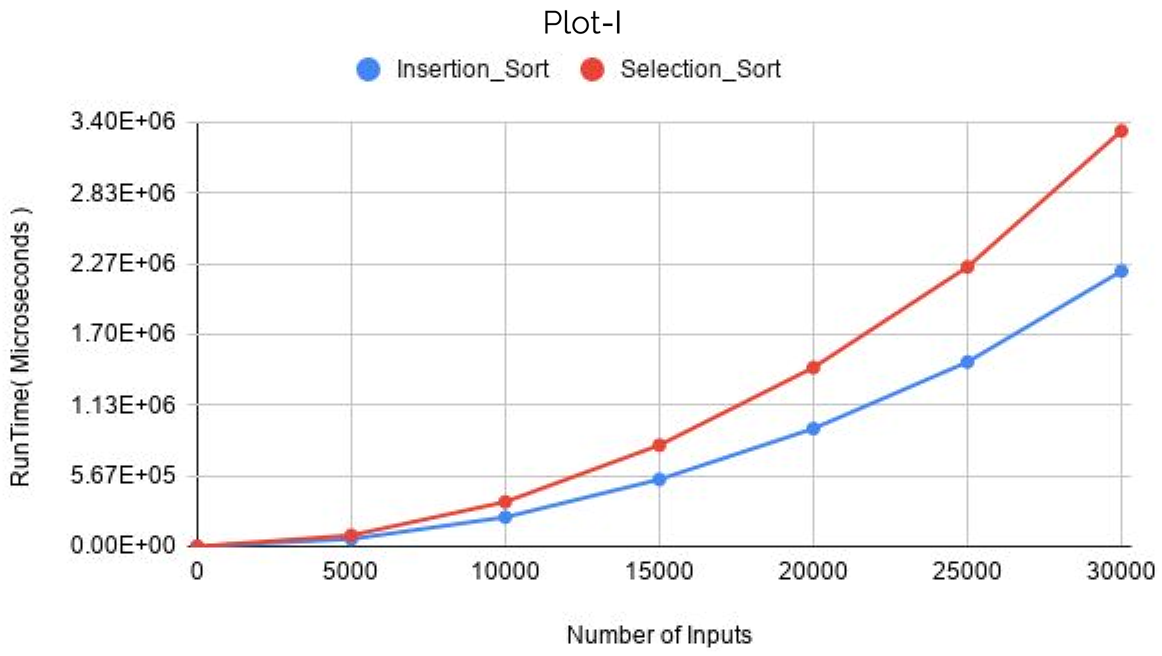
Submitted By - Sanyam Rajpal

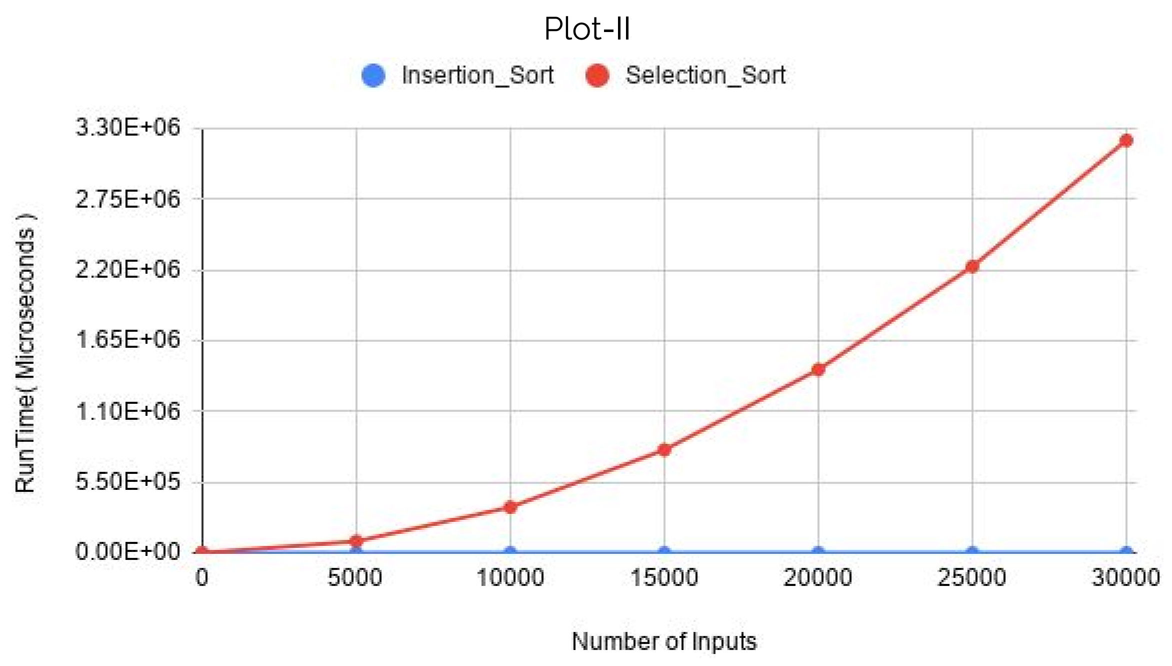
CSCI-B-505 Applied Algorithms
(FA19 - BL - 11503)
Assignment - I
WRITE-UP

I used CPP language for my code because it has faster runtime and one has to implement the algorithm from scratch helping you learn the basics thoroughly. I printed out my code in the terminal and saved them in the Google Sheets in the form of a table which I also used for plotting the graphs. I made some typical syntax errors, but not a runtime error. The observations I made have been explained below.



