Program Structures and Algorithms
Spring 2023 (SEC –3)
Assignment-5: Parallel Sorting.

NAME: Venkatesha Matam
NUID: 002740702

Task:

- Modify the code to receive an instance of a ForkJoinPool object we generated, rather than the predefined common Pool.
- Fix the main method to sort arrays with different sizes, cut-off values, and thread counts. Output the average time taken for sorting to the console.

Code Change Snapshots:

1) **main() method in Main.java –**

```
16  */
17 public class Main {
18
19     public static void main(String[] args) {
20         processArgs(args);
21         System.out.println("Degree of parallelism: " + ForkJoinPool.getCommonPoolParallelism());
22
23         int[] arrayLen = { 524288, 1048576, 2097152, 4194304 };
24         ForkJoinPool forkPool;
25
26         int[] threadCount = { 2, 4, 8, 16, 32 };
27         Random random = new Random();
28         int[] array;
29         ArrayList<Long> timesList = new ArrayList<>();
30
31         for (int l = 0; l < arrayLen.length; l++) {
32             int length = arrayLen[l];
33             array = new int[length];
34
35             int[] cutoffLen = { length / 1024 + 1, length / 512 + 1, length / 256 + 1, length / 128 + 1,
36                     length / 64 + 1, length / 32 + 1, length / 16 + 1, length / 8 + 1, length / 4 + 1,
37                     length / 2 + 1, length + 1 };
38
39             System.out.println("Array length is " + arrayLen[l]);
40             System.out.println();
41
42             for (int n = 0; n < threadCount.length; n++) {
43                 for (int c = 0; c < cutoffLen.length; c++) {
44                     ParSort.cutoff = cutoffLen[c];
45
46                     forkPool = new ForkJoinPool(threadCount[n]);
47                     long time;
48                     long startTime = System.currentTimeMillis();
49                     for (int t = 0; t < 10; t++) {
50                         for (int i = 0; i < array.length; i++)
51                             array[i] = random.nextInt(10000000);
52                         ParSort.sort(array, 0, array.length, forkPool);
53                     }
54                     long endTime = System.currentTimeMillis();
55                     time = (endTime - startTime);
56                     timesList.add(time);
57
58                     System.out.println("cutoff:" + (ParSort.cutoff) + " Thread Count: " + threadCount[n]
59                             + "\t10times\t Time:" + time + "ms");
60                 }
61             }
62         }
```

2) sort() method in ParSort.java :

```
public static void sort(int[] array, int from, int to, ForkJoinPool forkJoinPool) {
    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, forkJoinPool); // TO
                                                                                                        // IMPLEMENT
        CompletableFuture<int[]> parsort2 = parsort(array, from + (to - from) / 2, to, forkJoinPool); // 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));
        parsort.join();
    }
}
```

3) parsort() method in ParSort.java:

```
    private static CompletableFuture<int[]> parsort(int[] array, int from, int to, ForkJoinPool forkJoinPool) {
        return CompletableFuture.supplyAsync(
                () -> {
                    int[] res = new int[to - from];
                    System.arraycopy(array, from, res, 0, res.length);
                    sort(res, 0, to - from, forkJoinPool);
                    return res;
                }, forkJoinPool);
    }
}
```

