

Sayali Mahajan – (001576540)

# Program Structures & Algorithms

## Summer 2021

### Assignment No. 2

#### Task 1

You are to implement three methods 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. check your implementation by running the unit tests in *BenchmarkTest* and *TimerTest*.

#### Output

##### TimerTest



```
Run: TimerTest x
Tests passed: 10 of 10 tests - 2 s 367 ms
TimerTest (edu.neu.coe.inf 2 s 367 ms)
  testPauseAndLapResume0 319 ms
  testPauseAndLapResume1 316 ms
  testLap 211 ms
  testPause 211 ms
  testStop 105 ms
  testMillisecs 105 ms
  testRepeat1 130 ms
  testRepeat2 248 ms
  testRepeat3 617 ms
  testPauseAndLap 105 ms
Tests passed: 10
Process finished with exit code 0
```

##### BenchmarkTest



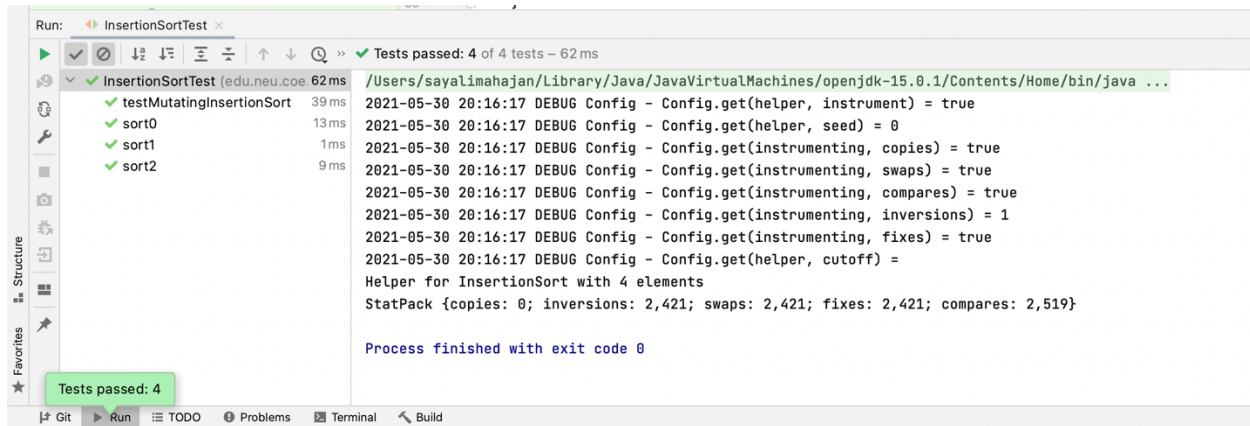
```
Run: BenchmarkTest x
Tests passed: 2 of 2 tests - 1 s 592 ms
BenchmarkTest (edu.neu.coe 1 s 592 ms)
  testWaitPeriods 1 s 592 ms
  getWarmupRuns 0 ms
Tests passed: 2
2021-05-28 17:50:52 INFO Benchmark_Timer - Begin run: testWaitPeriods with 2 runs
Process finished with exit code 0
```

## Task 2

Implement *InsertionSort* (in the *InsertionSort* class) by simply looking up the insertion code used by *Arrays.sort*. You should use the *helper.swap* method although you could also just copy that from the same source code. You should of course run the unit tests in *InsertionSortTest*.

