Program Structures and Algorithms
Spring 2024

NAME: Aneesh Arunjunai Saravanan
NUID: 002675639
GITHUB LINK: https://github.com/aneesharunjunai/INFO6205

Task: Assignment 3 – Benchmark

(Part 1) 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.

(Part 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.

(Part 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.

Observation & Conclusion:

The insertion sort test shows how well it works in different situations. When things are perfect, like with already sorted arrays, it's super quick—just O(n) time. It's cool because it checks each part without needing to swap much, making it fast.

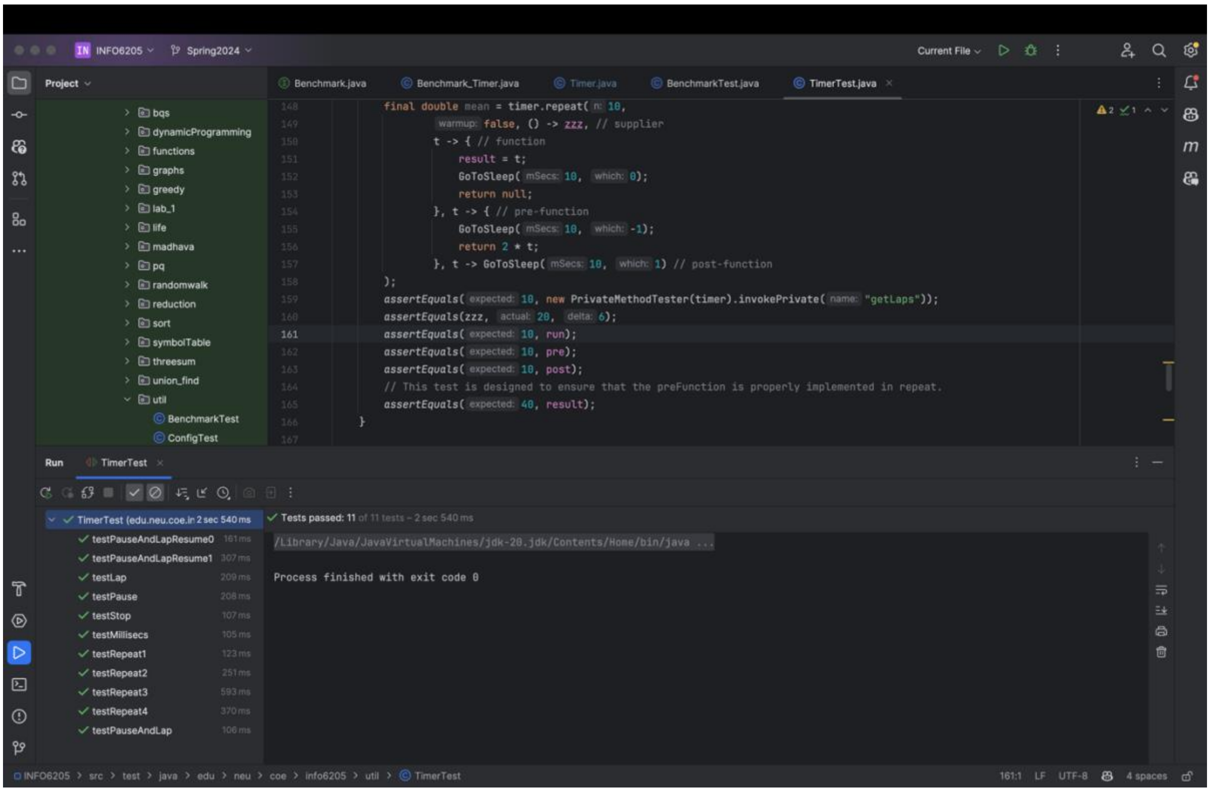
But, in normal cases where arrays are a bit messy, insertion sort can slow down. For partially or randomly sorted arrays, it's often more like O(n^2). That's because each part might need lots of checks and maybe some swapping before everything finds its right place.

