

Program Structures & Algorithms

Spring 2022

Assignment – 4

Name – Jayesh Kumar Khattar

NUID – 001568947

Task –

Please see the presentation on *Assignment on Parallel Sorting* under the *Exams. etc.* module.

Your task is to 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.

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.
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.

Output –

Output of the above code is shown in the below picture. We have executed experiments with different threads and array sizes, and it shows the following output –

The screenshots show the IntelliJ IDEA interface with two terminal panes displaying the execution results of a Java program. The program prints the size of the array and the number of threads used along with their cutoff values and execution times.

Left Screenshot (Array Size 2,000,000):

```
INFO6205-Spring2022 2 > src > main > java > edu > neu > coe > info6205 > sort > par  
INFO6205-Spring2022 2 > src > main > java > edu > neu > coe > info6205 > sort > par  
Run: Main  
Project Commit Pull Requests Bookmarks Structure  
/Library/Java/JavaVirtualMachines/jdk-17.0.2.jdk/Content  
Size of Array here is: 2000000  
#threads 1 | cutoff: 510000 | Time:1374 ms  
#threads 1 | cutoff: 610000 | Time:1247 ms  
#threads 1 | cutoff: 710000 | Time:1262 ms  
#threads 1 | cutoff: 810000 | Time:1247 ms  
#threads 1 | cutoff: 910000 | Time:1240 ms  
#threads 2 | cutoff: 510000 | Time:1117 ms  
#threads 2 | cutoff: 610000 | Time:1120 ms  
#threads 2 | cutoff: 710000 | Time:1125 ms  
#threads 2 | cutoff: 810000 | Time:1130 ms  
#threads 2 | cutoff: 910000 | Time:1153 ms  
#threads 4 | cutoff: 510000 | Time:882 ms  
#threads 4 | cutoff: 610000 | Time:896 ms  
#threads 4 | cutoff: 710000 | Time:887 ms  
#threads 4 | cutoff: 810000 | Time:867 ms  
#threads 4 | cutoff: 910000 | Time:878 ms  
#threads 8 | cutoff: 510000 | Time:606 ms  
#threads 8 | cutoff: 610000 | Time:605 ms  
#threads 8 | cutoff: 710000 | Time:605 ms  
#threads 8 | cutoff: 810000 | Time:604 ms  
#threads 8 | cutoff: 910000 | Time:607 ms  
#threads 16 | cutoff: 510000 | Time:607 ms  
#threads 16 | cutoff: 610000 | Time:607 ms  
#threads 16 | cutoff: 710000 | Time:604 ms  
#threads 16 | cutoff: 810000 | Time:605 ms  
#threads 16 | cutoff: 910000 | Time:605 ms  
Size of Array here is: 4000000  
#threads 1 | cutoff: 510000 | Time:2176 ms
```

