Program Structures and Algorithms
Spring 2023(SEC 01)

NAME: Shivani Datar
NUID: 002772160

Task:

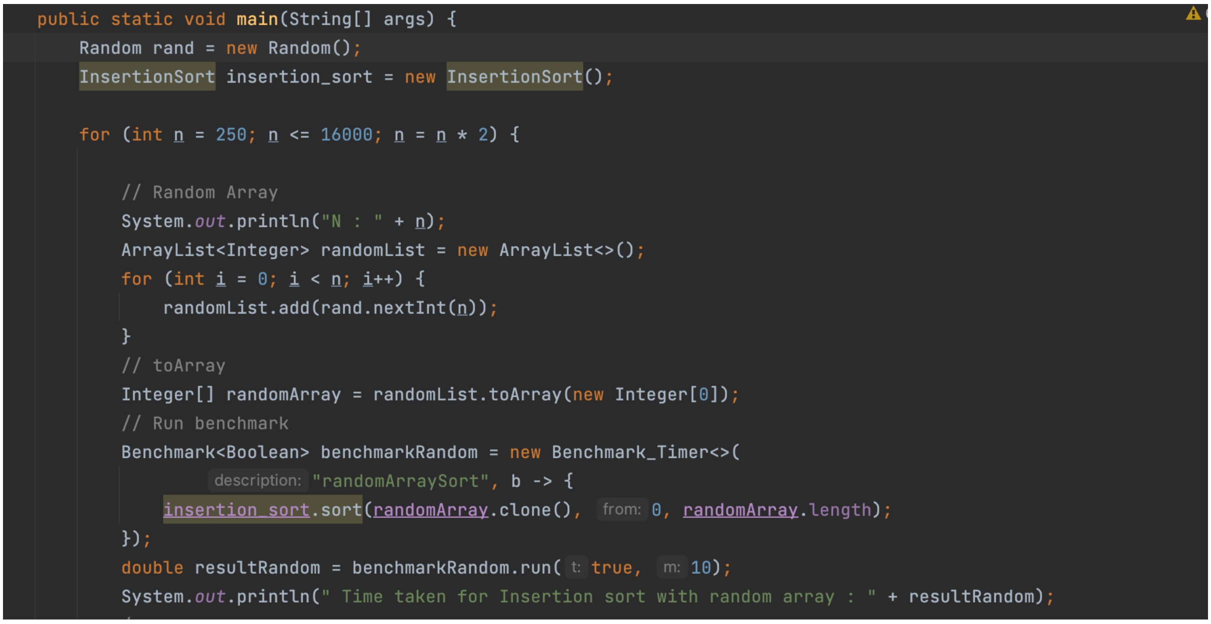
1. Implement three (3) methods (*repeat*, *getClock*, and *toMillisecs*) of a class called Timer.

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. Alternatively, if you are not using instrumenting, then you can write (or copy) your own compare/swap code. Either way, you must run the unit tests in InsertionSortTest.

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.

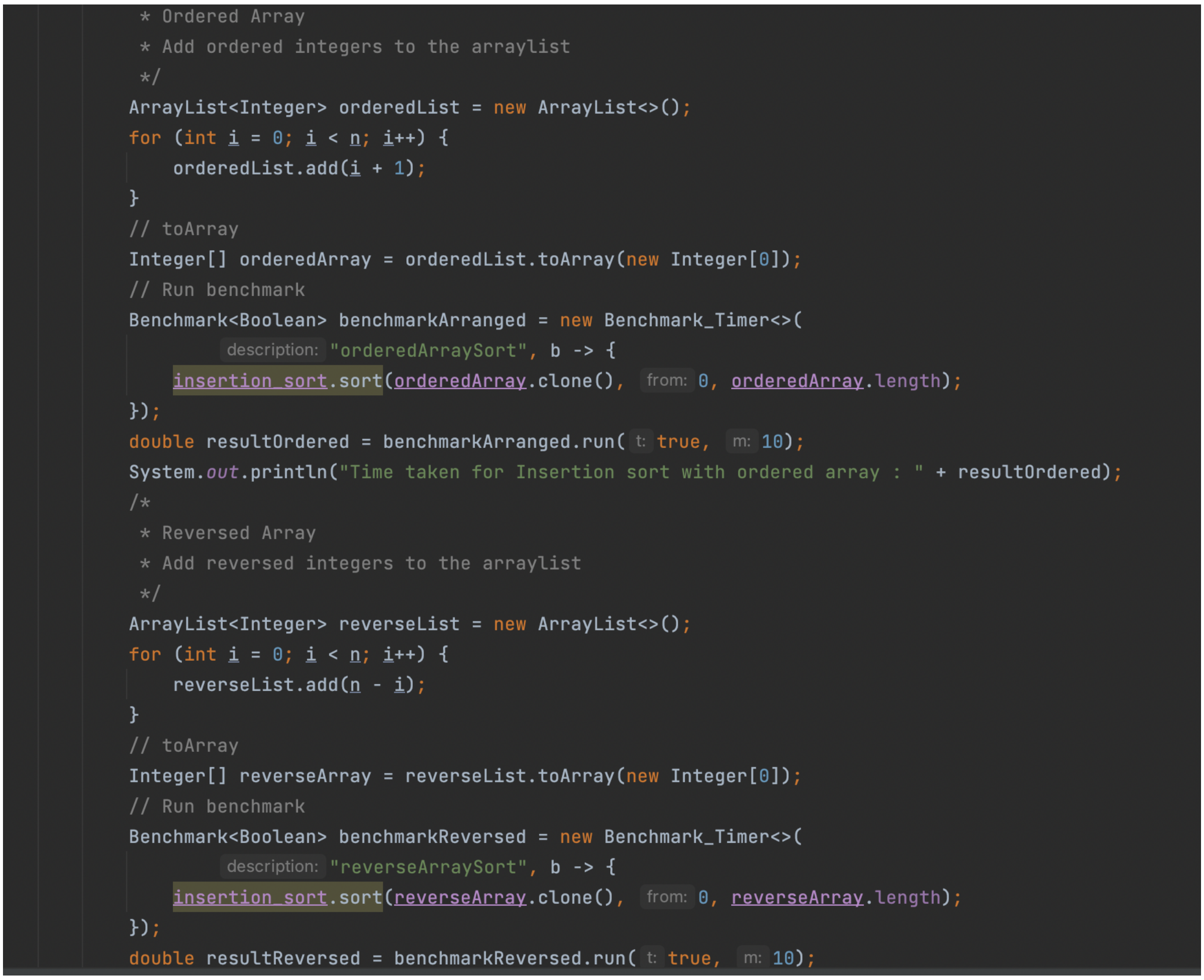
Relationship Conclusion:

For implementing the benchmarking for the insertion sort, I have added a main method inside the Benchmark_Timer class. Here we are creating objects for Random class and InsertionSort class. We first create a List and then convert it to Array and apply the insertion sort logic on it. We will execute this from N = 250 to N = 16000(using doubling method, that is multiplying N by 2).



