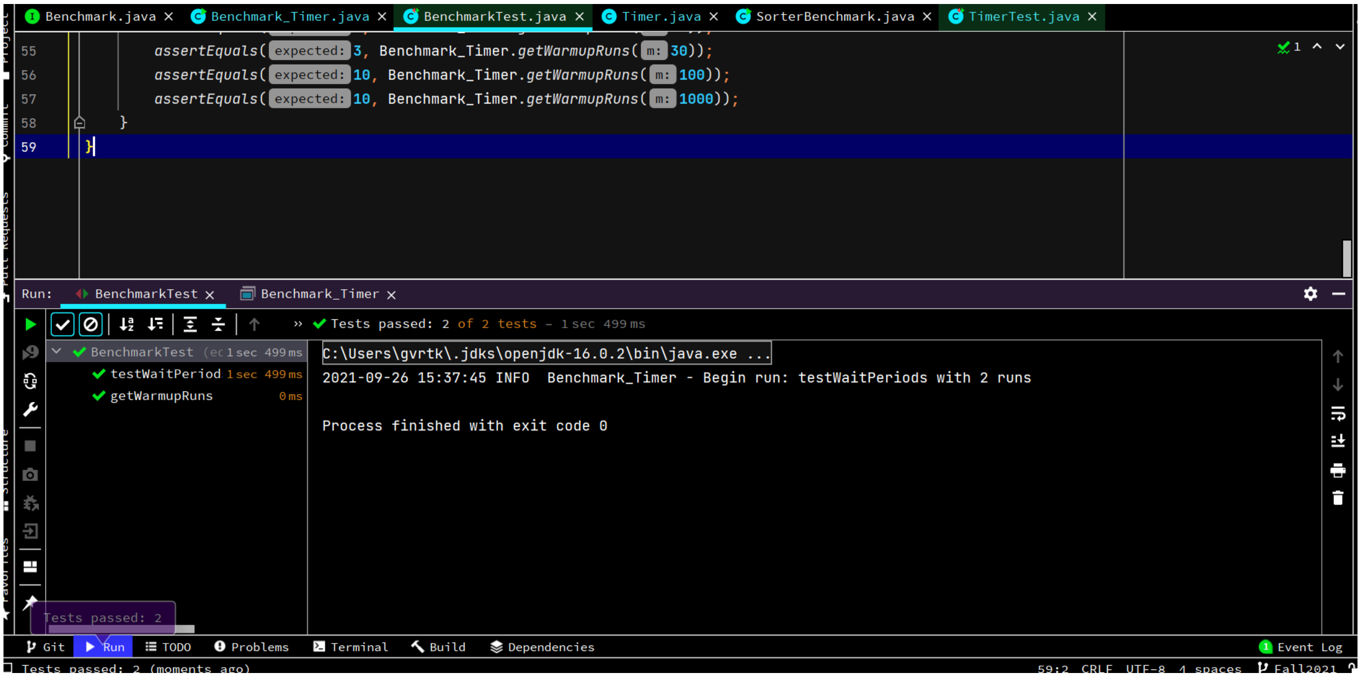
ASSIGNMENT 2 : BENCHMARK

Task 1: You are to implement three methods 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.

