Name: Hrishikesh Amulkumar Bodkhe

Enrolment No.: 2022CSM1006

Subject: CS506 – Data Structure (Lab Exercise 1)

System Specification: OS – Windows 11 64-bit, Processor - AMD Ryzen 5 5625U with Radeon Graphics 2.30 GHz, RAM – 8 GB, Execution Platform – Ubuntu Terminal with WSL.

Aim: To implement and analyse the time complexity of different comparison-based sorting algorithms.

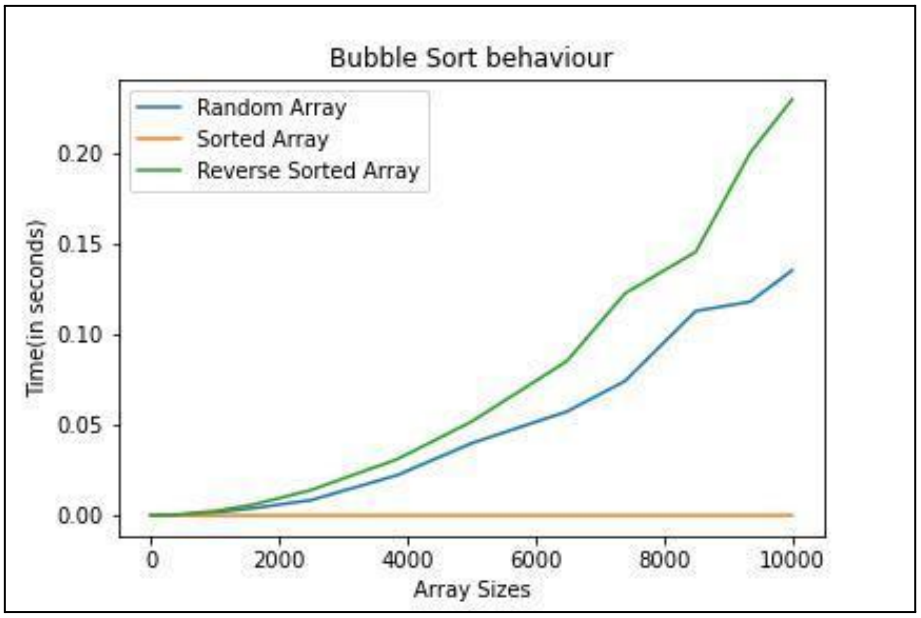
Programming Language: C

Observations:

Each of the sorting algorithms is tested on different types of arrays and the respective number of inputs and times are recorded. The time recorded is in seconds. For each type of array (random, sorted, reverse sorted), the same array of a particular size is passed to each of the sorting algorithms.

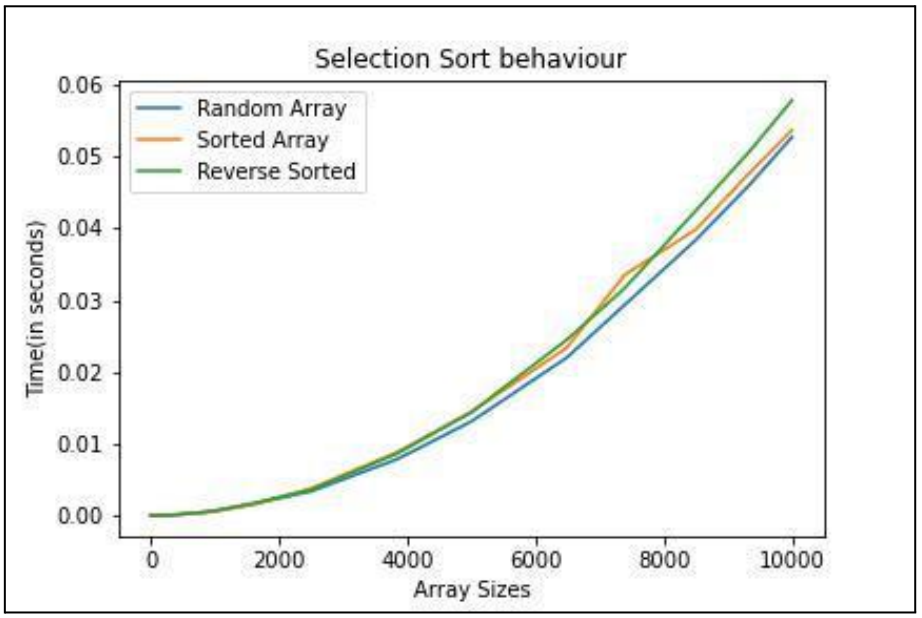
<u>Analysing Each Sorting Algorithm on random, sorted and reverse sorted arrays:</u>