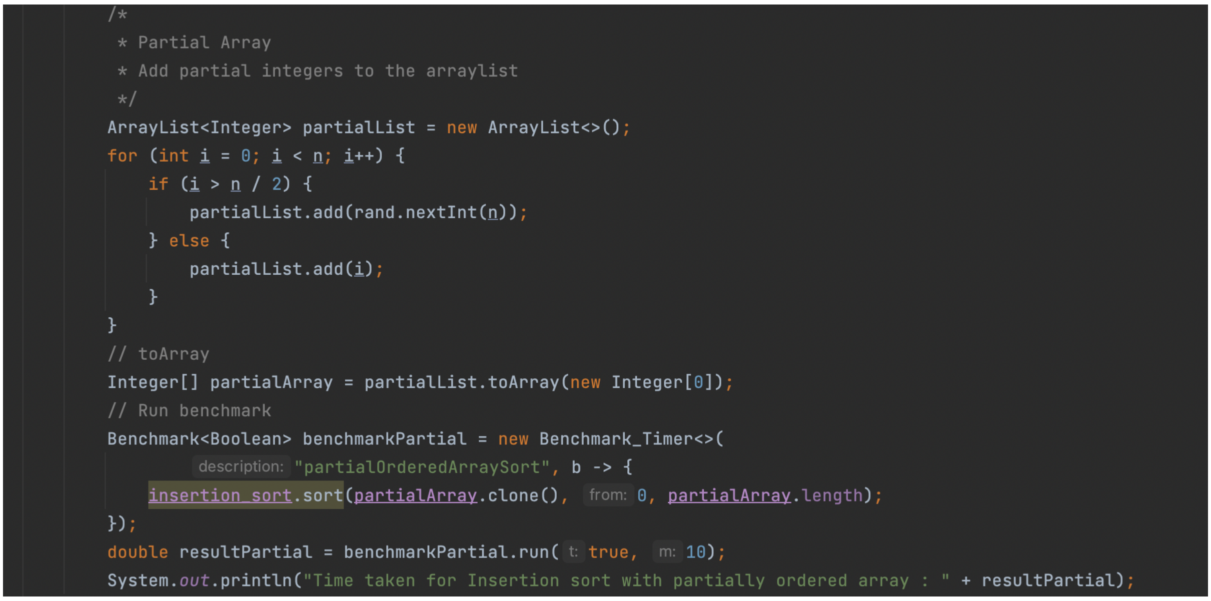
```
public static void main(String[] args) {
    Random rand = new Random();
    InsertionSort insertion_sort = new InsertionSort();

    for (int n = 250; n <= 16000; n = n * 2) {

        // Random Array
        System.out.println("N : " + n);
        ArrayList<Integer> randomList = new ArrayList<>();
        for (int i = 0; i < n; i++) {
            randomList.add(rand.nextInt(n));
        }
        // toArray
        Integer[] randomArray = randomList.toArray(new Integer[0]);
        // Run benchmark
        Benchmark<Boolean> benchmarkRandom = new Benchmark_Timer<>(
                description: "randomArraySort", b -> {
            insertion_sort.sort(randomArray.clone(),  from: 0, randomArray.length);
        });
        double resultRandom = benchmarkRandom.run( t: true,  m: 10);
        System.out.println(" Time taken for Insertion sort with random array : " + resultRandom);
```

For the random array I used rand.nextInt() for creating random Integers in the list. For the ordered array I have add ordered elements from 1 to N in the list. For reverse array from N to 1 added elements in the reverse order.



```
 * Ordered Array
 * Add ordered integers to the arraylist
 */
ArrayList<Integer> orderedList = new ArrayList<>();
for (int i = 0; i < n; i++) {
    orderedList.add(i + 1);
}
// toArray
Integer[] orderedArray = orderedList.toArray(new Integer[0]);
// Run benchmark
Benchmark<Boolean> benchmarkArranged = new Benchmark_Timer<>(
        description: "orderedArraySort", b -> {
    insertion_sort.sort(orderedArray.clone(),  from: 0, orderedArray.length);
});
double resultOrdered = benchmarkArranged.run( t: true,  m: 10);
System.out.println("Time taken for Insertion sort with ordered array : " + resultOrdered);
/*
 * Reversed Array
 * Add reversed integers to the arraylist
 */
ArrayList<Integer> reverseList = new ArrayList<>();
for (int i = 0; i < n; i++) {
    reverseList.add(n - i);
}
// toArray
Integer[] reverseArray = reverseList.toArray(new Integer[0]);
// Run benchmark
Benchmark<Boolean> benchmarkReversed = new Benchmark_Timer<>(
        description: "reverseArraySort", b -> {
    insertion_sort.sort(reverseArray.clone(),  from: 0, reverseArray.length);
});
double resultReversed = benchmarkReversed.run( t: true,  m: 10);
```