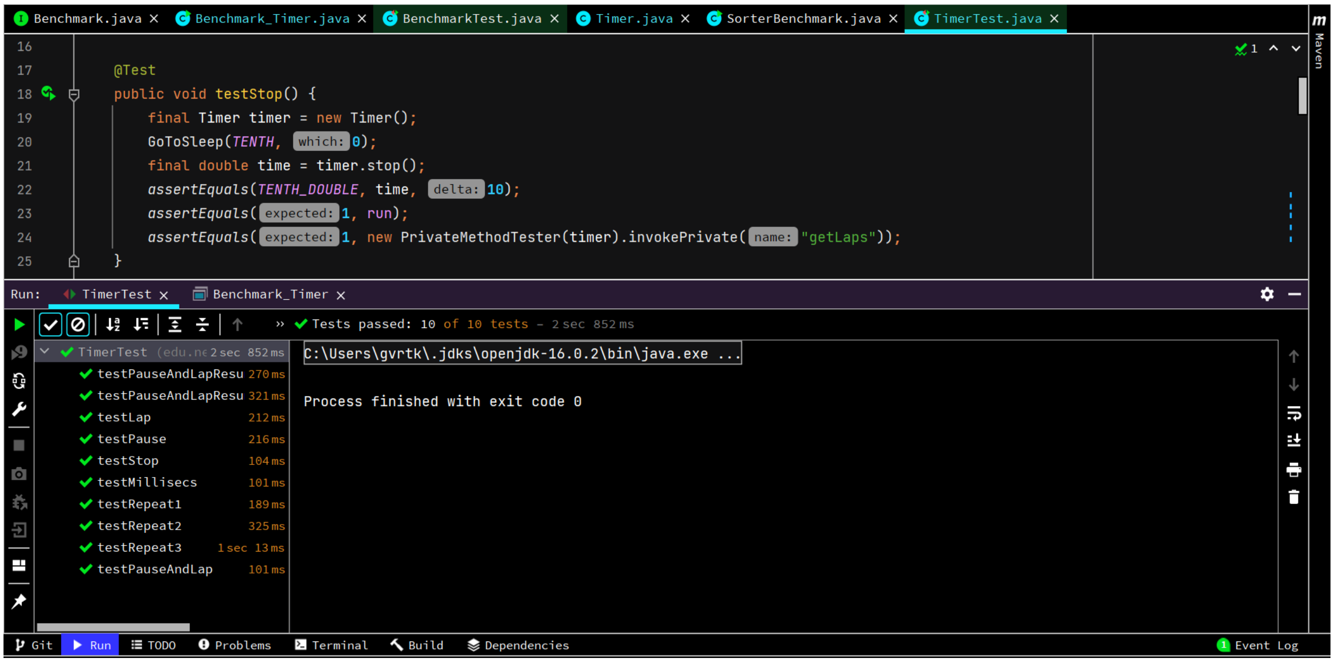
Benchmark_Test:



```
            assertEquals( expected: 3, Benchmark_Timer.getWarmupRuns( m: 30));
            assertEquals( expected: 10, Benchmark_Timer.getWarmupRuns( m: 100));
            assertEquals( expected: 10, Benchmark_Timer.getWarmupRuns( m: 1000));
        }
    }
```

```
Tests passed: 2 of 2 tests – 1 sec 499 ms

BenchmarkTest (ec 1 sec 499 ms
  testWaitPeriod 1 sec 499 ms
  getWarmupRuns          0 ms

C:\Users\gvrtk\.jdks\openjdk-16.0.2\bin\java.exe ...
2021-09-26 15:37:45 INFO  Benchmark_Timer - Begin run: testWaitPeriods with 2 runs

Process finished with exit code 0

Tests passed: 2
Tests passed: 2 (moments ago)                    59:2  CRLF  UTF-8  4 spaces  Fall2021
```

Timer_Test:

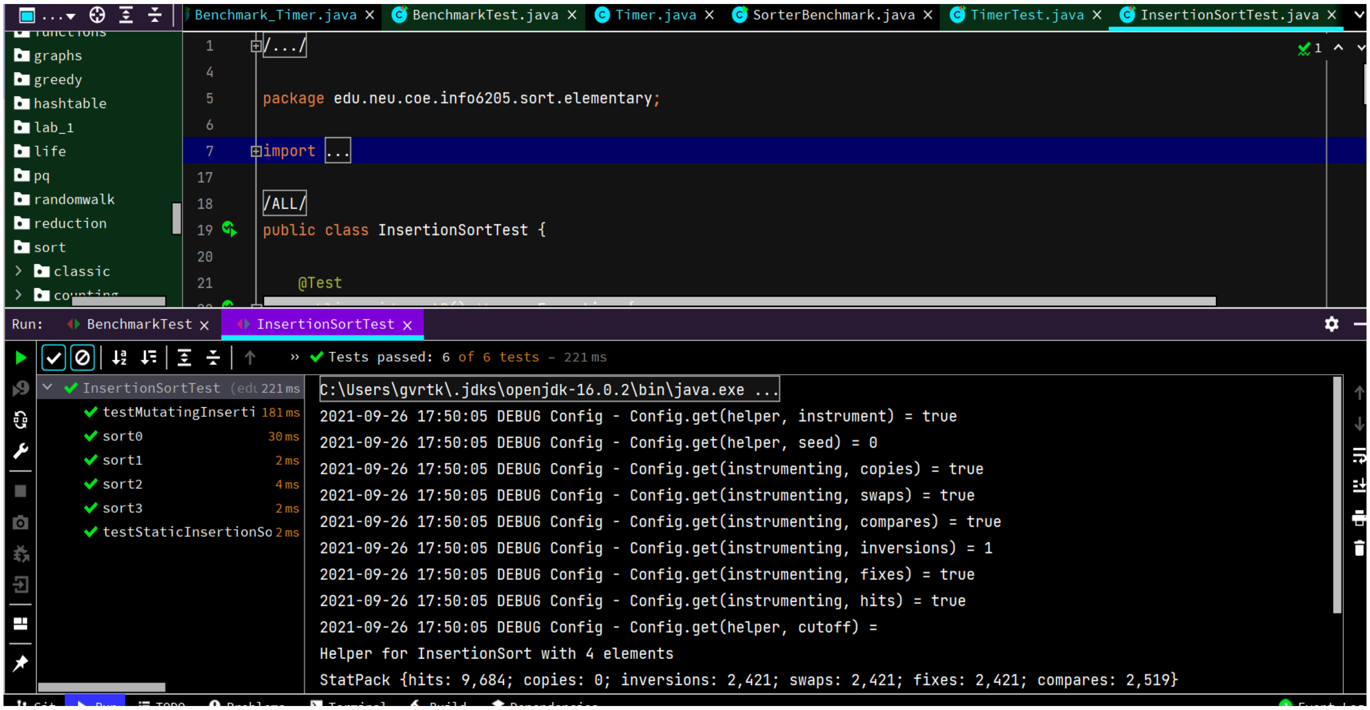


```
    @Test
    public void testStop() {
        final Timer timer = new Timer();
        GoToSleep(TENTH,  which: 0);
        final double time = timer.stop();
        assertEquals(TENTH_DOUBLE, time,  delta: 10);
        assertEquals( expected: 1, run);
        assertEquals( expected: 1, new PrivateMethodTester(timer).invokePrivate(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

```
Tests passed: 10 of 10 tests – 2 sec 852 ms

TimerTest (edu.ne 2 sec 852 ms
  testPauseAndLapResu 270 ms
  testPauseAndLapResu 321 ms
  testLap             212 ms
  testPause           216 ms
  testStop            104 ms
  testMillisecs       101 ms
  testRepeat1         189 ms
  testRepeat2         325 ms
  testRepeat3   1 sec  13 ms
  testPauseAndLap     101 ms

C:\Users\gvrtk\.jdks\openjdk-16.0.2\bin\java.exe ...

Process finished with exit code 0
```

Task 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*.

InsertionSort:



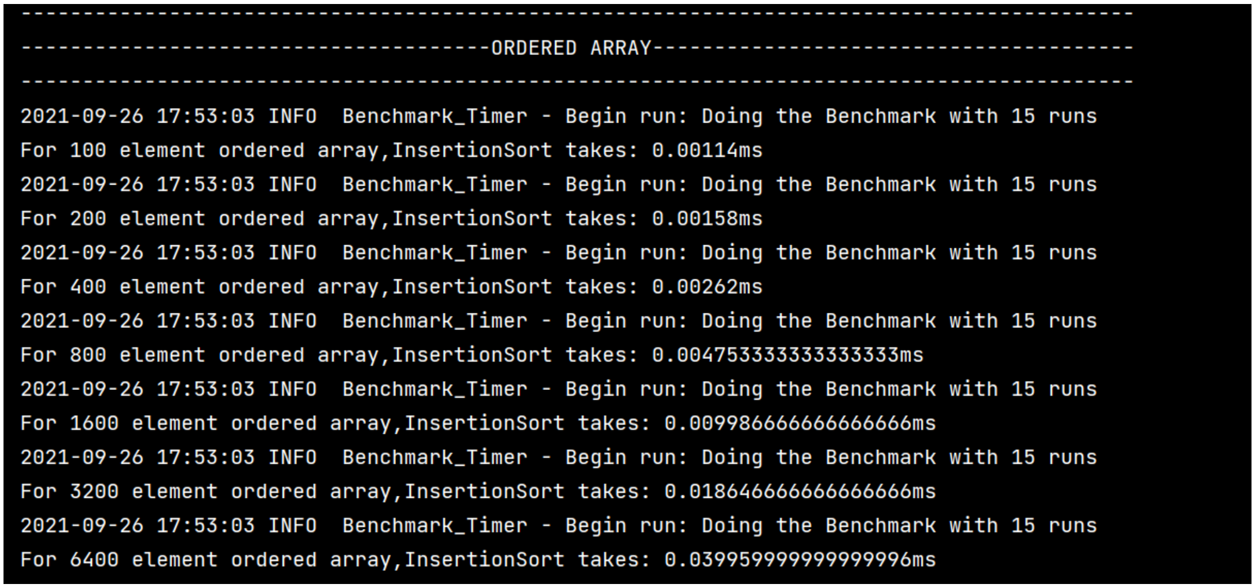
```
/.../

