

# Program Structures & Algorithms

## Spring 2022

### Assignment No - 2

Name: Naina Rajan  
NUID: 002922398

#### Task

Your task for this assignment is in three parts.

- (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. The APIs of these class are as follows:

```
public interface Benchmark<T> {  
    default double run(T t, int m) {  
        return runFromSupplier(() -> t, m);  
    }  
  
    double runFromSupplier(Supplier<T> supplier, int m);  
}
```

[*Supplier* is a Java function type which supplies values of type *T* using the method: *get()*.]

```
public class Benchmark_Timer<T> implements Benchmark<T> {  
    public Benchmark_Timer(String description, UnaryOperator<T>  
        fPre, Consumer<T> fRun, Consumer<T> fPost)
```

[*Consumer<T>* is a Java function type which consumes a type *T* with the method: *accept(t)*.]

*UnaryOperator<T>* is essentially an alias of *Function<T, T>* which defines *apply(t)* which takes a *T* and returns a *T*.]

```
    public Benchmark_Timer(String description, UnaryOperator<T>  
        fPre, Consumer<T> fRun)
```

```

public Benchmark_Timer(String description, Consumer<T> fRun,
Consumer<T> fPost)
public Benchmark_Timer(String description, Consumer<T> f)
public class Timer {
... // see below for methods to be implemented...
}
public <T, U> double repeat(int n, Supplier<T> supplier,
Function<T, U> function, UnaryOperator<T> preFunction,
Consumer<U> postFunction) {
// TO BE IMPLEMENTED
}

```

[*Function<T, U>* defines a method *U apply(t: T)*, which takes a value of *T* and returns a value of *U*.]

```

private static long getClock() {
// TO BE IMPLEMENTED
}
private static double toMillisecs(long ticks) {
// TO BE IMPLEMENTED
}

```

The function to be timed, hereinafter the "target" function, is the *Consumer* function *fRun* (or just *f*) passed in to one or other of the constructors. For example, you might create a function which sorts an array with *n* elements.

The generic type *T* is that of the input to the target function.

The first parameter to the first run method signature is the parameter that will, in turn, be passed to target function. In the second signature, *supplier* will be invoked each time to get a *t* which is passed to the other run method.

The second parameter to the *run* function (*m*) is the number of times the target function will be called.

The return value from *run* is the average number of milliseconds taken for each run of the target function.

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.

- (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). 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*.
- (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.