Output Snapshots:

```
Degree of parallelism: 11
Array length is 524288

cutoff:513 Thread Count: 2      10times   Time:825ms
cutoff:1025 Thread Count: 2     10times   Time:563ms
cutoff:2049 Thread Count: 2     10times   Time:507ms
cutoff:4097 Thread Count: 2     10times   Time:345ms
cutoff:8193 Thread Count: 2     10times   Time:375ms
cutoff:16385 Thread Count: 2    10times   Time:272ms
cutoff:32769 Thread Count: 2    10times   Time:460ms
cutoff:65537 Thread Count: 2    10times   Time:646ms
cutoff:131073 Thread Count: 2   10times   Time:528ms
cutoff:262145 Thread Count: 2   10times   Time:391ms
cutoff:524289 Thread Count: 2   10times   Time:596ms
cutoff:513 Thread Count: 4      10times   Time:402ms
cutoff:1025 Thread Count: 4     10times   Time:388ms
cutoff:2049 Thread Count: 4     10times   Time:322ms
cutoff:4097 Thread Count: 4     10times   Time:353ms
cutoff:8193 Thread Count: 4     10times   Time:351ms
cutoff:16385 Thread Count: 4    10times   Time:327ms
cutoff:32769 Thread Count: 4    10times   Time:429ms
cutoff:65537 Thread Count: 4    10times   Time:416ms
cutoff:131073 Thread Count: 4   10times   Time:342ms
cutoff:262145 Thread Count: 4   10times   Time:316ms
cutoff:524289 Thread Count: 4   10times   Time:438ms
cutoff:513 Thread Count: 8      10times   Time:365ms
cutoff:1025 Thread Count: 8     10times   Time:323ms
cutoff:2049 Thread Count: 8     10times   Time:388ms
cutoff:4097 Thread Count: 8     10times   Time:325ms
cutoff:8193 Thread Count: 8     10times   Time:537ms
cutoff:16385 Thread Count: 8    10times   Time:617ms
cutoff:32769 Thread Count: 8    10times   Time:437ms
cutoff:65537 Thread Count: 8    10times   Time:288ms
cutoff:131073 Thread Count: 8   10times   Time:292ms
cutoff:262145 Thread Count: 8   10times   Time:406ms
cutoff:524289 Thread Count: 8   10times   Time:491ms
cutoff:513 Thread Count: 16     10times   Time:443ms
cutoff:1025 Thread Count: 16    10times   Time:376ms
cutoff:2049 Thread Count: 16    10times   Time:380ms
cutoff:4097 Thread Count: 16    10times   Time:451ms
```

```
cutoff:8193 Thread Count: 16     10times  Time:288ms
cutoff:16385 Thread Count: 16    10times  Time:302ms
cutoff:32769 Thread Count: 16    10times  Time:290ms
cutoff:65537 Thread Count: 16    10times  Time:230ms
cutoff:131073 Thread Count: 16   10times  Time:544ms
cutoff:262145 Thread Count: 16   10times  Time:480ms
cutoff:524289 Thread Count: 16   10times  Time:505ms
cutoff:513 Thread Count: 32      10times  Time:339ms
cutoff:1025 Thread Count: 32     10times  Time:303ms
cutoff:2049 Thread Count: 32     10times  Time:284ms
cutoff:4097 Thread Count: 32     10times  Time:271ms
cutoff:8193 Thread Count: 32     10times  Time:301ms
cutoff:16385 Thread Count: 32    10times  Time:453ms
cutoff:32769 Thread Count: 32    10times  Time:348ms
cutoff:65537 Thread Count: 32    10times  Time:281ms
cutoff:131073 Thread Count: 32   10times  Time:299ms
cutoff:262145 Thread Count: 32   10times  Time:353ms
cutoff:524289 Thread Count: 32   10times  Time:676ms
Array length is 1048576

cutoff:1025 Thread Count: 2      10times  Time:878ms
cutoff:2049 Thread Count: 2      10times  Time:634ms
cutoff:4097 Thread Count: 2      10times  Time:713ms
cutoff:8193 Thread Count: 2      10times  Time:624ms
cutoff:16385 Thread Count: 2     10times  Time:670ms
cutoff:32769 Thread Count: 2     10times  Time:1288ms
cutoff:65537 Thread Count: 2     10times  Time:829ms
