Program Structures & Algorithms
Spring 2022
Assignment No. 4 (Parallel Sorting)

Name: Sri Vyshnavi Kotha
(NUID): 002985810

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.

- 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.

Relationship Conclusion

The experiment below was conducted using a cut off range of 51000-75000 and threads ranging from 2 to 32. And it is observed that 51000 is the most efficient for 32 threads. It is working best when the cut off value is lowest and Threads are more.

Output screenshot

```
Size of the Array : 0
Degree of parallelism: 2
cutoff: 510000      10times Time:0ms
cutoff: 520000      10times Time:0ms
cutoff: 530000      10times Time:0ms
cutoff: 540000      10times Time:0ms
cutoff: 550000      10times Time:0ms
cutoff: 560000      10times Time:0ms
cutoff: 570000      10times Time:0ms
cutoff: 580000      10times Time:0ms
cutoff: 590000      10times Time:0ms
cutoff: 600000      10times Time:0ms
cutoff: 610000      10times Time:0ms
cutoff: 620000      10times Time:0ms
cutoff: 630000      10times Time:0ms
cutoff: 640000      10times Time:0ms
cutoff: 650000      10times Time:0ms
cutoff: 660000      10times Time:0ms
cutoff: 670000      10times Time:0ms
cutoff: 680000      10times Time:0ms
cutoff: 690000      10times Time:0ms
cutoff: 700000      10times Time:0ms
cutoff: 710000      10times Time:0ms
cutoff: 720000      10times Time:0ms
cutoff: 730000      10times Time:0ms
cutoff: 740000      10times Time:0ms
cutoff: 750000      10times Time:0ms
```

```
cutoff: 730000      10times Time:0ms
cutoff: 740000      10times Time:0ms
cutoff: 750000      10times Time:0ms
cutoff: 760000      10times Time:0ms
cutoff: 770000      10times Time:0ms
cutoff: 780000      10times Time:0ms
cutoff: 790000      10times Time:0ms
cutoff: 800000      10times Time:0ms
cutoff: 810000      10times Time:0ms
cutoff: 820000      10times Time:0ms
cutoff: 830000      10times Time:0ms
cutoff: 840000      10times Time:0ms
cutoff: 850000      10times Time:0ms
cutoff: 860000      10times Time:0ms
cutoff: 870000      10times Time:0ms
cutoff: 880000      10times Time:0ms
cutoff: 890000      10times Time:0ms
cutoff: 900000      10times Time:0ms
cutoff: 910000      10times Time:0ms
cutoff: 920000      10times Time:0ms
cutoff: 930000      10times Time:0ms
cutoff: 940000      10times Time:0ms
cutoff: 950000      10times Time:0ms
cutoff: 960000      10times Time:0ms
cutoff: 970000      10times Time:0ms
cutoff: 980000      10times Time:0ms
cutoff: 990000      10times Time:0ms
cutoff: 1000000     10times Time:0ms
```

```
Size of the Array : 2000
Degree of parallelism: 4
cutoff: 510000      10times Time:14ms
cutoff: 520000      10times Time:2ms
cutoff: 530000      10times Time:1ms
cutoff: 540000      10times Time:2ms
cutoff: 550000      10times Time:2ms
cutoff: 560000      10times Time:5ms
cutoff: 570000      10times Time:2ms
cutoff: 580000      10times Time:2ms
cutoff: 590000      10times Time:1ms
cutoff: 600000      10times Time:2ms
cutoff: 610000      10times Time:1ms
cutoff: 620000      10times Time:1ms
cutoff: 630000      10times Time:2ms
cutoff: 640000      10times Time:2ms
cutoff: 650000      10times Time:4ms
cutoff: 660000      10times Time:2ms
cutoff: 670000      10times Time:1ms
