Program Structures & Algorithms

Spring 2022

Assignment – 2

Name – Jayesh Kumar Khattar

NUID – 001568947

Task –

- (Part 1) You are to implement three (3) methods (*repeat*, *getClock*, and *toMillisecs*) 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. Don't forget to check your implementation by running the unit tests in *BenchmarkTest* and *TimerTest*.

- (Part 2) Implement *InsertionSort* (in the *InsertionSort* class) by simply looking up the insertion code used by *Arrays.sort.* If you have the *instrument = true* setting in *test/resources/config.ini*, then you will need to use the *helper* methods for comparing and swapping (so that they properly count the number of swaps/compares). You must run the unit tests in *InsertionSortTest*.

- (Part 3) Implement a main program (or you could do it via your own unit tests) to actually 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. 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.

Output –

Below is the output for running the benchmark time in the four cases – ordered, partially ordered, randomly ordered and reverse ordered.

Ordered and Partially ordered –

```
-----Ordered-----
2022-02-12 23:09:46 INFO  Benchmark_Timer - Begin run: Insertion Sort ->
50 runs with 500 in 0.020256720000000002
2022-02-12 23:09:46 INFO  Benchmark_Timer - Begin run: Insertion Sort ->
50 runs with 1000 in 0.0277067
2022-02-12 23:09:46 INFO  Benchmark_Timer - Begin run: Insertion Sort ->
50 runs with 2000 in 0.03553508
2022-02-12 23:09:47 INFO  Benchmark_Timer - Begin run: Insertion Sort ->
50 runs with 4000 in 0.038259919999999996
2022-02-12 23:09:47 INFO  Benchmark_Timer - Begin run: Insertion Sort ->
50 runs with 8000 in 0.00815838
2022-02-12 23:09:47 INFO  Benchmark_Timer - Begin run: Insertion Sort ->
50 runs with 16000 in 0.013850739999999999

-----Partially Ordered-----
2022-02-12 23:09:47 INFO  Benchmark_Timer - Begin run: Insertion Sort ->
50 runs with 500 in 0.44373574
2022-02-12 23:09:47 INFO  Benchmark_Timer - Begin run: Insertion Sort ->
50 runs with 1000 in 0.41757
2022-02-12 23:09:47 INFO  Benchmark_Timer - Begin run: Insertion Sort ->
50 runs with 2000 in 1.6270642000000002
2022-02-12 23:09:47 INFO  Benchmark_Timer - Begin run: Insertion Sort ->
50 runs with 4000 in 6.4777084
2022-02-12 23:09:47 INFO  Benchmark_Timer - Begin run: Insertion Sort ->
50 runs with 8000 in 26.3784742
2022-02-12 23:09:49 INFO  Benchmark_Timer - Begin run: Insertion Sort ->
50 runs with 16000 in 106.65565832
```

Randomly and reverse ordered –

```
50 runs with 16000 in 108.65585852

-----Randomly Ordered-----
2022-02-12 23:09:55 INFO  Benchmark_Timer - Begin run: Insertion Sort ->
50 runs with 500 in 0.20576504
2022-02-12 23:09:55 INFO  Benchmark_Timer - Begin run: Insertion Sort ->
50 runs with 1000 in 0.80822254
2022-02-12 23:09:55 INFO  Benchmark_Timer - Begin run: Insertion Sort ->
50 runs with 2000 in 3.20750246
2022-02-12 23:09:55 INFO  Benchmark_Timer - Begin run: Insertion Sort ->
50 runs with 4000 in 12.805573240000001
2022-02-12 23:09:56 INFO  Benchmark_Timer - Begin run: Insertion Sort ->
50 runs with 8000 in 51.74883344
2022-02-12 23:09:59 INFO  Benchmark_Timer - Begin run: Insertion Sort ->
50 runs with 16000 in 218.39703165999998
-----Reverse Ordered-----
2022-02-12 23:10:11 INFO  Benchmark_Timer - Begin run: Insertion Sort ->
50 runs with 500 in 0.45714666000000004
2022-02-12 23:10:11 INFO  Benchmark_Timer - Begin run: Insertion Sort ->
50 runs with 1000 in 1.69044498
2022-02-12 23:10:11 INFO  Benchmark_Timer - Begin run: Insertion Sort ->
50 runs with 2000 in 6.54711326
2022-02-12 23:10:11 INFO  Benchmark_Timer - Begin run: Insertion Sort ->
50 runs with 4000 in 26.07409834
2022-02-12 23:10:13 INFO  Benchmark_Timer - Begin run: Insertion Sort ->
50 runs with 8000 in 106.46089996
2022-02-12 23:10:19 INFO  Benchmark_Timer - Begin run: Insertion Sort ->
```