1. **Bubble Sort:**



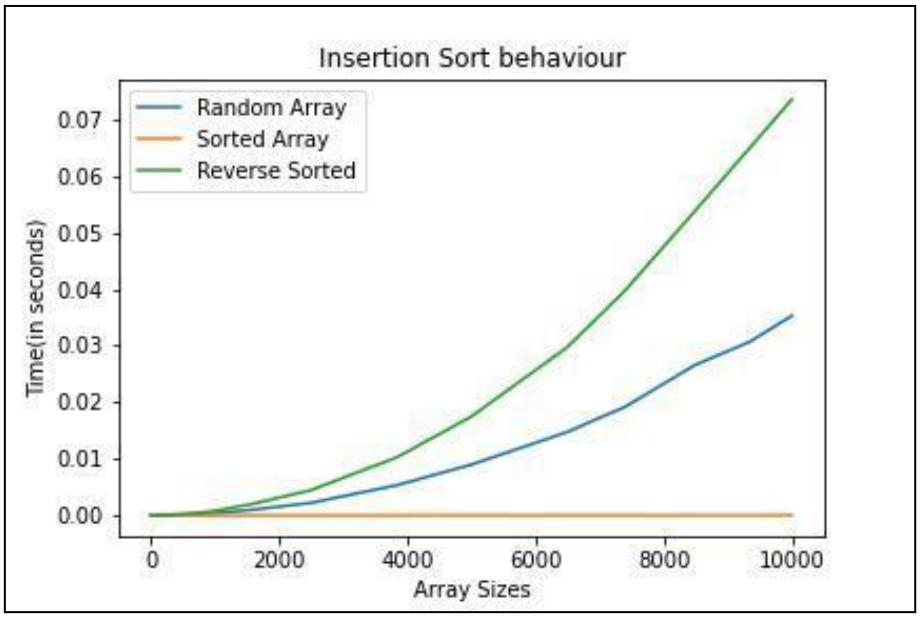
From the graph, it can be concluded that bubble sort is taking almost the same and least amount of time for a sorted array. And this is trivial, as per the algorithm, bubble sort can stop early if the array is already sorted. Now comparing with the random array and reverse sorted array, the latter is taking the highest time because the number of swaps is large as compared to random array, because it may be possible that the random array was partially sorted and thus this leads to a smaller number of swaps.

2. **Selection Sort:**



There is not much difference between the three cases in Selection sort. Because the best, average and worst-case time complexity of selection sort is $O(N^2)$. And, it doesn't care whether the array is sorted or not.

