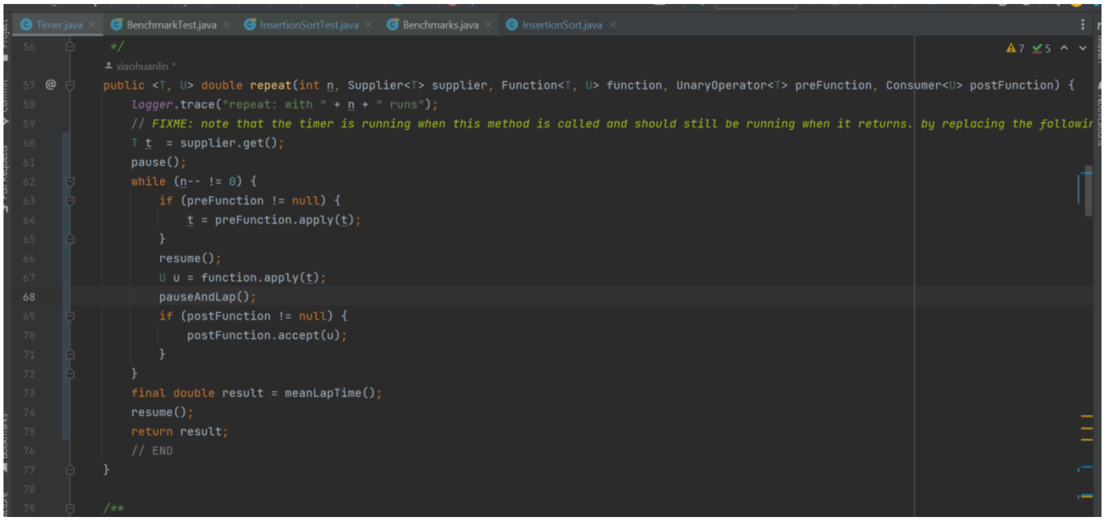
PSA – Assignment 3

Benchmark

Name: Vedantini Dilip Gaikwad
NUID: 002998254

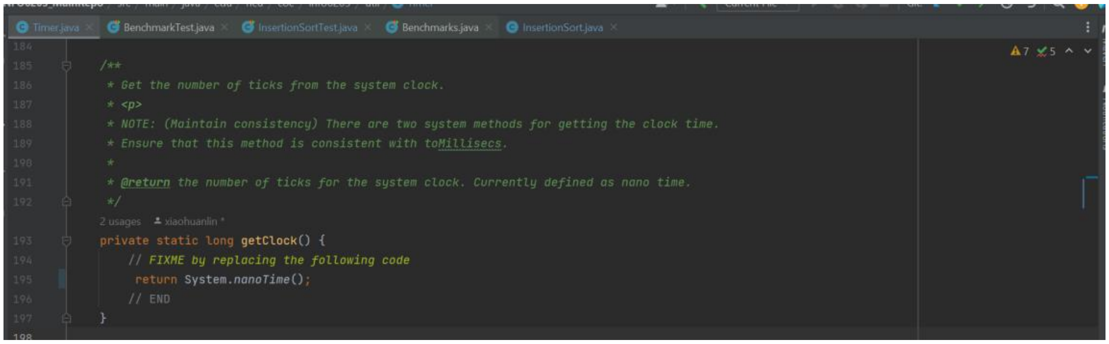
- **Part 1**

repeat method



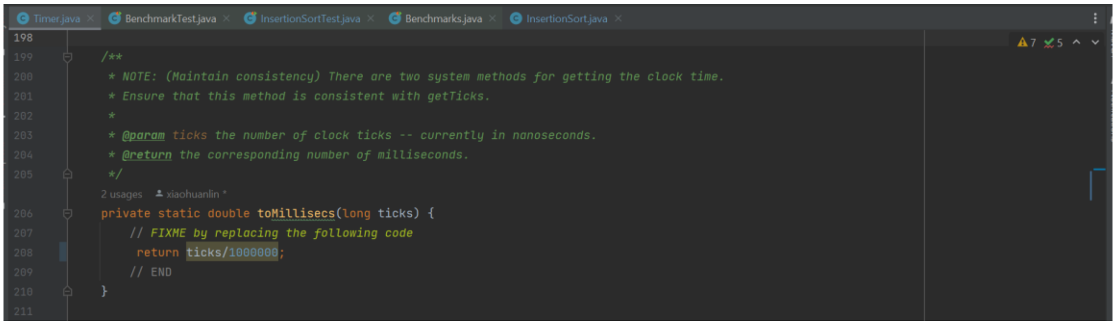
```java
    */
public <T, U> double repeat(int n, Supplier<T> supplier, Function<T, U> function, UnaryOperator<T> preFunction, Consumer<U> postFunction) {
    logger.trace("repeat: with " + n + " runs");
    // FIXME: note that the timer is running when this method is called and should still be running when it returns. by replacing the followir
    T t  = supplier.get();
    pause();
    while (n-- != 0) {
        if (preFunction != null) {
            t = preFunction.apply(t);
        }
        resume();
        U u = function.apply(t);
        pauseAndLap();
        if (postFunction != null) {
            postFunction.accept(u);
        }
    }
    final double result = meanLapTime();
    resume();
    return result;
    // END
}

/**
```