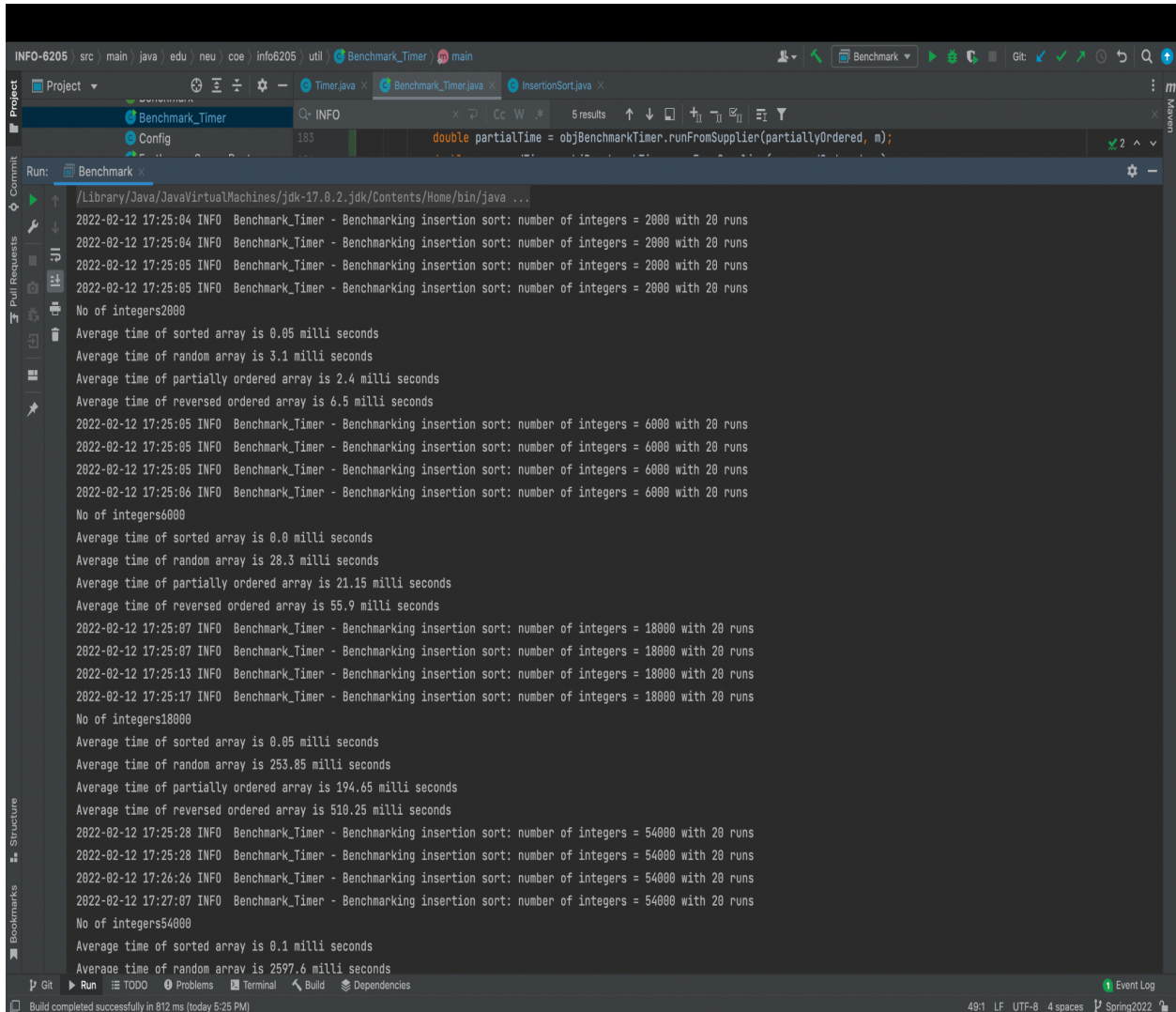
As usual, the submission will be your entire project (*clean, i.e. without the target and project folders*). There are stubs and unit tests in the repository.

Report on your observations and show screenshots of the runs and also the unit tests. Please note that you may have to adjust the required execution time for the insertion sort unit test(s) because your computer may not run at the same speed as mine.

Further notes: you should use the *System.nanoTime* method to get the clock time. This isn't guaranteed to be accurate which is one of the reasons you should run the experiment several times for each value of *n*. Also, for each invocation of *run*, run the given target function ten times to get the system "warmed up" before you start the timing properly.

The *Sort* interface takes care of copying the array when the *sort(array)* signature is called. It returns a new array as a result. The original array is unchanged. Therefore, you do not need to worry about the insertion-based sorts getting quicker because of the arrays getting more sorted (they don't).