cutoff:131073 Thread Count: 2    10times  Time:1000ms
cutoff:262145 Thread Count: 2    10times  Time:944ms
cutoff:524289 Thread Count: 2    10times  Time:755ms
cutoff:1048577 Thread Count: 2   10times  Time:1005ms
cutoff:1025 Thread Count: 4      10times  Time:963ms
cutoff:2049 Thread Count: 4      10times  Time:780ms
cutoff:4097 Thread Count: 4      10times  Time:792ms
cutoff:8193 Thread Count: 4      10times  Time:591ms
cutoff:16385 Thread Count: 4     10times  Time:776ms
cutoff:32769 Thread Count: 4     10times  Time:727ms
cutoff:65537 Thread Count: 4     10times  Time:791ms
cutoff:131073 Thread Count: 4    10times  Time:913ms
cutoff:262145 Thread Count: 4    10times  Time:812ms
cutoff:524289 Thread Count: 4    10times  Time:805ms
cutoff:1048577 Thread Count: 4   10times  Time:1104ms
cutoff:1025 Thread Count: 8      10times  Time:759ms
cutoff:2049 Thread Count: 8      10times  Time:1201ms
cutoff:4097 Thread Count: 8      10times  Time:965ms
cutoff:8193 Thread Count: 8      10times  Time:1117ms
cutoff:16385 Thread Count: 8     10times  Time:921ms
cutoff:32769 Thread Count: 8     10times  Time:720ms
cutoff:65537 Thread Count: 8     10times  Time:1016ms
cutoff:131073 Thread Count: 8    10times  Time:704ms
cutoff:262145 Thread Count: 8    10times  Time:556ms
cutoff:524289 Thread Count: 8    10times  Time:757ms
cutoff:1048577 Thread Count: 8   10times  Time:888ms
cutoff:1025 Thread Count: 16     10times  Time:844ms
cutoff:2049 Thread Count: 16     10times  Time:734ms
cutoff:4097 Thread Count: 16     10times  Time:1598ms
cutoff:8193 Thread Count: 16     10times  Time:1028ms
cutoff:16385 Thread Count: 16    10times  Time:728ms
cutoff:32769 Thread Count: 16    10times  Time:938ms
cutoff:65537 Thread Count: 16    10times  Time:631ms
cutoff:131073 Thread Count: 16   10times  Time:678ms
cutoff:262145 Thread Count: 16   10times  Time:826ms
cutoff:524289 Thread Count: 16   10times  Time:710ms
cutoff:1048577 Thread Count: 16  10times  Time:823ms
cutoff:1025 Thread Count: 32     10times  Time:868ms
cutoff:2049 Thread Count: 32     10times  Time:715ms
cutoff:4097 Thread Count: 32     10times  Time:560ms
cutoff:8193 Thread Count: 32     10times  Time:912ms
cutoff:16385 Thread Count: 32    10times  Time:628ms
cutoff:32769 Thread Count: 32    10times  Time:457ms
cutoff:65537 Thread Count: 32    10times  Time:495ms
cutoff:131073 Thread Count: 32   10times  Time:619ms
cutoff:262145 Thread Count: 32   10times  Time:478ms
cutoff:524289 Thread Count: 32   10times  Time:641ms
cutoff:1048577 Thread Count: 32  10times  Time:841ms
Array length is 2097152

cutoff:2049 Thread Count: 2      10times  Time:1448ms
cutoff:4097 Thread Count: 2      10times  Time:1406ms
cutoff:8193 Thread Count: 2      10times  Time:1328ms
```

```
Array length is 2097152

cutoff:2049 Thread Count: 2      10times   Time:1448ms
cutoff:4097 Thread Count: 2      10times   Time:1406ms
cutoff:8193 Thread Count: 2      10times   Time:1328ms
cutoff:16385 Thread Count: 2     10times   Time:1153ms
cutoff:32769 Thread Count: 2     10times   Time:1178ms
cutoff:65537 Thread Count: 2     10times   Time:1534ms
cutoff:131073 Thread Count: 2    10times   Time:1363ms
cutoff:262145 Thread Count: 2    10times   Time:1524ms
cutoff:524289 Thread Count: 2    10times   Time:1704ms
cutoff:1048577 Thread Count: 2   10times   Time:1316ms
cutoff:2097153 Thread Count: 2   10times   Time:1951ms
cutoff:2049 Thread Count: 4      10times   Time:1529ms