## Output

### InsertionSortTest



```
Run: InsertionSortTest x
Tests passed: 4 of 4 tests - 62 ms

InsertionSortTest (edu.neu.coe.62 ms)
  testMutatingInsertionSort 39 ms
  sort0 13 ms
  sort1 1 ms
  sort2 9 ms

/Users/sayalimahajan/Library/Java/JavaVirtualMachines/openjdk-15.0.1/Contents/Home/bin/java ...
2021-05-30 20:16:17 DEBUG Config - Config.get(helper, instrument) = true
2021-05-30 20:16:17 DEBUG Config - Config.get(helper, seed) = 0
2021-05-30 20:16:17 DEBUG Config - Config.get(instrumenting, copies) = true
2021-05-30 20:16:17 DEBUG Config - Config.get(instrumenting, swaps) = true
2021-05-30 20:16:17 DEBUG Config - Config.get(instrumenting, compares) = true
2021-05-30 20:16:17 DEBUG Config - Config.get(instrumenting, inversions) = 1
2021-05-30 20:16:17 DEBUG Config - Config.get(instrumenting, fixes) = true
2021-05-30 20:16:17 DEBUG Config - Config.get(helper, cutoff) =
Helper for InsertionSort with 4 elements
StatPack {copies: 0; inversions: 2,421; swaps: 2,421; fixes: 2,421; compares: 2,519}

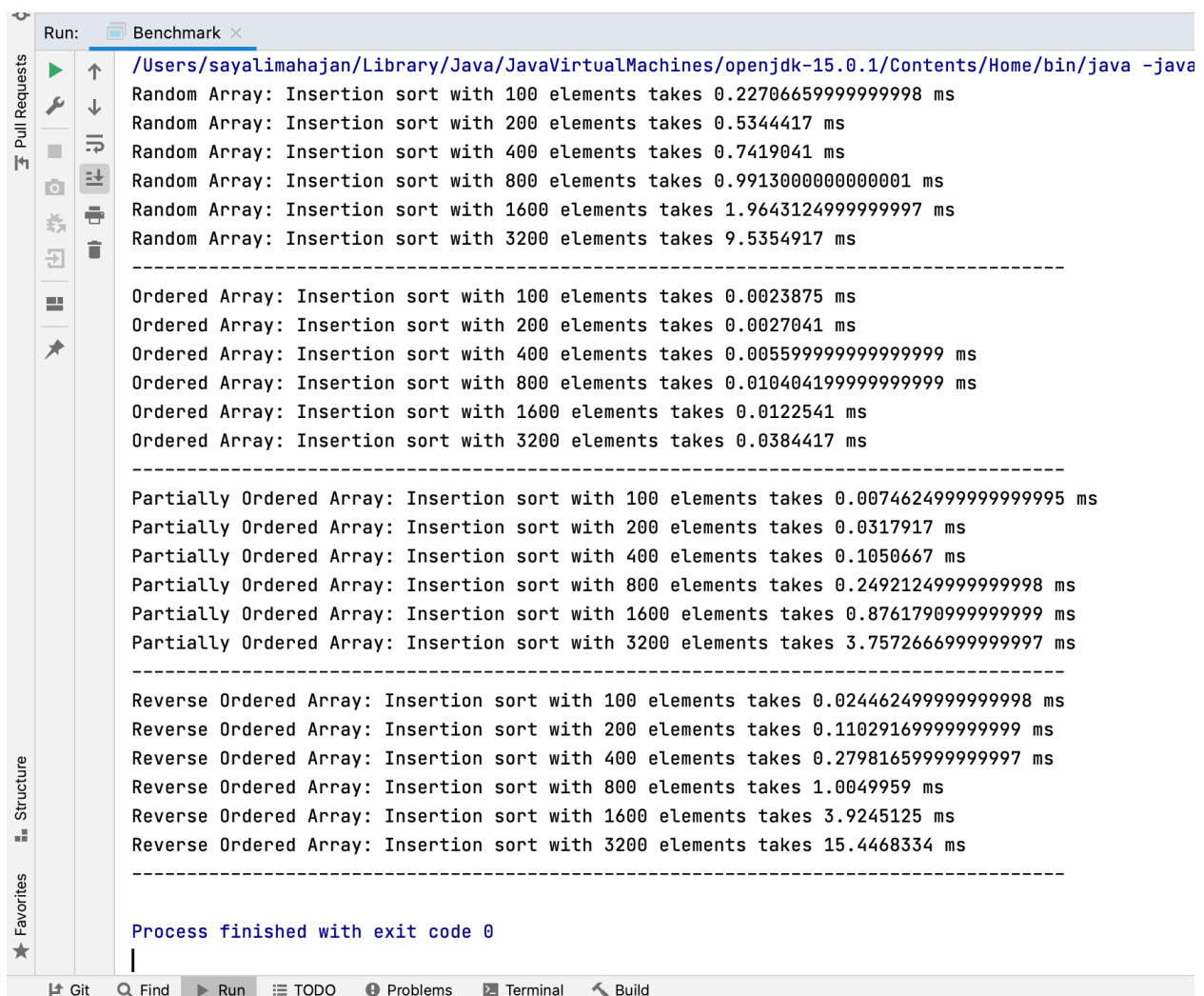
Process finished with exit code 0
```

### 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.

### Output

#### Benchmark



```
Run: Benchmark x
/Users/sayalimahajan/Library/Java/JavaVirtualMachines/openjdk-15.0.1/Contents/Home/bin/java -java
Random Array: Insertion sort with 100 elements takes 0.22706659999999998 ms
Random Array: Insertion sort with 200 elements takes 0.5344417 ms
Random Array: Insertion sort with 400 elements takes 0.7419041 ms
Random Array: Insertion sort with 800 elements takes 0.9913000000000001 ms
Random Array: Insertion sort with 1600 elements takes 1.9643124999999997 ms
Random Array: Insertion sort with 3200 elements takes 9.5354917 ms

-----

Ordered Array: Insertion sort with 100 elements takes 0.0023875 ms
Ordered Array: Insertion sort with 200 elements takes 0.0027041 ms
Ordered Array: Insertion sort with 400 elements takes 0.00559999999999999 ms
Ordered Array: Insertion sort with 800 elements takes 0.01040419999999999 ms
Ordered Array: Insertion sort with 1600 elements takes 0.0122541 ms
Ordered Array: Insertion sort with 3200 elements takes 0.0384417 ms

-----

Partially Ordered Array: Insertion sort with 100 elements takes 0.007462499999999995 ms
Partially Ordered Array: Insertion sort with 200 elements takes 0.0317917 ms
Partially Ordered Array: Insertion sort with 400 elements takes 0.1050667 ms
Partially Ordered Array: Insertion sort with 800 elements takes 0.24921249999999998 ms
Partially Ordered Array: Insertion sort with 1600 elements takes 0.8761790999999999 ms
Partially Ordered Array: Insertion sort with 3200 elements takes 3.7572666999999997 ms

-----

Reverse Ordered Array: Insertion sort with 100 elements takes 0.024462499999999998 ms
Reverse Ordered Array: Insertion sort with 200 elements takes 0.11029169999999999 ms
Reverse Ordered Array: Insertion sort with 400 elements takes 0.27981659999999997 ms
Reverse Ordered Array: Insertion sort with 800 elements takes 1.0049959 ms
Reverse Ordered Array: Insertion sort with 1600 elements takes 3.9245125 ms
Reverse Ordered Array: Insertion sort with 3200 elements takes 15.4468334 ms

-----

Process finished with exit code 0
```

