

Program Structures and Algorithms
Fall 2022(SEC 06)

NAME: Mahathi Siddavatam
NUID: 002198134

Task:

Please see the presentation on *Assignment on Parallel Sorting under the Exams. etc.* module.

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.

For varying the number of threads available, you might want to consult the following resources:

- <https://www.callicoder.com/java-8-completablefuture-tutorial/#a-note-about-executor-and-thread-pool>
- [Links to an external site.](#)

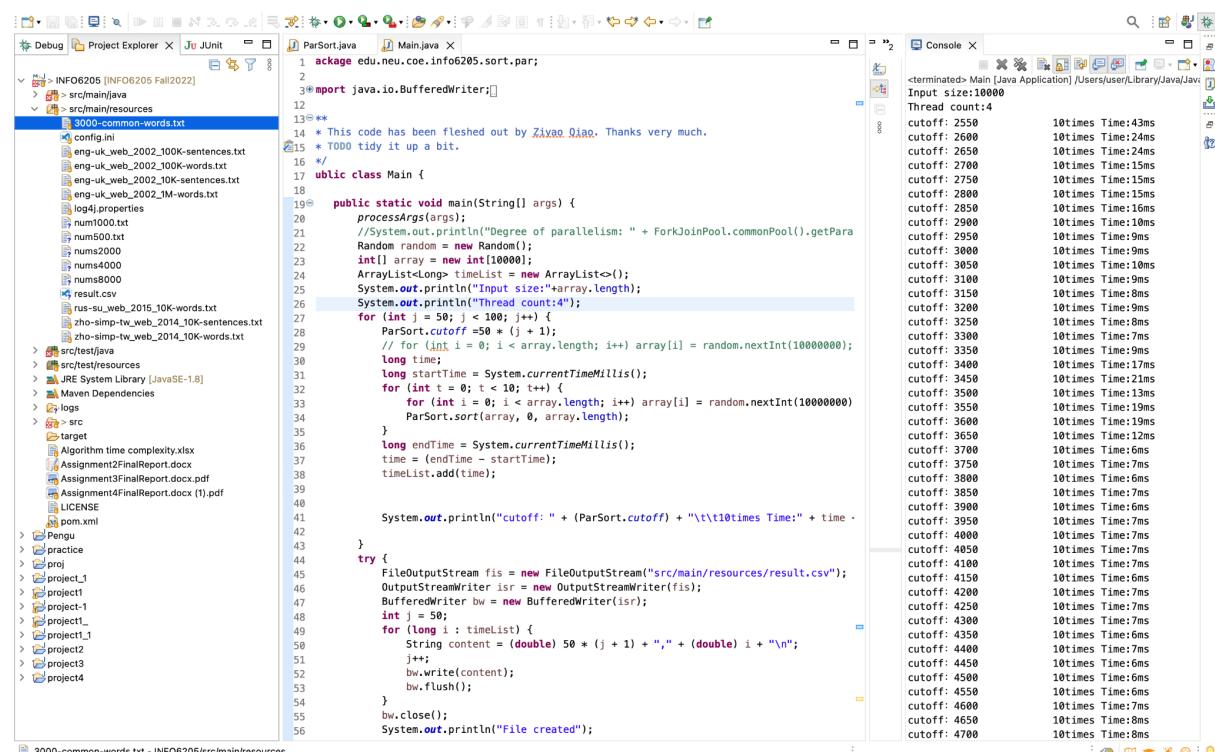
- <https://stackoverflow.com/questions/36569775/how-to-set-forkjoinpool-with-the-desired-number-of-worker-threads-in-completable>
- [Links to an external site.](#)
-

Good luck and enjoy.

Relationship Conclusion: The conclusion regarding the efficiency of parallel sorting is that it is better to have one thread doing multiple tasks at a time than have multiple threads synchronizing and doing multiple tasks. This is evident in the graphs that suggest in most cases that, the more the threads the more time taken to complete the task.

The cut-off value is most optimal when it is 25%-30% of the input size. N/8 is also an efficient cut-off value for many cases.

Evidence to support that conclusion:



The screenshot shows an IDE interface with several windows open. The left window is the Project Explorer, showing a file structure for a Java project named 'INFO6205 [INFO6205 Fall2022]'. It contains a 'src/main/resources' folder with files like '3000-common-words.txt', 'config.ini', and 'result.csv'. The main window displays the 'ParSort.java' file, which contains Java code for parallel sorting. The right window is the 'Console' window, which shows the execution of the program and its performance metrics. The console output includes the following text:

```
<terminated> Main [Java Application] /Users/user/Library/Java/Jav
Input size:10000
Thread count:4
cutoff: 2550 10times Time:43ms
cutoff: 2600 10times Time:24ms
cutoff: 2650 10times Time:24ms
cutoff: 2700 10times Time:15ms
cutoff: 2750 10times Time:15ms
cutoff: 2800 10times Time:15ms
cutoff: 2850 10times Time:16ms
cutoff: 2900 10times Time:10ms
cutoff: 2950 10times Time:9ms
cutoff: 3000 10times Time:9ms
cutoff: 3050 10times Time:10ms
cutoff: 3100 10times Time:9ms
cutoff: 3150 10times Time:8ms
cutoff: 3200 10times Time:9ms
cutoff: 3250 10times Time:8ms
cutoff: 3300 10times Time:7ms
cutoff: 3350 10times Time:9ms
cutoff: 3400 10times Time:17ms
cutoff: 3450 10times Time:21ms
cutoff: 3500 10times Time:13ms
cutoff: 3550 10times Time:19ms
cutoff: 3600 10times Time:19ms
cutoff: 3650 10times Time:12ms
cutoff: 3700 10times Time:6ms
cutoff: 3750 10times Time:7ms
cutoff: 3800 10times Time:6ms
cutoff: 3850 10times Time:7ms
cutoff: 3900 10times Time:6ms
cutoff: 3950 10times Time:7ms
cutoff: 4000 10times Time:7ms
cutoff: 4050 10times Time:7ms
cutoff: 4100 10times Time:7ms
cutoff: 4150 10times Time:6ms
cutoff: 4200 10times Time:7ms
cutoff: 4250 10times Time:7ms
cutoff: 4300 10times Time:7ms
cutoff: 4350 10times Time:6ms
cutoff: 4400 10times Time:7ms
cutoff: 4450 10times Time:6ms
cutoff: 4500 10times Time:6ms
cutoff: 4550 10times Time:6ms
cutoff: 4600 10times Time:7ms
cutoff: 4650 10times Time:8ms
cutoff: 4700 10times Time:8ms
```

INFO6205 [INFO6205 Fall2022]

```

1 package edu.neu.coe.info6205.sort;
2
3 import java.io.BufferedReader;
4
5 /**
6 * This code has been fleshed out by Ziyao Qiao. Thanks very much.
7 * TODO tidy it up a bit.
8 */
9 public class Main {
10
11     public static void main(String[] args) {
12         processArgs(args);
13         //System.out.println("Degree of parallelism: " + ForkJoinPool.commonPool().getParallelism());
14         Random random = new Random();
15         int[] array = new int[10000];
16         ArrayList<Long> timeList = new ArrayList<>();
17         System.out.println("Input size:" + array.length);
18         System.out.println("Thread count:" + args.length);
19         for (int j = 50; j < 100; j++) {
20             ParSort.cutoff = 50 * (j + 1);
21             // For (int i = 0; i < array.length; i++) array[i] = random.nextInt(10000000);
22             long startTime = System.currentTimeMillis();
23             for (int t = 0; t < 10; t++) {
24                 for (int i = 0; i < array.length; i++) {
25                     array[i] = random.nextInt(10000000);
26                 }
27             }
28             long endTime = System.currentTimeMillis();
29             time = (endTime - startTime);
30             timeList.add(time);
31         }
32         System.out.println("cutoff: " + (ParSort.cutoff) + "\t" + "10times Time:" + time);
33     }
34
35     try {
36         FileOutputStream fis = new FileOutputStream("src/main/resources/result.csv");
37         OutputStreamWriter isr = new OutputStreamWriter(fis);
38         BufferedWriter bw = new BufferedWriter(isr);
39         int i = 50;
40         for (long l : timeList) {
41             String content = (double) 50 * (j + 1) + "," + (double) i + "\n";
42             i++;
43             bw.write(content);
44             bw.flush();
45         }
46         bw.close();
47         System.out.println("File created");
48     } catch (Exception e) {
49         e.printStackTrace();
50     }
51 }

