Manik Kumar (001063023) – Section 02

INFO 6205 - Fall 2020

Program Structures & Algorithms

Assignment 2

1. **Task:**
   - Complete the implementation of Timer class and Benchmark_Timer class.
   - Implement insertion sort in the InsertionSort class.
   - Measure the running time of insertion sort in the following cases; Ordered array, Randomly ordered, Partially ordered array, and Reverse ordered array.
2. **Output**

   To produce the desired outputs, I coded the insertion sort to be timed for increasingly doubling elements {400, 800, 1600, 3200, 6200, 12800, 25600, 51200}. For each number, the time is averaged over 5 trails, and a warm-up session ran before running each trial.

   Outputs are calculated for the following four cases.
   a. Ordered Elements
   b. Random Elements
   c. Partially Ordered Elements
   d. Reverse Ordered Elements

Elements	Ordered	Partial	Random	Reverse
400	0.0ms	0.4ms	0.6ms	0.8ms
800	0.0ms	0.4ms	0.8ms	2.2ms
1600	0.0ms	0.8ms	4.0ms	7.8ms
3200	0.0ms	4.2ms	16.2ms	31.4ms
6400	0.0ms	18.0ms	72.0ms	142.4ms
12800	0.0ms	95.6ms	257.0ms	529.0ms
25600	0.1ms	333.4ms	1233.2ms	1903.6ms
51260	0.1ms	1337.2ms	5497.8ms	7771.2ms

3. Observation

From the above-shown output, we can observe certain things about insertion sort.

- With the increasingly doubling elements, we can observe the quadratic growth in the time taken by the algorithm. The average time taken is $O(n^2)$.
- Insertion sort is the quickest for ordered or sorted arrays. It is the best-case scenario for insertion sort where the algorithm only has to traverse through the array without performing any swapping. Time taken is $O(n)$.
- The absolute worst case is the reverse ordered list, where the algorithm case to perform several swaps for every single element. Time taken is $O(n^2)$.
- Partially ordered lists are a relatively better case where the number of swaps the algorithm must perform is a relatively smaller number, allowing the algorithm to be quicker. Time taken is again $O(n^2)$.
- The big difference in time taken between the $O(n)$ case when compared to $O(n^2)$ depicts how much having a non-quadratic run-time can be beneficial for algorithms.

The following graphs are plotting to represent the quadratic growth of time taken by the algorithm, and the difference between the Partially Ordered, Random, and Reverse Ordered Array.

Note: Ordered array was left out because the majority of its values were 0.

