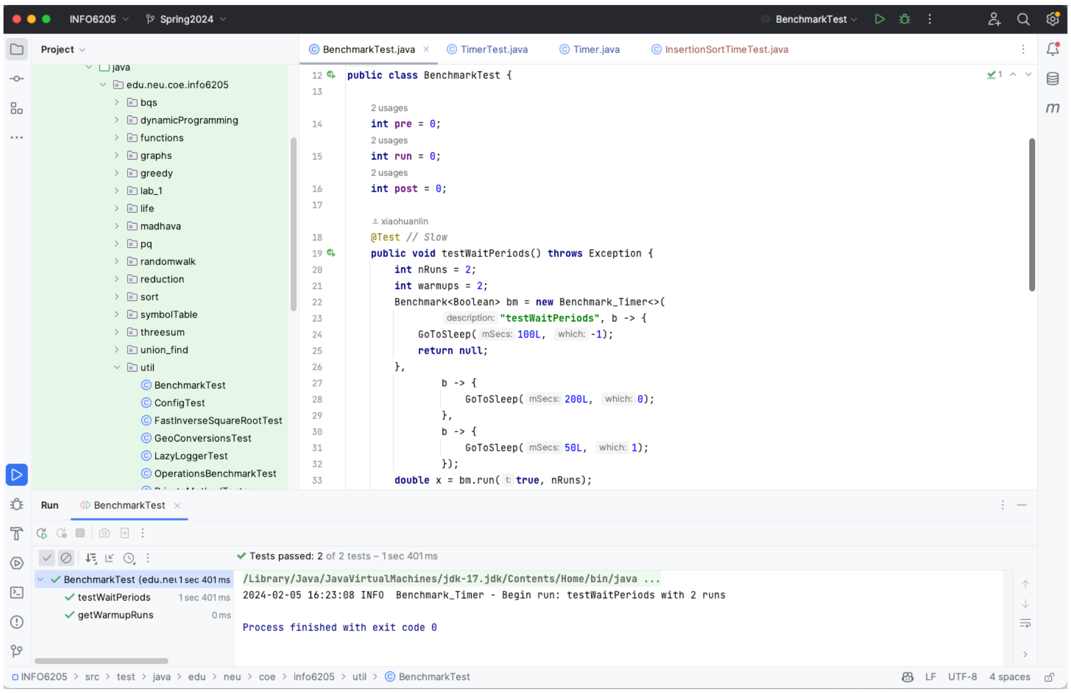
Program Structures and Algorithms
Spring 2024

NAME: Zenan Fan
NUID: 002854067
GITHUB LINK: https://github.com/LearningMach1ne/INFO6205

Task: Assignment 3 (Benchmark)

Benchmark Test and Timer Test Screenshots:

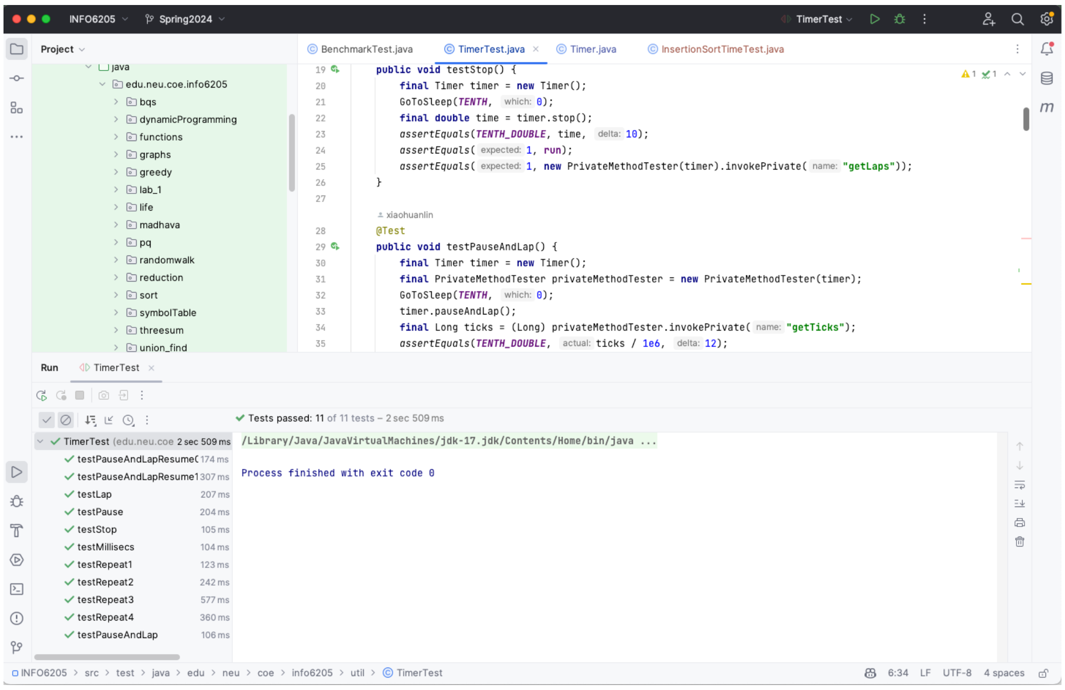


```java
public class BenchmarkTest {

    2 usages
    int pre = 0;
    2 usages
    int run = 0;
    2 usages
    int post = 0;

    xiaohuanlin
    @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);
                });
        double x = bm.run( t: true, nRuns);
```