getClock method



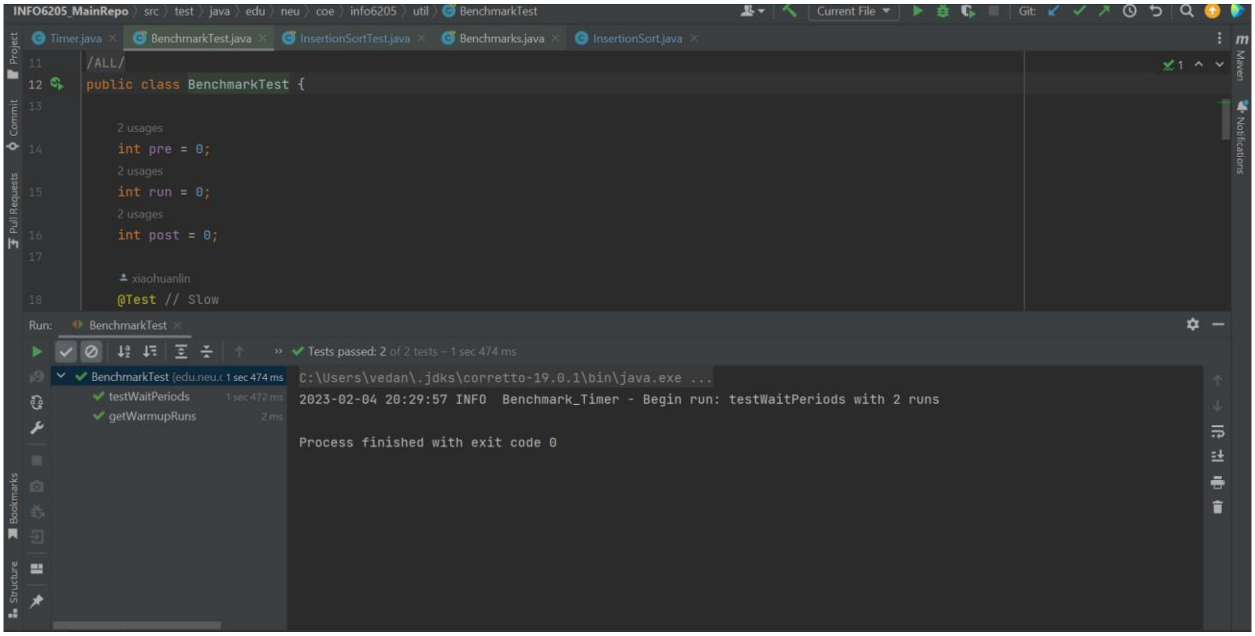
```java
/**
 * Get the number of ticks from the system clock.
 * <p>
 * NOTE: (Maintain consistency) There are two system methods for getting the clock time.
 * Ensure that this method is consistent with toMillisecs.
 *
 * @return the number of ticks for the system clock. Currently defined as nano time.
 */
private static long getClock() {
    // FIXME by replacing the following code
     return System.nanoTime();
    // END
}
```

toMillisecs method



```java
/**
 * NOTE: (Maintain consistency) There are two system methods for getting the clock time.
 * Ensure that this method is consistent with getTicks.
 *
 * @param ticks the number of clock ticks -- currently in nanoseconds.
 * @return the corresponding number of milliseconds.
 */
private static double toMillisecs(long ticks) {
    // FIXME by replacing the following code
     return ticks/1000000;
    // END
}
```