```

Console

```

<terminated> Main [Java Application] /Users/user/Library/Java/Java
Input size:10000
Thread count:16
cutoff: 2550 10times Time:48ms
cutoff: 2600 10times Time:31ms
cutoff: 2650 10times Time:33ms
cutoff: 2700 10times Time:34ms
cutoff: 2750 10times Time:23ms
cutoff: 2800 10times Time:16ms
cutoff: 2850 10times Time:17ms
cutoff: 2900 10times Time:16ms
cutoff: 2950 10times Time:17ms
cutoff: 3000 10times Time:17ms
cutoff: 3050 10times Time:8ms
cutoff: 3100 10times Time:6ms
cutoff: 3150 10times Time:5ms
cutoff: 3200 10times Time:7ms
cutoff: 3250 10times Time:5ms
cutoff: 3300 10times Time:4ms
cutoff: 3350 10times Time:5ms
cutoff: 3400 10times Time:5ms
cutoff: 3450 10times Time:5ms
cutoff: 3500 10times Time:4ms
cutoff: 3550 10times Time:5ms
cutoff: 3600 10times Time:5ms
cutoff: 3650 10times Time:4ms
cutoff: 3700 10times Time:4ms
cutoff: 3750 10times Time:21ms
cutoff: 3800 10times Time:23ms
cutoff: 3850 10times Time:6ms
cutoff: 3900 10times Time:5ms
cutoff: 3950 10times Time:5ms
cutoff: 4000 10times Time:5ms
cutoff: 4050 10times Time:5ms
cutoff: 4100 10times Time:5ms
cutoff: 4150 10times Time:5ms
cutoff: 4200 10times Time:6ms
cutoff: 4250 10times Time:4ms
cutoff: 4300 10times Time:5ms
cutoff: 4350 10times Time:5ms
cutoff: 4400 10times Time:5ms
cutoff: 4450 10times Time:4ms
cutoff: 4500 10times Time:5ms
cutoff: 4550 10times Time:5ms
cutoff: 4600 10times Time:5ms
cutoff: 4650 10times Time:4ms
cutoff: 4700 10times Time:5ms

```

Two screenshots of the Eclipse IDE interface showing the development of a Java application named 'ParSort'.

Screenshot 1: The code implements a simple parallel sorting algorithm using ForkJoinPool. The main class 'Main' contains a static void main method that processes command-line arguments, prints the degree of parallelism, and sorts an array of integers. It then writes the sorted array to a CSV file named 'result.csv'. The 'Console' tab shows the output of the application, which includes the thread count and execution times for 10 iterations of the sort operation.

```

1 package edu.neu.coe.info6205.sort.par;
2
3 import java.io.BufferedReader;
4 import java.io.BufferedWriter;
5 import java.io.File;
6 import java.io.FileOutputStream;
7 import java.io.IOException;
8 import java.io.OutputStreamWriter;
9 import java.util.ArrayList;
10 import java.util.Random;
11 import java.util.Scanner;
12
13 /**
14 * This code has been fleshed out by Ziyao Qiao. Thanks very much.
15 * TODO tidy it up a bit.
16 */
17 public class Main {
18
19     public static void main(String[] args) {
20         processArgs(args);
21         //System.out.println("Degree of parallelism: " + ForkJoinPool.commonPool().getParallelism());
22         Random random = new Random();
23         int[] array = new int[20000];
24         ArrayList<Long> timeList = new ArrayList<>();
25         System.out.println("Input size:" + array.length);
26         System.out.println("Thread count:" + 20000);
27         for (int j = 50; j < 100; j++) {
28             ParSort.cutoff = 1000 * (j + 1);
29             // for (int i = 0; i < array.length; i++) array[i] = random.nextInt(10000000);
30             long time;
31             long startTime = System.currentTimeMillis();
32             for (int t = 0; t < 10; t++) {
33                 for (int i = 0; i < array.length; i++) array[i] = random.nextInt(10000000);
34                 ParSort.sort(array, 0, array.length);
35             }
36             long endTime = System.currentTimeMillis();
37             time = (endTime - startTime);
38             timeList.add(time);
39         }
40         System.out.println("cutoff: " + (ParSort.cutoff) + "\t\t10times Time:" + time -
41
42     }
43
44     try {
45         FileOutputStream fis = new FileOutputStream("src/main/resources/result.csv");
46         OutputStreamWriter isr = new OutputStreamWriter(fis);
47         BufferedWriter bw = new BufferedWriter(isr);
48         int j = 50;
49         for (long i : timeList) {
50             String content = (double) 50 * (j + 1) + "," + (double) i + "\n";
51             j++;
52             bw.write(content);
53             bw.flush();
54         }
55         bw.close();
56         System.out.println("File created");
57     } catch (IOException e) {
58         e.printStackTrace();
59     }
60
61 }

```

Screenshot 2: The code has been modified to use ForkJoinPool's built-in parallel sorting mechanism. The 'ParSort' class now contains a static void sort method that takes an array, a from index, and a to index. It uses a recursive divide-and-conquer approach to sort segments of the array. The main class 'Main' remains largely the same, with the addition of a line to print the degree of parallelism. The 'Console' tab shows the execution times for 10 iterations of the sort operation.

```

1 package edu.neu.coe.info6205.sort.par;
2
3 import java.io.File;
4
5 /**
6 * This code has been fleshed out by Ziyao Qiao. Thanks very much.
7 * TODO tidy it up a bit.
8 */
9
10 class ParSort {
11
12     public static int cutoff = 1000;
13     public static ForkJoinPool myPool = new ForkJoinPool(4);
14
15     public static void sort(int[] array, int from, int to) {
16         if (to - from < cutoff) Arrays.sort(array, from, to);
17         else {
18             //FIXME next few lines should be removed from public repo.
19             //CompletableFuture<int[]> parsort1 = parsort(from, from + (to - from) / 2, to);
20             //CompletableFuture<int[]> parsort2 = parsort(from, from + (to - from) / 2, to);
21             //CompletableFuture<int[]> parsort = parsort1.thenCombine(parsort2, (xs1, xs2) ->
22             int[] result = new int[xs1.length + xs2.length];
23             // TO IMPLEMENT
24             int i = 0;
25             int j = 0;
26             for (int k = 0; k < result.length; k++) {
27                 if (i >= xs1.length) {
28                     result[k] = xs2[j];
29                 } else if (j >= xs2.length) {
30                     result[k] = xs1[i];
31                 } else if (xs2[j] < xs1[i]) {
32                     result[k] = xs2[j];
33                 } else {
34                     result[k] = xs1[i];
35                 }
36             }
37             return result;
38         };
39         parsort.whenComplete((result, throwable) -> System.arraycopy(result, 0, array,
40             //System.out.println("# threads: " + ForkJoinPool.commonPool().getRunningThreads());
41             from));
42         parsort.join();
43     }
44
45     private static CompletableFuture<int[]> parsort(int[] array, int from, int to) {
46
47         //System.out.println("Degree of parallelism: " + myPool.getParallelism());
48
49         return CompletableFuture.supplyAsync(
50             () -> {
51                 int[] result = new int[to - from];
52                 // TO IMPLEMENT
53                 CustomResourcePool(result, from, result.length);
54             }
55         );
56     }
57
58 }
59
60
61
62
63
64
65
66

