Pau Balcells Sanchez

Results Reflection

After doing multiple tests and seeing the final results that I got for each configuration, overall, there is a clear winner that performed better in all 9 different configurations, MergeSort, and a clear looser that had the worst time also for all of the different configurations, Simple Pivot QuickSort. To be honest, I was a little shocked to see that Simple Pivot QuickSort performed better than MergeSort in the random configuration, seeing how in the example files that were provided for assignment 4, the roles were reversed, and Simple Pivot QuickSort was the most efficient Sorting Algorithm compared with MergeSort that was the slowest. This would make more sense since both MergeSort and all the different types of QuickSort have a likely case of O(Nln(N)) but MergeSort has a lot more overhead since it needs multiple arrays to complete the algorithm. However, consistently with all the different array sizes, MergeSort outperformed Simple Pivot QuickSort by a factor of 10 or more. On the other hand, with the data already sorted, the results went as expected and MergeSort consistently compiled 10^4 or more times faster than Simple Pivot QuickSort. This result makes complete sense since for Simple Pivot QuickSort the worst possible run-time of O(N^2) occurs when the data is already sorted, whereas MergeSort has a consistent run-time of O(Nln(N)).

In terms of the different configuration depending on the sizes, when looking at the results with the already sorted data, both for the sizes 10000 and 20000, the worst run-time happens with Simple Pivot QuickSort when the minimum size is 5. Theoretically this setting is the one that makes the most sense since it reduces the run-time from O(N^2) to O(N) only when the minimum size is 5, instead of reducing it at 75 which is the maximum minimum size that we could choose. However, interestingly, for the sorted arrays with sizes 40000 and 80000, the worst-case scenarios that I got were when the minimum size was 75. I would have expected the worst case for both of the outputs to be with a minimum size of 5 as it happened with the ones before, so this result makes me think that there might be something in my code that is not efficiently implemented and could be affecting my results. As for the configurations where the data was not already presorted, the worst-case scenarios consistently happened when the minimum size was 75.

In conclusion, from the results that I got, it seems that the best way to sort any array weather its data is already sorted or not is to use MergeSort. In all my results with the different array sizes I got a wide variety of minimum sizes for the best case, so I do not think that the Insertion Sort at the end has a big impact on the run-time. To be honest, these results that I have gotten shock me a bit since they are not consistent with what we have talked in class. In theory, since QuickSort has a lot less overhead and both QuickSort and MergeSort have the same expected run-time, except in the case of already sorted data, QuickSort should be the most efficient algorithm. However, if I had to choose an algorithm based on my results, I would have to go with MergeSort.