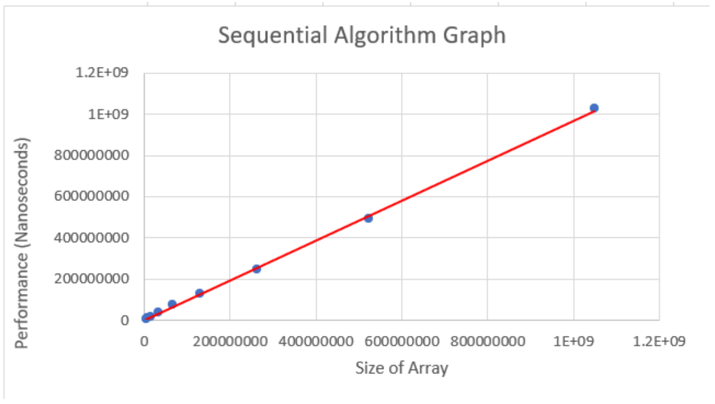
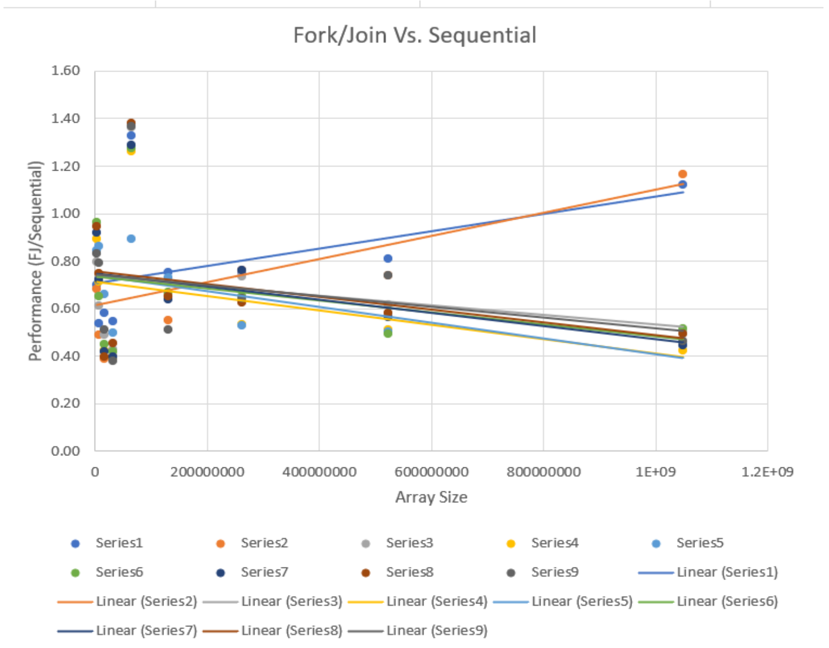
CountStringsFJ

Jonathan Wenger

Graph #1: Sequential Algorithm (Performance Vs. Size)



Graph #2: Ratio of FJ & Sequential Vs. Array Sizes



- Series1: Threshold of 1,000
- Series2: Threshold of 10,000
- Series3: Threshold of 30,000
- Series4: Threshold of 40,000
- Series5: Threshold of 45,000
- Series6: Threshold of 50,000
- Series7 Threshold of 60,000
- Series8: Threshold of 100,000
- Series9: Threshold of 1,000,000

My computer has 4 cores. I did achieve a significant speedup, especially as the array size got larger (I got as much as a 55% speedup when the array size was large), but as the array size got closer to 0, the speedup of FJ got less fast compared to sequential (sometimes FJ being even slower than sequential). I believe the reason it was slower is because creating multiple processors/threads also takes time, which can reduce the overall speed of the algorithm. When the array size is huge, that can really speed things up cutting down the array, but when the array is small, cutting the array down into smaller parts can be a waste compared to sequential. One exception to this is when the threshold was 1,000 and 10,000 for when my array was huge, where the FJ algorithm was significantly slower than the sequential one. I believe this was the case due to the fact that so many processes were running concurrently that it overall slowed down my computer from the amount of processes happening at the same time. Especially for only a 4 core computer like mine, running in parallel really takes a toll on my machine (as I noticed my machine froze sometimes for a second and the fan inside was going crazy). One smaller array size (65,536,000) actually caused the FJ algorithm to run slower than sequential, which I'm really not sure why (as array sizes both bigger and smaller ran faster with FJ compared to sequential). Overall, FJ definitely sped up the algorithm, and helped produce more results quicker.