Chenghuan Li (NUID: 001069554)

INFO 6205

Program Structures & Algorithms

Fall 2020

Assignment 5

- **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.

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).

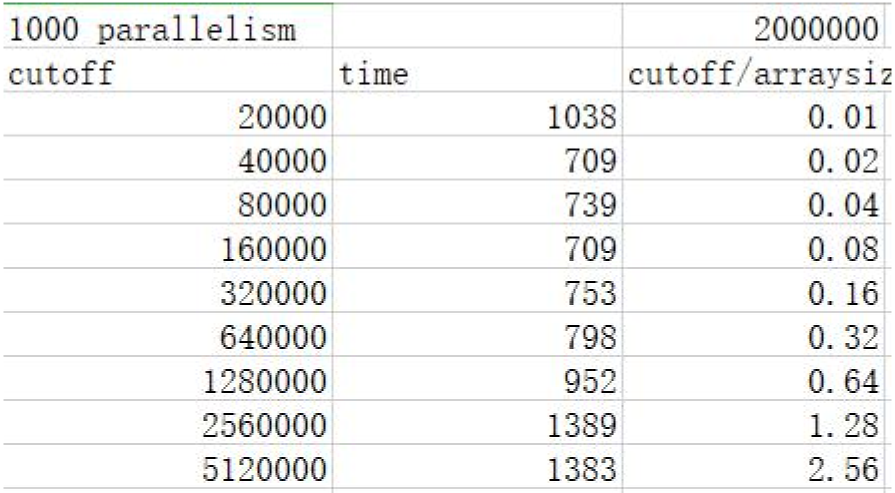
An appropriate combination of these.

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..."

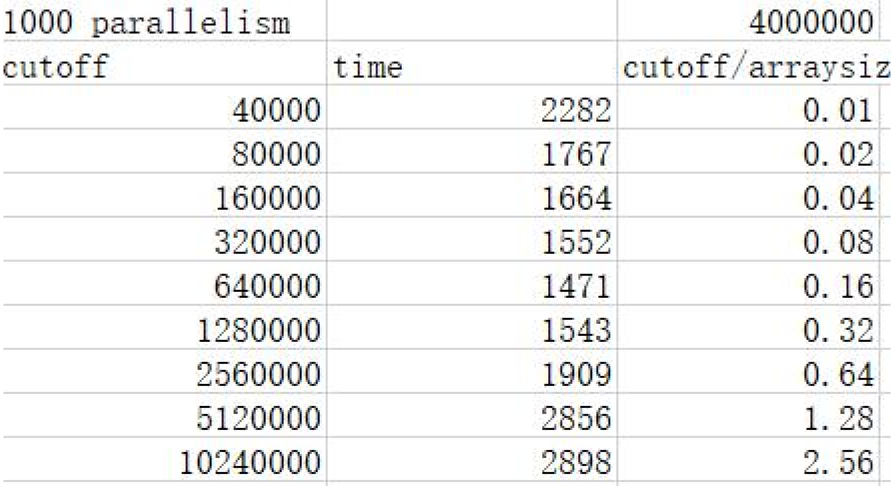
Unless you have a good reason not to, you should just go along with the Java8-style future implementations provided for you in the class repository.Assignment Parallel Sort.pdf

