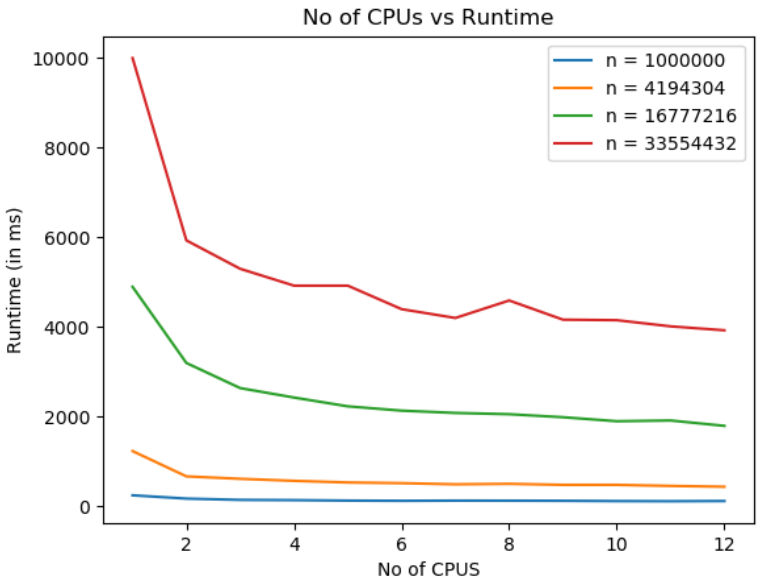
COL380 - Assignment 1

Name: Bommakanti Venkata Naga Sai Aditya
Entry No.: 2019CS0471

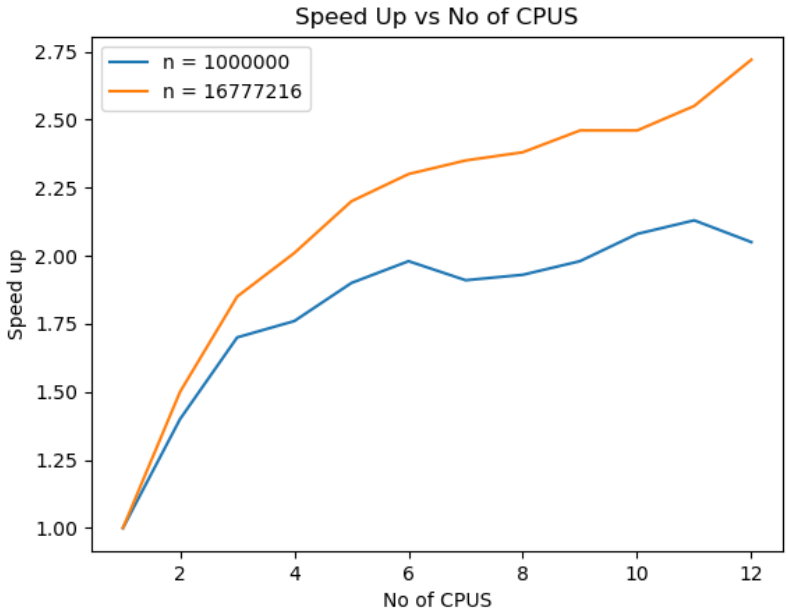
Outline of Implementation and design choices:

- I used merge sort as SequentialSort
- In step 4 where we have to split A into p partitions B_0, B_1, .., B_{p-1}: I have divided A into 25 (almost) equal sized parts and I have created 25 tasks (because each thread can perform ~1 task, given max no of cores = 24). In each task, we place n/25 elements into their respective partitions. To avoid synchronization issues, I have created a 2D array for each of the p partitions (i.e., B_0, B_1, .., B_{p-1}, each B_j is of size 25 x no of elements in B_j. no of elements in B_j is calculated beforehand using a similar method) and then in task i, if an element belongs to B_j, then we add that element to the array $B_j[i]$. Now, as all tasks append to different arrays there is no problem of synchronization among tasks
- Later, I merged all the $B_j[i]$'s into B_j, where $B_j = \bigcup_{i=0}^{24} B_j[i]$. This step is also performed using tasks (total p tasks) where each task merges one B_j.
- Now, while sorting each of B_j's I used tasks (total p tasks) where each task sorts one B_j.
- Now, we have to concatenate all the sorted B_j's into A. Here, I used tasks where each task (total p tasks) correctly places one B_j in A.

<u>Graph of Runtime vs no of cpus (p = 40)</u>



Graph of Speedup vs no of cpus



As we increase the number of CPUs, the runtime is decreasing and for larger inputs and Speed Up increasing. Also, the Speed Up for larger inputs is higher than Speed Up for smaller inputs. So, the code is fairly scalable.