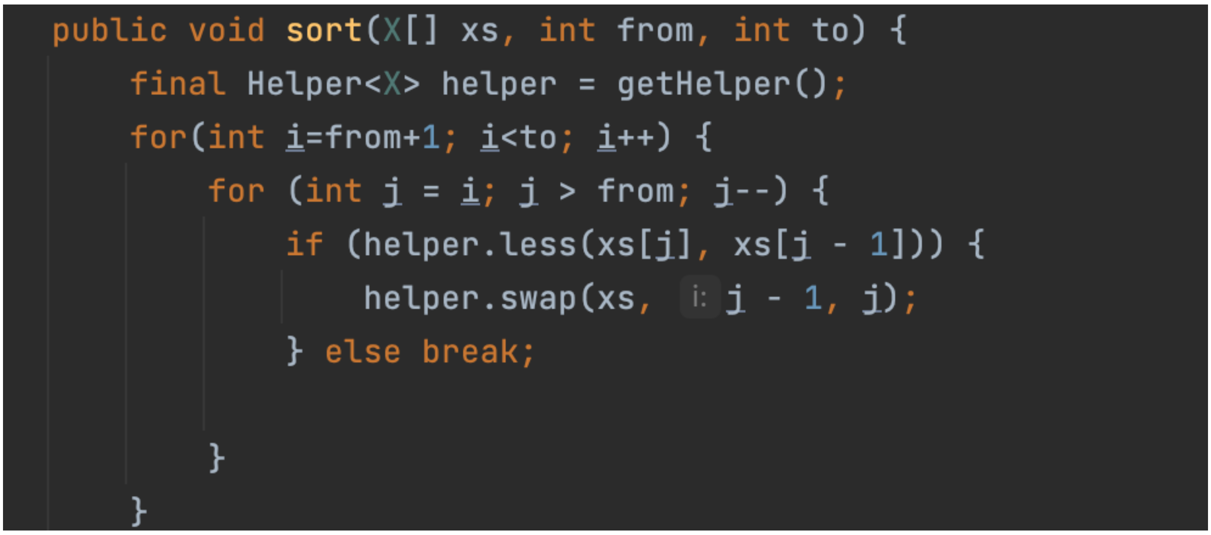
Lastly for the partially ordered array, for the first half of array, it has randomly generated integers and for the second half it has ordered array.



```
/*
 * Partial Array
 * Add partial integers to the arraylist
 */
ArrayList<Integer> partialList = new ArrayList<>();
for (int i = 0; i < n; i++) {
    if (i > n / 2) {
        partialList.add(rand.nextInt(n));
    } else {
        partialList.add(i);
    }
}
// toArray
Integer[] partialArray = partialList.toArray(new Integer[0]);
// Run benchmark
Benchmark<Boolean> benchmarkPartial = new Benchmark_Timer<>(
        description: "partialOrderedArraySort", b -> {
    insertion_sort.sort(partialArray.clone(),  from: 0, partialArray.length);
});
double resultPartial = benchmarkPartial.run( t: true,  m: 10);
System.out.println("Time taken for Insertion sort with partially ordered array : " + resultPartial);
```

Following is the code snippet implementing the Insertion sort algorithm :-

Here I am using 2 methods from the helper interface. We are given the from and to points; in between we need to implement the insertion sort. We compare a xs[j] with xs[j-1], if xs[j] is less than xs[j-1] it means the array is not sorted and I am swapping the elements.

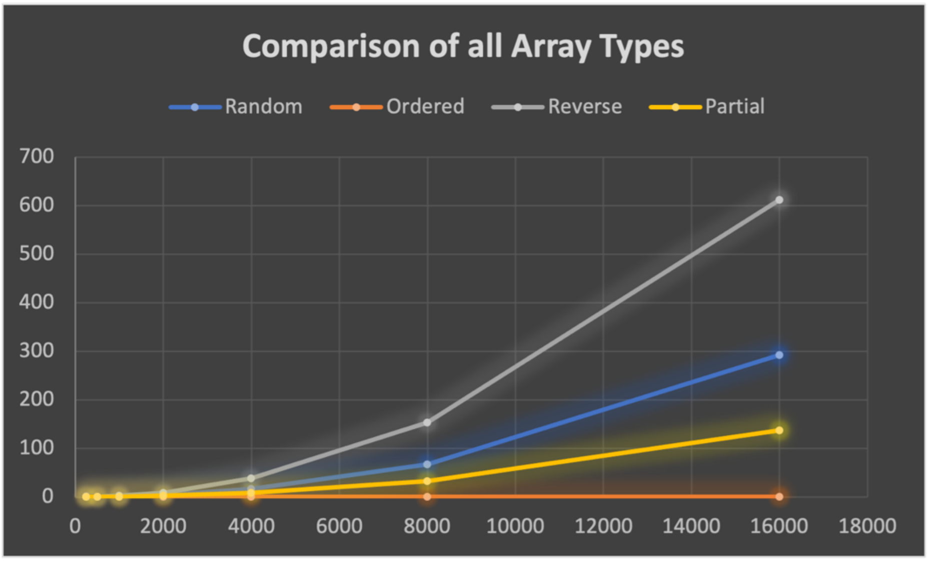


```
public void sort(X[] xs, int from, int to) {
    final Helper<X> helper = getHelper();
    for(int i=from+1; i<to; i++) {
        for (int j = i; j > from; j--) {
            if (helper.less(xs[j], xs[j - 1])) {
                helper.swap(xs, i: j - 1, j);
            } else break;

        }
    }
```

Following are the timings taken by different types of array :-

N	Random	Ordered	Reverse	Partial
250	1.0855876	0.0059414	0.3526124	0.1133542
500	0.2844707	0.0037336	0.5390208	0.1603084
1000	1.0477832	0.0154541	2.0889583	0.4991041
2000	4.1703041	0.0125042	8.2576502	2.124529
4000	16.5528458	0.026325	38.4466748	8.2852959
8000	66.4618998	0.047554	152.741333	32.8193914
16000	292.428913	0.0903001	612.394675	137.337408

The graph below shows that insertion sort takes the most time when the array is reversely sorted because it needs to perform swapping for every subsequent element in the array, which is the worst case scenario for an insertion sort algorithm. It also takes the least amount of time when the array is already sorted.

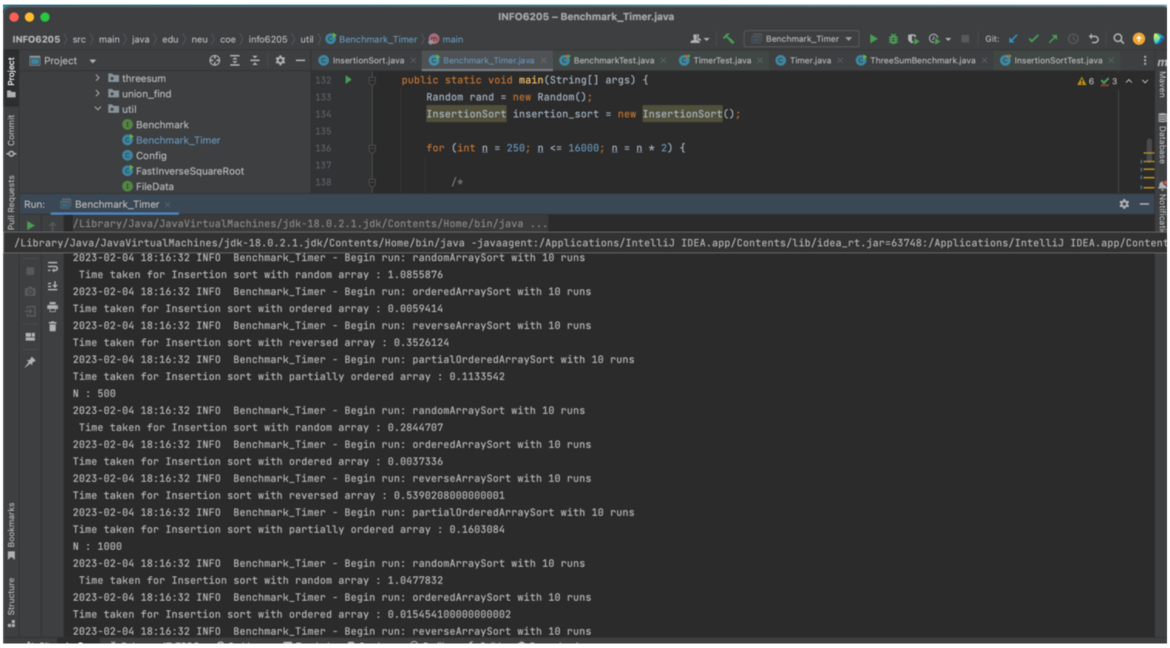


Hence for 4 types of array we can conclude following relation:-

t (Ordered Array) < t (Partially Ordered) < t (Random Array) < t (Reverse Array)