Benchmark Test



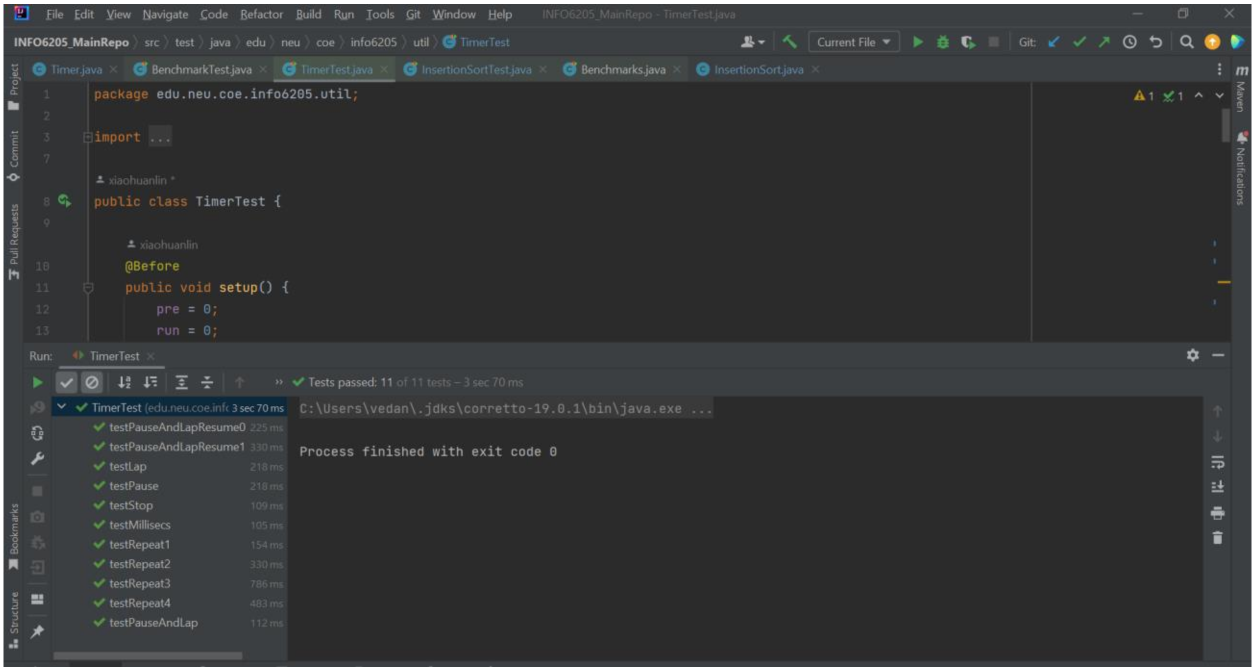
```
/ALL/
public class BenchmarkTest {

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

    xiaohuanlin
    @Test // Slow
```

✓ Tests passed: 2 of 2 tests – 1 sec 474 ms

```
✓ BenchmarkTest (edu.neu.c 1 sec 474 ms
    ✓ testWaitPeriods      1 sec 472 ms
    ✓ getWarmupRuns               2 ms
```

```
C:\Users\vedan\.jdks\corretto-19.0.1\bin\java.exe ...
2023-02-04 20:29:57 INFO  Benchmark_Timer - Begin run: testWaitPeriods with 2 runs

Process finished with exit code 0
```

Timer Test


File Edit View Navigate Code Refactor Build Run Tools Git Window Help
INFO6205_MainRepo - TimerTest.java
INFO6205_MainRepo 〉 src 〉 test 〉 java 〉 edu 〉 neu 〉 coe 〉 info6205 〉 util 〉 TimerTest
Current File
Git:
Timer.java
BenchmarkTest.java
TimerTest.java
InsertionSortTest.java
Benchmarks.java
InsertionSort.java
Project
Commit
Pull Requests
Bookmarks
Structure
Maven
Notifications
Run:
TimerTest

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

import ...

