Zixuan.Zhu (002196924)

**Program Structures & Algorithms**

**Fall 2021**

**Assignment No. 2**

* **Task**

Your task for this assignment is in three parts.

(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. 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);  
}

public class Benchmark\_Timer<T> implements Benchmark<T> {

public Benchmark\_Timer(String description, UnaryOperator<T> fPre, Consumer<T> fRun, Consumer<T> fPost)

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  
}

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:**

Comparison of the running time of four differently sorted but same

length(n) arrays (random, ordered, partially-ordered and reverse-o-

dered)when using insertion sort:

reverse-ordered>random>partially-ordered>ordered

(“partially-ordered” in this assignment means that array is ordered from

“array.length/2” to “array.length”)

* **Evidence to support the conclusion:**

1. **Output**

*B-E-G-I-N*

*Round 1*

*Random: n = 40*

*2021-09-26 16:59:17 INFO Benchmark\_Timer - Begin run: InsertionSort Time with 30 runs*

*meantime:1.3333333333333333*

*meantime:1.0333333333333334*

*Meantime: 1.0333333333333334*

*Partially-ordered: n = 40*

*2021-09-26 16:59:17 INFO Benchmark\_Timer - Begin run: InsertionSort Time with 30 runs*

*meantime:0.6666666666666666*

*meantime:0.8333333333333334*

*Meantime: 0.8333333333333334*

*Ordered: n = 40*

*2021-09-26 16:59:17 INFO Benchmark\_Timer - Begin run: InsertionSort Time with 30 runs*

*meantime:0.6666666666666666*

*meantime:0.7333333333333333*

*Meantime: 0.7333333333333333*

*Reverse-ordered n = 40*

*2021-09-26 16:59:17 INFO Benchmark\_Timer - Begin run: InsertionSort Time with 30 runs*

*meantime:0.6666666666666666*

*meantime:0.7333333333333333*

*Mean lap time: 0.7333333333333333*

*Round 2*

*Random: n = 80*

*2021-09-26 16:59:17 INFO Benchmark\_Timer - Begin run: InsertionSort Time with 30 runs*

*meantime:0.3333333333333333*

*meantime:0.5666666666666667*

*Meantime: 0.5666666666666667*

*Partially-ordered: n = 80*

*2021-09-26 16:59:17 INFO Benchmark\_Timer - Begin run: InsertionSort Time with 30 runs*

*meantime:0.3333333333333333*

*meantime:0.7333333333333333*

*Meantime: 0.7333333333333333*

*Ordered: n = 80*

*2021-09-26 16:59:17 INFO Benchmark\_Timer - Begin run: InsertionSort Time with 30 runs*

*meantime:0.3333333333333333*

*meantime:0.3333333333333333*

*Meantime: 0.3333333333333333*

*Reverse-ordered n = 80*

*2021-09-26 16:59:17 INFO Benchmark\_Timer - Begin run: InsertionSort Time with 30 runs*

*meantime:0.3333333333333333*

*meantime:0.5*

*Mean lap time: 0.5*

*Round 3*

*Random: n = 160*

*2021-09-26 16:59:17 INFO Benchmark\_Timer - Begin run: InsertionSort Time with 30 runs*

*meantime:0.3333333333333333*

*meantime:0.5*

*Meantime: 0.5*

*Partially-ordered: n = 160*

*2021-09-26 16:59:17 INFO Benchmark\_Timer - Begin run: InsertionSort Time with 30 runs*

*meantime:0.3333333333333333*

*meantime:0.36666666666666664*

*Meantime: 0.36666666666666664*

*Ordered: n = 160*

*2021-09-26 16:59:17 INFO Benchmark\_Timer - Begin run: InsertionSort Time with 30 runs*

*meantime:0.3333333333333333*

*meantime:0.8*

*Meantime: 0.8*

*Reverse-ordered n = 160*

*2021-09-26 16:59:17 INFO Benchmark\_Timer - Begin run: InsertionSort Time with 30 runs*