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

import ...

/ALL/
public class InsertionSortTest {

    @Test
```

✓ Tests passed: 6 of 6 tests – 221 ms

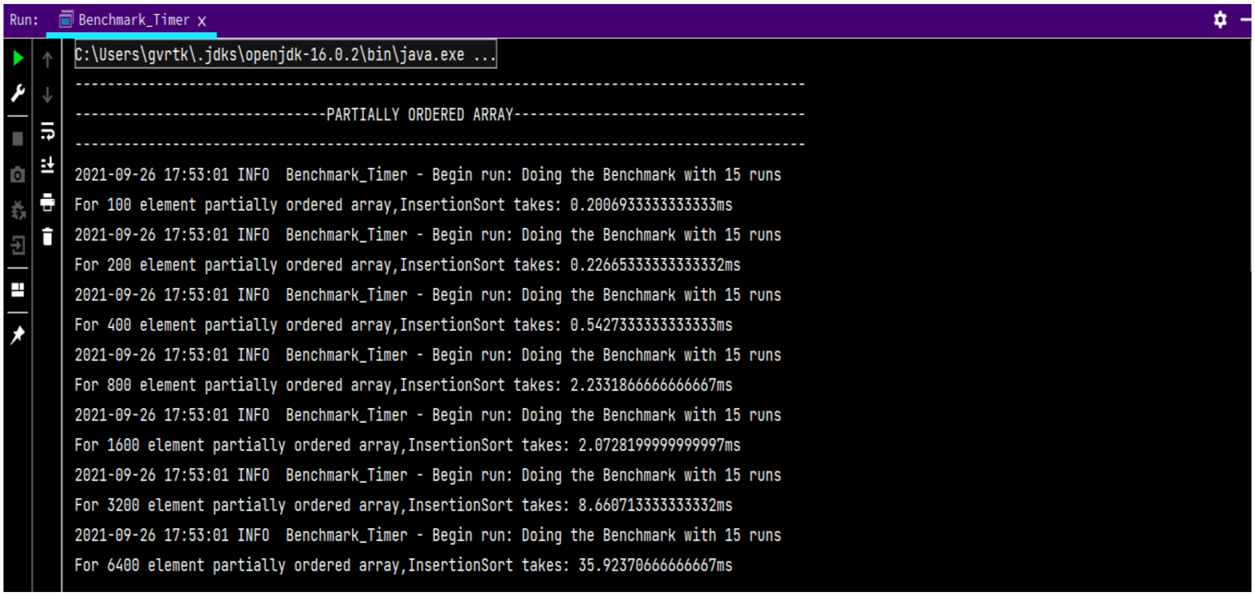
✓ InsertionSortTest (edu	221 ms
✓ testMutatingInserti	181 ms
✓ sort0	30 ms
✓ sort1	2 ms
✓ sort2	4 ms
✓ sort3	2 ms
✓ testStaticInsertionSo	2 ms

