Eric Fiore
NETID EJF96
RUID: 181008642

Michelle Dry Moran
NETID MDRY
RUID: 023007716

Analysis

1) Is the comparison between runtimes a fair one? Why or why not?

   The comparison is fair because both threading and forking attempt to do the same function. That is, they both function well for software that can perform multiple operations at once. In our project we search through multiple subdirectories, process any csv files we find, and finally sort the resulting data. Each one of those tasks need to be completed multiple times so it is desirous to run them in parallel. That is because while those tasks are running they do not depend on each other so can be completed independently. Both threading and forking are designed to do this exact type of work. Since each do this task in different ways one would like to know which way is more efficient.

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

   The reason for discrepancies in time is the heavy way in which forking works as opposed to threading. That is because, when forking, an entire new stack and heap must be set up for each fork. This work takes time which robs some of the benefit of forking.
   On the other hand, threading is much lighter so each thread can be set up in a much shorter time. This is because each thread shares the same heap so less work needs to be done when creating a thread. For this reason we can expect threading to be faster for doing workloads such as the one we were assigned to accomplish.

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?

   Since forking and threading have strengths making one faster or slower is really a question of when should we use one as opposed to another. In other words, when deciding forking or threading one needs to keep in mind what each is designed for and choose the best option. If sharing is information is critical to the functioning of the code then threading is will be faster because sharing of information in forking is a heavy and slow process. On the hand if no sharing of information is necessary then forking might be the faster option. Since no locking is necessary in forking and no chance of overwriting of information is possible the coder can simply set up a fork and let the code run.

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

   Mergesort is a good option or multithreading because the algorithm can be broken down into many small pieces. Each small piece can be handled by a one thread and then joined back together later on.