INFO 6205 – Section 01

Giang Vu – NUID: 001537937

# Assignment 2: Benchmarking/ Insertion Sort

1. Task

Your task for this assignment is in three parts.

- (Part 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. 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);
}

public class Benchmark_Timer<T> implements Benchmark<T> {

public Benchmark_Timer(String description, UnaryOperator<T> fPre, Consumer<T> fRun, C
onsumer<T> fPost)

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, Una
ryOperator<T> preFunction, Consumer<U> postFunction) {
// TO BE IMPLEMENTED
}

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*.

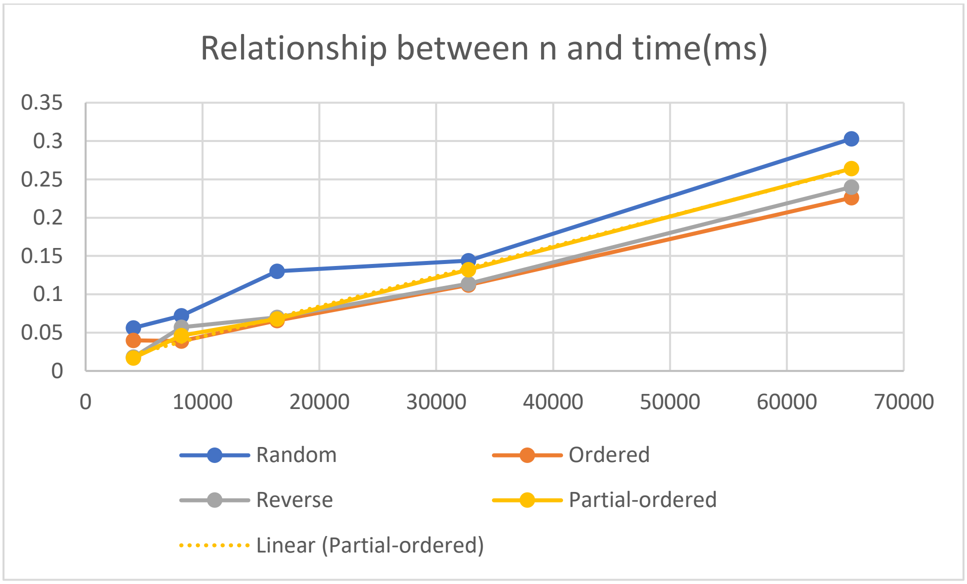
- (Part 2) Implement *InsertionSort* (in the *InsertionSort* class) by simply looking up the insertion code used by *Arrays.sort.* You should use the *helper.swap* method although you could also just copy that from the same source code. You should of course run the unit tests in *InsertionSortTest*.
- (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.

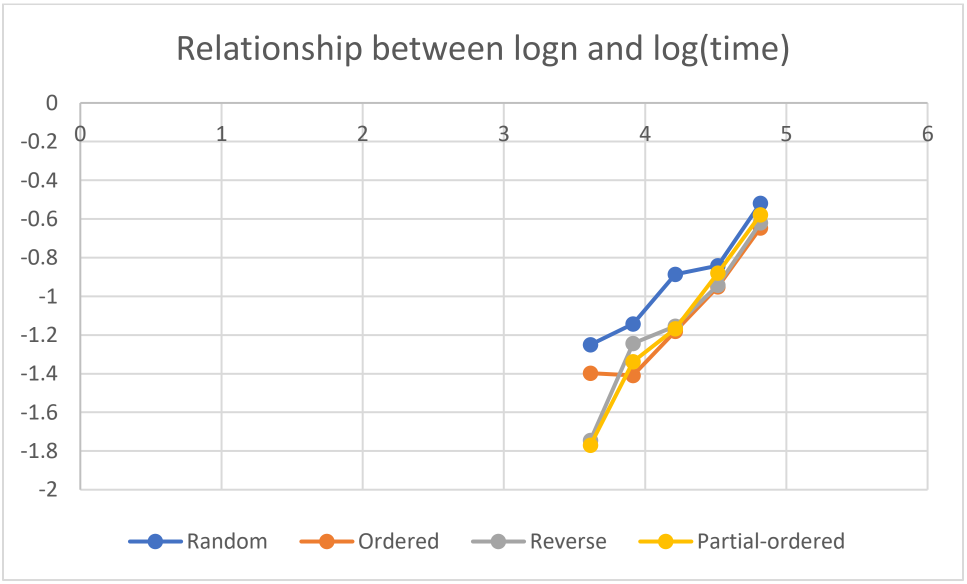
2. <u>Conclusion</u>

- Based on the theory and experiments ran which I will show below, the relationship between array size (N) and time running(ms) is **exponential growth (On^2).** Or as we said, as the array size of input increase double, the running time will be increase by 4 times.
- About particular running time, as the array size(N) getting bigger, the general conclusion about running time of 4 different array sequences:
  **+ ordered < partial ordered ~ random < reverse ordered**. This also strengthen what we learnt as reverse ordered array would be the worst case for insertion sort. Thus, it will have larger average running time.

