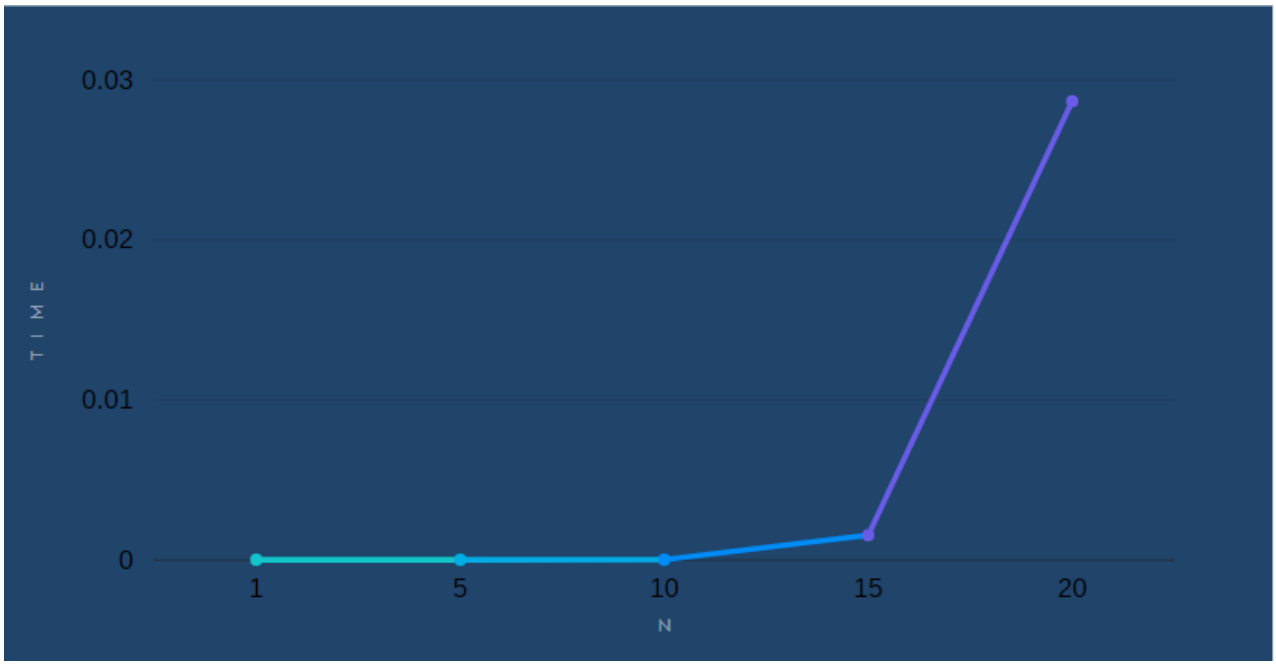
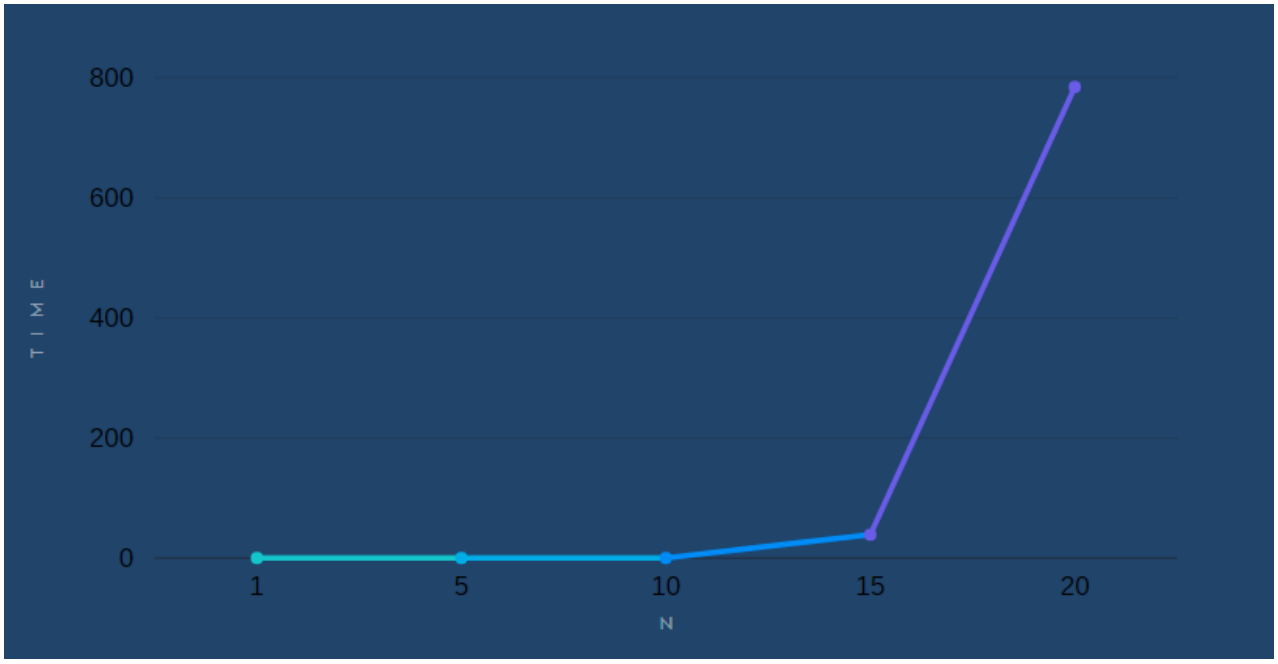
Report – PA1