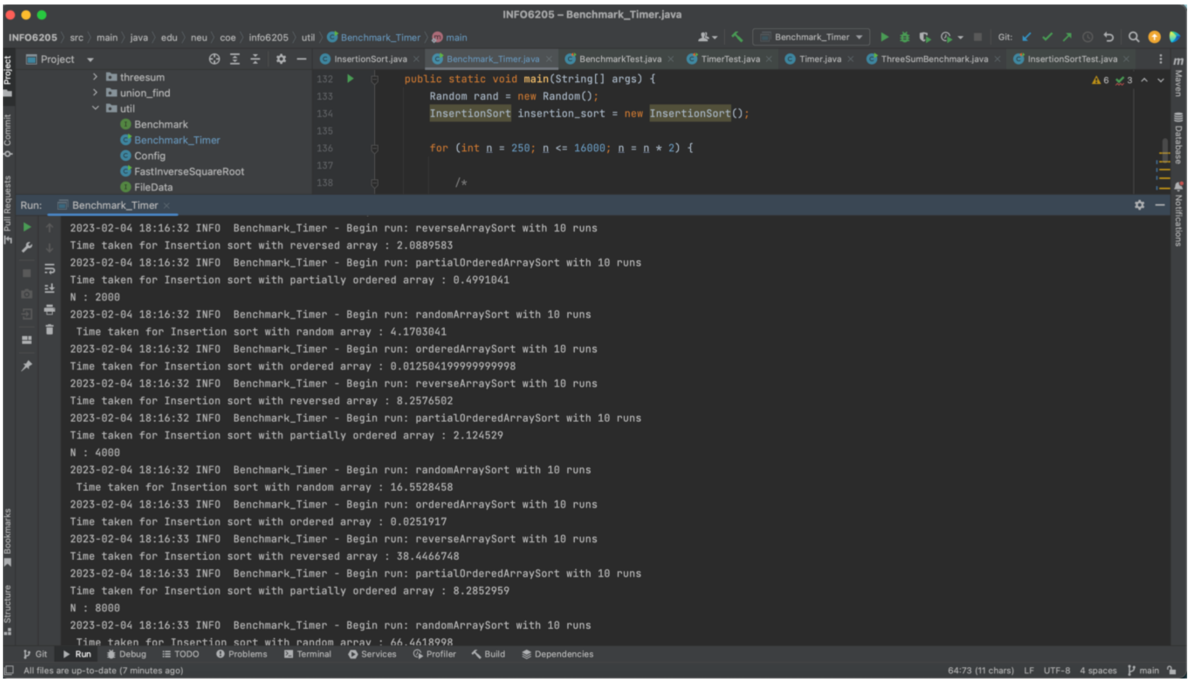
Evidence to support that conclusion:

Output for the timings of all 4 types of array :-



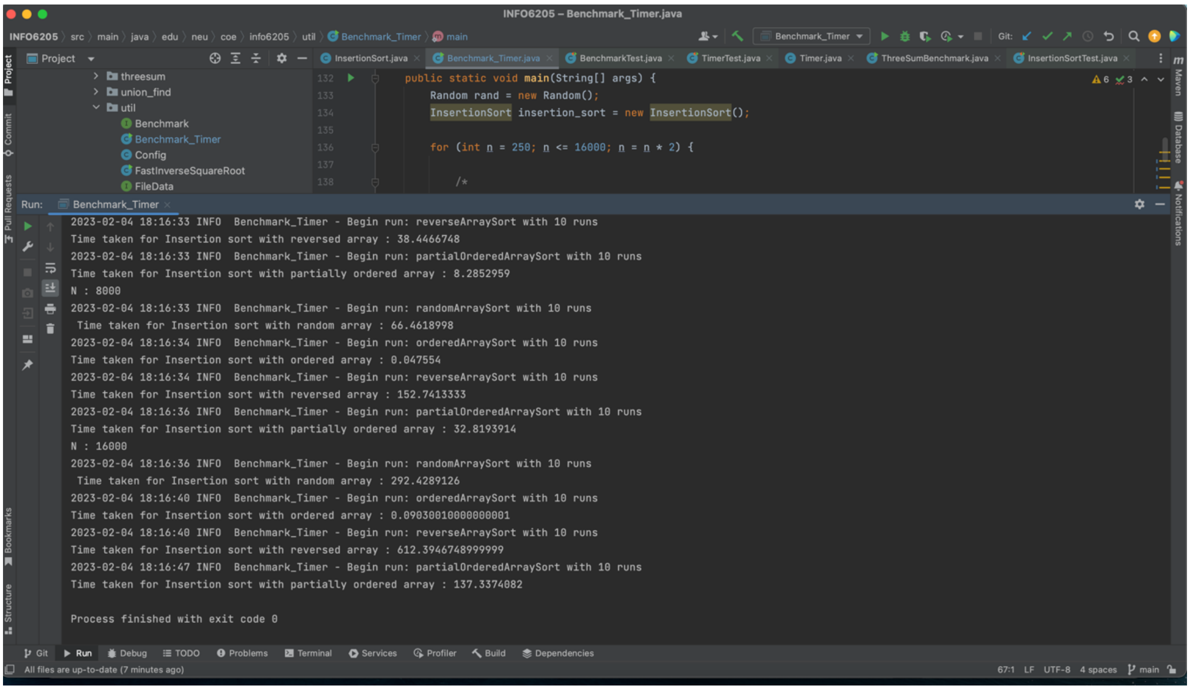
```java
    public static void main(String[] args) {
        Random rand = new Random();
        InsertionSort insertion_sort = new InsertionSort();

        for (int n = 250; n <= 16000; n = n * 2) {

            /*
```

```
/Library/Java/JavaVirtualMachines/jdk-18.0.2.1.jdk/Contents/Home/bin/java ...
/Library/Java/JavaVirtualMachines/jdk-18.0.2.1.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=63748:/Applications/IntelliJ IDEA.app/Content
2023-02-04 18:16:32 INFO  Benchmark_Timer - Begin run: randomArraySort with 10 runs
 Time taken for Insertion sort with random array : 1.0855876
2023-02-04 18:16:32 INFO  Benchmark_Timer - Begin run: orderedArraySort with 10 runs
Time taken for Insertion sort with ordered array : 0.0059414
2023-02-04 18:16:32 INFO  Benchmark_Timer - Begin run: reverseArraySort with 10 runs
Time taken for Insertion sort with reversed array : 0.3526124
2023-02-04 18:16:32 INFO  Benchmark_Timer - Begin run: partialOrderedArraySort with 10 runs
Time taken for Insertion sort with partially ordered array : 0.1133542
N : 500
2023-02-04 18:16:32 INFO  Benchmark_Timer - Begin run: randomArraySort with 10 runs
 Time taken for Insertion sort with random array : 0.2844707
2023-02-04 18:16:32 INFO  Benchmark_Timer - Begin run: orderedArraySort with 10 runs
Time taken for Insertion sort with ordered array : 0.0037336
2023-02-04 18:16:32 INFO  Benchmark_Timer - Begin run: reverseArraySort with 10 runs
Time taken for Insertion sort with reversed array : 0.5390208000000001
2023-02-04 18:16:32 INFO  Benchmark_Timer - Begin run: partialOrderedArraySort with 10 runs
Time taken for Insertion sort with partially ordered array : 0.1603084
N : 1000
2023-02-04 18:16:32 INFO  Benchmark_Timer - Begin run: randomArraySort with 10 runs
 Time taken for Insertion sort with random array : 1.0477832
2023-02-04 18:16:32 INFO  Benchmark_Timer - Begin run: orderedArraySort with 10 runs
Time taken for Insertion sort with ordered array : 0.015454100000000002
2023-02-04 18:16:32 INFO  Benchmark_Timer - Begin run: reverseArraySort with 10 runs
```



```java
public static void main(String[] args) {
    Random rand = new Random();
    InsertionSort insertion_sort = new InsertionSort();

    for (int n = 250; n <= 16000; n = n * 2) {

        /*
```

```
2023-02-04 18:16:32 INFO  Benchmark_Timer - Begin run: reverseArraySort with 10 runs
Time taken for Insertion sort with reversed array : 2.0889583
2023-02-04 18:16:32 INFO  Benchmark_Timer - Begin run: partialOrderedArraySort with 10 runs
Time taken for Insertion sort with partially ordered array : 0.4991041
N : 2000
2023-02-04 18:16:32 INFO  Benchmark_Timer - Begin run: randomArraySort with 10 runs
 Time taken for Insertion sort with random array : 4.1703041
2023-02-04 18:16:32 INFO  Benchmark_Timer - Begin run: orderedArraySort with 10 runs
Time taken for Insertion sort with ordered array : 0.012504199999999998
2023-02-04 18:16:32 INFO  Benchmark_Timer - Begin run: reverseArraySort with 10 runs
Time taken for Insertion sort with reversed array : 8.2576502
2023-02-04 18:16:32 INFO  Benchmark_Timer - Begin run: partialOrderedArraySort with 10 runs
Time taken for Insertion sort with partially ordered array : 2.124529
N : 4000
2023-02-04 18:16:32 INFO  Benchmark_Timer - Begin run: randomArraySort with 10 runs
 Time taken for Insertion sort with random array : 16.5528458
2023-02-04 18:16:33 INFO  Benchmark_Timer - Begin run: orderedArraySort with 10 runs
Time taken for Insertion sort with ordered array : 0.0251917
2023-02-04 18:16:33 INFO  Benchmark_Timer - Begin run: reverseArraySort with 10 runs
Time taken for Insertion sort with reversed array : 38.4466748
2023-02-04 18:16:33 INFO  Benchmark_Timer - Begin run: partialOrderedArraySort with 10 runs
Time taken for Insertion sort with partially ordered array : 8.2852959
N : 8000
2023-02-04 18:16:33 INFO  Benchmark_Timer - Begin run: randomArraySort with 10 runs
 Time taken for Insertion sort with random array : 66.4618998
```



```java
public static void main(String[] args) {
    Random rand = new Random();
    InsertionSort insertion_sort = new InsertionSort();

    for (int n = 250; n <= 16000; n = n * 2) {

        /*
```

```
2023-02-04 18:16:33 INFO  Benchmark_Timer - Begin run: reverseArraySort with 10 runs
Time taken for Insertion sort with reversed array : 38.4466748
2023-02-04 18:16:33 INFO  Benchmark_Timer - Begin run: partialOrderedArraySort with 10 runs
Time taken for Insertion sort with partially ordered array : 8.2852959
N : 8000
2023-02-04 18:16:33 INFO  Benchmark_Timer - Begin run: randomArraySort with 10 runs
 Time taken for Insertion sort with random array : 66.4618998
2023-02-04 18:16:34 INFO  Benchmark_Timer - Begin run: orderedArraySort with 10 runs
Time taken for Insertion sort with ordered array : 0.047554
2023-02-04 18:16:34 INFO  Benchmark_Timer - Begin run: reverseArraySort with 10 runs
Time taken for Insertion sort with reversed array : 152.7413333
2023-02-04 18:16:36 INFO  Benchmark_Timer - Begin run: partialOrderedArraySort with 10 runs
Time taken for Insertion sort with partially ordered array : 32.8193914
N : 16000
2023-02-04 18:16:36 INFO  Benchmark_Timer - Begin run: randomArraySort with 10 runs
 Time taken for Insertion sort with random array : 292.4289126
2023-02-04 18:16:40 INFO  Benchmark_Timer - Begin run: orderedArraySort with 10 runs
Time taken for Insertion sort with ordered array : 0.09030010000000001
2023-02-04 18:16:40 INFO  Benchmark_Timer - Begin run: reverseArraySort with 10 runs
Time taken for Insertion sort with reversed array : 612.3946748999999
2023-02-04 18:16:47 INFO  Benchmark_Timer - Begin run: partialOrderedArraySort with 10 runs
Time taken for Insertion sort with partially ordered array : 137.3374082

Process finished with exit code 0
```