```

Project Explorer

```

1 package edu.neu.coe.info6205.sort.par;
2
3 import java.io.File;
4
5 /**
6 * This code has been fleshed out by Zixiao Qiao. Thanks very much.
7 */
8
9 class ParSort {
10
11     public static int cutoff = 1000;
12     public static ForkJoinPool myPool = new ForkJoinPool(8);
13
14     public static void sort(int[] array, int from, int to) {
15         if (to - from < cutoff)
16             Arrays.sort(array, from, to);
17         else {
18             // FIXME: next few lines should be removed from public repo.
19             CompletableFuture<int[]> parsort1 = parsort(array, from, from + (to - from) / 2);
20             CompletableFuture<int[]> parsort2 = parsort(array, from + (to - from) / 2, to);
21             parsort1.thenCombine(parsort2, (xs1, xs2) -> {
22                 int[] result = new int[xs1.length + xs2.length];
23                 int i = 0;
24                 int j = 0;
25                 for (int k = 0; k < result.length; k++) {
26                     if (xs1[i] < xs2[j])
27                         result[k] = xs1[i]++;
28                     else if (j >= xs2.length)
29                         result[k] = xs1[i]++;
30                     else if (xs2[j] < xs1[i])
31                         result[k] = xs2[j]++;
32                     else {
33                         result[k] = xs1[i]++;
34                     }
35                 }
36                 return result;
37             });
38             parsort1.whenComplete((result, throwable) -> System.arraycopy(result, 0, array,
39                 //System.out.println("# threads: " + ForkJoinPool.commonPool().getRunningThreads());
40             );
41             parsort2.join();
42         }
43     }
44
45     private static CompletableFuture<int[]> parsort(int[] array, int from, int to) {
46
47         // TO IMPLEMENT
48         int[] result = new int[to - from];
49         //System.out.println("Degree of parallelism: " + myPool.getParallelism());
50
51         return CompletableFuture.supplyAsync(
52             () -> {
53                 int[] result = new int[to - from];
54                 // TO IMPLEMENT
55                 CustomArrayUtil.parallelFor(from, result, A_result.length);
56             }
57         );
58     }
59
60     public static void main(String[] args) {
61         processArgs(args);
62         //System.out.println("Degree of parallelism: " + ForkJoinPool.commonPool().getParallelism());
63         Random random = new Random();
64         int[] array = new int[20000];
65         ArrayList<Long> timeList = new ArrayList<>();
66         System.out.println("Input size:" + array.length);
67         System.out.println("Thread count:16");
68         for (int j = 50; j < 100; j++) {
69             ParSort.cutoff = 100 * (j + 1);
70             // for (int i = 0; i < array.length; i++) array[i] = random.nextInt(10000000);
71             long time;
72             long startTime = System.currentTimeMillis();
73             for (int t = 0; t < 10; t++) {
74                 for (int i = 0; i < array.length; i++) array[i] = random.nextInt(10000000);
75                 ParSort.sort(array, 0, array.length);
76             }
77             long endTime = System.currentTimeMillis();
78             time = (endTime - startTime);
79             timeList.add(time);
80         }
81         System.out.println("cutoff: " + (ParSort.cutoff) + "\t\t10times Time: " + time -
82             timeList.get(0));
83         try {
84             FileOutputStream fis = new FileOutputStream("src/main/resources/result.csv");
85             OutputStreamWriter isr = new OutputStreamWriter(fis);
86             BufferedWriter bw = new BufferedWriter(isr);
87             int i = 100;
88             for (long l : timeList) {
89                 String content = (double) 50 * (i + 1) + "," + (double) i + "\n";
90                 i++;
91                 bw.write(content);
92                 bw.flush();
93             }
94             bw.close();
95             System.out.println("File created");
96         } catch (IOException e) {
97             e.printStackTrace();
98         }
99     }
100
101     private static void processArgs(String[] args) {
102         String[] xs = args;
103         while (xs.length > 0)
104             if (xs[0].startsWith("-")) xs = processArgs(xs);
105     }
106 }

```

Console

```

<terminated> Main [Java Application] /Users/user/Library/Java/Java
Input size:20000
Thread count:20000
cutoff: 5100 10times Time:86ms
cutoff: 5200 10times Time:62ms
cutoff: 5300 10times Time:61ms
cutoff: 5400 10times Time:51ms
cutoff: 5500 10times Time:34ms
cutoff: 5600 10times Time:14ms
cutoff: 5700 10times Time:12ms
cutoff: 5800 10times Time:11ms
cutoff: 5900 10times Time:9ms
cutoff: 6000 10times Time:10ms
cutoff: 6100 10times Time:9ms
cutoff: 6200 10times Time:9ms
cutoff: 6300 10times Time:11ms
cutoff: 6400 10times Time:9ms
cutoff: 6500 10times Time:10ms
cutoff: 6600 10times Time:10ms
cutoff: 6700 10times Time:9ms
cutoff: 6800 10times Time:10ms
cutoff: 6900 10times Time:10ms
cutoff: 7000 10times Time:10ms
cutoff: 7100 10times Time:10ms
cutoff: 7200 10times Time:10ms
cutoff: 7300 10times Time:11ms
cutoff: 7400 10times Time:36ms
cutoff: 7500 10times Time:31ms
cutoff: 7600 10times Time:13ms
cutoff: 7700 10times Time:11ms
cutoff: 7800 10times Time:13ms
cutoff: 7900 10times Time:35ms
cutoff: 8000 10times Time:10ms
cutoff: 8100 10times Time:10ms
cutoff: 8200 10times Time:10ms
cutoff: 8300 10times Time:10ms
cutoff: 8400 10times Time:9ms
cutoff: 8500 10times Time:9ms
cutoff: 8600 10times Time:9ms
cutoff: 8700 10times Time:9ms
cutoff: 8800 10times Time:9ms
cutoff: 8900 10times Time:9ms
cutoff: 9000 10times Time:10ms
cutoff: 9100 10times Time:9ms
cutoff: 9200 10times Time:11ms
cutoff: 9300 10times Time:10ms
cutoff: 9400 10times Time:10ms

```

Project Explorer

```

1 package edu.neu.coe.info6205.sort.par;
2
3 import java.io.File;
4
5 /**
6 * This code has been fleshed out by Zixiao Qiao. Thanks very much.
7 */
8
9 class ParSort {
10
11     public static int cutoff = 100 * (50 + 1);
12     public static ForkJoinPool myPool = new ForkJoinPool(8);
13
14     public static void sort(int[] array, int from, int to) {
15         if (to - from < cutoff)
16             Arrays.sort(array, from, to);
17         else {
18             // FIXME: next few lines should be removed from public repo.
19             CompletableFuture<int[]> parsort1 = parsort(array, from, from + (to - from) / 2);
20             CompletableFuture<int[]> parsort2 = parsort(array, from + (to - from) / 2, to);
21             parsort1.thenCombine(parsort2, (xs1, xs2) -> {
22                 int[] result = new int[xs1.length + xs2.length];
23                 int i = 0;
24                 int j = 0;
25                 for (int k = 0; k < result.length; k++) {
26                     if (xs1[i] < xs2[j])
27                         result[k] = xs1[i]++;
28                     else if (j >= xs2.length)
29                         result[k] = xs1[i]++;
30                     else if (xs2[j] < xs1[i])
31                         result[k] = xs2[j]++;
32                     else {
33                         result[k] = xs1[i]++;
34                     }
35                 }
36                 return result;
37             });
38             parsort1.whenComplete((result, throwable) -> System.arraycopy(result, 0, array,
39                 //System.out.println("# threads: " + ForkJoinPool.commonPool().getRunningThreads());
40             );
41             parsort2.join();
42         }
43     }
44
45     private static CompletableFuture<int[]> parsort(int[] array, int from, int to) {
46
47         // TO IMPLEMENT
48         int[] result = new int[to - from];
49         //System.out.println("Degree of parallelism: " + myPool.getParallelism());
50
51         return CompletableFuture.supplyAsync(
52             () -> {
53                 int[] result = new int[to - from];
54                 // TO IMPLEMENT
55                 CustomArrayUtil.parallelFor(from, result, A_result.length);
56             }
57         );
58     }
59
60     public static void main(String[] args) {
61         processArgs(args);
62         //System.out.println("Degree of parallelism: " + ForkJoinPool.commonPool().getParallelism());
63         Random random = new Random();
64         int[] array = new int[20000];
65         ArrayList<Long> timeList = new ArrayList<>();
66         System.out.println("Input size:" + array.length);
67         System.out.println("Thread count:16");
68         for (int j = 50; j < 100; j++) {
69             ParSort.cutoff = 100 * (j + 1);
70             // for (int i = 0; i < array.length; i++) array[i] = random.nextInt(10000000);
71             long time;
72             long startTime = System.currentTimeMillis();
73             for (int t = 0; t < 10; t++) {
74                 for (int i = 0; i < array.length; i++) array[i] = random.nextInt(10000000);
75                 ParSort.sort(array, 0, array.length);
76             }
77             long endTime = System.currentTimeMillis();
78             time = (endTime - startTime);
79             timeList.add(time);
80         }
81         System.out.println("cutoff: " + (ParSort.cutoff) + "\t\t10times Time: " + time -
82             timeList.get(0));
83         try {
84             FileOutputStream fis = new FileOutputStream("src/main/resources/result.csv");
85             OutputStreamWriter isr = new OutputStreamWriter(fis);
86             BufferedWriter bw = new BufferedWriter(isr);
87             int i = 100;
88             for (long l : timeList) {
89                 String content = (double) 50 * (i + 1) + "," + (double) i + "\n";
90                 i++;
91                 bw.write(content);
92                 bw.flush();
93             }
94             bw.close();
95             System.out.println("File created");
96         } catch (IOException e) {
97             e.printStackTrace();
98         }
99     }
100
101     private static void processArgs(String[] args) {
102         String[] xs = args;
103         while (xs.length > 0)
104             if (xs[0].startsWith("-")) xs = processArgs(xs);
105     }
106 }