cutoff:4097 Thread Count: 4      10times   Time:1302ms
cutoff:8193 Thread Count: 4      10times   Time:1321ms
cutoff:16385 Thread Count: 4     10times   Time:1535ms
cutoff:32769 Thread Count: 4     10times   Time:1351ms
cutoff:65537 Thread Count: 4     10times   Time:1484ms
cutoff:131073 Thread Count: 4    10times   Time:1338ms
cutoff:262145 Thread Count: 4    10times   Time:1729ms
cutoff:524289 Thread Count: 4    10times   Time:1568ms
cutoff:1048577 Thread Count: 4   10times   Time:1447ms
cutoff:2097153 Thread Count: 4   10times   Time:2162ms
cutoff:2049 Thread Count: 8      10times   Time:1444ms
cutoff:4097 Thread Count: 8      10times   Time:1483ms
cutoff:8193 Thread Count: 8      10times   Time:1138ms
cutoff:16385 Thread Count: 8     10times   Time:1041ms
cutoff:32769 Thread Count: 8     10times   Time:1176ms
cutoff:65537 Thread Count: 8     10times   Time:1524ms
cutoff:131073 Thread Count: 8    10times   Time:1573ms
cutoff:262145 Thread Count: 8    10times   Time:1404ms
cutoff:524289 Thread Count: 8    10times   Time:1092ms
cutoff:1048577 Thread Count: 8   10times   Time:1298ms
cutoff:2097153 Thread Count: 8   10times   Time:1823ms
cutoff:2049 Thread Count: 16     10times   Time:1459ms
cutoff:4097 Thread Count: 16     10times   Time:1382ms
cutoff:8193 Thread Count: 16     10times   Time:3628ms
cutoff:16385 Thread Count: 16    10times   Time:2050ms
cutoff:32769 Thread Count: 16    10times   Time:1518ms
```

```
Array length is 4194304

cutoff:4097 Thread Count: 2      10times  Time:3084ms
cutoff:8193 Thread Count: 2      10times  Time:3055ms
cutoff:16385 Thread Count: 2     10times  Time:2859ms
cutoff:32769 Thread Count: 2     10times  Time:3570ms
cutoff:65537 Thread Count: 2     10times  Time:2677ms
cutoff:131073 Thread Count: 2    10times  Time:2737ms
cutoff:262145 Thread Count: 2    10times  Time:2991ms
cutoff:524289 Thread Count: 2    10times  Time:2979ms
cutoff:1048577 Thread Count: 2   10times  Time:3720ms
cutoff:2097153 Thread Count: 2   10times  Time:2476ms
cutoff:4194305 Thread Count: 2   10times  Time:4021ms
cutoff:4097 Thread Count: 4      10times  Time:2812ms
cutoff:8193 Thread Count: 4      10times  Time:3121ms
cutoff:16385 Thread Count: 4     10times  Time:2633ms
cutoff:32769 Thread Count: 4     10times  Time:2589ms
cutoff:65537 Thread Count: 4     10times  Time:2643ms
cutoff:131073 Thread Count: 4    10times  Time:3859ms
cutoff:262145 Thread Count: 4    10times  Time:4107ms
cutoff:524289 Thread Count: 4    10times  Time:3646ms
cutoff:1048577 Thread Count: 4   10times  Time:3115ms
cutoff:2097153 Thread Count: 4   10times  Time:2394ms
cutoff:4194305 Thread Count: 4   10times  Time:3679ms
cutoff:4097 Thread Count: 8      10times  Time:3266ms
cutoff:8193 Thread Count: 8      10times  Time:3364ms
cutoff:16385 Thread Count: 8     10times  Time:2961ms
cutoff:32769 Thread Count: 8     10times  Time:3088ms
cutoff:65537 Thread Count: 8     10times  Time:2998ms
cutoff:131073 Thread Count: 8    10times  Time:2699ms
cutoff:262145 Thread Count: 8    10times  Time:2810ms
cutoff:524289 Thread Count: 8    10times  Time:2441ms
cutoff:1048577 Thread Count: 8   10times  Time:2130ms
cutoff:2097153 Thread Count: 8   10times  Time:2399ms
cutoff:4194305 Thread Count: 8   10times  Time:3902ms
cutoff:4097 Thread Count: 16     10times  Time:2975ms
cutoff:8193 Thread Count: 16     10times  Time:2887ms
cutoff:16385 Thread Count: 16    10times  Time:2621ms
```