Actions

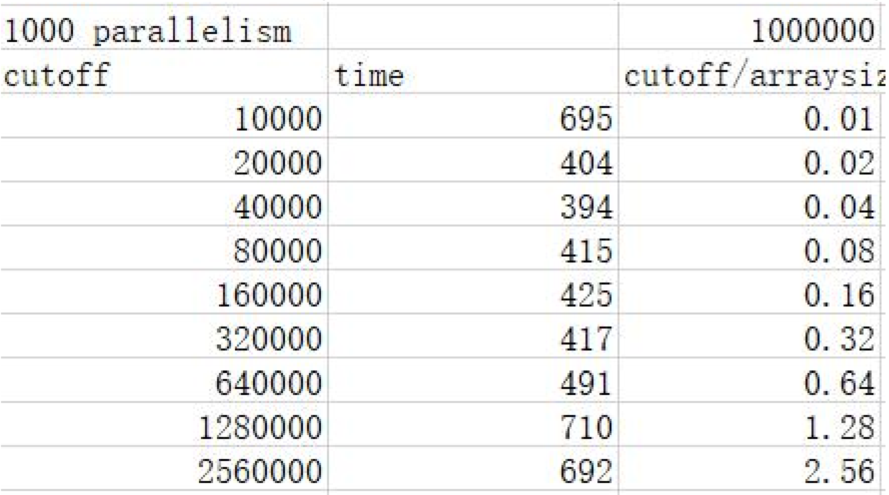
- **Output**



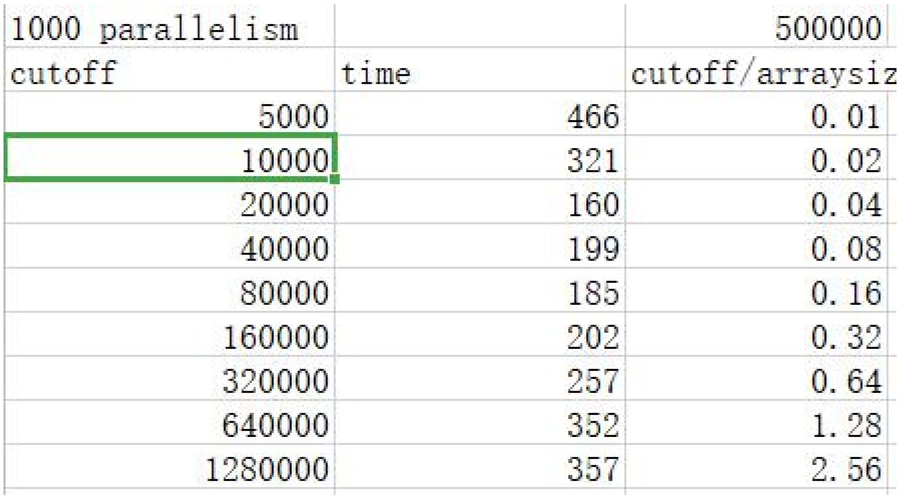
1000 parallelism		2000000
cutoff	time	cutoff/arraysiz
20000	1038	0.01
40000	709	0.02
80000	739	0.04
160000	709	0.08
320000	753	0.16
640000	798	0.32
1280000	952	0.64
2560000	1389	1.28
5120000	1383	2.56



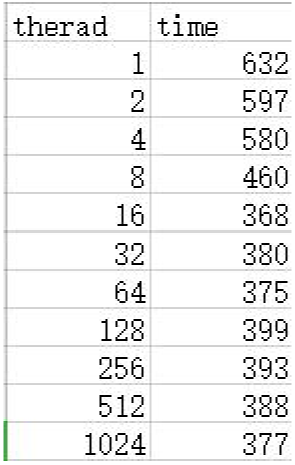
1000 parallelism		4000000
cutoff	time	cutoff/arraysiz
40000	2282	0.01
80000	1767	0.02
160000	1664	0.04
320000	1552	0.08
640000	1471	0.16
1280000	1543	0.32
2560000	1909	0.64
5120000	2856	1.28
10240000	2898	2.56



1000 parallelism		1000000
cutoff	time	cutoff/arraysiz
10000	695	0.01
20000	404	0.02
40000	394	0.04
80000	415	0.08
160000	425	0.16
320000	417	0.32
640000	491	0.64
1280000	710	1.28
2560000	692	2.56



1000 parallelism		500000
cutoff	time	cutoff/arraysiz
5000	466	0.01
10000	321	0.02
20000	160	0.04
40000	199	0.08
80000	185	0.16
160000	202	0.32
320000	257	0.64
640000	352	1.28
1280000	357	2.56



therad	time
1	632
2	597
4	580
8	460
16	368
32	380
64	375
128	399
256	393
512	388
1024	377

- **Relationship conclusion**

1,When the ratio of cutoff and arraysize near 16%, The sort program have the best efficacy.

