Lan Gao (001568670)

Program Structures & Algorithms

Fall 2021

Assignment No. 5

- **Task (List down the tasks performed in the Assignment)**

  ParSort

```
*/
class ParSort {

    public static int cutoff = 1000;
    public static ForkJoinPool myPool = new ForkJoinPool( parallelism: 4);

    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, to: from + (to - from) / 2); // TO IMPLEMENT
            CompletableFuture<int[]> parsort2 = parsort(array, from: 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, srcPos: 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, destPos: 0, result.length);
                sort(result, from: 0, to: to - from);
                return result;
            }, myPool
    );

}
```

- **Relationship Conclusion:**

From the data sheets of this experiment, we can tell the quickest sort is under the situation of the number of thread is 16.

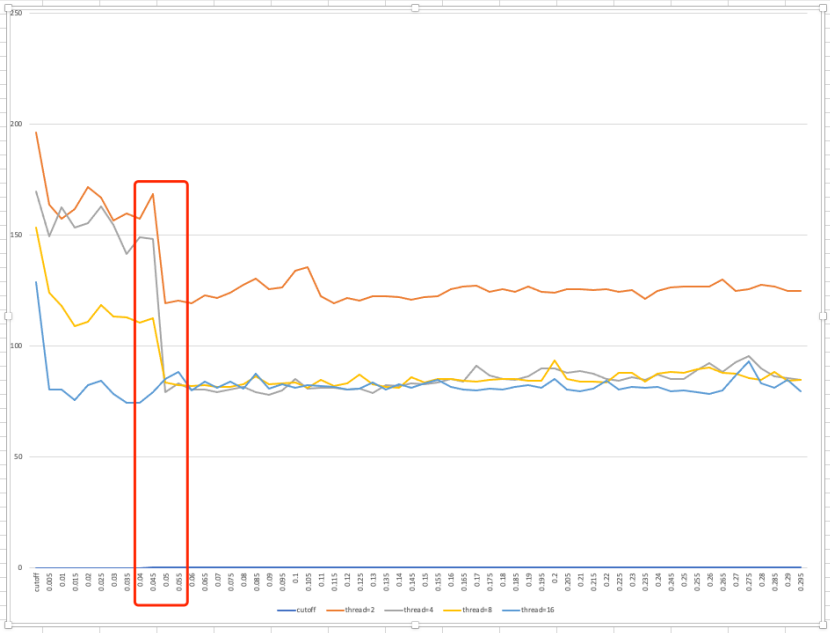
And as the array size grows larger, the time of sort goes up as well with the same number of thread.

Finally, when cutoff is 5.5%, there is the most efficient way.

- **Evidence to support the conclusion:**

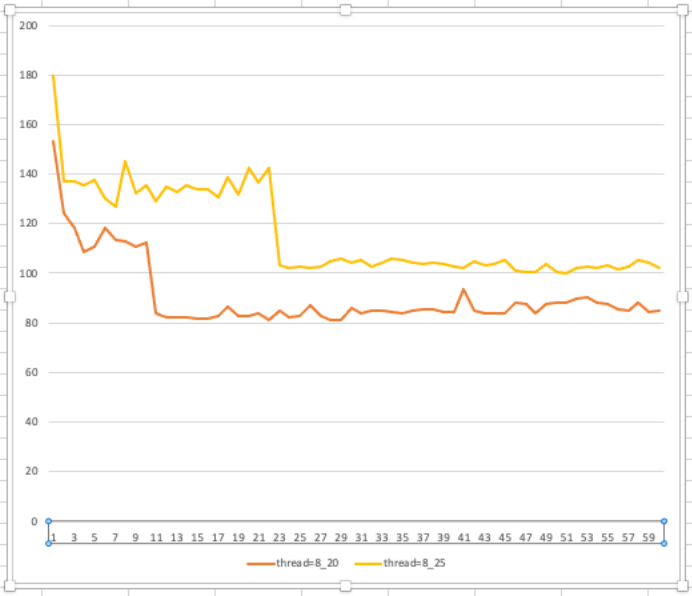
When array size is 2000000
The screenshot is under the next page

