Archana Rajaseharan

Assignment 4 Results

Random data:

Overall the results that I have garnered from sorting random data are consistent with how it should be. The average time taken does vary slightly when compared to the output that I was given as reference. In that, my output took a little longer but since there are other external factors that could influence the time taken, I would not necessarily classify that as incorrect output. For my algorithms specifically, the output matched what he had learned this term, exclusively how QuickSort has a faster actual runtime than MergeSort. For my outputs, I got SimplePivot as my best setup three time while MedOfThree was my best setup the other two times. All my worst setups were in fact MergeSort. MergeSort which has a Big O runtime of O(N lg N) in the average and worst case would intuitively seem like the better algorithm when compared to QuickSort which has a worst case runtime of $O(N^2)$ and best case runtime of O(N lg N). Despite this, QuickSort outperforms MergeSort in terms of speed because QuickSort is done “in place” (does not require additional storage of data) while MergeSort requires an extra temporary array to merge the sorted arrays after the recursive calls are made. Copying from the temporary array to the original array requires more time than if the sort was directly done onto a single array. This process increases the overhead significantly which has an impact on the overall actual runtime, but not the asymptotic runtime. Therefore, QuickSort is faster and more preferred when sorting through primitive types in arrays.

Sorted data:

My results for the presorted data remained consistent with what was expected. In all my outputs, MergeSort end up as the best setup while Simple Pivot QuickSort was the worst setup. When comparing the QuickSort sorts amongst each other, I noticed that the MedOfThree was the best QuickSort. In fact, the average time of MedOfThree was closer in value to MergeSort than SimplePivot. The reason why SimplePivot QuickSort is coming out as the worst setup has much to do with how the data is divided in respect to the pivot. Since the data is already sorted, Simple Pivot QuickSort is facing its worst case of which it will produce an asymptotic runtime of $O(N^2)$. This is because, the pivot is always going to be chosen as an extreme element in the partition. Thus, only a single recursive call of N-1 rather than two recursive calls of size N-2 are spawned. This defeats the purpose of a divide and conquer algorithm because the recursive calls are only decrementing by 1 each time. Therefore, the SimplePivot QuickSort algorithm is at a disadvantage when it has to sort already sorted data. MedianOfThree QuickSort significantly outperforms SimplePivot because the pivot chosen in already sorted data will always be the middle element of the array. Therefore, it is a clean and efficient divide for the partition which has a favorable outcome in terms of actual runtime. MergeSort works best if the data is already sorted because it does not have to do many comparisons and therefore reduces its overhead. Overall, MergseSort works best for an already sorted array whereas SimplePivot QuickSort will give you the worst case.