

Program Structure and Algorithms

Spring 2022

Assignment 4

Himanshu Walia

NUID: 002960393

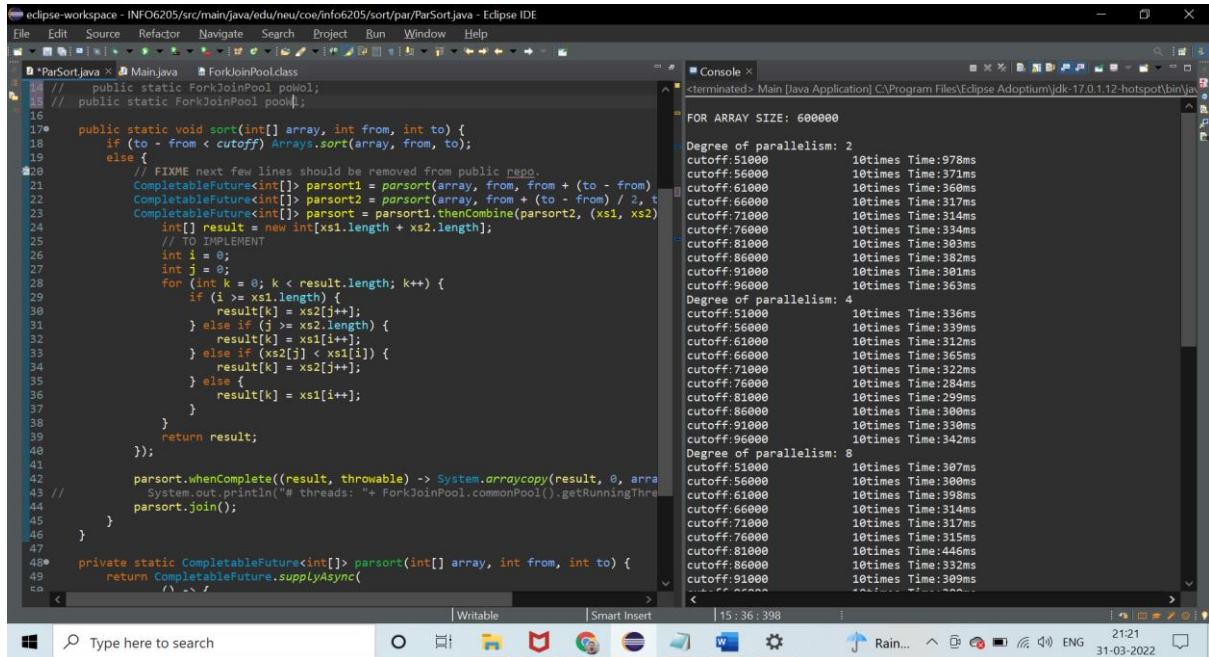
Task: Your task is to implement a parallel sorting algorithm such that each partition of the array is sorted in parallel. You will consider two different schemes for deciding whether to sort in parallel.

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 SCREENSHOT:



The screenshot shows the Eclipse IDE with a Java project named 'INFO6205/src/main/java/edu/neu/coe/info6205/sort/par/ParSort.java'. The code implements a parallel sorting algorithm using `CompletableFuture` and a `ForkJoinPool`. The `sort` method is recursive, splitting the array into two halves and sorting them in parallel. The `main` method uses a `ForkJoinPool` to execute the sorting task. The console output shows the results of the sorting process for an array size of 600,000, including the degree of parallelism and the time taken for each step.

