

# Program Structures & Algorithms Spring 2023 Assignment No. 3

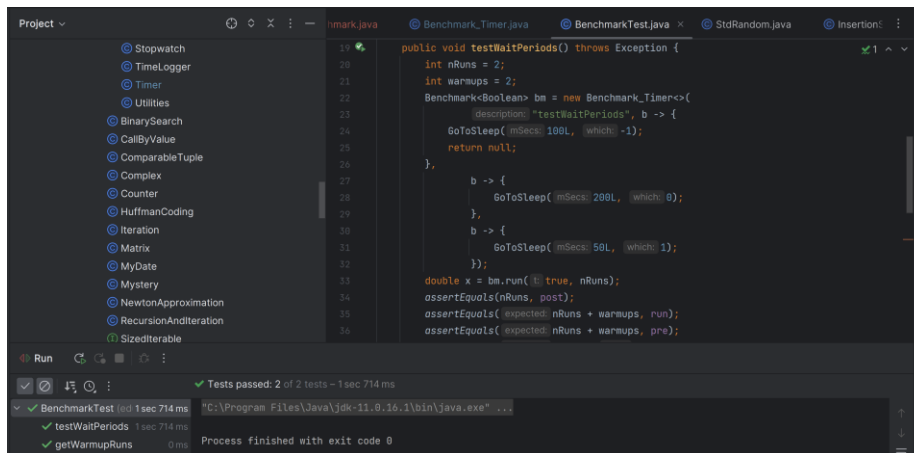
Name: Srikanth Nandikonda

NUID: 002737724

## TASK:

- 1) Implement three method of Timer class
- 2) Implement Insertion Sort
- 3) Benchmarking for insertion sort

## Unit Test Case:



```
public void testWaitPeriods() throws Exception {
    int nRuns = 2;
    int warmups = 2;
    Benchmark<Boolean> bm = new Benchmark_Timer<>(
        description: "testWaitPeriods", b -> {
            GoToSleep(mSecs: 100L, which: -1);
            return null;
        });
    b -> {
        GoToSleep(mSecs: 200L, which: 0);
    },
    b -> {
        GoToSleep(mSecs: 50L, which: 1);
    });
    double x = bm.run(1, true, nRuns);
    assertEquals(nRuns, post);
    assertEquals(expected: nRuns + warmups, run);
    assertEquals(expected: nRuns + warmups, pre);
}
```

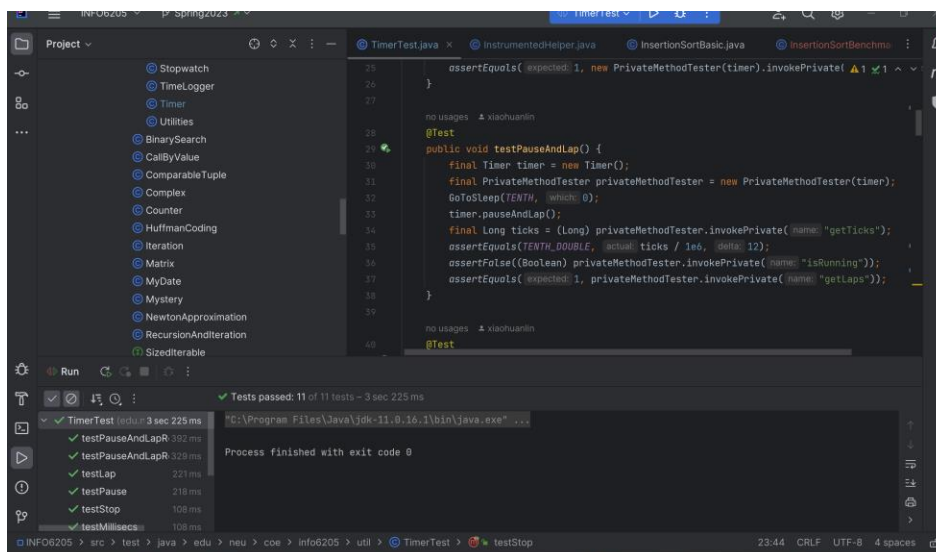
Tests passed: 2 of 2 tests - 1 sec 714 ms

