**Divya Kulkarni (001543678) SEC -01**

**Program Structures & Algorithms**

**Spring 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);  
  }

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*.

* (Part 2) Implement *InsertionSort*(in the *InsertionSort* class) by simply looking up the insertion code used by*Arrays.sort.* You should use the *helper.swap* method although you could also just copy that from the same source code. You should of course 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).

If you need clarification, ask on Slack.

**Output/Observations: I have created Benchmark\_InsertionSort.java for main program**

**Case 1:**

**Array Elements for Ordered Array: 5,000**

**Array Elements for Reversed Array: 5,000**

**Runs: 10 for 5 times**

2021-02-03 17:13:36 INFO Benchmark\_Timer - Begin run: testWaitPeriods with 10 runs

xOrdered for 10 runs: 0.2

2021-02-03 17:13:36 INFO Benchmark\_Timer - Begin run: testWaitPeriods with 10 runs

xReverse for 10 runs: 0.0

2021-02-03 17:13:36 INFO Benchmark\_Timer - Begin run: testWaitPeriods with 20 runs

xOrdered for 20 runs: 0.0

2021-02-03 17:13:36 INFO Benchmark\_Timer - Begin run: testWaitPeriods with 20 runs

xReverse for 20 runs: 0.0

2021-02-03 17:13:36 INFO Benchmark\_Timer - Begin run: testWaitPeriods with 40 runs

xOrdered for 40 runs: 0.025

2021-02-03 17:13:36 INFO Benchmark\_Timer - Begin run: testWaitPeriods with 40 runs

xReverse for 40 runs: 0.025

2021-02-03 17:13:36 INFO Benchmark\_Timer - Begin run: testWaitPeriods with 80 runs

xOrdered for 80 runs: 0.075

2021-02-03 17:13:36 INFO Benchmark\_Timer - Begin run: testWaitPeriods with 80 runs

xReverse for 80 runs: 0.025

2021-02-03 17:13:36 INFO Benchmark\_Timer - Begin run: testWaitPeriods with 160 runs

xOrdered for 160 runs: 0.01875

2021-02-03 17:13:36 INFO Benchmark\_Timer - Begin run: testWaitPeriods with 160 runs

xReverse for 160 runs: 0.01875

2021-02-03 17:13:36 INFO Benchmark\_Timer - Begin run: testWaitPeriods with 320 runs

xOrdered for 320 runs: 0.01875

2021-02-03 17:13:36 INFO Benchmark\_Timer - Begin run: testWaitPeriods with 320 runs

xReverse for 320 runs: 0.01875

**Observation: Runtime improved with number of runs**

**Case 2:**

**Array Elements for Ordered Array: 10,000**

**Array Elements for Reversed Array: 10,000**

**Runs: 10 for 5 times**

2021-02-03 16:59:55 INFO Benchmark\_Timer - Begin run: testWaitPeriods with 10 runs

xOrdered for 10 runs: 0.2

2021-02-03 16:59:55 INFO Benchmark\_Timer - Begin run: testWaitPeriods with 10 runs

xReverse for 10 runs: 0.0

2021-02-03 16:59:55 INFO Benchmark\_Timer - Begin run: testWaitPeriods with 20 runs

xOrdered for 20 runs: 0.05

2021-02-03 16:59:55 INFO Benchmark\_Timer - Begin run: testWaitPeriods with 20 runs

xReverse for 20 runs: 0.05

2021-02-03 16:59:55 INFO Benchmark\_Timer - Begin run: testWaitPeriods with 40 runs

xOrdered for 40 runs: 0.05

2021-02-03 16:59:55 INFO Benchmark\_Timer - Begin run: testWaitPeriods with 40 runs

xReverse for 40 runs: 0.025

2021-02-03 16:59:55 INFO Benchmark\_Timer - Begin run: testWaitPeriods with 80 runs

xOrdered for 80 runs: 0.05

2021-02-03 16:59:55 INFO Benchmark\_Timer - Begin run: testWaitPeriods with 80 runs

xReverse for 80 runs: 0.05

2021-02-03 16:59:55 INFO Benchmark\_Timer - Begin run: testWaitPeriods with 160 runs

xOrdered for 160 runs: 0.05625

2021-02-03 16:59:55 INFO Benchmark\_Timer - Begin run: testWaitPeriods with 160 runs

xReverse for 160 runs: 0.05

2021-02-03 16:59:55 INFO Benchmark\_Timer - Begin run: testWaitPeriods with 320 runs

xOrdered for 320 runs: 0.053125

2021-02-03 16:59:55 INFO Benchmark\_Timer - Begin run: testWaitPeriods with 320 runs

xReverse for 320 runs: 0.05625

**Case 3:**

**Array Elements for Ordered Array: 40,000**

**Array Elements for Reversed Array: 40,000**

**Runs: 10 for 5 times**

2021-02-03 17:03:44 INFO Benchmark\_Timer - Begin run: testWaitPeriods with 10 runs

