Assignment 2: Benchmark

Problem Statement:

Part 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.

```
public <T, U> double repeat(int n, Supplier<T> supplier, Function<T, U> function, UnaryOperator<T> preFunction,
Consumer<U> postFunction) {
// TO BE IMPLEMENTED
}
```

```
private static long getClock() {
    // TO BE IMPLEMENTED
}
```

```
private static double toMillisecs(long ticks) {
    // TO BE IMPLEMENTED
}
```

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.

Evidence:

Insertion Sort is benchmarked for 4 types of arrays:

1. Ordered Array
2. Partially Ordered Array
3. Random Array
4. Reversed Array

Each type of array is tested for different lengths by the doubling method hence making the lengths:

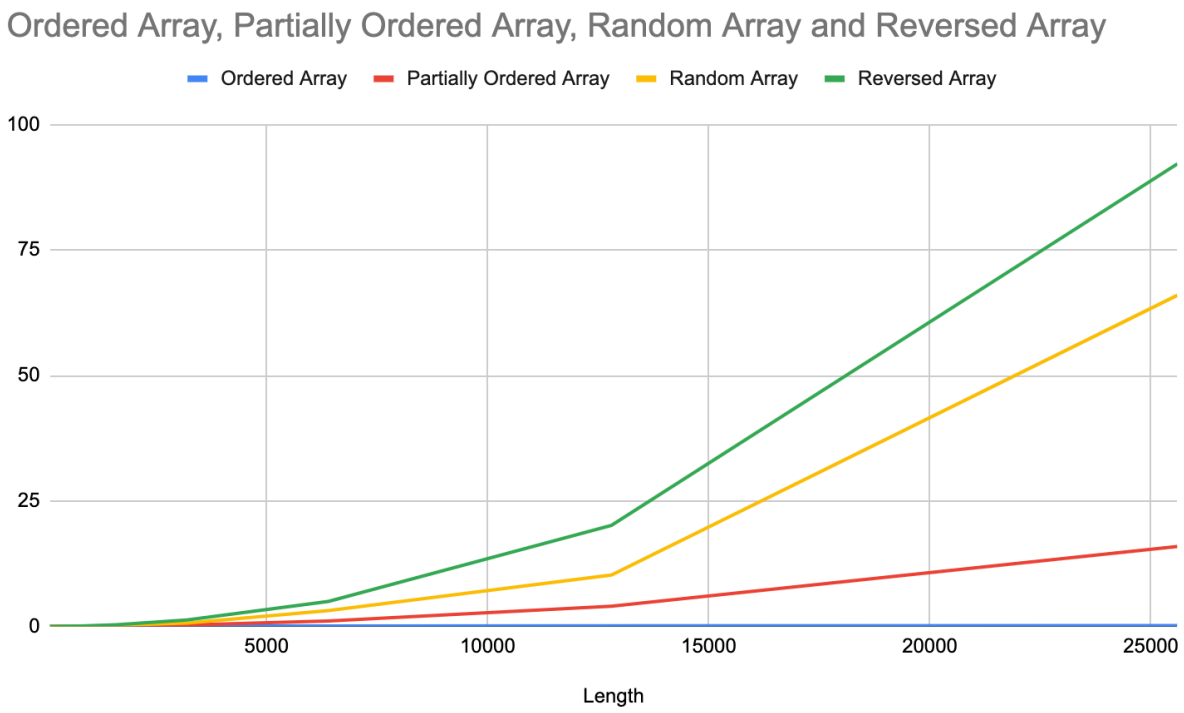
1. 100
2. 200
3. 400
4. 800
5. 1600
6. 3200
7. 6400
8. 12800
9. 25600

