Programming Assignment - 2

Randomized Quick Sort vs Insertion Sort

Plots:

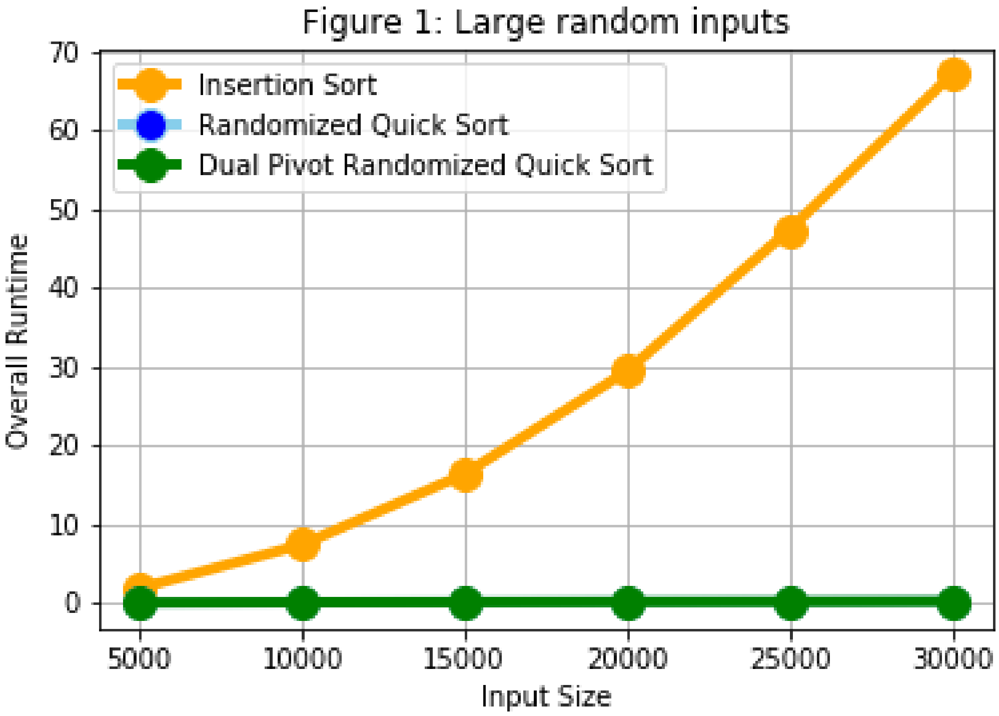


Figure 2: Non-decreasing inputs

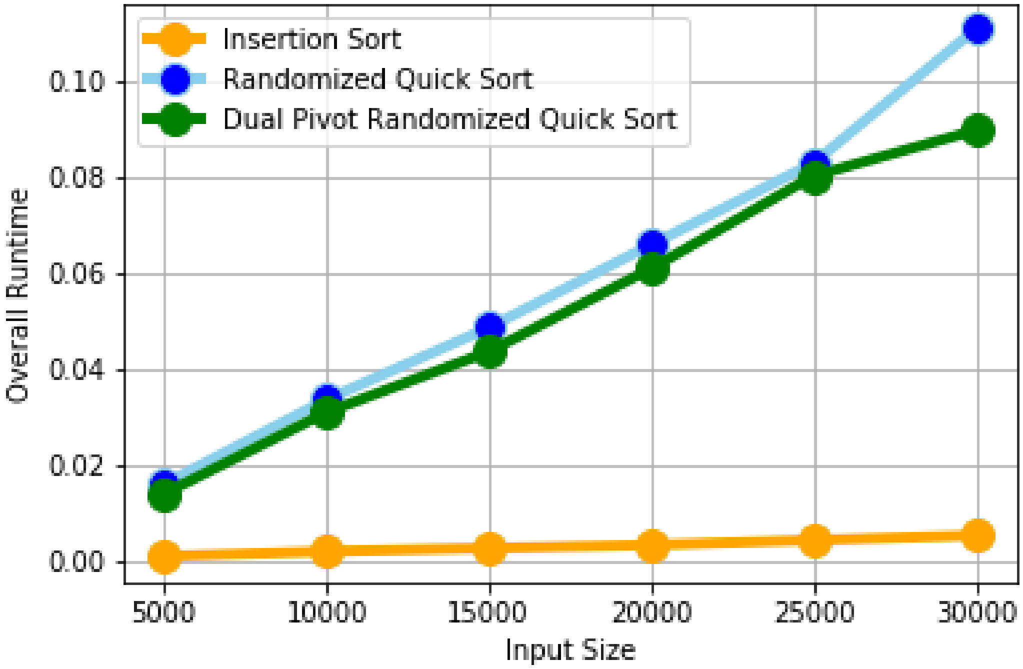


