INFO 6205
Program Structures & Algorithms
Fall 2020
Assignment No 2

- **Task**

  Our task was to complete the implementation of class Timer.java and InsertionSort.java and also conduct a benchmark test for insertion sort for four types of arrays: ordered, randomly ordered, partially ordered and reverse-ordered arrays. The task was finished in the following steps:
  1. Complete the implementation of Timer.java
  2. Complete the implementation of InsertionSort.java
  3. Check implementation of Timer.java by running TimerTest.java and BenchmarkTest.java
  4. Implement InsertionSort.java
  5. Use Benchmark_Timer.java to conduct the benchmark test for insertion sort
  6. Plot and report the results
  7. Make observations and draw conclusions

- **Output**

  I conducted the benchmark test for insertion sort by randomly generating an input array of size n and running the experiment 10 times for each n. I doubled n at each run and reported the time in milliseconds. The size of n was: 800, 1600, 3200, 6400, 12800, 51200, 102400. I repeated the experiment for an ordered array, randomly ordered array, reverse ordered array and partially ordered array.

  ### Console Output
```
2020-09-24 23:36:59 INFO  Benchmark_Timer — Begin run: Benchmark Test for Insertion Array with 10 runs
For input size: 800 and ordered array,  insertion sort takes: 0.0
2020-09-24 23:36:59 INFO  Benchmark_Timer — Begin run: Benchmark Test for Insertion Array with 10 runs
For input size: 1600 and ordered array,  insertion sort takes: 0.0
2020-09-24 23:36:59 INFO  Benchmark_Timer — Begin run: Benchmark Test for Insertion Array with 10 runs
For input size: 3200 and ordered array,  insertion sort takes: 0.1
2020-09-24 23:36:59 INFO  Benchmark_Timer — Begin run: Benchmark Test for Insertion Array with 10 runs
For input size: 6400 and ordered array,  insertion sort takes: 0.2
2020-09-24 23:36:59 INFO  Benchmark_Timer — Begin run: Benchmark Test for Insertion Array with 10 runs
For input size: 12800 and ordered array,  insertion sort takes: 0.2
2020-09-24 23:36:59 INFO  Benchmark_Timer — Begin run: Benchmark Test for Insertion Array with 10 runs
For input size: 25600 and ordered array,  insertion sort takes: 0.2
2020-09-24 23:37:00 INFO  Benchmark_Timer — Begin run: Benchmark Test for Insertion Array with 10 runs
For input size: 51200 and ordered array,  insertion sort takes: 0.3
2020-09-24 23:37:00 INFO  Benchmark_Timer — Begin run: Benchmark Test for Insertion Array with 10 runs
For input size: 102400 and ordered array,  insertion sort takes: 0.3
..........................................................................
2020-09-24 23:01:49 INFO  Benchmark_Timer — Begin run: Benchmark Test for Insertion Array with 10 runs
For input size: 800 and randomly ordered array,  insertion sort takes: 0.7
2020-09-24 23:01:49 INFO  Benchmark_Timer — Begin run: Benchmark Test for Insertion Array with 10 runs
For input size: 1600 and randomly ordered array,  insertion sort takes: 2.9
2020-09-24 23:01:49 INFO  Benchmark_Timer — Begin run: Benchmark Test for Insertion Array with 10 runs
For input size: 3200 and randomly ordered array,  insertion sort takes: 20.0
2020-09-24 23:01:49 INFO  Benchmark_Timer — Begin run: Benchmark Test for Insertion Array with 10 runs
For input size: 6400 and randomly ordered array,  insertion sort takes: 43.8
2020-09-24 23:01:50 INFO  Benchmark_Timer — Begin run: Benchmark Test for Insertion Array with 10 runs
For input size: 12800 and randomly ordered array,  insertion sort takes: 175.5
2020-09-24 23:01:52 INFO  Benchmark_Timer — Begin run: Benchmark Test for Insertion Array with 10 runs
For input size: 25600 and randomly ordered array,  insertion sort takes: 733.8
```

```
2020-09-24 23:02:01 INFO  Benchmark_Timer - Begin run: Benchmark Test for Insertion Array with 10 runs
For input size: 51200 and randomly ordered array,  insertion sort takes: 3078.9
2020-09-24 23:02:38 INFO  Benchmark_Timer - Begin run: Benchmark Test for Insertion Array with 10 runs
For input size: 102400 and randomly ordered array,  insertion sort takes: 12954.6
