Program Structures and Algorithms
Spring 2023(SEC –01)
Assignment-3

NAME: Ashi Tyagi
NUID: 002706544

Task:

Your task for this assignment is in three parts.

- (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. The APIs of these class are as follows:

```
public interface Benchmark<T> {
    default double run(T t, int m) {
        return runFromSupplier(() -> t, m);
    }

    double runFromSupplier(Supplier<T> supplier, int m);
}
```

[*Supplier* is a Java function type which supplies values of type *T* using the method: *get()*.]

```
public class Benchmark_Timer<T> implements Benchmark<T> {
public Benchmark_Timer(String description, UnaryOperator<T> fPre,
Consumer<T> fRun, Consumer<T> fPost)
```

[*Consumer<T>* is a Java function type which consumes a type *T* with the method: *accept*(t).

UnaryOperator<T> is essentially an alias of *Function<T, T>* which defines *apply(t)* which takes a *T* and returns a *T*.]

```
public Benchmark_Timer(String description, UnaryOperator<T> fPre,
Consumer<T> fRun)
public Benchmark_Timer(String description, Consumer<T> fRun, Consumer<T>
fPost)
public Benchmark_Timer(String description, Consumer<T> f)
public class Timer {
... // see below for methods to be implemented...
}
public <T, U> double repeat(int n, Supplier<T> supplier, Function<T, U>
function, UnaryOperator<T> preFunction, Consumer<U> postFunction) {
// TO BE IMPLEMENTED
}
```

[*Function<T, U>* defines a method *U apply(t: T)*, which takes a value of *T* and returns a value of *U*.]

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

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

The function to be timed, hereinafter the "target" function, is the *Consumer* function *fRun* (or just *f*) passed in to one or other of the constructors. For example, you might create a function which sorts an array with *n* elements.

The generic type *T* is that of the input to the target function.

The first parameter to the first run method signature is the parameter that will, in turn, be passed to target function. In the second signature, *supplier* will be invoked each time to get a *t* which is passed to the other run method.

The second parameter to the *run* function (*m)* is the number of times the target function will be called.

The return value from *run* is the average number of milliseconds taken for each run of the target function.

Don't forget to check your implementation by running the unit tests in *BenchmarkTest* and *TimerTest*. If you have trouble with the exact timings in the unit tests, it's quite OK (in this assignment only) to change parameters until the tests run. Different machine architectures will result in different behavior.

- (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.

As usual, the submission will be your entire project (*clean, i.e. without the target and project folders).* There are stubs and unit tests in the repository.

Report on your observations and show screenshots of the runs and also the unit tests. Please note that you may have to adjust the required execution time for the insertion sort unit test(s) because your computer may not run at the same speed as mine.

Further notes: you should use the *System.nanoTime* method to get the clock time. This isn't guaranteed to be accurate which is one of the reasons you should run the experiment several

times for each value of *n*. Also, for each invocation of *run*, run the given target function ten times to get the system "warmed up" before you start the timing properly.

The *Sort* interface takes care of copying the array when the *sort(array)* signature is called. It returns a new array as a result. The original array is unchanged. Therefore, you do not need to worry about the insertion-based sorts getting quicker because of the arrays getting more sorted (they don't).

Relationship Conclusion:

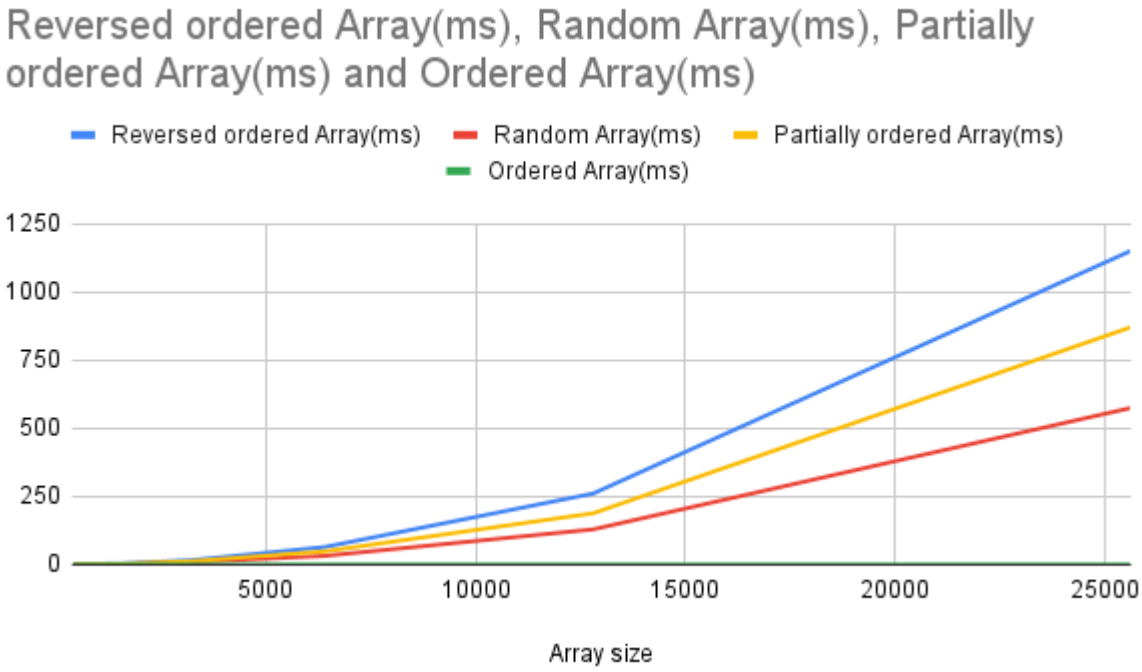
Array size	Reversed ordered Array(ms)	Random Array(ms)	Partially ordered Array(ms)	Ordered Array(ms)
400	2.35	1.09	0.76	0.4
800	3.12	1.52	1.26	0.3
1600	4.47	3.89	3.31	0.3
3200	16.03	14.02	12.29	0.25
6400	64.41	55.87	48.69	0.39
12800	260.96	201.2	188.56	0.33
25600	1151.3	1001.3	871.2	0.47

From the graph above we can conclude that:

1. Reverse Order (N^2) > Random Order (N^2) > Partially Ordered (N^2) > Ordered (N)
2. Thus, it can be concluded by observing the experimental values above that reverse order takes the most amount of time while the ordered takes the least amount of time.

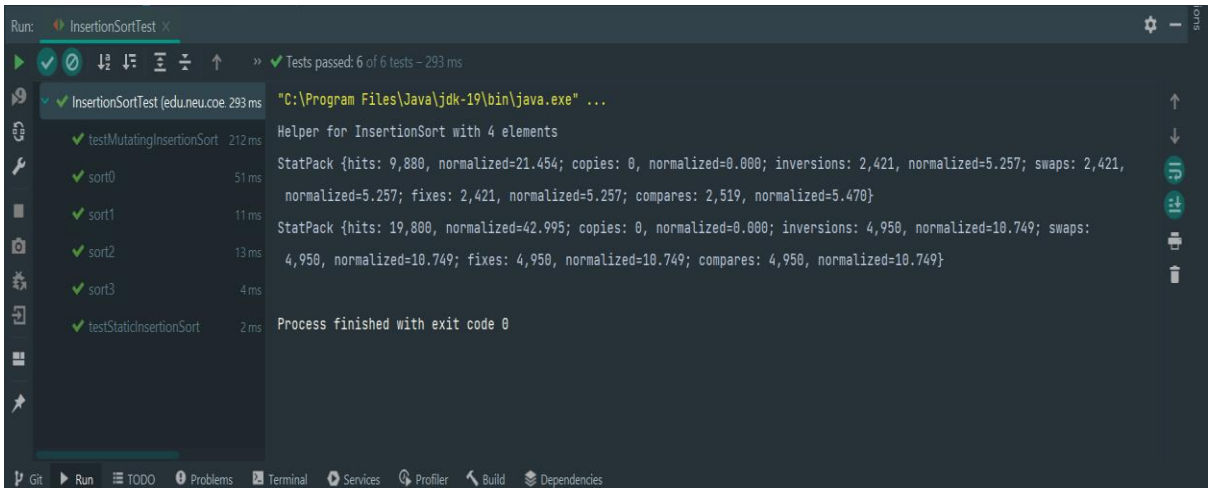
Evidence Graph:

Array size	Reversed ordered Array(ms)	Random Array(ms)	Partially ordered Array(ms)	Ordered Array(ms)
400	2.35	1.09	0.76	0.4
800	3.12	1.52	1.26	0.3
1600	4.47	3.89	3.31	0.3
3200	16.03	14.02	12.29	0.25
6400	64.41	55.87	48.69	0.39
12800	260.96	201.2	188.56	0.33
25600	1151.3	1001.3	871.2	0.47

