Assignment 1 – Analysis of Sort Algorithms

Student ID – 801053064
Name – Venkataramana Hegde

Theoretical Analysis

1. Selection Sort

   As the name indicates, selection sort algorithm sorts an array repeatedly by selecting an unsorted element and considering it as minimum element and comparing with other unsorted element. This process repeats until the whole set of numbers sorted. Selection sort is considered one of the simplest sorting solutions. The selection sort generates temporary arrays while performing the sort, one array which is being sorted and another array which yet to be sorted. Below table shows the time complexity for the Selection sort, we can notice that for all the cases, the time complexity is n^2.

Best case	Worst case	Average Case
$\Omega(n^2)$	$O(n^2)$	$\Theta(n^2)$

Table 1 - Time Complexity for Selection Sort

2. Insertion Sort

   Insertion sort is another simple sorting technique, we also can say it is the most intuitive sorting solution. This sorting algorithm works more efficient when the number of elements need to be sorted are small. However, as the size of element grows, insertion sorting become less efficient as it linearly scans each element in the set of inputs. If the input array is already sorted or almost sorted, then insertion sort comes handy to operate upon that.

Best case	Worst case	Average Case
$\Omega(n)$	$O(n^2)$	$\Theta(n^2)$

Table 2 - Time Complexity for Insertion Sort

   The time complexity for the Insertion sort also shows that it works well for sorted array compared to other cases unlike Selection sort.

3. Merge Sort

   Merge sort is little more complicated sorting algorithm compared to previous two sorting methods. However, as the number of input size grows merge sort become more efficient than selection an insertion sorting technique. Merge sort follows divide and conquer technique, it first divides an input array into two halves and sort the sub arrays and then merge the sorted array. So, the implementation of this sorting takes two parts, first sorting and then merging. Below table shows the time complexity of merge sort for best case (on, a sorted array), worst case (sorted in reverse order) and average case (a random array).

Best case	Worst case	Average Case
Ω(n log (n))	O(n log (n))	Θ(n log (n))

Table 3 - Time Complexity for Merge Sort

The time complexity for all the cases is n log (n), since it involves diving and merging irrespective of order of an input array.

4. Bubble Sort
   In terms of simplicity, the bubble sort algorithm falls in the same category as selection and insertion sorts. Bubble sort compares the adjacent elements and swaps them if the order is not proper. This process reaped until the whole array is swapped. From the below time complexity table, we can notice that it is huge for the average case and the worst case when compared to best case as the number of comparisons more in these two cases.

Best case	Worst case	Average Case
$\Omega(n)$	$O(n^2)$	$\Theta(n^2)$

Table 4 - Time Complexity for Bubble Sort

Below table provides a theoretical comparison of these four sorting algorithms based on the Time Complexity.

Sort Algorithm	Best Case	Worst Case	Average case
Selection Sort	$\Omega(n^2)$	$O(n^2)$	$\Theta(n^2)$
Insertion Sort	$\Omega(n)$	$O(n^2)$	$\Theta(n^2)$
Merge Sort	Ω(n log (n))	O(n log (n))	Θ(n log (n))
Bubble Sort	$\Omega(n)$	$O(n^2)$	$\Theta(n^2)$

Table 5 - Comparison of Sorting algorithms based on time complexity

Empirical Analysis

To conduct the empirical analysis of sort algorithms I've run the implementation of four algorithms by following the instructions specified in the assignment.