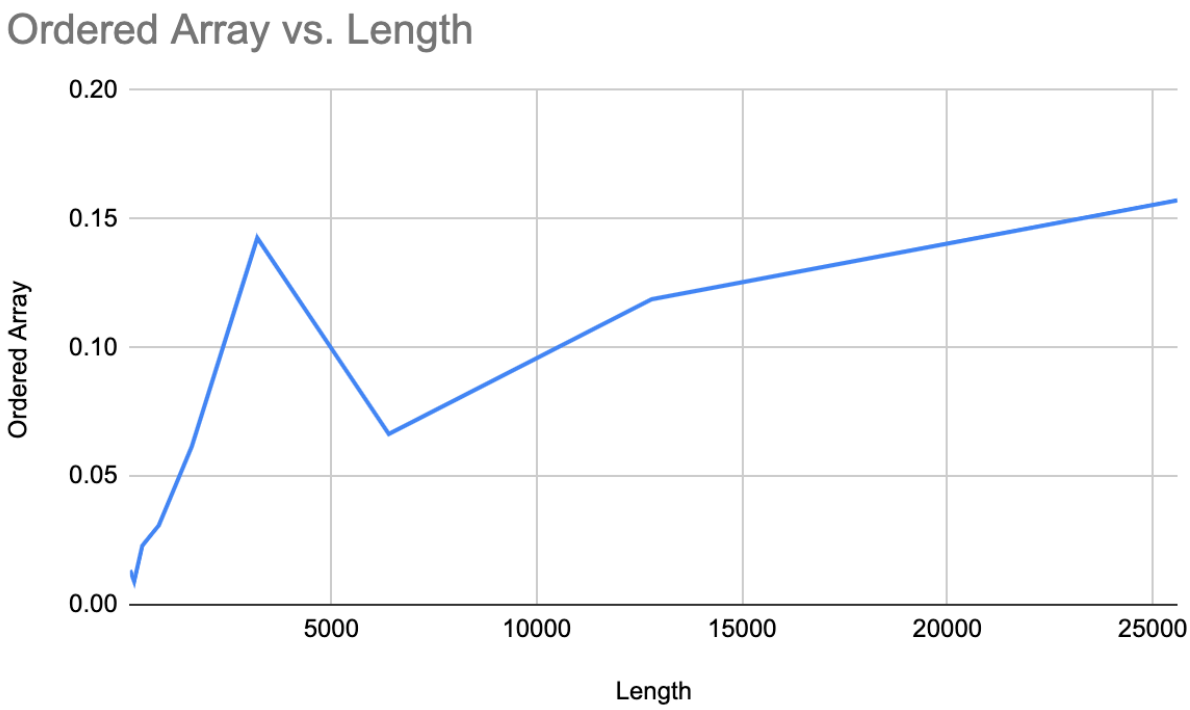
For each length variation the experiment is run ‘m’ number of times (10 times) and the results are present in the files:

1. Raw Data from Output.pdf
2. Graph and Data on sheets.xlsx

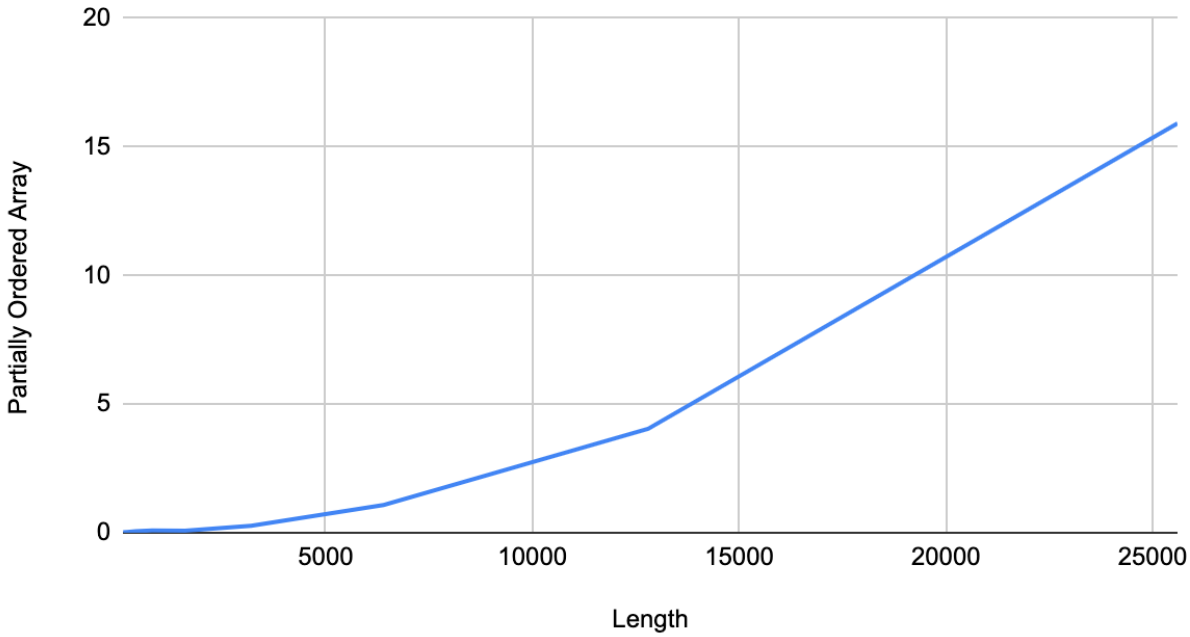
The meantimes collected from the generated data is then plotted on to a graph and the conclusion can be inferred from it.