xOrdered for 10 runs: 0.4

2021-02-03 17:03:44 INFO Benchmark\_Timer - Begin run: testWaitPeriods with 10 runs

xReverse for 10 runs: 0.0

2021-02-03 17:03:49 INFO Benchmark\_Timer - Begin run: testWaitPeriods with 20 runs

xOrdered for 20 runs: 0.05

2021-02-03 17:03:49 INFO Benchmark\_Timer - Begin run: testWaitPeriods with 20 runs

xReverse for 20 runs: 0.05

2021-02-03 17:03:49 INFO Benchmark\_Timer - Begin run: testWaitPeriods with 40 runs

xOrdered for 40 runs: 0.15

2021-02-03 17:03:49 INFO Benchmark\_Timer - Begin run: testWaitPeriods with 40 runs

xReverse for 40 runs: 0.175

2021-02-03 17:03:49 INFO Benchmark\_Timer - Begin run: testWaitPeriods with 80 runs

xOrdered for 80 runs: 0.175

2021-02-03 17:03:49 INFO Benchmark\_Timer - Begin run: testWaitPeriods with 80 runs

xReverse for 80 runs: 0.175

2021-02-03 17:03:49 INFO Benchmark\_Timer - Begin run: testWaitPeriods with 160 runs

xOrdered for 160 runs: 0.175

2021-02-03 17:03:49 INFO Benchmark\_Timer - Begin run: testWaitPeriods with 160 runs

xReverse for 160 runs: 0.175

2021-02-03 17:03:49 INFO Benchmark\_Timer - Begin run: testWaitPeriods with 320 runs

xOrdered for 320 runs: 0.171875

2021-02-03 17:03:49 INFO Benchmark\_Timer - Begin run: testWaitPeriods with 320 runs

xReverse for 320 runs: 0.18125

**Case 4:**

**Array Elements for Ordered Array: 80,000**

**Array Elements for Reversed Array: 80,000**

**Runs: 10 for 5 times**

2021-02-03 19:41:48 INFO Benchmark\_Timer - Begin run: testWaitPeriods with 10 runs

xOrdered for 10 runs: 0.5

2021-02-03 19:41:48 INFO Benchmark\_Timer - Begin run: testWaitPeriods with 10 runs

xReverse for 10 runs: 0.1

2021-02-03 19:42:07 INFO Benchmark\_Timer - Begin run: testWaitPeriods with 20 runs

xOrdered for 20 runs: 0.15

2021-02-03 19:42:07 INFO Benchmark\_Timer - Begin run: testWaitPeriods with 20 runs

xReverse for 20 runs: 0.3

2021-02-03 19:42:07 INFO Benchmark\_Timer - Begin run: testWaitPeriods with 40 runs

xOrdered for 40 runs: 0.375

2021-02-03 19:42:07 INFO Benchmark\_Timer - Begin run: testWaitPeriods with 40 runs

xReverse for 40 runs: 0.375

2021-02-03 19:42:07 INFO Benchmark\_Timer - Begin run: testWaitPeriods with 80 runs

xOrdered for 80 runs: 0.375

2021-02-03 19:42:07 INFO Benchmark\_Timer - Begin run: testWaitPeriods with 80 runs

xReverse for 80 runs: 0.375

2021-02-03 19:42:07 INFO Benchmark\_Timer - Begin run: testWaitPeriods with 160 runs

xOrdered for 160 runs: 0.3875

2021-02-03 19:42:08 INFO Benchmark\_Timer - Begin run: testWaitPeriods with 160 runs

xReverse for 160 runs: 0.3875

2021-02-03 19:42:08 INFO Benchmark\_Timer - Begin run: testWaitPeriods with 320 runs

xOrdered for 320 runs: 0.390625

2021-02-03 19:42:08 INFO Benchmark\_Timer - Begin run: testWaitPeriods with 320 runs

xReverse for 320 runs: 0.396875

**Conclusion:**

While running experiment for the input size of 5000 elements, it took 0.075 milliseconds for 80 runs. After comparing other experiments for array of size 10000, 40000, and 80000, it took 0.05, 0.175, and 0.375 milliseconds for the same number of runs. Time increased with the increase in the number of elements in an array. It took more number of swaps in the reversed array than an ordered array, which is reflected by time taken to sort the array. The complexity of Insertion sort in the reverse array was worst case, whereas it behaved the best in the sorted array.

**Passed Test Cases:**

**1: Timer Test**

**Graphical user interface, text

Description automatically generated**

**2: Benchmark Test**

Graphical user interface, text

Description automatically generated

**3: Insertion Sort Test**

**Graphical user interface, text

Description automatically generated**