Program Structures and Algorithms
Spring 2024

NAME: RAHUL POTHIRENDI
NUID: 002889957
GITHUB LINK: https://github.com/Pothirendirahul/INFO6205.git

Task:

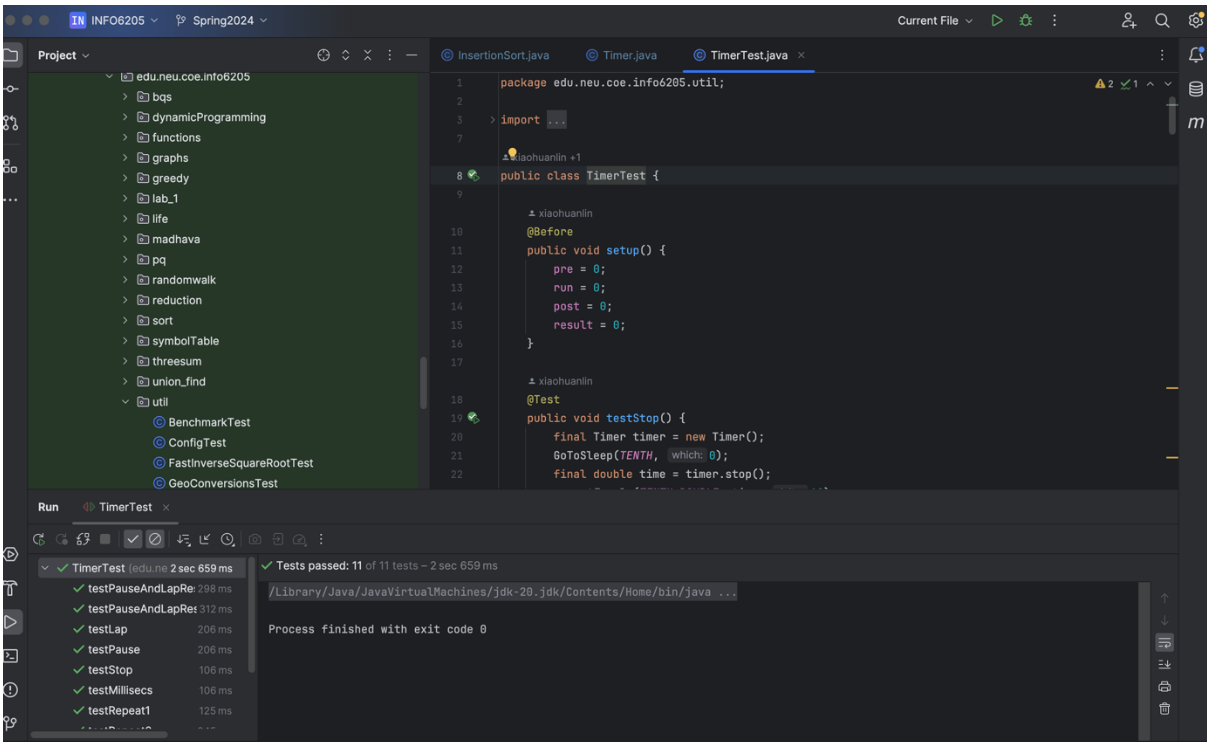
PART I
You are to implement three (3) methods (*repeat*, *getClock*, and *toMillisecs*) of a class called *Timer*. Please see the skeleton class that I created in the repository. *Timer* is invoked from a class called *Benchmark_Timer* which implements the *Benchmark* interface. The APIs of these class are as follows

PART II
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*.

PART III

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.

Unit Test Screenshots:



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

import ...

