**Program Structures & Algorithms  
Spring 2022  
Assignment No. 4**

Name: Ankur Thakur  
(NUID): 002927572

**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 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 cutoff schemes.

**Relationship Conclusion**

We have run simulations of experiments with different combinations of the cut-off 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.

T**he lowest runtime is achieved when the cut-off value is 25% of the array size.**

For recursion depth and number of thread variable

Maximum depth possible:

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

**Evidence to the Conclusion**

Below are the runtimes in ‘milli seconds(ms)’ for different combinations of Array size, threads, and cutoffs.

Array size = 100000

A picture containing text, white, screenshot

Description automatically generated

Array size = 200000

**A screenshot of a computer

Description automatically generated with low confidence**

Array size = 400000

**Table

Description automatically generated**

**Output Screenshot**

Array size = 100000

A screenshot of a computer

Description automatically generated

Array size = 200000

A screenshot of a computer

Description automatically generated

Array size = 400000

A screenshot of a computer

Description automatically generated

Size of an array: 400000

Degree of parallelism: 2

13333 403

26666 287

39999 204

53332 220

66665 220

79998 208

93331 232

106664 228

119997 228

133330 231

146663 245

159996 222

173329 217

186662 226

199995 210

Degree of parallelism: 4

13333 206

26666 156

39999 155

53332 167

66665 160

79998 163

93331 159

106664 156

119997 156

133330 156

146663 159

159996 158

173329 152

186662 158

199995 184

Degree of parallelism: 8

13333 202

26666 168

39999 139

53332 137

66665 133

79998 136

93331 133

106664 150

119997 158

133330 145

146663 159

159996 152

173329 157

186662 153

199995 145

Degree of parallelism: 16

13333 181

26666 137

39999 127

53332 146

66665 133

79998 159

93331 154

106664 168

119997 146

133330 156

146663 137

159996 155

173329 158

186662 154

199995 151

Degree of parallelism: 32

13333 136

26666 136

39999 128

53332 131

66665 113

79998 150

93331 131

106664 151

119997 156

133330 147

146663 145

159996 143

173329 161

186662 169

199995 152

Degree of parallelism: 64

13333 134

26666 133

39999 125

53332 114

66665 135

79998 131

93331 119

106664 153

119997 161

133330 135

146663 153

159996 152

173329 155

186662 154

199995 148