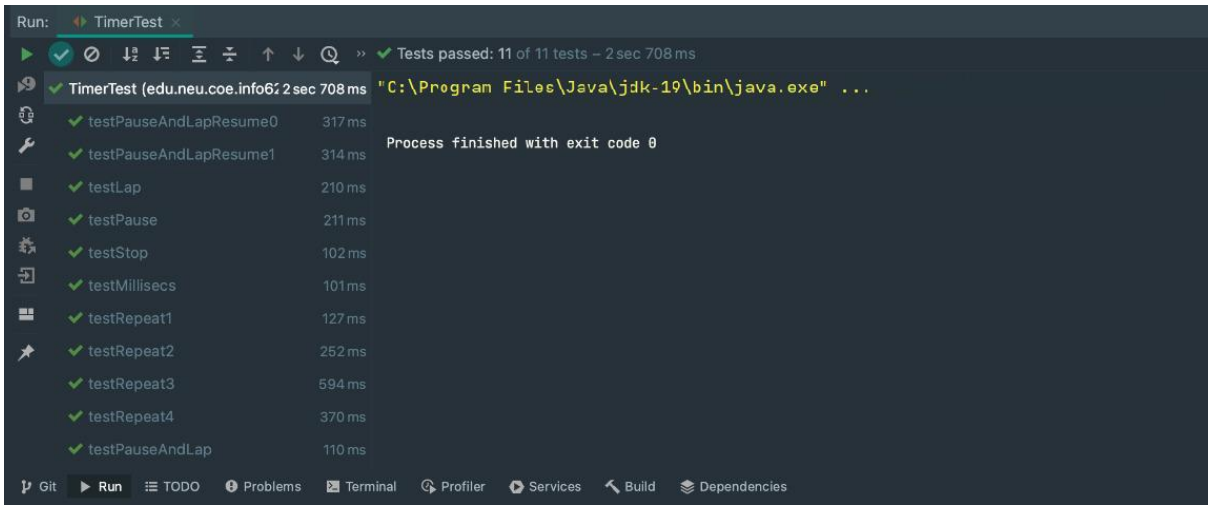


Unit Test Cases:

Insertion Sort:

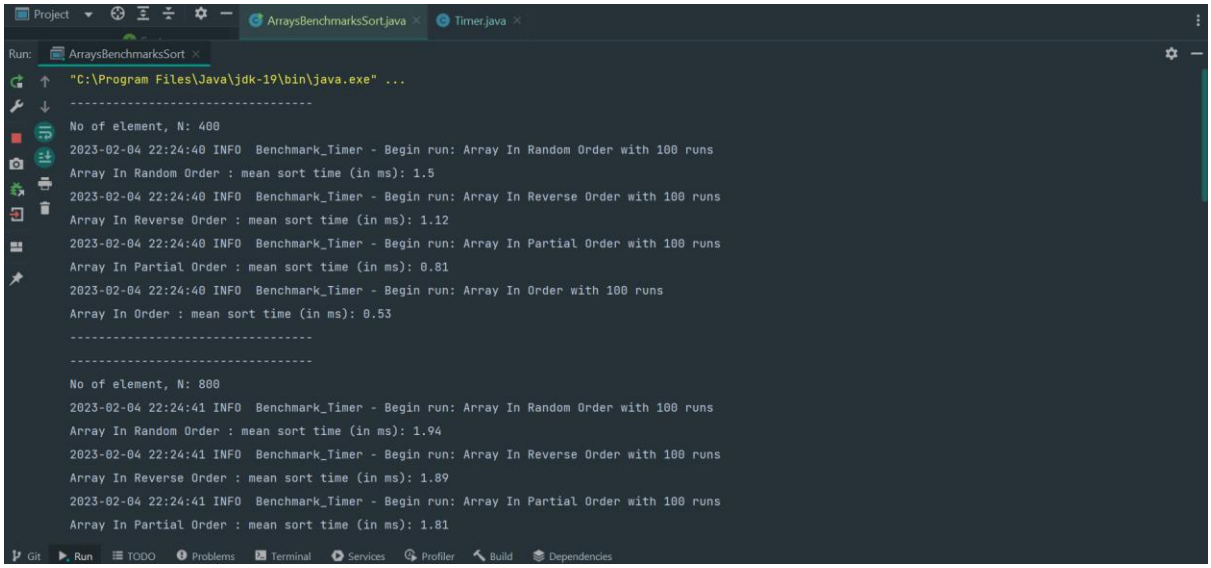


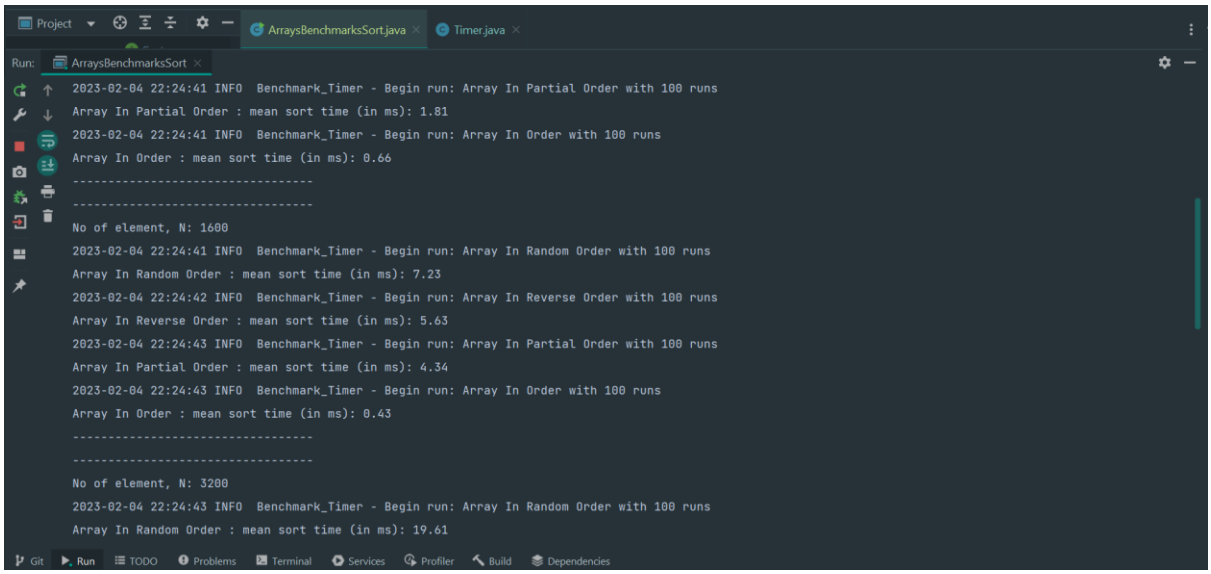
Timer:

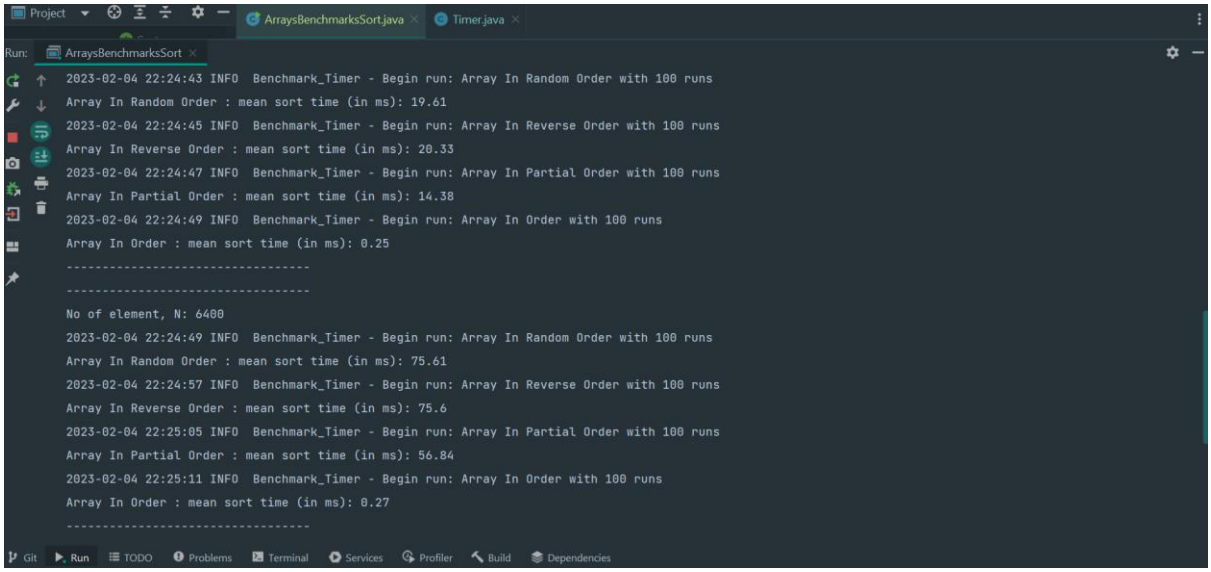


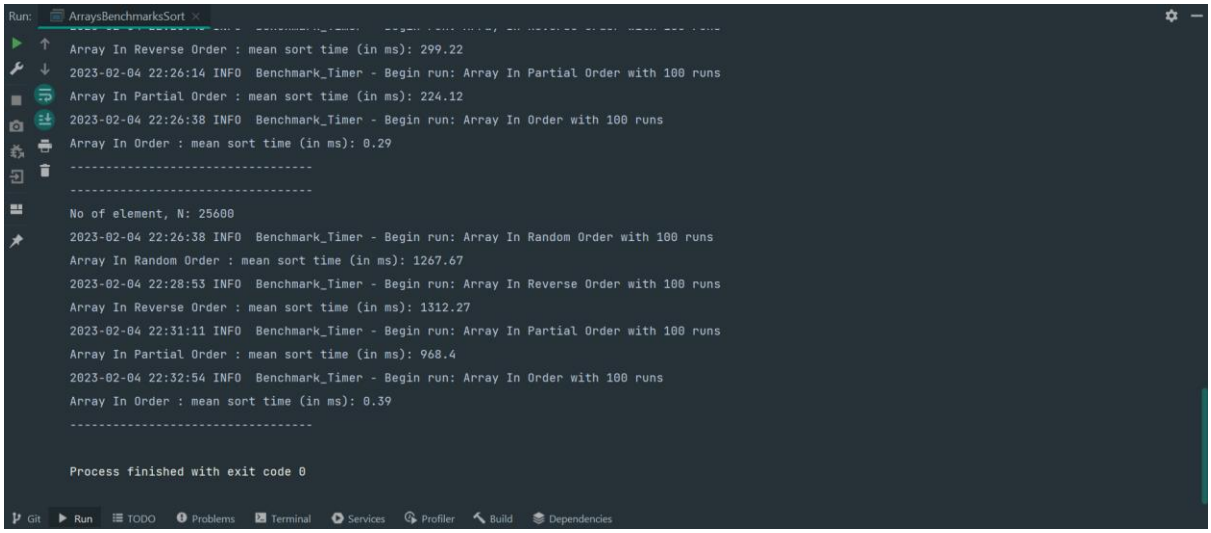
Benchmark test:

Arrays Benchmark testcases:


Project
ArraysBenchmarksSort.java
Timer.java
Run:
ArraysBenchmarksSort
"C:\Program Files\Java\jdk-19\bin\java.exe" ...
No of element, N: 400
2023-02-04 22:24:40 INFO Benchmark_Timer - Begin run: Array In Random Order with 100 runs
Array In Random Order : mean sort time (in ms): 1.5
2023-02-04 22:24:40 INFO Benchmark_Timer - Begin run: Array In Reverse Order with 100 runs
Array In Reverse Order : mean sort time (in ms): 1.12
2023-02-04 22:24:40 INFO Benchmark_Timer - Begin run: Array In Partial Order with 100 runs
Array In Partial Order : mean sort time (in ms): 0.81
2023-02-04 22:24:40 INFO Benchmark_Timer - Begin run: Array In Order with 100 runs
Array In Order : mean sort time (in ms): 0.53
No of element, N: 800
2023-02-04 22:24:41 INFO Benchmark_Timer - Begin run: Array In Random Order with 100 runs
Array In Random Order : mean sort time (in ms): 1.94
2023-02-04 22:24:41 INFO Benchmark_Timer - Begin run: Array In Reverse Order with 100 runs
Array In Reverse Order : mean sort time (in ms): 1.89
2023-02-04 22:24:41 INFO Benchmark_Timer - Begin run: Array In Partial Order with 100 runs
Array In Partial Order : mean sort time (in ms): 1.81
Git
Run
TODO
Problems
Terminal
Services
Profiler
Build
Dependencies