Relationship Conclusion –

Order of growth of running for insertion is –

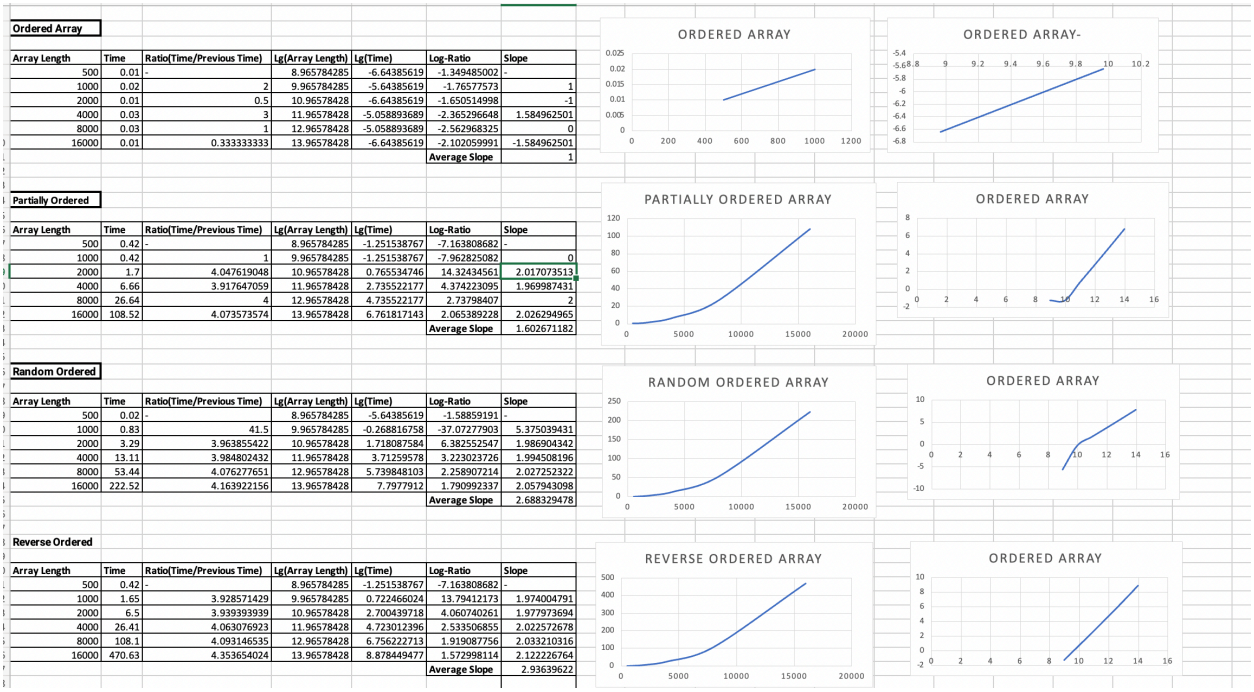
Ordered array – ≈ 1
Partially Ordered array – ≈ 1.6
Random Ordered array – ≈ 2.67
Reverse Ordered array – ≈ 2.92

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

Ordered < Partially Ordered < Randomly Ordered < Reverse Ordered

Evidence –

The below graphs plotted for the four uses cases depicts and confirms the conclusion that the above ordered elements works the same.



