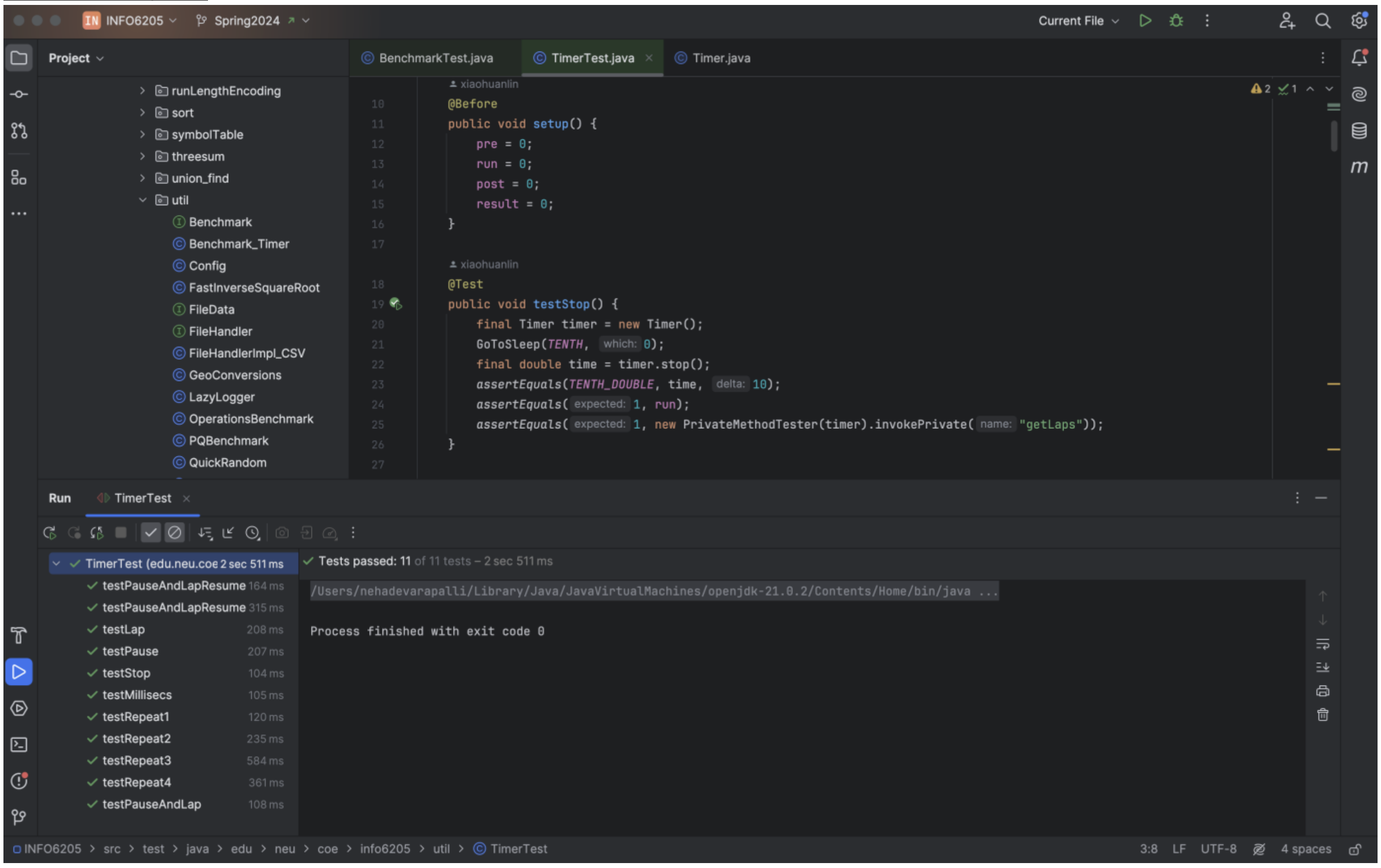
INFO 6205 - Program Structures and Algorithms
Spring 2024

NAME: Neha Devarapalli
NUID: 002883137
GITHUB LINK: https://github.com/nehadevarapalli/INFO6205

Task: Assignment 3 (Benchmark)
Part 1: To implement repeat(), getClock() and toMilliseconds() in Timer.java and perform unit tests
Part 2: To implement InsertionSort and perform its corresponding unit tests.
Part 3: To perform benchmarks for randomly ordered, partially ordered, ordered and reverse ordered input arrays for insertion sort and observe the statistics.

