Program Structures and Algorithms
Spring 2023(SEC – 1)

NAME: PAWAN KUMAR KRISHNAN
NUID: 002743773

Task: (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.

(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.

Relationship Conclusion:
there seems to be a correlation between the number of elements (N) and the time taken for the four operations (Reverse Ordered, Ordered, Randomly Ordered, Partially Ordered). As the number of elements increases, the time taken for all operations increases.

The Reverse Ordered operation has the highest time taken compared to the other operations for all values of N.

The Ordered operation has the lowest time taken compared to the other operations for all values of N.

The Randomly Ordered and Partially Ordered operations have intermediate values of time taken compared to the other operations.

When sorting a sorted array, Insertion sort has advantage because the array is already in order, and each element only needs to be compared to elements before it once. In this case, the time complexity is O(n), where n is the number of elements in the array, since each element only needs to be compared to one element before it on average.

When sorting a reversely sorted array, Insertion sort will perform poorly because each element will need to be compared to all elements before it, leading to the worst-case time complexity of O(n^2). This is because the algorithm will need to swap each element with the first element before it that is larger than it, leading to a large number of swaps and comparisons.

When sorting a partially sorted array, Insertion sort's performance will depend on how sorted the array is. If the array is only partially sorted, then the time complexity will still be O(n^2) in the worst-case scenario, but if the array is mostly sorted, then the time complexity will be closer to O(n). This is because if the array is mostly sorted, then each element will only need to be compared to a few elements before it, reducing the number of comparisons and swaps.

When sorting a randomly sorted array, Insertion sort's performance will be somewhere in between a sorted and a reversely sorted array, with an average time complexity of O(n^2). This is because the algorithm will need to compare and swap elements in both directions, leading to an average number of comparisons and swaps.

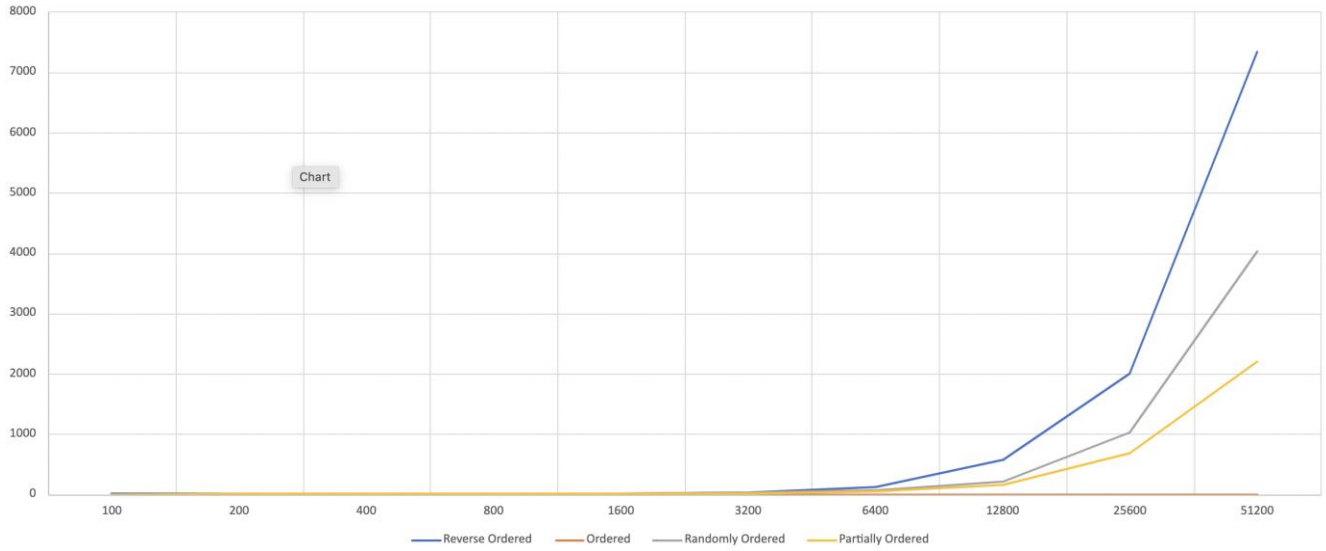
Evidence to support that conclusion:

Time in milliseconds for insertion sorting of different type of array inputs vs N.

N	Reverse Ordered	Ordered	Randomly Ordered	Partially Ordered
100	3.0955422	0.3085086	0.6877584	1.0375582
200	4.1728084	0.2836418	0.8256832	1.356367
400	4.2969338	0.448433	1.8263	1.456108
800	7.275475	0.3491586	2.7101086	1.8521
1600	11.1136918	0.2814336	8.0901	2.8742584
3200	32.9989914	0.2716586	19.6271086	7.0096746
6400	114.01795	0.3615918	55.7328748	47.9463084
12800	565.4831664	0.3644254	207.1537166	152.6360084
25600	2008.117992	0.59876	1030.01785	684.4534418
51200	7344.665133	0.7857998	4028.509983	2203.911417

Graphical Representation:

N(x-axis) vs Time in milliseconds (y-axis)



Unit Test Screenshots:

Insertion Sort Test