Program Structures & Algorithms
Fall 2021
Assignment No. 5

- **Task (List down the tasks performed in the Assignment)**
  - Implement a parallel sorting algorithm such that each partition of the array is sorted in parallel. Consider two different schemes for deciding whether to sort in parallel. Experiments should involve sorting arrays of sufficient size for the parallel sort to make a difference. Run with many different array sizes (they must be sufficiently large to make parallel sorting worthwhile, obviously) and different cutoff schemes.
  - A cutoff (defaults to, say, 1000) which will update according to the first argument in the command line when running. Experiment and come up with a good value for this cutoff. If there are fewer elements to sort than the cutoff, then use the system sort instead.
  - Recursion depth or the number of available threads. Using this determination, 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.

ParSort.java | Main.java | ParSortImprove.java | MainImprovement.java

```
19    public static void main(String[] args) {
20        processArgs(args);
21        int thread = 2;
22        while(thread < 128) {
23            ForkJoinPool pool = new ForkJoinPool(thread);
24            System.out.println("Degree of parallelism: " + pool.getParallelism());
25            //System.out.println("Degree of parallelism: " + ForkJoinPool.getCommonPoolParallelism());
26            Random random = new Random();
27            ArrayList<Long> timeList = new ArrayList<>();
28            for(int arraysize = 30000; arraysize < 2100000; arraysize *= 2)
29            {
30                System.out.println("Size of Array: " + arraysize);
31                int[] array = new int[arraysize];
32                for (int j = 50; j < 100; j++) {
33                    ParSort.cutoff = 10000 * (j + 1);
34                // for (int i = 0; i < array.length; i++) array[i] = random.nextInt(10000000);
35                    long time;
36                    long startTime = System.currentTimeMillis();
37                    for (int t = 0; t < 10; t++) {
38                        for (int i = 0; i < array.length; i++) array[i] = random.nextInt(10000000);
39                        ParSort.sort(array, 0, array.length);
40                    }
41                    long endTime = System.currentTimeMillis();
42                    time = (endTime - startTime);
43                    timeList.add(time);
44                    System.out.println("cutoff: " + (ParSort.cutoff) + "\t\t10times Time:" + time + "ms");
45                } try {
46                    FileOutputStream fis = new FileOutputStream("./src/result.csv");
47                    OutputStreamWriter isr = new OutputStreamWriter(fis);
48                    BufferedWriter bw = new BufferedWriter(isr);
49                    int j = 0;
50                    for (long i : timeList) {
51                        String content = (double) 10000 * (j + 1) / 2000000 + "," + (double) i / 10 + "\n";
52                        j++;
53                        bw.write(content);
54                        bw.flush();
55                    } bw.close();}
56                catch (IOException e) {
57                    e.printStackTrace();
58                }
59            }
60            thread *= 2;
61        }
62    }
```

- **Relationship Conclusion: (For ex: z = a * b)**

  I use the cutoff value from 510000 to 100000 (every10000) and use the loop to test with 7 array sizes from 30000 to 1920000 (times with 2). Also, I test each cutoff and array size with 6 different threads from 2 to 64 (powers of 2).

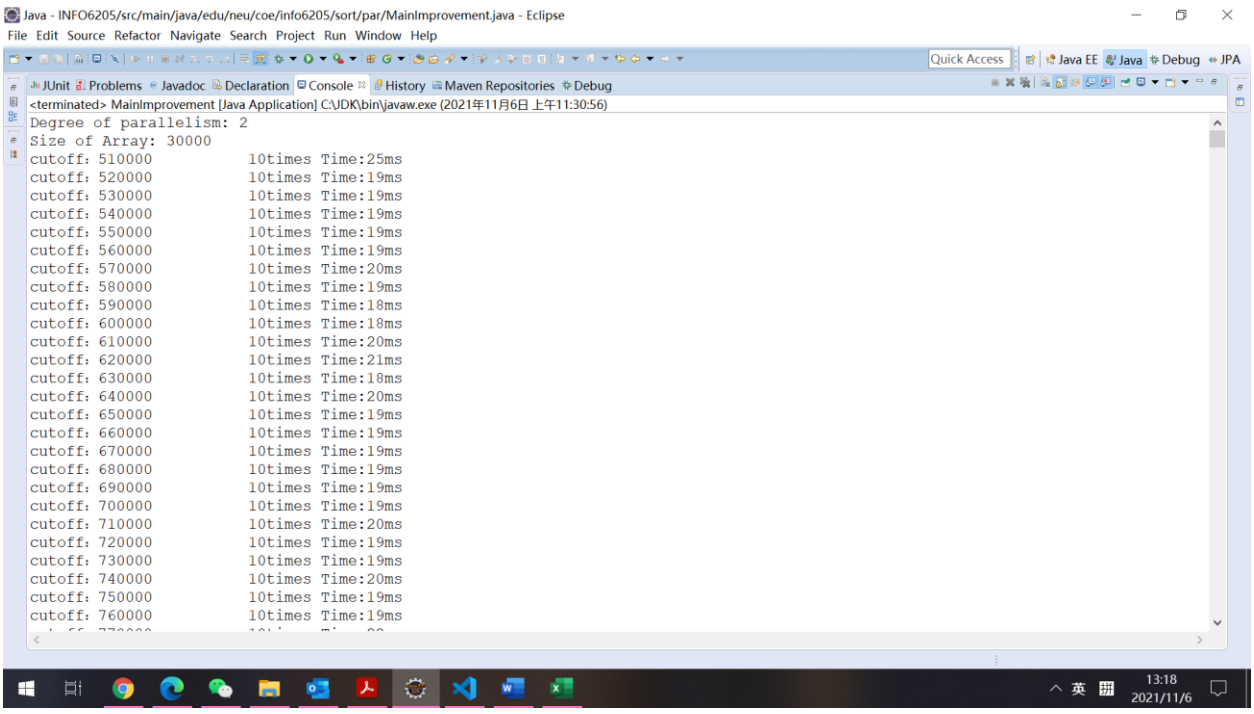
  I put all test result in the excel and make the chart of the cutoff and thread with different array size. From the observation of the chart, it can be seen that in general the thread value of 8, 16, 32, 64 performances worse than small thread, especially when the array size are small. The thread of 2 generally have better performance in my output.

  Also, we can see from the chart, the lines growth like fluctuating and floating up and down. The lower points are all round 53, 58, 63, 68, 73, 78, 83, 88, 93, 98 (*10000) in all threads values when array size is bigger than 240000. It means that in these cutoff vale are better choice.

  Besides, from the value in excel we can see that as the value of the array size increased by a factor of two, the processing time increased at a much slower rate, which means no linear relationship between the array size processed by the parallel sorting and the processing run time. When array size = 1920000 which is 64 times of 3000, the running time just 10 times bigger than the beginning, which indicated that parallel sorting performs better on big data scale.

- **Evidence to support the conclusion:**

  1. **Output (Snapshot of Code output in the terminal)**



(More output snapshots in the folder Assignment5)

2. Graphical Representation (Observations from experiments should be tabulated and analyzed by plotting graphs(usually in excel) to arrive on the relationship conclusion)

I drew 7 line charts of 7 different array sizes, and took the value of cutoff as the x-axis, the run time as the y-axis, and made lines with 6 different thread values in different colors.