```
C:\Users\gvrtk\.jdks\openjdk-16.0.2\bin\java.exe ...
2021-09-26 17:50:05 DEBUG Config - Config.get(helper, instrument) = true
2021-09-26 17:50:05 DEBUG Config - Config.get(helper, seed) = 0
2021-09-26 17:50:05 DEBUG Config - Config.get(instrumenting, copies) = true
2021-09-26 17:50:05 DEBUG Config - Config.get(instrumenting, swaps) = true
2021-09-26 17:50:05 DEBUG Config - Config.get(instrumenting, compares) = true
2021-09-26 17:50:05 DEBUG Config - Config.get(instrumenting, inversions) = 1
2021-09-26 17:50:05 DEBUG Config - Config.get(instrumenting, fixes) = true
2021-09-26 17:50:05 DEBUG Config - Config.get(instrumenting, hits) = true
2021-09-26 17:50:05 DEBUG Config - Config.get(helper, cutoff) =
Helper for InsertionSort with 4 elements
StatPack {hits: 9,684; copies: 0; inversions: 2,421; swaps: 2,421; fixes: 2,421; compares: 2,519}
```

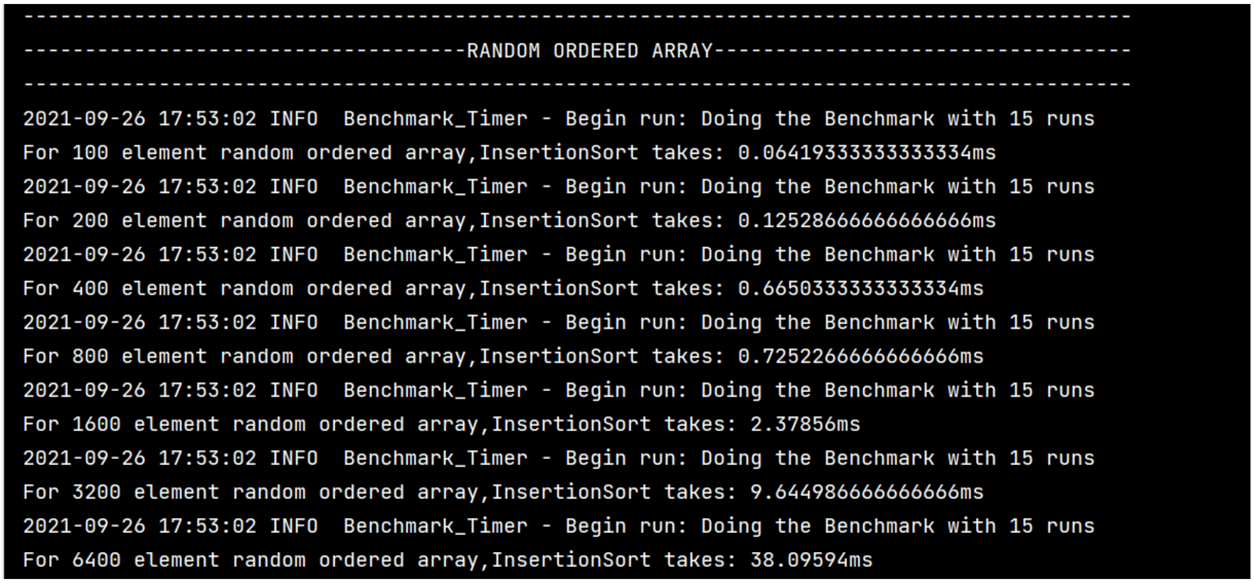
Task 3 : Implement a main program (or you could do it via your own unit tests) to actually run the following benchmarks: measure the running times of this sort, using four different initial array ordering situations: random, ordered, partially-ordered and reverse-ordered. I suggest that your arrays to 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



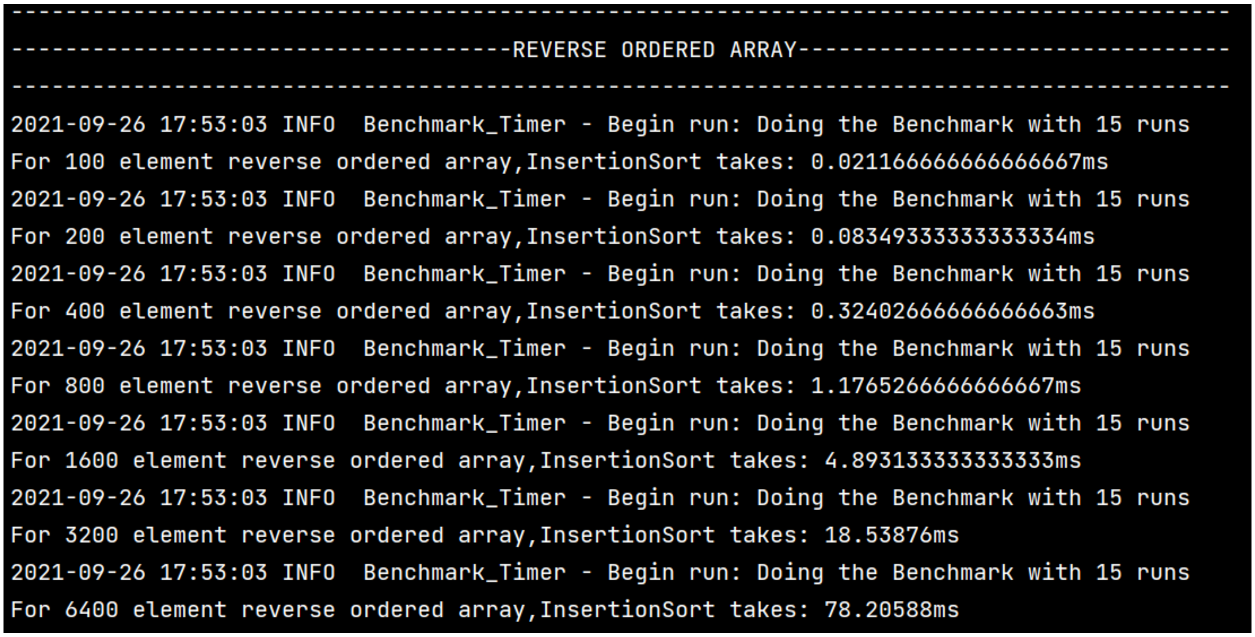
```
-----------------------------------------------------------------------------------------
-------------------------------------ORDERED ARRAY---------------------------------------
-----------------------------------------------------------------------------------------
2021-09-26 17:53:03 INFO  Benchmark_Timer - Begin run: Doing the Benchmark with 15 runs
For 100 element ordered array,InsertionSort takes: 0.00114ms
2021-09-26 17:53:03 INFO  Benchmark_Timer - Begin run: Doing the Benchmark with 15 runs
For 200 element ordered array,InsertionSort takes: 0.00158ms
2021-09-26 17:53:03 INFO  Benchmark_Timer - Begin run: Doing the Benchmark with 15 runs
For 400 element ordered array,InsertionSort takes: 0.00262ms
2021-09-26 17:53:03 INFO  Benchmark_Timer - Begin run: Doing the Benchmark with 15 runs
For 800 element ordered array,InsertionSort takes: 0.004753333333333333ms
2021-09-26 17:53:03 INFO  Benchmark_Timer - Begin run: Doing the Benchmark with 15 runs
For 1600 element ordered array,InsertionSort takes: 0.009986666666666666ms
2021-09-26 17:53:03 INFO  Benchmark_Timer - Begin run: Doing the Benchmark with 15 runs
For 3200 element ordered array,InsertionSort takes: 0.018646666666666666ms
2021-09-26 17:53:03 INFO  Benchmark_Timer - Begin run: Doing the Benchmark with 15 runs
For 6400 element ordered array,InsertionSort takes: 0.039959999999999996ms
```