Project
ArraysBenchmarksSort.java
Timer.java
Run:
ArraysBenchmarksSort
2023-02-04 22:24:41 INFO Benchmark_Timer - Begin run: Array In Partial Order with 100 runs
Array In Partial Order : mean sort time (in ms): 1.81
2023-02-04 22:24:41 INFO Benchmark_Timer - Begin run: Array In Order with 100 runs
Array In Order : mean sort time (in ms): 0.66
No of element, N: 1600
2023-02-04 22:24:41 INFO Benchmark_Timer - Begin run: Array In Random Order with 100 runs
Array In Random Order : mean sort time (in ms): 7.23
2023-02-04 22:24:42 INFO Benchmark_Timer - Begin run: Array In Reverse Order with 100 runs
Array In Reverse Order : mean sort time (in ms): 5.63
2023-02-04 22:24:43 INFO Benchmark_Timer - Begin run: Array In Partial Order with 100 runs
Array In Partial Order : mean sort time (in ms): 4.34
2023-02-04 22:24:43 INFO Benchmark_Timer - Begin run: Array In Order with 100 runs
Array In Order : mean sort time (in ms): 0.43
No of element, N: 3200
2023-02-04 22:24:43 INFO Benchmark_Timer - Begin run: Array In Random Order with 100 runs
Array In Random Order : mean sort time (in ms): 19.61
Git
Run
TODO
Problems
Terminal
Services
Profiler
Build
Dependencies


Project
ArraysBenchmarksSort.java
Timer.java
Run:
ArraysBenchmarksSort
2023-02-04 22:24:43 INFO Benchmark_Timer - Begin run: Array In Random Order with 100 runs
Array In Random Order : mean sort time (in ms): 19.61
2023-02-04 22:24:45 INFO Benchmark_Timer - Begin run: Array In Reverse Order with 100 runs
Array In Reverse Order : mean sort time (in ms): 20.33
2023-02-04 22:24:47 INFO Benchmark_Timer - Begin run: Array In Partial Order with 100 runs
Array In Partial Order : mean sort time (in ms): 14.38
2023-02-04 22:24:49 INFO Benchmark_Timer - Begin run: Array In Order with 100 runs
Array In Order : mean sort time (in ms): 0.25
No of element, N: 6400
2023-02-04 22:24:49 INFO Benchmark_Timer - Begin run: Array In Random Order with 100 runs
Array In Random Order : mean sort time (in ms): 75.61
2023-02-04 22:24:57 INFO Benchmark_Timer - Begin run: Array In Reverse Order with 100 runs
Array In Reverse Order : mean sort time (in ms): 75.6
2023-02-04 22:25:05 INFO Benchmark_Timer - Begin run: Array In Partial Order with 100 runs
Array In Partial Order : mean sort time (in ms): 56.84
2023-02-04 22:25:11 INFO Benchmark_Timer - Begin run: Array In Order with 100 runs
Array In Order : mean sort time (in ms): 0.27
Git
Run
TODO
Problems
Terminal
Services
Profiler
Build
Dependencies


Run:
ArraysBenchmarksSort
Array In Reverse Order : mean sort time (in ms): 299.22
2023-02-04 22:26:14 INFO Benchmark_Timer - Begin run: Array In Partial Order with 100 runs
Array In Partial Order : mean sort time (in ms): 224.12
2023-02-04 22:26:38 INFO Benchmark_Timer - Begin run: Array In Order with 100 runs
Array In Order : mean sort time (in ms): 0.29
No of element, N: 25600
2023-02-04 22:26:38 INFO Benchmark_Timer - Begin run: Array In Random Order with 100 runs
Array In Random Order : mean sort time (in ms): 1267.67
2023-02-04 22:28:53 INFO Benchmark_Timer - Begin run: Array In Reverse Order with 100 runs
Array In Reverse Order : mean sort time (in ms): 1312.27
2023-02-04 22:31:11 INFO Benchmark_Timer - Begin run: Array In Partial Order with 100 runs
Array In Partial Order : mean sort time (in ms): 968.4
2023-02-04 22:32:54 INFO Benchmark_Timer - Begin run: Array In Order with 100 runs
Array In Order : mean sort time (in ms): 0.39
Process finished with exit code 0
Git
Run
TODO
Problems
Terminal
Services
Profiler
Build
Dependencies