## Evidence to support the conclusion

No. of Elements in Array(n)						
Array ordering	100	200	400	800	1600	3200
<b>Random</b>	0.2270666	0.5344417	0.7419041	0.9913000	1.9643125	9.5354917
<b>Ordered</b>	0.0023875	0.0027041	0.0055910	0.0104042	0.0122541	0.0384417
<b>Partially ordered</b>	0.0074625	0.0317917	0.1050667	0.2492125	0.8761791	3.7572667
<b>Reversed</b>	0.0244625	0.1102917	0.2798166	1.0049959	3.9245125	15.446833

## Graphical Representation

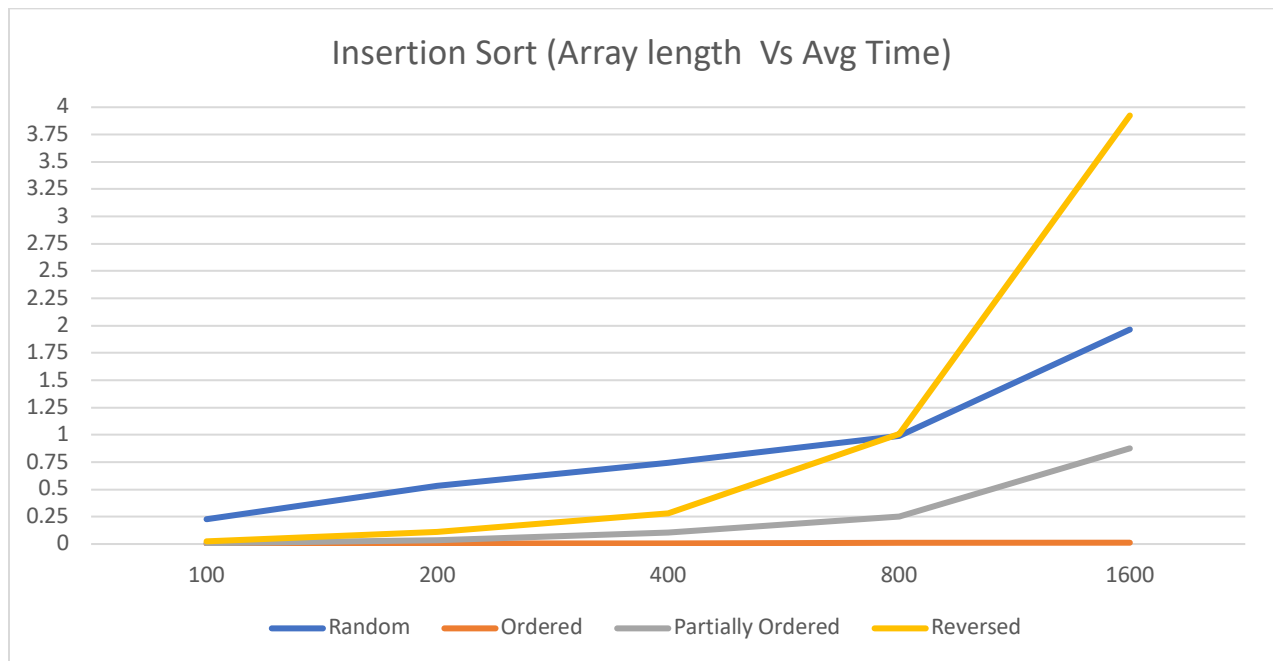


Figure 1 : X represents number of elements in Array whereas Y represents Avg time taken to sort the array

## Conclusion

1. Ordered array took minimum time (almost no time) to sort the array using insertion sort among all of types of arrays.
2. In partially ordered array, when you double the size of the array, Time taken to sort the array becomes triple.

3. Reversed array took maximum time to sort the array using insertion sort among all of types of arrays.