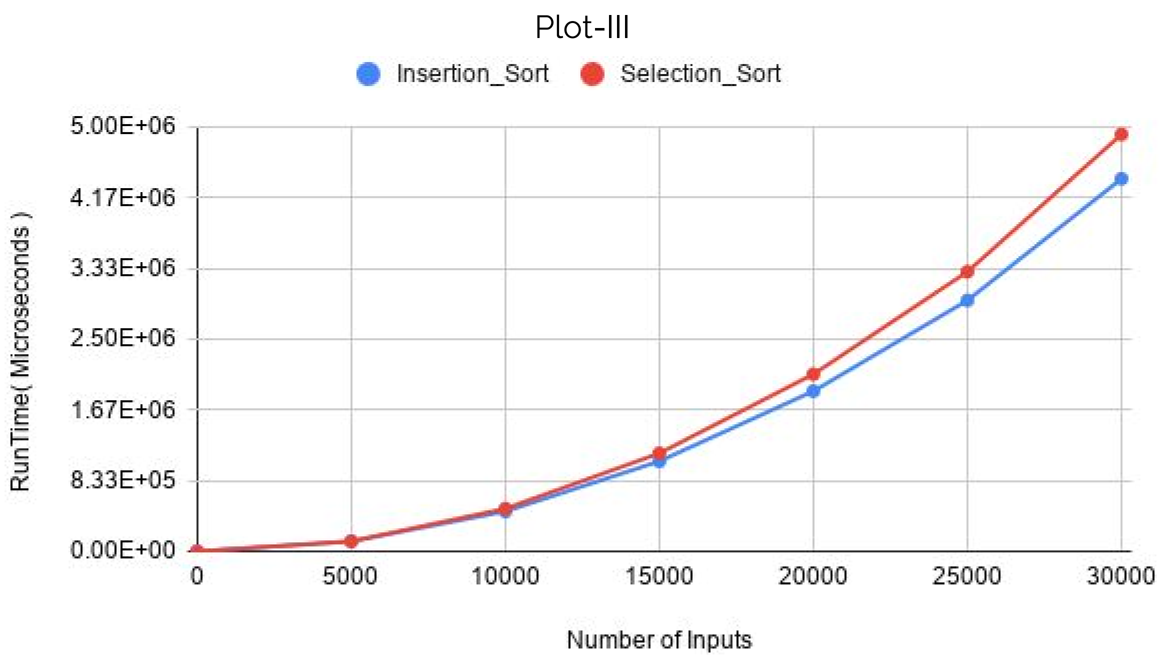
Inputs Runtime(μs)	Insertion Sort(μs)	Selection Sort(μs)
5000	59514	89481
10000	234942	357448
15000	537281	811998
20000	944317	1.43354e+06
25000	1.47812e+06	2.23957e+06
30000	2.20674e+06	3.32999e+06

This is the plot and table of average of 3 runs for Insertion and Selection sort with a random input array. As can be inferred from the plot, insertion sort performs significantly well then selection sort when the order of input is increased. For data of small order, the difference is insignificant. Both the algorithms are of order 2 with respect to the number of inputs and the plots show the same trend(quadratic). Selection sort performs slightly better for Plot 4 than Plot 2 for this case, but it will be considered insignificant.