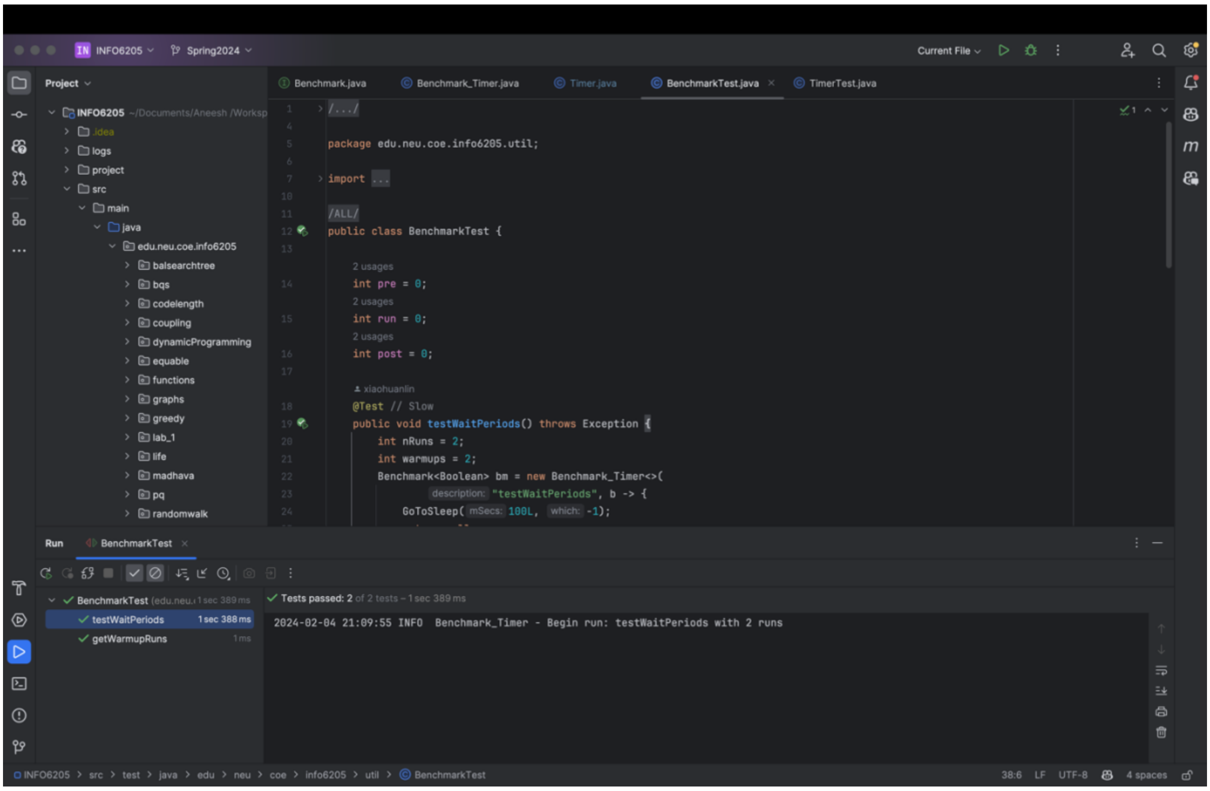
Now, when things are the messiest, like when arrays are sorted backward, insertion sort struggles with time complexity of O(n^2). It's like a puzzle where the last piece needs a bunch of checks and swaps to fit in. Overall, it ends up doing n(n-1)/2 operations.

