Matthew Gutkin

Dr. Ramirez

CS0445 Assignment 4

13 April 2020

Based on the program's conclusions, I can conclude that there is no such thing as an efficient algorithm at scale, just a most efficient one for a specific purpose. According to my data, it suggests that the best configuration for sorted data was MergeSort with a low minimum recursive number. MergeSort only swaps numbers if they are not in order, but because the data is already sorted, there is very little actual work that needs to be done. This saved time because there was less work that needed to be done, which made this configuration more efficient. The worst case tended to be Random Pivot QuickSort with a high minimum recursive number. This is most likely due to the fact that the algorithm has to split the array the maximum amount of times and do the maximum amount of comparisons before it is clear that the data is already sorted. The results for random data were almost the opposite, with the worst-case scenario being MergeSort, with a low recursive number, and the best case was Simple Pivot QuickSort. For MergeSort with random data, the algorithm would split the array numerous times, after putting it back together as well, to make more comparisons because there is no guarantee that after the first pass the data is sorted. This was not entirely efficient because it had to go back multiple times to sort the data. The best case for random data would be Simple Pivot QuickSort because there is a high likelihood that the first pivot chosen is a good median point for the array, making it significantly more likely for the algorithm to do less comparisons in the end.