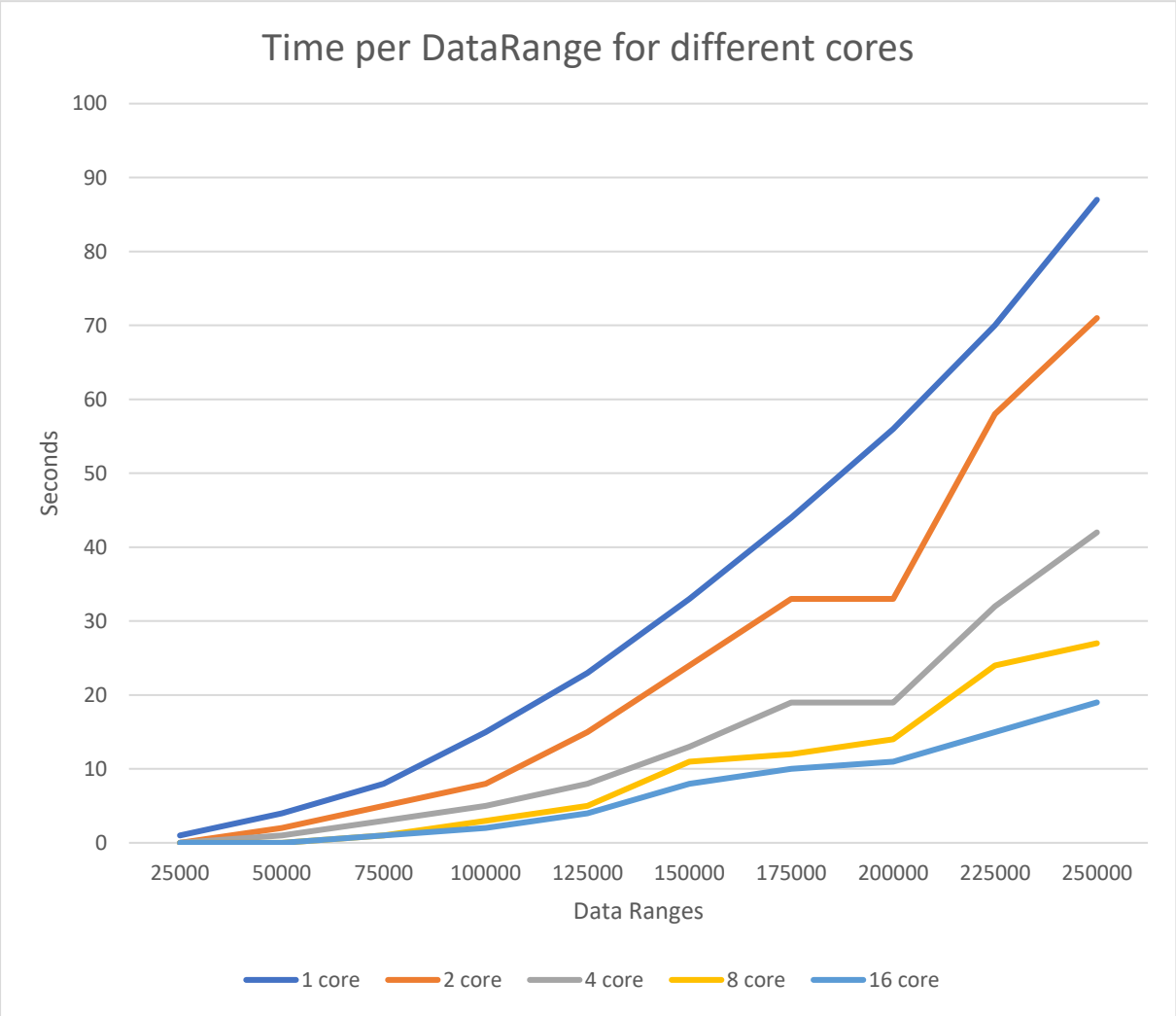
Nicholas Hegarty 18317256

Task 1:

I generated 10 sets of work overall to be processed by the pool function. The sets consisted of ranges 25000,50000,75000… all the way up to 250000, using increments of 25000. I processed each set with pools using 1, 2, 4, 8 and 16 cores each, and plotted the results to a graph(below).



To quantify the speed up, the difference between 1 and 16 cores for the final set was 68 seconds. This meant the 16-core run was 458% faster than the single core.