3. <u>Evidence</u>

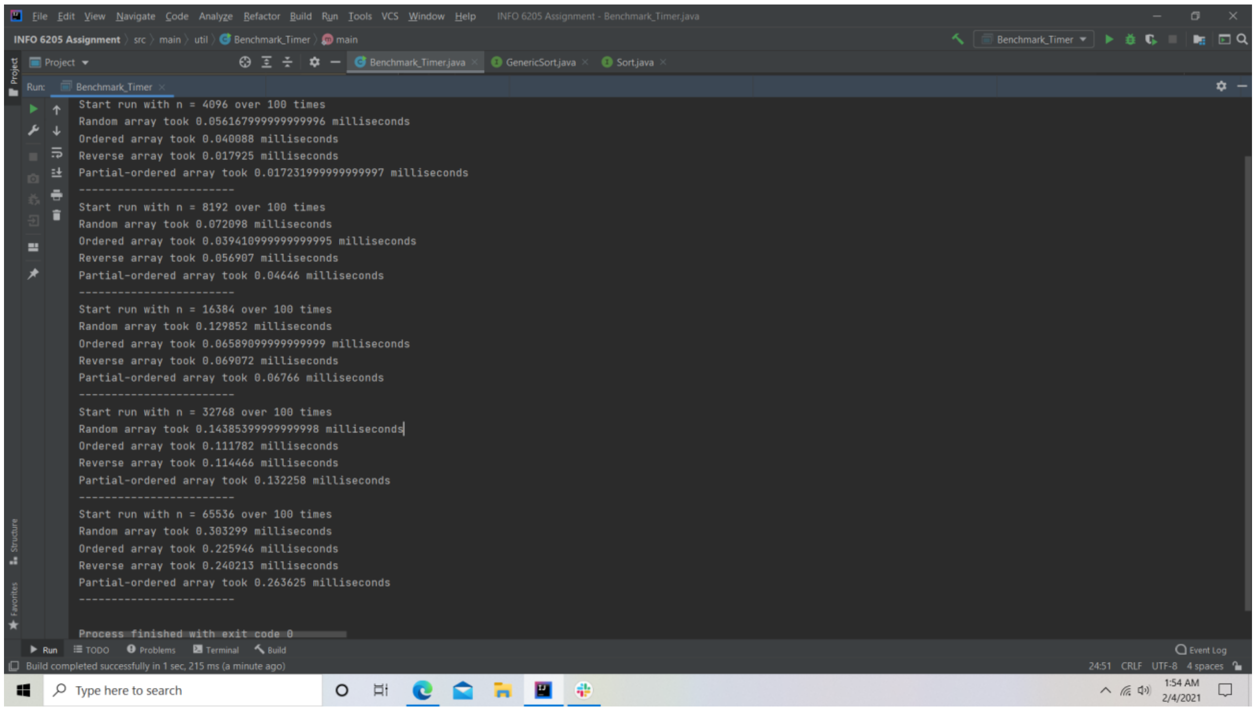
| Log(N) | Random | Ordered | Reverse | Partial-ordered |
|---|---|---|---|---|
| 3.61235995 | -1.2503 | -1.3979 | -1.7447 | -1.769551079 |
| 3.91338994 | -1.1421 | -1.4089 | -1.2441 | -1.337242168 |
| 4.21441994 | -0.8861 | -1.1805 | -1.1549 | -1.167491087 |
| 4.51544993 | -0.8416 | -0.9508 | -0.9431 | -0.879426069 |
| 4.81647993 | -0.5186 | -0.6459 | -0.6198 | -0.578396073 |





Although the 1st chart didn't show a clear relationship for n and time, the 2nd chart taking logn and log(time) def prove that this is close to an exponential growth (On^2) rather than a linear growth (On).