BenchmarkTest 1 sec 714 ms

testWaitPeriods 1 sec 714 ms

getWarmupRuns 0 ms

Process finished with exit code 0



```
public void testPauseAndLap() {
    final Timer timer = new Timer();
    final PrivateMethodTester privateMethodTester = new PrivateMethodTester(timer);
    GoToSleep(TENTH, which: 0);
    timer.pauseAndLap();
    final Long ticks = (Long) privateMethodTester.invokePrivate("getTicks");
    assertEquals(TENTH.DOUBLE, actual ticks / 100, delta: 12);
    assertFalse((Boolean) privateMethodTester.invokePrivate("isRunning"));
    assertEquals(expected: 1, privateMethodTester.invokePrivate("getLaps"));
}

@Test
public void testLap() {
    // ...
}
```

Tests passed: 11 of 11 tests - 3 sec 225 ms

TimerTest 3 sec 225 ms

testPauseAndLap 392 ms

testPauseAndLap 329 ms

testLap 221 ms

testPause 216 ms

testStop 108 ms

testMilliSecs 108 ms

Process finished with exit code 0

```

package edu.neu.coe.info6205.sort.elementary;

import java.util.*;

public class InsertionSortTest {

    no usages: 1 xiaohuimin +1

    @test
    public void sort0() throws Exception {
        final List<Integer> list = new ArrayList<>();
        list.add(1);
        list.add(2);
        list.add(3);
        list.add(4);
        Integer[] xs = list.toArray(new Integer[0]);
        final Config config = Config.setupConfig("true", "8", "inversions: 1");
        Helper<Integer> helper = HelperFactory.create("InsertionSort", list.size());
    }
}

```

Tests passed: 6 of 6 tests - 406 ms

- testMutatingInsert: 22.4 ms
- sort0: 40 ms
- sort1: 27 ms
- sort2: 11 ms
- sort3: 2 ms
- testStaticInsertionSort: 2 ms

## Conclusion

- In case of ordered arrays, the time taken by the algorithm is less compared to other cases as the elements are in a sorted order and the number of operations needed to sort the array is less.
- In the case of random arrays, the time taken by the algorithm increases as the number of elements in the array increases and this is close to the  $T = aN^{1.777}$
- In the case of partially sorted arrays, the time taken is intermediate between the ordered and reversed arrays as the elements are partially sorted, and the number of operations needed to sort the array is intermediate.  $T = aN^{1.777}$
- In the case of reversed arrays, the time taken by the algorithm is highest as the elements are in the reverse order and the number of operations needed to sort the array is highest. This is close to the  $N^2$  time complexity.  $T = aN^{1.9034}$

**Ordered < Partial-ordered < Random < Reverse order**

## Evidence

Following data is collected after calling Insertion sort method by varying array size from 500 and doubling the array size till 32000. Mean time is calculated after running 50 times.

```

"C:\Program Files\Java\jdk-11.0.16.1\bin\java.exe" ...

```

Items	Random	Ordered	Partial	Reverse
500	1.672386	0.824456	1.064938	1.517074
1000	1.849216	0.546914	2.396808	4.281678
2000	5.642682	0.519566	4.582608	13.071038
4000	36.451446	0.98396	25.403782	60.579666
8000	99.771728	1.100052	141.949168	194.523164
16000	410.501706	0.628752	298.550246	845.388672
32000	1864.819106	0.79482	1336.759452	4050.624084

