

# Program Structures and Algorithms

## Assignment-3

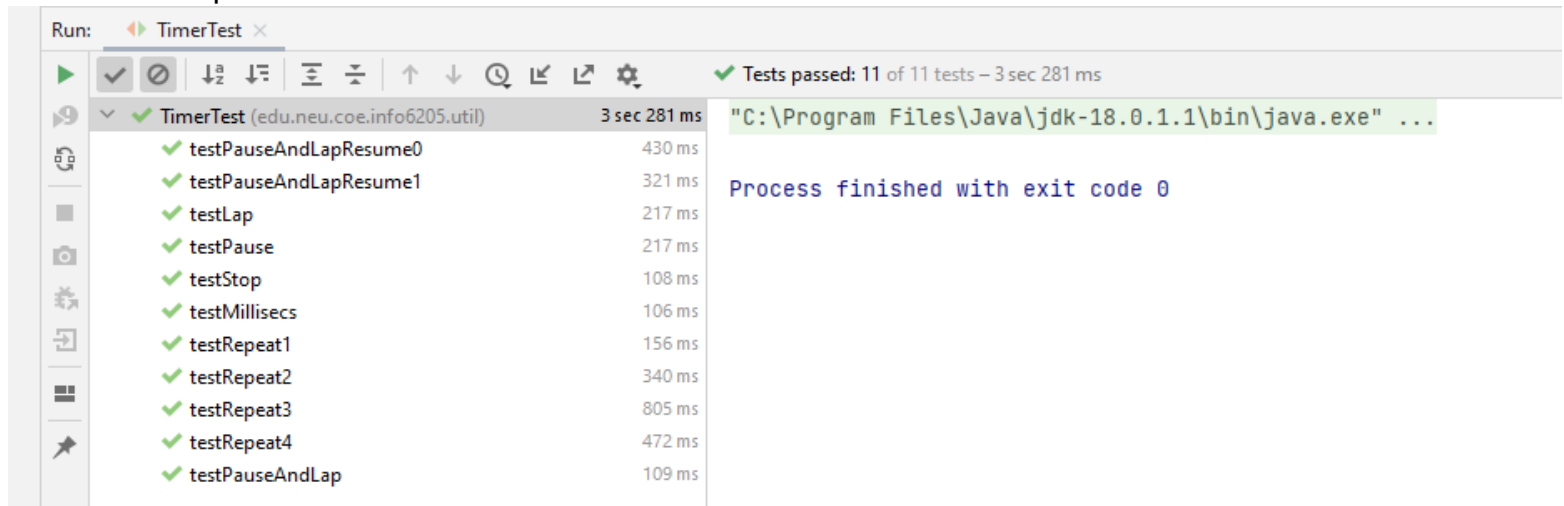
Summer-2022

Dimpleben Kanjibhai Patel – 002965372

**Task-1:** You are to implement three (3) methods (repeat, getClock, and toMillisecs) of a class called Timer. Please see the skeleton class that is 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. If you have trouble with the exact timings in the unit tests, it's quite OK (in this assignment only) to change parameters until the tests run. Different machine architectures will result in different behavior.

### Output:

TimerTest Output:

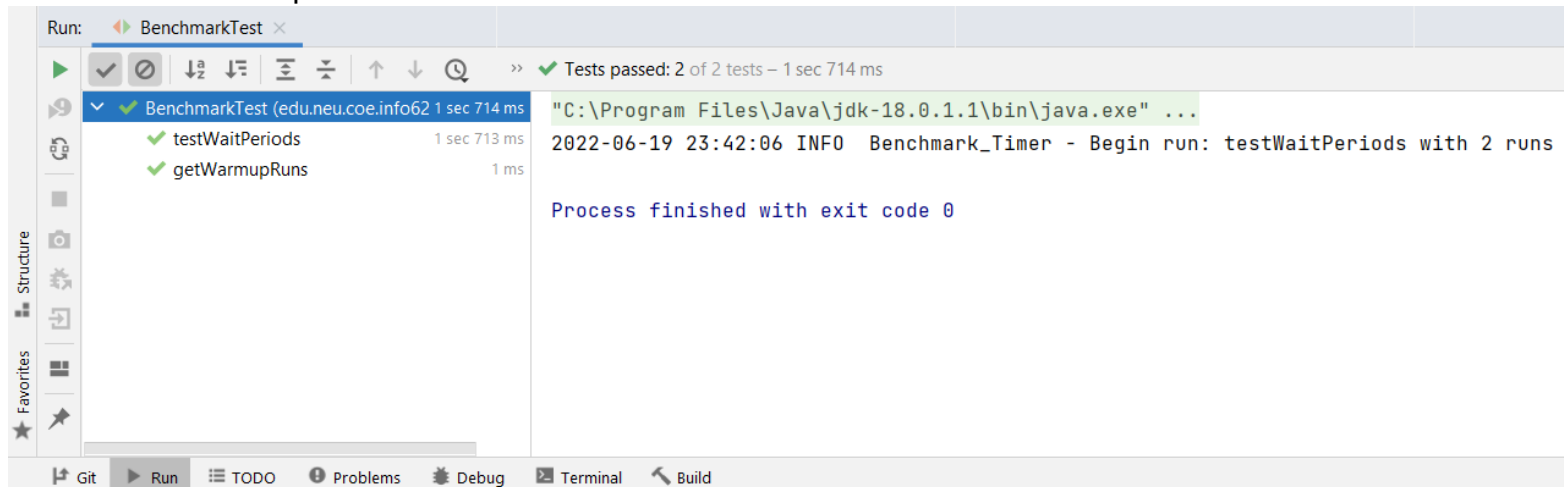


The screenshot shows the IDE's output window for the TimerTest class. The test suite passed all 11 tests in 3 seconds and 281 milliseconds. The tests and their durations are listed in the table below:

| Test Name              | Duration |
|------------------------|----------|
| testPauseAndLapResume0 | 430 ms   |
| testPauseAndLapResume1 | 321 ms   |
| testLap                | 217 ms   |
| testPause              | 217 ms   |
| testStop               | 108 ms   |
| testMillisecs          | 106 ms   |
| testRepeat1            | 156 ms   |
| testRepeat2            | 340 ms   |
| testRepeat3            | 805 ms   |
| testRepeat4            | 472 ms   |
| testPauseAndLap        | 109 ms   |

The output also shows the command used to run the tests: "C:\Program Files\Java\jdk-18.0.1.1\bin\java.exe" ... and the message "Process finished with exit code 0".

BenchmarkTest Output:



The screenshot shows the IDE's output window for the BenchmarkTest class. The test suite passed both tests in 1 second and 714 milliseconds. The tests and their durations are listed in the table below:

| Test Name       | Duration     |
|-----------------|--------------|
| testWaitPeriods | 1 sec 713 ms |
| getWarmupRuns   | 1 ms         |

The output also shows the command used to run the tests: "C:\Program Files\Java\jdk-18.0.1.1\bin\java.exe" ... and the message "Process finished with exit code 0".

**Task-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). The easiest is to use the *helper.swapStableConditional* method, continuing if it returns true, otherwise breaking the loop. Alternatively, if you are not using instrumenting, then you can write (or copy) your own compare/swap code. Either way, you must run the unit tests in *InsertionSortTest*.

## Output:

### InsertionSortTest

```
Run: InsertionSortTest x
>> Tests passed: 6 of 6 tests - 486 ms

InsertionSortTest (edu.neu.coe.info6205:486 ms)
  testMutatingInsertionSort 339 ms
  sort0 104 ms
  sort1 10 ms
  sort2 16 ms
  sort3 4 ms
  testStaticInsertionSort 13 ms

"C:\Program Files\Java\jdk-18.0.1.1\bin\java.exe" ...
2022-06-19 23:52:21 DEBUG Config - Config.get(helper, instrument) = true
2022-06-19 23:52:21 DEBUG Config - Config.get(helper, seed) = 0
2022-06-19 23:52:21 DEBUG Config - Config.get(instrumenting, copies) = true
2022-06-19 23:52:21 DEBUG Config - Config.get(instrumenting, swaps) = true
2022-06-19 23:52:21 DEBUG Config - Config.get(instrumenting, compares) = true
2022-06-19 23:52:21 DEBUG Config - Config.get(instrumenting, inversions) = 1
2022-06-19 23:52:21 DEBUG Config - Config.get(instrumenting, fixes) = true
2022-06-19 23:52:21 DEBUG Config - Config.get(instrumenting, hits) = true
2022-06-19 23:52:21 DEBUG Config - Config.get(helper, cutoff) =
Helper for InsertionSort with 4 elements
```

**Task-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.

## Ordered Array Output:

```
Run: Benchmark_Timer x

"C:\Program Files\Java\jdk-18.0.1.1\bin\java.exe" ...
Ordered Array: Insertion sort with 100 elements takes 0.015803333333333332 ms
Ordered Array: Insertion sort with 200 elements takes 0.01444 ms
Ordered Array: Insertion sort with 400 elements takes 0.016723333333333333 ms
Ordered Array: Insertion sort with 800 elements takes 0.022683333333333333 ms
Ordered Array: Insertion sort with 1600 elements takes 0.025803333333333334 ms
Ordered Array: Insertion sort with 3200 elements takes 0.048596666666666666 ms

Process finished with exit code 0
```

### Partially Ordered Array Output:

```
Run: Benchmark_Timer x
"C:\Program Files\Java\jdk-18.0.1.1\bin\java.exe" ...
Partailly Ordered Array: Insertion sort with 100 elements takes 0.011843333333333334 ms
Partailly Ordered Array: Insertion sort with 200 elements takes 0.017546666666666665 ms
Partailly Ordered Array: Insertion sort with 400 elements takes 0.026453333333333332 ms
Partailly Ordered Array: Insertion sort with 800 elements takes 0.028296666666666668 ms
Partailly Ordered Array: Insertion sort with 1600 elements takes 0.04560333333333333 ms
Partailly Ordered Array: Insertion sort with 3200 elements takes 0.060129999999999996 ms

Process finished with exit code 0
```