cutoff: 680000      10times Time:1ms
cutoff: 690000      10times Time:1ms
cutoff: 700000      10times Time:1ms
cutoff: 710000      10times Time:2ms
cutoff: 720000      10times Time:9ms
cutoff: 730000      10times Time:4ms
cutoff: 740000      10times Time:7ms
cutoff: 750000      10times Time:3ms
cutoff: 760000      10times Time:1ms
```

```
cutoff: 770000      10times Time:2ms
cutoff: 780000      10times Time:1ms
cutoff: 790000      10times Time:3ms
cutoff: 800000      10times Time:2ms
cutoff: 810000      10times Time:1ms
cutoff: 820000      10times Time:1ms
cutoff: 830000      10times Time:2ms
cutoff: 840000      10times Time:2ms
cutoff: 850000      10times Time:1ms
cutoff: 860000      10times Time:2ms
cutoff: 870000      10times Time:3ms
cutoff: 880000      10times Time:1ms
cutoff: 890000      10times Time:1ms
cutoff: 900000      10times Time:1ms
cutoff: 910000      10times Time:1ms
cutoff: 920000      10times Time:1ms
cutoff: 930000      10times Time:1ms
cutoff: 940000      10times Time:1ms
cutoff: 950000      10times Time:1ms
cutoff: 960000      10times Time:1ms
cutoff: 970000      10times Time:1ms
cutoff: 980000      10times Time:1ms
cutoff: 990000      10times Time:1ms
cutoff: 1000000     10times Time:1ms
```

```
Size of the Array : 2000
Degree of parallelism: 8
cutoff: 510000      10times Time:1ms
cutoff: 520000      10times Time:0ms
cutoff: 530000      10times Time:1ms
cutoff: 540000      10times Time:1ms
cutoff: 550000      10times Time:1ms
cutoff: 560000      10times Time:1ms
cutoff: 570000      10times Time:1ms
cutoff: 580000      10times Time:0ms
cutoff: 590000      10times Time:1ms
cutoff: 600000      10times Time:1ms
cutoff: 610000      10times Time:1ms
cutoff: 620000      10times Time:1ms
cutoff: 630000      10times Time:1ms
cutoff: 640000      10times Time:1ms
cutoff: 650000      10times Time:1ms
cutoff: 660000      10times Time:1ms
cutoff: 670000      10times Time:0ms
cutoff: 680000      10times Time:1ms
cutoff: 690000      10times Time:1ms
cutoff: 700000      10times Time:1ms
cutoff: 710000      10times Time:1ms
cutoff: 720000      10times Time:1ms
cutoff: 730000      10times Time:1ms
cutoff: 740000      10times Time:1ms
cutoff: 750000      10times Time:1ms
cutoff: 760000      10times Time:1ms
cutoff: 870000      10times Time:1ms
cutoff: 880000      10times Time:1ms
cutoff: 890000      10times Time:0ms
cutoff: 900000      10times Time:1ms
cutoff: 910000      10times Time:1ms
cutoff: 920000      10times Time:1ms
cutoff: 930000      10times Time:1ms
cutoff: 940000      10times Time:1ms
cutoff: 950000      10times Time:1ms
cutoff: 960000      10times Time:0ms
cutoff: 970000      10times Time:1ms
cutoff: 980000      10times Time:1ms
cutoff: 990000      10times Time:4ms
cutoff: 1000000     10times Time:3ms
Size of the Array : 2000
Degree of parallelism: 16
cutoff: 510000      10times Time:2ms
cutoff: 520000      10times Time:2ms
cutoff: 530000      10times Time:1ms
cutoff: 540000      10times Time:1ms
cutoff: 550000      10times Time:2ms
cutoff: 560000      10times Time:1ms
cutoff: 570000      10times Time:2ms
cutoff: 580000      10times Time:1ms
cutoff: 590000      10times Time:2ms
cutoff: 600000      10times Time:1ms
cutoff: 610000      10times Time:2ms
cutoff: 620000      10times Time:1ms
```