Array size=2000000				
cutoff	thread=2	thread=4	thread=8	thread=16
0.005	196.3	169.8	153.4	128.8
0.01	163.9	149.6	124	80.3
0.015	157.5	162.5	118.1	80.5
0.02	161.7	153.3	108.8	75.8
0.025	171.5	155.3	110.8	82.3
0.03	166.8	162.9	118.3	84.2
0.035	156.4	154.4	113.2	78.6
0.04	159.6	141.3	113.1	74.4
0.045	157.2	149	110.5	74.3
0.05	168.4	148.1	112.6	79
0.055	119.1	79.2	83.6	85.3
0.06	120.4	83	82.5	88.4
0.065	119.3	80.4	82	79.8
0.07	123	80.2	82.3	83.8
0.075	121.8	79.1	81.6	81.2
0.08	124.1	80.3	81.7	83.8
0.085	127.7	81.5	82.6	80.7
0.09	130.5	79.1	86.4	87.4
0.095	125.6	78.2	82.8	80.9
0.1	126.5	79.9	83	82.7
0.105	133.8	85.3	83.6	81.2
0.11	135.5	80.6	81.2	82.2
0.115	122.3	81	84.7	82
0.12	119.1	81.2	82	81.5
0.125	121.6	80.4	83	80.4
0.13	120.3	80.8	87.2	80.9
0.135	122.3	78.9	82.8	83.5
0.14	122.4	82.5	81.4	80.3
0.145	122	81.9	81.1	82.7
0.15	120.9	83.3	85.9	81
0.155	121.9	82.6	83.7	83.3
0.16	122.3	83.6	85	84.7
0.165	125.8	85.2	85.1	81.4
0.17	126.7	84.1	84.3	80.3
0.175	127.2	91.1	84	79.9
0.18	124.3	86.7	84.9	80.8
0.185	125.5	85.3	85.3	80.4
0.19	124.5	84.9	85.2	81.7
0.195	126.9	86.5	84.3	82.5
0.2	124.6	89.9	84.5	81.3
0.205	124.1	89.8	93.5	85
0.21	125.6	87.9	85.1	80.5
0.215	125.6	88.9	83.9	79.7
0.22	125.2	87.5	83.9	80.7
0.225	125.7	85.2	83.6	84.4
0.23	124.5	84.5	88.1	80.4
0.235	125.1	85.8	87.8	81.5
0.24	121.3	84.8	83.8	81
0.245	124.9	87.2	87.4	81.6
0.25	126.4	85.2	88.4	79.5
0.255	126.9	85	87.9	80.1
0.26	126.7	89.3	89.6	79.1
0.265	126.7	92.3	90.2	78.4
0.27	130	88.5	87.9	79.8
0.275	124.8	92.6	87.4	86.8
0.28	125.5	95.6	85.5	93.1



With different array size:
Thread = 8,
Array size = 2000000 & 25000000

cutoff	thread=8_2	cutoff	thread=8_25
0.005	153.4	0.004	179.3
0.01	124	0.008	137.2
0.015	118.1	0.012	137.1
0.02	108.8	0.016	135.7
0.025	110.8	0.02	137.7
0.03	118.3	0.024	130.2
0.035	113.2	0.028	126.6
0.04	113.1	0.032	145.2
0.045	110.5	0.036	132
0.05	112.6	0.04	135.5
0.055	83.6	0.044	129
0.06	82.5	0.048	135.1
0.065	82	0.052	132.6
0.07	82.3	0.056	135.4
0.075	81.6	0.06	134.1
0.08	81.7	0.064	134
0.085	82.6	0.068	130.6
0.09	86.4	0.072	138.5
0.095	82.8	0.076	131.6
0.1	83	0.08	142.4
0.105	83.6	0.084	136.6
0.11	81.2	0.088	142.2
0.115	84.7	0.092	103.4
0.12	82	0.096	102.2
0.125	83	0.1	102.7
0.13	87.2	0.104	102.4
0.135	82.8	0.108	102.6
0.14	81.4	0.112	104.8
0.145	81.1	0.116	105.7
0.15	85.9	0.12	104.5
0.155	83.7	0.124	105.1
0.16	85	0.128	102.6
0.165	85.1	0.132	104.3
0.17	84.3	0.136	105.8
0.175	84	0.14	105.3
0.18	84.9	0.144	104.5
0.185	85.3	0.148	103.7
0.19	85.2	0.152	104.1
0.195	84.3	0.156	103.7
0.2	84.5	0.16	102.9
0.205	93.5	0.164	102.2
0.21	85.1	0.168	105
0.215	83.9	0.172	103.4
0.22	83.9	0.176	103.7
0.225	83.6	0.18	105.4
0.23	88.1	0.184	101.2
0.235	87.8	0.188	100.4
0.24	83.8	0.192	100.5
0.245	87.4	0.196	104



