benchmark_Timer - Begin run: Insertion Sort with 25 runs
N= 16000, Time Taken: 243.0

REVERSE ORDERED ARRAY
2021-09-26 11:44:19 INFO  Benchmark_Timer - Begin run: Insertion Sort with 25 runs
N= 1000, Time Taken: 3.44
2021-09-26 11:44:19 INFO  Benchmark_Timer - Begin run: Insertion Sort with 25 runs
N= 2000, Time Taken: 14.04
2021-09-26 11:44:19 INFO  Benchmark_Timer - Begin run: Insertion Sort with 25 runs
N= 4000, Time Taken: 59.4
2021-09-26 11:44:21 INFO  Benchmark_Timer - Begin run: Insertion Sort with 25 runs
N= 8000, Time Taken: 250.68
2021-09-26 11:44:28 INFO  Benchmark_Timer - Begin run: Insertion Sort with 25 runs
N= 16000, Time Taken: 1064.08

Process finished with exit code 0
```

RANDOMLY ORDERED ARRAY
2021-09-26 11:43:52 INFO Benchmark_Timer - Begin run: Insertion Sort with 25 runs
N= 1000, Time Taken: 3.04
2021-09-26 11:43:53 INFO Benchmark_Timer - Begin run: Insertion Sort with 25 runs
N= 2000, Time Taken: 7.2
2021-09-26 11:43:53 INFO Benchmark_Timer - Begin run: Insertion Sort with 25 runs
N= 4000, Time Taken: 29.28