*meantime:0.3333333333333333*

*meantime:0.4666666666666667*

*Mean lap time: 0.4666666666666667*

*Round 4*

*Random: n = 320*

*2021-09-26 16:59:17 INFO Benchmark\_Timer - Begin run: InsertionSort Time with 30 runs*

*meantime:0.3333333333333333*

*meantime:0.5333333333333333*

*Meantime: 0.5333333333333333*

*Partially-ordered: n = 320*

*2021-09-26 16:59:17 INFO Benchmark\_Timer - Begin run: InsertionSort Time with 30 runs*

*meantime:0.3333333333333333*

*meantime:0.36666666666666664*

*Meantime: 0.36666666666666664*

*Ordered: n = 320*

*2021-09-26 16:59:17 INFO Benchmark\_Timer - Begin run: InsertionSort Time with 30 runs*

*meantime:0.0*

*meantime:0.3333333333333333*

*Meantime: 0.3333333333333333*

*Reverse-ordered n = 320*

*2021-09-26 16:59:17 INFO Benchmark\_Timer - Begin run: InsertionSort Time with 30 runs*

*meantime:0.6666666666666666*

*meantime:0.9666666666666667*

*Mean lap time: 0.9666666666666667*

*Round 5*

*Random: n = 640*

*2021-09-26 16:59:17 INFO Benchmark\_Timer - Begin run: InsertionSort Time with 30 runs*

*meantime:1.3333333333333333*

*meantime:0.9666666666666667*

*Meantime: 0.9666666666666667*

*Partially-ordered: n = 640*

*2021-09-26 16:59:17 INFO Benchmark\_Timer - Begin run: InsertionSort Time with 30 runs*

*meantime:0.6666666666666666*

*meantime:0.6*

*Meantime: 0.6*

*Ordered: n = 640*

*2021-09-26 16:59:17 INFO Benchmark\_Timer - Begin run: InsertionSort Time with 30 runs*

*meantime:0.3333333333333333*

*meantime:0.6*

*Meantime: 0.6*

*Reverse-ordered n = 640*

*2021-09-26 16:59:17 INFO Benchmark\_Timer - Begin run: InsertionSort Time with 30 runs*

*meantime:1.0*

*meantime:1.2666666666666666*

*Mean lap time: 1.2666666666666666*

*Round 6*

*Random: n = 1280*

*2021-09-26 16:59:17 INFO Benchmark\_Timer - Begin run: InsertionSort Time with 30 runs*

*meantime:1.6666666666666667*

*meantime:1.9666666666666666*

*Meantime: 1.9666666666666666*

*Partially-ordered: n = 1280*

*2021-09-26 16:59:17 INFO Benchmark\_Timer - Begin run: InsertionSort Time with 30 runs*

*meantime:1.3333333333333333*

*meantime:2.2666666666666666*

*Meantime: 2.2666666666666666*

*Ordered: n = 1280*

*2021-09-26 16:59:17 INFO Benchmark\_Timer - Begin run: InsertionSort Time with 30 runs*

*meantime:0.6666666666666666*

*meantime:0.23333333333333334*

*Meantime: 0.23333333333333334*

*Reverse-ordered n = 1280*

*2021-09-26 16:59:17 INFO Benchmark\_Timer - Begin run: InsertionSort Time with 30 runs*

*meantime:3.6666666666666665*

*meantime:5.033333333333333*

*Mean lap time: 5.033333333333333*

*Round 7*

*Random: n = 2560*

*2021-09-26 16:59:17 INFO Benchmark\_Timer - Begin run: InsertionSort Time with 30 runs*

*meantime:6.666666666666667*

*meantime:7.633333333333334*

*Meantime: 7.633333333333334*

*Partially-ordered: n = 2560*

*2021-09-26 16:59:18 INFO Benchmark\_Timer - Begin run: InsertionSort Time with 30 runs*

*meantime:4.666666666666667*

*meantime:5.166666666666667*

*Meantime: 5.166666666666667*

*Ordered: n = 2560*

*2021-09-26 16:59:18 INFO Benchmark\_Timer - Begin run: InsertionSort Time with 30 runs*