Length	Ordered Array	Partially Ordered Array	Random Array	Reversed Array
100	0.0133917	0.0119041	0.0018125	0.0022333
200	0.009075	0.02895	0.0042583	0.0070459
400	0.0229751	0.0576249	0.0145249	0.0239956
800	0.0309041	0.0906541	0.0487582	0.0853832
1600	0.0614209	0.0756082	0.1867793	0.3221418
3200	0.1426583	0.2731791	0.6549834	1.2654833
6400	0.0664	1.0747251	3.1561418	4.9669999
12800	0.1187207	4.0384417	10.2389878	20.1043418
25600	0.1572377	15.9145669	65.9875668	92.1550915

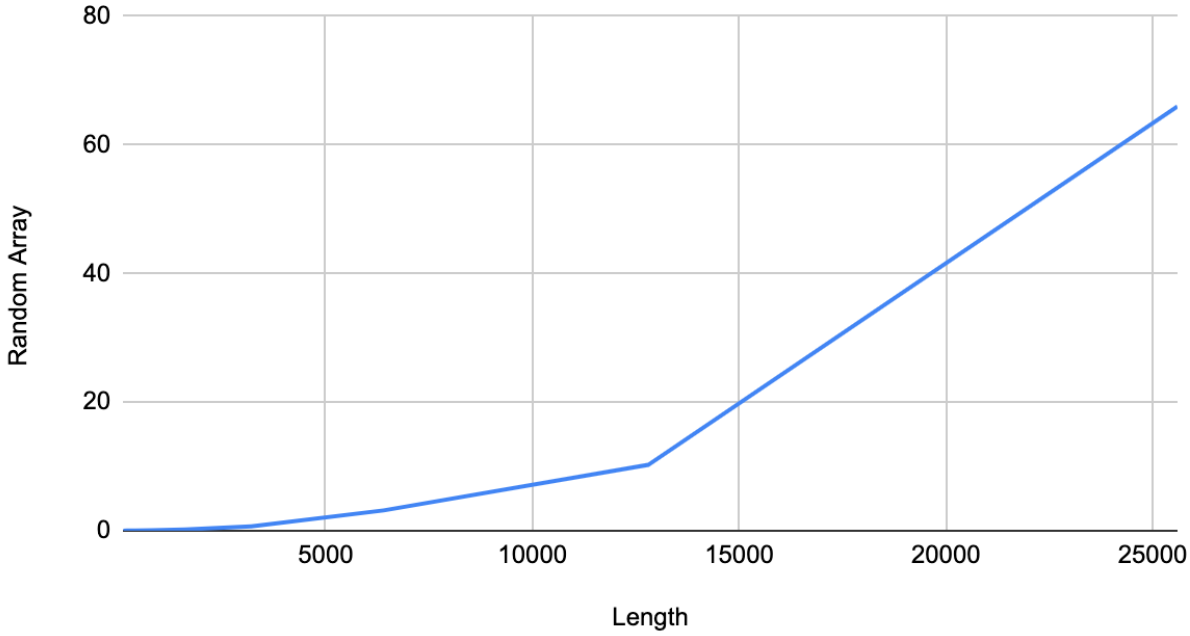

Ordered Array, Partially Ordered Array, Random Array and Reversed Array
Ordered Array
Partially Ordered Array
Random Array
Reversed Array
100
75
50
25
0
5000
10000
15000
20000
25000
Length