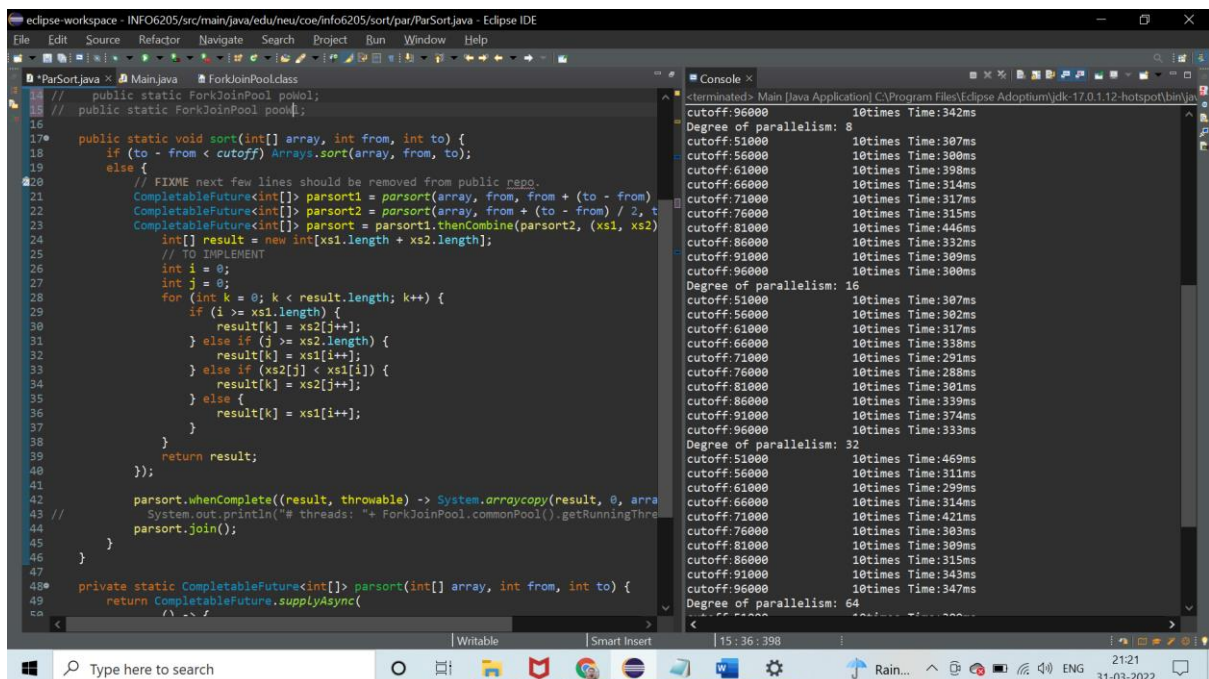
```
public static ForkJoinPool pool;

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);
        CompletableFuture<int[]> parsort2 = parsort(array, from + (to - from) / 2, to);
        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(
        () -> {
            sort(array, from, to);
        }, pool);
}
```

Console Output:

```
sterminated: Main [Java Application] C:\Program Files\Eclipse Adoptium\jdk-17.0.12-hotspot\bin\java.exe
FOR ARRAY SIZE: 600000
Degree of parallelism: 2
cutoff:51000 10times Time:978ms
cutoff:56000 10times Time:371ms
cutoff:61000 10times Time:360ms
cutoff:66000 10times Time:317ms
cutoff:71000 10times Time:314ms
cutoff:76000 10times Time:334ms
cutoff:81000 10times Time:303ms
cutoff:86000 10times Time:382ms
cutoff:91000 10times Time:301ms
cutoff:96000 10times Time:363ms
Degree of parallelism: 4
cutoff:51000 10times Time:336ms
cutoff:56000 10times Time:339ms
cutoff:61000 10times Time:312ms
cutoff:66000 10times Time:365ms
cutoff:71000 10times Time:322ms
cutoff:76000 10times Time:284ms
cutoff:81000 10times Time:299ms
cutoff:86000 10times Time:300ms
cutoff:91000 10times Time:330ms
cutoff:96000 10times Time:342ms
Degree of parallelism: 8
cutoff:51000 10times Time:307ms
cutoff:56000 10times Time:300ms
cutoff:61000 10times Time:398ms
cutoff:66000 10times Time:314ms
cutoff:71000 10times Time:317ms
cutoff:76000 10times Time:315ms
cutoff:81000 10times Time:446ms
cutoff:86000 10times Time:332ms
cutoff:91000 10times Time:309ms
cutoff:96000 10times Time:300ms
Degree of parallelism: 16
cutoff:51000 10times Time:307ms
cutoff:56000 10times Time:302ms
cutoff:61000 10times Time:317ms
cutoff:66000 10times Time:338ms
cutoff:71000 10times Time:291ms
cutoff:76000 10times Time:288ms
cutoff:81000 10times Time:301ms
cutoff:86000 10times Time:339ms
cutoff:91000 10times Time:374ms
cutoff:96000 10times Time:333ms
Degree of parallelism: 32
cutoff:51000 10times Time:469ms
cutoff:56000 10times Time:311ms
cutoff:61000 10times Time:299ms
cutoff:66000 10times Time:314ms
cutoff:71000 10times Time:421ms
cutoff:76000 10times Time:303ms
cutoff:81000 10times Time:309ms
cutoff:86000 10times Time:315ms
cutoff:91000 10times Time:343ms
cutoff:96000 10times Time:347ms
Degree of parallelism: 64
cutoff:51000 10times Time:342ms
```



The screenshot shows the Eclipse IDE with a Java project named 'INFO6205/src/main/java/edu/neu/coe/info6205/sort/par/ParSort.java'. The code implements a parallel sorting algorithm using `CompletableFuture` and a `ForkJoinPool`. The `sort` method is recursive, splitting the array into two halves and sorting them in parallel. The `main` method uses a `ForkJoinPool` to execute the sorting task. The console output shows the results of the sorting process for an array size of 600,000, including the degree of parallelism and the time taken for each step.

```
public static ForkJoinPool pool;

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);
        CompletableFuture<int[]> parsort2 = parsort(array, from + (to - from) / 2, to);
        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(
        () -> {
            sort(array, from, to);
        }, pool);
}
```

