CSCI B505 Programming assignment 2

Gopi Kiran(gkiran@iu.edu)

September 27, 2019

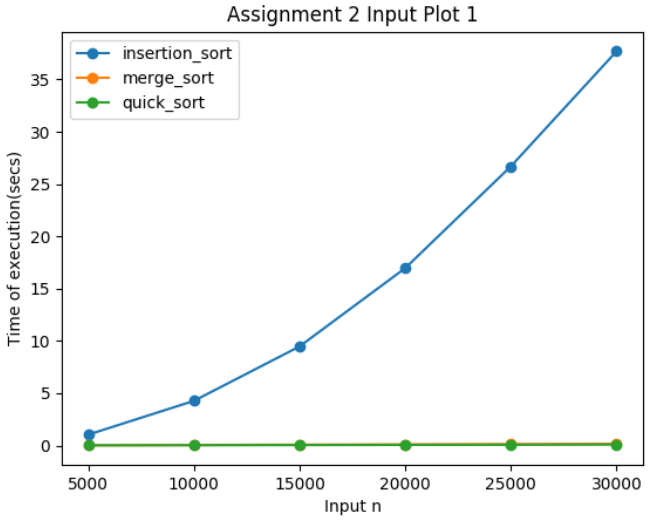
1 Explaining choices

- Please find the code for the assignment 2 in this link
- 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.

2 Conclusion

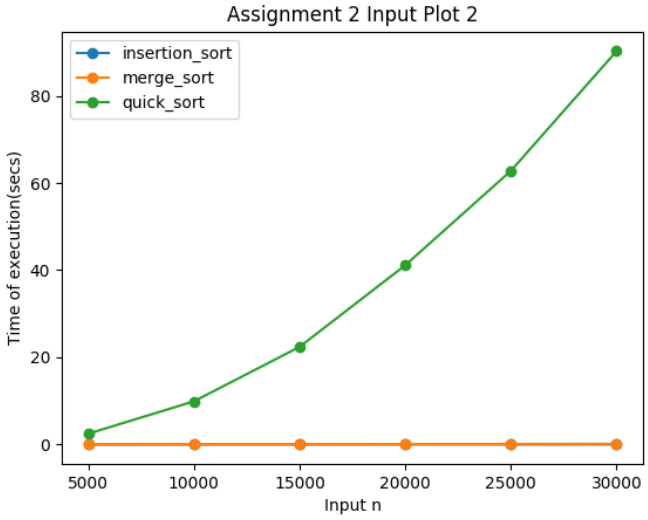
- In case of input plot 1, Insertion sort have the running time $O(n^2)$ and the other 2 sorts i.e., merge sort and deterministic quick sort have the running time $O(n \log n)$. Even though, both merge sort and deterministic quick sort have the same complexity deterministic quick sort performs better than the merge sort as merge sort takes $O(n)$ space too.

Figure 1: Input Plot 1



- In case of input plot 2, the array is already sorted and hence the deterministic quick sort takes $O(n^2)$ which is the worst case for deterministic quick sort. Merge sort takes $O(n \log n)$ which would be case for merge sort irrespective of whether the array is sorted or not. Insertion sort takes $O(n)$ because of the fact that array is already sorted and all the elements are in their respective position.

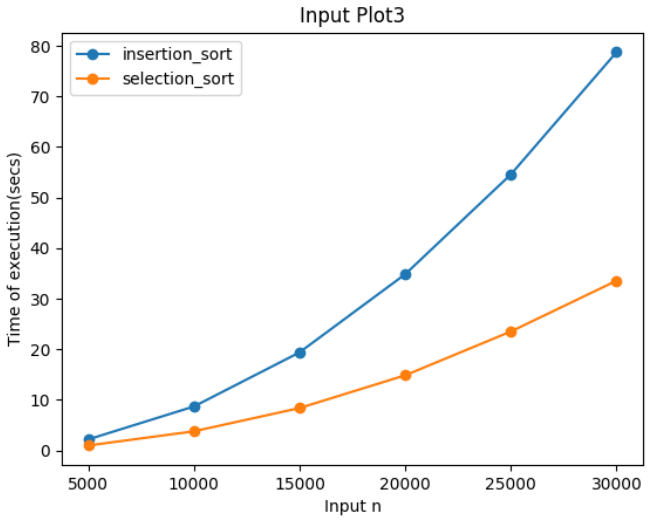
Figure 2: Input Plot 2



- In case of input plot 3, the arrays are sorted in the reverse order and it is the worst possible case for insertion sort as $O(n^2)$ numbers need to be moved in order to insert the element into it's right place. Merge sort takes $O(n \log n)$ irrespective of whether the array is sorted in any order whatsoever. Deterministic quick sort takes $O(n^2)$ as it