Figure 3: Non-increasing inputs

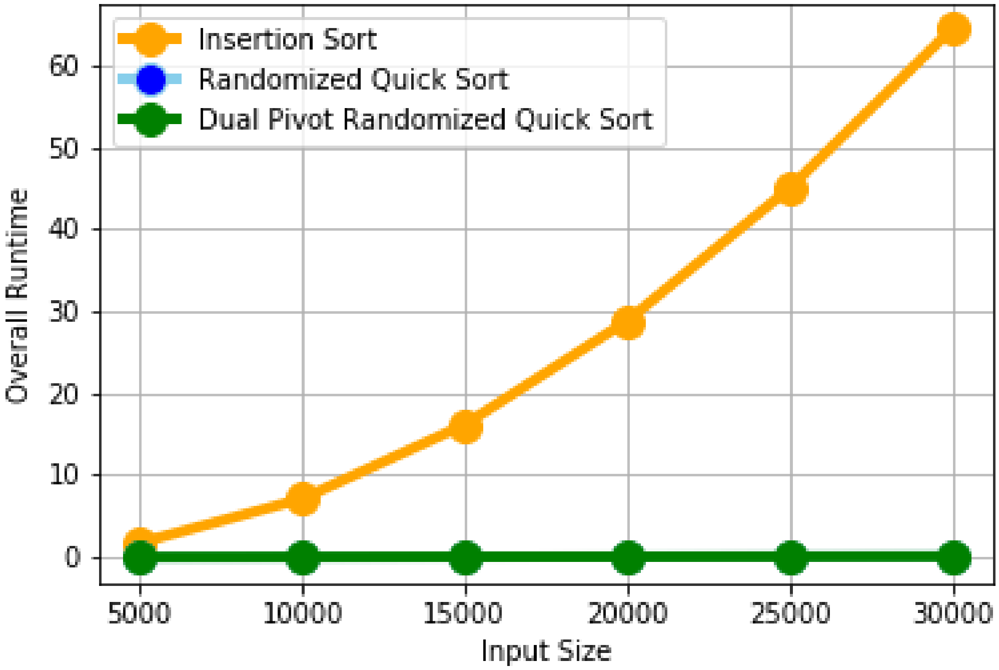


Figure 4: Noisy non-decreasing inputs

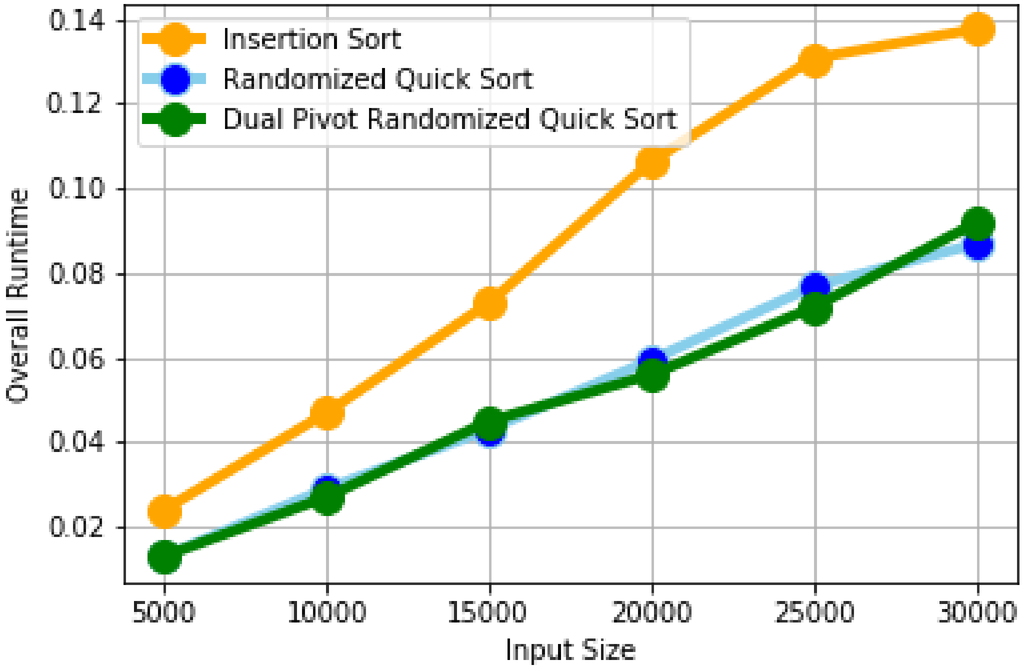
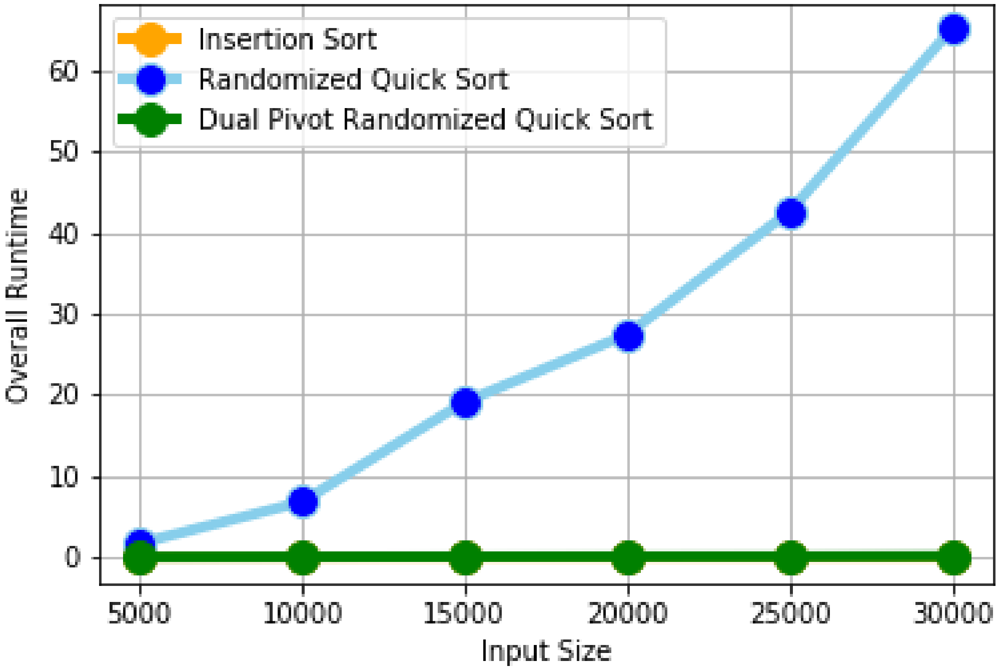


