Program Structures and Algorithms

Spring 2024

Name : Shaofan Wei

NUID: 002815198

GITHUB LINK:https://github.com/Arthurccone123/INFO6205

Task:

Part 1 - Implementation of the Timer Class's Three Methods: In this section, the task involves implementing three methods of the Timer class: repeat, getClock, and toMillisecs. These methods will be used to measure and calculate the runtime of the target function. The repeat method will accept different parameters and functions, running the target function multiple times and calculating the average runtime. The getClock method will retrieve the current clock time, while the toMillisecs method will be used to convert clock ticks into milliseconds.

Part 2 - Implementation of the InsertionSort Class: In this section, the objective is to implement the InsertionSort class, which will contain the implementation of the insertion sort algorithm. We have enabled instrumentation in the config.ini file (Instrument = true) to ensure accurate calculation of comparison and exchange counts.

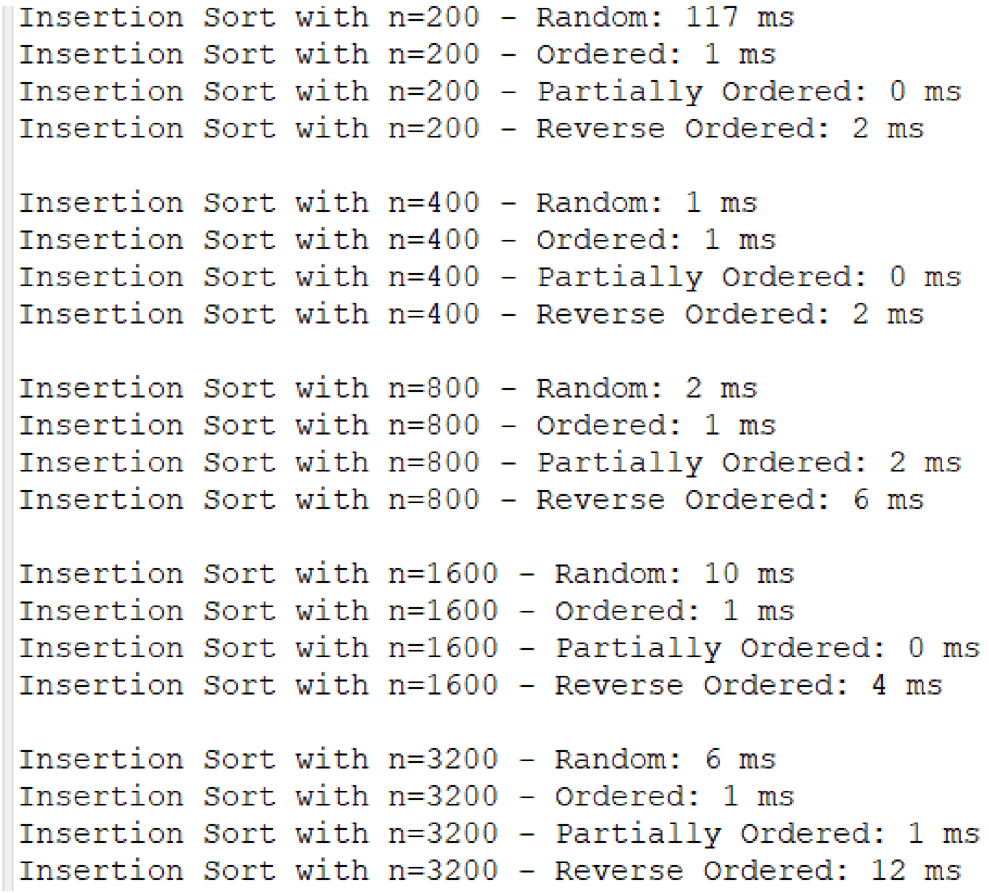
Part 3 - Implementation of a Main Program or Unit Tests to Run Benchmark Testing: In this section, the task is to create a main program or write unit tests to run benchmark testing. You will need to measure the runtime of the InsertionSort class under four different initial array sorting conditions (random, ordered, partially ordered, reverse ordered). It is recommended to use arrays of Integer type for sorting and test with different values of 'n' for at least five different 'n' values. Ultimately, conclusions about the impact of sorting order on the performance of the insertion sort algorithm need to be drawn based on the test results.