2021-09-26 11:43:54 INFO Benchmark_Timer - Begin run: Insertion Sort with 25 runs N= 8000, Time Taken: 110.36
2021-09-26 11:43:57 INFO Benchmark_Timer - Begin run: Insertion Sort with 25 runs N= 16000, Time Taken: 494.04

ORDERED ARRAY
2021-09-26 11:44:10 INFO Benchmark_Timer - Begin run: Insertion Sort with 25 runs N= 1000, Time Taken: 0.0
2021-09-26 11:44:10 INFO Benchmark_Timer - Begin run: Insertion Sort with 25 runs N= 2000, Time Taken: 0.0
2021-09-26 11:44:10 INFO Benchmark_Timer - Begin run: Insertion Sort with 25 runs N= 4000, Time Taken: 0.0
2021-09-26 11:44:10 INFO Benchmark_Timer - Begin run: Insertion Sort with 25 runs N= 8000, Time Taken: 0.04
2021-09-26 11:44:10 INFO Benchmark_Timer - Begin run: Insertion Sort with 25 runs N= 16000, Time Taken: 0.08

PARTIALLY ORDERED ARRAY
2021-09-26 11:44:10 INFO Benchmark_Timer - Begin run: Insertion Sort with 25 runs N= 1000, Time Taken: 0.88
2021-09-26 11:44:10 INFO Benchmark_Timer - Begin run: Insertion Sort with 25 runs N= 2000, Time Taken: 3.4
2021-09-26 11:44:10 INFO Benchmark_Timer - Begin run: Insertion Sort with 25 runs N= 4000, Time Taken: 14.16