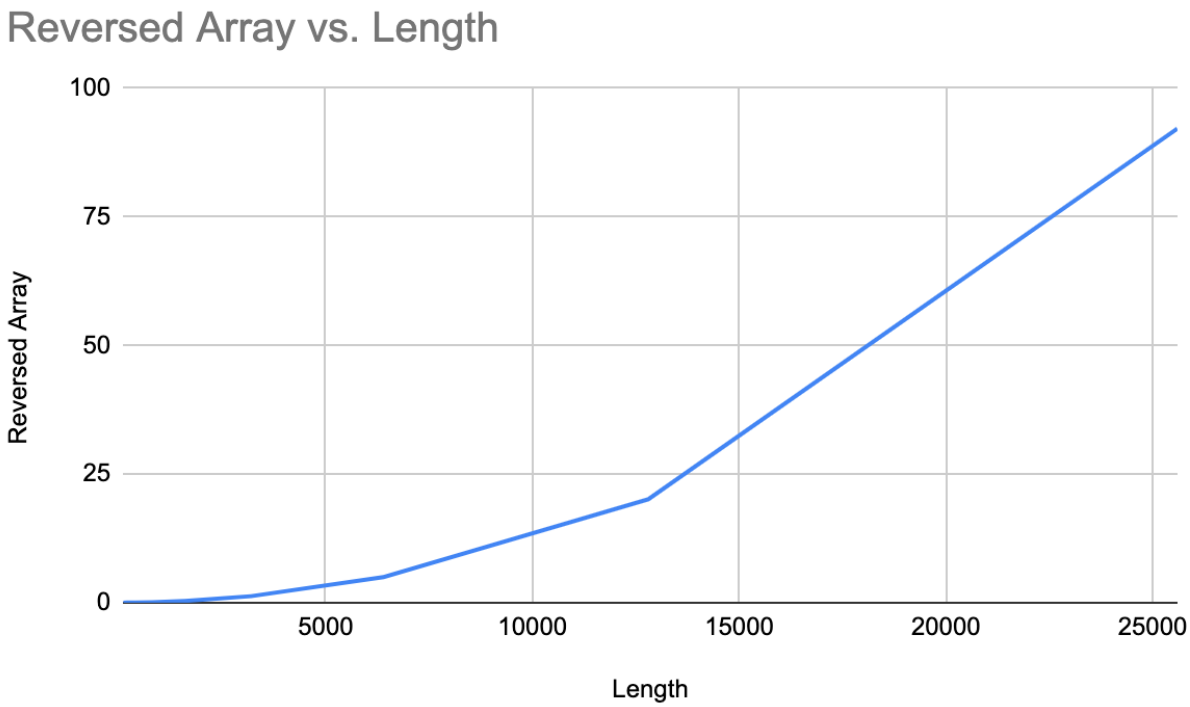

Ordered Array vs. Length
0.20
0.15
0.10
0.05
0.00
Ordered Array
5000
10000
15000
20000
25000
Length

Partially Ordered Array vs. Length



Random Array vs. Length





Conclusion:

We can conclude from the data presented above the following order of types of arrays based on the time taken by each of them:

Ordered Array < Partially Ordered Array < Randomly Ordered Array < Reversed Array

This conclusion is pretty straightforward and is proven by the results that the program has generated.

Another conclusion that can be arrived at by looking at the data is:

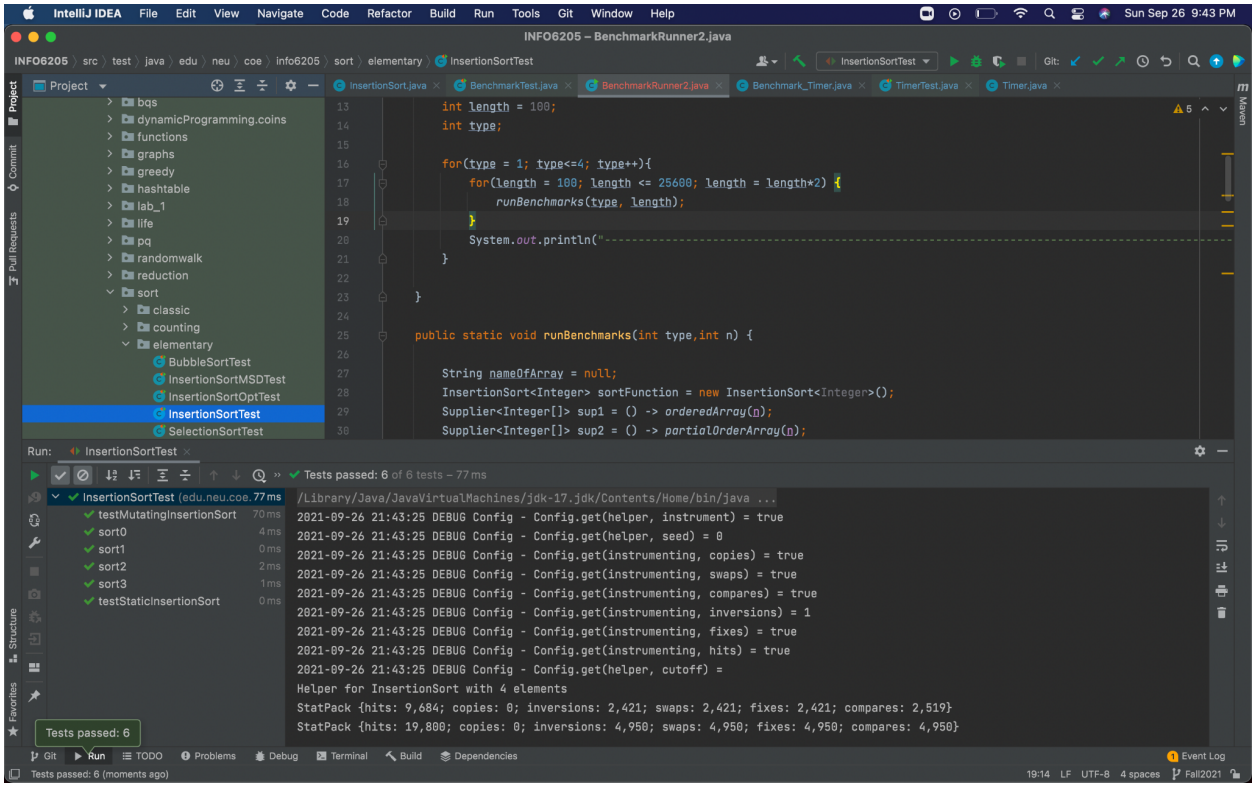
As the length of the array doubles the mean time taken also increases drastically.