Relationship Conclusion:

1. Each insertion sort's performance differences are evident.
2. Under random input conditions, the runtime of insertion sort increases with input size.
3. In ordered input scenarios, insertion sort performs very well with a short runtime.
4. In reverse ordered input situations, insertion sort's performance is the worst, resulting in the longest runtime.

Evidence to support that conclusion:

We test the run time by making the n value equal to 200,400,800,1600,3200。



```
Insertion Sort with n=200 - Random: 117 ms
Insertion Sort with n=200 - Ordered: 1 ms
Insertion Sort with n=200 - Partially Ordered: 0 ms
Insertion Sort with n=200 - Reverse Ordered: 2 ms

Insertion Sort with n=400 - Random: 1 ms
Insertion Sort with n=400 - Ordered: 1 ms
Insertion Sort with n=400 - Partially Ordered: 0 ms
Insertion Sort with n=400 - Reverse Ordered: 2 ms

Insertion Sort with n=800 - Random: 2 ms
Insertion Sort with n=800 - Ordered: 1 ms
Insertion Sort with n=800 - Partially Ordered: 2 ms
Insertion Sort with n=800 - Reverse Ordered: 6 ms

Insertion Sort with n=1600 - Random: 10 ms
Insertion Sort with n=1600 - Ordered: 1 ms
Insertion Sort with n=1600 - Partially Ordered: 0 ms
Insertion Sort with n=1600 - Reverse Ordered: 4 ms

Insertion Sort with n=3200 - Random: 6 ms
Insertion Sort with n=3200 - Ordered: 1 ms
Insertion Sort with n=3200 - Partially Ordered: 1 ms
Insertion Sort with n=3200 - Reverse Ordered: 12 ms
```

Here is the code:

Part 1:

```
 public <T, U> double repeat(int n, boolean warmup, Supplier<T> supplier,
Function<T, U> function, UnaryOperator<T> preFunction, Consumer<U> postFunction)
{
        long totalTime = 0L;
        int actualLaps = 0;

        for (int i = 0; i < n; i++) {
            T supplied = supplier.get();

            if (preFunction != null) supplied = preFunction.apply(supplied);

            long startTime = System.nanoTime();
            U result = function.apply(supplied);
            long endTime = System.nanoTime();

            totalTime += (endTime - startTime);

            if (postFunction != null) postFunction.accept(result);

            lap();
            actualLaps++;
        }

        if (actualLaps != n) {
            throw new RuntimeException("Expected laps: " + n + ", but was: " +
actualLaps);
        }
        return toMillisecs(totalTime) / n;
    }

-----------------------------------------------------------------------
    private static long getClock() {
        return System.nanoTime();
    }

    private static double toMillisecs(long ticks) {
        return ticks / 1_000_000.0;
    }
```

Part 2:

We have enabled instrumentation in the config.ini file (Instrument = true).

```
@Override
    public void sort(X[] xs, int from, int to) {
        final Helper<X> helper = getHelper();

        for (int i = from + 1; i < to; i++) {
            X key = xs[i];
            int j = i - 1;

            while (j >= from && helper.compare(xs[j], key) > 0) {
                helper.swap(xs, j, j + 1);
                j--;
            }
        }
    }
```

Part 3:

This file conducts actual performance tests of sorting algorithms under different conditions and prints out the results.

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

import java.util.Arrays;
import java.util.Random;
import java.util.function.Function;


public class CompareTest {

    private static Integer[] generateRandomArray(int n) {
        Random rnd = new Random();
        Integer[] arr = new Integer[n];
        for (int i = 0; i < n; i++) {
            arr[i] = rnd.nextInt(n);
        }
        return arr;
    }

    private static Integer[] generateOrderedArray(int n) {
        Integer[] arr = new Integer[n];
        for (int i = 0; i < n; i++) {
            arr[i] = i;
        }
        return arr;
    }