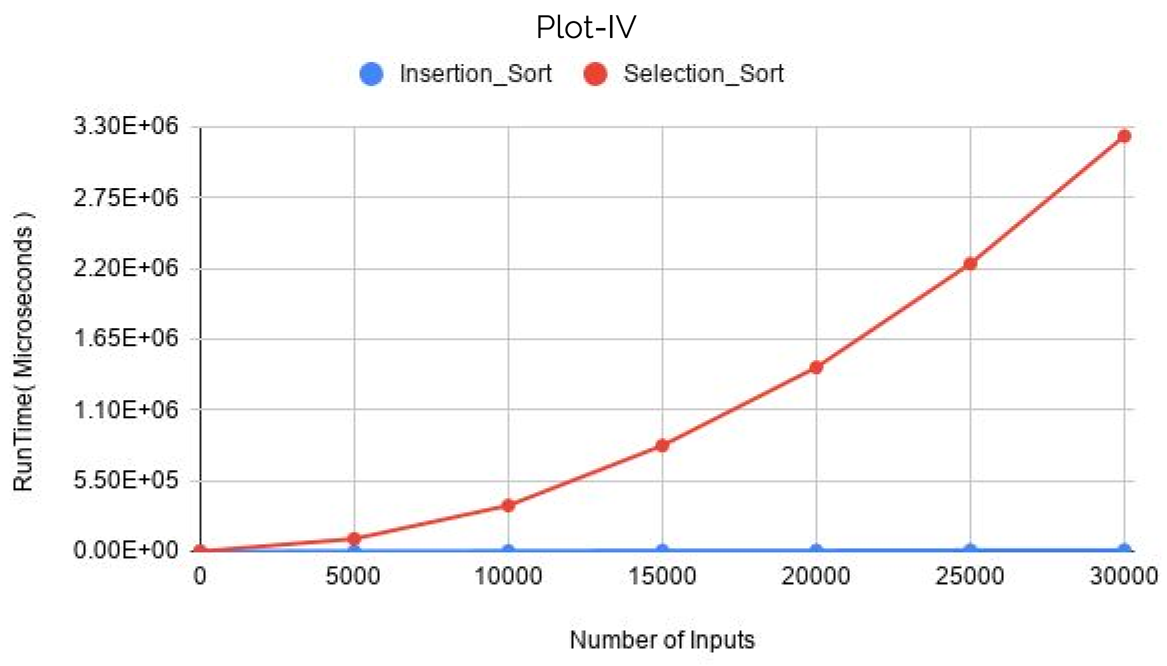
Inputs Runtime(μs)	Insertion Sort(μs)	Selection Sort(μs)
5000	2097	97876
10000	3432	356435
15000	6378	822266
20000	6670	1.43007e+06
25000	9236	2.23551e+06
30000	10806	3.22784e+06

This is the plot and table for Insertion and Selection sort with a non-decreasing array as input. As can be inferred from the plot, insertion sort performs exceptionally well than selection sort when the input is already sorted(non-decreasing) for all orders of data. Insertion sort's time complexity is O(n) for a sorted array, but for selection sort, it is still O(n^2). And trying to plot an O(n) and O(n^2) curve on a single graph compresses the O(n) to a line parallel to y-axis because a change in that is insignificant compared to that O(n^2) and it shows that decrement of time complexity by even 1 is very significant.



Inputs Runtime(µs)	Insertion Sort(µs)	Selection Sort(µs)
5000	118330	119252
10000	470847	503240
15000	1.06136e+06	1.15557e+06
20000	1.88681e+06	2.08678e+06
25000	2.95567e+06	3.29331e+06
30000	4.38782e+06	4.90961e+06

This is the plot and table for Insertion and Selection sort with a non-increasing array as input. As can be inferred from the plot, selection sort performs comparably to the insertion sort when the array is in non-increasing order, for all orders of data (Insertion is slightly better than selection). The reason for a slight difference could be because Insertion Sort doesn't swap keys which have the same values. For Insertion sort, the worst case is the reverse ordered array(current case) for which it is of O(n^2). Selection sort performs O(n^2) for this case as well, so the two algorithms have almost the same performance. And as you can see the x-axis range(0 - 5e+06) is way larger for this graph than any other plots(0 - 3.3e+06) so both insertion and selection algorithms perform their worst when the array is reverse sorted.



Inputs Runtime(μs)	Insertion Sort(μs)	Selection Sort(μs)
5000	2097	97876
10000	3432	356435
15000	6378	822266
20000	6670	1.43007e+06
25000	9236	2.23551e+06
30000	10806	3.22784e+06

This is the plot and table of average of 3 runs for Insertion and Selection sort with a non-decreasing array as input with 50 swaps between random indexes. As can be inferred from the plot, insertion sort performs exceptionally well than selection sort when the input is almost sorted(non-decreasing array with an insignificant margin of random index values swapped(50)) making it almost the same as plot 2. Selection sort performs slightly better for Plot 4 than Plot 1 for this case, but it will be considered insignificant.

Conclusion

For Input-5. We had 100,000 input values in the range [1 , 50]. Since 50 << 100,000 there will be plenty of repeated values. The runtime for Insertion sort is **2.32701e+07** and the runtime for Selection sort is **3.57066e+07**. Insertion sort sorted the array in almost 2/3rd the time of selection sort which again tells us that insertion sort performs fairly better than selection for any random input. The insertion sort doesn't change the order of elements with equal keys. Again justifying the significant difference in the runtime of the two.

Insertion Sort is good for small datasets. Since it performs better for already sorted data we can say that the algorithm doesn't always give $O(n^2)$ complexity. It doesn't require any extra space so its space complexity is O(1). As can be inferred from the algorithm, it doesn't change the order of elements with equal values.

Selection sort performs $O(n^2)$ operations for any kind of data(including sorted data). It doesn't require any extra space so its space complexity is O(1).

So from the properties stated above Insertion Sort seems a better choice than Selection Sort. And since insertion sort, sorts an already sorted array(or an almost sorted array-like Plot 4, almost O(n) time complexity) faster than $O(n^2)$ time complexity, we can use Insertion Sort in that case. In cases where space is of importance and we can compromise on time complexity for minimal space usage, we can use Insertion Sort or Selection Sort.