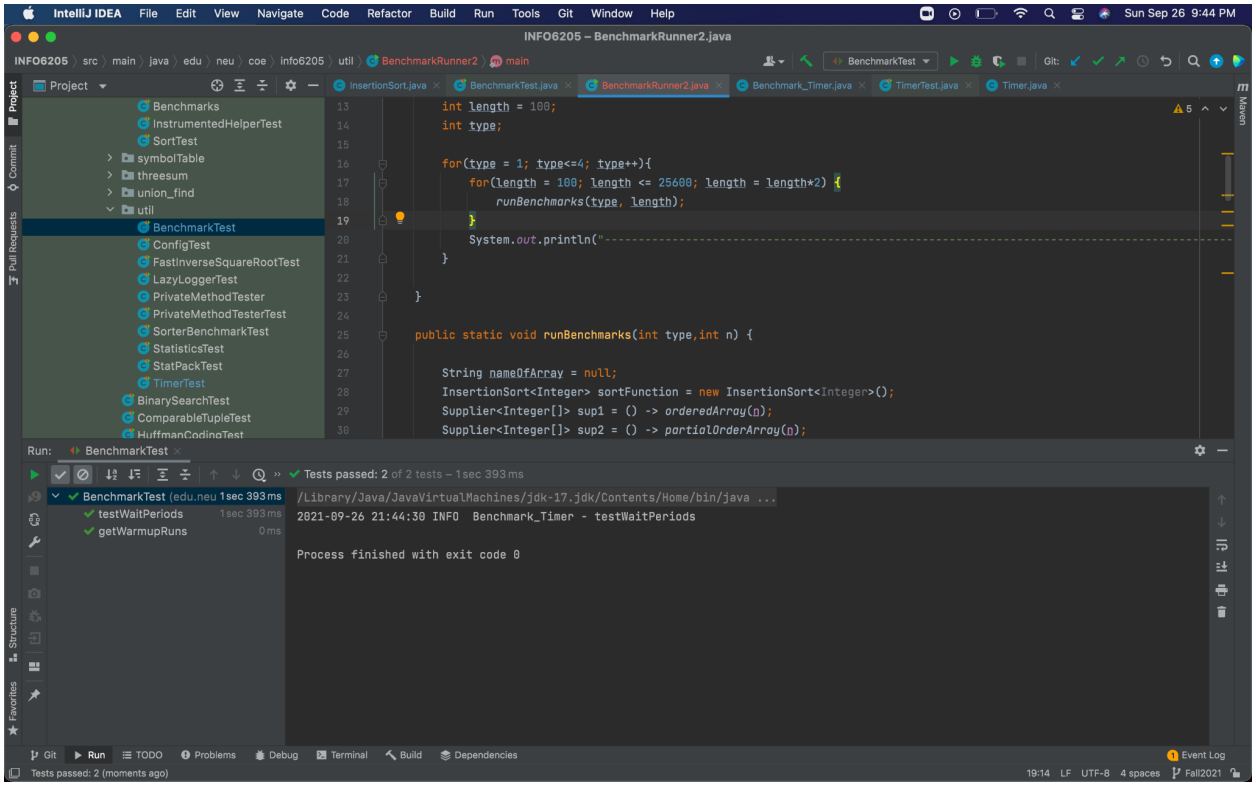
For an ordered array the time taken is proportional to O(n) time.
As the array goes out of order the time taken increases. It increases proportionally to the amount of disorderliness. This effect can be seen in the data for partially ordered and randomly ordered arrays.
For the worst case scenario, that is the reversed array the time it takes is proportional to the time O(n^2).