Observations and Analysis:

The algorithm utilizes a specific logic to determine the cut-off values in the recursion process. The cut-off point is set at a certain level in the recursion tree where the system sort is applied. To set the cut-off point at a desired level 'l', the formula for calculating the cut-off value is $(array\ length) / 2^l + 1$.

Array Length	Cut-off size	Time (2 Threads)	Time (4 Threads)	Time (8 Threads)	Time (16 Threads)	Time (32 Threads)
524288	513	825ms	402ms	365ms	443ms	339ms
524288	1025	563ms	388ms	323ms	376ms	303ms
524288	2049	507ms	322ms	388ms	380ms	284ms
524288	4097	345ms	353ms	325ms	451ms	271ms
524288	8193	375ms	351ms	537ms	288ms	301ms
524288	16385	272ms	327ms	617ms	302ms	453ms
524288	32769	460ms	429ms	437ms	290ms	348ms
524288	65537	646ms	416ms	288ms	230ms	281ms
524288	131073	528ms	342ms	292ms	544ms	299ms
524288	262145	391ms	316ms	406ms	480ms	353ms
524288	524289	596ms	438ms	491ms	505ms	676ms

Array Length	Cut-off size	Time (2 Threads)	Time (4 Threads)	Time (8 Threads)	Time (16 Threads)	Time (32 Threads)
1048576	1025	878ms	963ms	759ms	844ms	868ms
1048576	2049	634ms	780ms	1201ms	734ms	715ms
1048576	4097	713ms	792ms	965ms	1598ms	560ms
1048576	8193	624ms	591ms	1117ms	1028ms	912ms
1048576	16385	670ms	776ms	921ms	728ms	628ms
1048576	32769	1288ms	727ms	720ms	938ms	457ms
1048576	65537	829ms	791ms	1016ms	631ms	495ms
1048576	131073	1000ms	913ms	704ms	678ms	619ms
1048576	262145	944ms	812ms	556ms	826ms	478ms
1048576	524289	755ms	805ms	757ms	710ms	641ms
1048576	1048577	1005ms	1104ms	888ms	823ms	841ms

Array Length	Cut-off size	Time (2 Threads)	Time (4 Threads)	Time (8 Threads)	Time (16 Threads)	Time (32 Threads)
2097152	2049	1448ms	1529ms	1444ms	1459ms	1549ms
2097152	4097	1406ms	1302ms	1483ms	1382ms	1416ms
2097152	8193	1328ms	1321ms	1138ms	3628ms	1434ms
2097152	16385	1153ms	1535ms	1041ms	2050ms	1748ms
2097152	32769	1178ms	1351ms	1176ms	1518ms	1837ms
2097152	65537	1534ms	1484ms	1524ms	1574ms	1773ms
2097152	131073	1363ms	1338ms	1573ms	1651ms	1183ms
2097152	262145	1524ms	1729ms	1404ms	884ms	1553ms
2097152	524289	1704ms	1568ms	1092ms	988ms	1102ms
2097152	1048577	1316ms	1447ms	1298ms	1255ms	1316ms
2097152	2097153	1951ms	2162ms	1823ms	1772ms	1922ms

Array Length	Cut-off size	Time (2 Threads)	Time (4 Threads)	Time (8 Threads)	Time (16 Threads)	Time (32 Threads)
4194304	4097	3084ms	2812ms	3266ms	2975ms	2999ms
4194304	8193	3055ms	3121ms	3364ms	2887ms	4805ms
4194304	16385	2859ms	2633ms	2961ms	2621ms	3533ms
4194304	32769	3570ms	2589ms	3088ms	2528ms	7790ms
4194304	65537	2677ms	2643ms	2998ms	2595ms	3034ms
4194304	131073	2737ms	3859ms	2699ms	2953ms	4503ms
4194304	262145	2991ms	4107ms	2810ms	2457ms	2666ms
4194304	524289	2979ms	3646ms	2441ms	2149ms	2635ms
4194304	1048577	3720ms	3115ms	2130ms	3114ms	2227ms
4194304	2097153	2476ms	2394ms	2399ms	2894ms	2721ms
4194304	4194305	4021ms	3679ms	3902ms	3874ms	4646ms

- In each table, the final row indicates the duration it took for the system sort to sort the arrays because the cut-off value is equivalent to the array size. The quantity of threads utilized does not affect the outcome since parallel merge sorting is not implemented. The time required for the system sort is more than that of parallel sorting.
- When the cut-off values are large and close to the array length, the depth of the recursion tree is low. This means that by the time we switch to the System sort, there will be fewer concurrent tasks to execute.

Conclusion:

- If the cut-off value is set high and is almost the same as the array size, then it's feasible to increase the number of threads. In such scenarios, the arrays are sorted by the system instead of a parallel sort.
- Parallel merge sort can achieve faster sorting times than system sort for large arrays, especially when using multiple threads and appropriately chosen cut-off values.

- The optimal number of threads and cut-off value can depend on the size of the array being sorted and may require some experimentation to determine.