Sambhav Mattoo

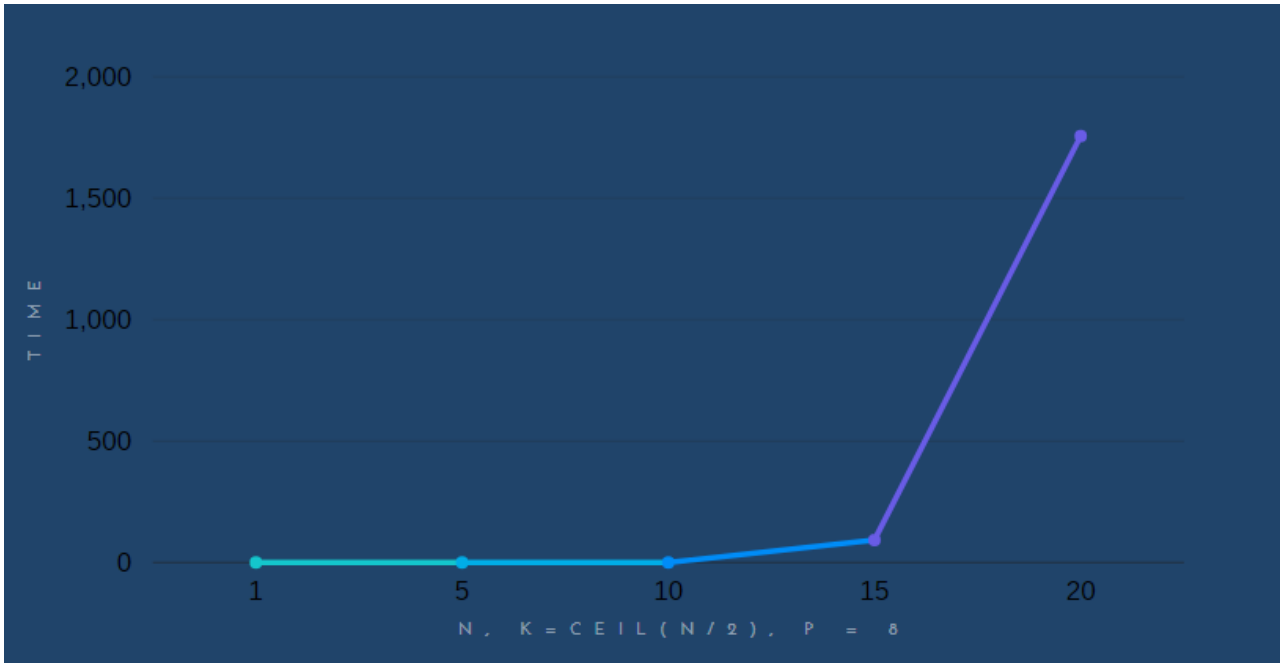
Here is how the sequential code performs with exit on first, across varying N.



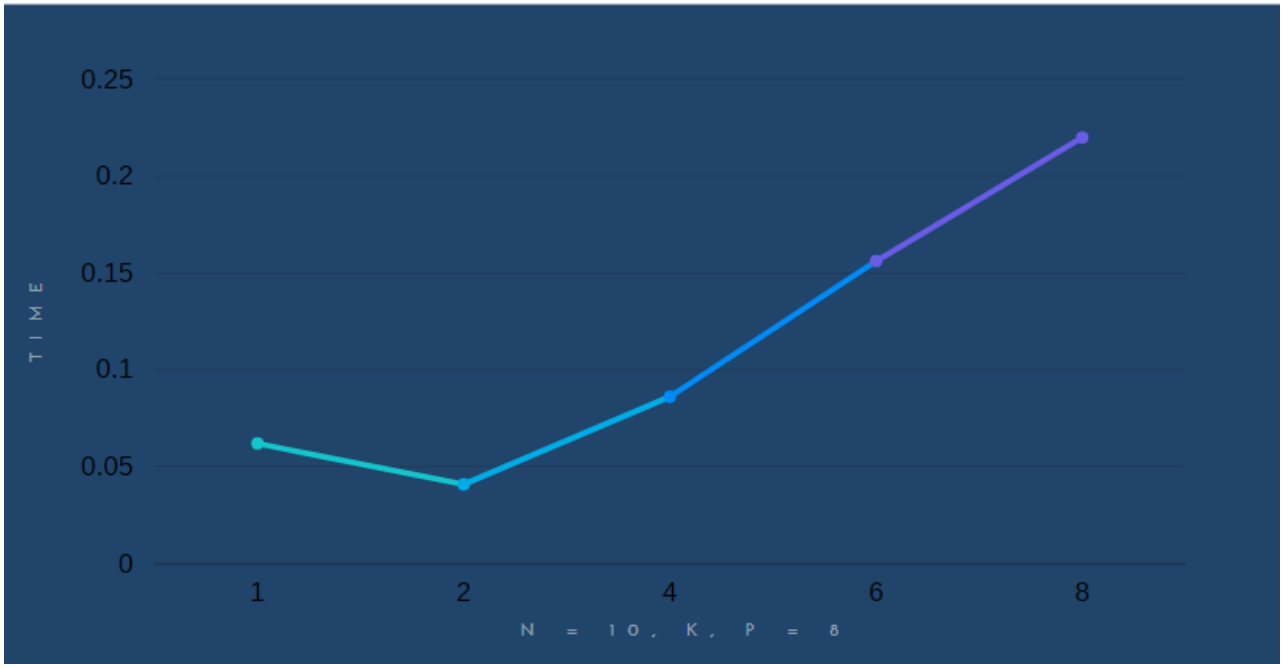
And this is the same, except for the complete solution.