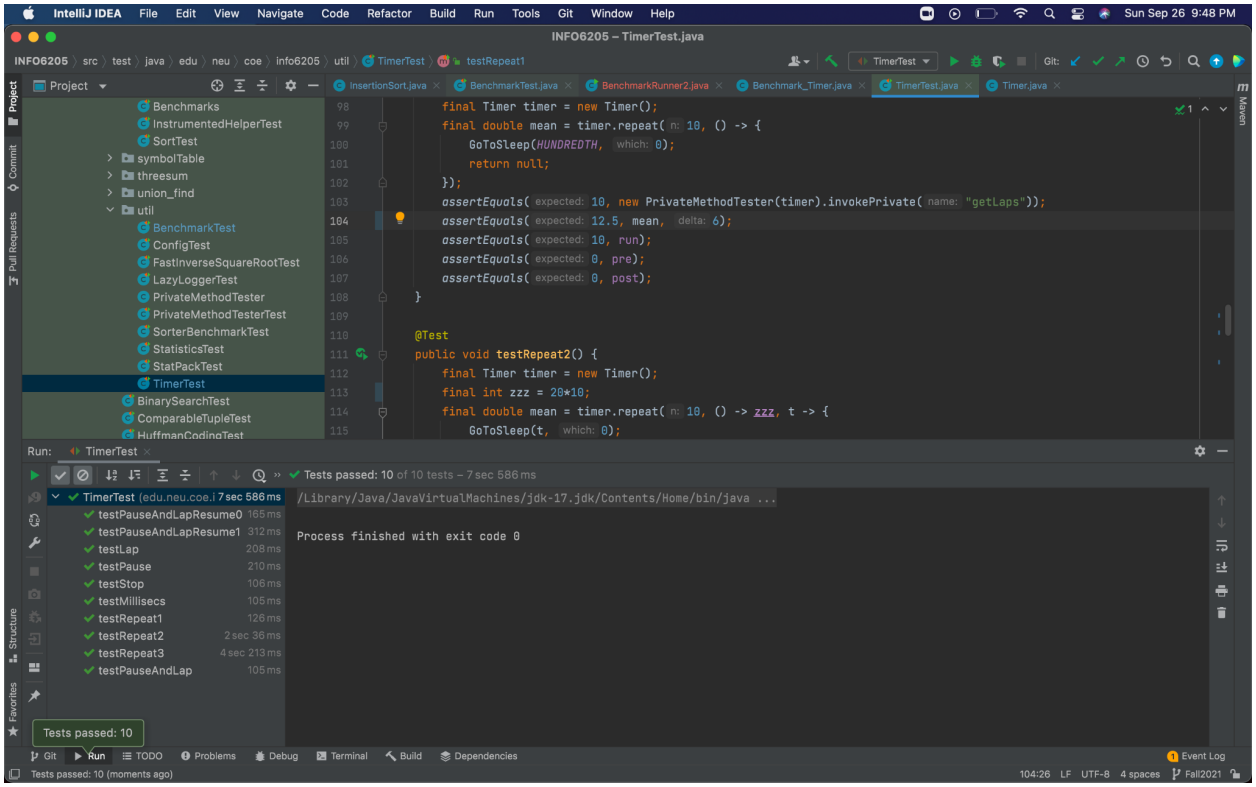
The final conclusion is:

For any array type the mean time is proportional to the value (k * n), where k is a constant that varies from 1 to n.