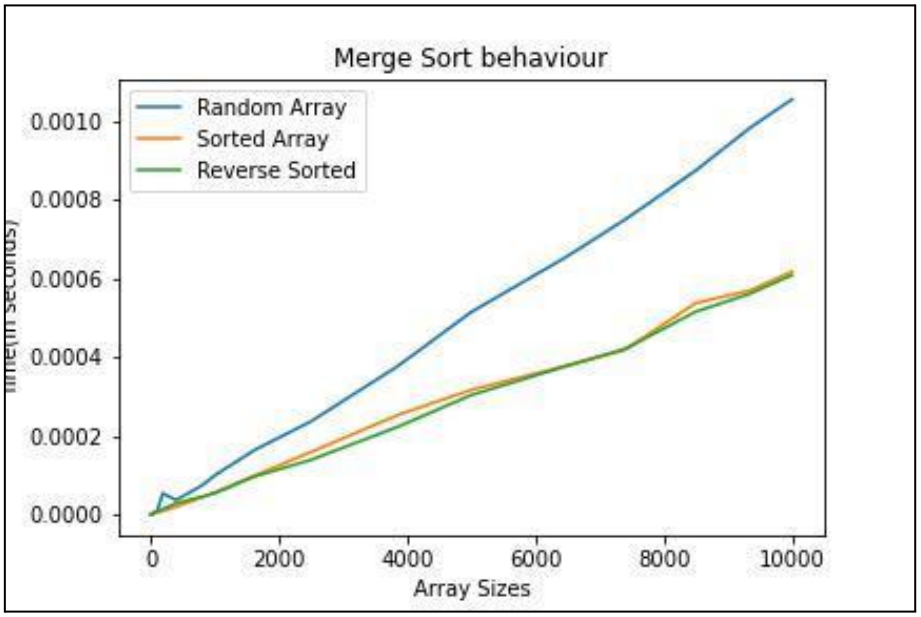
3. **Insertion Sort:**



Insertion sort performs best when the array is already sorted, with time complexity of O(N). Because in this case there will be no sliding of elements. Coming to the random array case, it may be possible that the array is partially sorted, and this will again lead to a few numbers of slides. Insertion sort will still perform with time complexity of $\theta(N^2)$ but the time taken

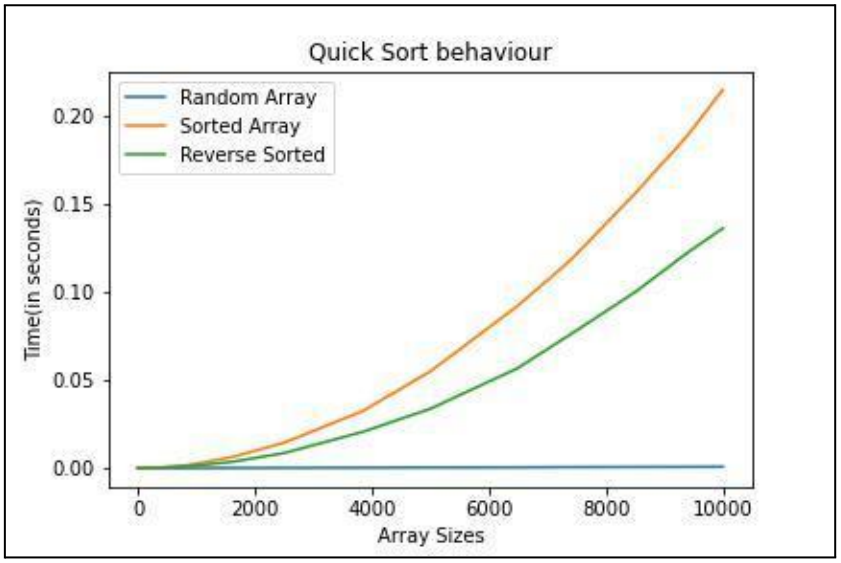
will be less as compared to the worst case, which is encountered when the array is reverse sorted. In the reverse sorted case, it is taking the largest amount of time because there will be $O(N^2)$ sliding of elements. Each element needs to slide if the key is less than every element to its left.

4. **Merge Sort:**



Theoretically speaking merge sort takes θ (NlogN) time in best, average and worst cases. And it doesn't care whether the array is sorted or not. That's why there is not much difference in the execution time. Graphically it is looking that for a random array it takes more time but if we see the values of time, it is too small.

