**Gaurav Thorat**

**002957343**

**INFO – 6205 Section 6**

**ASSIGNEMENT 5**

**Task: -** To implement a parallel sorting algorithm such that each partition of the array is sorted in parallel. Consider the following while parallel sorting

1. A cutoff (defaults to, say, 1000) which you will update according to the first argument in the command line when running. It's your job to experiment and come up with a good value for this cutoff. If there are fewer elements to sort than the cutoff, then you should use the system sort instead.
2. Recursion depth or the number of available threads. Using this determination, you might decide on an ideal number (t) of separate threads (stick to powers of 2) and arrange for that number of partitions to be parallelized (by preventing recursion after the depth of lg t is reached).
3. An appropriate combination of these.

Observations: -

1. For arrays of same size and same cutoff we can see that there is a performance difference when using less threads. If we use a smaller number of threads the sorting is slower.
2. Similarly, we can also conclude that after a certain number of threads there is no performance gain by having a greater number of threads.
3. After this observation about threads, the other thing that **significantly affects** the performance of parallel sort is the **cutoff** value. Cutoff value for smaller is inefficient or not that fast, and the performance while having a high cutoff is also not that efficient.
4. When the **cutoff value** is between **25-30% percent** of the **array size** we see a considerable performance difference. As this where it takes the **least** amount of time to sort irrespective of the number threads but by having the same array size and cutoff values.
5. In the below fig the above observation is highlighted in yellow.