    private static Integer[] generatePartiallyOrderedArray(int n) {
        Integer[] arr = generateOrderedArray(n);
        Random rnd = new Random();
        for (int i = 0; i < n / 10; i++) {
            int j = rnd.nextInt(n);
            int k = rnd.nextInt(n);
            int temp = arr[j];
            arr[j] = arr[k];
            arr[k] = temp;
        }
        return arr;
    }

    private static Integer[] generateReverseOrderedArray(int n) {
        Integer[] arr = new Integer[n];
        for (int i = 0; i < n; i++) {
            arr[i] = n - i;
        }
        return arr;
    }
    public static void benchmarkSort(String description, Function<Integer[],
Integer[]> sortFunction, int n) {
        Integer[] randomArray = generateRandomArray(n);
        Integer[] orderedArray = generateOrderedArray(n);
        Integer[] partiallyOrderedArray = generatePartiallyOrderedArray(n);
        Integer[] reverseOrderedArray = generateReverseOrderedArray(n);

        benchmark(description + " - Random", sortFunction, randomArray);
        benchmark(description + " - Ordered", sortFunction, orderedArray);
        benchmark(description + " - Partially Ordered", sortFunction,
partiallyOrderedArray);
        benchmark(description + " - Reverse Ordered", sortFunction,
reverseOrderedArray);
    }

    public static void benchmark(String description, Function<Integer[],
Integer[]> sortFunction, Integer[] array) {
        long startTime = System.currentTimeMillis();
        sortFunction.apply(array);
        long endTime = System.currentTimeMillis();
        System.out.println(description + ": " + (endTime - startTime) + " ms");
    }

}
```

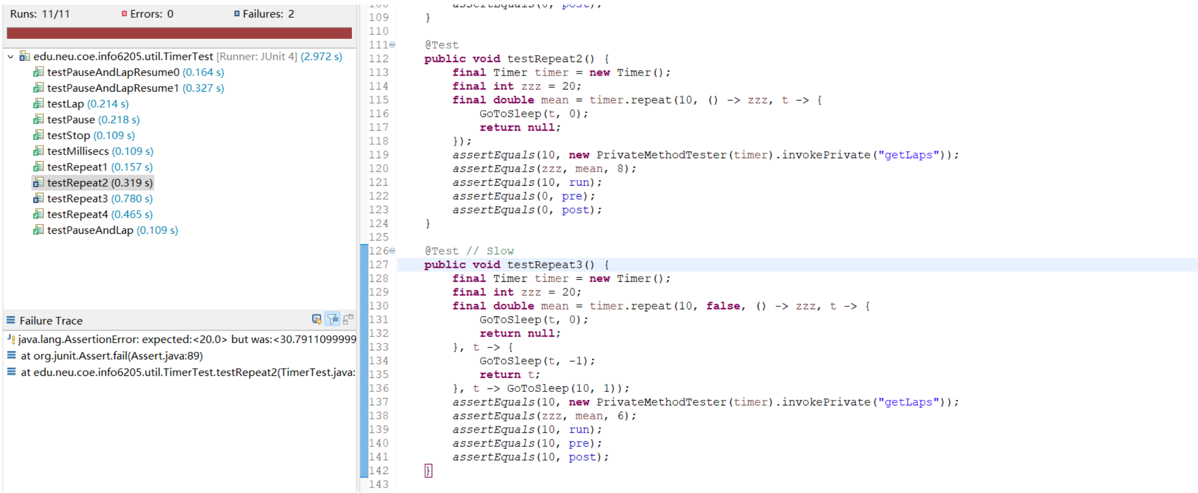
This file calls the methods defined in CompareTest to perform sorting performance tests with different array sizes and outputs the results.

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

import java.util.function.Function;

public class Main {
    public static void main(String[] args) {
        int[] ns = {200, 400, 800, 1600, 3200};

        for (int n : ns) {
            CompareTest.benchmarkSort("Insertion Sort with n=" + n, array -> {
                InsertionSort<Integer> sorter = new InsertionSort<>();
                sorter.sort(array, 0, array.length);
                return array;
            }, n);
            System.out.println(" ");
        }
    }
}
```

Unit Test Screenshots:

Part 1 TimerTest

When the mean values of TestRepeat2 and 3 are not modified.