Console Output:

```
sterminated: Main [Java Application] C:\Program Files\Eclipse Adoptium\jdk-17.0.12-hotspot\bin\java.exe
FOR ARRAY SIZE: 600000
Degree of parallelism: 8
cutoff:51000 10times Time:342ms
cutoff:56000 10times Time:307ms
cutoff:61000 10times Time:300ms
cutoff:66000 10times Time:398ms
cutoff:71000 10times Time:314ms
cutoff:76000 10times Time:317ms
cutoff:81000 10times Time:315ms
cutoff:86000 10times Time:446ms
cutoff:91000 10times Time:332ms
cutoff:96000 10times Time:309ms
cutoff:96000 10times Time:300ms
Degree of parallelism: 16
cutoff:51000 10times Time:307ms
cutoff:56000 10times Time:302ms
cutoff:61000 10times Time:317ms
cutoff:66000 10times Time:338ms
cutoff:71000 10times Time:291ms
cutoff:76000 10times Time:288ms
cutoff:81000 10times Time:301ms
cutoff:86000 10times Time:339ms
cutoff:91000 10times Time:374ms
cutoff:96000 10times Time:333ms
Degree of parallelism: 32
cutoff:51000 10times Time:469ms
cutoff:56000 10times Time:311ms
cutoff:61000 10times Time:299ms
cutoff:66000 10times Time:314ms
cutoff:71000 10times Time:421ms
cutoff:76000 10times Time:303ms
cutoff:81000 10times Time:309ms
cutoff:86000 10times Time:315ms
cutoff:91000 10times Time:343ms
cutoff:96000 10times Time:347ms
Degree of parallelism: 64
cutoff:51000 10times Time:342ms
```

```
eclipse-workspace - INFO6205/src/main/java/edu/neu/coe/info6205/sort/par/ParSort.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
ParSort.java Main.java ForkJoinPool.class
14 // public static ForkJoinPool pool1;
15 // public static ForkJoinPool pool2;
16
17 public static void sort(int[] array, int from, int to) {
18     if (to - from < cutoff) Arrays.sort(array, from, to);
19     else {
20         // FIXME next few lines should be removed from public repo.
21         CompletableFuture<int[]> parsort1 = parsort(array, from, from + (to - from) / 2, t);
22         CompletableFuture<int[]> parsort2 = parsort(array, from + (to - from) / 2, to, t);
23         CompletableFuture<int[]> parsort = parsort1.thenCombine(parsort2, (xs1, xs2)
24             {
25                 int[] result = new int[xs1.length + xs2.length];
26                 // TO IMPLEMENT
27                 int i = 0;
28                 int j = 0;
29                 for (int k = 0; k < result.length; k++) {
30                     if (i >= xs1.length) {
31                         result[k] = xs2[j++];
32                     } else if (j >= xs2.length) {
33                         result[k] = xs1[i++];
34                     } else if (xs2[j] < xs1[i]) {
35                         result[k] = xs2[j++];
36                     } else {
37                         result[k] = xs1[i++];
38                     }
39                 }
40                 return result;
41             });
42         parsort.whenComplete((result, throwable) -> System.arraycopy(result, 0, array, from, result.length));
43         System.out.println("# threads: " + ForkJoinPool.commonPool().getRunningThreadCount());
44         parsort.join();
45     }
46 }
47
48 private static CompletableFuture<int[]> parsort(int[] array, int from, int to) {
49     return CompletableFuture.supplyAsync(
50         () -> {
51             sort(array, from, to);
52         }
53     );
54 }
55
56 Console
57 C:\Program Files\Eclipse Adoptium\jdk-17.0.12-hotspot\bin\java
58
59 Degree of parallelism: 16
60 cutoff: 96000 10times Time: 300ms
61 cutoff: 51000 10times Time: 307ms
62 cutoff: 56000 10times Time: 302ms
63 cutoff: 61000 10times Time: 317ms
64 cutoff: 66000 10times Time: 338ms
65 cutoff: 71000 10times Time: 291ms
66 cutoff: 76000 10times Time: 288ms
67 cutoff: 81000 10times Time: 301ms
68 cutoff: 86000 10times Time: 339ms
69 cutoff: 91000 10times Time: 374ms
70 cutoff: 96000 10times Time: 333ms
71
72 Degree of parallelism: 32
73 cutoff: 51000 10times Time: 469ms
74 cutoff: 56000 10times Time: 311ms
75 cutoff: 61000 10times Time: 299ms
76 cutoff: 66000 10times Time: 314ms
