**Assignment 3 - Program Structures and Algorithms Spring 2023(SEC - 1)**

**NAME: Paurush Batish NUID: 002755631**

**Task - 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.

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  
}

Solution –

Text

Description automatically generated with medium confidence

Graphical user interface, text

Description automatically generated

**Unit Test –**

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

**Task - 2:**

(Part 2) Implement InsertionSort(in the InsertionSort class) by simply looking up the insertion code used byArrays.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.

Text

Description automatically generated

I have also implemented the insertion sort basic code –

Graphical user interface, text, application

Description automatically generated

**Unit Test –**

Graphical user interface, text, application, email

Description automatically generated

**Task - 3:**

(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.

Solution –

Taking no of operations as 5

Random array:

Time elapsed: 23ms

Ordered array:

Time elapsed: 1ms

Partially ordered array:

Time elapsed: 17ms

Reverse ordered array:

Time elapsed: 7ms

Random array:

Time elapsed: 5ms

Ordered array:

Time elapsed: 1ms

Partially ordered array:

Time elapsed: 4ms

Reverse ordered array:

Time elapsed: 10ms

Random array:

Time elapsed: 19ms

Ordered array:

Time elapsed: 0ms

Partially ordered array:

Time elapsed: 19ms

Reverse ordered array:

Time elapsed: 38ms

Random array:

Time elapsed: 75ms

Ordered array:

Time elapsed: 0ms

Partially ordered array:

Time elapsed: 75ms

Reverse ordered array:

Time elapsed: 154ms

Random array:

Time elapsed: 369ms

Ordered array:

Time elapsed: 0ms

Partially ordered array:

Time elapsed: 304ms

Reverse ordered array:

Time elapsed: 607ms

Chart, line chart

Description automatically generated

Taking no of operations as 10

Random array:

Time elapsed: 27ms

Ordered array:

Time elapsed: 0ms

Partially ordered array:

Time elapsed: 17ms

Reverse ordered array:

Time elapsed: 6ms

Random array:

Time elapsed: 6ms

Ordered array:

Time elapsed: 0ms

Partially ordered array:

Time elapsed: 5ms

Reverse ordered array:

Time elapsed: 9ms

Random array:

Time elapsed: 19ms

Ordered array:

Time elapsed: 0ms

Partially ordered array:

Time elapsed: 19ms

Reverse ordered array:

Time elapsed: 38ms

Random array:

Time elapsed: 75ms

Ordered array:

Time elapsed: 0ms

Partially ordered array:

Time elapsed: 77ms

Reverse ordered array:

Time elapsed: 153ms

Random array:

Time elapsed: 359ms

Ordered array:

Time elapsed: 1ms

Partially ordered array:

Time elapsed: 305ms

Reverse ordered array:

Time elapsed: 613ms

Random array:

Time elapsed: 1227ms

Ordered array:

Time elapsed: 0ms

Partially ordered array:

Time elapsed: 1220ms

Reverse ordered array:

Time elapsed: 2439ms

Random array:

Time elapsed: 7941ms

Ordered array:

Time elapsed: 0ms

Partially ordered array:

Time elapsed: 4877ms

Reverse ordered array:

Time elapsed: 9869ms

Random array:

Time elapsed: 22153ms

Ordered array:

Time elapsed: 0ms

Partially ordered array:

Time elapsed: 20949ms

Reverse ordered array:

Time elapsed: 39393ms

Random array:

Time elapsed: 141614ms

Ordered array:

Time elapsed: 1ms

Partially ordered array:

Time elapsed: 83242ms

Reverse ordered array:

Time elapsed: 167183ms

Random array:

Time elapsed: 902150ms

Ordered array:

Time elapsed: 2ms

Partially ordered array:

Time elapsed: 1126381ms

Reverse ordered array:

Time elapsed: 1784869ms

Chart, line chart

Description automatically generated

Based on the operations we can see that sorted array takes minimum time to perform insertion sort whereas reverse ordered array takes maximum time.

For a sorted list 🡪

Since the list is already in sorted order the complexity will be in O(n).

For reverse sorted list 🡪

We have to traverse n-1 times the inner loop and same with outer loop. Then the O(n\*(n-1)) = O()