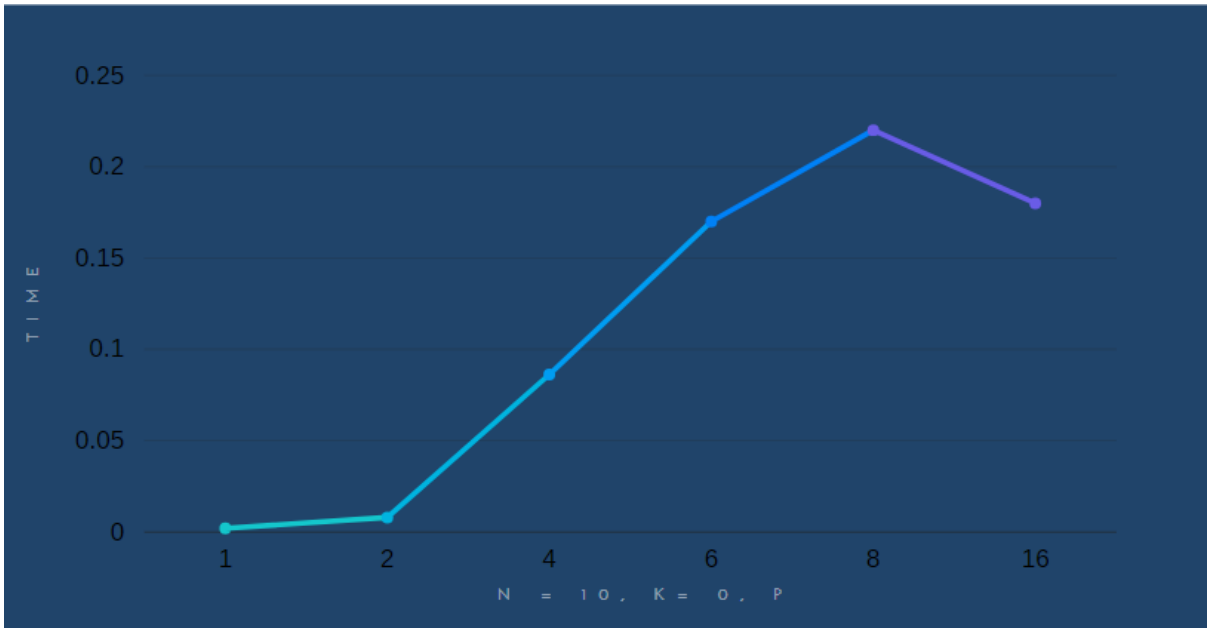
Now, we vary N, with P = 8, and K fixed at Ceil(N/2) to keep it fair across all processes.



Next we vary K when we fix N at 10 and P at 8.



Finally we vary P from 1 to 16 while keeping N fixed at 10 and K fixed at 0.



These graphs show several interesting trends:

1.) The graph when EoF is activated follows the logarithmically increasing time complexity of the full solution, but at a far smaller scale, as expected.
2.) The above is mirrored by the parallel algorithm in varying N.
3.) At our scale, varying K seems to cause what seems an almost linear time increase. This is also probably because at K = 1 we degenerate to sequential.
4.) When varying P, we don't start to get a speed-up effect due to parallelism till after using 8 processors.

My largest test was on N = 10, K = 0 and P = 16. This gave me a ~0.18 second running time. But sequential alone for N = 20 took me the most time at 782 seconds.

My implementation details are elucidated well in my code comments, but the basic idea sequential idea is to pick a row in each column, store that, and see the rows in the next column that are free from any checks. Pick one more, and so on and so forth. Then, once you come to a solution, you backtrack to the last chosen row, and try to find other solutions; do this recursively till you find all solutions. This is pretty much the same in parallel, with the master doing the kth partial sums of the work, and each worker then getting that and then doing the rest of the work across all available threads, then getting killed in sequence they finish in. The master continuously compiles the answer, so the threads don't really wait. Its as efficient as I could make it. The exact layout is exactly as laid out in the TAs code comments.