xiaohuanlin *
public class TimerTest {

    xiaohuanlin
    @Before
    public void setup() {
        pre = 0;
        run = 0;
```

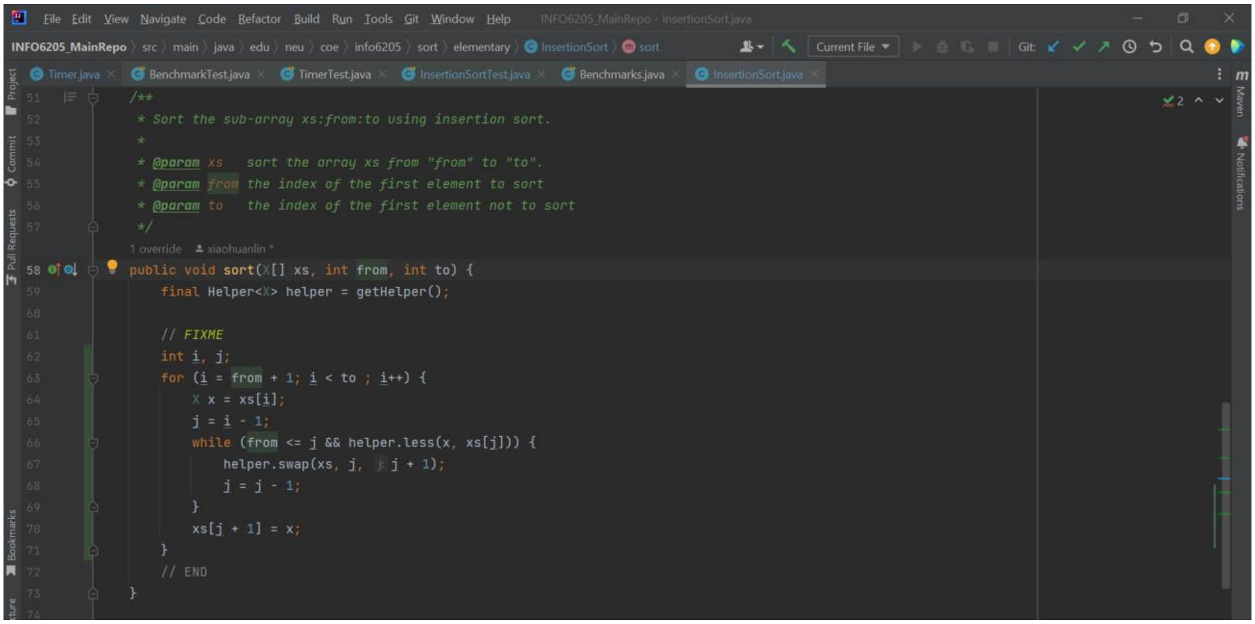
✓ Tests passed: 11 of 11 tests – 3 sec 70 ms

```
✓ TimerTest (edu.neu.coe.infc 3 sec 70 ms
    ✓ testPauseAndLapResume0  225 ms
    ✓ testPauseAndLapResume1  330 ms
    ✓ testLap                 218 ms
    ✓ testPause               218 ms
    ✓ testStop                109 ms
    ✓ testMillisecs           105 ms
    ✓ testRepeat1             154 ms
    ✓ testRepeat2             330 ms
    ✓ testRepeat3             786 ms
    ✓ testRepeat4             483 ms
    ✓ testPauseAndLap         112 ms
```

```
C:\Users\vedan\.jdks\corretto-19.0.1\bin\java.exe ...

Process finished with exit code 0
```

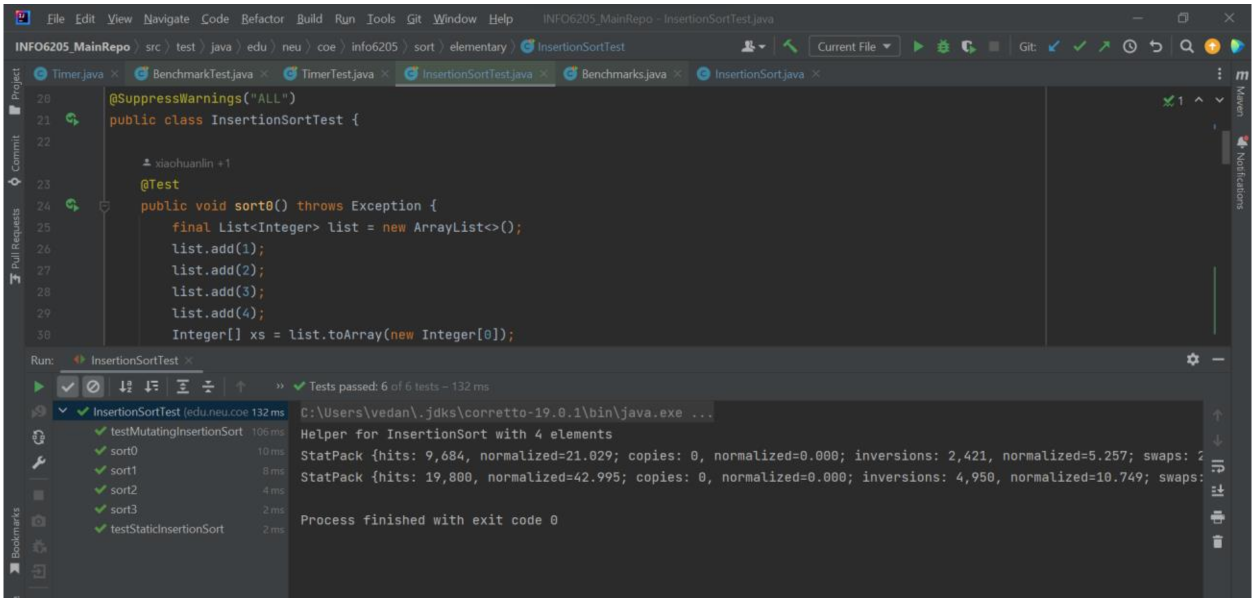
- **Part 2**

InsertionSort implementation


File Edit View Navigate Code Refactor Build Run Tools Git Window Help
INFO6205_MainRepo - InsertionSort.java
INFO6205_MainRepo › src › main › java › edu › neu › coe › info6205 › sort › elementary › InsertionSort › sort
Current File
Timer.java
BenchmarkTest.java
TimerTest.java
InsertionSortTest.java
Benchmarks.java
InsertionSort.java
1 override
xiaohuanlin *