Unit Test Screenshots & Observations:

Part 1:
TimerTest.java



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

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

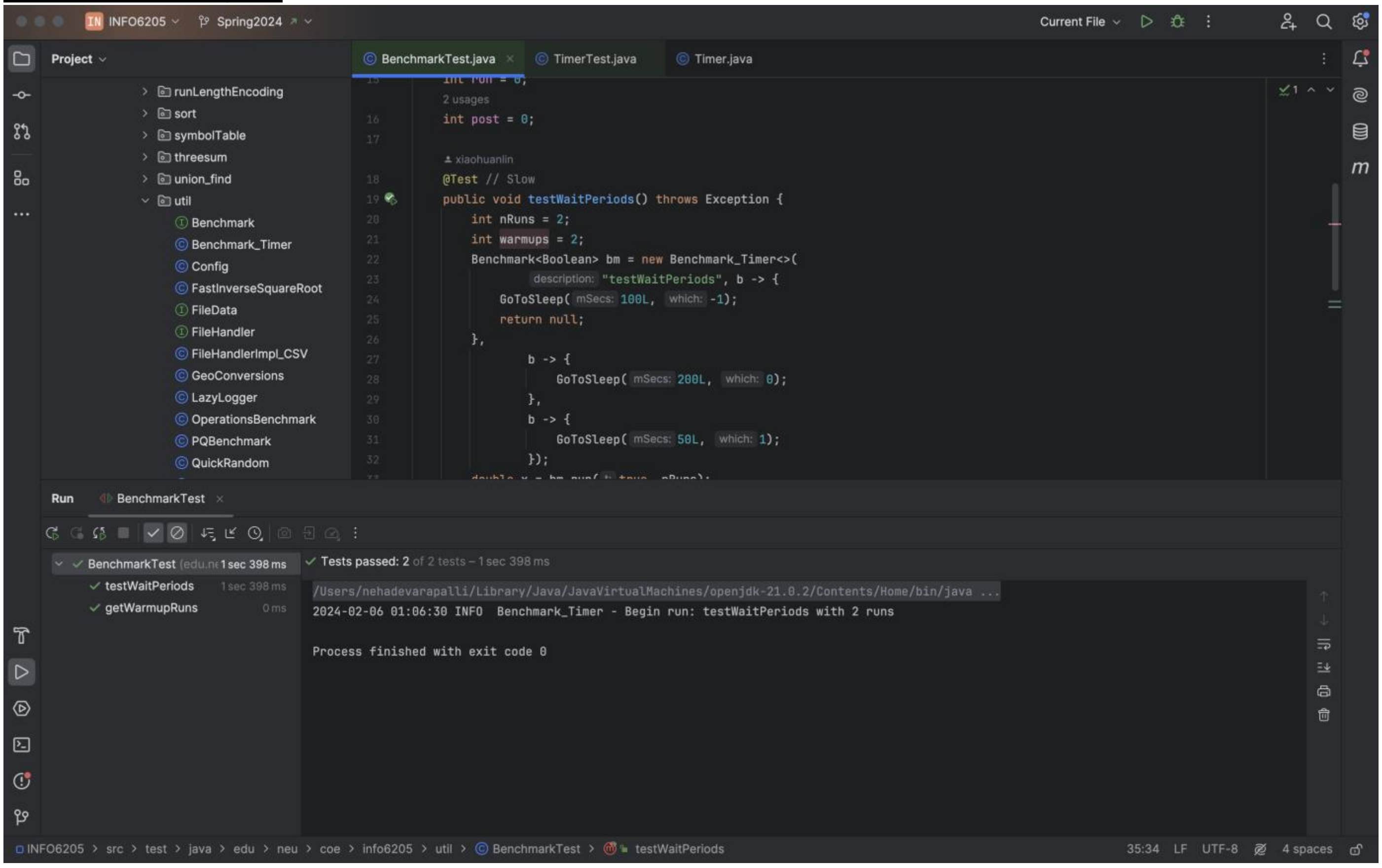
```
✓ TimerTest (edu.neu.coe 2 sec 511 ms
  ✓ testPauseAndLapResume 164 ms
  ✓ testPauseAndLapResume 315 ms
  ✓ testLap               208 ms
  ✓ testPause             207 ms
  ✓ testStop              104 ms
  ✓ testMillisecs         105 ms
  ✓ testRepeat1           120 ms
  ✓ testRepeat2           235 ms
  ✓ testRepeat3           584 ms
  ✓ testRepeat4           361 ms
  ✓ testPauseAndLap       108 ms
```

```
✓ Tests passed: 11 of 11 tests – 2 sec 511 ms

/Users/nehadevarapalli/Library/Java/JavaVirtualMachines/openjdk-21.0.2/Contents/Home/bin/java ...

Process finished with exit code 0
```