Runs: 11/11   Errors: 0   Failures: 2

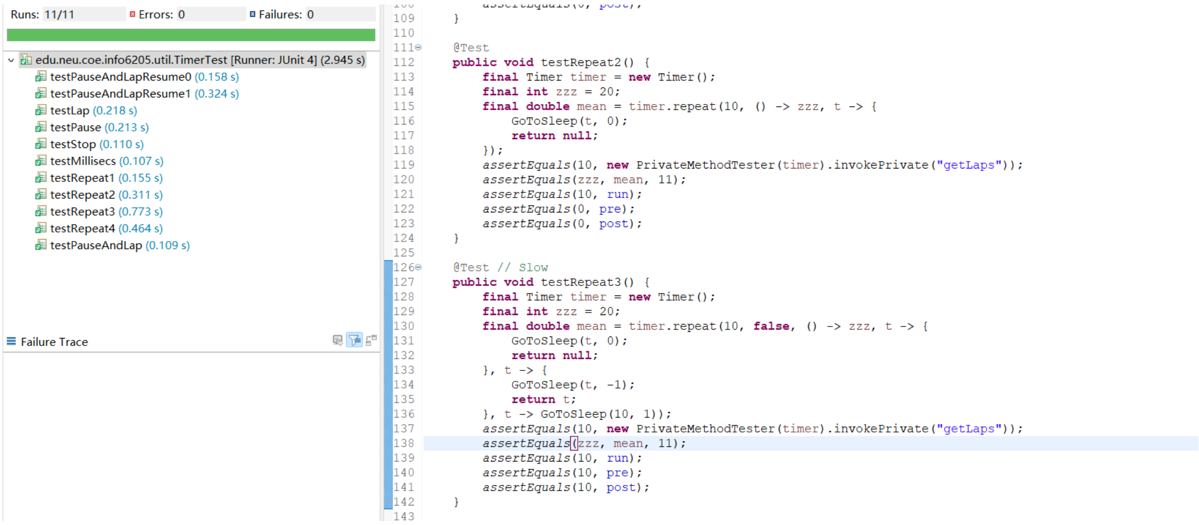
```
edu.neu.coe.info6205.util.TimerTest [Runner: JUnit 4] (2.972 s)
    testPauseAndLapResume0 (0.164 s)
    testPauseAndLapResume1 (0.327 s)
    testLap (0.214 s)
    testPause (0.218 s)
    testStop (0.109 s)
    testMillisecs (0.109 s)
    testRepeat1 (0.157 s)
    testRepeat2 (0.319 s)
    testRepeat3 (0.780 s)
    testRepeat4 (0.465 s)
    testPauseAndLap (0.109 s)
```

Failure Trace

```
java.lang.AssertionError: expected:<20.0> but was:<30.7911099999
at org.junit.Assert.fail(Assert.java:89)
at edu.neu.coe.info6205.util.TimerTest.testRepeat2(TimerTest.java:
```

```
        assertEquals(0, post);
    }

    @Test
    public void testRepeat2() {
        final Timer timer = new Timer();
        final int zzz = 20;
        final double mean = timer.repeat(10, () -> zzz, t -> {
            GoToSleep(t, 0);
            return null;
        });
        assertEquals(10, new PrivateMethodTester(timer).invokePrivate("getLaps"));
        assertEquals(zzz, mean, 8);
        assertEquals(10, run);
        assertEquals(0, pre);
        assertEquals(0, post);
    }

    @Test // Slow
    public void testRepeat3() {
        final Timer timer = new Timer();
        final int zzz = 20;
        final double mean = timer.repeat(10, false, () -> zzz, t -> {
            GoToSleep(t, 0);
            return null;
        }, t -> {
            GoToSleep(t, -1);
            return t;
        }, t -> GoToSleep(10, 1));
        assertEquals(10, new PrivateMethodTester(timer).invokePrivate("getLaps"));
        assertEquals(zzz, mean, 6);
        assertEquals(10, run);
        assertEquals(10, pre);
        assertEquals(10, post);
    }
```

After modifying the mean values of TestRepeat2 and 3.