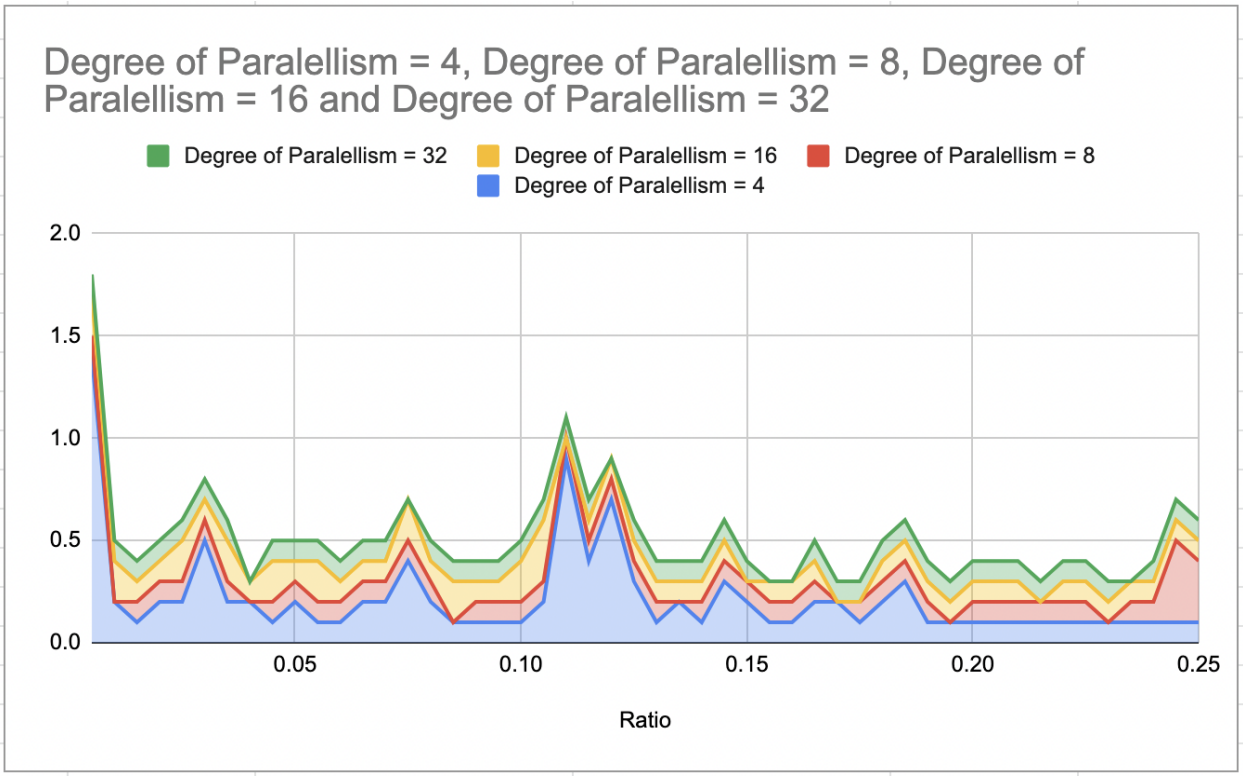
Unit tests result

Ratio	Degree of Paralellism = 4	Degree of Paralellism = 8	Degree of Paralellism = 16	Degree of Paralellism = 32
0.005	**1.4**	**0.1**	**0.2**	**0.1**
0.01	0.2	0	0.2	0.1
0.015	0.1	0.1	0.1	0.1
0.02	0.2	0.1	0.1	0.1
0.025	0.2	0.1	0.2	0.1

0.03	0.5	0.1	0.1	0.1
0.035	0.2	0.1	0.2	0.1
0.04	0.2	0	0.1	0
0.045	0.1	0.1	0.2	0.1
0.05	0.2	0.1	0.1	0.1
0.055	0.1	0.1	0.2	0.1
0.06	0.1	0.1	0.1	0.1
0.065	0.2	0.1	0.1	0.1
0.07	0.2	0.1	0.1	0.1
0.075	0.4	0.1	0.2	0
0.08	0.2	0.1	0.1	0.1
0.085	0.1	0	0.2	0.1
0.09	0.1	0.1	0.1	0.1
0.095	0.1	0.1	0.1	0.1
0.1	0.1	0.1	0.2	0.1
0.105	0.2	0.1	0.3	0.1
0.11	0.9	0.1	0	0.1
0.115	0.4	0.1	0.1	0.1
0.12	0.7	0.1	0.1	0
0.125	0.3	0.1	0.1	0.1
0.13	0.1	0.1	0.1	0.1
0.135	0.2	0	0.1	0.1
0.14	0.1	0.1	0.1	0.1
0.145	0.3	0.1	0.1	0.1
0.15	0.2	0.1	0	0.1
0.155	0.1	0.1	0.1	0
0.16	0.1	0.1	0.1	0
0.165	0.2	0.1	0.1	0.1
0.17	0.2	0	0	0.1
0.175	0.1	0.1	0	0.1
0.18	0.2	0.1	0.1	0.1
0.185	0.3	0.1	0.1	0.1
0.19	0.1	0.1	0.1	0.1
0.195	0.1	0	0.1	0.1
0.2	0.1	0.1	0.1	0.1
0.205	0.1	0.1	0.1	0.1
0.21	0.1	0.1	0.1	0.1
0.215	0.1	0.1	0	0.1
0.22	0.1	0.1	0.1	0.1