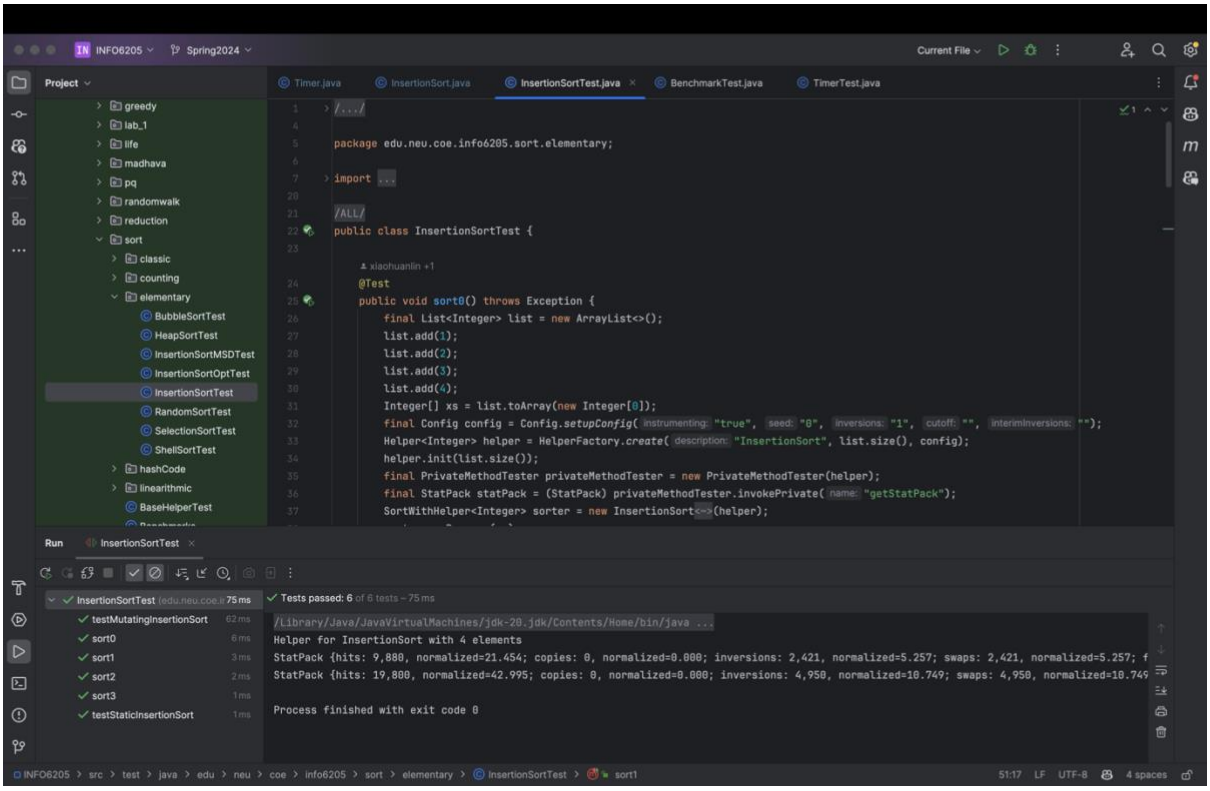
So, if we rank how good insertion sort is in different situations, it's like this: Sorted < Partially Sorted < Randomly Sorted < Reverse Sorted. This shows that insertion sort is best for almost sorted or small sets of data. The starting order of the data really affects how well it works.

In essence, the performance of insertion sort is deeply intertwined with the initial state of the array. The efficiency shines when dealing with nearly sorted or compact datasets. Its simplicity and effectiveness in scenarios where data is already in order make it a solid choice for certain applications. However, it's crucial to be mindful of its limitations, especially in scenarios where the array is more disorganized. Being aware of these nuances helps in making informed decisions about when to leverage insertion sort for optimal results.