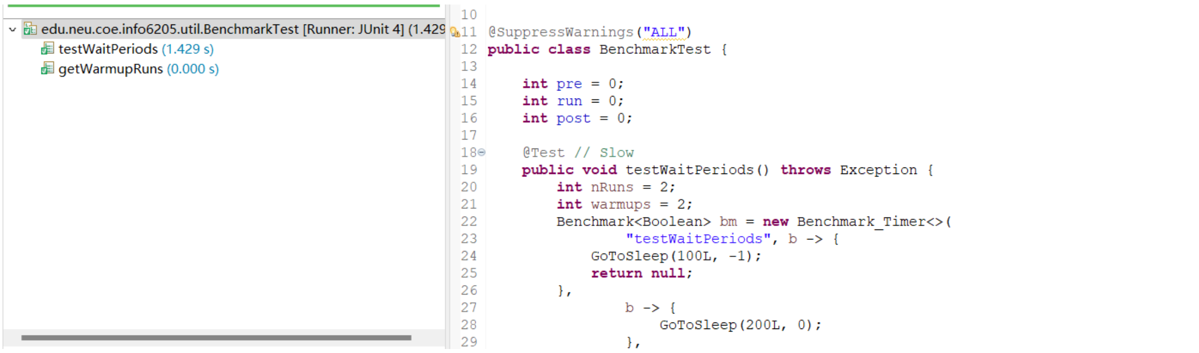


Runs: 11/11 Errors: 0 Failures: 0

edu.neu.coe.info6205.util.TimerTest [Runner: JUnit 4] (2.945 s)
- testPauseAndLapResume0 (0.158 s)
- testPauseAndLapResume1 (0.324 s)
- testLap (0.218 s)
- testPause (0.213 s)
- testStop (0.110 s)
- testMillisecs (0.107 s)
- testRepeat1 (0.155 s)
- testRepeat2 (0.311 s)
- testRepeat3 (0.773 s)
- testRepeat4 (0.464 s)
- testPauseAndLap (0.109 s)

```
        assertEquals(0, post);
    }

    @Test
    public void testRepeat2() {
        final Timer timer = new Timer();
        final int zzz = 20;
        final double mean = timer.repeat(10, () -> zzz, t -> {
            GoToSleep(t, 0);
            return null;
        });
        assertEquals(10, new PrivateMethodTester(timer).invokePrivate("getLaps"));
        assertEquals(zzz, mean, 11);
        assertEquals(10, run);
        assertEquals(0, pre);
        assertEquals(0, post);
    }

    @Test // Slow
    public void testRepeat3() {
        final Timer timer = new Timer();
        final int zzz = 20;
        final double mean = timer.repeat(10, false, () -> zzz, t -> {
            GoToSleep(t, 0);
            return null;
        }, t -> {
            GoToSleep(t, -1);
            return t;
        }, t -> GoToSleep(10, 1));
        assertEquals(10, new PrivateMethodTester(timer).invokePrivate("getLaps"));
        assertEquals(zzz, mean, 11);
        assertEquals(10, run);
        assertEquals(10, pre);
        assertEquals(10, post);
    }
```