Figure 5: Constant-value input



Input 6 overall runtimes:

Insertion Sort: 17.84 seconds

Randomized Quick Sort: 14.02 seconds

Dual-Pivot Randomized Quick Sort: 17.90

Explain your choices:

For this assignment, I have used python for programming. I have generated random inputs using python NumPy library. Also, saved and plotted graphs using matplotlib library. To record time, I used python time library. For storing the data for plots, I created three dictionaries, one for each sorting technique, whose key is input type and values are list of runtimes for each size.

Conclusions:

1) Does The last two algorithms have asymptotic running time of O(n log n). Does the first plot reflect this? How do the three algorithms compare in terms of the running time?

Yes, they have asymptotic running time of O(n log n). Yes, the first plot reflects this. Insertion sort takes very long time as compared to two other sorts since it has time complexity of O(n*n).

2) How about the second plot? Which algorithm do you think is the best one here?

Since array is already sorted, insertion sort runs very fast because of no swaps. Hence insertion sort is better. Quick sort also performs well but in comparison to insertion, it’s not better. Quick sort takes longer time because, we find pivot and swaps it with last element before finding partition index.

3) How does the third plot compare to the first and second?

Insertion sort takes very long time than other two since the array is decreasing, hence insertion sort does huge number of swaps. First plot almost resembles third one. Whereas, insertion sort performed better in second plot and not the third plot.

4) What about the fourth plot? What kind of functions do you think you’re observing in the three plots (linear, logarithmic, quadratic, exponential, etc.)?

For fourth, all algorithms perform in almost the same time. Also, this plot resembles linear function.

5) Can you explain behavior for plot 5? Can you suggest a way to avoid such behavior which is sufficiently general?

For fifth plot, insertion does no swapping at all hence, time needed is almost zero. Similarly, for dual pivot, since all elements are same and arrays are recursively divided into 3 partitions, it is faster and there are no unnecessary swaps. But for randomized quick sort, we are checking less than or equal to condition in partition function, hence it is performing unnecessary swaps. One way to avoid is rethinking all the checking conditions.

6) Which algorithm does perform better for input 6? Why?

Randomized quick sort takes the least time for input 6. But all the algorithms have almost similar running time with difference of few seconds. For smaller input, insertion sort does lots of checks and swaps, hence, takes longer time. Also, we are using divide and conquer for quick sort algorithms, hence it is expected to perform better than insertion sort.

7) Would you use these algorithms for real data and if so why?

Yes, I would. I would prefer randomized quick sort since it has worst running time of O(n log n) which is better than insertion sort with O(n* n) worst time complexity. For the larger data, its better to use dual pivot. Overall, quick sort is better and its being widely used as built-in sort function for most of the languages. If we know the array is going to be almost sorted, insertion sort would be a better choice but then again in real life we will not know what kind of data we will be working on.