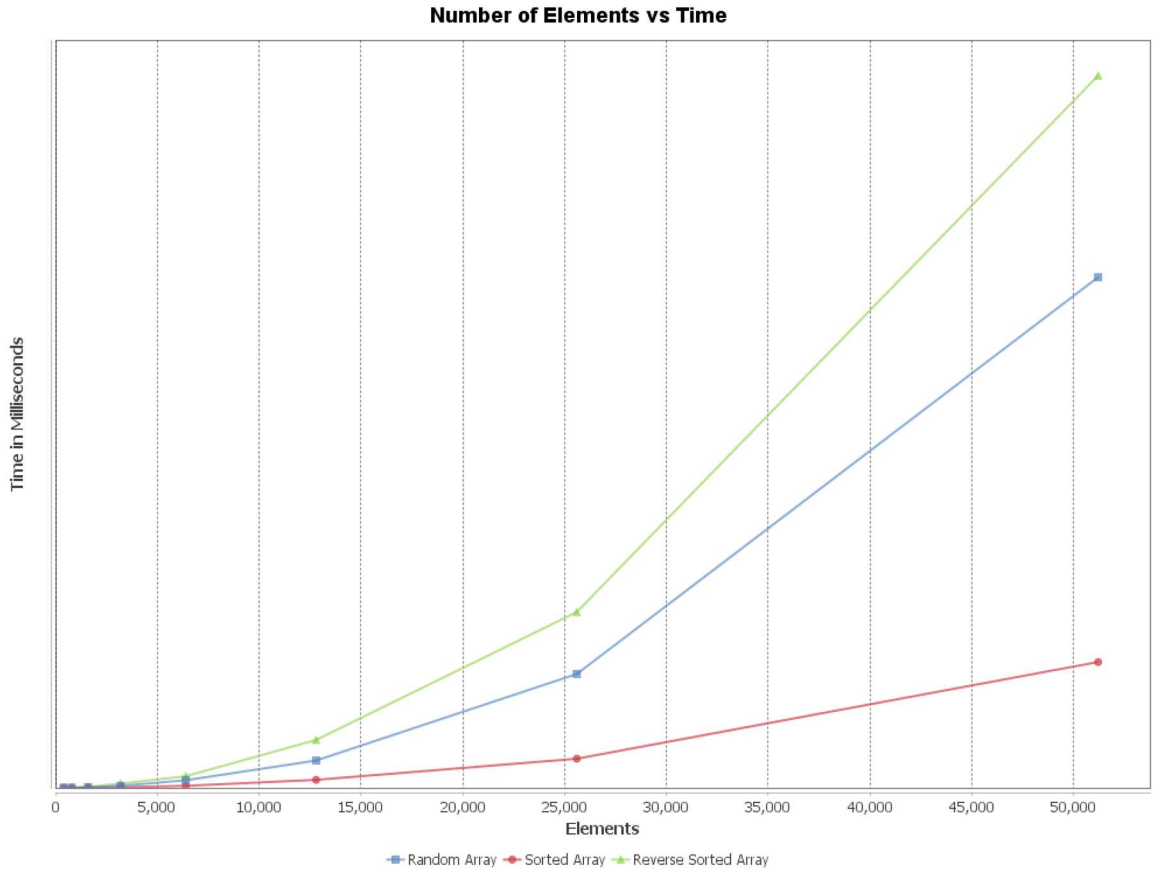

Figure 1: Time vs Number of Elements

To prove a quadratic relationship between the number of elements in the array being sorted and time taken by the insertion sort algorithm, I plotted a graph of the logarithm of time taken against the logarithm