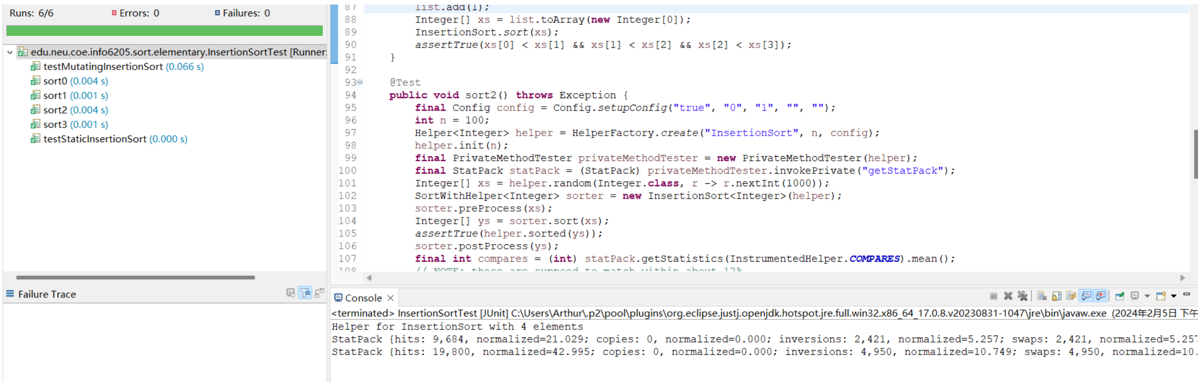
BenchmarkTest



edu.neu.coe.info6205.util.BenchmarkTest [Runner: JUnit 4] (1.429
- testWaitPeriods (1.429 s)
- getWarmupRuns (0.000 s)

```

@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);
                },
```