4. <u>Code</u>
   File modified including Timer.java, Benchmark_Timer.java & InsertionSort.java. I also changed the required running time in TimerTest due to hardware running time difference. I have uploaded 2 version on github. One with just assignment 2 related files. If you have trouble running it. Please download the master folder on repo for running/testing.
5. <u>Unit Test</u>
   (Benchmark Main output)

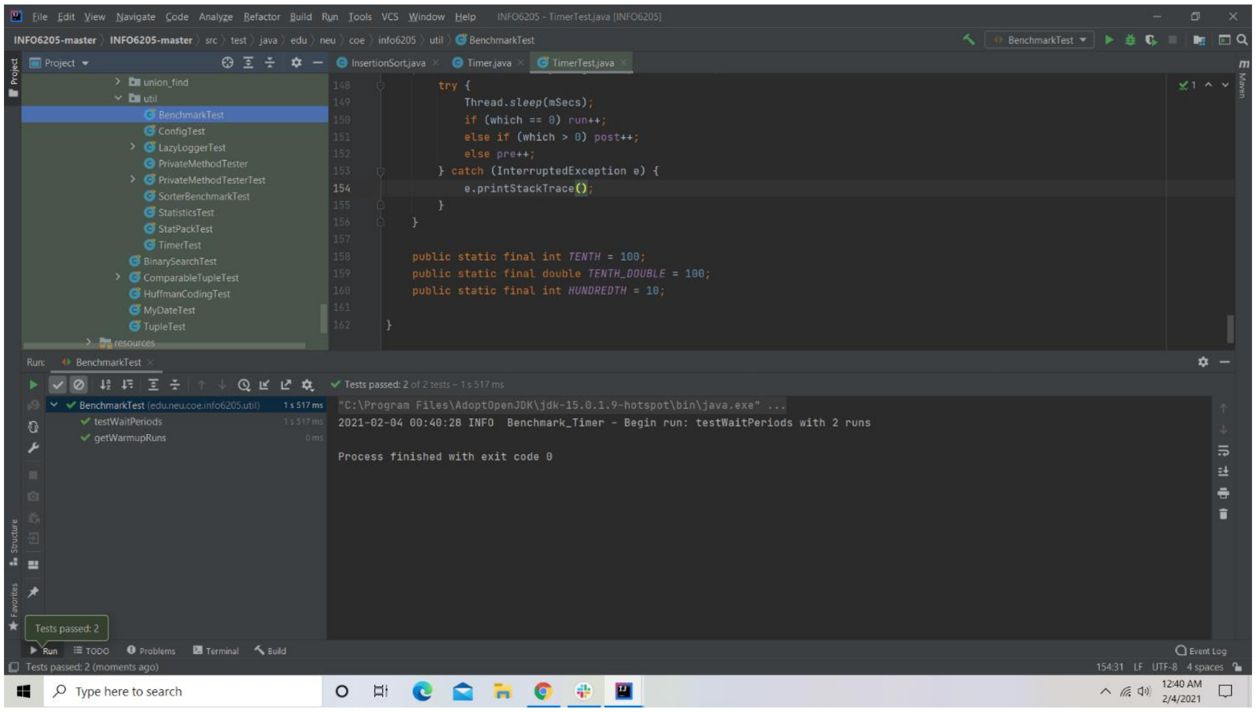


```
Start run with n = 4096 over 100 times
Random array took 0.056167999999999996 milliseconds
Ordered array took 0.040088 milliseconds
Reverse array took 0.017925 milliseconds
Partial-ordered array took 0.017231999999999997 milliseconds
------------------------
Start run with n = 8192 over 100 times
Random array took 0.072098 milliseconds
Ordered array took 0.039410999999999995 milliseconds
Reverse array took 0.056907 milliseconds
Partial-ordered array took 0.04646 milliseconds
------------------------
Start run with n = 16384 over 100 times
Random array took 0.129852 milliseconds
Ordered array took 0.065809999999999999 milliseconds
Reverse array took 0.069072 milliseconds
Partial-ordered array took 0.06766 milliseconds
------------------------
Start run with n = 32768 over 100 times
Random array took 0.14385399999999998 milliseconds
Ordered array took 0.111782 milliseconds
Reverse array took 0.114466 milliseconds
Partial-ordered array took 0.132258 milliseconds
------------------------
Start run with n = 65536 over 100 times
Random array took 0.303299 milliseconds
Ordered array took 0.225946 milliseconds
Reverse array took 0.240213 milliseconds
Partial-ordered array took 0.263625 milliseconds
------------------------

Process finished with exit code 0
```

Build completed successfully in 1 sec, 215 ms (a minute ago)

(Benchmark Test)



```
        try {
            Thread.sleep(mSecs);
            if (which == 0) run++;
            else if (which > 0) post++;
            else pre++;
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    public static final int TENTH = 100;
    public static final double TENTH_DOUBLE = 100;
    public static final int HUNDREDTH = 10;

}
```