.........................................................................................
2020-09-24 23:05:13 INFO  Benchmark_Timer - Begin run: Benchmark Test for Insertion Array with 10 runs
For input size: 800 and reverse ordered array,  insertion sort takes: 1.3
2020-09-24 23:05:13 INFO  Benchmark_Timer - Begin run: Benchmark Test for Insertion Array with 10 runs
For input size: 1600 and reverse ordered array,  insertion sort takes: 7.2
2020-09-24 23:05:14 INFO  Benchmark_Timer - Begin run: Benchmark Test for Insertion Array with 10 runs
For input size: 3200 and reverse ordered array,  insertion sort takes: 25.3
2020-09-24 23:05:14 INFO  Benchmark_Timer - Begin run: Benchmark Test for Insertion Array with 10 runs
For input size: 6400 and reverse ordered array,  insertion sort takes: 87.2
2020-09-24 23:05:15 INFO  Benchmark_Timer - Begin run: Benchmark Test for Insertion Array with 10 runs
For input size: 12800 and reverse ordered array,  insertion sort takes: 361.6
2020-09-24 23:05:19 INFO  Benchmark_Timer - Begin run: Benchmark Test for Insertion Array with 10 runs
For input size: 25600 and reverse ordered array,  insertion sort takes: 1594.9
2020-09-24 23:05:38 INFO  Benchmark_Timer - Begin run: Benchmark Test for Insertion Array with 10 runs
For input size: 51200 and reverse ordered array,  insertion sort takes: 7035.2
2020-09-24 23:07:06 INFO  Benchmark_Timer - Begin run: Benchmark Test for Insertion Array with 10 runs
For input size: 102400 and reverse ordered array,  insertion sort takes: 30998.9
.........................................................................................
2020-09-24 23:13:23 INFO  Benchmark_Timer - Begin run: Benchmark Test for Insertion Array with 10 runs
For input size: 800 and partially array,  insertion sort takes: 0.2
2020-09-24 23:13:23 INFO  Benchmark_Timer - Begin run: Benchmark Test for Insertion Array with 10 runs
For input size: 1600 and partially array,  insertion sort takes: 0.8
2020-09-24 23:13:23 INFO  Benchmark_Timer - Begin run: Benchmark Test for Insertion Array with 10 runs
For input size: 3200 and partially array,  insertion sort takes: 3.9
2020-09-24 23:13:23 INFO  Benchmark_Timer - Begin run: Benchmark Test for Insertion Array with 10 runs
For input size: 6400 and partially array,  insertion sort takes: 13.8
2020-09-24 23:13:23 INFO  Benchmark_Timer - Begin run: Benchmark Test for Insertion Array with 10 runs
For input size: 12800 and partially array,  insertion sort takes: 41.4
2020-09-24 23:13:24 INFO  Benchmark_Timer - Begin run: Benchmark Test for Insertion Array with 10 runs
For input size: 25600 and partially array,  insertion sort takes: 173.2
2020-09-24 23:13:26 INFO  Benchmark_Timer - Begin run: Benchmark Test for Insertion Array with 10 runs
For input size: 51200 and partially array,  insertion sort takes: 732.3
2020-09-24 23:13:35 INFO  Benchmark_Timer - Begin run: Benchmark Test for Insertion Array with 10 runs
For input size: 102400 and partially array,  insertion sort takes: 3149.8
```

I have provided a summary below (Table 2.1) for the results of the time in milliseconds taken by insertion sort to run on array orderings of the our types.

Size of n	Ordered Array	Randomly Ordered	Reverse Ordered	Partially Ordered
800	0.0ms	0.7ms	1.3ms	0.2ms
1600	0.0ms	2.9ms	7.2ms	0.8ms
3200	0.1ms	20.0ms	25.3ms	3.9ms
6400	0.2ms	43.8ms	87.2.ms	13.8ms
12800	0.2ms	175.5ms	361.6ms	41.4ms
25600	0.2ms	733.8ms	1594.9ms	173.2ms
51200	0.3ms	3078.9ms	7035.2ms	732.3ms
102400	0.3ms	12954.6ms	30998.9ms	3149.8ms

Table 2.1: Summary of time taken by insertion sort (in milliseconds) to run on array orderings of four types: ordered array, randomly ordered array, reverse ordered array and partially ordered array

- **Observation and Conclusions**
  Insertion sort scans the array, compares each pair of elements and swaps elements if they are out of order. Since each operation contributes to the runtime of the algorithm, insertion sort can run in O(n) or O(n²) time depending on how the array to be sorted is ordered.