BenchmarkTest.java



```
    int post = 0;

    @Test // Slow
    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);
                });
```

```
✓ BenchmarkTest (edu.ne 1 sec 398 ms
  ✓ testWaitPeriods     1 sec 398 ms
  ✓ getWarmupRuns       0 ms

✓ Tests passed: 2 of 2 tests – 1 sec 398 ms

/Users/nehadevarapalli/Library/Java/JavaVirtualMachines/openjdk-21.0.2/Contents/Home/bin/java ...
2024-02-06 01:06:30 INFO  Benchmark_Timer - Begin run: testWaitPeriods with 2 runs

Process finished with exit code 0
```

Part 2:

Insertion sort takes up O(n^2) time complexity.

InsertionSortTest.java

Sort Method:

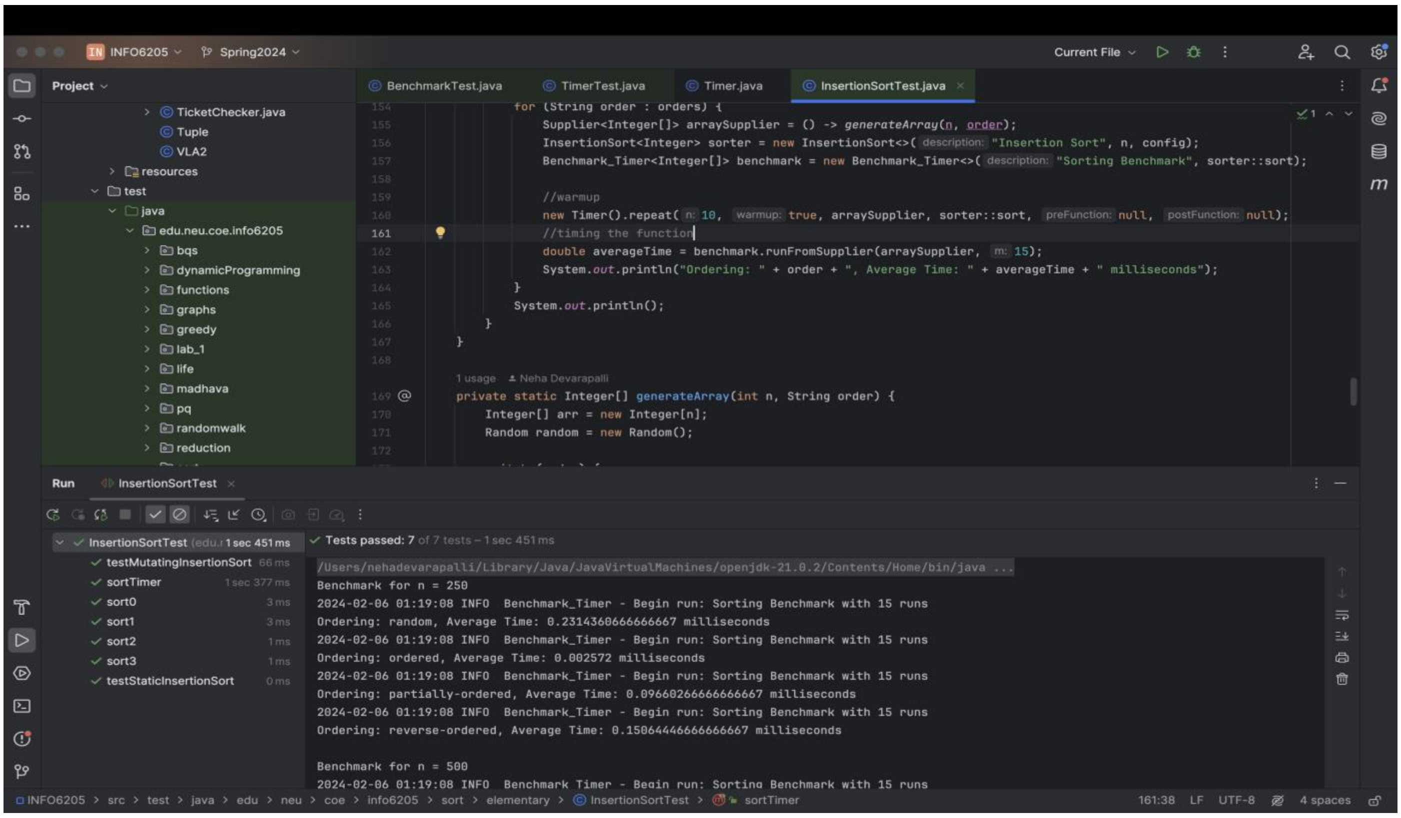
- Performs in-place insertion sort on a sub-array **xs** from index ‘**from**’ to ‘**to**’.