Ordered Array

Array Length	Time	Ratio(Time/Previous Time)	Lg(Array Length)	Lg(Time)	Log-Ratio	Slope
500	0.01	-	8.965784285	-6.64385619	-1.349485002	-
1000	0.02	2	9.965784285	-5.64385619	-1.76577573	1
2000	0.01	0.5	10.96578428	-6.64385619	-1.650514998	-1
4000	0.03	3	11.96578428	-5.058893689	-2.365296648	1.584962501
8000	0.03	1	12.96578428	-5.058893689	-2.562968325	0
16000	0.01	0.333333333	13.96578428	-6.64385619	-2.102059991	-1.584962501
					Average Slope	1

Partially Ordered

Array Length	Time	Ratio(Time/Previous Time)	Lg(Array Length)	Lg(Time)	Log-Ratio	Slope
500	0.42	-	8.965784285	-1.251538767	-7.163808682	-
1000	0.42	1	9.965784285	-1.251538767	-7.962825082	0
2000	1.7	4.047619048	10.96578428	0.765534746	14.32434561	2.017073513
4000	6.66	3.917647059	11.96578428	2.735522177	4.374223095	1.969987431
8000	26.64	4	12.96578428	4.735522177	2.73798407	2
16000	108.52	4.073573574	13.96578428	6.761817143	2.065389228	2.026294965
					Average Slope	1.602671182

Random Ordered

Array Length	Time	Ratio(Time/Previous Time)	Lg(Array Length)	Lg(Time)	Log-Ratio	Slope
500	0.02	-	8.965784285	-5.64385619	-1.58859191	-
1000	0.83	41.5	9.965784285	-0.268816758	-37.07277903	5.375039431
2000	3.29	3.963855422	10.96578428	1.718087584	6.382552547	1.986904342
4000	13.11	3.984802432	11.96578428	3.71259578	3.223023726	1.994508196
8000	53.44	4.076277651	12.96578428	5.739848103	2.258907214	2.027252322
16000	222.52	4.163922156	13.96578428	7.7977912	1.790992337	2.057943098
					Average Slope	2.688329478

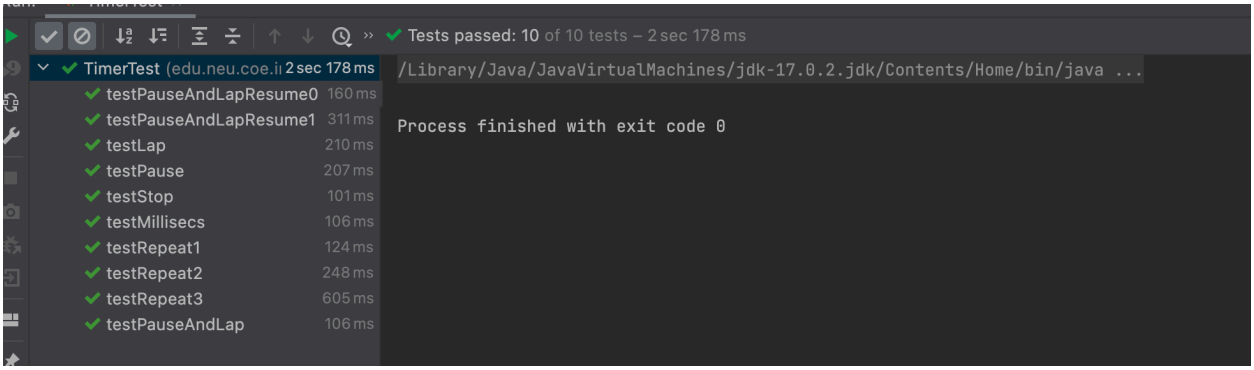
Reverse Ordered

Array Length	Time	Ratio(Time/Previous Time)	Lg(Array Length)	Lg(Time)	Log-Ratio	Slope
500	0.42	-	8.965784285	-1.251538767	-7.163808682	-
1000	1.65	3.928571429	9.965784285	0.722466024	13.79412173	1.974004791
2000	6.5	3.939393939	10.96578428	2.700439718	4.060740261	1.977973694
4000	26.41	4.063076923	11.96578428	4.723012396	2.533506855	2.022572678
8000	108.1	4.093146535	12.96578428	6.756222713	1.919087756	2.033210316
16000	470.63	4.353654024	13.96578428	8.878449477	1.572998114	2.122226764
					Average Slope	2.93639622

Unit Test Result –

Here is a screenshot of unit test results.

TimerTest



BenchmarkTest

InsertionSortTest