public class TimerTest {

    @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();
```

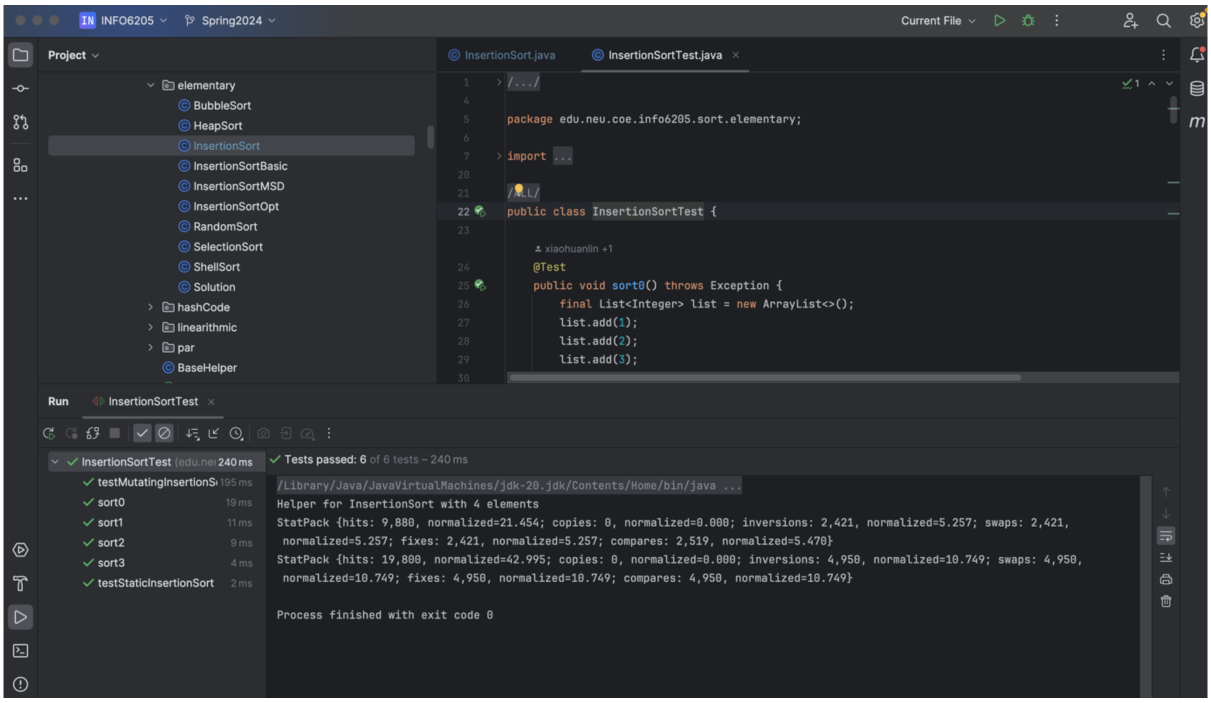
```
TimerTest (edu.ne 2 sec 659 ms
  testPauseAndLapRe  298 ms
  testPauseAndLapRe  312 ms
  testLap            206 ms
  testPause          206 ms
  testStop           106 ms
  testMillisecs      106 ms
  testRepeat1        125 ms
```

```
Tests passed: 11 of 11 tests – 2 sec 659 ms
/Library/Java/JavaVirtualMachines/jdk-20.jdk/Contents/Home/bin/java ...

Process finished with exit code 0
```

Timer Test passed

PART II



```java
/.../

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

import ...

/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);
```

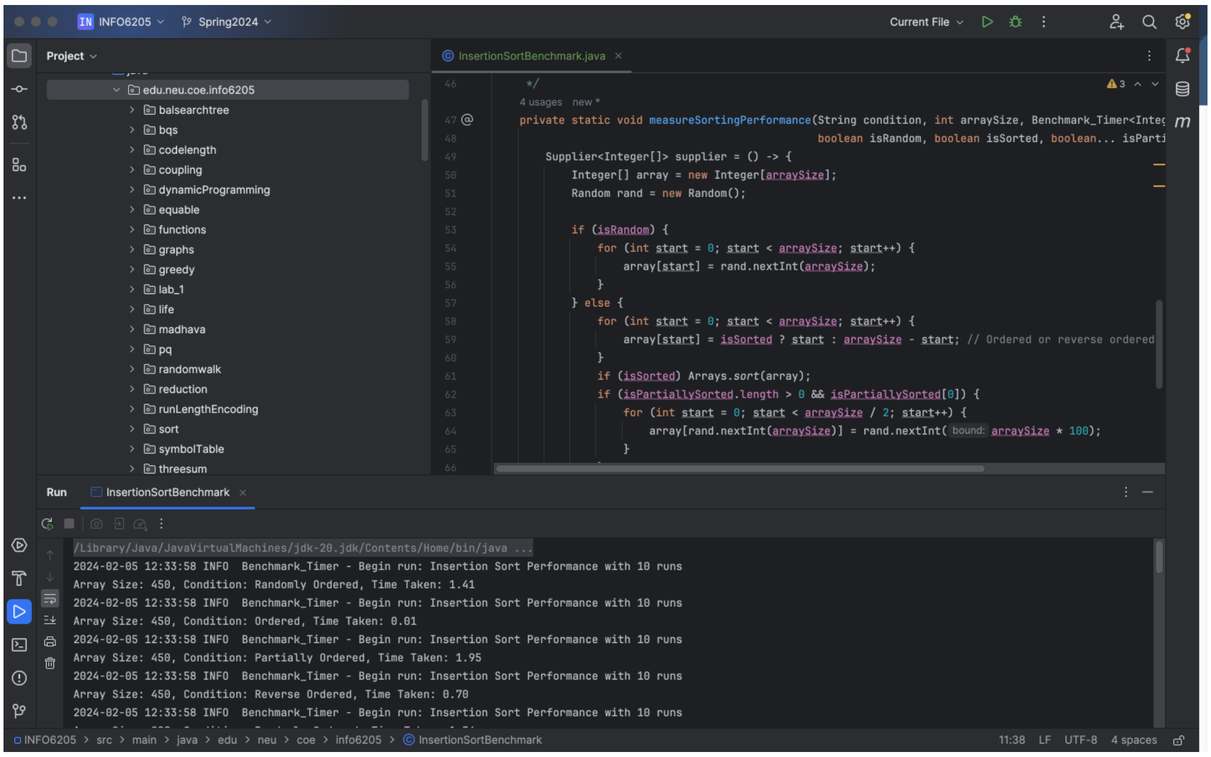
```
InsertionSortTest (edu.ne 240 ms
  testMutatingInsertionS  195 ms
  sort0                    19 ms
  sort1                    11 ms
  sort2                     9 ms
  sort3                     4 ms
  testStaticInsertionSort   2 ms
```