Right Screenshot (Array Sizes 4,000,000 and 12,000,000):

```
INFO6205-Spring2022 2 > src > main > java > edu > neu > coe > info6205 > sort > par  
INFO6205-Spring2022 2 > src > main > java > edu > neu > coe > info6205 > sort > par  
Run: Main  
Project Commit Pull Requests Bookmarks Structure  
#threads 10 | cutoff: 710000 | Time:1003 ms  
Size of Array here is: 4000000  
#threads 1 | cutoff: 510000 | Time:2176 ms  
#threads 1 | cutoff: 610000 | Time:2187 ms  
#threads 1 | cutoff: 710000 | Time:2164 ms  
#threads 1 | cutoff: 810000 | Time:2135 ms  
#threads 1 | cutoff: 910000 | Time:2002 ms  
#threads 2 | cutoff: 510000 | Time:1936 ms  
#threads 2 | cutoff: 610000 | Time:1735 ms  
#threads 2 | cutoff: 710000 | Time:1825 ms  
#threads 2 | cutoff: 810000 | Time:1781 ms  
#threads 2 | cutoff: 910000 | Time:1841 ms  
#threads 4 | cutoff: 510000 | Time:2101 ms  
#threads 4 | cutoff: 610000 | Time:2018 ms  
#threads 4 | cutoff: 710000 | Time:2050 ms  
#threads 4 | cutoff: 810000 | Time:2004 ms  
#threads 4 | cutoff: 910000 | Time:2039 ms  
#threads 8 | cutoff: 510000 | Time:1599 ms  
#threads 8 | cutoff: 610000 | Time:1606 ms  
#threads 8 | cutoff: 710000 | Time:1608 ms  
#threads 8 | cutoff: 810000 | Time:1584 ms  
#threads 8 | cutoff: 910000 | Time:1601 ms  
#threads 16 | cutoff: 510000 | Time:1114 ms  
#threads 16 | cutoff: 610000 | Time:1106 ms  
#threads 16 | cutoff: 710000 | Time:1112 ms  
#threads 16 | cutoff: 810000 | Time:1113 ms  
#threads 16 | cutoff: 910000 | Time:1105 ms  
Size of Array here is: 12000000  
#threads 1 | cutoff: 510000 | Time:4875 ms  
#threads 1 | cutoff: 610000 | Time:4482 ms  
Download pre-built shared indexes: Reduce the indexing time and CPU load with pre-built Maven index
```

```

INFO6205-Spring2022 2 > src > main > java > edu > neu > coe > info6205 > sort > pa
Commit to mas...
UF_HWQUPC.java × BubbleSort.java × Main.java

Run: Main ×
Size of Array here is: 12000000
#threads 1 | cutoff: 510000 | Time:4875 ms
#threads 1 | cutoff: 610000 | Time:4482 ms
#threads 1 | cutoff: 710000 | Time:4664 ms
#threads 1 | cutoff: 810000 | Time:5224 ms
#threads 1 | cutoff: 910000 | Time:5005 ms
#threads 2 | cutoff: 510000 | Time:4825 ms
#threads 2 | cutoff: 610000 | Time:4328 ms
#threads 2 | cutoff: 710000 | Time:4470 ms
#threads 2 | cutoff: 810000 | Time:4397 ms
#threads 2 | cutoff: 910000 | Time:4409 ms
#threads 4 | cutoff: 510000 | Time:4540 ms
#threads 4 | cutoff: 610000 | Time:4233 ms
#threads 4 | cutoff: 710000 | Time:4638 ms
#threads 4 | cutoff: 810000 | Time:3833 ms
#threads 4 | cutoff: 910000 | Time:3933 ms
#threads 8 | cutoff: 510000 | Time:4382 ms
#threads 8 | cutoff: 610000 | Time:3830 ms
#threads 8 | cutoff: 710000 | Time:3952 ms
#threads 8 | cutoff: 810000 | Time:3549 ms
#threads 8 | cutoff: 910000 | Time:3586 ms
#threads 16 | cutoff: 510000 | Time:4747 ms
#threads 16 | cutoff: 610000 | Time:3884 ms
#threads 16 | cutoff: 710000 | Time:4589 ms
#threads 16 | cutoff: 810000 | Time:3783 ms
#threads 16 | cutoff: 910000 | Time:3851 ms
Size of Array here is: 48000000
#threads 1 | cutoff: 510000 | Time:20733 ms
#threads 1 | cutoff: 610000 | Time:16817 ms

```

```

INFO6205-Spring2022 2 > src > main > java > edu > neu > coe > info6205 > sort > pa
Commit to mas...
UF_HWQUPC.java × BubbleSort.java × Main.java

Run: Main ×
#threads 16 | cutoff: 810000 | Time:3783 ms
#threads 16 | cutoff: 910000 | Time:3851 ms
Size of Array here is: 48000000
#threads 1 | cutoff: 510000 | Time:20733 ms
#threads 1 | cutoff: 610000 | Time:16817 ms
#threads 1 | cutoff: 710000 | Time:15732 ms
#threads 1 | cutoff: 810000 | Time:16108 ms
#threads 1 | cutoff: 910000 | Time:18820 ms
#threads 2 | cutoff: 510000 | Time:17881 ms
#threads 2 | cutoff: 610000 | Time:16366 ms
#threads 2 | cutoff: 710000 | Time:16090 ms
#threads 2 | cutoff: 810000 | Time:15358 ms
#threads 2 | cutoff: 910000 | Time:15336 ms
#threads 4 | cutoff: 510000 | Time:16909 ms
#threads 4 | cutoff: 610000 | Time:15784 ms
#threads 4 | cutoff: 710000 | Time:15522 ms
#threads 4 | cutoff: 810000 | Time:15422 ms
#threads 4 | cutoff: 910000 | Time:16445 ms
#threads 8 | cutoff: 510000 | Time:20206 ms
#threads 8 | cutoff: 610000 | Time:17689 ms
#threads 8 | cutoff: 710000 | Time:16392 ms
#threads 8 | cutoff: 810000 | Time:15028 ms
#threads 8 | cutoff: 910000 | Time:15233 ms
#threads 16 | cutoff: 510000 | Time:19170 ms
#threads 16 | cutoff: 610000 | Time:18127 ms
#threads 16 | cutoff: 710000 | Time:15783 ms
#threads 16 | cutoff: 810000 | Time:15253 ms
#threads 16 | cutoff: 910000 | Time:15284 ms
Size of Array here is: 24000000

