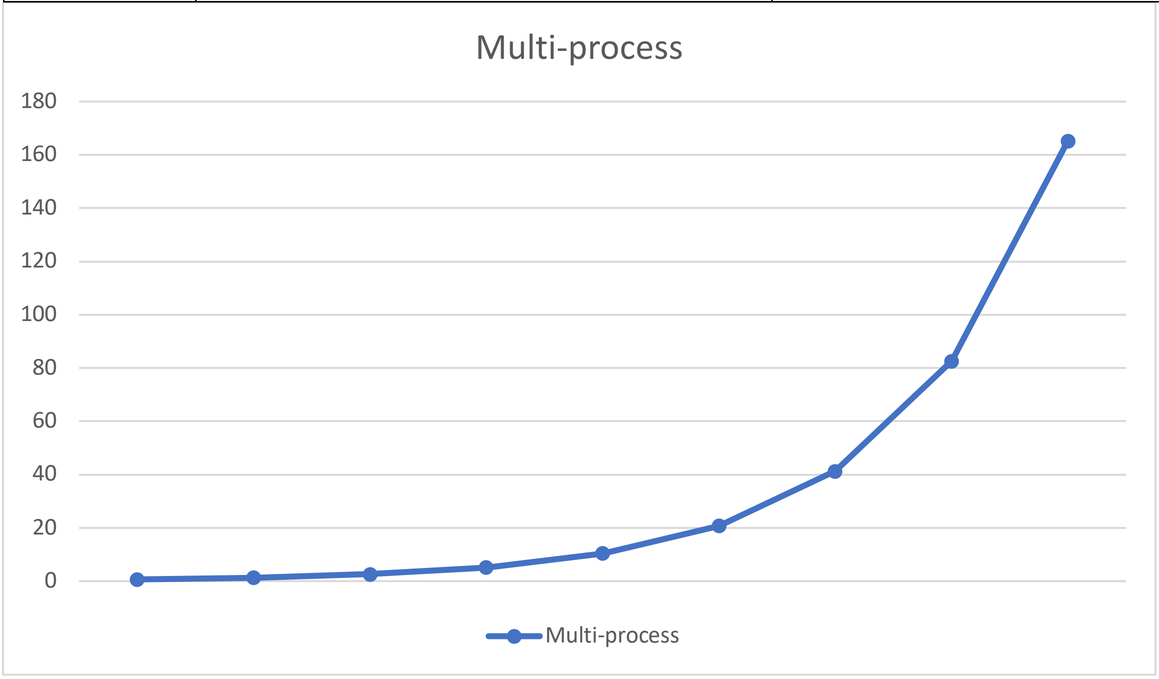
Files Count	Average Multi-process runtime (seconds)	Average Multi-thread runtime (seconds)
1	(.647, .646, .645) ➔ .646	
2	(1.289, 1.315, 1.297) ➔ 1.303	
3	(2.582, 2.615, 2.601) ➔ 2.599	
8	(5.138, 5.192, 5.241) ➔ 5.190	
16	(10.430, 10.356, 10.406) ➔ 10.397	
32	(20.936, 20.831, 20.527) ➔ 20.765	
64	(41.124, 41.224, 41.392) ➔ 41.247	
128	(82.763, 82.343, 82.655) ➔ 82.587	
256	(164.790,166.138, 164.991) ➔ 165.306	



Discussion

Our method was to create folders corresponding to the file counts listed above. Each file would contain a "gross" column to be sorted by, and the output file would be written to an "out" folder at the same level of each of the folders containing the files to be sorted. The sorters was run 3 times per file count

Creating a stable, isolated testing environment without any variables is almost impossible – varying amounts of students running various programs on the iLab machines will change our results every time we test. This calls for an average case analysis. Overall, though, the multi-thread process runs much faster on average.

The sorting algorithm used was mergesort – a sorting algorithm with an O(nlogn) runtime for n inputs. Our mergesort algorithm is identical in both our multi-process and our multi-threaded programs. This allows us to directly compare the speed of creating multiple processes vs creating multiple threads.

However, this comparison between threads and processes is somewhat frivolous. Creating a new process involves copying and allocating memory for the new process whereas threading shares data and operates concurrently on it at the same time as all other threads. The desired structure of each program also calls for sorting m files of varying sizes (multi-process) or sorting the total amount of rows from m files (multi-threaded). This leads to a much smaller threaded run time for larger file counts.

There is more leeway in speeding up the threaded program and also having a fair comparison with the multi-processed program. The multi-threaded program could create a new thread on each recursive call in order to speed up the sorting and merging. The multi-threaded program could also follow the same directions as the multi-process program by sorting each file individually rather than aggregating every valid row. This would allow for a fair comparison.