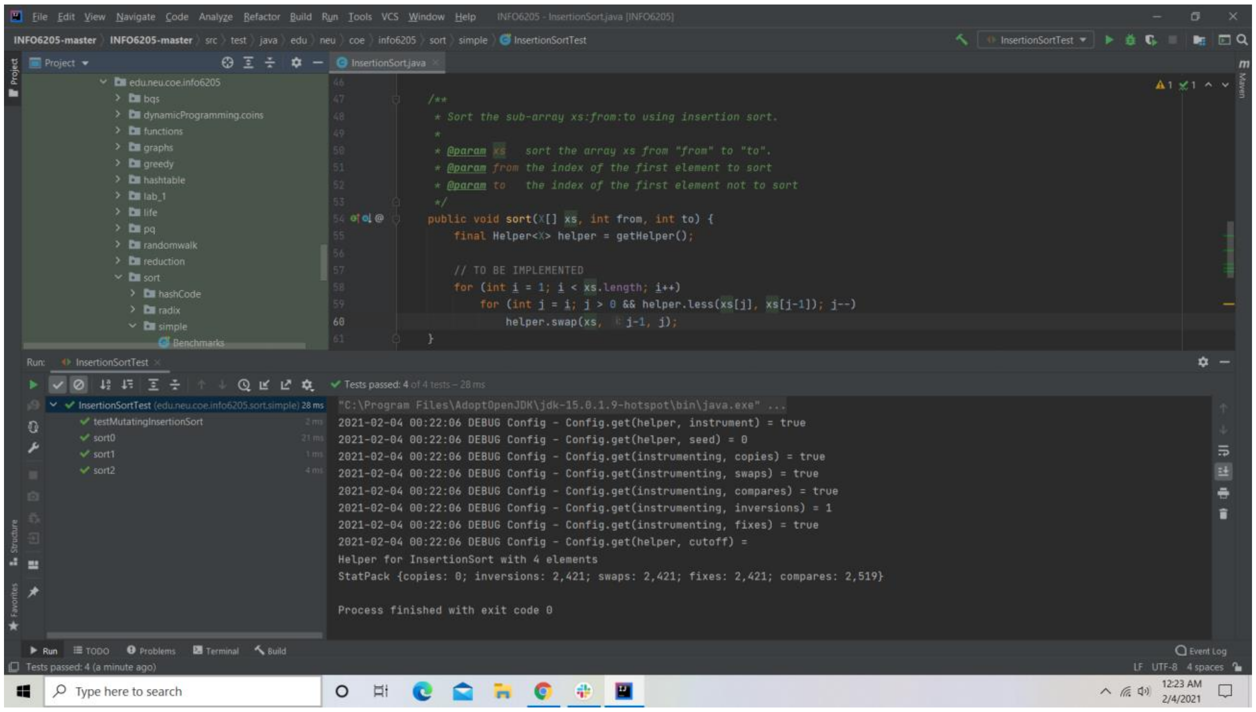
Tests passed: 2 of 2 tests – 1 s 517 ms

```
"C:\Program Files\AdoptOpenJDK\jdk-15.0.1.9-hotspot\bin\java.exe" ...
2021-02-04 00:40:28 INFO  Benchmark_Timer - Begin run: testWaitPeriods with 2 runs

Process finished with exit code 0
```

Tests passed: 2 (moments ago)

(InsertionSort Test)



```java
    /**
     * Sort the sub-array xs:from:to using insertion sort.
     *
     * @param xs   sort the array xs from "from" to "to".
     * @param from the index of the first element to sort
     * @param to   the index of the first element not to sort
     */
    public void sort(X[] xs, int from, int to) {
        final Helper<X> helper = getHelper();

        // TO BE IMPLEMENTED
        for (int i = 1; i < xs.length; i++)
            for (int j = i; j > 0 && helper.less(xs[j], xs[j-1]); j--)
                helper.swap(xs, j-1, j);
    }
```

```
InsertionSortTest (edu.neu.coe.info6205.sort.simple) 28 ms
  ✓ testMutatingInsertionSort   2 ms
  ✓ sort0                      21 ms
  ✓ sort1                       1 ms
  ✓ sort2                       4 ms
```