77 cutoff: 71000 10times Time: 421ms
78 cutoff: 76000 10times Time: 303ms
79 cutoff: 81000 10times Time: 309ms
80 cutoff: 86000 10times Time: 315ms
81 cutoff: 91000 10times Time: 343ms
82 cutoff: 96000 10times Time: 347ms
83
84 Degree of parallelism: 64
85 cutoff: 51000 10times Time: 309ms
86 cutoff: 56000 10times Time: 292ms
87 cutoff: 61000 10times Time: 338ms
88 cutoff: 66000 10times Time: 301ms
89 cutoff: 71000 10times Time: 307ms
90 cutoff: 76000 10times Time: 303ms
91 cutoff: 81000 10times Time: 372ms
92 cutoff: 86000 10times Time: 320ms
93 cutoff: 91000 10times Time: 332ms
94 cutoff: 96000 10times Time: 336ms
95
96 Type here to search
97 Rain... 21:21 31-03-2022
```

```
eclipse-workspace - INFO6205/src/main/java/edu/neu/coe/info6205/sort/par/Main.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
ParSort.java Main.java ForkJoinPool.class
18
19 public class Main {
20
21     public static void main(String[] args) {
22         int ccc=1;
23         processArgs(args);
24         while(ccc<5)
25         {
26
27             int s=1200000;
28             //int tt = 2;
29             System.out.println("\nFOR ARRAY SIZE: " + s + "\n");
30             for(int tt=2;tt<=64;tt*=2)
31             {
32                 ForkJoinPool k = new ForkJoinPool(tt);
33                 int para=k.getParallelism();
34                 System.out.println("Degree of parallelism: " + para);
35                 //ParSort.pool=k;
36                 Random random = new Random();
37                 int[] array = new int[s];
38                 ArrayList<Long> timeList = new ArrayList<>();
39                 for (int j = 50; j < 100; j=j+5) {
40                     ParSort.cutoff = 1000 * (j + 1);
41                     // for (int i = 0; i < array.length; i++) array[i] = random.nextInt(1000000);
42                     long startime = System.currentTimeMillis();
43                     for (int t = 0; t < 10; t++) {
44                         for (int i = 0; i < array.length; i++) array[i] = random.nextInt(1000000);
45                         ParSort.sort(array, 0, array.length);
46                     }
47                     long endTime = System.currentTimeMillis();
48                     time = (endTime - startime);
49                     timeList.add(time);
50                     System.out.println("cutoff: " + (ParSort.cutoff) + "\t\t10times Time: " + time);
51                 }
52                 try {
53                     FileOutputStream fis = new FileOutputStream("../src/result.csv");
54
55 Console
56 C:\Program Files\Eclipse Adoptium\jdk-17.0.12-hotspot\bin\java
57
58 FOR ARRAY SIZE: 1200000
59
60 Degree of parallelism: 2
61 cutoff: 51000 10times Time: 1356ms
62 cutoff: 56000 10times Time: 678ms
63 cutoff: 61000 10times Time: 708ms
64 cutoff: 66000 10times Time: 703ms
65 cutoff: 71000 10times Time: 667ms
66 cutoff: 76000 10times Time: 662ms
67 cutoff: 81000 10times Time: 886ms
68 cutoff: 86000 10times Time: 714ms
69 cutoff: 91000 10times Time: 727ms
70 cutoff: 96000 10times Time: 745ms
71
72 Degree of parallelism: 4
73 cutoff: 51000 10times Time: 672ms
74 cutoff: 56000 10times Time: 764ms
75 cutoff: 61000 10times Time: 710ms
76 cutoff: 66000 10times Time: 805ms
77 cutoff: 71000 10times Time: 643ms
78 cutoff: 76000 10times Time: 665ms
79 cutoff: 81000 10times Time: 666ms
80 cutoff: 86000 10times Time: 747ms
81 cutoff: 91000 10times Time: 664ms
82 cutoff: 96000 10times Time: 670ms
83
84 Degree of parallelism: 8
85 cutoff: 51000 10times Time: 668ms
86 cutoff: 56000 10times Time: 727ms
87 cutoff: 61000 10times Time: 660ms
88 cutoff: 66000 10times Time: 694ms
89 cutoff: 71000 10times Time: 736ms
90 cutoff: 76000 10times Time: 680ms
91 cutoff: 81000 10times Time: 658ms
92 cutoff: 86000 10times Time: 639ms
93 cutoff: 91000 10times Time: 731ms
94 cutoff: 96000 10times Time: 731ms
95
96 Type here to search
97 Rain... 21:50 31-03-2022
```