```
Tests passed: 6 of 6 tests – 240 ms
/Library/Java/JavaVirtualMachines/jdk-20.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,
 normalized=5.257; fixes: 2,421, normalized=5.257; compares: 2,519, normalized=5.470}
StatPack {hits: 19,800, normalized=42.995; copies: 0, normalized=0.000; inversions: 4,950, normalized=10.749; swaps: 4,950,
 normalized=10.749; fixes: 4,950, normalized=10.749; compares: 4,950, normalized=10.749}

Process finished with exit code 0
```

This part of the question is to complete insertionSort.java in order to achieve it. The **insertionSort** method takes an array of integers as input and sorts it in ascending order using the Insertion Sort algorithm.

PART III



```java
    */
private static void measureSortingPerformance(String condition, int arraySize, Benchmark_Timer<Integ
                                              boolean isRandom, boolean isSorted, boolean... isParti
    Supplier<Integer[]> supplier = () -> {
        Integer[] array = new Integer[arraySize];
        Random rand = new Random();

        if (isRandom) {
            for (int start = 0; start < arraySize; start++) {
                array[start] = rand.nextInt(arraySize);
            }
        } else {
            for (int start = 0; start < arraySize; start++) {
                array[start] = isSorted ? start : arraySize - start; // Ordered or reverse ordered
            }
            if (isSorted) Arrays.sort(array);
            if (isPartiallySorted.length > 0 && isPartiallySorted[0]) {
                for (int start = 0; start < arraySize / 2; start++) {
                    array[rand.nextInt(arraySize)] = rand.nextInt( bound: arraySize * 100);
                }
```