  We make the following observations and conclusions from our benchmark test:
  1. In the best-case scenario, insertion sort will simply compare elements in O(n) time and perform 0 swaps. Hence, the best case scenario happens when the array is sorted, and insertion sort runs in O(n) time.
  2. In the worst-case scenario, to insert the last element alone we will need n-1 comparisons and perform n-1 swaps. Similarly, for second last element, we will need n-2 comparisons and n-2 swaps, and so on. The number of operations for the worst case in insertion sort is:

  $$2\text{x}(1 + 2 + \cdots + n - 2 + n - 1)$$

  Using the summation formula, we get:

  $$\frac{2(n - 1)(n - 1 + 1)}{2} = n(n - 1)$$

  So, in the worst case, insertion sort takes O(n²), that is, when the array is sorted in the descending order.
  We observe that on an average (randomly ordered array, partially ordered array) insertion sort roughly takes O(n²) time. We can view this quadratic growth in Fig 3.1.
  3. Analyzing the run time for each algorithm, we can conclude that the run time for insertion sort for differently ordered arrays follows the following trend:
     Reverse Sorted Array > Randomly Ordered Array > Partially Ordered Array > Ordered Array
  4. Observing the n vs T graph (Fig 3.1) for the time taken by insertion sort on differently ordered arrays, we can see why a jump from a quadratic runtime (O(n²)) to a linear runtime (O(n)) is so sought after while developing an algorithm.