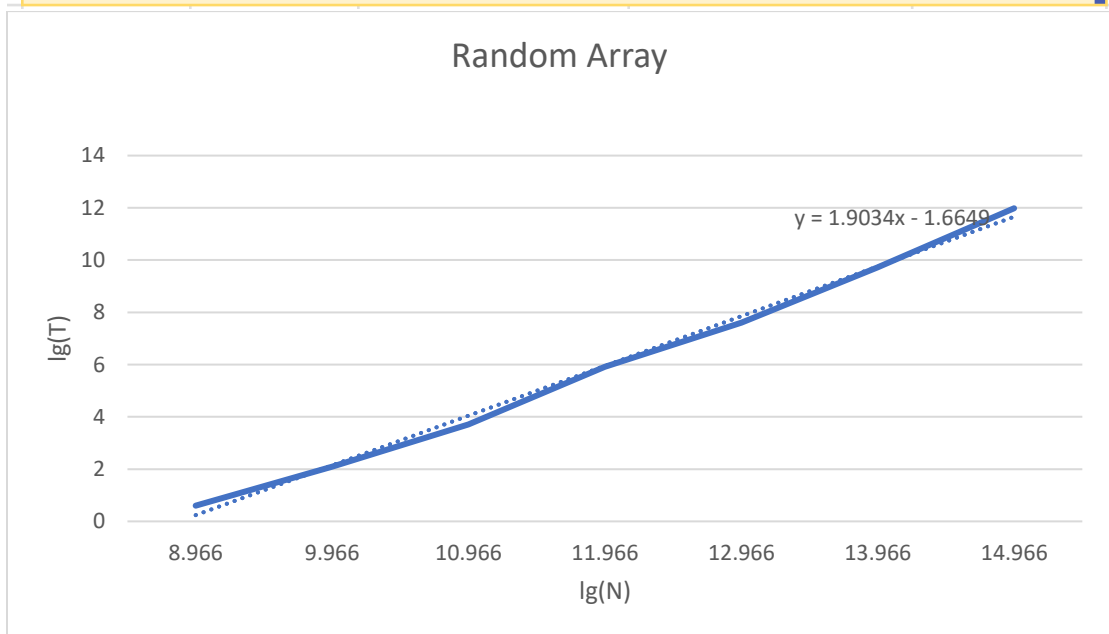
```

6205 > src > main > java > edu > neu > coe > info6205 > util > Timer > repeat

```

# 1) Elements of the array in Random way

Random					
N	T	lg(N)	lg(T)	Log Ratio	
500	1.672386	8.966	0.741907871		
1000	1.849216	9.966	0.88691375	0.1450059	
2000	5.642682	10.966	2.496381047	1.6094673	
4000	36.45145	11.966	5.187904141	2.6915231	
8000	99.77173	12.966	6.640559156	1.452655	
16000	410.5017	13.966	8.681244407	2.0406853	
32000	1864.819	14.966	10.86481998	2.1835756	