5. **Quick Sort:**

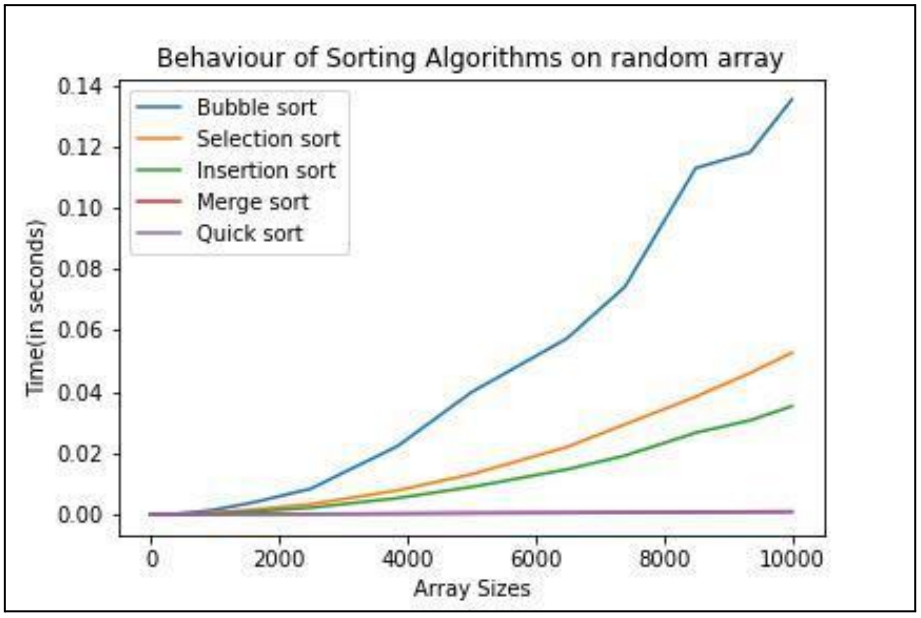


Since quicksort average time complexity is Θ(NlogN), which is followed in the random array case as it is taking the least amount of time and almost the same for all input sizes. And it performs worse when the array is already sorted, either in ascending or descending, thus can be seen in the graph.

Behaviour of Sorting Algorithms on a Random Array:

The following table shows the sample data of array sizes and the time taken by each sorting algorithm in seconds.

Array Size	Bubble Sort	Selection Sort	Insertion Sort	Merge Sort	Quick Sort
3	0.000001	0.000000	0.000000	0.000000	0.000000
5	0.000001	0.000000	0.000001	0.000000	0.000000
25	0.000003	0.000002	0.000001	0.000002	0.000002
50	0.000008	0.000005	0.000003	0.000006	0.000004
100	0.000021	0.000011	0.000005	0.000008	0.000006
190	0.000069	0.000041	0.000017	0.000054	0.000014
400	0.000271	0.000122	0.000067	0.000037	0.000025
799	0.001036	0.000429	0.000262	0.000074	0.000058
1020	0.001658	0.000680	0.000422	0.000102	0.000077
1620	0.004056	0.001652	0.001035	0.000164	0.000122
2500	0.008445	0.003377	0.002228	0.000237	0.000184
3850	0.022292	0.007877	0.005373	0.000377	0.000293
4999	0.039810	0.013096	0.008976	0.000515	0.000423
6489	0.057341	0.022001	0.014716	0.000656	0.000487
7395	0.074168	0.029370	0.019181	0.000750	0.000622
8500	0.112919	0.038373	0.026659	0.000875	0.000703
9349	0.118008	0.046167	0.030729	0.000984	0.000754
10000	0.135282	0.052701	0.035308	0.001056	0.000844



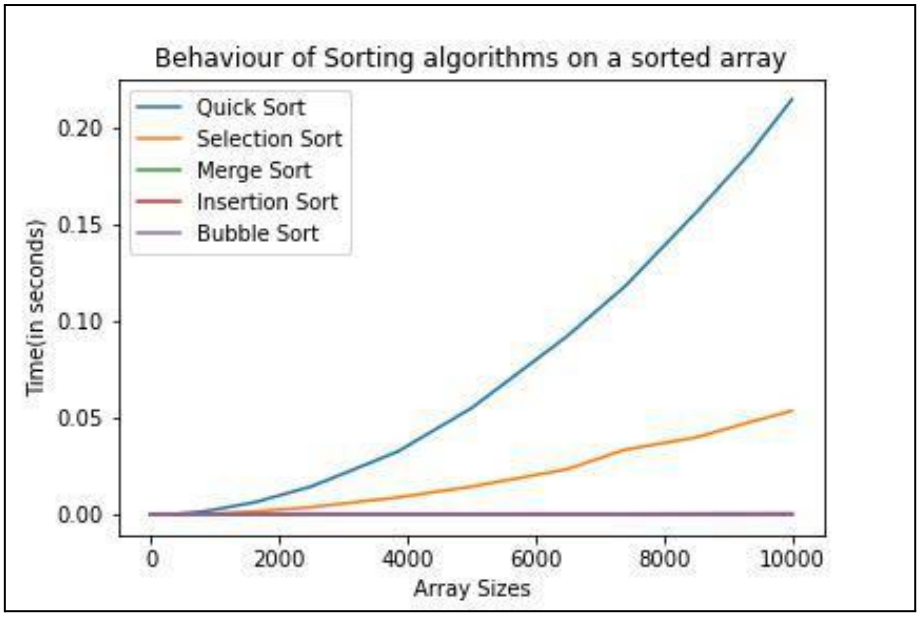
From the graph, we can conclude that for the same array, merge sort and quick sort are taking almost the same time and least among all. This is because on average the time complexity for merge and quick sort is O(NlogN). Now, Bubble sort (modified version), Selection sort and Insertion Sort - all have average time complexity of $O(N^2)$, but we can see that bubble sort is taking the largest time as compared to others. In bubble sort in each pass, we swap the element with the next element until each reaches its correct position. As this is the case the number of swaps in bubble sort is of order $O(N^2)$. But in selection sort, max swaps are of order O(N), due to this bubble sort takes more time as compared to selection sort. Now coming to the case of Insertion Sort is taking less amount of time than selection sort because if somehow the array becomes partially sorted or is already sorted, then insertion sort concludes early but this is not the case with selection sort. Selection sort doesn't care whether the array is sorted or not.