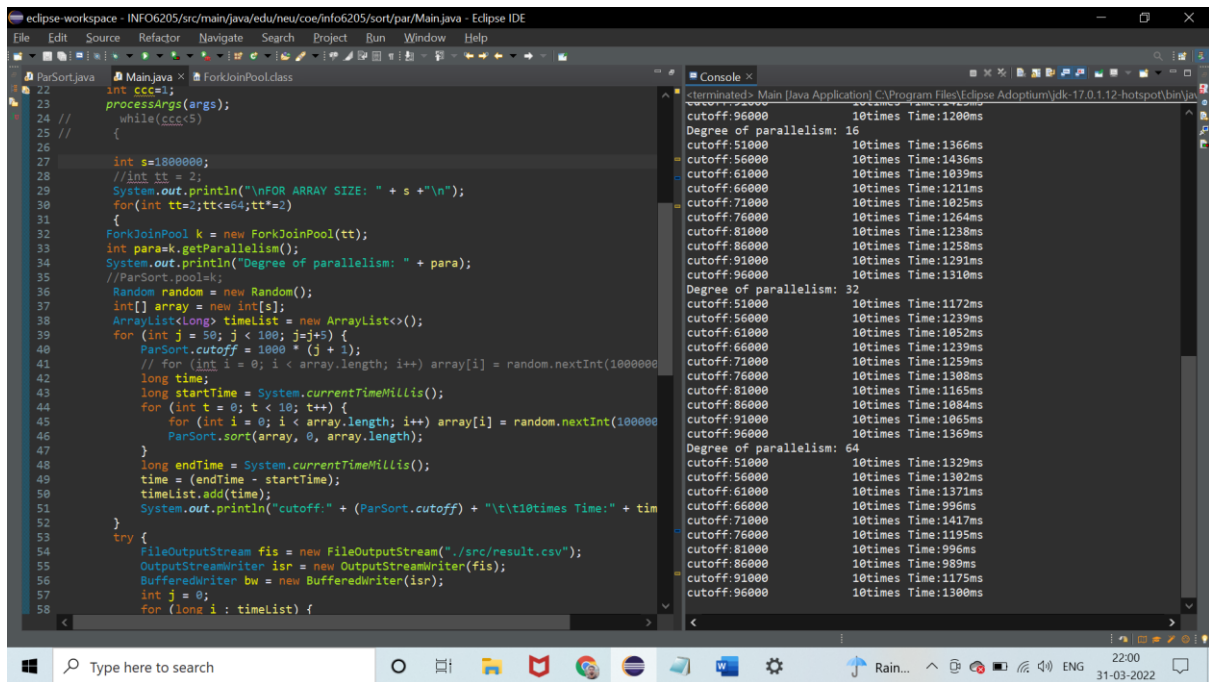
```
eclipse-workspace - INFO6205/src/main/java/edu/neu/coe/info6205/sort/par/Main.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help

ParSort.java Main.java ForkJoinPool.class
18 //
19 public class Main {
20
21     public static void main(String[] args) {
22         int ccc=1;
23         processArgs(args);
24         while(ccc<5)
25         {
26
27             int s=1200000;
28             //int tt = 2;
29             System.out.println("\nFOR ARRAY SIZE: " + s + "\n");
30             for(int tt=2;tt<=64;tt*=2)
31             {
32                 ForkJoinPool k = new ForkJoinPool(tt);
33                 int para=k.getParallelism();
34                 System.out.println("Degree of parallelism: " + para);
35                 //ParSort.pool=k;
36                 Random random = new Random();
37                 int[] array = new int[s];
38                 ArrayList<Long> timelist = new ArrayList<>();
39                 for (int j = 50; j < 100; j=j+5) {
40                     ParSort.cutoff = 1000 * (j + 1);
41                     // for (int i = 0; i < array.length; i++) array[i] = random.nextInt(1000000);
42                     long time;
43                     long startTime = System.currentTimeMillis();
44                     for (int t = 0; t < 10; t++) {
45                         for (int i = 0; i < array.length; i++) array[i] = random.nextInt(1000000);
46                         ParSort.sort(array, 0, array.length);
47                     }
48                     long endTime = System.currentTimeMillis();
49                     time = (endTime - startTime);
50                     timelist.add(time);
51                     System.out.println("cutoff: " + (ParSort.cutoff) + "\t\t10times Time: " + time);
52                 }
53                 try {
54                     FileOutputStream fis = new FileOutputStream("../src/result.csv");
55
56                     k.terminated();
57                     Main [Java Application] C:\Program Files\Eclipse Adoptium\jdk-17.0.12-hotspot\bin\java
58                     cutoff:96000 10times Time:662ms
59                     Degree of parallelism: 16
60                     cutoff:51000 10times Time:672ms
61                     cutoff:56000 10times Time:676ms
62                     cutoff:61000 10times Time:1018ms
63                     cutoff:66000 10times Time:1218ms
64                     cutoff:71000 10times Time:1129ms
65                     cutoff:76000 10times Time:872ms
66                     cutoff:81000 10times Time:695ms
67                     cutoff:86000 10times Time:666ms
68                     cutoff:91000 10times Time:938ms
69                     cutoff:96000 10times Time:1151ms
70                     Degree of parallelism: 32
71                     cutoff:51000 10times Time:1094ms
72                     cutoff:56000 10times Time:946ms
73                     cutoff:61000 10times Time:902ms
74                     cutoff:66000 10times Time:757ms
75                     cutoff:71000 10times Time:668ms
76                     cutoff:76000 10times Time:639ms
77                     cutoff:81000 10times Time:690ms
78                     cutoff:86000 10times Time:680ms
79                     cutoff:91000 10times Time:685ms
80                     cutoff:96000 10times Time:706ms
81                     Degree of parallelism: 64
82                     cutoff:51000 10times Time:681ms
83                     cutoff:56000 10times Time:711ms
84                     cutoff:61000 10times Time:735ms
85                     cutoff:66000 10times Time:694ms
86                     cutoff:71000 10times Time:756ms
