**Parallel Sort Assignment – 001569753**

**Implementation:**

* I’ve implemented the parallel sort for 3 different **array sizes**(2 million, 4 million, 10million)
* Ran the main function for 6 different **thread counts** incremented as a power of 2 (1,2,4,8,16,32)
* For each array size, taken 6 different **cut-off** values to analyze the efficiency of parallel sorting algorithm
* The duration is shown in milli seconds in entire report
* To summarize, the cut-off value after which the system sort is to be used to make the algorithm most efficient seems to be **around 40-50% of the total size** of the array.
* That is, in recursion when we reach the array size **around half** of the given array’s size, **shifting to system sort** method **instead of the recursive sorting** algorithm will make the algorithm most efficient and gives us optimal results
* Multi-threading will help the parallel sorting algorithm work more efficiently when the array size ki significantly high

**Firstly, for 2 million** array size, the cutoffs chosen are (50000, 100000, 200000, 400000, 800000, 1600000).

* After observing, as the 2million size is comparatively smaller, this has been a little inconsistent with the cut-offs, which is not the case for larger array sizes**(4million and 10million, please check next pages).**
* But on an average, 8,00,000(i.e., 0.8mil cut-off) seems to be optimal for 2M sized array.
* The **bolded time** values are where the minimum value occurred for different thread count
* The minimum average shown up when the **thread count is 2**

Chart

Description automatically generated

--------------------------------------------------------------------------------------------------------------------------------

**For 4million** array size, the results are very consistent and evident

* For all the thread counts, the minimum time taken occurred at 1600000(1.6mil) cutoff
* Check the bolded values, the minimum time taken occurred at that cutoff point
* The minimum average time shown up when the thread count is 8

Graphical user interface, application

Description automatically generated

**For 10million array size**, also the results are very consistent. We can clearly observe at what cut-off point the algorithm is efficient.

* Out of the taken cut-off values, the 3200000(3.2Mil) cut-off seems to be very optimal for 10mil sized array.
* Check the bolded duration values.
* For all the thread counts, this cut-off seems to make the algorithm efficient
* **The thread count is 32**, where the algorithm is efficient for 10M array size

Graphical user interface, chart, application

Description automatically generated

------------------------------------------------------------------------------------------------------------------------------------------

**Array Size vs Thread Count :**

Increasing the no.of threads, when the array size changes significantly, appreciably increases the efficiency of algorithm.

**Chart, line chart

Description automatically generated**

**Conclusion:**

* The cut-off value after which the system sort is to be used is **around 40-50% of the total size** of the array
* That is, in recursion when we reach the array size **around half** of the given array’s size, **shifting to system sort** method **instead of the recursive sorting** algorithm will make the algorithm most efficient and gives us optimal results
* Multi-threading will help the parallel sorting algorithm work more efficiently when the array size ki significantly high
* Increasing the number of threads when there is a significant increase in the array size, helps for the optimal use of parallel sort algorithm
* The time values in milli seconds may vary from system to system. Mostly, the conclusions will be similar in all the machines