Is the comparison between run times a fair one? Why or why not?

Yes, the comparison between run times in this case is a fair one. This is because depending on the file size, the thread algorithm that we developed in assignment 2 may be slower. In general, threads are created much faster, as a whole process does not need to be duplicated and created, but rather data is transferred within the same process and stack. However, in assignment 2 we were instructed to print the output to one sorted CSV file. Since the thread sorter prints to one file, the file printing cannot run in parallel like the processes do. So, with large files, the time taken to print to files makes the thread sorter slower, as the work is not split.

What are some reasons for the discrepancies of the times or for lack of discrepancies?

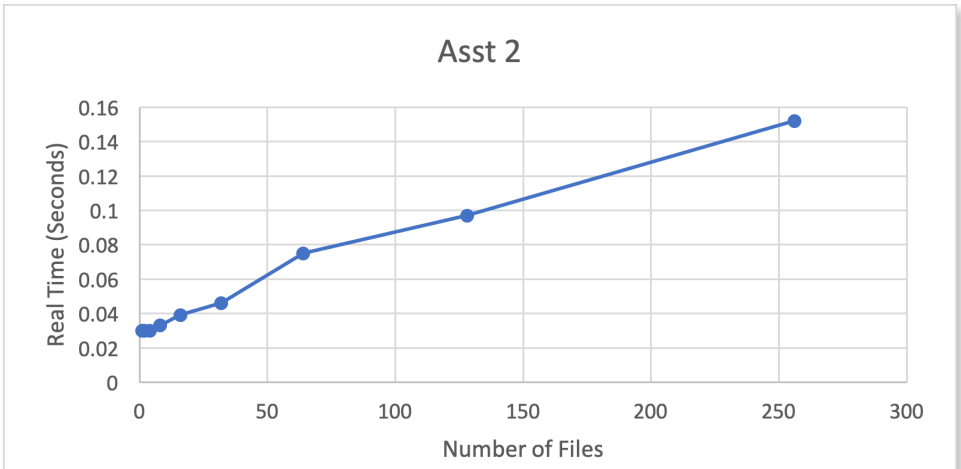
While the multithread sorter adds to the end of one linked list, the processes can all run simultaneously. This is the main reason why assignment 1 seems to be running faster than assignment 2, as you can see from the data/graphs below. However, with smaller data files, the time taken to print does not overcome the difference in time to spawn processes vs threads, so Asst2 remains much faster.

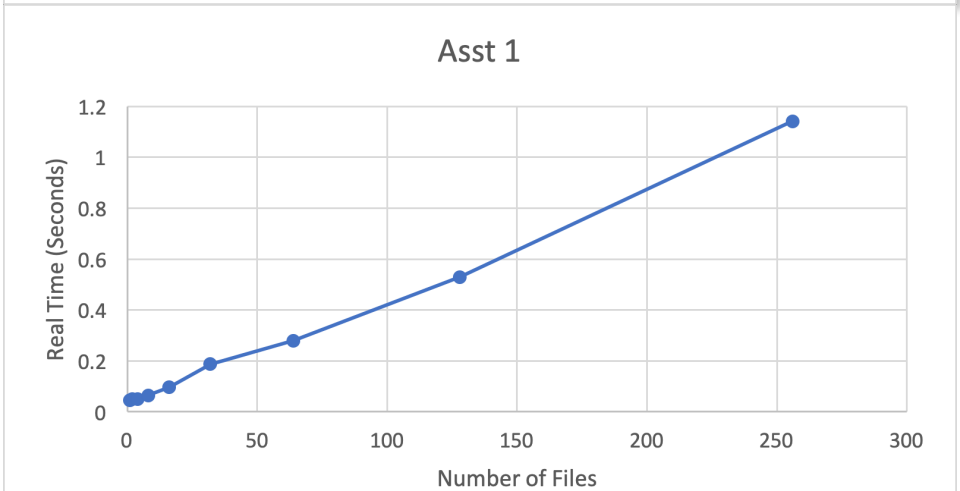
If there are differences, is it possible to make the slower one faster? How? If there were no differences, is it possible to make one faster than the other? How?

The multithread process always ran faster with small files compared to the assignment 1 project where we used processes. Threads are always faster if they are printed into their own files. In our case, we had to print all of the sorted data to a single CSV file, therefore there is no way that we could make it faster than the processes used in assignment 1. However, if in Asst2 we printed to individual files, we figure that the threads would then be faster for all cases. Also, we could make our thread process faster by using a faster sorting algorithm on the global linked list, as each files portion will already be sorter. However, we did not implement this. How we would do this is in the next question.