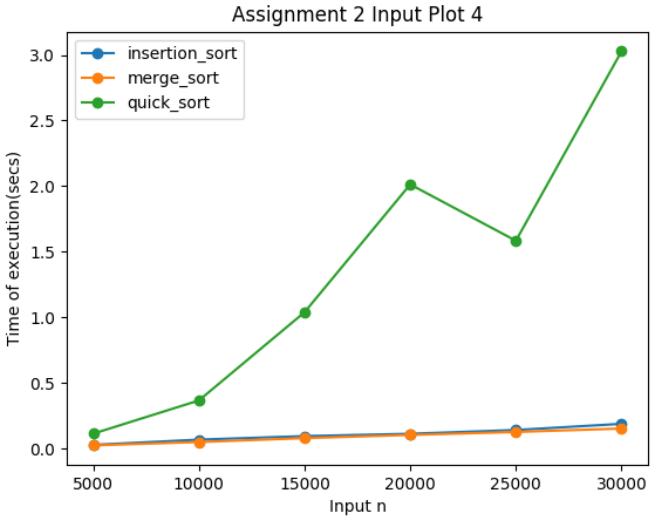
is sorted. The worst case for deterministic quick sort is when the array is sorted either in ascending or descending order.

Figure 3: Input Plot 3



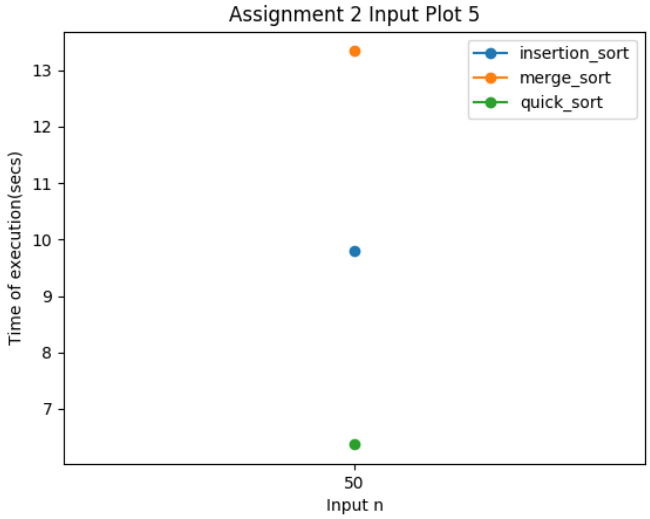
- In the input plot 4, the numbers are partially unsorted. Hence, we see negligible increase in the execution time for large numbers in case of insertion sort which of when compared to the second plot where numbers are fully sorted. The complexity of insertion sort is still $O(n^2)$ but, since the array is partially unsorted where the unsorted array size is $\leq 1\%$ the complexity is nearly $O(n)$. In case of merge sort the complexity is $O(n \log n)$. In case of deterministic quick sort, the complexity is technically $O(n \log n)$ as the array is not sorted completely. However, in comparison with the insertion sort and merge sort it is still quadratic. The time taken to sort the partially unsorted array is about 97% lesser than the time taken to sort the already sorted array.

Figure 4: Input Plot 4



- In case of input plot 5, deterministic quick sort outperforms insertion sort and merge sort. Interesting case here is that, even though merge sort has the complexity of $O(n \log n)$ and insertion sort has the complexity of $O(n^2)$ insertion sort outperforms the merge sort. Moreover, quick sort takes 50% of the time than the time taken by merge sort even though, both have the same complexity. Hence, the merge sort is not preferred even though it has $O(n \log n)$ complexity. Total running time of quick sort = 6.367117825000005, merge sort = 13.339636953000191, insertion sort : 9.800650559998942

Figure 5: Input Plot 5



- I would use the combination of quick sort and insertion sort in the real world. I would

never use merge sort as it requires auxiliary array. In the standard library of C++, hybrid sort (quick + insertion) is used.