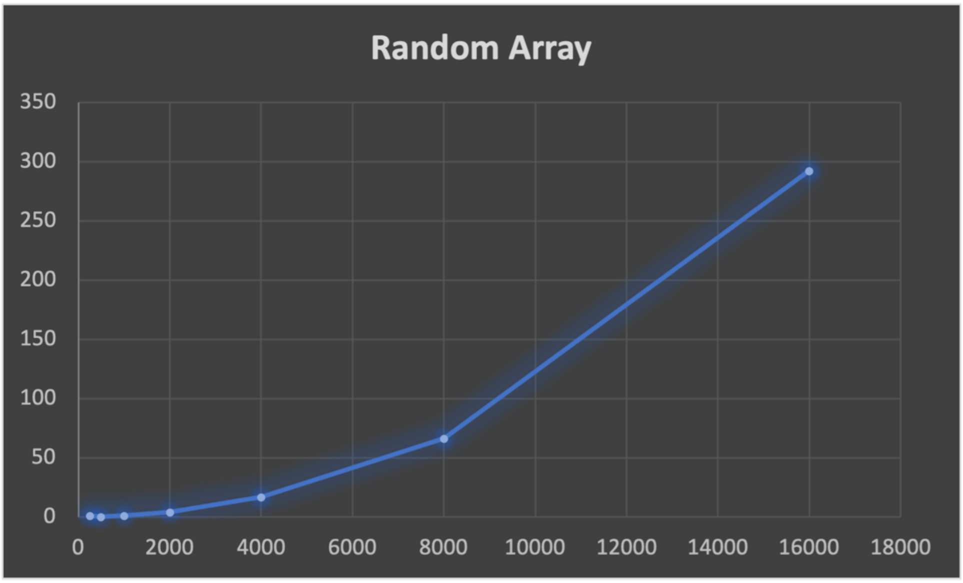
Graphical Representation:

Graph for Random Array :-

Observations:-

N	Random
250	1.0855876
500	0.2844707
1000	1.0477832

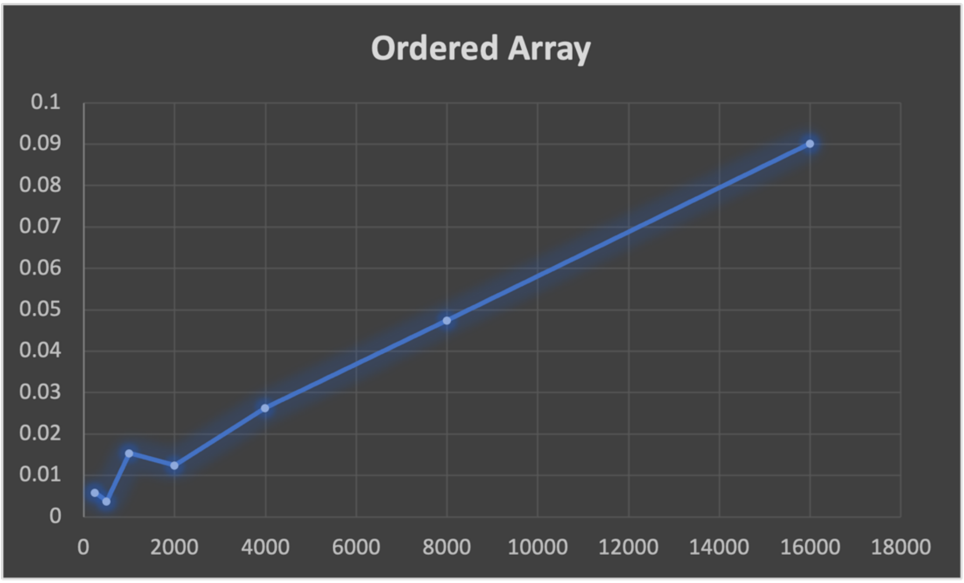
2000	4.1703041
4000	16.5528458
8000	66.4618998
16000	292.428913



Graph for Ordered Array:-

Observations:-

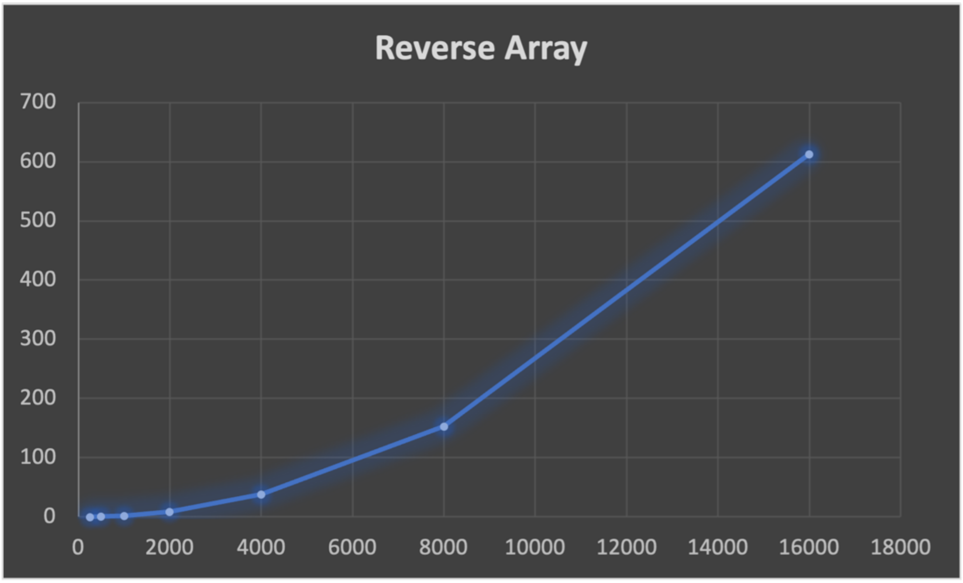
N	Ordered
250	0.0059414
500	0.0037336
1000	0.0154541
2000	0.0125042
4000	0.026325
8000	0.047554
16000	0.0903001



Graph for Reverse Array:-

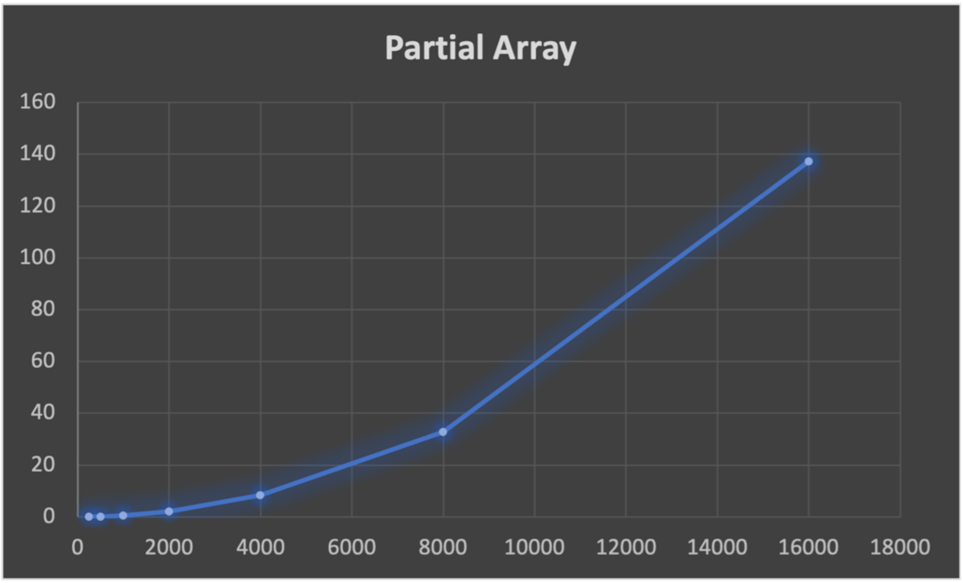
Observations:-

N	Reverse
250	0.3526124
500	0.5390208
1000	2.0889583
2000	8.2576502
4000	38.4466748
8000	152.741333
16000	612.394675