Is mergesort the right option for a multithreaded sorting program? Why or why not?

Yes, if sorted to their respective files. If all data is printed to a global sorted file, it would probably be quicker to sort each file individually, and then do a merge of each files' data at the very end. In this case, we would act like the recursive part of merge sort is done, and just do the recombination at the end, as each file's section is sorted and there would be no need to recurse past the file portions.



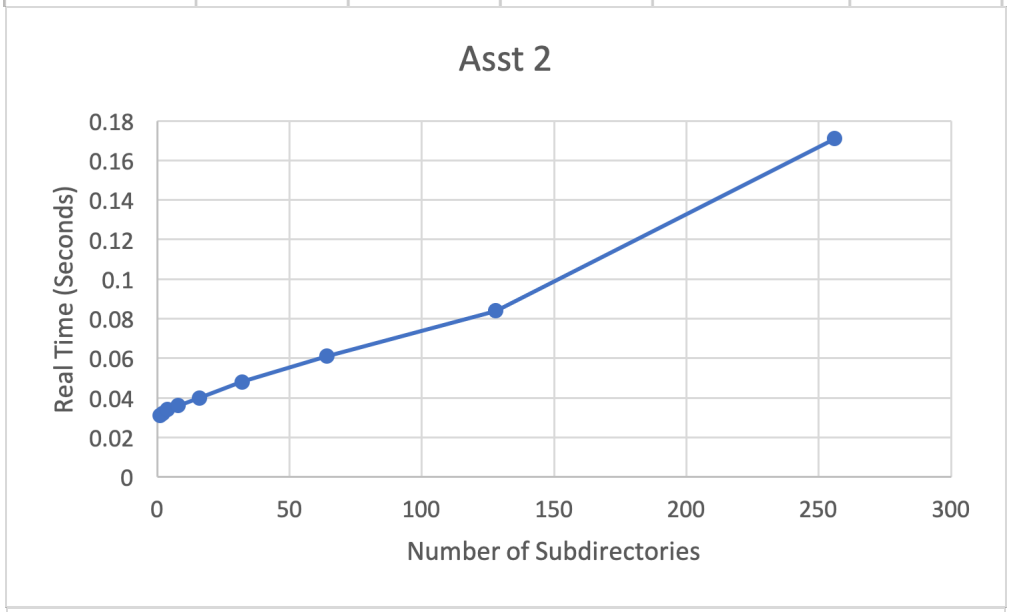


Files	Asst 2			Files	Asst 1
1	0.03			1	0.045
2	0.03			2	0.05
4	0.03			4	0.05
8	0.033			8	0.065
16	0.039			16	0.096
32	0.046			32	0.187
64	0.075			64	0.28
128	0.097			128	0.53
256	0.152			256	1.141

This is the comparison between Asst1 and Asst2 with a single directory of many files. Asst2 was much faster, as these were smaller files, and by the end was running over seven times faster.

This is the comparison between Asst1 and Asst2 with a long chain of subdirectories. In the last subdirectory, there were a few smaller CSV files. Asst2 was much faster again, as the files were small. Ass1 could not recursively spawn more than 128 processes, as each one had to wait for its subdirectory to finish and the computer we tested it on was unable to go past 128.

Subdirectories	Asst 2			Subdirectories	Asst 1
1	0.031			1	0.073
2	0.032			2	0.086
4	0.034			4	0.096
8	0.036			8	0.12
16	0.04			16	0.193
32	0.048			32	0.35
64	0.061			64	0.628
128	0.084			128	1.277
256	0.171				



This is the comparison between Asst1 and Asst2 when run with many large files. We ran them both with a number of copies of Movie_metadata.csv, which has 5000 lines. Here, Asst1 was faster because of the parallel file printing. However, it was never many times faster. Above, Asst2 had been many times faster with smaller files. So, with a different algorithm or parallel file printing in Asst2, it would have been much faster.

Number of mov	Asst 2		Number of n	Asst 1
2	0.898		2	0.675
4	1.895		4	1.524
8	3.845		8	2.934
16	7.563		16	6.012
32	15.348		32	11.497
64	31.055		64	23.045