## Output Screenshot



```
INFO-6205 | src | main | java | edu | neu | coe | info6205 | util | Benchmark_Timer | main
Project | Benchmark_Timer | Timer.java | Benchmark_Timer.java | InsertionSort.java | Maven
Run: Benchmark
/Library/Java/JavaVirtualMachines/jdk-17.0.2.jdk/Contents/Home/bin/java ...
2022-02-12 17:25:04 INFO Benchmark_Timer - Benchmarking insertion sort: number of integers = 2000 with 20 runs
2022-02-12 17:25:04 INFO Benchmark_Timer - Benchmarking insertion sort: number of integers = 2000 with 20 runs
2022-02-12 17:25:05 INFO Benchmark_Timer - Benchmarking insertion sort: number of integers = 2000 with 20 runs
2022-02-12 17:25:05 INFO Benchmark_Timer - Benchmarking insertion sort: number of integers = 2000 with 20 runs
No of integers2000
Average time of sorted array is 0.05 milli seconds
Average time of random array is 3.1 milli seconds
Average time of partially ordered array is 2.4 milli seconds
Average time of reversed ordered array is 6.5 milli seconds
2022-02-12 17:25:05 INFO Benchmark_Timer - Benchmarking insertion sort: number of integers = 6000 with 20 runs
2022-02-12 17:25:05 INFO Benchmark_Timer - Benchmarking insertion sort: number of integers = 6000 with 20 runs
2022-02-12 17:25:05 INFO Benchmark_Timer - Benchmarking insertion sort: number of integers = 6000 with 20 runs
2022-02-12 17:25:06 INFO Benchmark_Timer - Benchmarking insertion sort: number of integers = 6000 with 20 runs
No of integers6000
Average time of sorted array is 0.0 milli seconds
Average time of random array is 28.3 milli seconds
Average time of partially ordered array is 21.15 milli seconds
Average time of reversed ordered array is 55.9 milli seconds
2022-02-12 17:25:07 INFO Benchmark_Timer - Benchmarking insertion sort: number of integers = 18000 with 20 runs
2022-02-12 17:25:07 INFO Benchmark_Timer - Benchmarking insertion sort: number of integers = 18000 with 20 runs
2022-02-12 17:25:13 INFO Benchmark_Timer - Benchmarking insertion sort: number of integers = 18000 with 20 runs
2022-02-12 17:25:17 INFO Benchmark_Timer - Benchmarking insertion sort: number of integers = 18000 with 20 runs
No of integers18000
Average time of sorted array is 0.05 milli seconds
Average time of random array is 253.85 milli seconds
Average time of partially ordered array is 194.65 milli seconds
Average time of reversed ordered array is 510.25 milli seconds
2022-02-12 17:25:28 INFO Benchmark_Timer - Benchmarking insertion sort: number of integers = 54000 with 20 runs
2022-02-12 17:25:28 INFO Benchmark_Timer - Benchmarking insertion sort: number of integers = 54000 with 20 runs
2022-02-12 17:26:26 INFO Benchmark_Timer - Benchmarking insertion sort: number of integers = 54000 with 20 runs
2022-02-12 17:27:07 INFO Benchmark_Timer - Benchmarking insertion sort: number of integers = 54000 with 20 runs
No of integers54000
Average time of sorted array is 0.1 milli seconds
Average time of random array is 2597.6 milli seconds
Build completed successfully in 812 ms (today 5:25 PM) 49:1 LF UTF-8 4 spaces Spring2022
```

```
INFO-6205 src/main/java/edu/neu/coe/info6205/util/Benchmark_Timer main
Project Benchmark_Timer Benchmark_Timer.java InsertionSort.java
INFO 183 double partialTime = objBenchmarkTimer.runFromSupplier(partiallyOrdered, m);
Run: Benchmark
No of integers6000
Average time of sorted array is 0.0 milli seconds
Average time of random array is 28.3 milli seconds
Average time of partially ordered array is 21.15 milli seconds
Average time of reversed ordered array is 55.9 milli seconds
2022-02-12 17:25:07 INFO Benchmark_Timer - Benchmarking insertion sort: number of integers = 18000 with 20 runs
2022-02-12 17:25:07 INFO Benchmark_Timer - Benchmarking insertion sort: number of integers = 18000 with 20 runs
2022-02-12 17:25:13 INFO Benchmark_Timer - Benchmarking insertion sort: number of integers = 18000 with 20 runs
2022-02-12 17:25:17 INFO Benchmark_Timer - Benchmarking insertion sort: number of integers = 18000 with 20 runs
No of integers18000
Average time of sorted array is 0.05 milli seconds
Average time of random array is 253.85 milli seconds
Average time of partially ordered array is 194.65 milli seconds
Average time of reversed ordered array is 510.25 milli seconds
2022-02-12 17:25:28 INFO Benchmark_Timer - Benchmarking insertion sort: number of integers = 54000 with 20 runs
2022-02-12 17:25:28 INFO Benchmark_Timer - Benchmarking insertion sort: number of integers = 54000 with 20 runs
2022-02-12 17:26:26 INFO Benchmark_Timer - Benchmarking insertion sort: number of integers = 54000 with 20 runs
2022-02-12 17:27:07 INFO Benchmark_Timer - Benchmarking insertion sort: number of integers = 54000 with 20 runs
No of integers54000
Average time of sorted array is 0.1 milli seconds
Average time of random array is 2597.6 milli seconds
Average time of partially ordered array is 1832.55 milli seconds
Average time of reversed ordered array is 4684.45 milli seconds
2022-02-12 17:28:49 INFO Benchmark_Timer - Benchmarking insertion sort: number of integers = 162000 with 20 runs
2022-02-12 17:28:49 INFO Benchmark_Timer - Benchmarking insertion sort: number of integers = 162000 with 20 runs
2022-02-12 17:40:37 INFO Benchmark_Timer - Benchmarking insertion sort: number of integers = 162000 with 20 runs
2022-02-12 17:47:26 INFO Benchmark_Timer - Benchmarking insertion sort: number of integers = 162000 with 20 runs
No of integers162000
Average time of sorted array is 0.45 milli seconds
Average time of random array is 32390.1 milli seconds
Average time of partially ordered array is 18586.0 milli seconds
Average time of reversed ordered array is 46563.75 milli seconds

Process finished with exit code 0
Git Run TODO Problems Terminal Build Dependencies
Build completed successfully in 812 ms (today 5:25 PM) 49:1 LF UTF-8 4 spaces Spring2022
```