```
C:\Users\gvrtk\.jdks\openjdk-16.0.2\bin\java.exe ...
------------------------------------------------------------------------------------------
-----------------------------PARTIALLY ORDERED ARRAY--------------------------------------
------------------------------------------------------------------------------------------
2021-09-26 17:53:01 INFO  Benchmark_Timer - Begin run: Doing the Benchmark with 15 runs
For 100 element partially ordered array,InsertionSort takes: 0.2006933333333333ms
2021-09-26 17:53:01 INFO  Benchmark_Timer - Begin run: Doing the Benchmark with 15 runs
For 200 element partially ordered array,InsertionSort takes: 0.22665333333333332ms
2021-09-26 17:53:01 INFO  Benchmark_Timer - Begin run: Doing the Benchmark with 15 runs
For 400 element partially ordered array,InsertionSort takes: 0.5427333333333333ms
2021-09-26 17:53:01 INFO  Benchmark_Timer - Begin run: Doing the Benchmark with 15 runs
For 800 element partially ordered array,InsertionSort takes: 2.2331866666666667ms
2021-09-26 17:53:01 INFO  Benchmark_Timer - Begin run: Doing the Benchmark with 15 runs
For 1600 element partially ordered array,InsertionSort takes: 2.0728199999999997ms
2021-09-26 17:53:01 INFO  Benchmark_Timer - Begin run: Doing the Benchmark with 15 runs
For 3200 element partially ordered array,InsertionSort takes: 8.660713333333332ms
2021-09-26 17:53:01 INFO  Benchmark_Timer - Begin run: Doing the Benchmark with 15 runs
For 6400 element partially ordered array,InsertionSort takes: 35.92370666666667ms
```

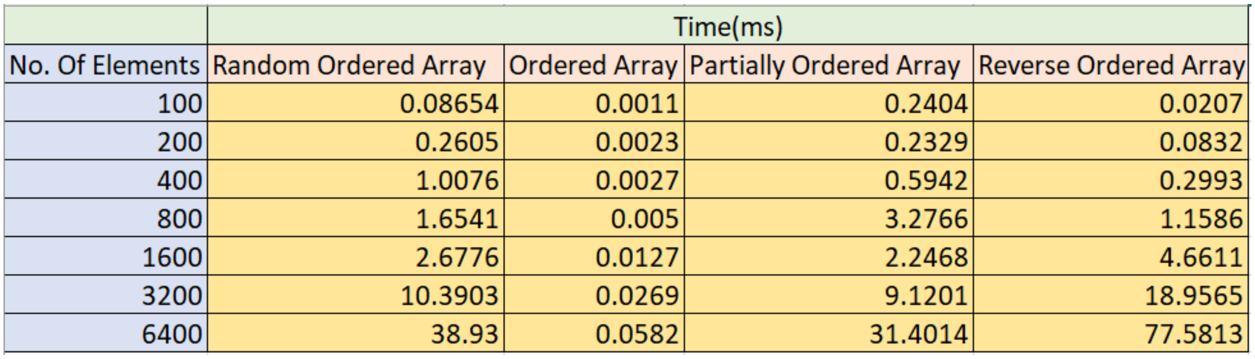