Insertion Sort Implementation:

- The implementation includes two variants based on whether the helper is instrumented for gathering statistics.
- If instrumented, used a stable conditional swap implemented in **Helper.java** to insert elements in the sorted order.
- If not instrumented, used a standard nested loop approach to compare and swap elements for sorting.

Time Complexity:

- Worst Case: O(n^2) when the array is in reverse order.
- Best Case: O(n) when the array is already sorted.
- Average Case: O(n^2).
- The inner loop performs comparisons and swaps, changing the overall time complexity in each case.



```java
            for (String order : orders) {
                Supplier<Integer[]> arraySupplier = () -> generateArray(n, order);
                InsertionSort<Integer> sorter = new InsertionSort<>("Insertion Sort", n, config);
                Benchmark_Timer<Integer[]> benchmark = new Benchmark_Timer<>("Sorting Benchmark", sorter::sort);

                //warmup
                new Timer().repeat(10, true, arraySupplier, sorter::sort, null, null);
                //timing the function
                double averageTime = benchmark.runFromSupplier(arraySupplier, 15);
                System.out.println("Ordering: " + order + ", Average Time: " + averageTime + " milliseconds");
            }
            System.out.println();
        }
    }

    private static Integer[] generateArray(int n, String order) {
        Integer[] arr = new Integer[n];
        Random random = new Random();
```

```
Tests passed: 7 of 7 tests – 1 sec 451 ms
/Users/nehadevarapalli/Library/Java/JavaVirtualMachines/openjdk-21.0.2/Contents/Home/bin/java ...
Benchmark for n = 250
2024-02-06 01:19:08 INFO  Benchmark_Timer - Begin run: Sorting Benchmark with 15 runs
Ordering: random, Average Time: 0.2314360666666667 milliseconds
2024-02-06 01:19:08 INFO  Benchmark_Timer - Begin run: Sorting Benchmark with 15 runs
Ordering: ordered, Average Time: 0.002572 milliseconds
2024-02-06 01:19:08 INFO  Benchmark_Timer - Begin run: Sorting Benchmark with 15 runs
Ordering: partially-ordered, Average Time: 0.09660266666666667 milliseconds
2024-02-06 01:19:08 INFO  Benchmark_Timer - Begin run: Sorting Benchmark with 15 runs
Ordering: reverse-ordered, Average Time: 0.15064446666666667 milliseconds

Benchmark for n = 500
2024-02-06 01:19:08 INFO  Benchmark_Timer - Begin run: Sorting Benchmark with 15 runs
```

```
InsertionSortTest         1 sec 451 ms
  testMutatingInsertionSort      66 ms
  sortTimer              1 sec 377 ms
  sort0                          3 ms
  sort1                          3 ms
  sort2                          1 ms
  sort3                          1 ms
  testStaticInsertionSort        0 ms
```