	A	B	C	D	E
1	Array size=2000000				
2	cutoff	thread=2	thread=4	thread=8	thread=16
3	0.005	196.3	169.8	153.4	128.8
4	0.01	163.9	149.6	124	80.3
5	0.015	157.5	162.5	118.1	80.5
6	0.02	161.7	153.3	108.8	75.8
7	0.025	171.5	155.3	110.8	82.3
8	0.03	166.8	162.9	118.3	84.2
9	0.035	156.4	154.4	113.2	78.6
10	0.04	159.6	141.3	113.1	74.4
11	0.045	157.2	149	110.5	74.3
12	0.05	168.4	148.1	112.6	79
13	0.055	119.1	79.2	83.6	85.3
14	0.06	120.4	83	82.5	88.4
15	0.065	119.3	80.4	82	79.8
16	0.07	123	80.2	82.3	83.8
17	0.075	121.8	79.1	81.6	81.2
18	0.08	124.1	80.3	81.7	83.8
19	0.085	127.7	81.5	82.6	80.7
20	0.09	130.5	79.1	86.4	87.4
21	0.095	125.6	78.2	82.8	80.9
22	0.1	126.5	79.9	83	82.7
23	0.105	133.8	85.3	83.6	81.2
24	0.11	135.5	80.6	81.2	82.2
25	0.115	122.3	81	84.7	82
26	0.12	119.1	81.2	82	81.5
27	0.125	121.6	80.4	83	80.4
28	0.13	120.3	80.8	87.2	80.9
29	0.135	122.3	78.9	82.8	83.5
30	0.14	122.4	82.5	81.4	80.3
31	0.145	122	81.9	81.1	82.7
32	0.15	120.9	83.3	85.9	81
33	0.155	121.9	82.6	83.7	83.3
34	0.16	122.3	83.6	85	84.7
35	0.165	125.8	85.2	85.1	81.4
36	0.17	126.7	84.1	84.3	80.3
37	0.175	127.2	91.1	84	79.9
38	0.18	124.3	86.7	84.9	80.8
39	0.185	125.5	85.3	85.3	80.4
40	0.19	124.5	84.9	85.2	81.7
41	0.195	126.9	86.5	84.3	82.5
42	0.2	124.6	89.9	84.5	81.3
43	0.205	124.1	89.8	93.5	85
44	0.21	125.6	87.9	85.1	80.5
45	0.215	125.6	88.9	83.9	79.7
46	0.22	125.2	87.5	83.9	80.7
47	0.225	125.7	85.2	83.6	84.4
48	0.23	124.5	84.5	88.1	80.4
49	0.235	125.1	85.8	87.8	81.5
50	0.24	121.3	84.8	83.8	81
51	0.245	124.9	87.2	87.4	81.6
52	0.25	126.4	85.2	88.4	79.5
53	0.255	126.9	85	87.9	80.1
54	0.26	126.7	89.3	89.6	79.1
55	0.265	126.7	92.3	90.2	78.4
56	0.27	130	88.5	87.9	79.8
57	0.275	124.8	92.6	87.4	86.8
58	0.28	125.5	95.6	85.5	93.1
59	0.285	127.5	89.8	84.7	83.2