```

Console

```

<terminated> Main [Java Application] /Users/user/Library/Java/Java
Input size:20000
Thread count:16
cutoff: 5100 10times Time:95ms
cutoff: 5200 10times Time:62ms
cutoff: 5300 10times Time:42ms
cutoff: 5400 10times Time:27ms
cutoff: 5500 10times Time:13ms
cutoff: 5600 10times Time:14ms
cutoff: 5700 10times Time:14ms
cutoff: 5800 10times Time:13ms
cutoff: 5900 10times Time:13ms
cutoff: 6000 10times Time:11ms
cutoff: 6100 10times Time:11ms
cutoff: 6200 10times Time:11ms
cutoff: 6300 10times Time:11ms
cutoff: 6400 10times Time:10ms
cutoff: 6500 10times Time:10ms
cutoff: 6600 10times Time:10ms
cutoff: 6700 10times Time:11ms
cutoff: 6800 10times Time:37ms
cutoff: 6900 10times Time:13ms
cutoff: 7000 10times Time:12ms
cutoff: 7100 10times Time:12ms
cutoff: 7200 10times Time:12ms
cutoff: 7300 10times Time:11ms
cutoff: 7400 10times Time:38ms
cutoff: 7500 10times Time:11ms
cutoff: 7600 10times Time:11ms
cutoff: 7700 10times Time:10ms
cutoff: 7800 10times Time:10ms
cutoff: 7900 10times Time:10ms
cutoff: 8000 10times Time:10ms
cutoff: 8100 10times Time:9ms
cutoff: 8200 10times Time:9ms
cutoff: 8300 10times Time:9ms
cutoff: 8400 10times Time:9ms
cutoff: 8500 10times Time:9ms
cutoff: 8600 10times Time:10ms
cutoff: 8700 10times Time:10ms
cutoff: 8800 10times Time:10ms
cutoff: 8900 10times Time:10ms
cutoff: 9000 10times Time:10ms
cutoff: 9100 10times Time:10ms
cutoff: 9200 10times Time:11ms
cutoff: 9300 10times Time:11ms
cutoff: 9400 10times Time:10ms

```

Project Explorer

```

public static void main(String[] args) {
    processArgs(args);
    //System.out.println("Degree of parallelism: " + ForkJoinPool.commonPool().getParallelism());
    Random random = new Random();
    int[] array = new int[20000];
    ArrayList<Long> timeList = new ArrayList<>();
    System.out.println("Input size:" + array.length);
    System.out.println("Thread count:4");
    for (int j = 50; j < 100; j++) {
        ParSort.cutoff = 100 * (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/main/resources/result.csv");
    OutputStreamWriter isr = new OutputStreamWriter(fis);
    BufferedWriter bw = new BufferedWriter(isr);
    int j = 100;
    for (long l : timeList) {
        String content = (double) 50 * (j + 1) + "," + (double) j + "\n";
        j++;
        bw.write(content);
        bw.flush();
    }
    bw.close();
    System.out.println("File created");
} catch (IOException e) {
    e.printStackTrace();
}
}

private static void processArgs(String[] args) {
    String[] xs = args;
    while (xs.length > 0) {
        if (xs[0].startsWith("-")) xs = processArgs(xs);
    }
}

```

Console

```

<terminated> Main [Java Application] /Users/user/Library/Java/JavaSE-18/bin/java INFO6205
Input size:20000
Thread count:16
cutoff: 5100 10times Time:95ms
cutoff: 5200 10times Time:62ms
cutoff: 5300 10times Time:43ms
cutoff: 5400 10times Time:27ms
cutoff: 5500 10times Time:13ms
cutoff: 5600 10times Time:14ms
cutoff: 5700 10times Time:14ms
cutoff: 5800 10times Time:13ms
cutoff: 5900 10times Time:13ms
cutoff: 6000 10times Time:11ms
cutoff: 6100 10times Time:11ms
cutoff: 6200 10times Time:11ms
cutoff: 6300 10times Time:11ms
cutoff: 6400 10times Time:10ms
cutoff: 6500 10times Time:10ms
cutoff: 6600 10times Time:10ms
cutoff: 6700 10times Time:11ms
cutoff: 6800 10times Time:37ms
cutoff: 6900 10times Time:13ms
cutoff: 7000 10times Time:12ms
cutoff: 7100 10times Time:12ms
cutoff: 7200 10times Time:12ms
cutoff: 7300 10times Time:11ms
cutoff: 7400 10times Time:38ms
cutoff: 7500 10times Time:11ms
cutoff: 7600 10times Time:11ms
cutoff: 7700 10times Time:10ms
cutoff: 7800 10times Time:10ms
cutoff: 7900 10times Time:10ms
cutoff: 8000 10times Time:10ms
cutoff: 8100 10times Time:9ms
cutoff: 8200 10times Time:9ms
cutoff: 8300 10times Time:9ms
cutoff: 8400 10times Time:9ms
cutoff: 8500 10times Time:9ms
cutoff: 8600 10times Time:9ms
cutoff: 8700 10times Time:10ms
cutoff: 8800 10times Time:10ms
cutoff: 8900 10times Time:10ms
cutoff: 9000 10times Time:10ms
cutoff: 9100 10times Time:10ms
cutoff: 9200 10times Time:11ms
cutoff: 9300 10times Time:11ms
cutoff: 9400 10times Time:10ms

```

Project Explorer

```

package edu.neu.coe.info6205.sort.par;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.OutputStreamWriter;
import java.util.ArrayList;
import java.util.Random;

public class Main {
    public static void main(String[] args) {
        processArgs(args);
        //System.out.println("Degree of parallelism: " + ForkJoinPool.commonPool().getParallelism());
        Random random = new Random();
        int[] array = new int[40000];
        ArrayList<Long> timeList = new ArrayList<>();
        System.out.println("Input size:" + array.length);
        System.out.println("Thread count:4");
        for (int j = 50; j < 100; j++) {
            ParSort.cutoff = 350 * (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/main/resources/result.csv");
        OutputStreamWriter isr = new OutputStreamWriter(fis);
        BufferedWriter bw = new BufferedWriter(isr);
        int j = 50;
        for (long l : timeList) {
            String content = (double) 350 * (j + 1) + "," + (double) j + "\n";
            j++;
            bw.write(content);
            bw.flush();
        }
        bw.close();
        System.out.println("File created");
    } catch (IOException e) {
        e.printStackTrace();
    }
}

private static void processArgs(String[] args) {
    String[] xs = args;
    while (xs.length > 0) {
        if (xs[0].startsWith("-")) xs = processArgs(xs);
    }
}

```

Console

```

<terminated> Main [Java Application] /Users/user/Library/Java/JavaSE-18/bin/java INFO6205
Input size:40000
Thread count:4
cutoff: 17850 10times Time:148ms
cutoff: 18200 10times Time:101ms
cutoff: 18550 10times Time:60ms
cutoff: 18900 10times Time:49ms
cutoff: 19250 10times Time:24ms
cutoff: 19600 10times Time:24ms
cutoff: 19950 10times Time:24ms
cutoff: 20300 10times Time:46ms
cutoff: 20650 10times Time:23ms
cutoff: 21000 10times Time:24ms
cutoff: 21350 10times Time:23ms
cutoff: 21700 10times Time:23ms
cutoff: 22050 10times Time:44ms
cutoff: 22400 10times Time:25ms
cutoff: 22750 10times Time:22ms
cutoff: 23100 10times Time:23ms
cutoff: 23450 10times Time:22ms
cutoff: 23800 10times Time:22ms
cutoff: 24150 10times Time:20ms
cutoff: 24500 10times Time:21ms
cutoff: 24850 10times Time:20ms
cutoff: 25200 10times Time:21ms
cutoff: 25550 10times Time:22ms
cutoff: 25900 10times Time:22ms
cutoff: 26250 10times Time:34ms
cutoff: 26600 10times Time:27ms
cutoff: 26950 10times Time:22ms
cutoff: 27300 10times Time:23ms
cutoff: 27650 10times Time:23ms
cutoff: 28000 10times Time:22ms
cutoff: 28350 10times Time:22ms
cutoff: 28700 10times Time:21ms
cutoff: 29050 10times Time:21ms
cutoff: 29400 10times Time:22ms
cutoff: 29750 10times Time:22ms
cutoff: 30100 10times Time:22ms
cutoff: 30450 10times Time:22ms
cutoff: 30800 10times Time:23ms
cutoff: 31150 10times Time:21ms
cutoff: 31500 10times Time:23ms
cutoff: 31850 10times Time:21ms
cutoff: 32200 10times Time:21ms
cutoff: 32550 10times Time:22ms
cutoff: 32900 10times Time:22ms