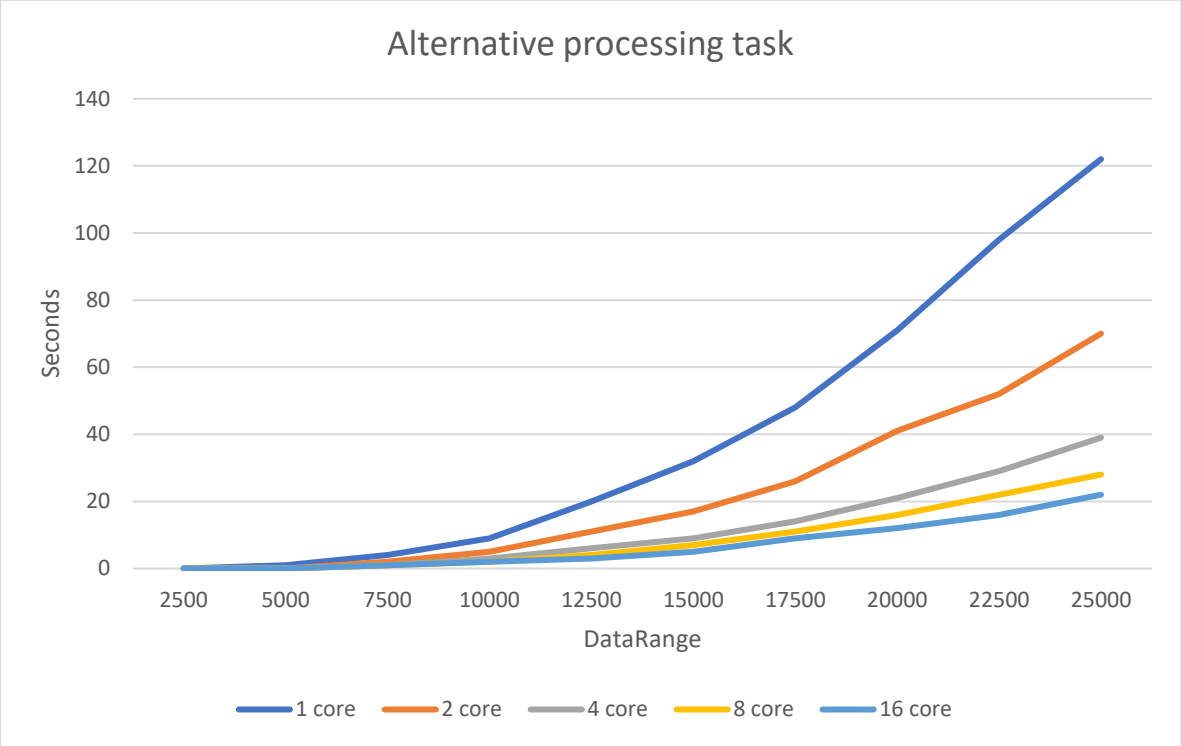
I calculated this percentage for all data ranges and found that the 16 core runs were on average 551% faster than the single core runs.

We learned from this exercise that multiprocessing provides massive benefits in time taken due to processing speeds, and this speed difference becomes increasingly noticeable as the size of the dataset to be processed scales rapidly. The speed of the individual cores is all roughly the same, but multiprocessing allows you to utilise extra cores to the task, the performance increases substantially.

The benefits are also apparent not just using 16 processors, we also saw comparable benefits at the 2, 4, 8 processor counts.

Task 2:

The alternative task I implemented was finding factorials of a set of numbers using the python math.factorial() function for all the datasets. This time I used the previous ranges, but each max value was divided by 10 (eg 2500 – 25000) as this task took significantly longer than the previous check_prime function. I processed each set with pools using 1, 2, 4, 8 and 16 cores each, and plotted the results to a graph(below).



To quantify the speed up, the difference between 1 and 16 cores for the final set was 100 seconds. This meant the 16-core run was 555% faster than the single core.

I calculated this percentage for all data ranges and found that the 16 core runs were on average 556% faster than the single core runs.

From this task we confirmed what we knew from task 1; that multiprocessing provides massive benefits in time taken due to processing speeds, and this speed difference becomes increasingly noticeable as the size of the dataset to be processed scales rapidly. The speeds of the individual cores are all roughly the same, but multiprocessing allows you to utilise extra cores to the task, the performance increases substantially.

In this test, using our range of values, we found that the jump between 1 & 2, and 2 & 4 cores was more substantial, compared to the later higher processor jumps. I expect that if we continued to increase our data ranges further, these jumps would become even wider, based on the lines' trajectories.