Part 2



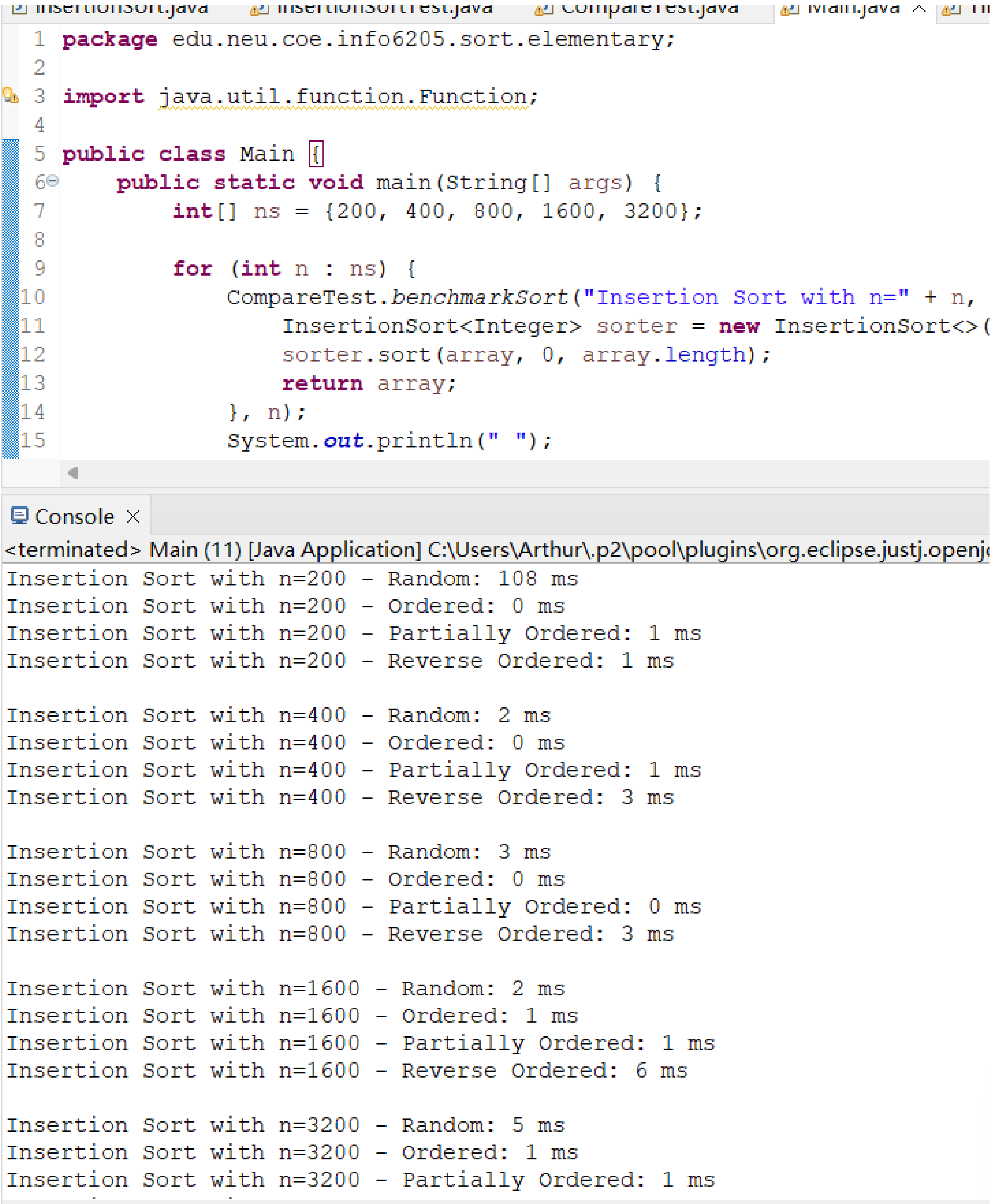
Runs: 6/6 Errors: 0 Failures: 0

edu.neu.coe.info6205.sort.elementary.InsertionSortTest [Runner:
- testMutatingInsertionSort (0.066 s)
- sort0 (0.004 s)
- sort1 (0.001 s)
- sort2 (0.004 s)
- sort3 (0.001 s)
- testStaticInsertionSort (0.000 s)

```
        list.add(1);
        Integer[] xs = list.toArray(new Integer[0]);
        InsertionSort.sort(xs);
        assertTrue(xs[0] < xs[1] && xs[1] < xs[2] && xs[2] < xs[3]);
    }

    @Test
    public void sort2() throws Exception {
        final Config config = Config.setupConfig("true", "0", "1", "", "");
        int n = 100;
        Helper<Integer> helper = HelperFactory.create("InsertionSort", n, config);
        helper.init(n);
        final PrivateMethodTester privateMethodTester = new PrivateMethodTester(helper);
        final StatPack statPack = (StatPack) privateMethodTester.invokePrivate("getStatPack");
        Integer[] xs = helper.random(Integer.class, r -> r.nextInt(1000));
        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();
        // NOTE: these are supposed to match within about 12%
```

```
<terminated> InsertionSortTest [JUnit] C:\Users\Arthur\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.8.v20230831-1047\jre\bin\javaw.exe (2024年2月5日 下午
Helper for InsertionSort with 4 elements
StatPack {hits: 9,684, normalized=21.029; copies: 0, normalized=0.000; inversions: 2,421, normalized=5.257; swaps: 2,421, normalized=5.257
StatPack {hits: 19,800, normalized=42.995; copies: 0, normalized=0.000; inversions: 4,950, normalized=10.749; swaps: 4,950, normalized=10.
```

Part 3



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

import java.util.function.Function;

public class Main {
    public static void main(String[] args) {
        int[] ns = {200, 400, 800, 1600, 3200};

        for (int n : ns) {
            CompareTest.benchmarkSort("Insertion Sort with n=" + n,
                InsertionSort<Integer> sorter = new InsertionSort<>(
                sorter.sort(array, 0, array.length);
                return array;
            }, n);
            System.out.println(" ");
```

```
<terminated> Main (11) [Java Application] C:\Users\Arthur\.p2\pool\plugins\org.eclipse.justj.openj
Insertion Sort with n=200 - Random: 108 ms
Insertion Sort with n=200 - Ordered: 0 ms
Insertion Sort with n=200 - Partially Ordered: 1 ms
Insertion Sort with n=200 - Reverse Ordered: 1 ms

Insertion Sort with n=400 - Random: 2 ms
Insertion Sort with n=400 - Ordered: 0 ms
Insertion Sort with n=400 - Partially Ordered: 1 ms
Insertion Sort with n=400 - Reverse Ordered: 3 ms

Insertion Sort with n=800 - Random: 3 ms
Insertion Sort with n=800 - Ordered: 0 ms
Insertion Sort with n=800 - Partially Ordered: 0 ms
Insertion Sort with n=800 - Reverse Ordered: 3 ms

Insertion Sort with n=1600 - Random: 2 ms
Insertion Sort with n=1600 - Ordered: 1 ms
Insertion Sort with n=1600 - Partially Ordered: 1 ms
Insertion Sort with n=1600 - Reverse Ordered: 6 ms

Insertion Sort with n=3200 - Random: 5 ms
Insertion Sort with n=3200 - Ordered: 1 ms
Insertion Sort with n=3200 - Partially Ordered: 1 ms
```