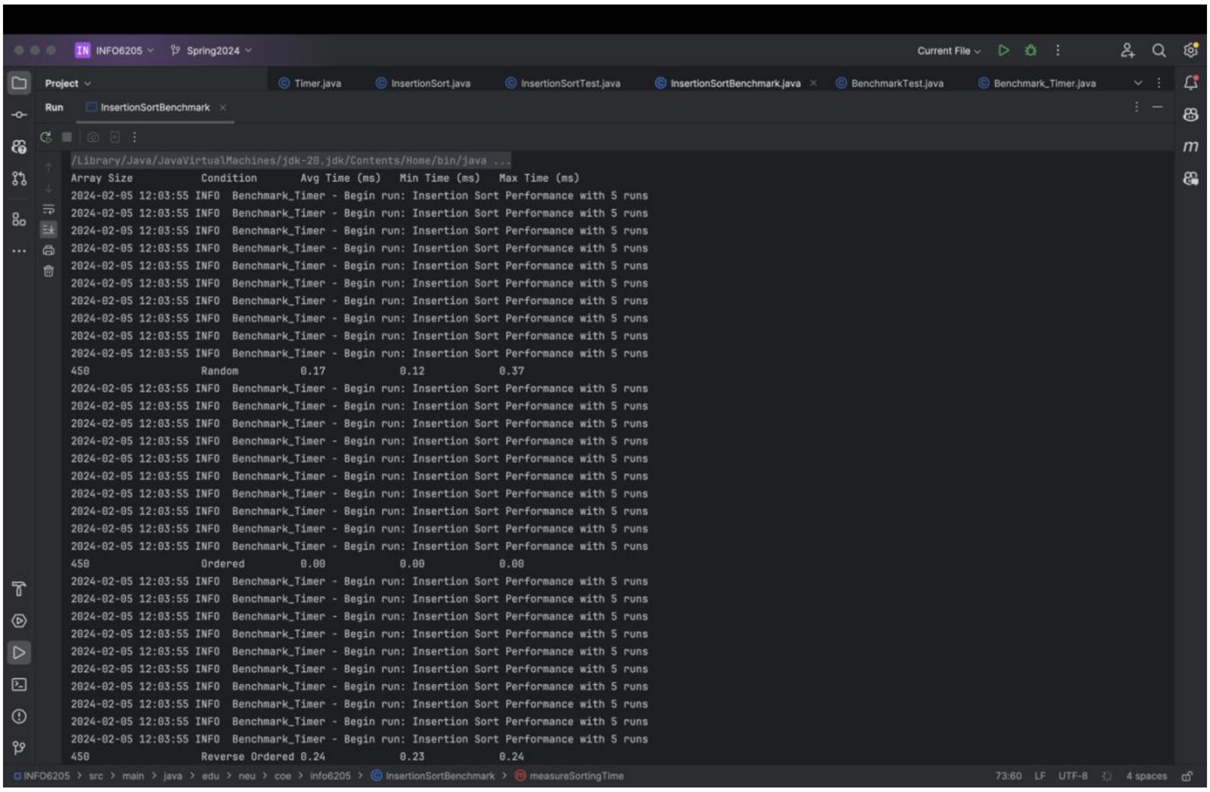
Unit Test Screenshots:



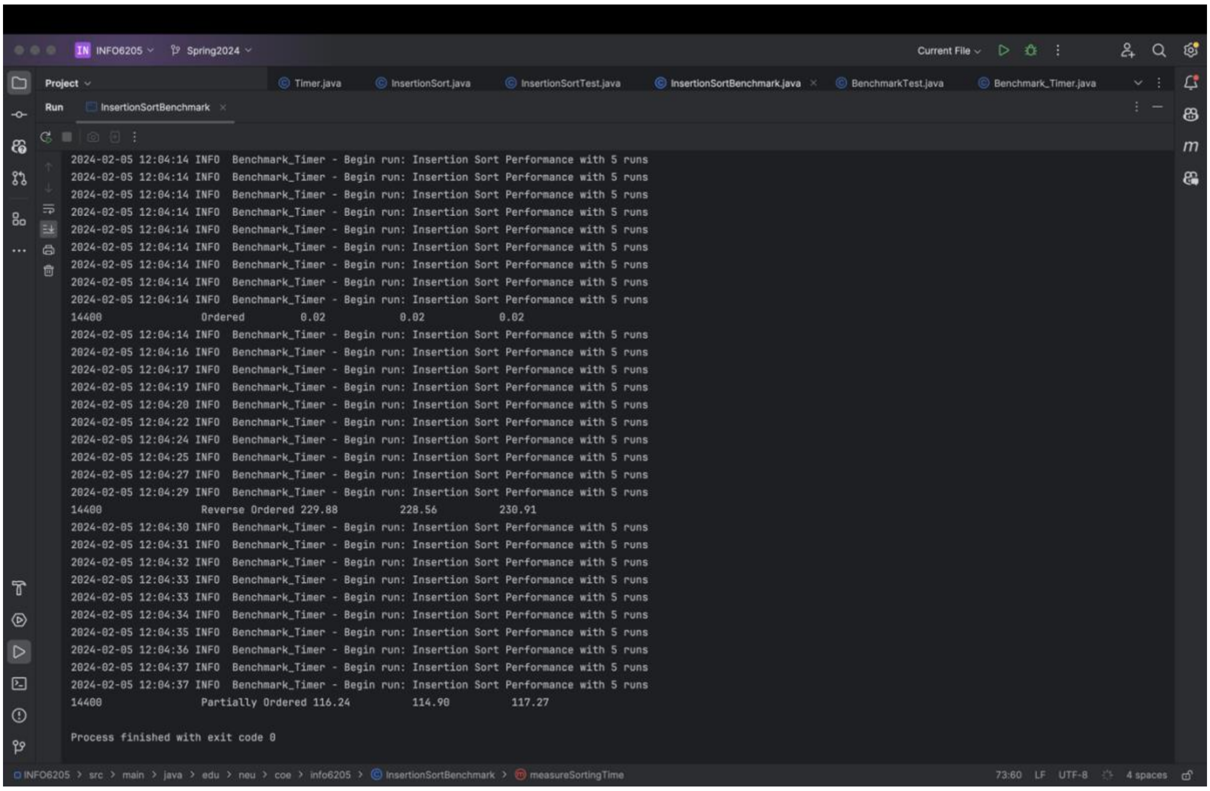



INFO6205
Spring2024
Current File
Project
greedy
lab_1
life
madhava
pq
randomwalk
reduction
sort
classic
counting
elementary
BubbleSortTest
HeapSortTest
InsertionSortMSDTest
InsertionSortOptTest
InsertionSortTest
RandomSortTest
SelectionSortTest
ShellSortTest
hashCode
linearithmic
BaseHelperTest
Timer.java
InsertionSort.java
InsertionSortTest.java
BenchmarkTest.java
TimerTest.java
/.../
package edu.neu.coe.info6205.sort.elementary;
import ...
/ALL/
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<>(helper);
Run
InsertionSortTest
InsertionSortTest 75 ms
testMutatingInsertionSort 62 ms
sort0 6 ms
sort1 3 ms
sort2 2 ms
sort3 1 ms
testStaticInsertionSort 1 ms
Tests passed: 6 of 6 tests – 75 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; f
StatPack {hits: 19,800, normalized=42.995; copies: 0, normalized=0.000; inversions: 4,950, normalized=10.749; swaps: 4,950, normalized=10.749
Process finished with exit code 0
INFO6205 > src > test > java > edu > neu > coe > info6205 > sort > elementary > InsertionSortTest > sort1
51:17 LF UTF-8 4 spaces