Behaviour of Sorting Algorithms on a Sorted Array:

Array Size	Bubble Sort	Selection Sort	Insertion Sort	Merge Sort	Quick Sort
3	0.000001	0.000000	0.000000	0.000000	0.000000
5	0.000001	0.000000	0.000000	0.000000	0.000000
25	0.000001	0.000001	0.000000	0.000001	0.000002
50	0.000001	0.000003	0.000000	0.000003	0.000009
100	0.000001	0.000007	0.000000	0.000006	0.000030
190	0.000001	0.000023	0.000000	0.000009	0.000096
400	0.000001	0.000126	0.000002	0.000020	0.000387
799	0.000001	0.000386	0.000002	0.000044	0.001484
1020	0.000003	0.000631	0.000003	0.000057	0.002709
1620	0.000003	0.001576	0.000005	0.000099	0.006328
2500	0.000005	0.003781	0.000006	0.000159	0.014437
3850	0.000006	0.008855	0.000009	0.000253	0.032509
4999	0.000007	0.014439	0.000011	0.000317	0.054908
6489	0.000009	0.023389	0.000013	0.000379	0.092029
7395	0.000011	0.033531	0.000017	0.000419	0.117887
8500	0.000012	0.039826	0.000017	0.000538	0.155871
9349	0.000014	0.047826	0.000020	0.000570	0.186879
10000	0.000014	0.053678	0.000021	0.000618	0.214582

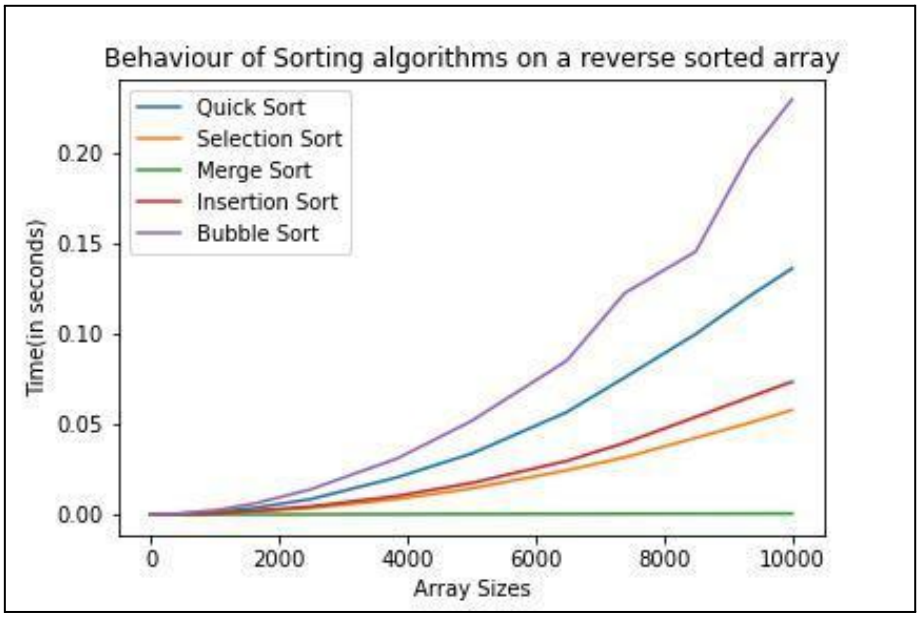
The above table shows the sample data of array sizes and the time taken by each sorting algorithm in seconds when the array is already sorted.

