- **This file includes readme as well as analysis.**

# Readme

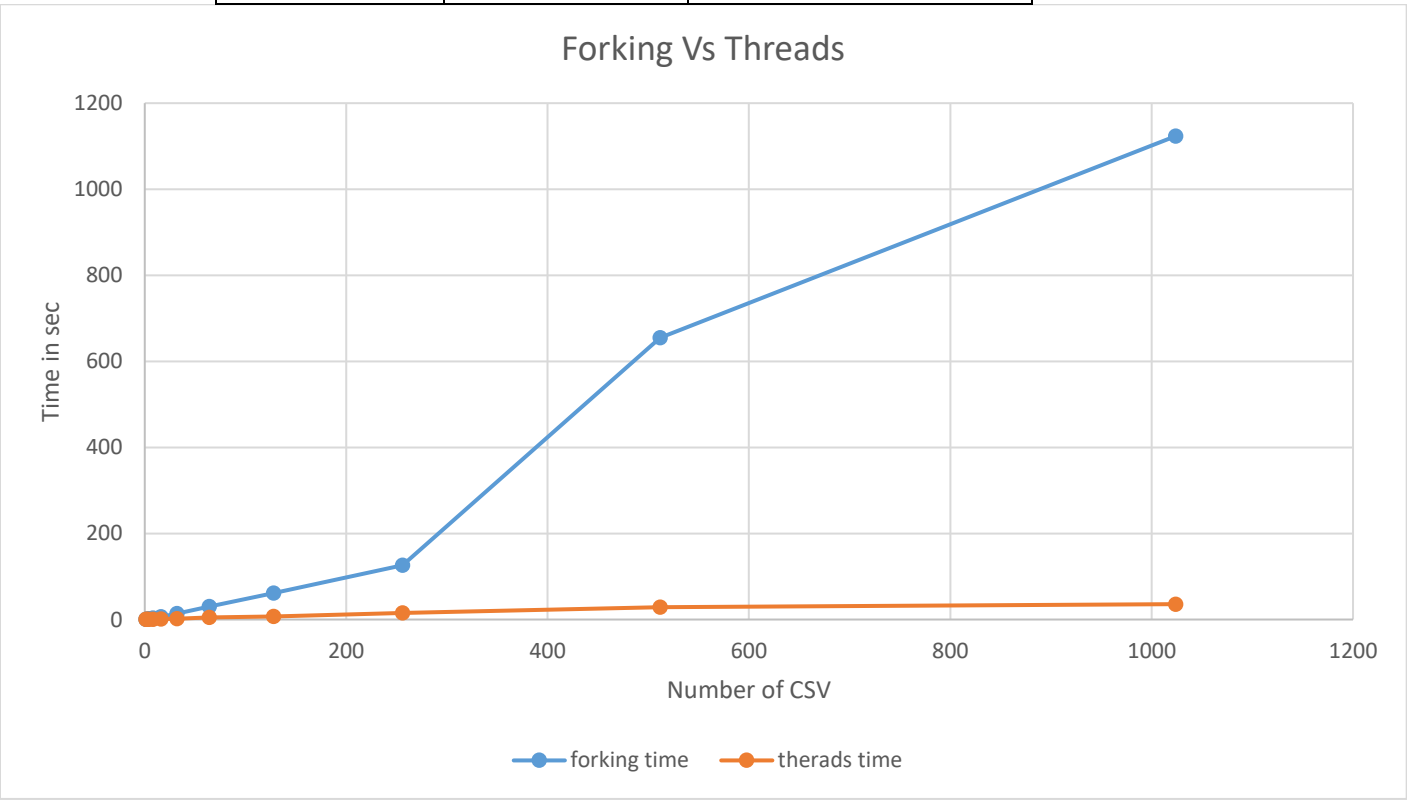
1. Compilation: (Default Behavior) –c flag only.
    a. gcc –o sorter_threads.c  mergeString.c sorter.h
    b. ./sorter –c <columnName>
        i. Write columnName same as in movie_metadata.csv
2. Algorithm:
    a. In this project we are creating threads when we hit directory as well as CSVs.
    b. After the csv is opened we are sorting and then storing it into global struct.
    c. Then after the content is copied into global struct, we are sorting it for the final time.
3. Code Design
    a. Everything is same as proj1, only changes are as follows
        i. Added two new functions
            1. One for directory
                a. Pthread_create is calling this function to traverse thru all the directories.
            2. Second for csv
                a. This function will copy the content from all csv files into one huge global struct.
                b. Call the Mergesort
4. Time Complexity: The time complexity of our code is O(nlogn) as we are purely rely on mergesort, where n is the number of movies from all the csv files.

## Analysis

This graph below is **Number of csv files VS Time in sec**.

This test is the mentioned on **1 Dir and 2^n** according to instructions sakai. **csv files.**

| Number Of csv | forking time | therads time |
|---|---|---|
| 1 | 0.228 | 0.033 |
| 2 | 0.736 | 0.1 |
| 4 | 1.7 | 0.21 |
| 8 | 3.32 | 0.42 |
| 16 | 6.47 | 1.002 |
| 32 | 13.82 | 2.08 |
| 64 | 30.065 | 4.52 |
| 128 | 61.09 | 7.43 |
| 256 | 126.01 | 15.54 |
| 512 | 654.89 | 28.283 |
| 1024 | 1123.5 | 35.7 |



***Observations:**

Using the time utility we found that multi-threading led to a faster program than multiprocessing. This can be explained for the reason that in our processing program we put all sorted information in different files, rather than sorting one large file. Due to the number of merge-sort calls and fopens for output files this lead to a remarkably slower program in comparison to the threaded application. We can also say that creating new processes does take more time than creating new threads but that should not have a great effect on run time. The wait function, which requires processes to be active until the child processes die also, can have a minimal effect on run time. The fact that these programs produce different outputs on the other hand does make this an unfair comparison. The main discrepancies in time is how multi-processing takes more time than multi-threading regarding csvs. The discrepancies between these two programs can also be explained by the different factors as stated above. I think the way to make processing faster than threading is to switch the outputs of the two programs. By doing so the run time of both programs should be drastically different. This will involve multi-processing being faster than multi-threading. I would say as well that the merge-sort algorithm is the best algorithm for a multi-threaded sorting program as it can be easily parallelizable, quicksort is also possible to parallelize but parallelizing quicksort is more difficult than merge-sort on a common basis. It may differ depending on data distribution, but for the most part merge-sort is the right option for multi-threaded programming.