multithreads:

“# -” denotes the amount of csv files that were sorted in a directory.
Each csv file is the largest possible csv file that we are supposed to sort.
1 -
Total number of threads: 2

real 0m0.058s
user 0m0.040s
sys 0m0.000s

2 -
Total number of threads: 3

real 0m0.107s
user 0m0.076s
sys 0m0.008s

4 -
Total number of threads: 5

real 0m0.204s
user 0m0.160s
sys 0m0.016s

8 -
Total number of threads: 9

real 0m0.446s
user 0m0.320s
sys 0m0.020s

16 -
Total number of threads: 17

real 0m0.841s
user 0m0.668s
sys 0m0.072s

32 -
Total number of threads: 33

real 0m1.795s
user 0m1.308s
sys 0m0.104s

64 -
Total number of threads: 65

real 0m3.315s
user 0m2.544s
sys 0m0.248s

128 -
Total number of threads: 129

real 0m7.173s
user 0m5.148s
sys 0m0.448s

256 -
Total number of threads: 257

real 0m13.751s
user 0m10.400s
sys 0m0.756s

512 -
Total number of threads: 513

real 0m28.338s
user 0m20.540s
sys 0m1.712s

1024 -
Total number of threads: 1025

real 0m51.564s
user 0m40.968s
sys 0m3.540s

Multiprocess:

1 -
Total number of processes: 2

real 0m0.072s
user 0m0.040s
sys 0m0.000s

2 -
Total number of processes: 3

real 0m0.126s
user 0m0.076s
sys 0m0.004s

4 -
Total number of processes: 5

real 0m0.219s
user 0m0.140s
sys 0m0.028s

8 -

Total number of processes: 9

real 0m0.402s
user 0m0.304s
sys 0m0.028s

16 -
Total number of processes: 17

real 0m1.037s
user 0m0.640s
sys 0m0.024s

32 -
Total number of processes: 33

real 0m2.894s
user 0m1.284s
sys 0m0.036s

64 -
Total number of processes: 65

real 0m4.039s
user 0m2.588s
sys 0m0.132s

128 -
Total number of processes: 129

real 0m7.437s
user 0m5.272s
sys 0m0.340s

256 -
Total number of processes: 257

real 0m15.750s
user 0m13.344s
sys 0m1.156s

We were told we could assume the maximum number of files are 255 for the multiprocess project. I tried doing 1024 files and my code was not optimized well enough to handle that so I recorded up to 256.

Questions

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

I think do so. Because it depends on the operating system. Processes have their own resources..address spaces and other computer resources. On the other hand, threads live inside process and share the resources of the process. Both have their advantages and disadvantages and one being superior than the other has everything to do with how they are being implemented. In this case, threads proved to be faster.

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

A thread is a path of execution through the program. With a single thread, there is always a single path of execution. With multiple treads, there are two or more paths of execution. What this means is that when using a single-thread, only one task can be done at a time and the program waits until a task is finished before starting another one. For most programs and uses, one thread of execution is all that is needed or can be used. In other cases, and for specific tasks, the ability to do use multiple threads in parallel can result in significant performance gains. One such case is simulation. Simulation relies on multiple repetitions of the same sequence of events, thus the ability to run multiple simulations in parallel provides a significant benefit.

Running more than one thread increases the memory usage. In general, the number of threads used is proportional to the memory used (e.g., 2 threads uses 2X memory, etc.). Depending on the size and complexity of the program, this may cause an "out of memory" error, especially if on a 32-bit operating system.

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?

In general, to decrease the execution time, programmers use multi-threaded computing.
The processor is not the one who allocates the threads. The processor is the one who provides the resources (virtual CPUs / virtual processors) that can be used by threads by providing more than one execution unit / execution context. Programs need to create multiple threads themselves in order to utilize multiple CPU cores at the same time.

The two major reasons for multi-threading are:

Making use of multiple CPU cores which would otherwise be unused or at least not contribute to reducing the time it takes to solve a given problem - if the problem can be divided into subproblems which can be processed independently of each other (parallelization possible).
Making the program act and react on multiple things at the same time (i.e. Event Thread vs. Swing Worker).
There are programming languages and execution environments in which threads will be created automatically in order to process problems that can be parallelized. Java is not (yet) one of them, but since Java 8 it's on a good way to that, and Java 9 maybe will bring even more.

Usually you do not want significantly more threads than the CPU provides CPU cores, for the simple reason that thread-switching and thread-synchronization is overhead that slows down.

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

Because lower directory search and file content sorting is independently, if use multi-threading method then program execution speed can be increased.

Summary

The big takeaway based on the times that I recorded is pretty simple. The change in times as the number of csv files grow in multithreading shrinks, meaning the program is getting faster per additional csv file. For multi-process, this is not the case. The time for each addition csv file increases over time. This could be do to the quality of my code for it but I believe that when it comes to sorting csv files, multi-threads is faster in the long run that is for each additional csv file being sorted.