```

Project Explorer

```

1 package edu.neu.coe.info6205.sort.par;
2
3 import java.io.File;
4
5 /**
6  * This code has been fleshed out by Ziyao Qiao. Thanks very much.
7  * TODO tidy it up a bit.
8 */
9
10 class ParSort {
11
12    /**
13     * public static int cutoff = 1000;
14     * public static ForkJoinPool myPool = new ForkJoinPool(8);
15     */
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);
22            CompletableFuture<int[]> parsort2 = parsort(array, from + (to - from) / 2, to);
23            CompletableFuture<int[]> parsort = parsort1.thenCombine(parsort2, (xs1, xs2) ->
24                int[] result = new int[xs1.length + xs2.length];
25                // TO IMPLEMENT
26                int i = 0;
27                int j = 0;
28                for (int k = 0; k < result.length; k++) {
29                    if (i < xs1.length) {
30                        result[k] = xs1[i++];
31                    } else if (j >= xs2.length) {
32                        result[k] = xs2[j++];
33                    } else if (xs2[j] < xs1[i]) {
34                        result[k] = xs2[j++];
35                    } else {
36                        result[k] = xs1[i++];
37                    }
38                }
39                return result;
40            });
41            parsort.whenComplete((result, throwable) -> System.arraycopy(result, 0, array,
42                //System.out.println("# threads: " + ForkJoinPool.commonPool().getRunningThreads());
43            );
44        }
45    }
46
47    private static CompletableFuture<int[]> parsort(int[] array, int from, int to) {
48
49        //System.out.println("Degree of parallelism: " + myPool.getParallelism());
50
51        return CompletableFuture.supplyAsync(
52            () -> {
53                int[] result = new int[to - from];
54                // TO IMPLEMENT
55                CustomResourceAllocation.from(result, a(result.length));
56            }
57        );
58    }
59
60
61
62
63
64
65
66

```

Console

```

terminated: Main [Java Application] /Users/User/Library/Java/J
Input size:40000
Thread count:16
cutoff: 17850 10times Time:172ms
cutoff: 18200 10times Time:102ms
cutoff: 18550 10times Time:49ms
cutoff: 18900 10times Time:26ms
cutoff: 19250 10times Time:56ms
cutoff: 19600 10times Time:20ms
cutoff: 19950 10times Time:20ms
cutoff: 20300 10times Time:24ms
cutoff: 20650 10times Time:45ms
cutoff: 21000 10times Time:51ms
cutoff: 21350 10times Time:27ms
cutoff: 21700 10times Time:25ms
cutoff: 22050 10times Time:22ms
cutoff: 22400 10times Time:23ms
cutoff: 22750 10times Time:26ms
cutoff: 23100 10times Time:21ms
cutoff: 23450 10times Time:21ms
cutoff: 23800 10times Time:22ms
cutoff: 24150 10times Time:22ms
cutoff: 24500 10times Time:22ms
cutoff: 24850 10times Time:40ms
cutoff: 25200 10times Time:22ms
cutoff: 25550 10times Time:22ms
cutoff: 25900 10times Time:24ms
cutoff: 26250 10times Time:23ms
cutoff: 26600 10times Time:23ms
cutoff: 26950 10times Time:22ms
cutoff: 27300 10times Time:21ms
cutoff: 27650 10times Time:23ms
cutoff: 28000 10times Time:22ms
cutoff: 28350 10times Time:22ms
cutoff: 28700 10times Time:23ms
cutoff: 29050 10times Time:23ms
cutoff: 29400 10times Time:23ms
cutoff: 29750 10times Time:22ms
cutoff: 30100 10times Time:23ms
cutoff: 30450 10times Time:22ms
cutoff: 30800 10times Time:26ms
cutoff: 31150 10times Time:28ms
cutoff: 31500 10times Time:30ms
cutoff: 31850 10times Time:31ms
cutoff: 32200 10times Time:29ms
cutoff: 32550 10times Time:30ms
cutoff: 32900 10times Time:28ms

```

result.csv - INFO6205/src/main/resources

Project Explorer

```

1 package edu.neu.coe.info6205.sort.par;
2
3 import java.io.File;
4
5 /**
6  * This code has been fleshed out by Ziyao Qiao. Thanks very much.
7  * TODO tidy it up a bit.
8 */
9
10 class ParSort {
11
12    /**
13     * public static int cutoff = 1000;
14     * public static ForkJoinPool myPool = new ForkJoinPool(16);
15     */
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);
22            CompletableFuture<int[]> parsort2 = parsort(array, from + (to - from) / 2, to);
23            CompletableFuture<int[]> parsort = parsort1.thenCombine(parsort2, (xs1, xs2) ->
24                int[] result = new int[xs1.length + xs2.length];
25                // TO IMPLEMENT
26                int i = 0;
27                int j = 0;
28                for (int k = 0; k < result.length; k++) {
29                    if (i < xs1.length) {
30                        result[k] = xs1[i++];
31                    } else if (j >= xs2.length) {
32                        result[k] = xs2[j++];
33                    } else if (xs2[j] < xs1[i]) {
34                        result[k] = xs2[j++];
35                    } else {
36                        result[k] = xs1[i++];
37                    }
38                }
39                return result;
40            });
41            parsort.whenComplete((result, throwable) -> System.arraycopy(result, 0, array,
42                //System.out.println("# threads: " + ForkJoinPool.commonPool().getRunningThreads());
43            );
44        }
45    }
46
47    private static CompletableFuture<int[]> parsort(int[] array, int from, int to) {
48
49        //System.out.println("Degree of parallelism: " + myPool.getParallelism());
50
51        return CompletableFuture.supplyAsync(
52            () -> {
53                int[] result = new int[to - from];
54                // TO IMPLEMENT
55                CustomResourceAllocation.from(result, a(result.length));
56            }
57        );
58    }
59
60
61
62
63
64
65
66

```

Console

```

terminated: Main [Java Application] /Users/User/Library/Java/J
Input size:40000
Thread count:16
cutoff: 17850 10times Time:187ms
cutoff: 18200 10times Time:107ms
cutoff: 18550 10times Time:48ms
cutoff: 18900 10times Time:27ms
cutoff: 19250 10times Time:26ms
cutoff: 19600 10times Time:55ms
cutoff: 19950 10times Time:26ms
cutoff: 20300 10times Time:28ms
cutoff: 20650 10times Time:24ms
cutoff: 21000 10times Time:34ms
cutoff: 21350 10times Time:41ms
cutoff: 21700 10times Time:48ms
cutoff: 22050 10times Time:25ms
cutoff: 22400 10times Time:22ms
cutoff: 22750 10times Time:23ms
cutoff: 23100 10times Time:23ms
cutoff: 23450 10times Time:23ms
cutoff: 23800 10times Time:23ms
cutoff: 24150 10times Time:24ms
cutoff: 24500 10times Time:25ms
cutoff: 24850 10times Time:25ms
cutoff: 25200 10times Time:24ms
cutoff: 25550 10times Time:24ms
cutoff: 25900 10times Time:23ms
cutoff: 26250 10times Time:22ms
cutoff: 26600 10times Time:22ms
cutoff: 26950 10times Time:22ms
cutoff: 27300 10times Time:23ms
cutoff: 27650 10times Time:23ms
cutoff: 28000 10times Time:51ms
cutoff: 28350 10times Time:33ms
cutoff: 28700 10times Time:32ms
cutoff: 29050 10times Time:28ms
cutoff: 29400 10times Time:27ms
cutoff: 29750 10times Time:31ms
cutoff: 30100 10times Time:31ms
cutoff: 30450 10times Time:38ms
cutoff: 30800 10times Time:35ms
cutoff: 31150 10times Time:33ms
cutoff: 31500 10times Time:31ms
cutoff: 31850 10times Time:31ms
cutoff: 32200 10times Time:30ms
cutoff: 32550 10times Time:31ms
cutoff: 32900 10times Time:33ms

```

Two screenshots of the Eclipse IDE interface showing the execution of a Java application named 'ParSort'.