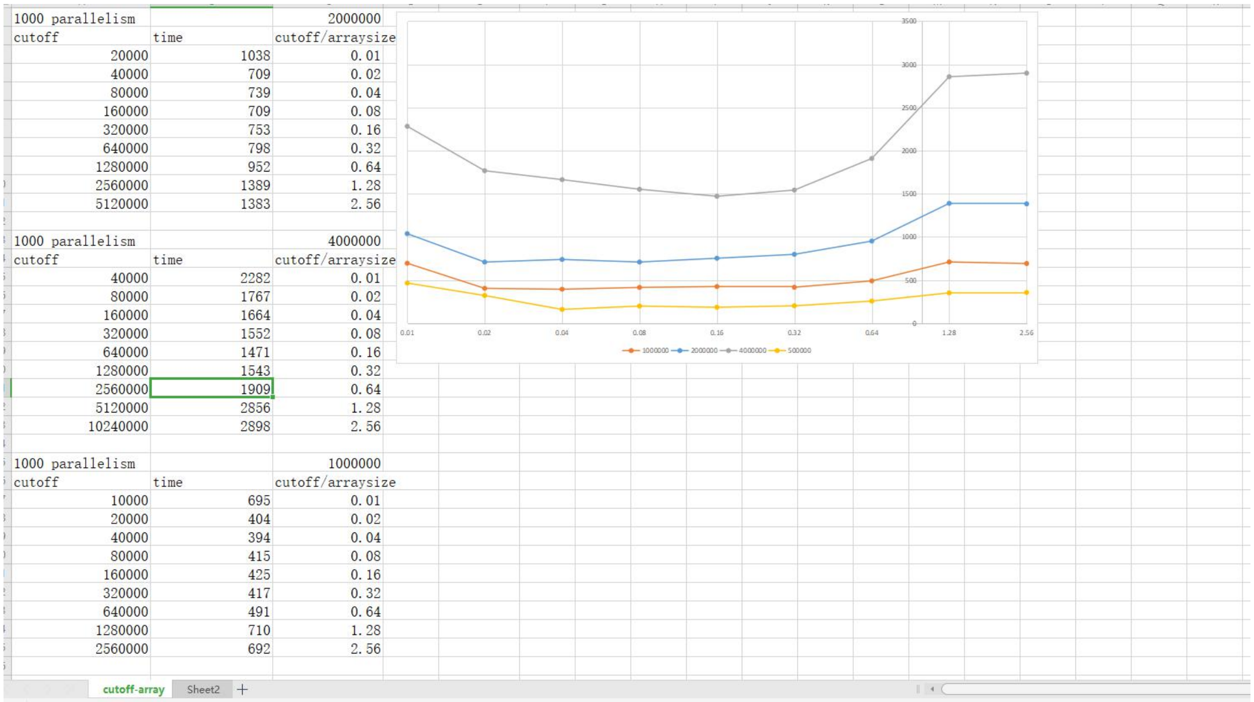
2, When the n(Thread)>16 the method of parallelizing sort have the best efficacy

- **Evidence to support relationship**

I did two experiments.

First, I fixed the number of thread pools, and then changed the sizes of cutoff and arraysize to see when the most appropriate cutoff value was obtained. Because the basis of this sort is mergesort. So the value of cutoff is related to log n. So the cutoff value I take every time follows the rule of powers of 2.

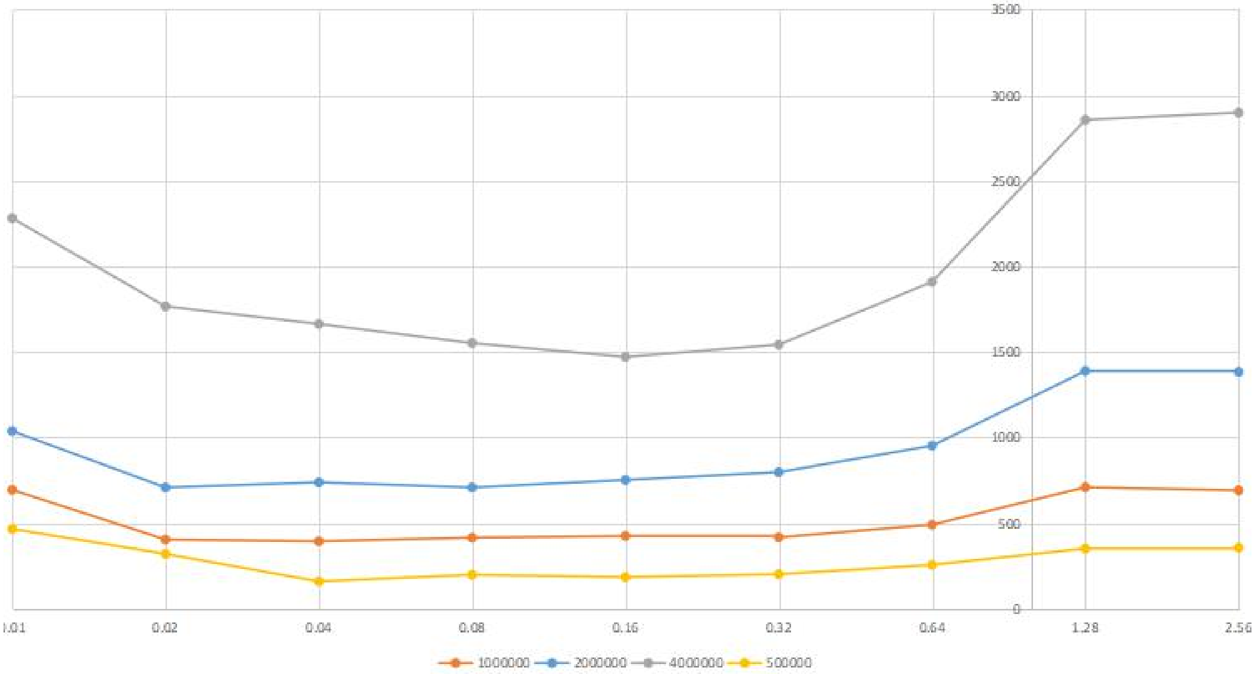
I defined the number of thread pools as 1000, and then I set 4 different arraysize 500000,1000000,2000000 and 4000000. For each size, I test different cutoff values. The result is as follow.