lg(T) vs lg(N) graph

Equation

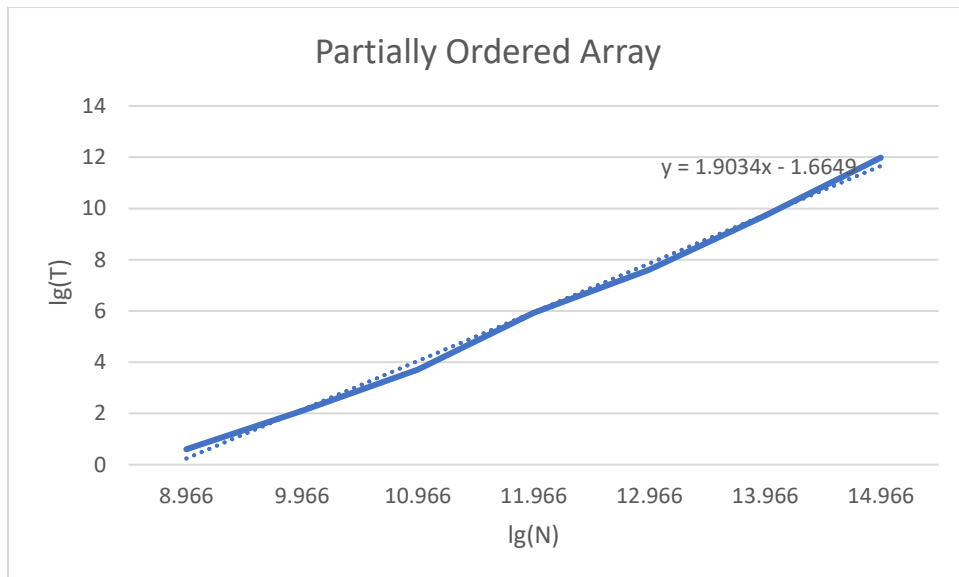
$$\lg(T) = 1.777\lg(N) - 2.086$$

Equivalent

$$T = aN^{1.777}$$

## 2) Elements of the array in partial ordered way

Partially Ordered				
N	T	lg(N)	lg(T)	Log Ratio
500	1.064938	8.966	0.09076944	
1000	2.396808	9.966	1.261114344	1.1703449
2000	4.582608	10.966	2.196168882	0.9350545
4000	25.40378	11.966	4.66697139	2.4708025
8000	141.9492	12.966	7.149230583	2.4822592
16000	298.5502	13.966	8.221829947	1.0725994
32000	1336.759	14.966	10.38452416	2.1626942



lg(T) vs lg(N) graph

Equation

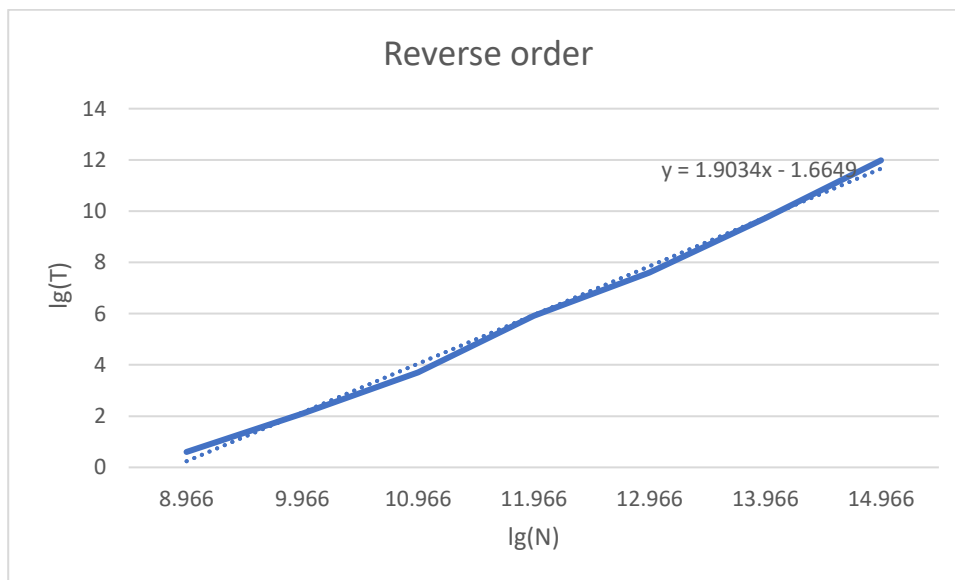
$$\lg(T) = 1.777\lg(N) - 2.255$$

Equivalent

$$T = aN^{1.777}$$

### 3) Elements of the array in reverse ordered way

Reverse Ordered				
N	T	lg(N)	lg(T)	Log Ratio
500	1.517074	8.966	0.601291459	
1000	4.281678	9.966	2.098176303	1.4968848
2000	13.07104	10.966	3.708301808	1.6101255
4000	60.57967	11.966	5.920761719	2.2124599
8000	194.5232	12.966	7.603798153	1.6830364
16000	845.3887	13.966	9.723470971	2.1196728
32000	4050.624	14.966	11.98392849	2.2604575



**$\lg(T)$  vs  $\lg(N)$  graph**

Equation

$$\lg(T) = 1.9034\lg(N) - 1.6649$$

Equivalent

$$T = aN^{1.9034}$$