```
------------------------------------------------------------------------------------------
------------------------------------RANDOM ORDERED ARRAY----------------------------------
------------------------------------------------------------------------------------------
2021-09-26 17:53:02 INFO  Benchmark_Timer - Begin run: Doing the Benchmark with 15 runs
For 100 element random ordered array,InsertionSort takes: 0.06419333333333334ms
2021-09-26 17:53:02 INFO  Benchmark_Timer - Begin run: Doing the Benchmark with 15 runs
For 200 element random ordered array,InsertionSort takes: 0.12528666666666666ms
2021-09-26 17:53:02 INFO  Benchmark_Timer - Begin run: Doing the Benchmark with 15 runs
For 400 element random ordered array,InsertionSort takes: 0.6650333333333334ms
2021-09-26 17:53:02 INFO  Benchmark_Timer - Begin run: Doing the Benchmark with 15 runs
For 800 element random ordered array,InsertionSort takes: 0.7252266666666666ms
2021-09-26 17:53:02 INFO  Benchmark_Timer - Begin run: Doing the Benchmark with 15 runs
For 1600 element random ordered array,InsertionSort takes: 2.37856ms
2021-09-26 17:53:02 INFO  Benchmark_Timer - Begin run: Doing the Benchmark with 15 runs
For 3200 element random ordered array,InsertionSort takes: 9.644986666666666ms
2021-09-26 17:53:02 INFO  Benchmark_Timer - Begin run: Doing the Benchmark with 15 runs
For 6400 element random ordered array,InsertionSort takes: 38.09594ms
```

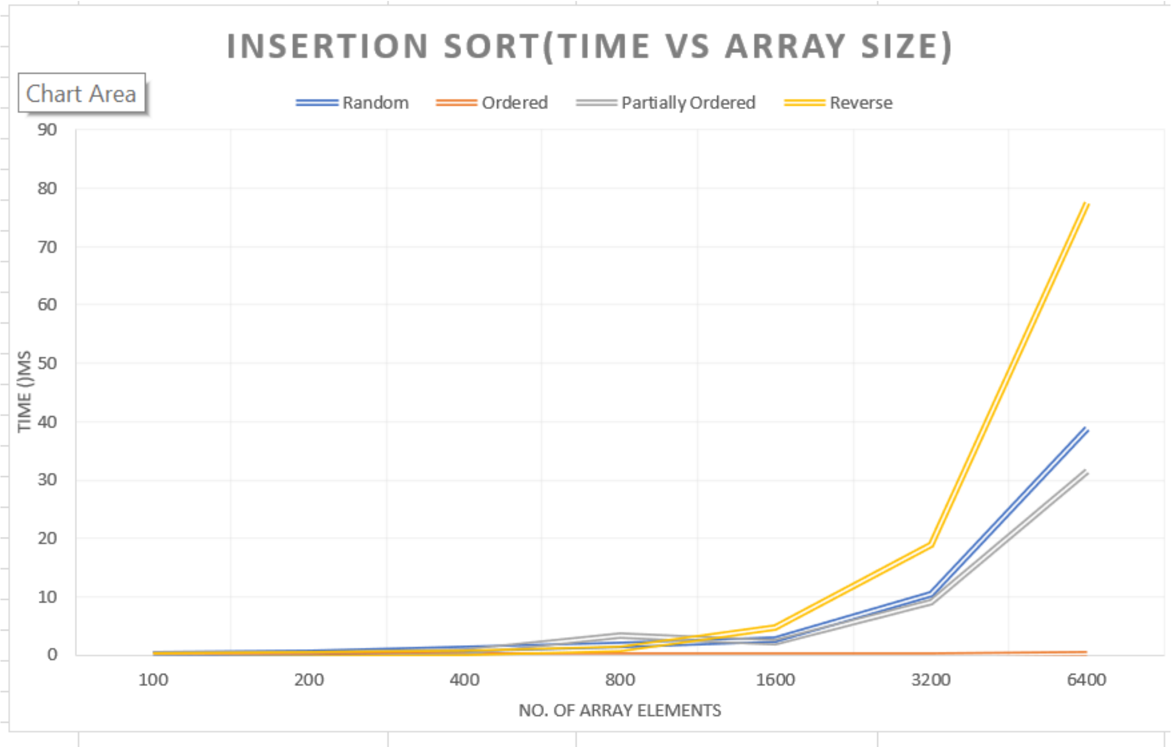