## Relationship Conclusion

Array Size	Sorted Array (ms)	Random Array(ms)	Partially Ordered Array(ms)	Reversed Ordered Array(ms)
2000	0.05	3.1	2.4	6.5
6000	0	28.3	21.15	55.9
18000	0.05	253.85	194.65	510.25
54000	0.1	2597.6	1832.55	4684.45
162000	0.45	32390.1	18586	46563.75

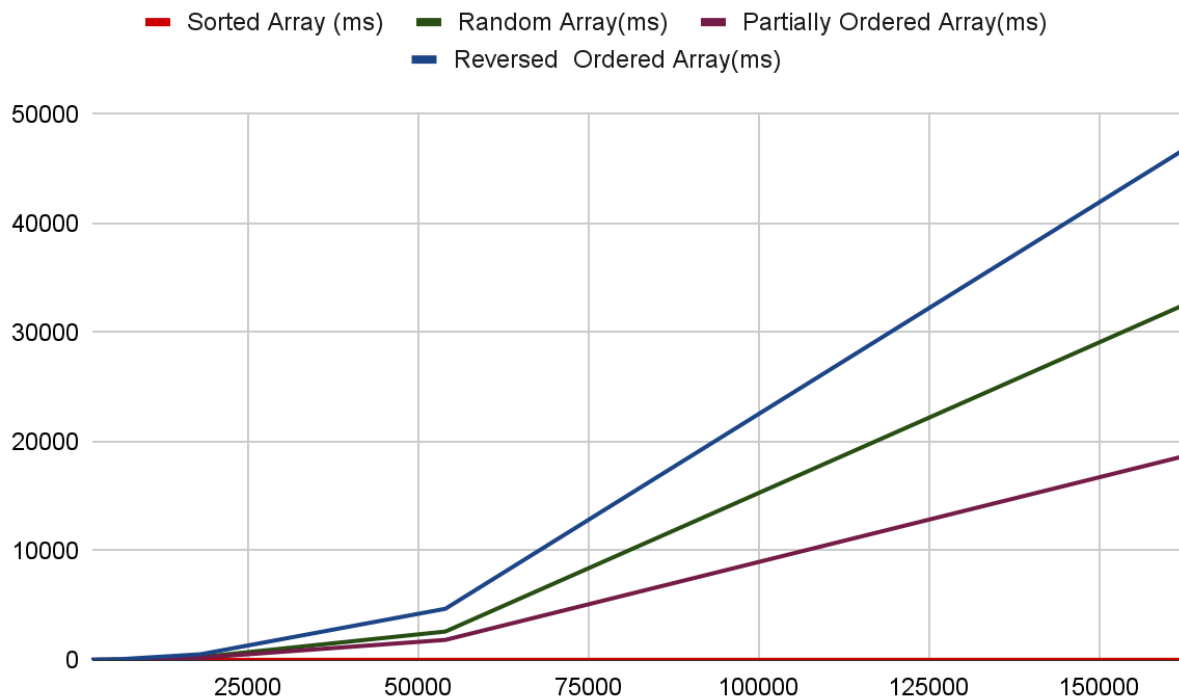
1. From the above Output Screenshot and Table, we can conclude:

Reverse Order > Random Order > Partially sorted > Sorted  
 $O(N^2)$                        $O(N^2)$                        $O(N^2)$                        $O(N)$

2. After experimental values obtained we can observe that “**Reverse Sorted Array**” takes the maximum time while the “**Sorted Array**” takes the minimum amount of time.

## Evidence/Graph

Array Size	Sorted Array (ms)	Random Array(ms)	Partially Ordered Array(ms)	Reversed Ordered Array(ms)
2000	0.05	3.1	2.4	6.5
6000	0	28.3	21.15	55.9
18000	0.05	253.85	194.65	510.25
54000	0.1	2597.6	1832.55	4684.45
162000	0.45	32390.1	18586	46563.75





# Unit Tests Result

```
Run: InsertionSortTest x
Tests passed: 6 of 6 tests - 66ms

InsertionSortTest (edu.neu.coe.info6205.sort.elementary) 66ms
  testMutatingInsertionSort 59ms
  sort0 4ms
  sort1 0ms
  sort2 2ms
  sort3 1ms
  testStaticInsertionSort 0ms

/Library/Java/JavaVirtualMachines/jdk-17.0.2.jdk/Contents/Home/bin/java ...
2022-02-12 18:41:11 DEBUG Config - Config.get(helper, instrument) = true
2022-02-12 18:41:11 DEBUG Config - Config.get(helper, seed) = 0
2022-02-12 18:41:11 DEBUG Config - Config.get(instrumenting, copies) = true
2022-02-12 18:41:11 DEBUG Config - Config.get(instrumenting, swaps) = true
2022-02-12 18:41:11 DEBUG Config - Config.get(instrumenting, compares) = true
2022-02-12 18:41:11 DEBUG Config - Config.get(instrumenting, inversions) = 1
2022-02-12 18:41:11 DEBUG Config - Config.get(instrumenting, fixes) = true
2022-02-12 18:41:11 DEBUG Config - Config.get(instrumenting, hits) = true
2022-02-12 18:41:11 DEBUG Config - Config.get(helper, cutoff) =
Helper for InsertionSort with 4 elements
StatPack {hits: 9,684; copies: 0; inversions: 2,421; swaps: 2,421; fixes: 2,421; compares: 2,519}
StatPack {hits: 19,800; copies: 0; inversions: 4,950; swaps: 4,950; fixes: 4,950; compares: 4,950}

Process finished with exit code 0
```

```
Run: TimerTest x
Tests passed: 10 of 10 tests - 2 sec 171ms

TimerTest (edu.neu.coe.info6205.util) 2sec 171ms
  testPauseAndLapResume0 155ms
  testPauseAndLapResume1 316ms
  testLap 211ms
  testPause 208ms
  testStop 105ms
  testMillisecs 102ms
  testRepeat1 125ms
  testRepeat2 241ms
  testRepeat3 602ms
  testPauseAndLap 106ms

/Library/Java/JavaVirtualMachines/jdk-17.0.2.jdk/Contents/Home/bin/java ...
Process finished with exit code 0
```

```
Run: BenchmarkTest x
Tests passed: 2 of 2 tests - 1 sec 397ms

BenchmarkTest (edu.neu.coe.info6205.util) 1sec 397ms
  testWaitPeriods 1sec 396ms
  getWarmupRuns 1ms

/Library/Java/JavaVirtualMachines/jdk-17.0.2.jdk/Contents/Home/bin/java ...
2022-02-12 18:45:04 INFO Benchmark_Timer - testWaitPeriods with 2 runs

Process finished with exit code 0
```



**Git Repository -**

<https://github.com/Naina-NEU/INFO6205/commit/520035576f931cd40db90cd40438fd46aed73a87>