87                     cutoff:76000 10times Time:699ms
88                     cutoff:81000 10times Time:667ms
89                     cutoff:86000 10times Time:734ms
90                     cutoff:91000 10times Time:672ms
91                     cutoff:96000 10times Time:786ms
92
93                     File fis.close();
94                 } catch (IOException e) {
95                     e.printStackTrace();
96                 }
97                 ccc++;
98             }
99         }
100     }
101 }
```

```
eclipse-workspace - INFO6205/src/main/java/edu/neu/coe/info6205/sort/par/Main.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help

ParSort.java Main.java ForkJoinPool.class
22 int ccc=1;
23 processArgs(args);
24 while(ccc<5)
25 {
26
27     int s=1800000;
28     //int tt = 2;
29     System.out.println("\nFOR ARRAY SIZE: " + s + "\n");
30     for(int tt=2;tt<=64;tt*=2)
31     {
32         ForkJoinPool k = new ForkJoinPool(tt);
33         int para=k.getParallelism();
34         System.out.println("Degree of parallelism: " + para);
35         //ParSort.pool=k;
36         Random random = new Random();
37         int[] array = new int[s];
38         ArrayList<Long> timelist = new ArrayList<>();
39         for (int j = 50; j < 100; j=j+5) {
40             ParSort.cutoff = 1000 * (j + 1);
41             // for (int i = 0; i < array.length; i++) array[i] = random.nextInt(1000000);
42             long time;
43             long startTime = System.currentTimeMillis();
44             for (int t = 0; t < 10; t++) {
45                 for (int i = 0; i < array.length; i++) array[i] = random.nextInt(1000000);
46                 ParSort.sort(array, 0, array.length);
47             }
48             long endTime = System.currentTimeMillis();
49             time = (endTime - startTime);
50             timelist.add(time);
51             System.out.println("cutoff: " + (ParSort.cutoff) + "\t\t10times Time: " + time);
52         }
53         try {
54             FileOutputStream fis = new FileOutputStream("../src/result.csv");
55             OutputStreamWriter isr = new OutputStreamWriter(fis);
56             BufferedWriter bw = new BufferedWriter(isr);
57             int j = 0;
58             for (long i : timelist) {
59                 bw.write(i + "\n");
60             }
61             bw.close();
62             ccc++;
63         } catch (IOException e) {
64             e.printStackTrace();
65         }
66     }
67 }

k.terminated();
Main [Java Application] C:\Program Files\Eclipse Adoptium\jdk-17.0.12-hotspot\bin\java
FOR ARRAY SIZE: 1800000
Degree of parallelism: 2
cutoff:51000 10times Time:1653ms
cutoff:56000 10times Time:1055ms
cutoff:61000 10times Time:1048ms
cutoff:66000 10times Time:1007ms
cutoff:71000 10times Time:1057ms
cutoff:76000 10times Time:1018ms
cutoff:81000 10times Time:1092ms
cutoff:86000 10times Time:1014ms
cutoff:91000 10times Time:1073ms
cutoff:96000 10times Time:989ms
Degree of parallelism: 4
cutoff:51000 10times Time:995ms
cutoff:56000 10times Time:1118ms
cutoff:61000 10times Time:1148ms
cutoff:66000 10times Time:1084ms
cutoff:71000 10times Time:1007ms
cutoff:76000 10times Time:1400ms
cutoff:81000 10times Time:1414ms
cutoff:86000 10times Time:1362ms
cutoff:91000 10times Time:1324ms
cutoff:96000 10times Time:1223ms
Degree of parallelism: 8
cutoff:51000 10times Time:1279ms
cutoff:56000 10times Time:1240ms
cutoff:61000 10times Time:1388ms
cutoff:66000 10times Time:1412ms
cutoff:71000 10times Time:1135ms
cutoff:76000 10times Time:1386ms
cutoff:81000 10times Time:1162ms
cutoff:86000 10times Time:965ms
cutoff:91000 10times Time:1425ms
cutoff:96000 10times Time:1425ms
```