0.225	0.1	0.1	0.1	0.1
0.23	0.1	0	0.1	0.1
0.235	0.1	0.1	0.1	0
0.24	0.1	0.1	0.1	0.1
0.245	0.1	0.4	0.1	0.1
0.25	0.1	0.3	0.1	0.1

Evidence / Graph



Code

Main.java

```
public static void main(String[] args) {
    processArgs(args);
    int initialNumberOfThreads = 2;
    int initialSize = 100000;
    Random random = new Random();
    int arraySize=0;
    for (int threads = initialNumberOfThreads; threads <= 32; threads = 2 *
threads, arraySize = 2
            * initialSize) {
        int[] array = new int[arraySize];
```

```
        System.out.println("Size of the Array : " + array.length);
        ArrayList<Long> timeList = new ArrayList<>();
        ParSort.poolThreads = new ForkJoinPool(threads);
        System.out.println("Degree of parallelism: " +
ParSort.poolThreads.getParallelism());
        for (int j = 50; j < 100; j++) {
            ParSort.cutoff = 10000 * (j + 1);
            // for (int i = 0; i < array.length; i++) array[i] =
random.nextInt(10000000);
            long time;
            long startTime = System.currentTimeMillis();
            for (int t = 0; t < 10; t++) {
                for (int i = 0; i < array.length; i++) array[i] =
random.nextInt(10000000);
                ParSort.sort(array, 0, array.length);
            }
            long endTime = System.currentTimeMillis();
            time = (endTime - startTime);
            timeList.add(time);


            System.out.println("cutoff : " + (ParSort.cutoff) + "\t\t10times
Time:" + time + "ms");
        }
        try {
            FileOutputStream fis = new
FileOutputStream("./src/result"+threads+".csv");
            OutputStreamWriter isr = new OutputStreamWriter(fis);
            BufferedWriter bw = new BufferedWriter(isr);
            int j = 0;
            for (long i : timeList) {
                String content = (double) 10000 * (j + 1) / 2000000 + "," +
(double) i / 10 + "\n";
                j++;
                bw.write(content);
                bw.flush();
            }
            bw.close();

        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

ParSort.Java

```
class ParSort {

   public static int cutoff = 1000;
   public static int threads = 2;
   public static ForkJoinPool poolThreads ;

   public static void sort(int[] array, int from, int to) {
       if (to - from < cutoff) Arrays.sort(array, from, to);
       else {
           // FIXME next few lines should be removed from public repo.
           CompletableFuture<int[]> parsort1 = parsort(array, from, from + (to
- from) / 2); // TO IMPLEMENT
           CompletableFuture<int[]> parsort2 = parsort(array, from + (to -
from) / 2, to); // TO IMPLEMENT
           CompletableFuture<int[]> parsort = parsort1.thenCombine(parsort2,
(xs1, xs2) -> {
               int[] result = new int[xs1.length + xs2.length];
               // TO IMPLEMENT
               int i = 0;
               int j = 0;
               for (int k = 0; k < result.length; k++) {
                   if (i >= xs1.length) {
                       result[k] = xs2[j++];
                   } else if (j >= xs2.length) {
                       result[k] = xs1[i++];
                   } else if (xs2[j] < xs1[i]) {
                       result[k] = xs2[j++];
                   } else {
                       result[k] = xs1[i++];
                   }
               }
               return result;
           });

           parsort.whenComplete((result, throwable) ->
System.arraycopy(result, 0, array, from, result.length));
//             System.out.println("# threads: "+
ForkJoinPool.commonPool().getRunningThreadCount());
           parsort.join();
       }
   }

   private static CompletableFuture<int[]> parsort(int[] array, int from, int
to) {
       return CompletableFuture.supplyAsync(
               () -> {
```

```
                    int[] result = new int[to - from];
                    // TO IMPLEMENT
                    System.arraycopy(array, from, result, 0, result.length);
                    sort(result, 0, to - from);
                    return result;
                }, poolThreads
        );
    }
}
```