Screenshots of tests:




IntelliJ IDEA File Edit View Navigate Code Refactor Build Run Tools Git Window Help
Sun Sep 26 9:44 PM
INFO6205 – BenchmarkRunner2.java
Project
Benchmarks
InstrumentedHelperTest
SortTest
symbolTable
threesum
union_find
util
BenchmarkTest
ConfigTest
FastInverseSquareRootTest
LazyLoggerTest
PrivateMethodTester
PrivateMethodTesterTest
SorterBenchmarkTest
StatisticsTest
StatPackTest
TimerTest
BinarySearchTest
ComparableTupleTest
int length = 100;
int type;
for(type = 1; type<=4; type++){
for(length = 100; length <= 25600; length = length*2) {
runBenchmarks(type, length);
}
System.out.println("
}
}
public static void runBenchmarks(int type,int n) {
String nameOfArray = null;
InsertionSort<Integer> sortFunction = new InsertionSort<Integer>();
Supplier<Integer[]> sup1 = () -> orderedArray(n);
Supplier<Integer[]> sup2 = () -> partialOrderArray(n);
Run: BenchmarkTest
Tests passed: 2 of 2 tests – 1 sec 393 ms
BenchmarkTest (edu.neu 1 sec 393 ms
testWaitPeriods 1 sec 393 ms
getWarmupRuns 0 ms
/Library/Java/JavaVirtualMachines/jdk-17.jdk/Contents/Home/bin/java ...
2021-09-26 21:44:30 INFO Benchmark_Timer - testWaitPeriods
Process finished with exit code 0
Git Run TODO Problems Debug Terminal Build Dependencies Event Log
Tests passed: 2 (moments ago)
19:14 LF UTF-8 4 spaces Fall2021


IntelliJ IDEA File Edit View Navigate Code Refactor Build Run Tools Git Window Help
Sun Sep 26 9:48 PM
INFO6205 – TimerTest.java
Project
Benchmarks
InstrumentedHelperTest
SortTest
symbolTable
threesum
union_find
util
BenchmarkTest
ConfigTest
FastInverseSquareRootTest
LazyLoggerTest
PrivateMethodTester
PrivateMethodTesterTest
SorterBenchmarkTest
StatisticsTest
StatPackTest
TimerTest
BinarySearchTest
ComparableTupleTest
final Timer timer = new Timer();
final double mean = timer.repeat(n: 10, () -> {
GoToSleep(HUNDREDTH, which: 0);
return null;
});
assertEquals(expected: 10, new PrivateMethodTester(timer).invokePrivate(name: "getLaps"));
assertEquals(expected: 12.5, mean, delta: 6);
assertEquals(expected: 10, run);
assertEquals(expected: 0, pre);
assertEquals(expected: 0, post);
}
@Test
public void testRepeat2() {
final Timer timer = new Timer();
final int zzz = 20*10;
final double mean = timer.repeat(n: 10, () -> zzz, t -> {
GoToSleep(t, which: 0);
Run: TimerTest
Tests passed: 10 of 10 tests – 7 sec 586 ms
TimerTest (edu.neu.coe.i 7 sec 586 ms
testPauseAndLapResume0 165 ms
testPauseAndLapResume1 312 ms
testLap 208 ms
testPause 210 ms
testStop 106 ms
testMillisecs 105 ms
testRepeat1 126 ms
testRepeat2 2 sec 36 ms
testRepeat3 4 sec 213 ms
testPauseAndLap 105 ms
/Library/Java/JavaVirtualMachines/jdk-17.jdk/Contents/Home/bin/java ...
Process finished with exit code 0
Tests passed: 10
Git Run TODO Problems Debug Terminal Build Dependencies Event Log
Tests passed: 10 (moments ago)
104:26 LF UTF-8 4 spaces Fall2021

Raw Data:

2021-09-26 21:18:00 INFO Benchmark_Timer - InsertionSort:
Array: Ordered Array Length: 100 time: 0.013391700000000001
2021-09-26 21:18:00 INFO Benchmark_Timer - InsertionSort:
Array: Ordered Array Length: 200 time: 0.009075
2021-09-26 21:18:00 INFO Benchmark_Timer - InsertionSort:
Array: Ordered Array Length: 400 time: 0.022975100000000002
2021-09-26 21:18:00 INFO Benchmark_Timer - InsertionSort:
Array: Ordered Array Length: 800 time: 0.0309041
2021-09-26 21:18:00 INFO Benchmark_Timer - InsertionSort:
Array: Ordered Array Length: 1600 time: 0.0614209
2021-09-26 21:18:00 INFO Benchmark_Timer - InsertionSort:
Array: Ordered Array Length: 3200 time: 0.1426583
2021-09-26 21:18:00 INFO Benchmark_Timer - InsertionSort:
Array: Ordered Array Length: 6400 time: 0.0664
2021-09-26 21:18:00 INFO Benchmark_Timer - InsertionSort:
Array: Ordered Array Length: 12800 time: 0.11872069999999998
2021-09-26 21:18:00 INFO Benchmark_Timer - InsertionSort:
Array: Ordered Array Length: 25600 time: 0.15723769999999998

