Program Structures and Algorithms
Spring 2023(SEC –1)

NAME: Manish Shivaprasad
NUID:002766390

## 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.

(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.

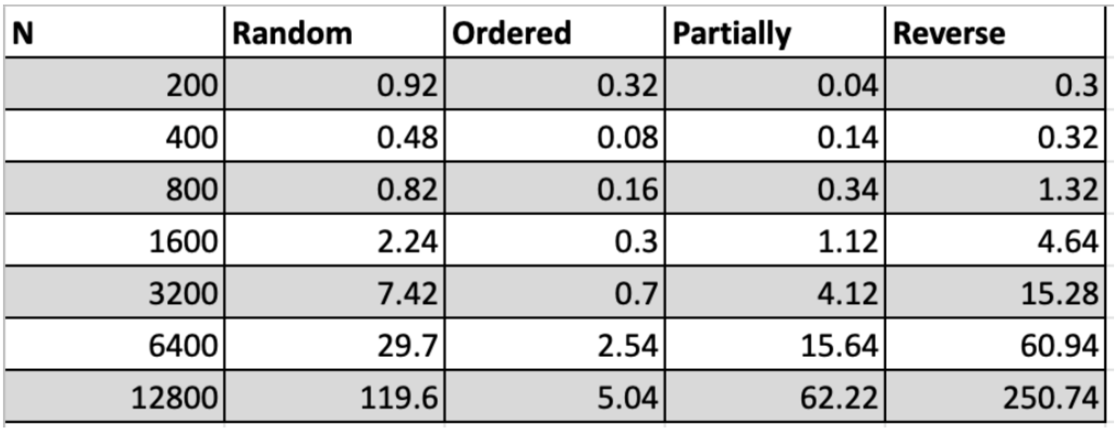
(Part 3) Implement a main program (or you could do it via your own unit tests) to actually run the following benchmarks: measure the running times of this sort, using four different initial array ordering situations: random, ordered, partially-ordered and reverse-ordered.

## Relationship Conclusion

- Time complexity of Insertion sort algorithm generally is O(N^2).
- Time Complexity for an Ordered array is O(N) which is the best case time complexity
- Time Complexity for an Reversed ordered array and Random ordered array is O(N^2) which is the worst case time complexity.
- Time Complexity for partially ordered array is in between O(N^2) and O(N).
- The order of growth is
  Ordered<Partially Ordered<Randomly ordered<Reverse Ordered.

## Evidence to support that conclusion

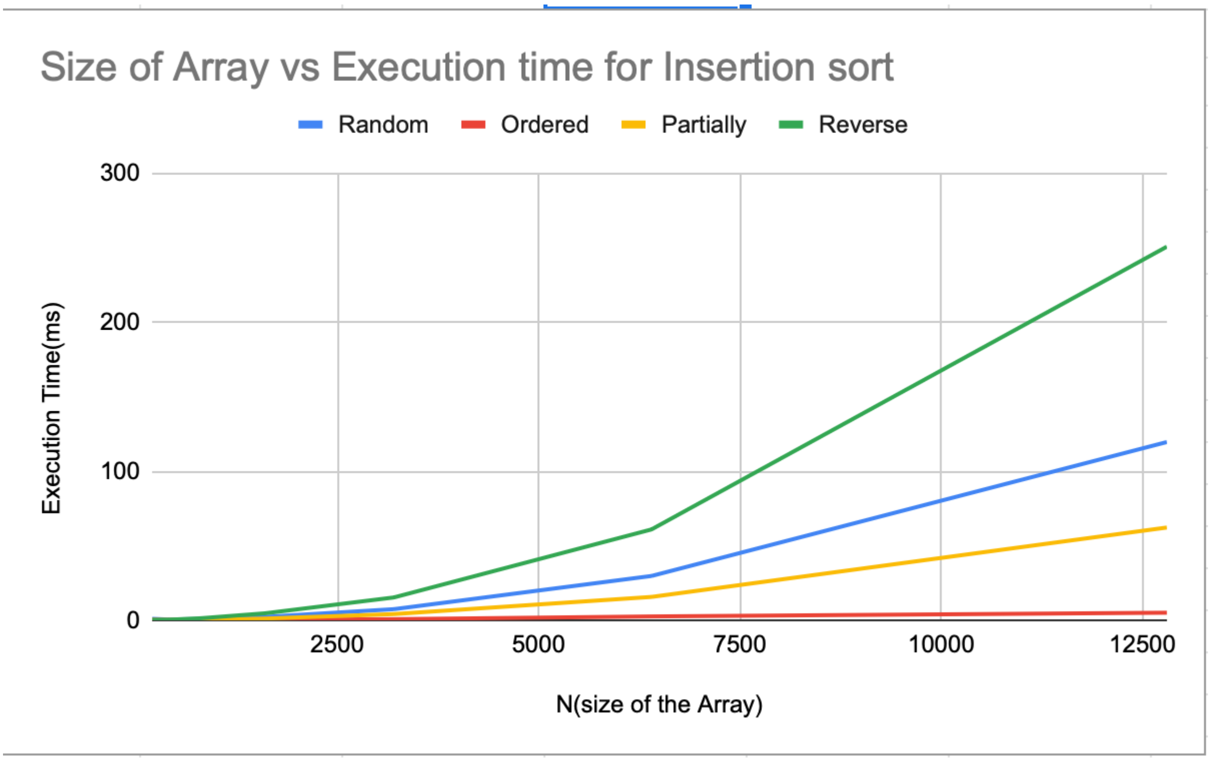
Below is the spreadsheet that provides execution time of randomly ordered, partially ordered, Reverse ordered, ordered for various array sizes.