```
/Library/Java/JavaVirtualMachines/jdk-20.jdk/Contents/Home/bin/java ...
2024-02-05 12:33:58 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with 10 runs
Array Size: 450, Condition: Randomly Ordered, Time Taken: 1.41
2024-02-05 12:33:58 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with 10 runs
Array Size: 450, Condition: Ordered, Time Taken: 0.01
2024-02-05 12:33:58 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with 10 runs
Array Size: 450, Condition: Partially Ordered, Time Taken: 1.95
2024-02-05 12:33:58 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with 10 runs
Array Size: 450, Condition: Reverse Ordered, Time Taken: 0.70
2024-02-05 12:33:58 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with 10 runs
```

Evidence to support that conclusion:

OUTPUT CONSOLE
/Library/Java/JavaVirtualMachines/jdk-20.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=51173:/Applications/IntelliJ IDEA.app/Contents/bin -Dfile.encoding=UTF-8 -Dsun.stdout.encoding=UTF-8 -Dsun.stderr.encoding=UTF-8 -classpath /Users/rahulpothirendi/Desktop/INFO6205/target/classes:/Users/rahulpothirendi/.m2/repository/com/phasmidsoftware/args_2.13/1.0.3/args_2.13-

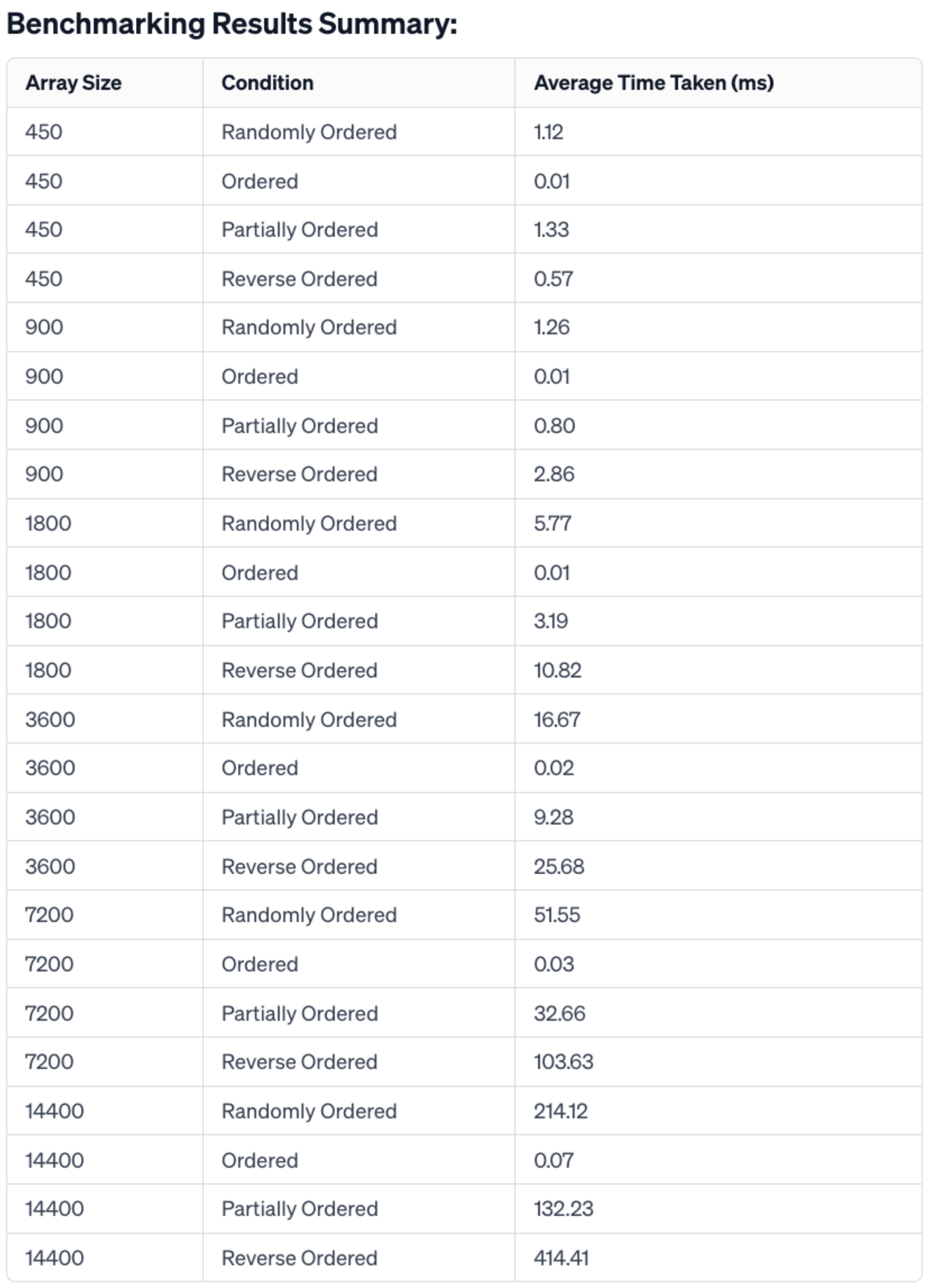
2024-02-05 12:33:58 INFO Benchmark_Timer - Begin run: Insertion Sort Performance with 10 runs
Array Size: 450, Condition: Randomly Ordered, Time Taken: 1.41
2024-02-05 12:33:58 INFO Benchmark_Timer - Begin run: Insertion Sort Performance with 10 runs
Array Size: 450, Condition: Ordered, Time Taken: 0.01
2024-02-05 12:33:58 INFO Benchmark_Timer - Begin run: Insertion Sort Performance with 10 runs
Array Size: 450, Condition: Partially Ordered, Time Taken: 1.95
2024-02-05 12:33:58 INFO Benchmark_Timer - Begin run: Insertion Sort Performance with 10 runs
Array Size: 450, Condition: Reverse Ordered, Time Taken: 0.70