Console Output:

FOR ARRAY SIZE: 600000

Degree of parallelism: 2

cutoff : 51000	10times Time:978ms
cutoff : 56000	10times Time:371ms
cutoff : 61000	10times Time:360ms
cutoff : 66000	10times Time:317ms
cutoff : 71000	10times Time:314ms
cutoff : 76000	10times Time:334ms
cutoff : 81000	10times Time:303ms
cutoff : 86000	10times Time:382ms
cutoff : 91000	10times Time:301ms
cutoff : 96000	10times Time:363ms

Degree of parallelism: 4

cutoff : 51000	10times Time:336ms
cutoff : 56000	10times Time:339ms
cutoff : 61000	10times Time:312ms
cutoff : 66000	10times Time:365ms
cutoff : 71000	10times Time:322ms
cutoff : 76000	10times Time:284ms
cutoff : 81000	10times Time:299ms
cutoff : 86000	10times Time:300ms
cutoff : 91000	10times Time:330ms
cutoff : 96000	10times Time:342ms

Degree of parallelism: 8

cutoff : 51000	10times Time:307ms
cutoff : 56000	10times Time:300ms
cutoff : 61000	10times Time:398ms
cutoff : 66000	10times Time:314ms
cutoff : 71000	10times Time:317ms
cutoff : 76000	10times Time:315ms
cutoff : 81000	10times Time:446ms
cutoff : 86000	10times Time:332ms
cutoff : 91000	10times Time:309ms
cutoff : 96000	10times Time:300ms

Degree of parallelism: 16

cutoff : 51000	10times Time:307ms
cutoff : 56000	10times Time:302ms
cutoff : 61000	10times Time:317ms
cutoff : 66000	10times Time:338ms
cutoff : 71000	10times Time:291ms
cutoff : 76000	10times Time:288ms
cutoff : 81000	10times Time:301ms
cutoff : 86000	10times Time:339ms
cutoff : 91000	10times Time:374ms
cutoff : 96000	10times Time:333ms

Degree of parallelism: 32

cutoff : 51000	10times Time:469ms
cutoff : 56000	10times Time:311ms
cutoff : 61000	10times Time:299ms
cutoff : 66000	10times Time:314ms
cutoff : 71000	10times Time:421ms
cutoff : 76000	10times Time:303ms
cutoff : 81000	10times Time:309ms
cutoff : 86000	10times Time:315ms
cutoff : 91000	10times Time:343ms
cutoff : 96000	10times Time:347ms

Degree of parallelism: 64

cutoff : 51000	10times Time:309ms
cutoff : 56000	10times Time:292ms
cutoff : 61000	10times Time:338ms
cutoff : 66000	10times Time:301ms
cutoff : 71000	10times Time:307ms
cutoff : 76000	10times Time:303ms
cutoff : 81000	10times Time:372ms
cutoff : 86000	10times Time:320ms
cutoff : 91000	10times Time:332ms
cutoff : 96000	10times Time:336ms

FOR ARRAY SIZE: 1200000

Degree of parallelism: 2

cutoff : 51000	10times Time:1356ms
cutoff : 56000	10times Time:678ms
cutoff : 61000	10times Time:708ms
cutoff : 66000	10times Time:703ms
cutoff : 71000	10times Time:667ms
cutoff : 76000	10times Time:662ms
cutoff : 81000	10times Time:886ms
cutoff : 86000	10times Time:714ms
cutoff : 91000	10times Time:727ms
cutoff : 96000	10times Time:745ms

Degree of parallelism: 4

cutoff : 51000	10times Time:672ms
cutoff : 56000	10times Time:764ms
cutoff : 61000	10times Time:710ms
cutoff : 66000	10times Time:805ms
cutoff : 71000	10times Time:643ms
cutoff : 76000	10times Time:665ms
cutoff : 81000	10times Time:666ms
cutoff : 86000	10times Time:747ms
cutoff : 91000	10times Time:664ms
cutoff : 96000	10times Time:670ms

Degree of parallelism: 8

cutoff : 51000	10times Time:668ms
cutoff : 56000	10times Time:727ms

cutoff : 61000	10times Time:660ms
cutoff : 66000	10times Time:684ms
cutoff : 71000	10times Time:736ms
cutoff : 76000	10times Time:680ms
cutoff : 81000	10times Time:658ms
cutoff : 86000	10times Time:639ms
cutoff : 91000	10times Time:731ms
cutoff : 96000	10times Time:662ms
Degree of parallelism: 16	
cutoff : 51000	10times Time:672ms
cutoff : 56000	10times Time:676ms
cutoff : 61000	10times Time:1018ms
cutoff : 66000	10times Time:1218ms
cutoff : 71000	10times Time:1129ms
cutoff : 76000	10times Time:872ms
cutoff : 81000	10times Time:695ms
cutoff : 86000	10times Time:666ms
cutoff : 91000	10times Time:938ms
cutoff : 96000	10times Time:1151ms
Degree of parallelism: 32	
cutoff : 51000	10times Time:1094ms
cutoff : 56000	10times Time:946ms
cutoff : 61000	10times Time:902ms
cutoff : 66000	10times Time:757ms
cutoff : 71000	10times Time:668ms
cutoff : 76000	10times Time:639ms
cutoff : 81000	10times Time:690ms
cutoff : 86000	10times Time:680ms
cutoff : 91000	10times Time:685ms
cutoff : 96000	10times Time:706ms
Degree of parallelism: 64	
cutoff : 51000	10times Time:681ms
cutoff : 56000	10times Time:711ms
cutoff : 61000	10times Time:735ms
cutoff : 66000	10times Time:694ms
cutoff : 71000	10times Time:756ms
cutoff : 76000	10times Time:699ms
cutoff : 81000	10times Time:667ms
cutoff : 86000	10times Time:734ms
cutoff : 91000	10times Time:672ms
cutoff : 96000	10times Time:786ms