Screenshot 1 (Top):

- Project Explorer:** Shows the project structure for 'INFO6205 [INFO6205 Fall2022]'. The 'src/main/java' folder contains 'ParSort.java' and 'Main.java'. Other files like 'result.csv' and various word count files are also listed.
- Code Editor:** The code for 'Main.java' is displayed. It includes imports for `java.io.BufferedReader` and `java.io.File`. The code implements a `main` method that processes command-line arguments, prints the degree of parallelism, generates random arrays, sorts them using a `ParSort` class, and then writes the results to a CSV file.
- Console:** The output shows the thread count and execution time for different cutoff values (from 2500 to 47000). The output ends with 'File created'.

```

package edu.neu.coe.info6205.sort.par;
import java.io.BufferedReader;
import java.io.File;

public class Main {
    public static void main(String[] args) {
        processArgs(args);
        System.out.println("Degree of parallelism: " + ForkJoinPool.commonPool().getParallelism());
        Random random = new Random();
        int[] array = new int[80000];
        ArrayList<Long> timelist = new ArrayList<>();
        System.out.println("Input size:" + array.length);
        System.out.println("Thread count:" + Thread.currentThread().getStackTrace().length);
        for (int j = 50; j < 100; j++) {
            ParSort.cutoff = 500 * (j + 1);
            long time = System.currentTimeMillis();
            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);
        }
        long endtime = System.currentTimeMillis();
        time = (endtime - startTime);
        timelist.add(time);
        System.out.println("cutoff: " + (ParSort.cutoff) + "\t\t10times Time:" + time +
    }
    try {
        FileOutputStream fis = new FileOutputStream("src/main/resources/result.csv");
        OutputStreamWriter isr = new OutputStreamWriter(fis);
        BufferedWriter bw = new BufferedWriter(isr);
        int j = 50;
        for (long t : timelist) {
            String content = (double) 500 * (j + 1) + "," + (double) t + "\n";
            j++;
            bw.write(content);
            bw.flush();
        }
        bw.close();
        System.out.println("File created");
    }
}

```

Screenshot 2 (Bottom):

- Project Explorer:** Same as Screenshot 1.
- Code Editor:** The code for 'Main.java' is identical to Screenshot 1.
- Console:** The output shows the execution time for different cutoff values (from 2500 to 47000), with a total execution time of 185ms.

```

package edu.neu.coe.info6205.sort.par;
import java.io.BufferedReader;
import java.io.File;

class ParSort {
    public static int cutoff = 1000;
    public static ForkJoinPool myPool = new ForkJoinPool(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(from, from + (to - from) / 2);
            CompletableFuture<int[]> parsort2 = parsort(from, 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] = xs1[i]++;
                    } else if (j <= xs2.length) {
                        result[k] = xs2[j]++;
                    } 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,
                //System.out.println("# threads: " + ForkJoinPool.commonPool().getRunningThreads());
                //parsort.join();
            });
        }
    }

    private static CompletableFuture<int[]> parsort(int[] array, int from, int to) {
        //System.out.println("Degree of parallelism: " + myPool.getParallelism());
        return CompletableFuture.supplyAsync(
            () -> {
                int[] result = new int[to - from];
                // TO IMPLEMENT
                CustomResourcePool(result, from, result.length);
                return result;
            }
        );
    }
}

```

Project Explorer

```

1 package edu.neu.coe.info6205.sort.par;
2
3 import java.io.BufferedReader;
4
5 /**
6  * This code has been fleshed out by Zixiao Qiao. Thanks very much.
7  * TODO tidy it up a bit.
8 */
9 public class Main {
10
11     public static void main(String[] args) {
12         processArgs(args);
13         //System.out.println("Degree of parallelism: " + ForkJoinPool.commonPool().getParallelism());
14         Random random = new Random();
15         int[] array = new int[80000];
16         ArrayList<Long> timelist = new ArrayList<>();
17         System.out.println("Input size:" + array.length);
18         System.out.println("Thread count:" + Thread.activeCount());
19         for (int j = 50; j < 100; j++) {
20             ParSort.cutoff = 500 * (j + 1);
21             // for (int i = 0; i < array.length; i++) array[i] = random.nextInt(10000000);
22             long time;
23             long startTime = System.currentTimeMillis();
24             for (int i = 0; i < array.length; i++) array[i] = random.nextInt(10000000);
25             ParSort.sort(array, 0, array.length);
26             long endTime = System.currentTimeMillis();
27             time = (endTime - startTime);
28             timelist.add(time);
29         }
30         long endtime = System.currentTimeMillis();
31         time = (endtime - startTime);
32         timelist.add(time);
33         System.out.println("cutoff: " + (ParSort.cutoff) + "\t\t10times Time:" + time +
34             "\n");
35     }
36
37     try {
38         FileOutputStream fis = new FileOutputStream("src/main/resources/result.csv");
39         OutputStreamWriter isr = new OutputStreamWriter(fis);
40         BufferedWriter bw = new BufferedWriter(isr);
41         int j = 50;
42         for (long i : timelist) {
43             String content = (double) 500 * (j + 1) + "," + (double) i + "\n";
44             j++;
45             bw.write(content);
46             bw.flush();
47         }
48         bw.close();
49         System.out.println("File created");
50     } catch (Exception e) {
51         e.printStackTrace();
52     }
53 }
54
55
56

```

Console

```

terminated: Main [Java Application] /Users/User/Library/Java/JDK/jdk-17.0.1/bin/java -jar /Users/User/Library/Java/JDK/jdk-17.0.1/libexec/jar/jar --main-class Main INFO6205 [INFO6205 Fall2022]
Thread count:8
cutoff: 25000 10times Time:342ms
cutoff: 26000 10times Time:13ms
cutoff: 26500 10times Time:57ms
cutoff: 27000 10times Time:47ms
cutoff: 27500 10times Time:48ms
cutoff: 28000 10times Time:78ms
cutoff: 28500 10times Time:54ms
cutoff: 29000 10times Time:57ms
cutoff: 29500 10times Time:54ms
cutoff: 30000 10times Time:54ms
cutoff: 30500 10times Time:49ms
cutoff: 31000 10times Time:51ms
cutoff: 31500 10times Time:47ms
cutoff: 32000 10times Time:45ms
cutoff: 32500 10times Time:44ms
cutoff: 33000 10times Time:46ms
cutoff: 33500 10times Time:44ms
cutoff: 34000 10times Time:45ms
cutoff: 34500 10times Time:46ms
cutoff: 35000 10times Time:47ms
cutoff: 35500 10times Time:42ms
cutoff: 36000 10times Time:39ms
cutoff: 36500 10times Time:45ms
cutoff: 37000 10times Time:43ms
cutoff: 37500 10times Time:40ms
cutoff: 38000 10times Time:44ms
cutoff: 38500 10times Time:46ms
cutoff: 39000 10times Time:55ms
cutoff: 39500 10times Time:48ms
cutoff: 40000 10times Time:45ms
cutoff: 40500 10times Time:57ms
cutoff: 41000 10times Time:52ms
cutoff: 41500 10times Time:48ms
cutoff: 42000 10times Time:46ms
cutoff: 42500 10times Time:45ms
cutoff: 43000 10times Time:44ms
cutoff: 43500 10times Time:47ms
cutoff: 44000 10times Time:47ms
cutoff: 44500 10times Time:52ms
cutoff: 45000 10times Time:66ms
cutoff: 45500 10times Time:62ms
cutoff: 46000 10times Time:54ms
cutoff: 46500 10times Time:54ms
cutoff: 47000 10times Time:52ms

```

Project Explorer

```

1 package edu.neu.coe.info6205.sort.par;
2
3 import java.io.File;
4
5 /**
6  * This code has been fleshed out by Zixiao Qiao. Thanks very much.
7  * TODO tidy it up a bit.
8 */
9 class ParSort {
10
11     public static int cutoff = 1000;
12     public static ForkJoinPool myPool = new ForkJoinPool(16);
13
14     public static void sort(int[] array, int from, int to) {
15         if (to - from < cutoff) Arrays.sort(array, from, to);
16         else {
17             //FIXME next few lines should be removed from public repo.
18             CompletableFuture<int[]> parsort1 = parsort(array, from, from + (to - from) / 2);
19             CompletableFuture<int[]> parsort2 = parsort(array, from + (to - from) / 2, to);
20             CompletableFuture<int[]> parsort = parsort1.thenCombine(parsort2, (xs1, xs2) ->
21                 int[] result = new int[xs1.length + xs2.length];
22                 // TO IMPLEMENT
23                 int i = 0;
24                 int j = 0;
25                 for (int k = 0; k < result.length; k++) {
26                     if (i <= xs1.length) {
27                         result[k] = xs1[i]++;
28                     } else if (j <= xs2.length) {
29                         result[k] = xs2[j]++;
30                     } else if (xs2[i] < xs1[j]) {
31                         result[k] = xs2[i]++;
32                     } else {
33                         result[k] = xs1[i++];
34                     }
35                 }
36                 return result;
37             });
38             parsort.whenComplete((result, throwable) -> System.arraycopy(result, 0, array,
39                 //System.out.println("# threads: " + ForkJoinPool.commonPool().getRunningThreads());
40                 from));
41             parsort.join();
42         }
43     }
44
45     private static CompletableFuture<int[]> parsort(int[] array, int from, int to) {
46
47         //System.out.println("Degree of parallelism: " + myPool.getParallelism());
48
49         return CompletableFuture.supplyAsync(
50             () -> {
51                 int[] result = new int[to - from];
52                 // TO IMPLEMENT
53                 Custom.arraycopy(result, from, array, from, to - from);
54             }
55         );
56     }
57
58
59
60
61
62
63
64
65
66