| N | Random | Ordered | Partially | Reverse |
|---|---|---|---|---|
| 200 | 0.92 | 0.32 | 0.04 | 0.3 |
| 400 | 0.48 | 0.08 | 0.14 | 0.32 |
| 800 | 0.82 | 0.16 | 0.34 | 1.32 |
| 1600 | 2.24 | 0.3 | 1.12 | 4.64 |
| 3200 | 7.42 | 0.7 | 4.12 | 15.28 |
| 6400 | 29.7 | 2.54 | 15.64 | 60.94 |
| 12800 | 119.6 | 5.04 | 62.22 | 250.74 |

It can be inferred from the above data that execution time is highest in case of reverse ordered list and least in case of ordered list. The execution time is obtained for various values of N(Array Sizes) using doubling method.

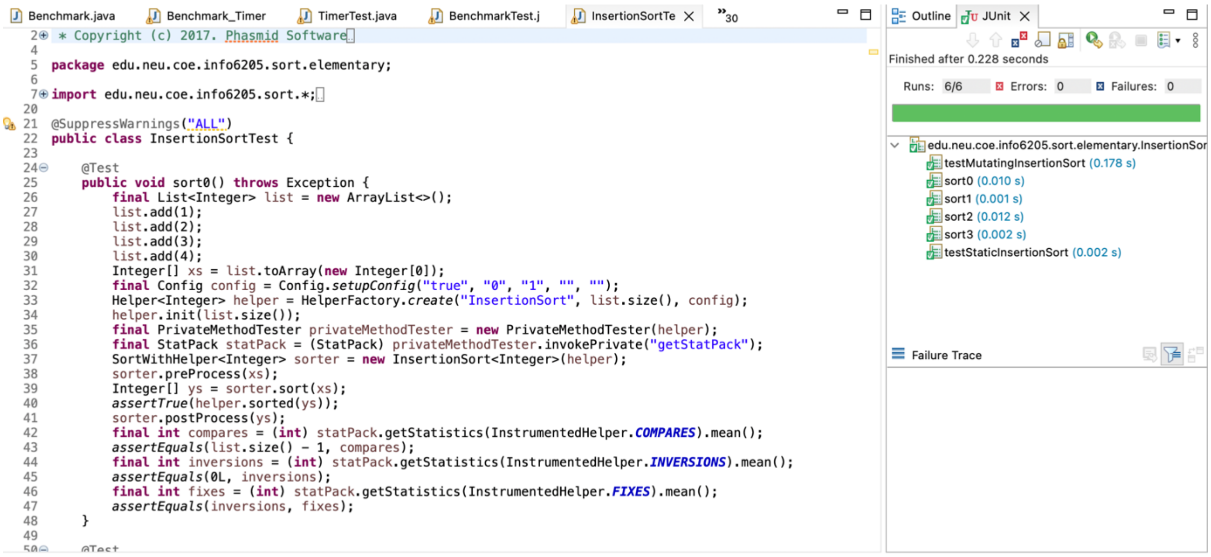
**Graphical Representation**



The above graph shows the graphical representation of the spreadsheet. It can be inferred that reverse ordered list takes most time and ordered array takes least time.