FOR ARRAY SIZE: 1800000

Degree of parallelism: 2	
cutoff : 51000	10times Time:1653ms
cutoff : 56000	10times Time:1055ms
cutoff : 61000	10times Time:1048ms
cutoff : 66000	10times Time:1007ms
cutoff : 71000	10times Time:1057ms
cutoff : 76000	10times Time:1018ms
cutoff : 81000	10times Time:1092ms
cutoff : 86000	10times Time:1014ms
cutoff : 91000	10times Time:1073ms
cutoff : 96000	10times Time:989ms
Degree of parallelism: 4	
cutoff : 51000	10times Time:995ms
cutoff : 56000	10times Time:1118ms
cutoff : 61000	10times Time:1148ms
cutoff : 66000	10times Time:1084ms
cutoff : 71000	10times Time:1007ms

cutoff : 76000	10times Time:1400ms
cutoff : 81000	10times Time:1414ms
cutoff : 86000	10times Time:1362ms
cutoff : 91000	10times Time:1324ms
cutoff : 96000	10times Time:1223ms
Degree of parallelism: 8	
cutoff : 51000	10times Time:1279ms
cutoff : 56000	10times Time:1240ms
cutoff : 61000	10times Time:1388ms
cutoff : 66000	10times Time:1412ms
cutoff : 71000	10times Time:1135ms
cutoff : 76000	10times Time:1386ms
cutoff : 81000	10times Time:1162ms
cutoff : 86000	10times Time:965ms
cutoff : 91000	10times Time:1425ms
cutoff : 96000	10times Time:1200ms
Degree of parallelism: 16	
cutoff : 51000	10times Time:1366ms
cutoff : 56000	10times Time:1436ms
cutoff : 61000	10times Time:1039ms
cutoff : 66000	10times Time:1211ms
cutoff : 71000	10times Time:1025ms
cutoff : 76000	10times Time:1264ms
cutoff : 81000	10times Time:1238ms
cutoff : 86000	10times Time:1258ms
cutoff : 91000	10times Time:1291ms
cutoff : 96000	10times Time:1310ms
Degree of parallelism: 32	
cutoff : 51000	10times Time:1172ms
cutoff : 56000	10times Time:1239ms
cutoff : 61000	10times Time:1052ms
cutoff : 66000	10times Time:1239ms
cutoff : 71000	10times Time:1259ms
cutoff : 76000	10times Time:1308ms
cutoff : 81000	10times Time:1165ms
cutoff : 86000	10times Time:1084ms
cutoff : 91000	10times Time:1065ms
cutoff : 96000	10times Time:1369ms
Degree of parallelism: 64	
cutoff : 51000	10times Time:1329ms
cutoff : 56000	10times Time:1302ms
cutoff : 61000	10times Time:1371ms
cutoff : 66000	10times Time:996ms
cutoff : 71000	10times Time:1417ms
cutoff : 76000	10times Time:1195ms
cutoff : 81000	10times Time:996ms
cutoff : 86000	10times Time:989ms
cutoff : 91000	10times Time:1175ms
cutoff : 96000	10times Time:1300ms

Relationship/Conclusion:

It is quite evident from the above outputs that even with the different cutoff values, array sizes there is no rich difference in the time taken after 4 threads, it all comes out to be the same (approximately). That is there is significant increase (especially when the cutoff value is least) in performance when threads are increased from 2 to 4 but not much difference is there when it(threads) is increased subsequently.

Thus, better performance is achieved when cutoff values increases and threads are increased from 2 to 4. But as stated below performance is best in certain cutoff range.

As depicted in the graph, the optimal cutoff range for which the best performance is archived is when it is between 0.20 to 0.40 percentage of the array size.

The performance takes a hit when it is between .50 and 1.

- 1) Best performance is when threads are 4 (optimal choice)
- 2) Best performance is when cutoff range is .20 - .40 % of the array size.
- 3) Thus, to have overall best performance in terms of threads and cutoff range I would say with 4 threads and cutoff of .30% of array size would give us the best performance.

Evidence: Below are the graphs to prove the relationship depicted above.

X axis: Cut off and Y-axis: Time(ms)

Three graphs for three different array sizes.