Tests passed: 2 of 2 tests – 1 sec 401 ms

```
BenchmarkTest (edu.neu  1 sec 401 ms
  testWaitPeriods       1 sec 401 ms
  getWarmupRuns         0 ms
```

```
/Library/Java/JavaVirtualMachines/jdk-17.jdk/Contents/Home/bin/java ...
2024-02-05 16:23:08 INFO  Benchmark_Timer - Begin run: testWaitPeriods with 2 runs

Process finished with exit code 0
```



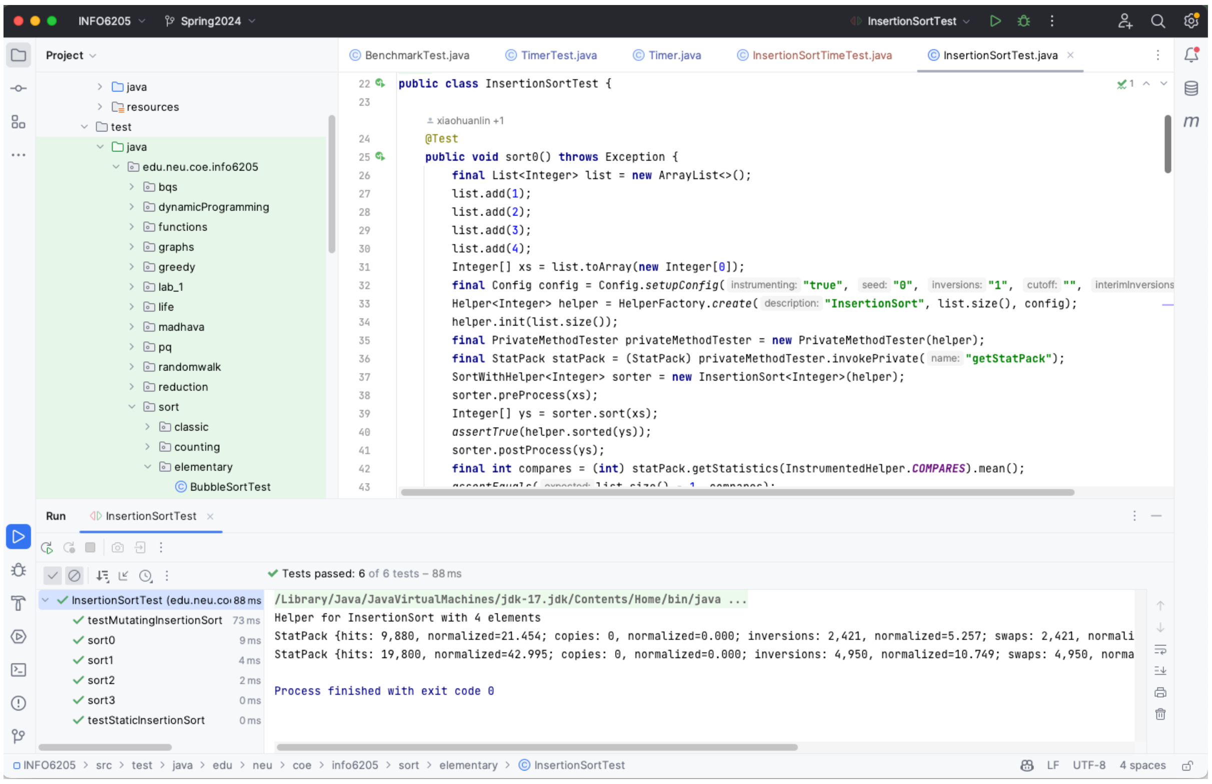
```java
    public void testStop() {
        final Timer timer = new Timer();
        GoToSleep(TENTH,  which: 0);
        final double time = timer.stop();
        assertEquals(TENTH_DOUBLE, time,  delta: 10);
        assertEquals( expected: 1, run);
        assertEquals( expected: 1, new PrivateMethodTester(timer).invokePrivate(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

    xiaohuanlin
    @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");
        assertEquals(TENTH_DOUBLE,  actual: ticks / 1e6,  delta: 12);
```

Tests passed: 11 of 11 tests – 2 sec 509 ms

```
TimerTest (edu.neu.coe  2 sec 509 ms
  testPauseAndLapResume(  174 ms
  testPauseAndLapResume1  307 ms
  testLap                 207 ms
  testPause               204 ms
  testStop                105 ms
  testMillisecs           104 ms
  testRepeat1             123 ms
  testRepeat2             242 ms
  testRepeat3             577 ms
  testRepeat4             360 ms
  testPauseAndLap         106 ms
```

```
/Library/Java/JavaVirtualMachines/jdk-17.jdk/Contents/Home/bin/java ...

Process finished with exit code 0
```

