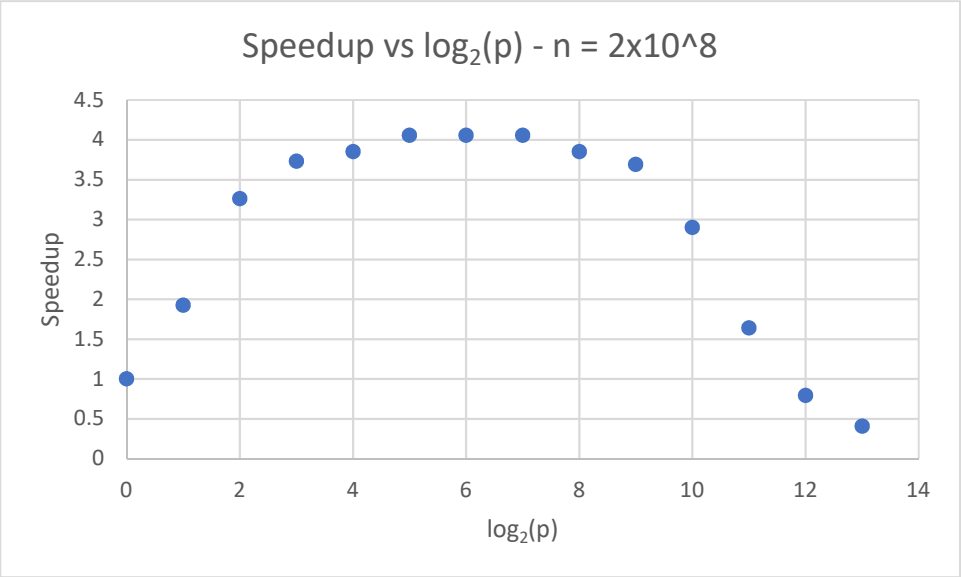
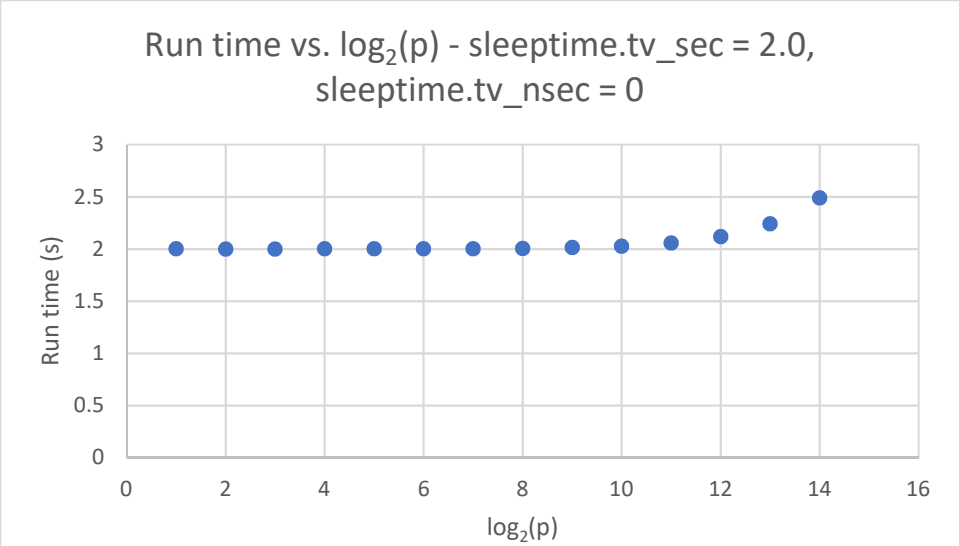
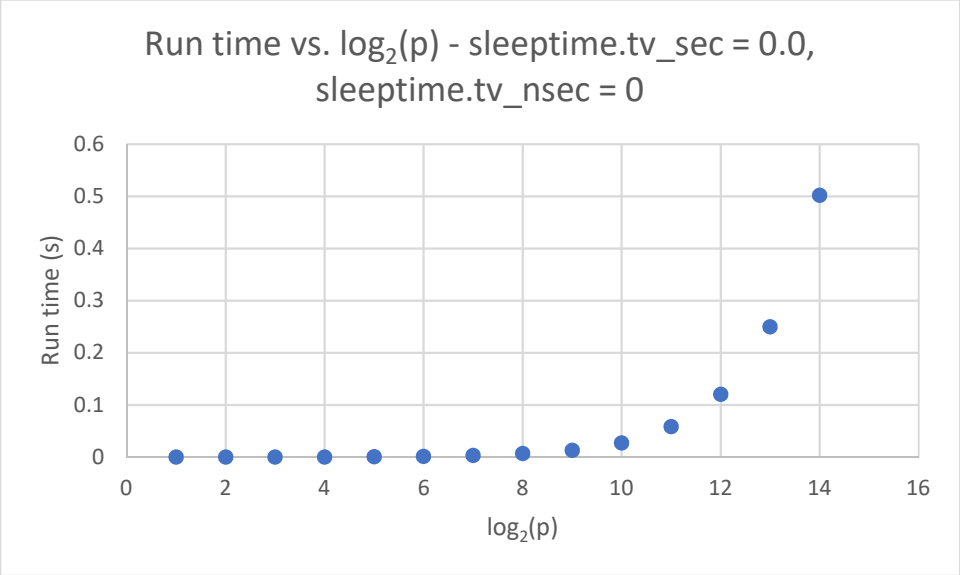
Jacob Hessong

CSCE 435 HW2



As $\log_2(p)$ varied from 1 to 13, the execution time decreased and speedup increased until leveling out from $\log_2(p) = 5$ to 7, after which the execution time began to increase and speedup decrease, eventually having worse performance than just one thread. This makes sense, as at first, adding more threads speeds up the process of finding the minimum because each thread has a much smaller list to find the minimum of. This cuts down the amount of time it takes to find the minimum much more than the extra time that the amount of overhead involved with using locks and switching between threads incurs. However, once the process reaches its minimum point, the large number of threads begin to weigh down the execution time of the program, as each thread finds the minimum of its list fairly quickly, meaning that the overhead of each thread waiting on the lock and of switching between threads now takes significantly longer than the actual finding of the minimum. Adding more threads then continues to make run time longer and longer until the program is eventually slower than just one thread finding the minimum of the list.





These two iterations of the barrier program grow in almost identical ways. Both demonstrate exponential growth in relation to $\log_2(p)$, with the primary difference being the change in y-intercept that comes from having the first iteration sleep for 2 seconds first. This makes sense, as the barrier has to wait on all of the threads to reach it before the program can move forward. Because p is growing exponentially, it follows that each time the number of threads increase, the program has to spend about twice the time waiting at the barrier for all threads to reach it.