Assignment 5

Fengyi Zhang SEC02 NeuId:001564247

**Task:**

1. Task: 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.

There is a *Main* class and the *ParSort* class in the *sort.par* package of the INFO6205 repository. The *Main* class can be used as is but the *ParSort* class needs to be implemented where you see "TODO..." [it turns out that these TODOs are already implemented].

Unless you have a good reason not to, you should just go along with the Java8-style future implementations provided for you in the class repository.

You must prepare a report that shows the results of your experiments and draws a conclusion (or more) about the efficacy of this method of parallelizing sort. Your 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.

**Answers:**

**Output:**

Array length = 1000000

**表格

描述已自动生成**

Array length = 2000000

**表格

描述已自动生成**

Array length = 4000000

**表格

描述已自动生成**

**Conclusion1:**

**I’ve set fixed number of thread pool and try to find the relationship of the ratio of cutoff to array size. I set the base as 0.02 and increase 0.02 each step. I’ve change the array size as 1000000, 2000000, 4000000. For each size, I’ve tested all different values of cutoff. The result graph as listed below. 图表, 折线图

描述已自动生成**

**The graph shows different time consumptions of different ratios of cutoff to array size. For the ratios below 0.16, the time consumptions go down as ratio increases. For the ratios over 0.16, the time consumptions go high as ratio increases. Time moves to stable when the ratio is over 1. So when the ratio is 0.16, time consumption is least.**

**Conclusion2:**

**I set the fixed array size and cutoff, and try to draw a conclusion by increasing number of threads and observe how time consumption differs. As it shows below, before it reaches 16, time consumption keep decreasing because more threads are involving calculation. After number of threads moves beyond 16, the time becomes stable because my cpu is 8 core 16 threads.图表

描述已自动生成**