Analysis -

| # of files | Multithread Average(seconds) | Multiprocess Average (seconds) |
|---|---|---|
| 1 | 0m0 280s | 0m0 009s |
| 2 | 0m0 290s | 0m0 010s |
| 4 | 0m0 370s | 0m0 012s |
| 8 | 0m0 432s | 0m0 016s |
| 16 | 0m0 512s | 0m0 040s |
| 32 | 0m0 614s | 0m0 082s |
| 64 | 0m1 003s | 0m0 172s |
| 128 | 0m1 732s | 0m0 253s |
| 256 | 0m3 512s | 0m0 430s |

Our multiprocess program was much faster than our multithreading program. This may however be unreliable because our multiprocess programming had some bugs that caused the program to either omit information or delete it entirely. As a result, we researched the differences between multithreading and multiprocessing. Since both programs were doing inherently the same thing, our multithreader used the same large csv file for all of the compiling however, and as a result, made use of the same resource, whereas multiprocess would have duplicated all of the data.

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

It is not fair, both because our multithread program made use of a larger csv file to compile all of the data, whereas the multiprocess program made use of opening many csv files. However, our multiprocess program also had some faults in it that caused a lot of the data for that end to be unreliable.

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

The trend for both programs is very predictable. However, our multiprocess times were unreliable because they made use of a program that was incomplete.

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?

It's possible to make our multithreader more efficient by make use of the concept that multithreading uses one resource. With this, we can focus on trying to either compile the data more efficiently or sorting the data more efficiently by using one csv file for either.

Is mergesort the right option for a multithreaded program?

Mergesort is ideal for very large files, this is because it's able to sort a large csv file with exponentially more efficiency than a linear
sorting method. Since multithreading uses one resource, making sure we can maximize the sorting by splitting the process of sorting
rather than the amount sorted is crucial.