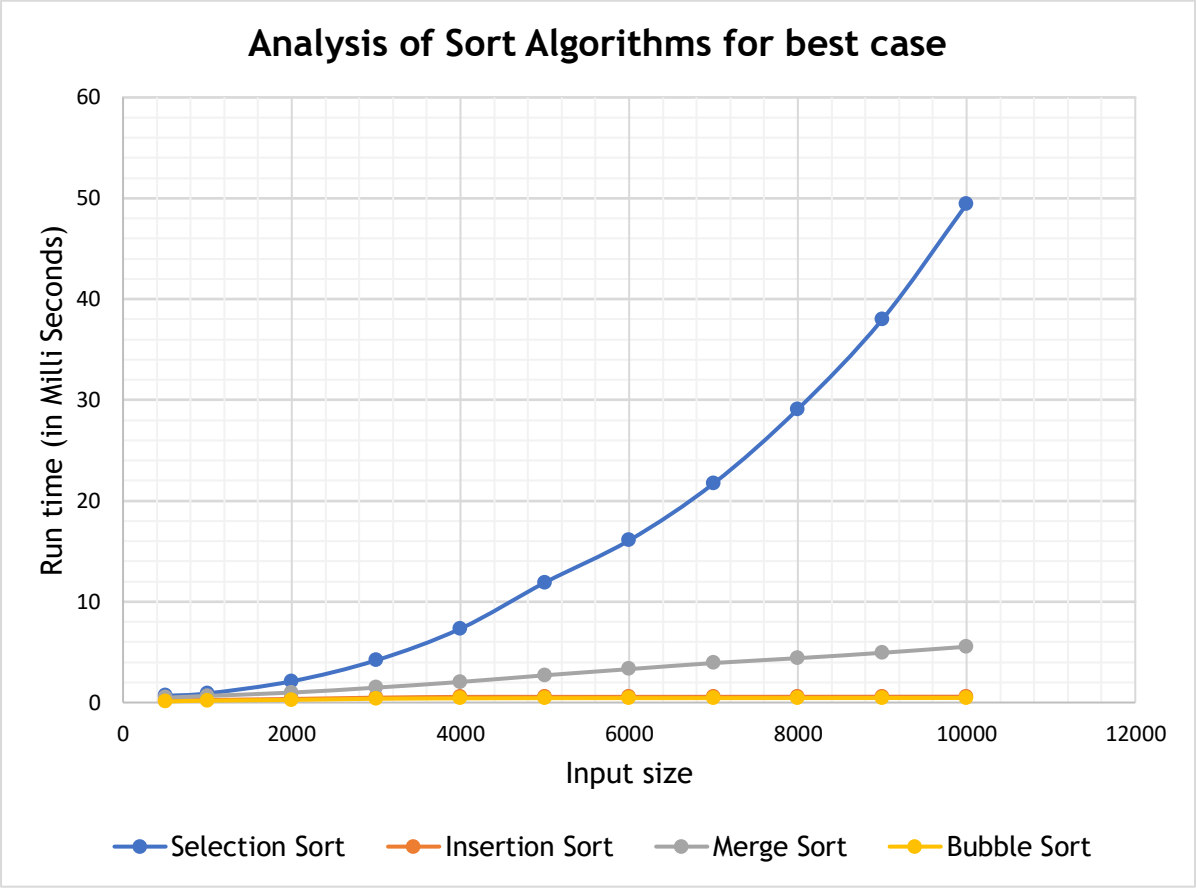
Implemented the Selection sort, Insertion sort, merge sort and bubble sort using Java programming language, and runtime are noted against various input size for Best case, average case and worst case. Up on obtaining the run time, a line graph is drawn to visualize the comparison of algorithms.

1. Best case
   For the best case, provided a sorted array of different size as an input to sort functions and run time is noted for each type for various size. Below table provides result of this experiment.

Analysis of Sort Algorithm for best case (Run time in Milli seconds)				
Input Size	Selection Sort	Insertion Sort	Merge Sort	Bubble Sort
500	0.665717	0.188056	0.4695	0.099683
1000	0.905537	0.255633	0.658188	0.18265
2000	2.107294	0.35379	0.98785	0.26116
3000	4.185246	0.482056	1.476392	0.370527
4000	7.324527	0.558879	2.041019	0.411094
5000	11.892759	0.56237	2.698955	0.431872
6000	16.079599	0.565102	3.311354	0.434525
7000	21.712555	0.568337	3.914725	0.437687
8000	29.093079	0.57211	4.397117	0.441403
9000	37.970976	0.575657	4.93275	0.444915
10000	49.407216	0.580798	5.535668	0.44981

Table 6 - Runtime of sort algorithms for different size (Best Case)

Below graph provides the pictorial representation of runtime laid out in the above table.



Graph 1 - Analysis of Sort Algorithms for Best case

- For the best case, that is when the input array is sorted, the two simple sort algorithms such as Bubble sort (Implemented the optimized bubble sort for the experiment) and insertion sort perform well, which is similar to what we seen in the theoretical analysis.

- Also, when the input size is small all the algorithms perform more less similarly.