```

Relationship Conclusion –

- The system used in this assignment is an 8 core CPU. Since we can utilize 2 threads per core at a time, so at max we can run 16 threads together.
- We can see that as the number of threads increases, the time taken to execute the system reduces.
- After running experiments with different threads, we can observe that the performance increase with more threads but with large values and large # of threads, it's not the same. With array size equal to 12 million and high cutoff, 8 threads perform better than 16 threads and array size equal to 24 million eventually 4 thread performs best with large values.

For maximum depth (d) and threads (t), we can observe that –

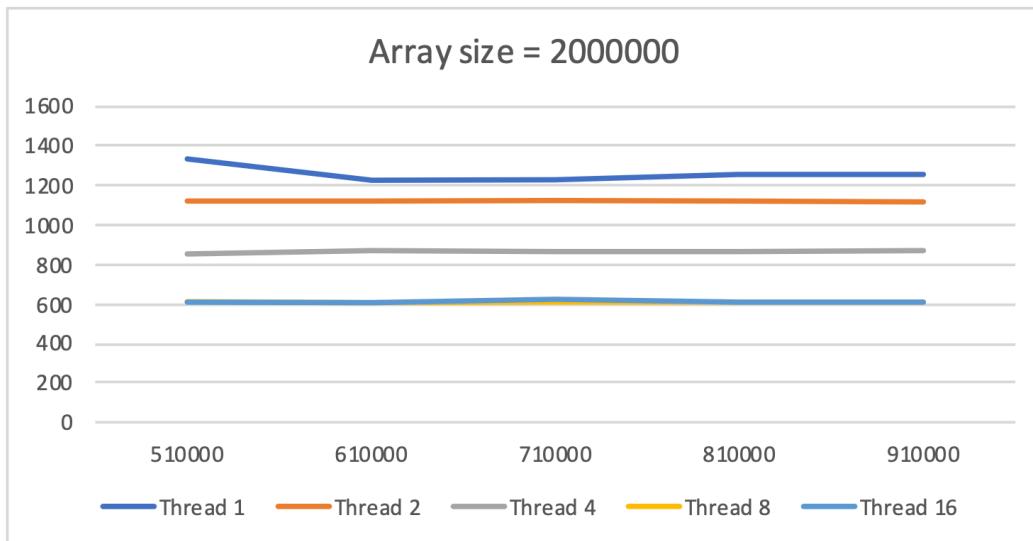
$$t = 2^d$$

Hence, maximum depth possible is $\lg \left(\frac{\text{array size}}{\text{cutoff}} \right)$

Evidence –

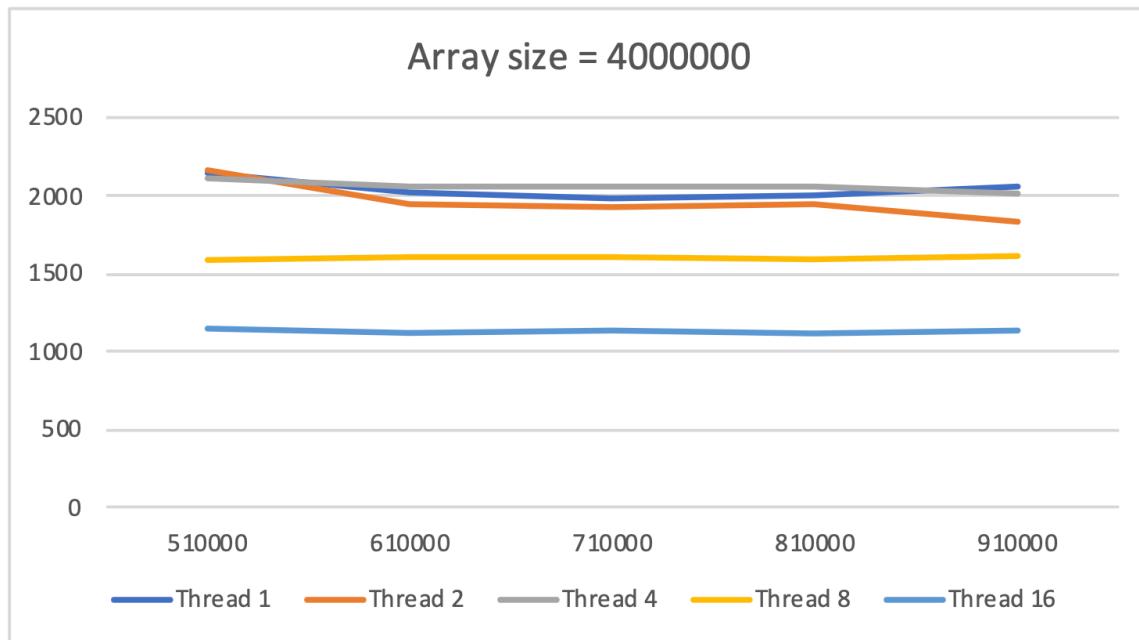
Array size	2000000
------------	---------