```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();

    // FIXME
    int i, j;
    for (i = from + 1; i < to ; i++) {
        X x = xs[i];
        j = i - 1;
        while (from <= j && helper.less(x, xs[j])) {
            helper.swap(xs, j, j + 1);
            j = j - 1;
        }
        xs[j + 1] = x;
    }
    // END
}
```

InsertionSort Test cases


File Edit View Navigate Code Refactor Build Run Tools Git Window Help
INFO6205_MainRepo - InsertionSortTest.java
INFO6205_MainRepo › src › test › java › edu › neu › coe › info6205 › sort › elementary › InsertionSortTest
Current File
Timer.java
BenchmarkTest.java
TimerTest.java
InsertionSortTest.java
Benchmarks.java
InsertionSort.java
xiaohuanlin +1
Run:
InsertionSortTest

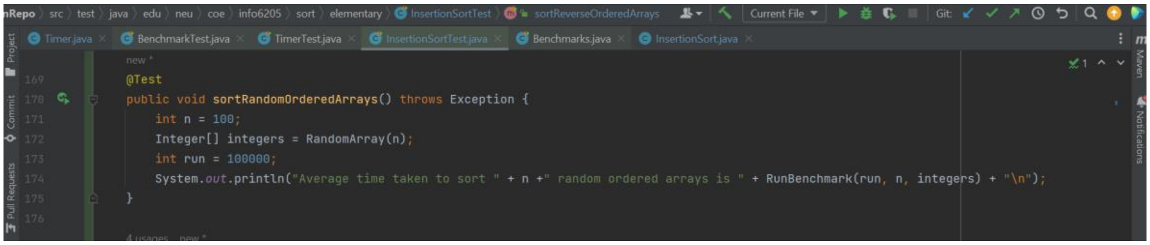
```java
@SuppressWarnings("ALL")
public class InsertionSortTest {

    @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]);
```

```
✓ Tests passed: 6 of 6 tests – 132 ms

✓ InsertionSortTest (edu.neu.coe 132 ms
    ✓ testMutatingInsertionSort  106 ms
    ✓ sort0                       10 ms
    ✓ sort1                        8 ms
    ✓ sort2                        4 ms
    ✓ sort3                        2 ms
    ✓ testStaticInsertionSort      2 ms
```

```
C:\Users\vedan\.jdks\corretto-19.0.1\bin\java.exe ...
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
StatPack {hits: 19,800, normalized=42.995; copies: 0, normalized=0.000; inversions: 4,950, normalized=10.749; swaps:

Process finished with exit code 0
```

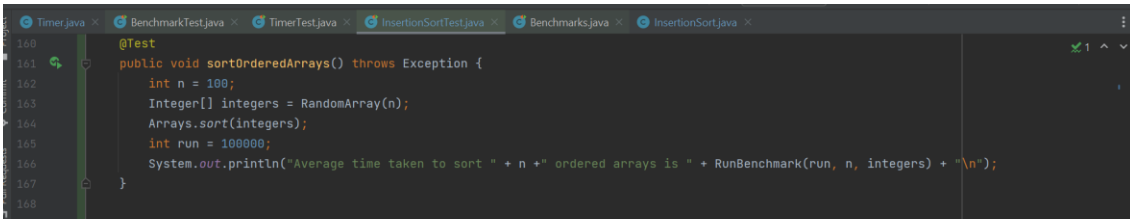
- **Part 3**

Random array:



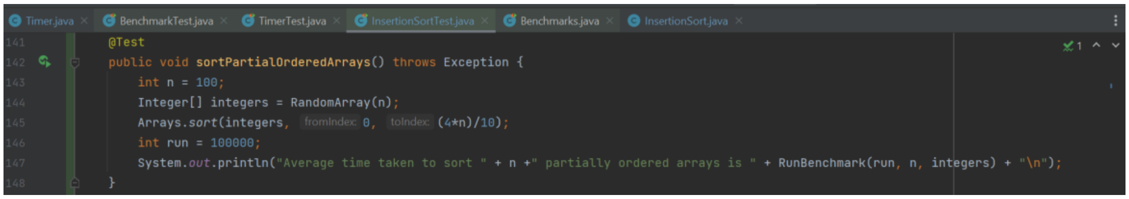
```java
@Test
public void sortRandomOrderedArrays() throws Exception {
    int n = 100;
    Integer[] integers = RandomArray(n);
    int run = 100000;
    System.out.println("Average time taken to sort " + n +" random ordered arrays is " + RunBenchmark(run, n, integers) + "\n");
}
```

Ordered array:



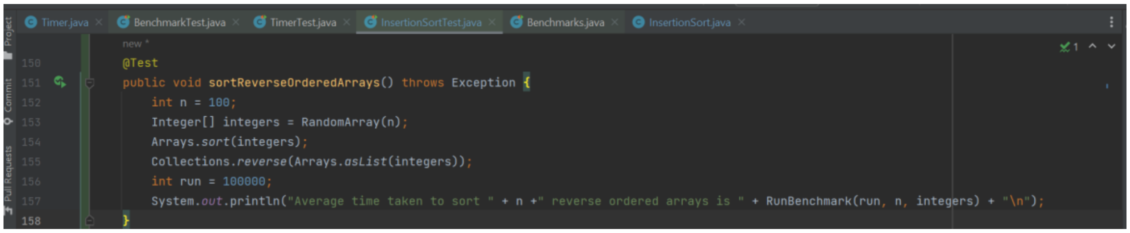
```java
@Test
public void sortOrderedArrays() throws Exception {
    int n = 100;
    Integer[] integers = RandomArray(n);
    Arrays.sort(integers);
    int run = 100000;
    System.out.println("Average time taken to sort " + n +" ordered arrays is " + RunBenchmark(run, n, integers) + "\n");
}
```

Partially ordered array:



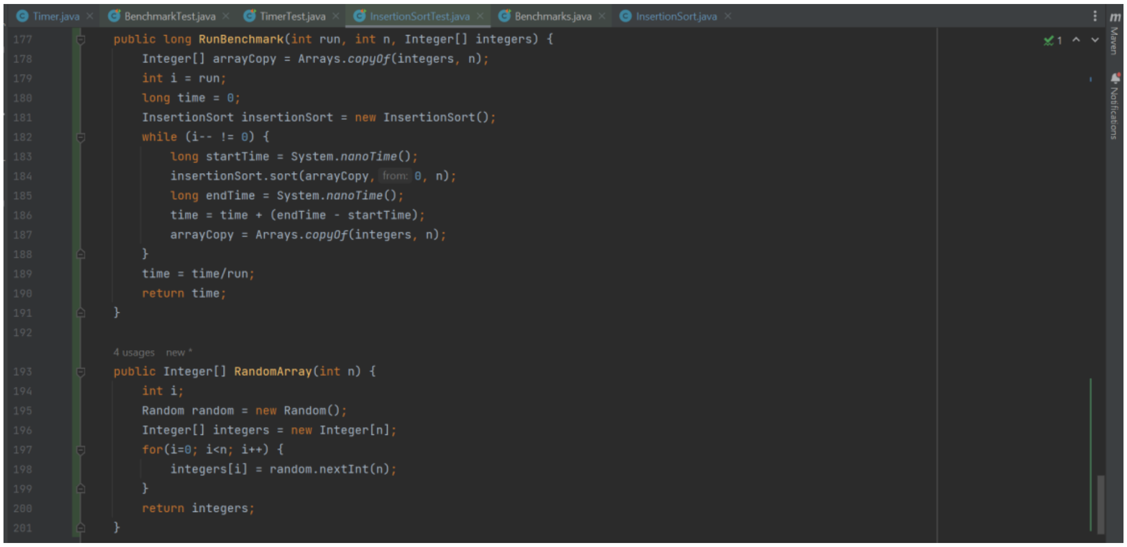
```java
@Test
public void sortPartialOrderedArrays() throws Exception {
    int n = 100;
    Integer[] integers = RandomArray(n);
    Arrays.sort(integers,  fromIndex: 0,  toIndex: (4*n)/10);
    int run = 100000;
    System.out.println("Average time taken to sort " + n +" partially ordered arrays is " + RunBenchmark(run, n, integers) + "\n");
}
```

Reverse ordered array:



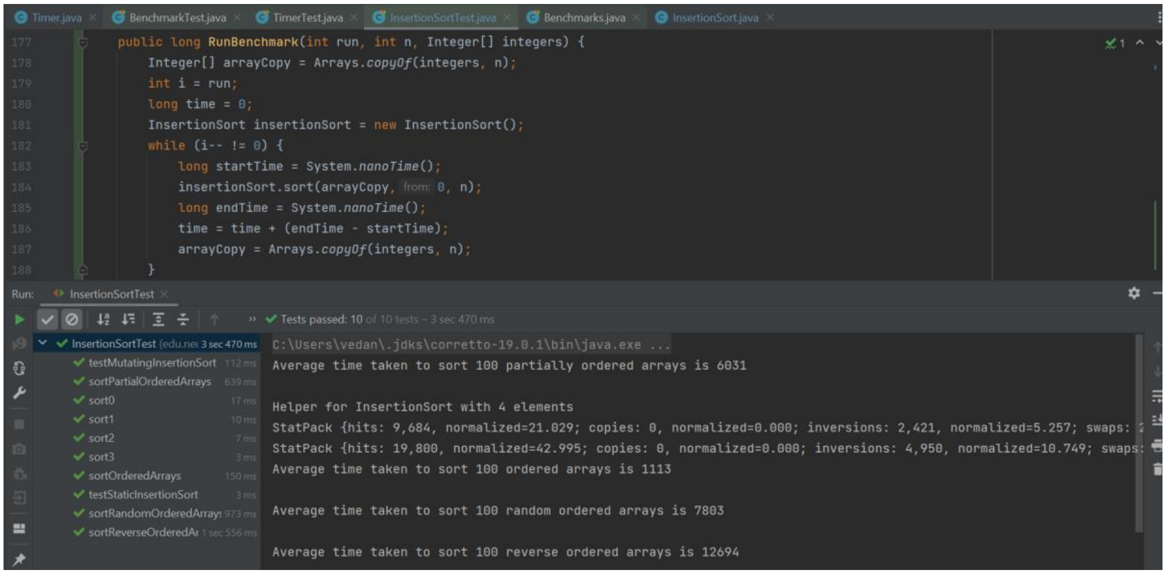
```java
@Test
public void sortReverseOrderedArrays() throws Exception {
    int n = 100;
    Integer[] integers = RandomArray(n);
    Arrays.sort(integers);
    Collections.reverse(Arrays.asList(integers));
    int run = 100000;
    System.out.println("Average time taken to sort " + n +" reverse ordered arrays is " + RunBenchmark(run, n, integers) + "\n");
}
```

Main program to run benchmarks:



```java
public long RunBenchmark(int run, int n, Integer[] integers) {
    Integer[] arrayCopy = Arrays.copyOf(integers, n);
    int i = run;
    long time = 0;
    InsertionSort insertionSort = new InsertionSort();
    while (i-- != 0) {
        long startTime = System.nanoTime();
        insertionSort.sort(arrayCopy,  from: 0, n);
        long endTime = System.nanoTime();
        time = time + (endTime - startTime);
        arrayCopy = Arrays.copyOf(integers, n);
    }
    time = time/run;
    return time;
}

public Integer[] RandomArray(int n) {
    int i;
    Random random = new Random();
    Integer[] integers = new Integer[n];
    for(i=0; i<n; i++) {
        integers[i] = random.nextInt(n);
    }
    return integers;
}
```

Running test cases:

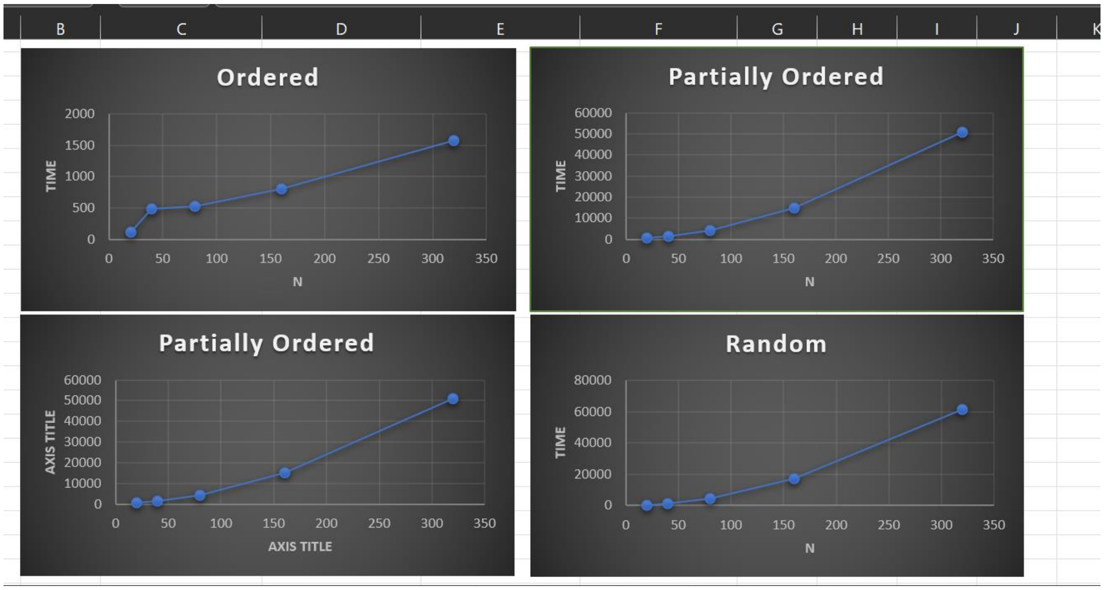


```java
    public long RunBenchmark(int run, int n, Integer[] integers) {
        Integer[] arrayCopy = Arrays.copyOf(integers, n);
        int i = run;
        long time = 0;
        InsertionSort insertionSort = new InsertionSort();
        while (i-- != 0) {
            long startTime = System.nanoTime();
            insertionSort.sort(arrayCopy, from: 0, n);
            long endTime = System.nanoTime();
            time = time + (endTime - startTime);
            arrayCopy = Arrays.copyOf(integers, n);
        }
```

```
Tests passed: 10 of 10 tests – 3 sec 470 ms

InsertionSortTest (edu.ne  3 sec 470 ms
  testMutatingInsertionSort      112 ms
  sortPartialOrderedArrays       639 ms
  sort0                           17 ms
  sort1                           10 ms
  sort2                            7 ms
  sort3                            3 ms
  sortOrderedArrays              150 ms
  testStaticInsertionSort          3 ms
  sortRandomOrderedArray:        973 ms
  sortReverseOrderedA     1 sec 556 ms
```

```
C:\Users\vedan\.jdks\corretto-19.0.1\bin\java.exe ...
Average time taken to sort 100 partially ordered arrays is 6031

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
StatPack {hits: 19,800, normalized=42.995; copies: 0, normalized=0.000; inversions: 4,950, normalized=10.749; swaps:
Average time taken to sort 100 ordered arrays is 1113

Average time taken to sort 100 random ordered arrays is 7803

Average time taken to sort 100 reverse ordered arrays is 12694
```

n	Ordered	Partially Ordered	Reverse Ordered	Random
20	115	537	511	309
40	488	1281	2073	1039
80	524	4182	8092	4572
160	812	14955	31087	17092
320	1571	50815	120846	61566



Observations:

- The Ordered array input provides the lowest running time as the size of the array (n) increases, close to a constant time graph.
- The Partial-Ordered input takes longer than the Ordered and increases exponentially as n grows, with a slightly steeper slope than $n(\log n)$.
- The Reverse-Ordered input takes the most time and increases exponentially with n at the highest rate, with a graph similar to $2n(\log n)^2$
- The Random array input takes even more time than the Partial-Ordered and increases similarly with a higher slope, closer to $n(\log n)$ but at a faster rate.