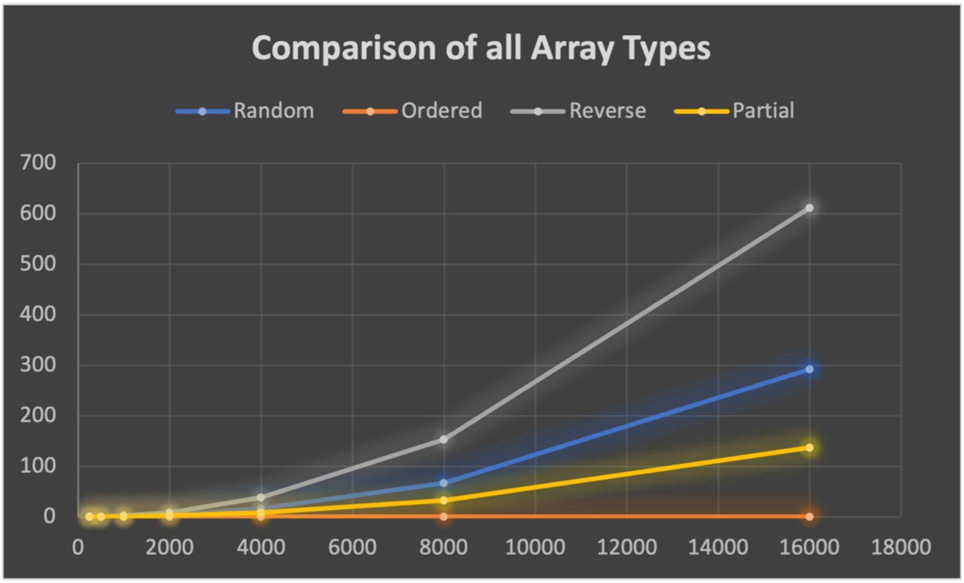


Graph for Partially Ordered Array:-

N	Partial
250	0.1133542
500	0.1603084
1000	0.4991041
2000	2.124529
4000	8.2852959
8000	32.8193914
16000	137.337408

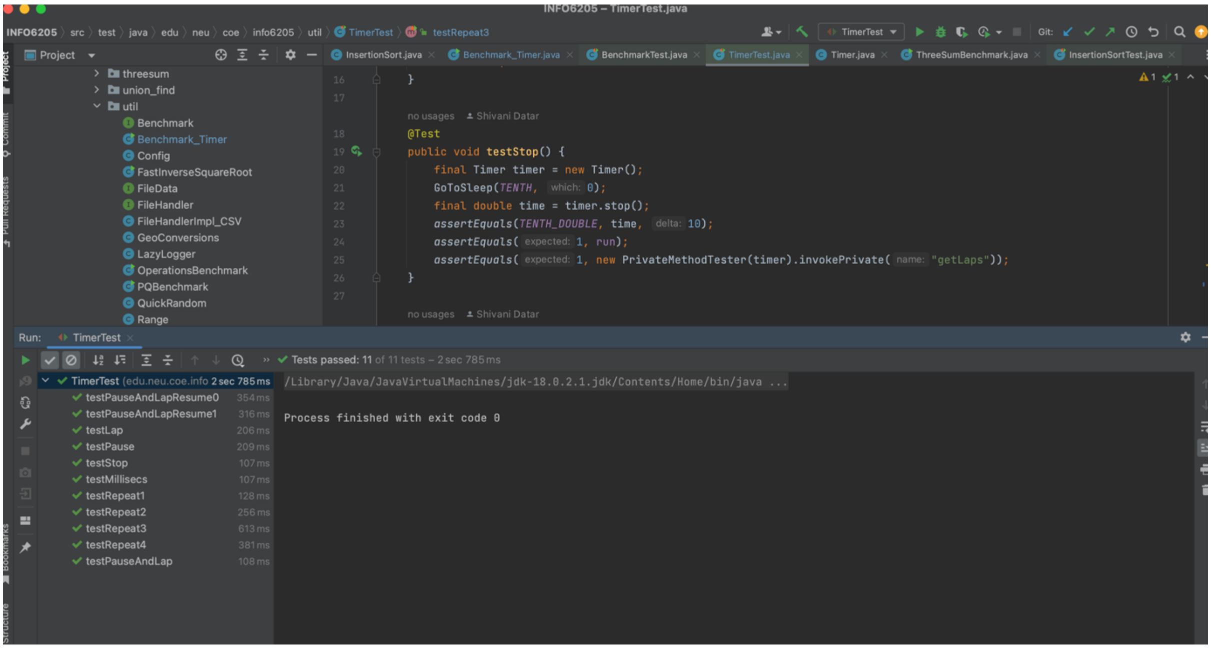


Graph For All array comparison: -



Unit Test Screenshots:

Screenshot for timer test cases running :-

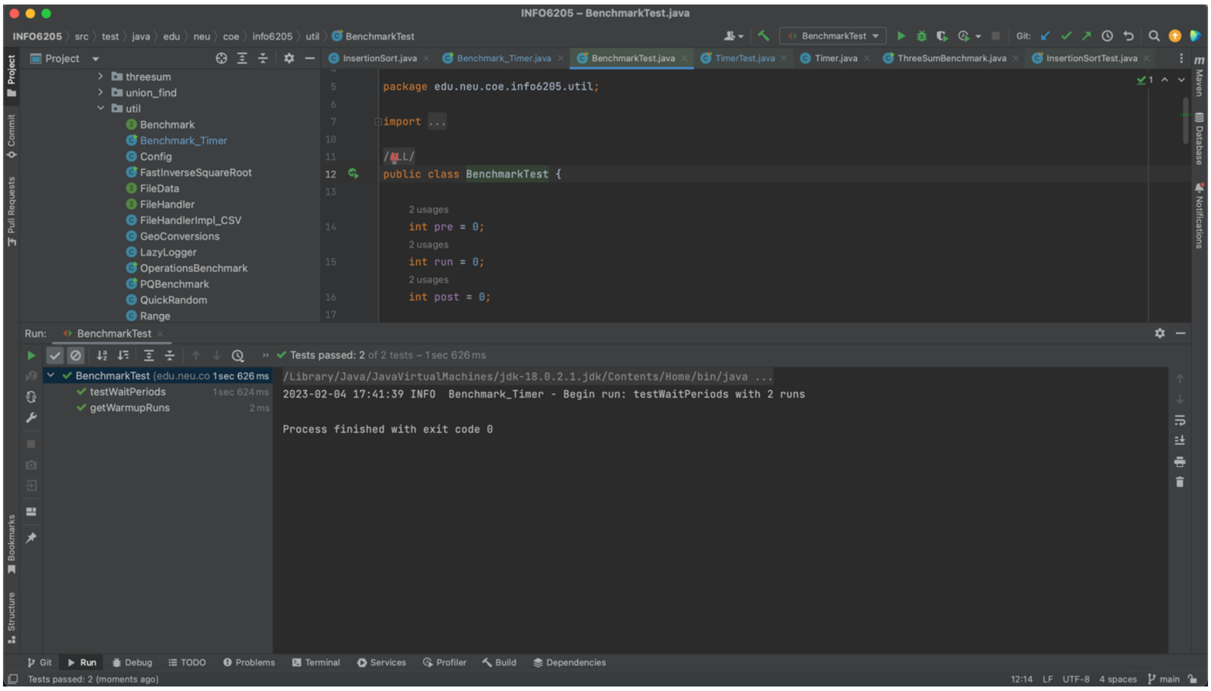


```
}

no usages  Shivani Datar
@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"));
}

no usages  Shivani Datar
```

```
Tests passed: 11 of 11 tests – 2 sec 785 ms

TimerTest (edu.neu.coe.info 2 sec 785 ms
    testPauseAndLapResume0   354 ms
    testPauseAndLapResume1   316 ms
    testLap                  206 ms
    testPause                209 ms
    testStop                 107 ms
    testMillisecs            107 ms
    testRepeat1              128 ms
    testRepeat2              256 ms
    testRepeat3              613 ms
    testRepeat4              381 ms
    testPauseAndLap          108 ms
```

```
/Library/Java/JavaVirtualMachines/jdk-18.0.2.1.jdk/Contents/Home/bin/java ...

Process finished with exit code 0
```

Screenshot for BenchMark_Timer testcases running :-



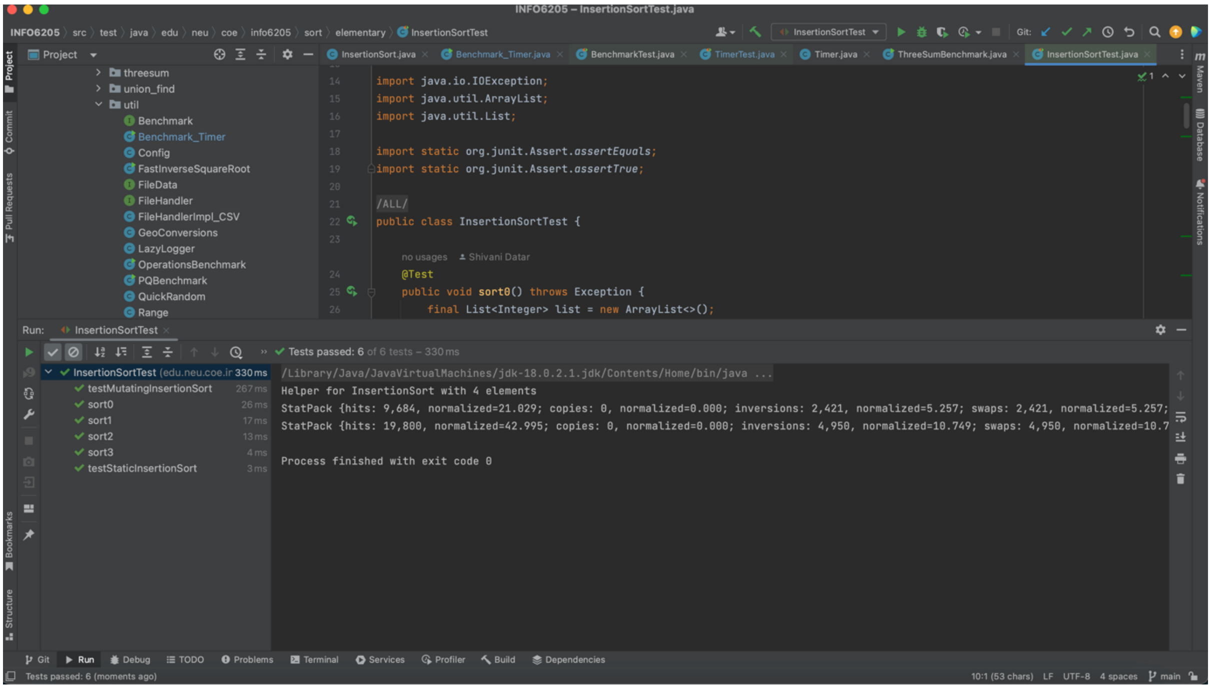
```
package edu.neu.coe.info6205.util;

import ...

/LL/
public class BenchmarkTest {

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

```
Tests passed: 2 of 2 tests – 1 sec 626 ms

BenchmarkTest (edu.neu.co 1 sec 626 ms
    testWaitPeriods   1 sec 624 ms
    getWarmupRuns     2 ms
```

```
/Library/Java/JavaVirtualMachines/jdk-18.0.2.1.jdk/Contents/Home/bin/java ...
2023-02-04 17:41:39 INFO  Benchmark_Timer - Begin run: testWaitPeriods with 2 runs

Process finished with exit code 0
```

Tests passed: 2 (moments ago)

Screenshot for InsertionSort testcases running :-



```java
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertTrue;

/ALL/
public class InsertionSortTest {

    no usages   Shivani Datar
    @Test
    public void sort0() throws Exception {
        final List<Integer> list = new ArrayList<>();
```

✓ Tests passed: 6 of 6 tests – 330 ms

```
✓ InsertionSortTest (edu.neu.coe.ir 330 ms
    ✓ testMutatingInsertionSort   267 ms
    ✓ sort0                        26 ms
    ✓ sort1                        17 ms
    ✓ sort2                        13 ms
    ✓ sort3                         4 ms
    ✓ testStaticInsertionSort       3 ms
```

```
/Library/Java/JavaVirtualMachines/jdk-18.0.2.1.jdk/Contents/Home/bin/java ...
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=10.7

Process finished with exit code 0
```

Tests passed: 6 (moments ago)