2021-09-26 21:18:00 INFO Benchmark_Timer - InsertionSort:
Array: PartiallyOrdered Array Length: 100 time: 0.011904099999999999
2021-09-26 21:18:00 INFO Benchmark_Timer - InsertionSort:
Array: PartiallyOrdered Array Length: 200 time: 0.028949999999999997
2021-09-26 21:18:00 INFO Benchmark_Timer - InsertionSort:
Array: PartiallyOrdered Array Length: 400 time: 0.0576249
2021-09-26 21:18:00 INFO Benchmark_Timer - InsertionSort:
Array: PartiallyOrdered Array Length: 800 time: 0.0906541
2021-09-26 21:18:00 INFO Benchmark_Timer - InsertionSort:
Array: PartiallyOrdered Array Length: 1600 time: 0.0756082
2021-09-26 21:18:00 INFO Benchmark_Timer - InsertionSort:
Array: PartiallyOrdered Array Length: 3200 time: 0.2731791
2021-09-26 21:18:00 INFO Benchmark_Timer - InsertionSort:
Array: PartiallyOrdered Array Length: 6400 time: 1.0747251
2021-09-26 21:18:00 INFO Benchmark_Timer - InsertionSort:
Array: PartiallyOrdered Array Length: 12800 time: 4.0384417
2021-09-26 21:18:00 INFO Benchmark_Timer - InsertionSort:
Array: PartiallyOrdered Array Length: 25600 time: 15.9145669

2021-09-26 21:18:01 INFO Benchmark_Timer - InsertionSort:
Array: Random Array Length: 100 time: 0.0018124999999999999
2021-09-26 21:18:01 INFO Benchmark_Timer - InsertionSort:

Array: Random Array Length: 200 time: 0.0042583000000000005
2021-09-26 21:18:01 INFO Benchmark_Timer - InsertionSort:
Array: Random Array Length: 400 time: 0.014524899999999999
2021-09-26 21:18:01 INFO Benchmark_Timer - InsertionSort:
Array: Random Array Length: 800 time: 0.0487582
2021-09-26 21:18:01 INFO Benchmark_Timer - InsertionSort:
Array: Random Array Length: 1600 time: 0.1867793
2021-09-26 21:18:01 INFO Benchmark_Timer - InsertionSort:
Array: Random Array Length: 3200 time: 0.6549834
2021-09-26 21:18:01 INFO Benchmark_Timer - InsertionSort:
Array: Random Array Length: 6400 time: 3.1561418
2021-09-26 21:18:01 INFO Benchmark_Timer - InsertionSort:
Array: Random Array Length: 12800 time: 10.2389878
2021-09-26 21:18:01 INFO Benchmark_Timer - InsertionSort:
Array: Random Array Length: 25600 time: 65.9875668

2021-09-26 21:18:02 INFO Benchmark_Timer - InsertionSort:
Array: Reversed Array Length: 100 time: 0.0022332999999999997
2021-09-26 21:18:02 INFO Benchmark_Timer - InsertionSort:
Array: Reversed Array Length: 200 time: 0.007045899999999999
2021-09-26 21:18:02 INFO Benchmark_Timer - InsertionSort:
Array: Reversed Array Length: 400 time: 0.0239956
2021-09-26 21:18:02 INFO Benchmark_Timer - InsertionSort:
Array: Reversed Array Length: 800 time: 0.0853832
2021-09-26 21:18:02 INFO Benchmark_Timer - InsertionSort:
Array: Reversed Array Length: 1600 time: 0.3221418
2021-09-26 21:18:02 INFO Benchmark_Timer - InsertionSort:
Array: Reversed Array Length: 3200 time: 1.2654833
2021-09-26 21:18:02 INFO Benchmark_Timer - InsertionSort:
Array: Reversed Array Length: 6400 time: 4.966999899999999
2021-09-26 21:18:02 INFO Benchmark_Timer - InsertionSort:
Array: Reversed Array Length: 12800 time: 20.1043418
2021-09-26 21:18:03 INFO Benchmark_Timer - InsertionSort:
Array: Reversed Array Length: 25600 time: 92.1550915

Process finished with exit code 0