Insertion Sort Test Screenshots:

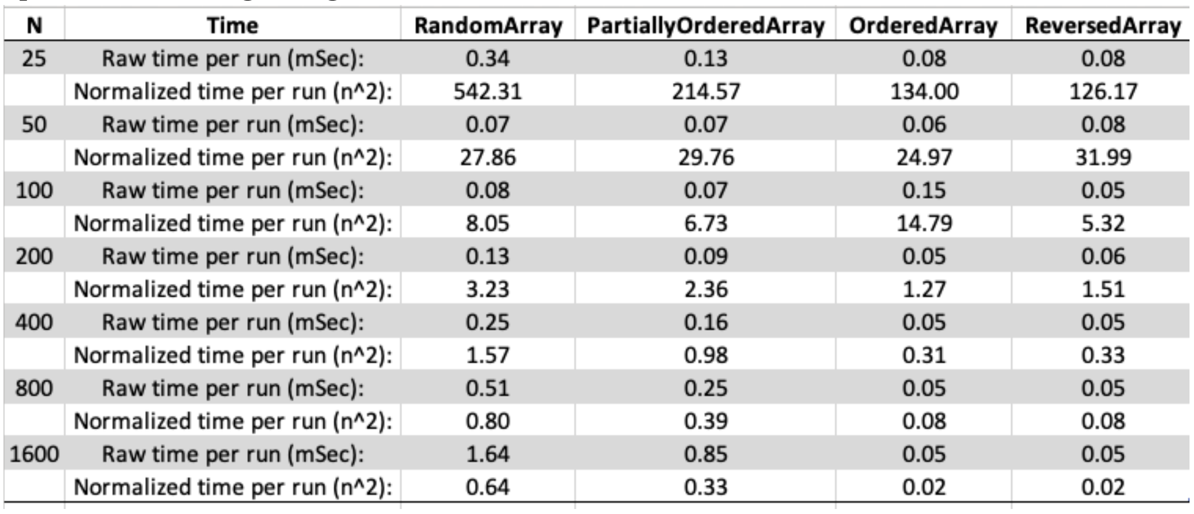


```java
public class InsertionSortTest {

    xiaohuanlin +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( instrumenting: "true",  seed: "0",  inversions: "1",  cutoff: "",  interimInversions
        Helper<Integer> helper = HelperFactory.create( description: "InsertionSort", list.size(), config);
        helper.init(list.size());
        final PrivateMethodTester privateMethodTester = new PrivateMethodTester(helper);
        final StatPack statPack = (StatPack) privateMethodTester.invokePrivate( name: "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();
```

```
✓ Tests passed: 6 of 6 tests – 88 ms

✓ InsertionSortTest (edu.neu.co 88 ms
    ✓ testMutatingInsertionSort 73 ms
    ✓ sort0                      9 ms
    ✓ sort1                      4 ms
    ✓ sort2                      2 ms
    ✓ sort3                      0 ms
    ✓ testStaticInsertionSort    0 ms
```

```
/Library/Java/JavaVirtualMachines/jdk-17.jdk/Contents/Home/bin/java ...
Helper for InsertionSort with 4 elements
StatPack {hits: 9,880, normalized=21.454; copies: 0, normalized=0.000; inversions: 2,421, normalized=5.257; swaps: 2,421, normali
StatPack {hits: 19,800, normalized=42.995; copies: 0, normalized=0.000; inversions: 4,950, normalized=10.749; swaps: 4,950, norma

Process finished with exit code 0
```

Spreadsheet showing timing observations:



N	Time	RandomArray	PartiallyOrderedArray	OrderedArray	ReversedArray
25	Raw time per run (mSec):	0.34	0.13	0.08	0.08
	Normalized time per run (n^2):	542.31	214.57	134.00	126.17
50	Raw time per run (mSec):	0.07	0.07	0.06	0.08
	Normalized time per run (n^2):	27.86	29.76	24.97	31.99
100	Raw time per run (mSec):	0.08	0.07	0.15	0.05
	Normalized time per run (n^2):	8.05	6.73	14.79	5.32
200	Raw time per run (mSec):	0.13	0.09	0.05	0.06
	Normalized time per run (n^2):	3.23	2.36	1.27	1.51
400	Raw time per run (mSec):	0.25	0.16	0.05	0.05
	Normalized time per run (n^2):	1.57	0.98	0.31	0.33
800	Raw time per run (mSec):	0.51	0.25	0.05	0.05
	Normalized time per run (n^2):	0.80	0.39	0.08	0.08
1600	Raw time per run (mSec):	1.64	0.85	0.05	0.05
	Normalized time per run (n^2):	0.64	0.33	0.02	0.02

Conclusion from the observation:

There is not much difference in the normalized time performance between random arrays and partially ordered arrays, whereas ordered and reversed arrays generally show lower normalized times, especially for larger N.
Times for ordered and reversed arrays are generally shorter than those for random and partially ordered arrays. This indicate that insertion sort is more efficient for ordered data.
As N increases, the normalized time decreases, indicating that the insertion sort is more efficient at handling larger datasets.