```
------------------------------------------------------------------------------------------
-------------------------------------REVERSE ORDERED ARRAY--------------------------------
------------------------------------------------------------------------------------------
2021-09-26 17:53:03 INFO  Benchmark_Timer - Begin run: Doing the Benchmark with 15 runs
For 100 element reverse ordered array,InsertionSort takes: 0.021166666666666667ms
2021-09-26 17:53:03 INFO  Benchmark_Timer - Begin run: Doing the Benchmark with 15 runs
For 200 element reverse ordered array,InsertionSort takes: 0.08349333333333334ms
2021-09-26 17:53:03 INFO  Benchmark_Timer - Begin run: Doing the Benchmark with 15 runs
For 400 element reverse ordered array,InsertionSort takes: 0.32402666666666663ms
2021-09-26 17:53:03 INFO  Benchmark_Timer - Begin run: Doing the Benchmark with 15 runs
For 800 element reverse ordered array,InsertionSort takes: 1.1765266666666667ms
2021-09-26 17:53:03 INFO  Benchmark_Timer - Begin run: Doing the Benchmark with 15 runs
For 1600 element reverse ordered array,InsertionSort takes: 4.893133333333333ms
2021-09-26 17:53:03 INFO  Benchmark_Timer - Begin run: Doing the Benchmark with 15 runs
For 3200 element reverse ordered array,InsertionSort takes: 18.53876ms
2021-09-26 17:53:03 INFO  Benchmark_Timer - Begin run: Doing the Benchmark with 15 runs
For 6400 element reverse ordered array,InsertionSort takes: 78.20588ms
```

INSERTION SORT DATA & ANALYSIS:



	Time(ms)			
No. Of Elements	Random Ordered Array	Ordered Array	Partially Ordered Array	Reverse Ordered Array
100	0.08654	0.0011	0.2404	0.0207
200	0.2605	0.0023	0.2329	0.0832
400	1.0076	0.0027	0.5942	0.2993
800	1.6541	0.005	3.2766	1.1586
1600	2.6776	0.0127	2.2468	4.6611
3200	10.3903	0.0269	9.1201	18.9565
6400	38.93	0.0582	31.4014	77.5813



CONCLUSION:

After implementing insertion sort on 5 different ordered arrays and observing the time taken to implement it, I have come to the following conclusion:

Arrays in the order of time taken to sort them:

Ordered Array < Partially Ordered Array < Random Array < Reverse Array

Reverse Ordered Array has a steep increase in time taken to sort as the array size is doubled .