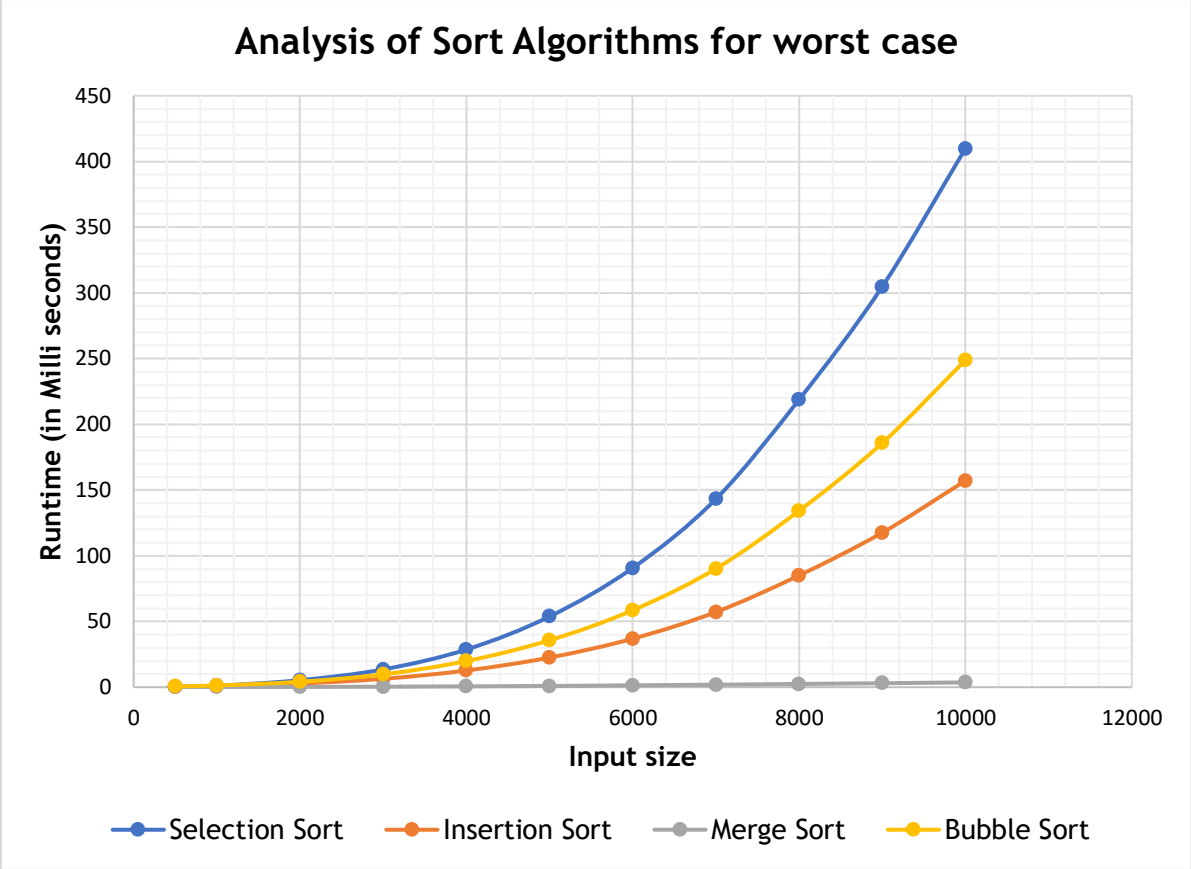
2. Worst Case

   For the worst-case analysis, provided reversely sorted array as an input to sort algorithms. Below table shows the result.

Analysis of Sort Algorithms for Worse Case (Time in Milli Second)				
Input Size	Selection Sort	Insertion Sort	Merge Sort	Bubble Sort
500	0.410664	0.375103	0.033945	0.671476
1000	1.123261	0.794299	0.091289	1.309539
2000	5.354594	2.811865	0.237027	4.153942
3000	13.549471	6.392122	0.413815	9.80351
4000	28.681944	12.715924	0.658309	19.89752
5000	53.814278	22.552636	0.969586	35.718289
6000	90.756073	36.921748	1.444294	58.578626
7000	143.21634	57.191847	1.916215	90.191819
8000	218.845319	85.049247	2.473745	134.232142
9000	304.792661	117.599307	3.031558	185.78092
10000	409.870388	157.148158	3.655582	248.815489

Table 7 - Runtime of sort algorithms for different size (Worst Case)

Below graph provides the pictorial representation of runtime from above table.



Graph 2 - Analysis of Sort Algorithms for Worst case

- Unlike in the best case, we see the simple sort algorithms such as Bubble sort, Insertion sort taking more time than Merge sort which supports the theoretical analysis presented in the first section.
- We could also see the huge difference in the runtime between Merge sort and other sorting techniques as the size grows. However, for the small input size, all algorithms perform more or less similar way.