### Random Array Output:

```
Run: Benchmark_Timer x
"C:\Program Files\Java\jdk-18.0.1.1\bin\java.exe" ...
Random Array: Insertion sort with 100 elements takes 0.01099 ms
Random Array: Insertion sort with 200 elements takes 0.01558 ms
Random Array: Insertion sort with 400 elements takes 0.018 ms
Random Array: Insertion sort with 800 elements takes 0.023719999999999998 ms
Random Array: Insertion sort with 1600 elements takes 0.0494 ms
Random Array: Insertion sort with 3200 elements takes 0.187 ms

Process finished with exit code 0
```

### Revese Ordered Array Output:

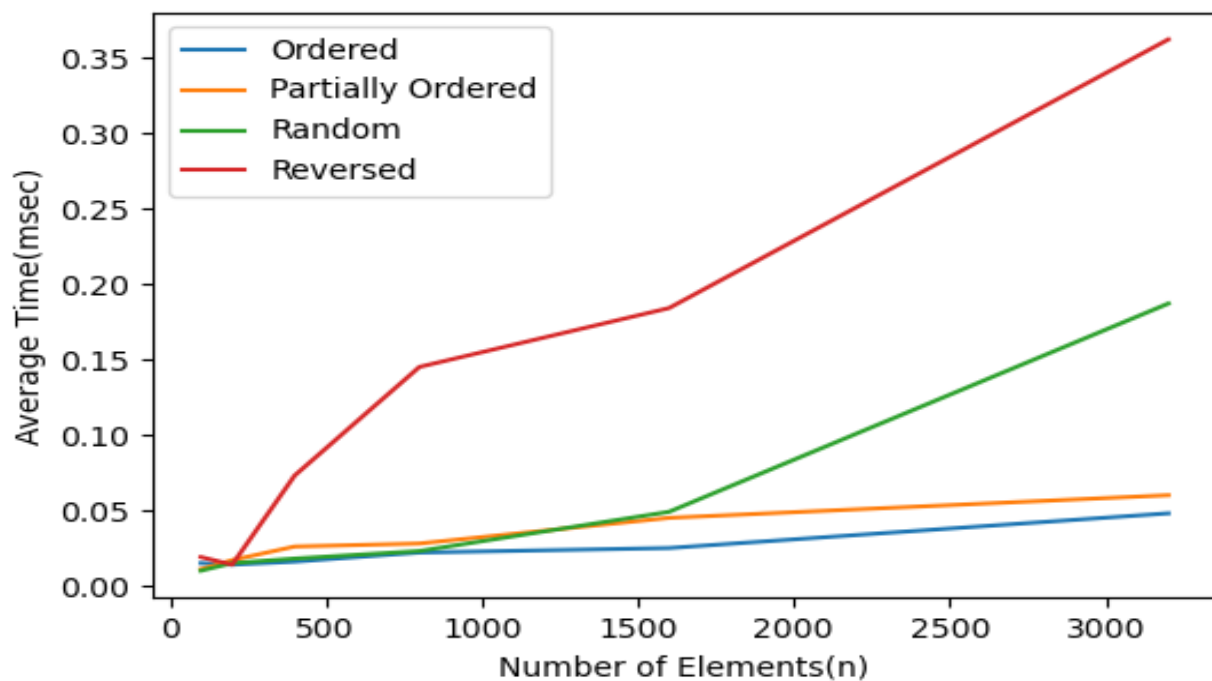
```
Run: Benchmark_Timer x
"C:\Program Files\Java\jdk-18.0.1.1\bin\java.exe" ...
Reverse Ordered Array: Insertion sort with 100 elements takes 0.0196 ms
Reverse Ordered Array: Insertion sort with 200 elements takes 0.01489 ms
Reverse Ordered Array: Insertion sort with 400 elements takes 0.0732 ms
Reverse Ordered Array: Insertion sort with 800 elements takes 0.1454 ms
Reverse Ordered Array: Insertion sort with 1600 elements takes 0.18439999999999998 ms
Reverse Ordered Array: Insertion sort with 3200 elements takes 0.3621 ms

Process finished with exit code 0
```

## Comparison Table:

| N    | Ordered Array | Partially Ordered Array | Random Array | Reversed Array |
|------|---------------|-------------------------|--------------|----------------|
| 100  | 0.015         | 0.011                   | 0.010        | 0.019          |
| 200  | 0.014         | 0.017                   | 0.015        | 0.014          |
| 400  | 0.016         | 0.026                   | 0.018        | 0.073          |
| 800  | 0.022         | 0.028                   | 0.023        | 0.145          |
| 1600 | 0.025         | 0.045                   | 0.049        | 0.184          |
| 3200 | 0.048         | 0.060                   | 0.187        | 0.362          |

## Comparison Graph:



## Conclusion :

From the above graph and analysis, we observe that the order of time taken by each different arrangement of elements in an array is -

Ordered array < partially ordered array < random array < reversed array