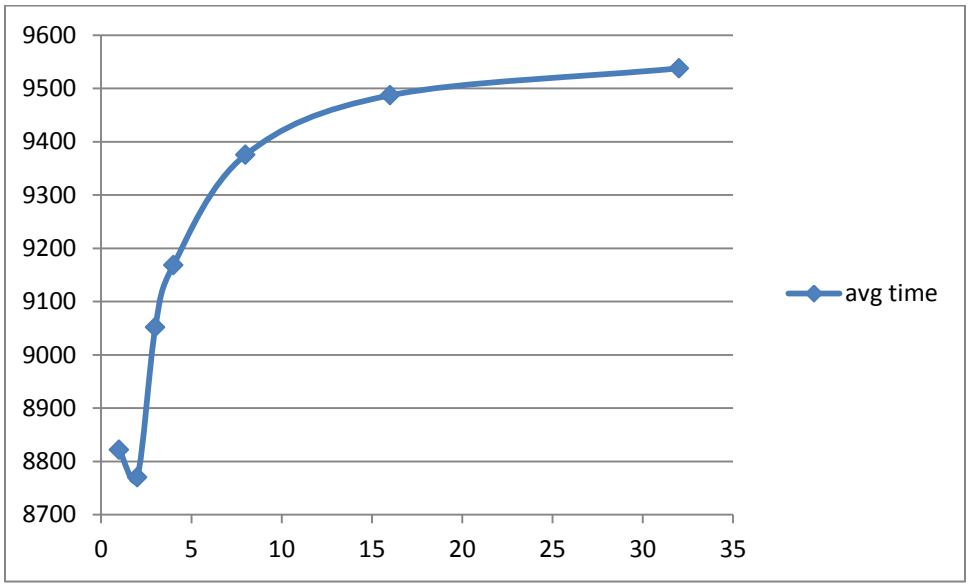
An Nguyen Dang

CSC 332

Average run time for thread count chart (ms)



Data table

#threads	1	2	3	4	8	16	32
avg time	8822	8770	9051.667	9168.333	9375.333	9487	9537.667
time1	8828	8768	9025	9180	9338	9504	9562
time2	8805	8768	9060	9152	9376	9473	9540
time3	8833	8774	9070	9173	9412	9484	9511

Explanation:

- To calculate all factor of the number, we need to go through all integers between 1 and the square root of the number. Compare to check if the number is prime number or not, finding factors don't have interrupt condition. Therefore, it still needs to go through the same amount of value as non-multithread algorithm.
- The computer only have four CPUs, therefore, only 4 threads can run synchronized at one, more threads after that will only increase the overhead. However, in actuality, not all CPU will be utilize by the program, for it nature of accessing main memory causing interrupt, and the OS running other program as well
- There is a lot of overhead for each thread, each need to update its local index variable i and check its conditions, check for interrupted, and other thread information separately. Besides, it also need to calculate the lower and upper bound for each thread, the more thread the more calculation.

With these three main reasons, we can see how the graph has its shape:

- From one to two threads, we see a small drop in run times, mainly because the sequential part of the code has bigger proportion than the parallel part.
- After two thread, the runtime increase at really high rate, probably because the overhead cost, while the size of each thread for loop is still quite large (using CPU constantly)
- Then the increasing rate decreasing, because the values need to be examined stay the same, the size of for loop in each thread get smaller (more memory request than CPU)

Using really big number of thread will firstly increase the run times, and eventually java runs out of heap space (memory) because of the memory overhead of the threads.