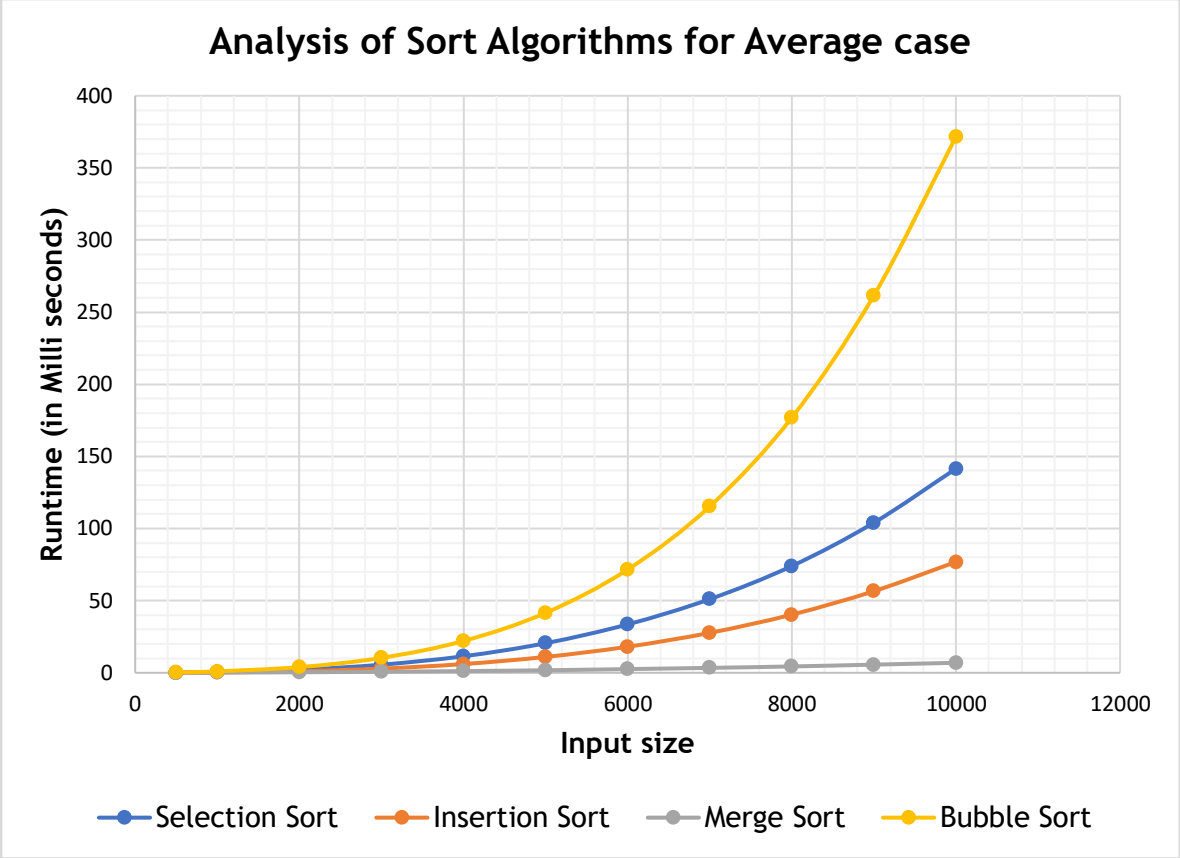
3. Average Case

For the average case, provided the random array of different sizes as input and obtained runtime as shown in the below table.

Analysis of Sort Algorithm for Average case				
Input Size	Selection Sort	Insertion Sort	Merge Sort	Bubble Sort
500	0.118423	0.055493	0.049214	0.232584
1000	0.523726	0.261723	0.151437	1.011873
2000	2.131164	1.094906	0.377455	3.910315
3000	5.629553	2.89632	0.723843	10.330127
4000	11.462751	6.068707	1.17879	22.060799
5000	20.515403	10.999711	1.75897	41.521953
6000	33.562246	18.06956	2.571637	71.551587
7000	51.040232	27.683	3.416172	115.278614
8000	73.992772	40.317205	4.399635	176.877428
9000	104.027268	56.579569	5.660984	261.53172
10000	141.487098	76.854432	6.937824	371.500346

Table 8 - Runtime of sort algorithms for different size (Average Case)

Graphical representation of above data is represented by the line graph shown below.



Graph 3 - Analysis of Sort Algorithms for Average case

- This graph looks much similar to the worst-case analysis graph. However, the selection sort overcoming the Bubble sort.
- The merge sort remains the best as the input size grows.

Conclusion

From the theoretical analysis and empirical analysis of four sort algorithms (Selection sort, Insertion Sort, Merge Sort and Bubble sort) we could make couple of important observations –

- If the input size is small, all algorithms perform more similar.
- For the large set of inputs, merge sort stands far ahead then other sorting techniques.
- If the input array is already sorted / most of it are sorted, then simple technique such as bubble sort, insertion sort gives better performance over more complicated merge sort.
- The empirical analysis supports the theoretical analysis of these algorithms.