*meantime:0.0*

*meantime:0.13333333333333333*

*Meantime: 0.13333333333333333*

*Reverse-ordered n = 2560*

*2021-09-26 16:59:18 INFO Benchmark\_Timer - Begin run: InsertionSort Time with 30 runs*

*meantime:13.0*

*meantime:12.7*

*Mean lap time: 12.7*

*Round 8*

*Random: n = 5120*

*2021-09-26 16:59:18 INFO Benchmark\_Timer - Begin run: InsertionSort Time with 30 runs*

*meantime:26.333333333333332*

*meantime:29.833333333333332*

*Meantime: 29.833333333333332*

*Partially-ordered: n = 5120*

*2021-09-26 16:59:19 INFO Benchmark\_Timer - Begin run: InsertionSort Time with 30 runs*

*meantime:20.666666666666668*

*meantime:22.766666666666666*

*Meantime: 22.766666666666666*

*Ordered: n = 5120*

*2021-09-26 16:59:20 INFO Benchmark\_Timer - Begin run: InsertionSort Time with 30 runs*

*meantime:0.0*

*meantime:0.13333333333333333*

*Meantime: 0.13333333333333333*

*Reverse-ordered n = 5120*

*2021-09-26 16:59:20 INFO Benchmark\_Timer - Begin run: InsertionSort Time with 30 runs*

*meantime:63.0*

*meantime:62.266666666666666*

*Mean lap time: 62.266666666666666*

*Round 9*

*Random: n = 10240*

*2021-09-26 16:59:22 INFO Benchmark\_Timer - Begin run: InsertionSort Time with 30 runs*

*meantime:154.0*

*meantime:123.2*

*Meantime: 123.2*

*Partially-ordered: n = 10240*

*2021-09-26 16:59:26 INFO Benchmark\_Timer - Begin run: InsertionSort Time with 30 runs*

*meantime:108.0*

*meantime:78.86666666666666*

*Meantime: 78.86666666666666*

*Ordered: n = 10240*

*2021-09-26 16:59:29 INFO Benchmark\_Timer - Begin run: InsertionSort Time with 30 runs*

*meantime:0.0*

*meantime:0.2*

*Meantime: 0.2*

*Reverse-ordered n = 10240*

*2021-09-26 16:59:29 INFO Benchmark\_Timer - Begin run: InsertionSort Time with 30 runs*

*meantime:216.0*

*meantime:206.66666666666666*

*Mean lap time: 206.66666666666666*

*Round 10*

*Random: n = 20480*

*2021-09-26 16:59:36 INFO Benchmark\_Timer - Begin run: InsertionSort Time with 30 runs*

*meantime:442.3333333333333*

*meantime:449.6666666666667*

*Meantime: 449.6666666666667*

*Partially-ordered: n = 20480*

*2021-09-26 16:59:51 INFO Benchmark\_Timer - Begin run: InsertionSort Time with 30 runs*

*meantime:336.0*

*meantime:324.73333333333335*

*Meantime: 324.73333333333335*

*Ordered: n = 20480*

*2021-09-26 17:00:02 INFO Benchmark\_Timer - Begin run: InsertionSort Time with 30 runs*

*meantime:0.3333333333333333*

*meantime:0.23333333333333334*

*Meantime: 0.23333333333333334*

*Reverse-ordered n = 20480*

*2021-09-26 17:00:02 INFO Benchmark\_Timer - Begin run: InsertionSort Time with 30 runs*

*meantime:967.3333333333334*

*meantime:956.1*

*Mean lap time: 956.1*

*E-N-D*

1. **Graphical Representation**

Data:

表格

描述已自动生成

“n” is the number of elements of the array.

Graph:

* **Unit tests result:**

BenchmarkTest:

图形用户界面, 文本, 应用程序

描述已自动生成

InsertionSortTest:图形用户界面, 文本, 应用程序

描述已自动生成

TimerTest:图形用户界面, 文本, 应用程序

描述已自动生成