Console Output:



```
/Library/Java/JavaVirtualMachines/jdk-20.jdk/Contents/Home/bin/java ...
Array Size          Condition       Avg Time (ms)   Min Time (ms)   Max Time (ms)
2024-02-05 12:03:55 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with 5 runs
2024-02-05 12:03:55 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with 5 runs
2024-02-05 12:03:55 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with 5 runs
2024-02-05 12:03:55 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with 5 runs
2024-02-05 12:03:55 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with 5 runs
2024-02-05 12:03:55 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with 5 runs
2024-02-05 12:03:55 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with 5 runs
2024-02-05 12:03:55 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with 5 runs
2024-02-05 12:03:55 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with 5 runs
2024-02-05 12:03:55 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with 5 runs
450                 Random          0.17            0.12            0.37
2024-02-05 12:03:55 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with 5 runs
2024-02-05 12:03:55 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with 5 runs
2024-02-05 12:03:55 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with 5 runs
2024-02-05 12:03:55 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with 5 runs
2024-02-05 12:03:55 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with 5 runs
2024-02-05 12:03:55 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with 5 runs
2024-02-05 12:03:55 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with 5 runs
2024-02-05 12:03:55 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with 5 runs
2024-02-05 12:03:55 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with 5 runs
2024-02-05 12:03:55 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with 5 runs
450                 Ordered         0.00            0.00            0.00
2024-02-05 12:03:55 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with 5 runs
2024-02-05 12:03:55 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with 5 runs
2024-02-05 12:03:55 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with 5 runs
2024-02-05 12:03:55 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with 5 runs
2024-02-05 12:03:55 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with 5 runs
2024-02-05 12:03:55 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with 5 runs
2024-02-05 12:03:55 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with 5 runs
2024-02-05 12:03:55 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with 5 runs
2024-02-05 12:03:55 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with 5 runs
2024-02-05 12:03:55 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with 5 runs
450                 Reverse Ordered 0.24            0.23            0.24
```



```
2024-02-05 12:04:14 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with 5 runs
2024-02-05 12:04:14 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with 5 runs
2024-02-05 12:04:14 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with 5 runs
2024-02-05 12:04:14 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with 5 runs
2024-02-05 12:04:14 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with 5 runs
2024-02-05 12:04:14 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with 5 runs
2024-02-05 12:04:14 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with 5 runs
2024-02-05 12:04:14 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with 5 runs
2024-02-05 12:04:14 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with 5 runs
14400               Ordered         0.02            0.02            0.02
2024-02-05 12:04:14 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with 5 runs
2024-02-05 12:04:16 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with 5 runs
2024-02-05 12:04:17 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with 5 runs
2024-02-05 12:04:19 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with 5 runs
2024-02-05 12:04:20 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with 5 runs
2024-02-05 12:04:22 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with 5 runs
2024-02-05 12:04:24 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with 5 runs
2024-02-05 12:04:25 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with 5 runs
2024-02-05 12:04:27 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with 5 runs
2024-02-05 12:04:29 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with 5 runs
14400               Reverse Ordered 229.88          228.56          230.91
2024-02-05 12:04:30 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with 5 runs
2024-02-05 12:04:31 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with 5 runs
2024-02-05 12:04:32 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with 5 runs
2024-02-05 12:04:33 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with 5 runs
2024-02-05 12:04:33 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with 5 runs
2024-02-05 12:04:34 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with 5 runs
2024-02-05 12:04:35 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with 5 runs
2024-02-05 12:04:36 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with 5 runs
2024-02-05 12:04:37 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with 5 runs
2024-02-05 12:04:37 INFO  Benchmark_Timer - Begin run: Insertion Sort Performance with 5 runs
14400               Partially Ordered 116.24        114.90          117.27

Process finished with exit code 0
```

Benchmark Observation:

Array Type	Array Size	Average Time (ms)	Minimum Time (ms)	Maximum Time (ms)
Randomly Ordered Arrays	450	0.19	0.12	0.39
	900	0.47	0.44	0.49
	1800	1.86	1.84	1.93
	3600	7.34	7.10	7.50
	7200	29.11	28.68	29.69
	14400	117.24	115.54	119.47
Ordered Arrays	450	0.00	0.00	0.00
	900	0.00	0.00	0.00
	1800	0.00	0.00	0.01
	3600	0.01	0.01	0.01
	7200	0.01	0.01	0.01
	14400	0.01	0.01	0.01
Reverse Ordered Arrays	450	0.23	0.22	0.24
	900	0.95	0.93	0.99
	1800	3.65	3.60	3.71
	3600	14.43	14.28	14.62
	7200	57.81	57.57	58.37
	14400	117.24	115.54	119.47
Partially Ordered Arrays	450	0.12	0.12	0.13
	900	0.49	0.46	0.50
	1800	1.84	1.79	1.88
	3600	7.28	7.15	7.42
	7200	28.99	28.84	29.21
	14400	117.24	115.54	119.47