Test Case	Multithreading Solution Time Average	Multiprocess Solution Time Average
16 Nested Subdirectories	1.18s	1.54s
1 CSV File	.344s	.153s
8 CSV File	.502s	.293s
16 CSV Files	.842s	.314s
32 CSV Files	1.34s	.441s
64 CSV Files	2.42s	.902s
128 CSV Files	1.84s	1.94s
256 CSV Files	2.94s	3.24s
512 CSV Files	5.32s	8.24s
1024 CSV Files	10.65s	18.43s

1. **Is the comparison between run times a fair one? Why or why not?**
   a. The comparisons between the runtimes are not fair. Once they are averaged out, and the outliers discarded, you will get a meaningful runtime estimation of both implementations. These are fine on there own, but not when compared. The two algorithms do much different things. Once sorts every file and exports it into one, and the other sorts each file individually and outputs a new CSV file for each one. It would be unwise to assume a fair comparison without going into depth on how these implementations differ.
2. **What are some reasons for the discrepancies of the times or for lack of discrepancies?**
   a. General time discrepancies: some discrepancies that I encountered were because of the iLab machines. It seems randomly, the machines would slow down and hang at a given point before continuing. It is hard to determine if that is because of the code, overuse of the iLab machine, or other iLab configuration settings. Sometimes the program would halt for up to 20 seconds before continuing, but would run normally when ran five minutes later.
   b. Discrepancies between implementations: Threads and processes have different attributes and pros and cons. It all depends on the use case. The run times for my implementations were roughly similar, although the multi-threaded application was a little quicker
3. **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?**

   a. The multi-threaded implementation could potentially be made faster, if every thread was treated as a “base case” in the mergesort algorithm, and then the main thread compiles all of the base cases back together for the outputted file. The multiprocess implementation seems to be quite efficient in its current form.

4. **Is mergesort the right option for a multithreaded sorting program? Why or why not?**
   a. I believe mergesort is definitely the right option. Having a dedicated thread to perform tiny parts of the mergesort tasks is exactly what mergesort calls for, and is exactly what threads aim to optimize. However, I may be wrong, and it would be interesting to compare runtimes for other sorting algorithms. My intuition tells me mergesort would be the best, though.