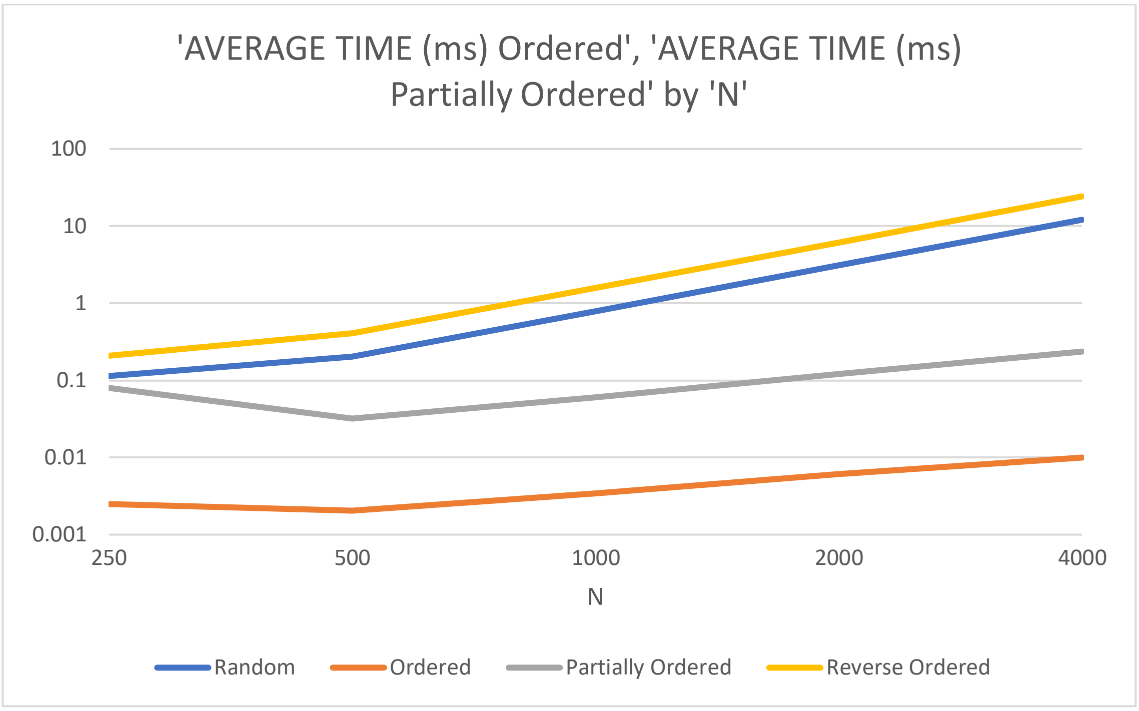
Part 3:

For this task, I have implemented the code to run the benchmarks in **InsertionSortTest.java** itself. The test is called **sortTimer()**.

These are the average time in milliseconds for each different type of ordered input which I have tabulated in the following manner:

N	AVERAGE TIME (ms)			
	Random	Ordered	Partially Ordered	Reverse Ordered
250	0.1138972	0.002477867	0.0802776	0.208749867
500	0.201536067	0.002047133	0.031983333	0.407749933
1000	0.788747333	0.0034472	0.060155667	1.578086133
2000	3.099536067	0.006069533	0.121355733	6.166386133
4000	12.06537233	0.0099834	0.236186267	24.2192306

Also, plotted a graph to better visualize the data.



As expected, Reverse Ordered input takes the most amount of time, followed by Random Ordered, Partially Ordered and at last also the least time taking is the Ordered input array.