```

Console

```

terminated: Main [Java Application] /Users/User/Library/Java/JDK/jdk-17.0.1/bin/java -jar /Users/User/Library/Java/JDK/jdk-17.0.1/libexec/jar/jar --main-class Main INFO6205 [INFO6205 Fall2022]
Thread count:16
cutoff: 25000 10times Time:293ms
cutoff: 26000 10times Time:166ms
cutoff: 26500 10times Time:58ms
cutoff: 27000 10times Time:54ms
cutoff: 27500 10times Time:50ms
cutoff: 28000 10times Time:75ms
cutoff: 28500 10times Time:48ms
cutoff: 29000 10times Time:49ms
cutoff: 29500 10times Time:48ms
cutoff: 30000 10times Time:49ms
cutoff: 30500 10times Time:48ms
cutoff: 31000 10times Time:45ms
cutoff: 31500 10times Time:47ms
cutoff: 32000 10times Time:45ms
cutoff: 32500 10times Time:42ms
cutoff: 33000 10times Time:42ms
cutoff: 33500 10times Time:43ms
cutoff: 34000 10times Time:42ms
cutoff: 34500 10times Time:40ms
cutoff: 35000 10times Time:41ms
cutoff: 35500 10times Time:43ms
cutoff: 36000 10times Time:45ms
cutoff: 36500 10times Time:45ms
cutoff: 37000 10times Time:42ms
cutoff: 37500 10times Time:44ms
cutoff: 38000 10times Time:42ms
cutoff: 38500 10times Time:43ms
cutoff: 39000 10times Time:41ms
cutoff: 39500 10times Time:43ms
cutoff: 40000 10times Time:45ms
cutoff: 40500 10times Time:55ms
cutoff: 41000 10times Time:51ms
cutoff: 41500 10times Time:51ms
cutoff: 42000 10times Time:46ms
cutoff: 42500 10times Time:51ms
cutoff: 43000 10times Time:47ms
cutoff: 43500 10times Time:48ms
cutoff: 44000 10times Time:47ms
cutoff: 44500 10times Time:49ms
cutoff: 45000 10times Time:47ms
cutoff: 45500 10times Time:46ms
cutoff: 46000 10times Time:51ms
cutoff: 46500 10times Time:51ms
cutoff: 47000 10times Time:52ms

```

Two screenshots of the Eclipse IDE interface showing the code editor and console output for the `ParSort.java` file.

Screenshot 1 (Top):

- Project Explorer:** Shows the project structure for `INFO6205 [INFO6205 Fall2022]`. It includes `src/main/java`, `src/main/resources` (containing `3000-common-words.txt`, `config.ini`, `eng-uk_web_2002_100K-sentences.txt`, `eng-uk_web_2002_100K-words.txt`, `eng-uk_web_2002_10K-sentences.txt`, `eng-uk_web_2002_1M-words.txt`, `log4j.properties`, `num1000.txt`, `num500.txt`, `nums2000`, `nums4000`, `nums8000`, `result.csv`), `src/test/java`, `src/test/resources` (containing `JRE System Library [JavaSE-1.8]`, `Maven Dependencies`, `logs`, `src`, `target`), and various assignment files like `Assignment2FinalReport.docx`, `Assignment3FinalReport.docx.pdf`, `Assignment4FinalReport.docx (1).pdf`, `LICENSE`, and `pom.xml`.
- Code Editor (ParSort.java):**

```

1 package edu.neu.coe.info6205.sort.par;
2
3 import java.io.File;
4
5 /**
6  * This code has been fleshed out by Ziyao Qiao. Thanks very much.
7  * TODO tidy it up a bit.
8 */
9 class ParSort {
10
11     public static int cutoff = 1000;
12     public static ForkJoinPool myPool = new ForkJoinPool(2);
13
14     public static void sort(int[] array, int from, int to) {
15         if (to - from < cutoff) Arrays.sort(array, from, to);
16         else {
17             //FIXME next few lines should be removed from public repo.
18             CompleteableFuture<int[]> parsort1 = parsort(array, from, from + (to - from) / 2);
19             CompleteableFuture<int[]> parsort2 = parsort(array, from + (to - from) / 2, to);
20             CompleteableFuture<int[]> parsort = parsort1.thenCombine(parsort2, (xs1, xs2) ->
21                 int[] result = new int[xs1.length + xs2.length];
22                 // TO IMPLEMENT
23                 int i = 0;
24                 int j = 0;
25                 for (int k = 0; k < result.length; k++) {
26                     if (i < xs1.length) {
27                         result[k] = xs1[i]++;
28                     } else if (j >= xs2.length) {
29                         result[k] = xs2[j]++;
30                     } else if (xs2[j] < xs1[i]) {
31                         result[k] = xs2[j]++;
32                     } else {
33                         result[k] = xs1[i++];
34                     }
35                 }
36                 return result;
37             });
38             parsort.whenComplete((result, throwable) -> System.arraycopy(result, 0, array,
39                 //System.out.println("# threads: " + ForkJoinPool.commonPool().getRunningThreadCount());
40             );
41             parsort.join();
42         }
43     }
44
45     private static CompleteableFuture<int[]> parsort(int[] array, int from, int to) {
46
47         //System.out.println("Degree of parallelism: " + myPool.getParallelism());
48
49         return CompletableFuture.supplyAsync(
50             () -> {
51                 int[] result = new int[to - from];
52                 // TO IMPLEMENT
53                 CustomResourceAllocationFromResult(result, from, result.length);
54             }
55         );
56     }
57
58     //System.out.println("Degree of parallelism: " + myPool.getParallelism());
59
60     return CompletableFuture.supplyAsync(
61         () -> {
62             int[] result = new int[to - from];
63             // TO IMPLEMENT
64             CustomResourceAllocationFromResult(result, from, result.length);
65         }
66     );
67 }

```
- Console:** Shows the execution results for `cutoff` values from 51000 to 94000. The output shows 10 times of execution time for each cutoff value.

Screenshot 2 (Bottom):

- Project Explorer:** Same as Screenshot 1.
- Code Editor (ParSort.java):**

```

1 package edu.neu.coe.info6205.sort.par;
2
3 import java.io.File;
4
5 /**
6  * This code has been fleshed out by Ziyao Qiao. Thanks very much.
7  * TODO tidy it up a bit.
8 */
9 class ParSort {
10
11     public static int cutoff = 1000;
12     public static ForkJoinPool myPool = new ForkJoinPool(4);
13
14     public static void sort(int[] array, int from, int to) {
15         if (to - from < cutoff) Arrays.sort(array, from, to);
16         else {
17             //FIXME next few lines should be removed from public repo.
18             CompleteableFuture<int[]> parsort1 = parsort(array, from, from + (to - from) / 2);
19             CompleteableFuture<int[]> parsort2 = parsort(array, from + (to - from) / 2, to);
20             CompleteableFuture<int[]> parsort = parsort1.thenCombine(parsort2, (xs1, xs2) ->
21                 int[] result = new int[xs1.length + xs2.length];
22                 // TO IMPLEMENT
23                 int i = 0;
24                 int j = 0;
25                 for (int k = 0; k < result.length; k++) {
26                     if (i < xs1.length) {
27                         result[k] = xs1[i]++;
28                     } else if (j >= xs2.length) {
29                         result[k] = xs2[j]++;
30                     } else if (xs2[j] < xs1[i]) {
31                         result[k] = xs2[j]++;
32                     } else {
33                         result[k] = xs1[i++];
34                     }
35                 }
36                 return result;
37             });
38             parsort.whenComplete((result, throwable) -> System.arraycopy(result, 0, array,
39                 //System.out.println("# threads: " + ForkJoinPool.commonPool().getRunningThreadCount());
40             );
41             parsort.join();
42         }
43     }
44
45     private static CompleteableFuture<int[]> parsort(int[] array, int from, int to) {
46
47         //System.out.println("Degree of parallelism: " + myPool.getParallelism());
48
49         return CompletableFuture.supplyAsync(
50             () -> {
51                 int[] result = new int[to - from];
52                 // TO IMPLEMENT
53                 CustomResourceAllocationFromResult(result, from, result.length);
54             }
55         );
56     }
57
58     //System.out.println("Degree of parallelism: " + myPool.getParallelism());
59
60     return CompletableFuture.supplyAsync(
61         () -> {
62             int[] result = new int[to - from];
63             // TO IMPLEMENT
64             CustomResourceAllocationFromResult(result, from, result.length);
65         }
66     );
67 }

```
- Console:** Shows the execution results for `cutoff` values from 51000 to 94000. The output shows 10 times of execution time for each cutoff value.

Two screenshots of the Eclipse IDE interface showing the execution of the `ParSort` Java application.