2021-09-26 11:44:11 INFO Benchmark_Timer - Begin run: Insertion Sort with 25 runs N= 8000, Time Taken: 55.76
2021-09-26 11:44:12 INFO Benchmark_Timer - Begin run: Insertion Sort with 25 runs N= 16000, Time Taken: 243.0

REVERSE ORDERED ARRAY
2021-09-26 11:44:19 INFO Benchmark_Timer - Begin run: Insertion Sort with 25 runs N= 1000, Time Taken: 3.44
2021-09-26 11:44:19 INFO Benchmark_Timer - Begin run: Insertion Sort with 25 runs N= 2000, Time Taken: 14.04
2021-09-26 11:44:19 INFO Benchmark_Timer - Begin run: Insertion Sort with 25 runs N= 4000, Time Taken: 59.4
2021-09-26 11:44:21 INFO Benchmark_Timer - Begin run: Insertion Sort with 25 runs N= 8000, Time Taken: 250.68
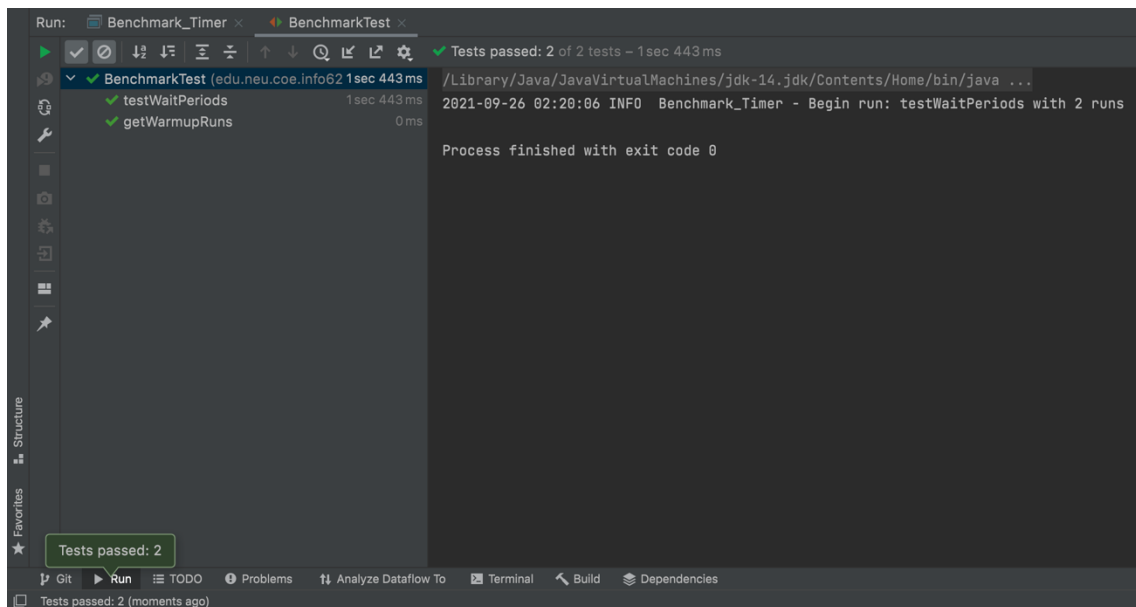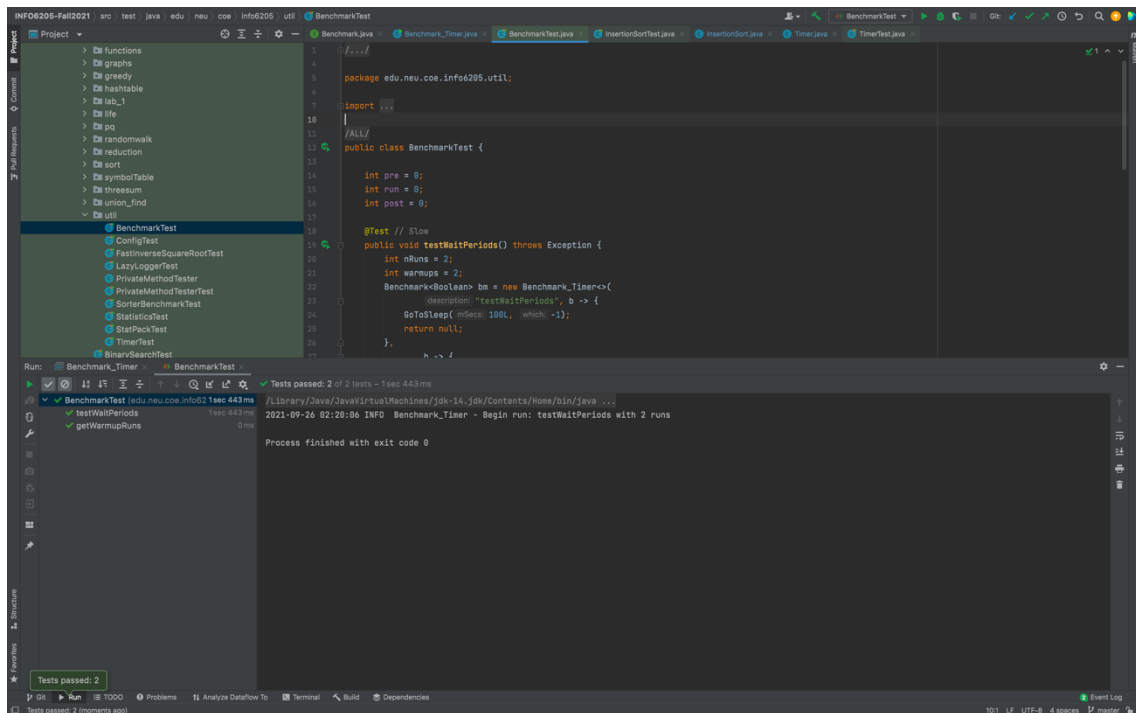2021-09-26 11:44:28 INFO Benchmark_Timer - Begin run: Insertion Sort with 25 runs N= 16000, Time Taken: 1064.08

Process finished with exit code 0

Unit tests result:

TimerTest.java
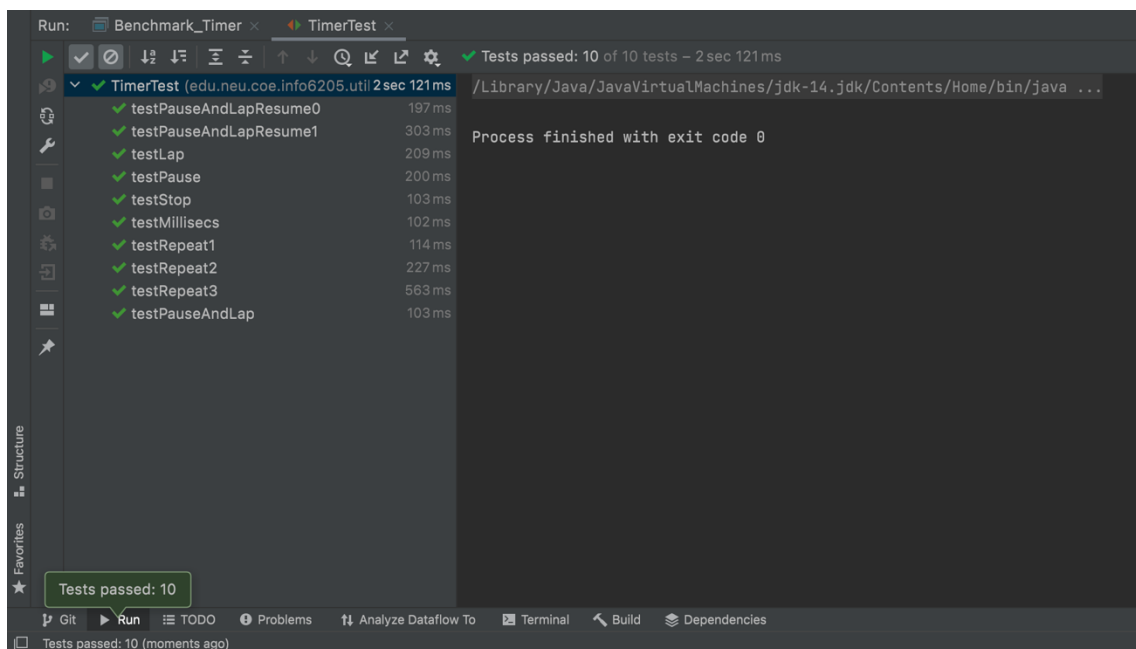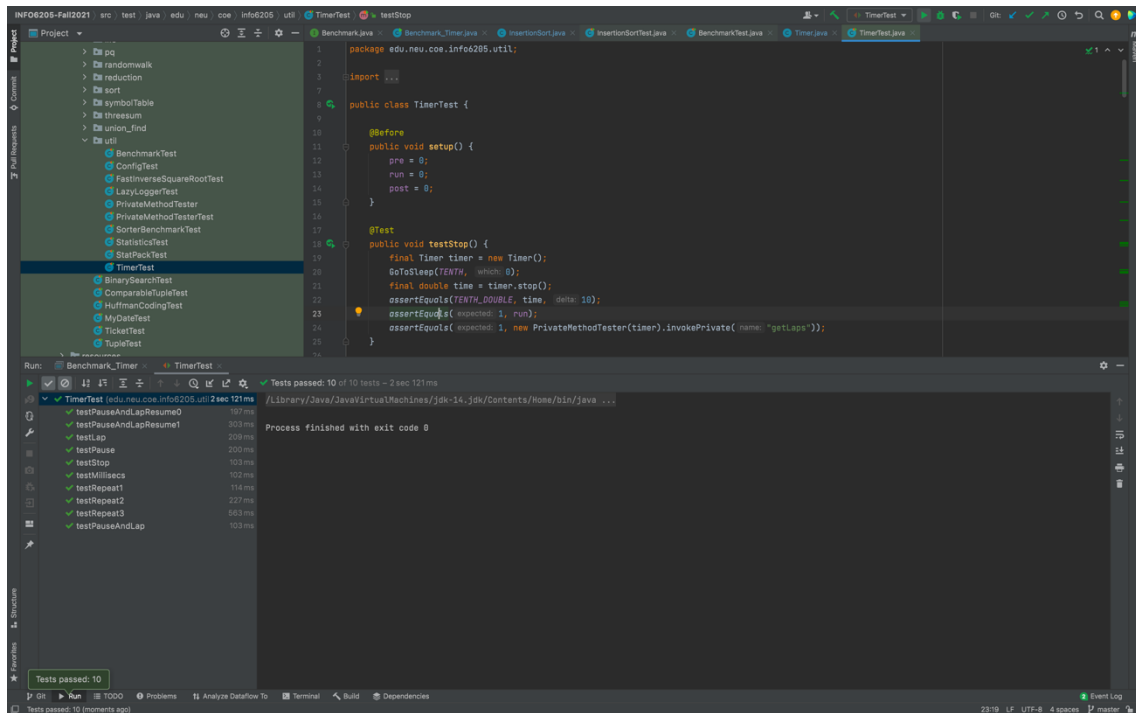
BenchmarkTest.java

InsertionSortTest.java