The graph constructed on the above data is shown below.



In this case, the graph shows that bubble sort, insertion sort and merge sort are taking almost the same amount of time and the least among the others. This is trivial because the best case (i.e., when the array is by default sorted) time complexity for each is O(N). And for quick sort default sorting is the worst case with time complexity of $O(N^2)$. As per my implementation, the last element is chosen as the pivot, so at each pass, we are traversing the array fully which is leading to an increase in time. Thus, quicksort is taking a huge amount of time as compared to others. Talking about selection sort, as stated it doesn't care about the sorted behaviour, but it is still taking less amount of time as compared to quick sort, as per my understanding this can be because of recursion calls of quicksort.

Behaviour of Sorting Algorithms on a Reverse Sorted Array:



Array Size	Bubble Sort	Selection Sort	Insertion Sort	Merge Sort	Quick Sort
3	0.000001	0.000000	0.000000	0.000000	0.000000
5	0.000001	0.000001	0.000000	0.000001	0.000001
25	0.000003	0.000002	0.000001	0.000002	0.000002
50	0.000007	0.000003	0.000003	0.000003	0.000005
100	0.000025	0.000008	0.000009	0.000007	0.000035
190	0.000120	0.000034	0.000047	0.000014	0.000088
400	0.000380	0.000151	0.000206	0.000029	0.000393
799	0.001820	0.000449	0.000515	0.000045	0.001048
1020	0.002417	0.000709	0.000839	0.000056	0.001560
1620	0.006107	0.001750	0.002115	0.000097	0.003678
2500	0.014002	0.003582	0.004465	0.000139	0.008549
3850	0.031124	0.008648	0.010327	0.000223	0.020677
4999	0.051703	0.014379	0.017459	0.000304	0.033670
6489	0.085173	0.024484	0.029667	0.000378	0.056678
7395	0.122564	0.031694	0.039692	0.000421	0.075891
8500	0.145543	0.042452	0.053982	0.000516	0.099872
9349	0.200263	0.050823	0.065057	0.000562	0.121127
10000	0.229716	0.057831	0.073526	0.000609	0.136105

From the above graph, we can clearly see that Merge sort takes the least amount of time among the others. This is because the worst-case time complexity of Merge sort is O(NlogN) and it doesn't care about the sorted or reverse sorted order of the array. Selection Sort and Insertion Sort nearly perform the same in this case. Insertion Sort will encounter the worst-case scenario here. At each iteration of insertion sort, the elements from the already sorted part will be shifted to the right by one place. Because of this scenario, insertion sort is taking some extra delay as compared to selection sort. Quick Sort again here will face its worst-case scenario. At each iteration of quick sort, the smallest element will be chosen as the pivot. Coming to bubble sort, as per my implementation, it is taking the largest amount of time. Bubble sort is also facing its worst case here. At each iteration of bubble sort, here it is only able to put the largest element at the end and there are a lot of swaps involved. Thus, this is the reason why quicksort is taking a huge amount of time.

Conclusion:

After analysing all the sorting algorithms, I found that Merge sort is taking the least amount of time as compared to others, for all three types of arrays. This is justified because the best case, average case and worst-case time complexity of Merge Sort is θ(NlogN). For sorted arrays, Bubble sort and Insertion sort perform equally likely to Merge Sort. But in general, if the array is in random order, then Quicksort also performs the same as Merge sort. And if we compare the time values of Merge Sort and Quick Sort in a random array case, then it is found that Quick Sort is faster. This may be the reason that Quick Sort is widely used in many built-in sorting implementations.