As we increased the number of files the program had to sort I saw that the time did not suffer as poorly as the number of programs increased. Our program suffered more when it came to surfing through directories because every directory would have to search, sort, and join while the sort function would complete. For 2 files and 4 directories we witnessed times of about 6 seconds and then doubling the number of files only lead to about a 1-2 second increase. However, when we increased the number of directories I saw a much higher jump.

Sometimes comparing runtimes is not fair because runtime can differ between machines with different specifications. However, when it comes to stable machine coding environments with limited factors, it would be more accurate so long as the tester would average the results. Discrepancies can occur when the programs are user based or the parameters differ in size. It would be faster to sort a 2 tuple csv than a 500 tuple csv.

Depending on data sizes, it is possible to make slower programs faster by implementing more efficient sorting algorithms. If there are no differences however, it would be more difficult if the code if already optimized.

Mergesort is good depending on the task at hand. Quicksort could get a job done faster but that all depends on the parameters of the program. Mergesort is a safe choice for multithreading because with an average time of O(n log n) it is not the best choice, however it is sufficient.