**Unit Test Screenshots**

## InsertionSort Test cases.



```java
 * Copyright (c) 2017. Phasmid Software

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

import edu.neu.coe.info6205.sort.*;

@SuppressWarnings("ALL")
public class InsertionSortTest {

    @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", "0", "1", "", "");
        Helper<Integer> helper = HelperFactory.create("InsertionSort", list.size(), config);
        helper.init(list.size());
        final PrivateMethodTester privateMethodTester = new PrivateMethodTester(helper);
        final StatPack statPack = (StatPack) privateMethodTester.invokePrivate("getStatPack");
        SortWithHelper<Integer> sorter = new InsertionSort<Integer>(helper);
        sorter.preProcess(xs);
        Integer[] ys = sorter.sort(xs);
        assertTrue(helper.sorted(ys));
        sorter.postProcess(ys);
        final int compares = (int) statPack.getStatistics(InstrumentedHelper.COMPARES).mean();
        assertEquals(list.size() - 1, compares);
        final int inversions = (int) statPack.getStatistics(InstrumentedHelper.INVERSIONS).mean();
        assertEquals(0L, inversions);
        final int fixes = (int) statPack.getStatistics(InstrumentedHelper.FIXES).mean();
        assertEquals(inversions, fixes);
    }

    @Test
```

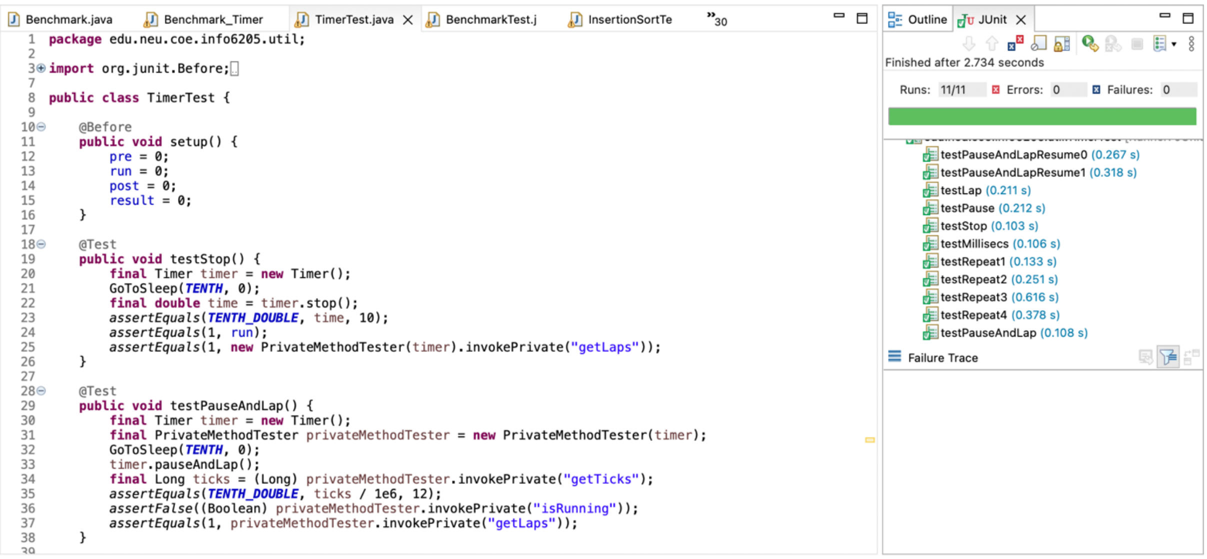
```
Finished after 0.228 seconds

Runs:  6/6    Errors:  0    Failures:  0

edu.neu.coe.info6205.sort.elementary.InsertionSor
    testMutatingInsertionSort (0.178 s)
    sort0 (0.010 s)
    sort1 (0.001 s)
    sort2 (0.012 s)
    sort3 (0.002 s)
    testStaticInsertionSort (0.002 s)

Failure Trace
```

## TimerTest Test cases



```java
package edu.neu.coe.info6205.util;

import org.junit.Before;

public class TimerTest {

    @Before
    public void setup() {
        pre = 0;
        run = 0;
        post = 0;
        result = 0;
    }

    @Test
    public void testStop() {
        final Timer timer = new Timer();
        GoToSleep(TENTH, 0);
        final double time = timer.stop();
        assertEquals(TENTH_DOUBLE, time, 10);
        assertEquals(1, run);
        assertEquals(1, new PrivateMethodTester(timer).invokePrivate("getLaps"));
    }

    @Test
    public void testPauseAndLap() {
        final Timer timer = new Timer();
        final PrivateMethodTester privateMethodTester = new PrivateMethodTester(timer);
        GoToSleep(TENTH, 0);
        timer.pauseAndLap();
        final Long ticks = (Long) privateMethodTester.invokePrivate("getTicks");
        assertEquals(TENTH_DOUBLE, ticks / 1e6, 12);
        assertFalse((Boolean) privateMethodTester.invokePrivate("isRunning"));
        assertEquals(1, privateMethodTester.invokePrivate("getLaps"));
    }
```