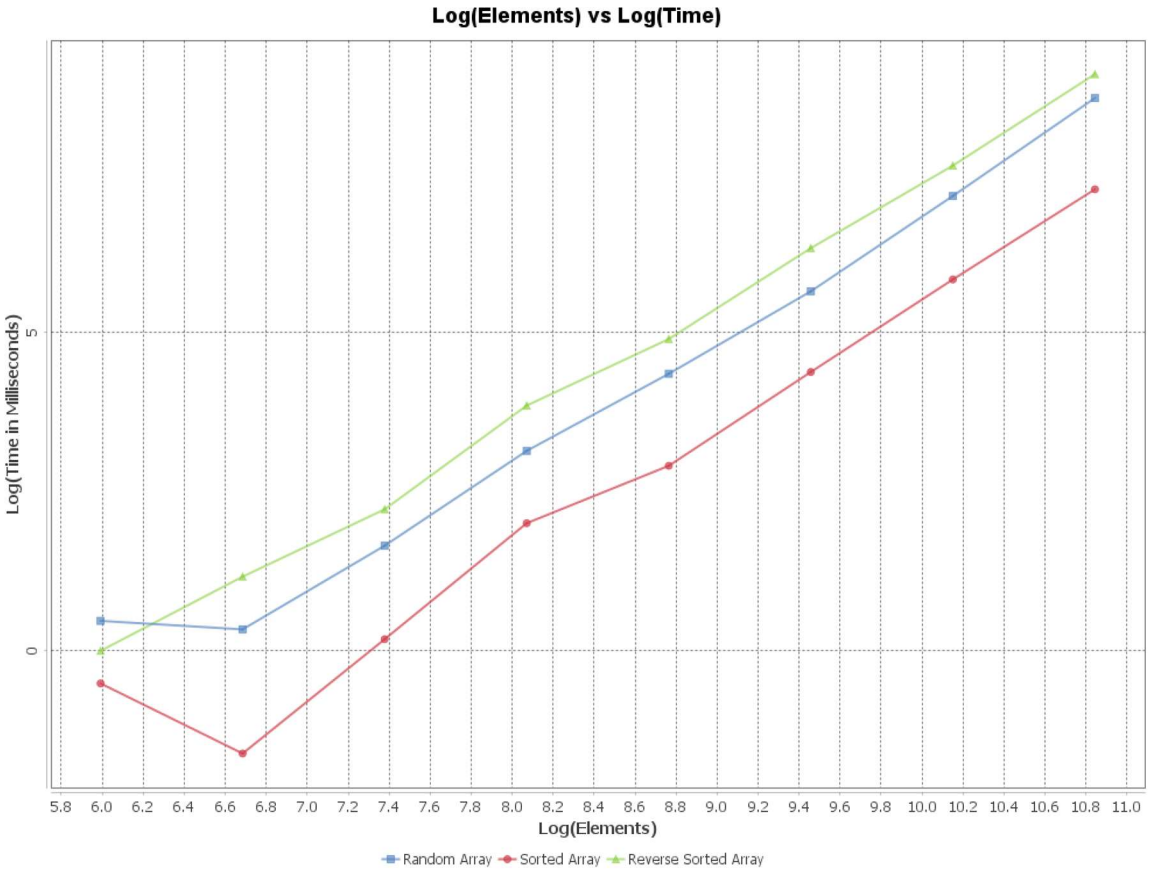

Figure 2: Log of Time vs Log of Elements

From the above graph, we can notice the graph following a straight-line trend, and hence we can state the time taken is a quadratic function of the number of elements. Hence proving the $O(n^2)$ average case time for insertion sort.

4. Screenshot of Unit-test passing

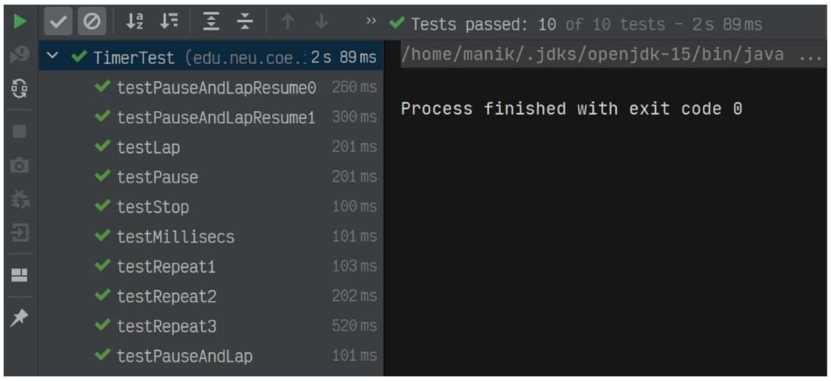

Figure 3: Screenshot of TimerTest passing.

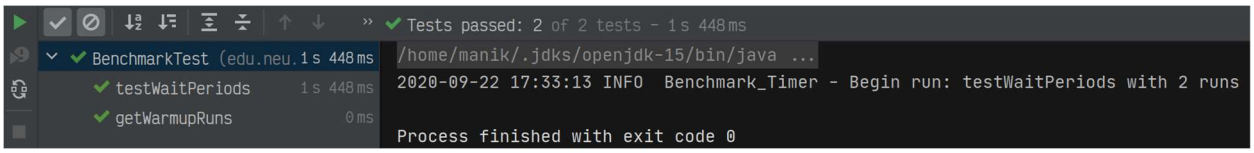

Figure 4: Screenshot of BenchmarkTest passing