```
2024-02-05 12:33:58 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with
10 runs
Array Size: 900, Condition: Randomly Ordered, Time Taken: 1.34
2024-02-05 12:33:58 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with
10 runs
Array Size: 900, Condition: Ordered, Time Taken: 0.01
2024-02-05 12:33:58 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with
10 runs
Array Size: 900, Condition: Partially Ordered, Time Taken: 1.08
2024-02-05 12:33:58 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with
10 runs
Array Size: 900, Condition: Reverse Ordered, Time Taken: 2.83
2024-02-05 12:33:58 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with
10 runs
Array Size: 1800, Condition: Randomly Ordered, Time Taken: 5.31
2024-02-05 12:33:58 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with
10 runs
Array Size: 1800, Condition: Ordered, Time Taken: 0.02
2024-02-05 12:33:58 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with
10 runs
Array Size: 1800, Condition: Partially Ordered, Time Taken: 4.26
2024-02-05 12:33:58 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with
10 runs
Array Size: 1800, Condition: Reverse Ordered, Time Taken: 13.99
2024-02-05 12:33:58 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with
10 runs
Array Size: 3600, Condition: Randomly Ordered, Time Taken: 23.13
2024-02-05 12:33:58 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with
10 runs
Array Size: 3600, Condition: Ordered, Time Taken: 0.03
2024-02-05 12:33:58 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with
10 runs
Array Size: 3600, Condition: Partially Ordered, Time Taken: 11.32
2024-02-05 12:33:59 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with
10 runs
Array Size: 3600, Condition: Reverse Ordered, Time Taken: 33.93
2024-02-05 12:33:59 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with
10 runs
Array Size: 7200, Condition: Randomly Ordered, Time Taken: 51.75
2024-02-05 12:34:00 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with
10 runs
Array Size: 7200, Condition: Ordered, Time Taken: 0.03
2024-02-05 12:34:00 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with
10 runs
Array Size: 7200, Condition: Partially Ordered, Time Taken: 32.62
2024-02-05 12:34:00 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with
10 runs
Array Size: 7200, Condition: Reverse Ordered, Time Taken: 103.29
```

```
2024-02-05 12:34:01 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with
10 runs
Array Size: 14400, Condition: Randomly Ordered, Time Taken: 212.74
2024-02-05 12:34:04 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with
10 runs
Array Size: 14400, Condition: Ordered, Time Taken: 0.07
2024-02-05 12:34:04 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with
10 runs
Array Size: 14400, Condition: Partially Ordered, Time Taken: 132.24
2024-02-05 12:34:05 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with
10 runs
Array Size: 14400, Condition: Reverse Ordered, Time Taken: 417.93

Process finished with exit code 0
```

Relationship Conclusion:



Benchmarking Results Summary:

Array Size	Condition	Average Time Taken (ms)
450	Randomly Ordered	1.12
450	Ordered	0.01
450	Partially Ordered	1.33
450	Reverse Ordered	0.57
900	Randomly Ordered	1.26
900	Ordered	0.01
900	Partially Ordered	0.80
900	Reverse Ordered	2.86
1800	Randomly Ordered	5.77
1800	Ordered	0.01
1800	Partially Ordered	3.19
1800	Reverse Ordered	10.82
3600	Randomly Ordered	16.67
3600	Ordered	0.02
3600	Partially Ordered	9.28
3600	Reverse Ordered	25.68
7200	Randomly Ordered	51.55
7200	Ordered	0.03
7200	Partially Ordered	32.66
7200	Reverse Ordered	103.63
14400	Randomly Ordered	214.12
14400	Ordered	0.07
14400	Partially Ordered	132.23
14400	Reverse Ordered	414.41

<u>Observations :</u>

We wrote a new class known as sorting benchmark where we have implemented the main method we gave a min arraySize of 450 and its max is upto 20000. The sizeArray is multiplied by 2 after each run

- The algorithm performs well on ordered and reverse-ordered arrays, with relatively low average times.

- Randomly ordered arrays show varying performance, with increasing times as the array size grows.

- As the array size doubles, the time taken by the algorithm increases significantly, indicating a potential O(n^2) time complexity.