**Program Structures & Algorithms**

**Spring 2022**

**Assignment No 4**

**Name : Roja Pinnamraju**

**Nuid : 002925814**

**Task :**

* Implement a parallel sorting algorithm such that each partition of the array is sorted in parallel. You will consider two different schemes for deciding whether to sort in parallel.
* (Part 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.
* (Part 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).
* (Part 3) Implement a main program to run the following benchmarks: measure the running times of this sort.
* Show the results of your experiments and draws a conclusion (or more) about the efficacy of this method of the parallelizing sort.
* Experiments should involve sorting arrays of sufficient size for the parallel sort to make a difference. You should run with many different array sizes (they must be sufficiently large to make parallel sorting worthwhile, obviously) and different cut-off schemes.

**Relationship Conclusion :**

We have run simulations of experiments with different combinations of the cutoff values, threads, and array sizes. From the observations of the runtimes, we can conclude that four threads are the optimal choice and there wouldn’t be much improvement in algorithm performance beyond four threads.

The lowest runtime is achieved when the cutoff value is 25% of the array size.

For recursion depth and number of threads available

Maximum depth possible:

Any depth more significant than the max depth is not feasible as the partitioned arrays hit the cutoff and turned into a system sort.

**Evidence to the Conclusion :**

Below are the runtimes for different combinations of Array size, threads, and cutoffs.

**Array size = 200000**

Table

Description automatically generated

**Array size = 100000**

**Table

Description automatically generated**

**Array size = 50000**

**Table

Description automatically generated with low confidence**

**OUTPUT :**

**Text

Description automatically generated**

**Text

Description automatically generated with medium confidence**

**Text

Description automatically generated with medium confidence**

**A picture containing text

Description automatically generated**

**Text

Description automatically generated with medium confidence**

**A picture containing text

Description automatically generated**