1000 parallelism		2000000
cutoff	time	cutoff/arraysize
20000	1038	0.01
40000	709	0.02
80000	739	0.04
160000	709	0.08
320000	753	0.16
640000	798	0.32
1280000	952	0.64
2560000	1389	1.28
5120000	1383	2.56

1000 parallelism		4000000
cutoff	time	cutoff/arraysize
40000	2282	0.01
80000	1767	0.02
160000	1664	0.04
320000	1552	0.08
640000	1471	0.16
1280000	1543	0.32
2560000	1909	0.64
5120000	2856	1.28
10240000	2898	2.56

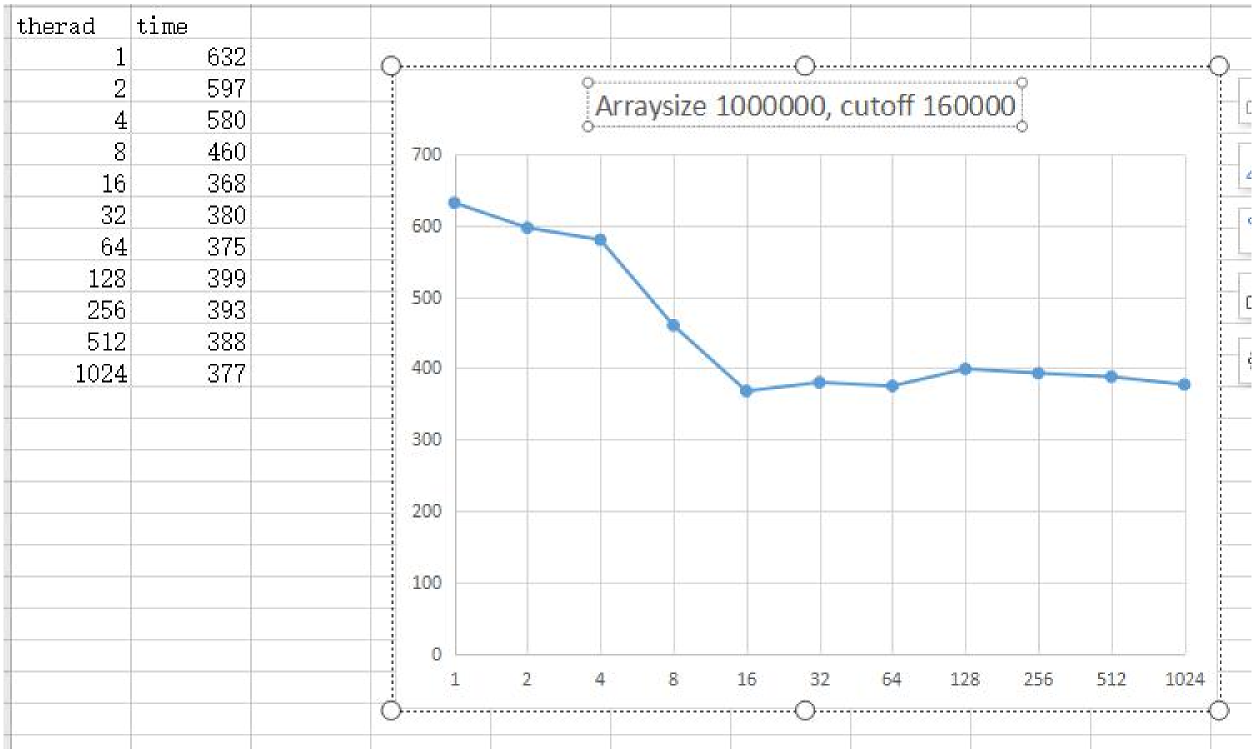
1000 parallelism		1000000
cutoff	time	cutoff/arraysize
10000	695	0.01
20000	404	0.02
40000	394	0.04
80000	415	0.08
160000	425	0.16
320000	417	0.32
640000	491	0.64
1280000	710	1.28
2560000	692	2.56



In this graph, we can see that the time consumed is different for different ratios of arraysize and cutoff. When the value of cutoff is too small, the time consumed is not the minimum. When the value of cutoff is greater than the value of arraysize, that is, when the ratio is greater than 1, the merging algorithm loses its meaning, so it takes a long time. We find that when the ratio is about 0.16, the whole algorithm consumes the least time.

So in the most efficient case. $\frac{cutoff}{Arraysize} \approx 16\%$

Second, I fixed the values of Arraysize and Cutoff and changed the size of the ForkJoinPool . As shown in the figure, the efficiency is the highest when the number of threads is greater than 16.



therad	time
1	632
2	597
4	580
8	460
16	368
32	380
64	375
128	399
256	393
512	388
1024	377