Cutoff	1 Thread	2 Threads	4 Threads	8 Threads	16 Threads
510000	1336	1120	854	612	610
610000	1228	1124	871	607	607
710000	1229	1125	870	610	624
810000	1254	1124	870	609	608
910000	1251	1118	871	608	612



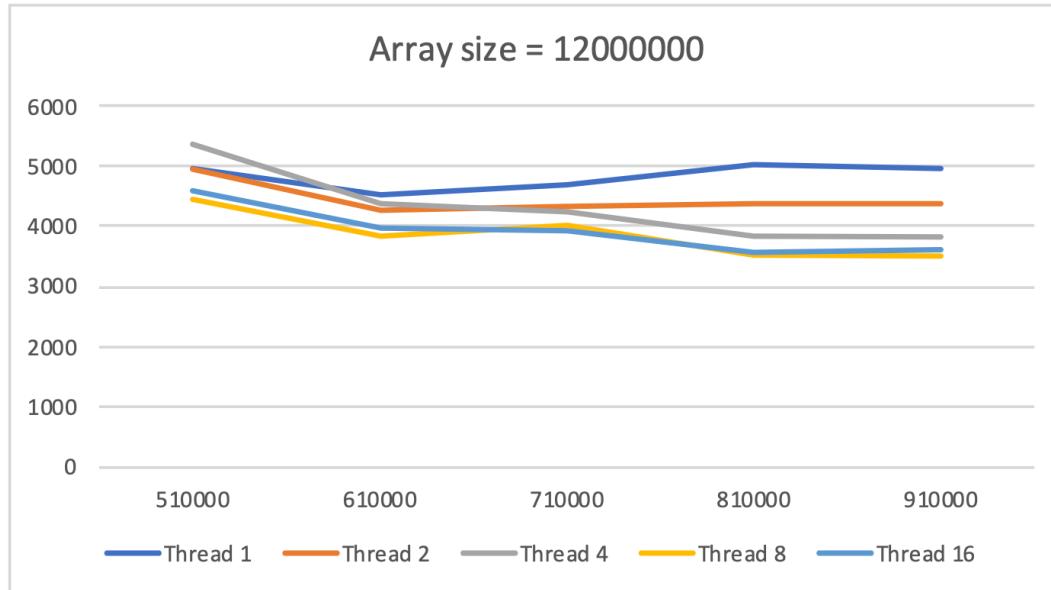
Array size	4000000
------------	---------

Cutoff	1 Thread	2 Threads	4 Threads	8 Threads	16 Threads
510000	2144	2162	2110	1594	1145
610000	2020	1953	2061	1599	1117
710000	1981	1932	2065	1609	1138
810000	2007	1936	2063	1590	1121
910000	2051	1831	2012	1611	1135



Array size **12000000**

Cutoff	1 Thread	2 Threads	4 Threads	8 Threads	16 Threads
510000	4944	4945	5358	4445	4587
610000	4520	4264	4388	3835	3983
710000	4709	4341	4259	4035	3932
810000	5020	4387	3828	3516	3566
910000	4966	4370	3822	3505	3610



Array size **48000000**

Cutoff	1 Thread	2 Threads	4 Threads	8 Threads	16 Threads
510000	18471	19618	17214	17207	18812
610000	17461	16498	15892	15931	16193
710000	17329	16132	15448	15226	17052
810000	15821	16199	15190	15753	16131
910000	17151	16063	15201	15977	16016