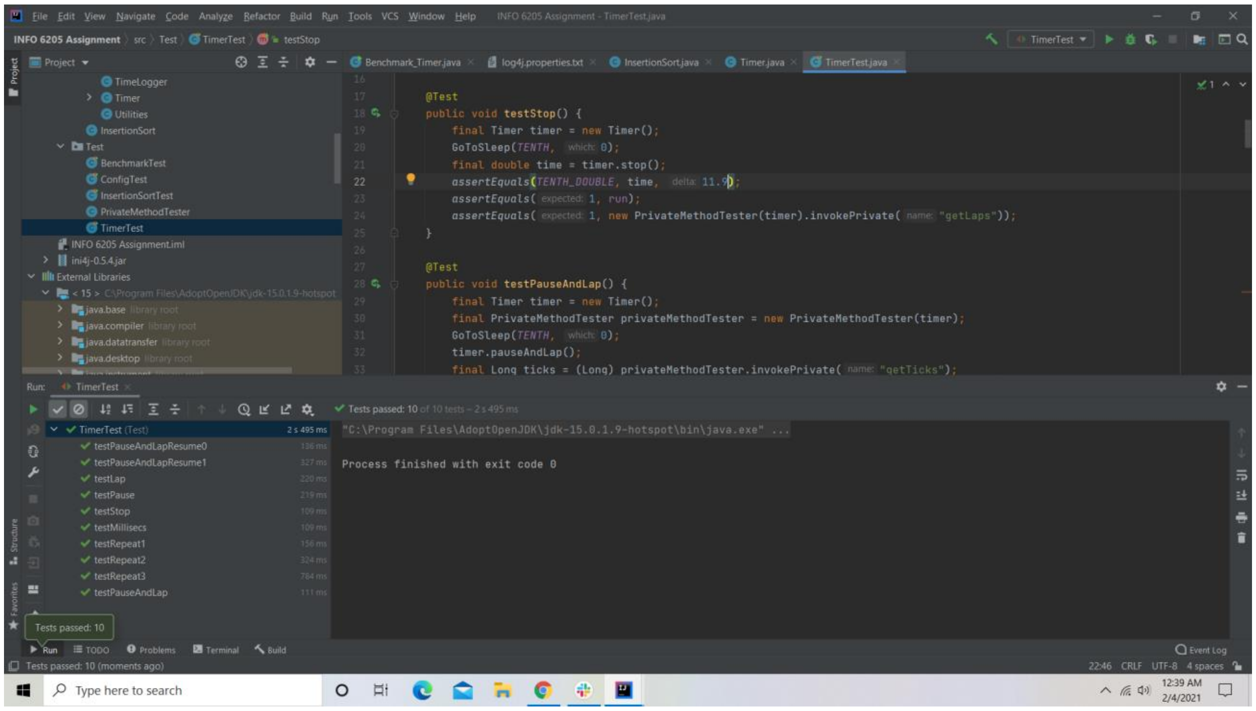
Tests passed: 4 of 4 tests – 28 ms

```
"C:\Program Files\AdoptOpenJDK\jdk-15.0.1.9-hotspot\bin\java.exe" ...
2021-02-04 00:22:06 DEBUG Config - Config.get(helper, instrument) = true
2021-02-04 00:22:06 DEBUG Config - Config.get(helper, seed) = 0
2021-02-04 00:22:06 DEBUG Config - Config.get(instrumenting, copies) = true
2021-02-04 00:22:06 DEBUG Config - Config.get(instrumenting, swaps) = true
2021-02-04 00:22:06 DEBUG Config - Config.get(instrumenting, compares) = true
2021-02-04 00:22:06 DEBUG Config - Config.get(instrumenting, inversions) = 1
2021-02-04 00:22:06 DEBUG Config - Config.get(instrumenting, fixes) = true
2021-02-04 00:22:06 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
```

Tests passed: 4 (a minute ago)

(Timer Test)



```java
    @Test
    public void testStop() {
        final Timer timer = new Timer();
        GoToSleep(TENTH, which: 0);
        final double time = timer.stop();
        assertEquals(TENTH_DOUBLE, time, delta: 11.9);
        assertEquals( expected: 1, run);
        assertEquals( expected: 1, new PrivateMethodTester(timer).invokePrivate( name: "getLaps"));
    }

    @Test
    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( name: "getTicks");
```

```
TimerTest (Test)                 2 s 495 ms
  ✓ testPauseAndLapResume0     136 ms
  ✓ testPauseAndLapResume1     327 ms
  ✓ testLap                    220 ms
  ✓ testPause                  219 ms
  ✓ testStop                   109 ms
  ✓ testMillisecs              109 ms
  ✓ testRepeat1                156 ms
  ✓ testRepeat2                324 ms
  ✓ testRepeat3                784 ms
  ✓ testPauseAndLap            111 ms
```

Tests passed: 10 of 10 tests – 2 s 495 ms

```
"C:\Program Files\AdoptOpenJDK\jdk-15.0.1.9-hotspot\bin\java.exe" ...

Process finished with exit code 0
```

Tests passed: 10 (moments ago)