  5. Observing the log-log graph (Fig 3.2), that is, log(n) vs log(T) graph, we can see the log-log asymptote to a straight line, indicating that a power relationship exists. Finding the slope of the line, we will find the slope to be ≈ 2, giving the power of the relationship.
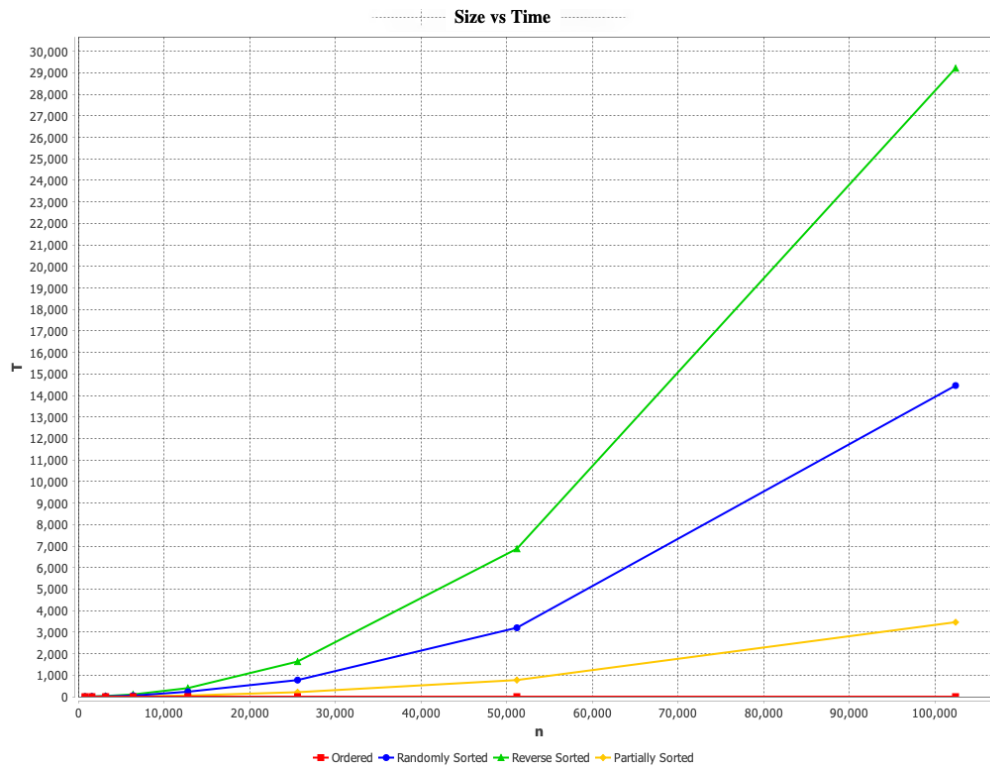
Fig 3.1: Plotting the size of the array vs time taken to run the insertion sort for array of four types: ordered, randomly ordered, partially ordered, reverse ordered.
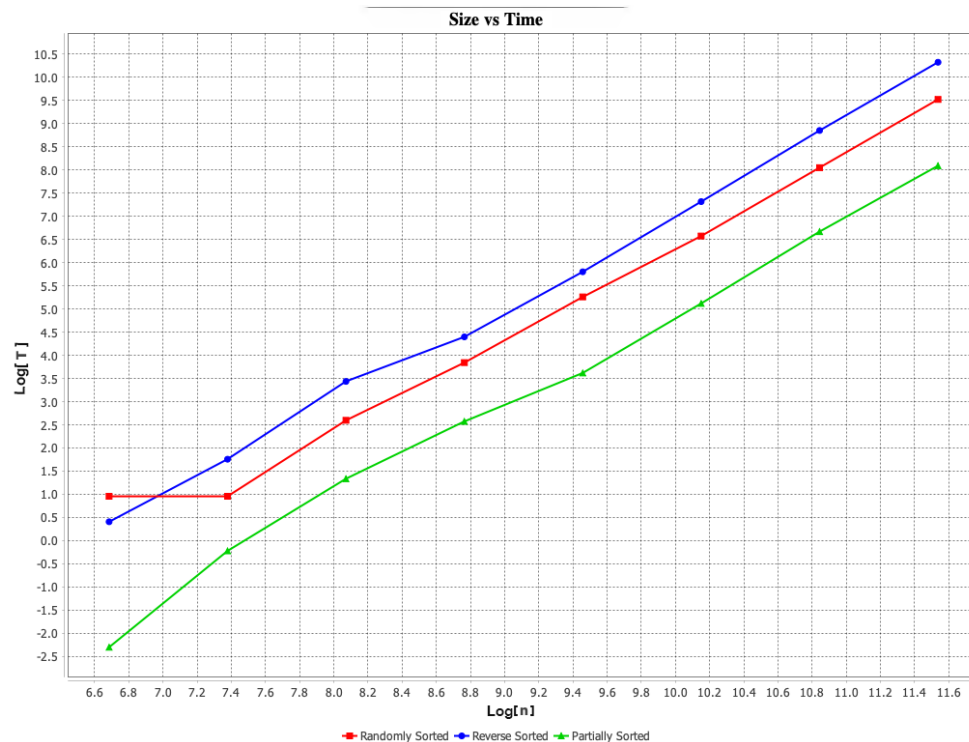
Fig 3.2: Plotting the log-log graph of log of size of the array vs log of time taken to run the insertion sort for array of three types: randomly ordered, partially ordered, reverse ordered.*
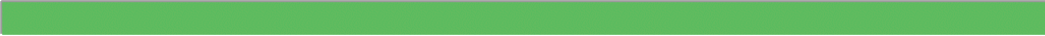
*Ordered array produced roughly 0.0 for most of the results, hence, log graph was not possible

- ## Screenshot of Unit test passing

TimerTest.java

Finished after 2.257 seconds

Runs:	10/10	☒ Errors:	0	☒ Failures:	0

▼ 🔲 edu.neu.coe.info6205.util.TimerTest [Runner: JUnit 4] (2.218 s)
- ✅ testPauseAndLapResume0 (0.224 s)
- ✅ testPauseAndLapResume1 (0.312 s)
- ✅ testLap (0.210 s)
- ✅ testPause (0.209 s)
- ✅ testStop (0.101 s)
- ✅ testMillisecs (0.105 s)
- ✅ testRepeat1 (0.161 s)
- ✅ testRepeat2 (0.237 s)
- ✅ testRepeat3 (0.556 s)
- ✅ testPauseAndLap (0.103 s)

BenchmarkTest.java

Finished after 1.586 seconds

Runs:	2/2	☒ Errors:	0	☒ Failures:	0

▼ 🔲 edu.neu.coe.info6205.util.BenchmarkTest [Runner: JUnit 4] (1.453 s)
- ✅ testWaitPeriods (1.453 s)
- ✅ getWarmupRuns (0.000 s)