CSCI B505 Programming assignment 1

Gopi Kiran(gkiran@iu.edu)

September 26, 2019

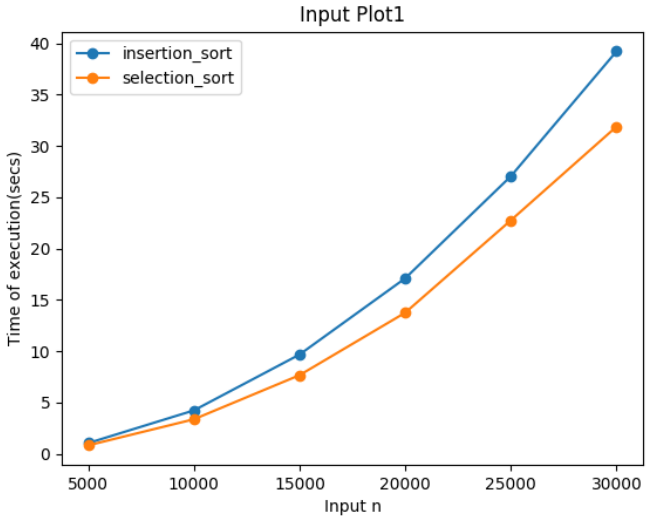
1 Explaining choices

- I chose Python language and UNIX platform because plotting graph is easier in python as compared to other languages that was mentioned. I used matplotlib module. Although I could have used python only for plotting graph by reading the output file, I chose to keep all the programs files related to this assignment in the same language (generation of input, sorting algorithm implementation, reading the output file and plotting graph).
- I created the input data in the file based on the requirements specified in the assignment questionnaire. The actual algorithm implementation program reads the input file and stores the necessary data acquired after sorting (time, sorting technique, sorted array) in the output file in the form of JSON. The third program reads the output file and plots the graph.
- For the input 1, I was getting quadratic time for selection sort and linear time for insertion sort. This was because of the programming mistake where instead of passing the copy of the input array, I was passing the input array. Hence, insertion sort executed on the already sorted array. Moreover, this paved way for reason on why insertion sort takes linear time on the sorted elements while the selection sort takes quadratic time. If it were to be the other way around, I might not have realized my mistake.

2 Conclusion

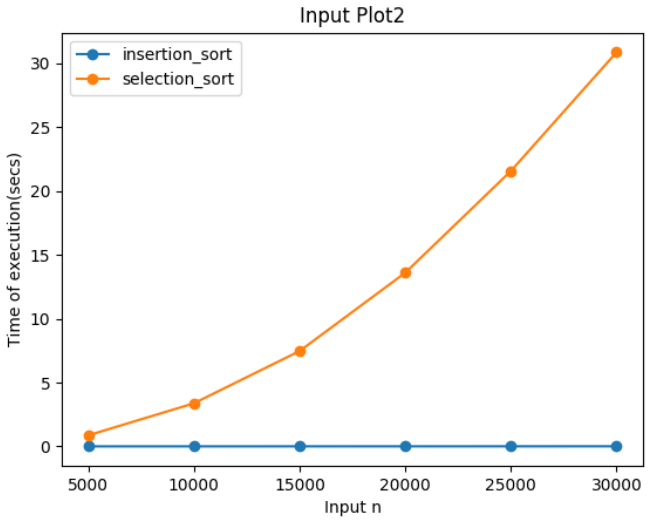
- Even though both the algorithms have running time ($O(n^2)$) in the input plot 1, selection sort performs better than insertion sort. Hence, for randomly distributed n, selection sort is better than the insertion sort. Additionally, for to the input 1, the nature of numbers chosen were completely random without any specified rules

Figure 1: Input Plot 1



- In the second plot since the array is already sorted, insertion sort takes linear time ($O(n)$) and selection sort takes ($O(n^2)$). This would be the best case scenario for insertion sort which takes linear time. Insertion sort works on the principle of placing the element in it's right position on every iteration. Hence, when the array is sorted insertion sort takes only n traversal without bothering to place the chosen element in it's right position.

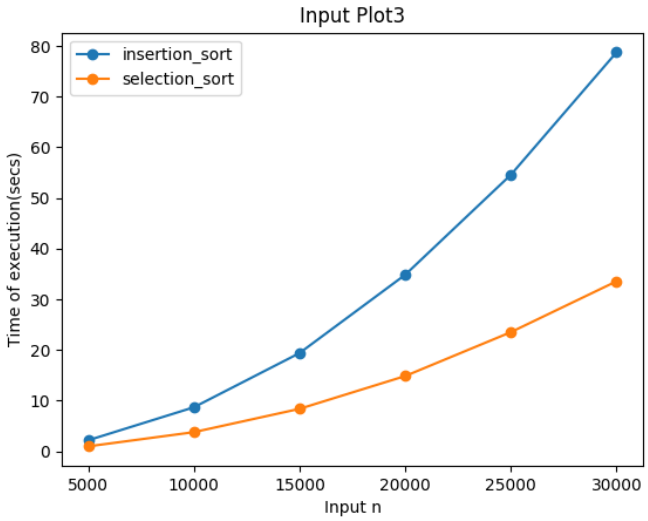
Figure 2: Input Plot 2



- In case of third plot the arrays are sorted in the reverse order and it is the worst possible case for insertion sort as (n^2) numbers need to be moved in order to insert

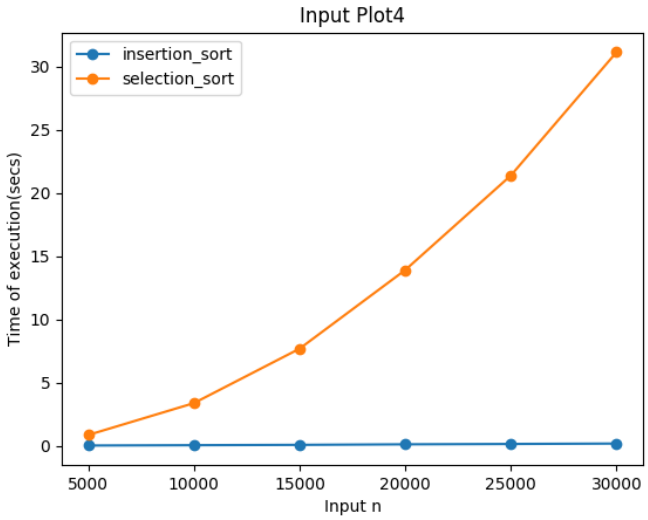
the element into it's right place. Selection sort always has quadratic time complexity irrespective of the type of arrangement of input number.

Figure 3: Input Plot 3



- In the fourth plot, the numbers are partially unsorted. Hence, we see negligible increase in the execution time for large numbers in case of insertion sort when compared to the second plot where numbers are fully sorted. In case of selection sort, this has negligible effect as it's execution time is quadratic anyway.

Figure 4: Input Plot 4



- In the fifth input type, insertion sort outperforms selection sort for the large input (100000) with mostly repeated numbers (rand range(50))

- I would use neither of these algorithms on the real world data as the average execution time of both of these algorithms are quadratic. I would prefer Merge Sort or Quick Sort on real world data as both of them have average time complexity of $\theta(n \log n)$.