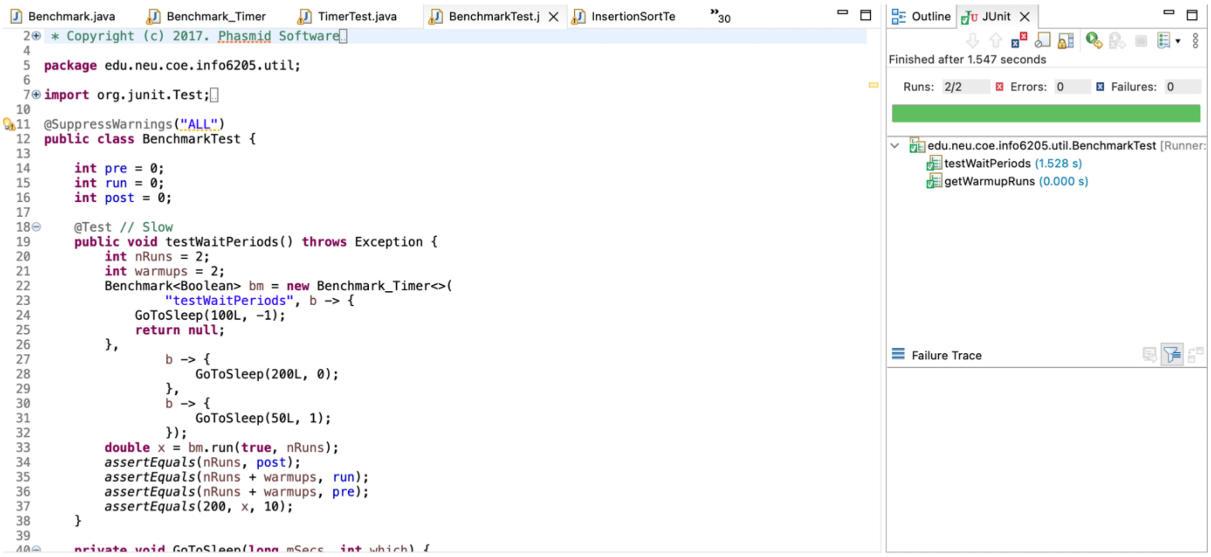
```
Finished after 2.734 seconds

Runs:  11/11    Errors:  0    Failures:  0

    testPauseAndLapResume0 (0.267 s)
    testPauseAndLapResume1 (0.318 s)
    testLap (0.211 s)
    testPause (0.212 s)
    testStop (0.103 s)
    testMillisecs (0.106 s)
    testRepeat1 (0.133 s)
    testRepeat2 (0.251 s)
    testRepeat3 (0.616 s)
    testRepeat4 (0.378 s)
    testPauseAndLap (0.108 s)

Failure Trace
```

## BenchmarkTest Testcases



```java
 * Copyright (c) 2017. Phasmid Software

package edu.neu.coe.info6205.util;

import org.junit.Test;

@SuppressWarnings("ALL")
public class BenchmarkTest {

    int pre = 0;
    int run = 0;
    int post = 0;

    @Test // Slow
    public void testWaitPeriods() throws Exception {
        int nRuns = 2;
        int warmups = 2;
        Benchmark<Boolean> bm = new Benchmark_Timer<>(
                "testWaitPeriods", b -> {
            GoToSleep(100L, -1);
            return null;
        },
                b -> {
                    GoToSleep(200L, 0);
                },
                b -> {
                    GoToSleep(50L, 1);
                });
        double x = bm.run(true, nRuns);
        assertEquals(nRuns, post);
        assertEquals(nRuns + warmups, run);
        assertEquals(nRuns + warmups, pre);
        assertEquals(200, x, 10);
    }

    private void GoToSleep(long mSecs, int which) {
```

Finished after 1.547 seconds

Runs: 2/2 ☒ Errors: 0 ☒ Failures: 0

```
edu.neu.coe.info6205.util.BenchmarkTest [Runner:
    testWaitPeriods (1.528 s)
    getWarmupRuns (0.000 s)
```