Screenshot 1 (Top):

- Project Explorer:** Shows the project structure for `INFO6205 [INFO6205 Fall2022]`. It includes source code, resources, and test files. A file named `result.csv` is visible.
- Code Editor:** Displays the `ParSort.java` file. The code implements a parallel sorting algorithm using `ForkJoinPool`. It includes comments about TODO items and fixme notes.
- Console:** Shows the output of the application. It prints the input size (12000), thread count (12000), and a series of log entries for each iteration of the sorting process. The log entries show the cutoff value, the number of times it was used, and the time taken for that iteration.

cutoff	10times Time
51000	429ms
52000	162ms
53000	86ms
54000	89ms
55000	70ms
56000	68ms
57000	70ms
58000	74ms
59000	69ms
60000	67ms
61000	79ms
62000	75ms
63000	83ms
64000	78ms
65000	85ms
66000	75ms
67000	73ms
68000	79ms
69000	76ms
70000	73ms
71000	72ms
72000	75ms
73000	79ms
74000	78ms
75000	73ms
76000	80ms
77000	75ms
78000	76ms
79000	73ms
80000	77ms
81000	79ms
82000	74ms
83000	76ms
84000	81ms
85000	78ms
86000	82ms
87000	78ms
88000	84ms
89000	79ms
90000	73ms
91000	80ms
92000	70ms
93000	74ms
94000	76ms

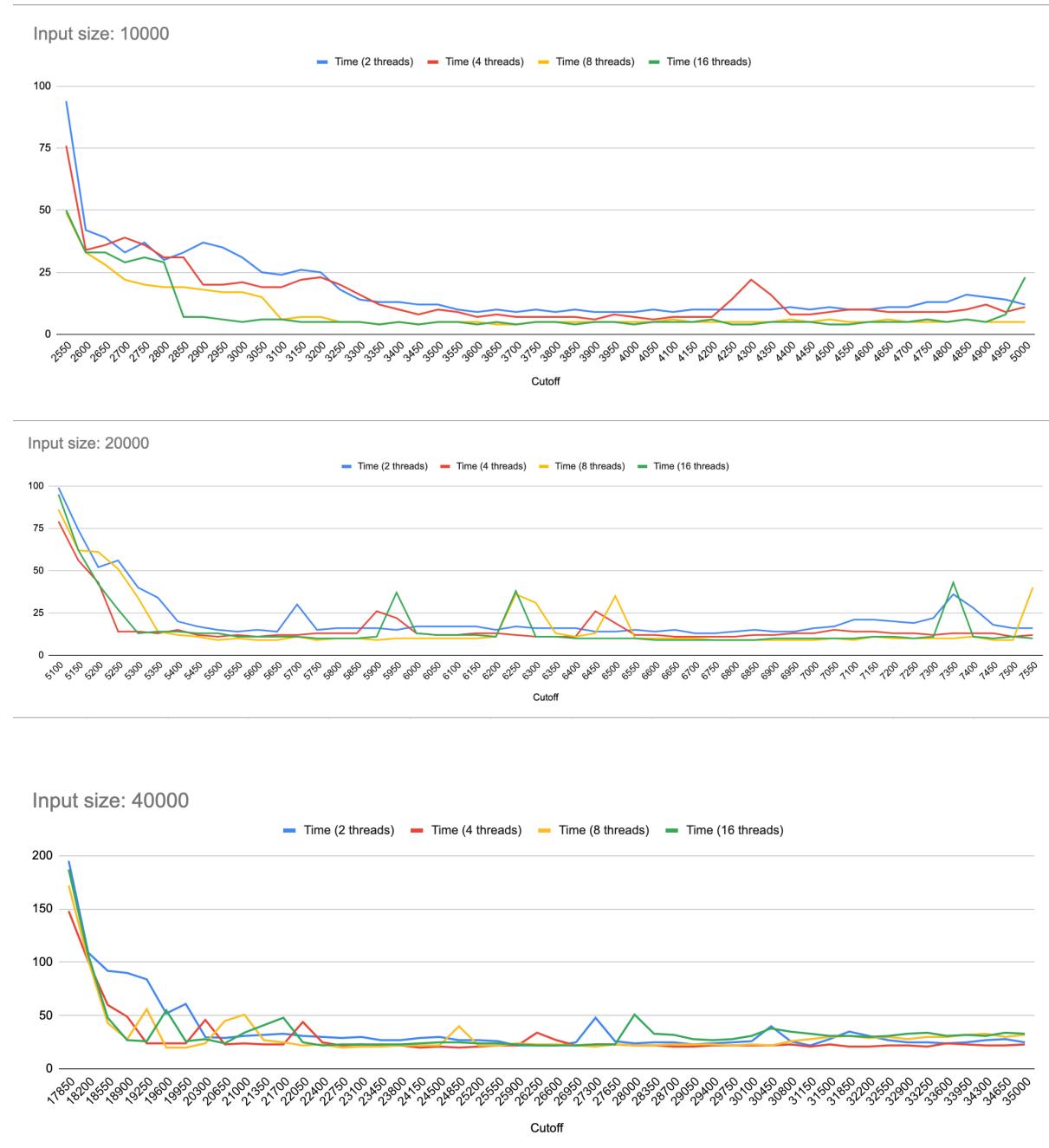
Screenshot 2 (Bottom):

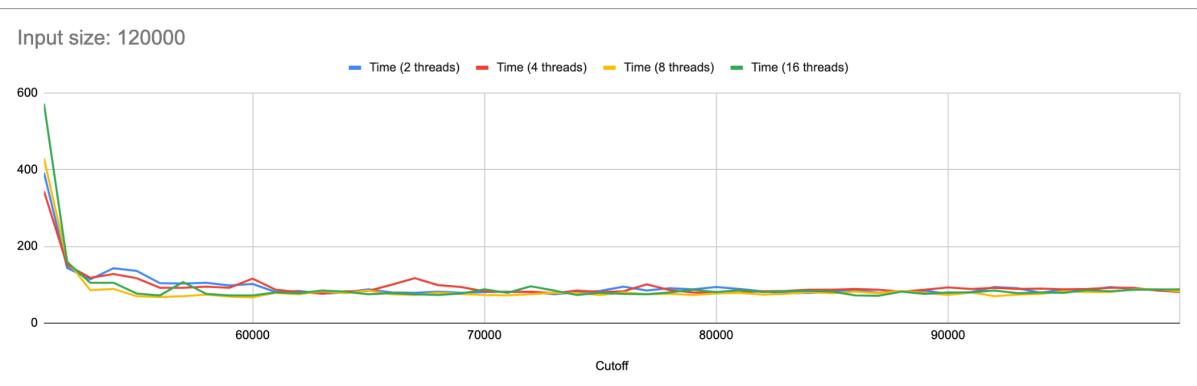
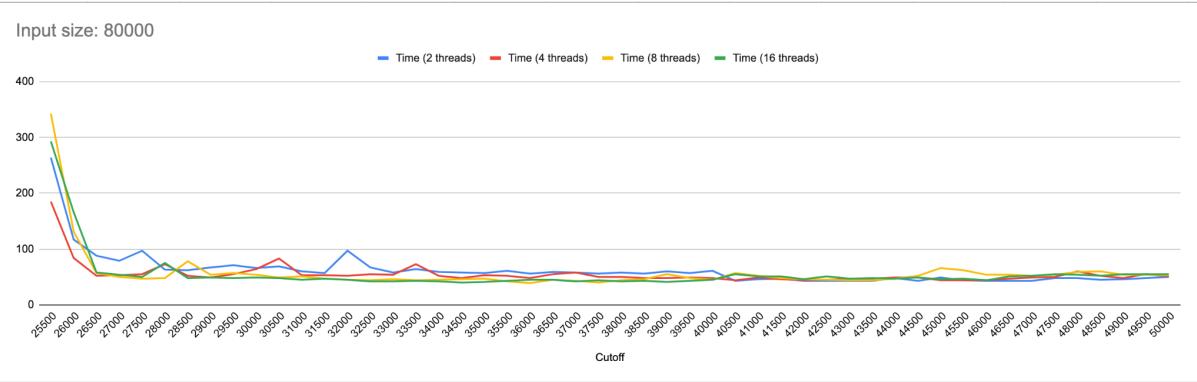
- Project Explorer:** Same as Screenshot 1.
- Code Editor:** Same as Screenshot 1.
- Console:** Shows the same log output as Screenshot 1, indicating the application has terminated. The log entries show the cutoff values and the corresponding execution times for 10 iterations.

cutoff	10times Time
51000	572ms
52000	159ms
53000	105ms
54000	105ms
55000	77ms
56000	107ms
57000	72ms
58000	76ms
59000	72ms
60000	72ms
61000	80ms
62000	78ms
63000	85ms
64000	82ms
65000	75ms
66000	78ms
67000	75ms
68000	73ms
69000	77ms
70000	88ms
71000	79ms
72000	96ms
73000	85ms
74000	73ms
75000	79ms
76000	76ms
77000	75ms
78000	80ms
79000	87ms
80000	81ms
81000	86ms
82000	83ms
83000	83ms
84000	84ms
85000	82ms
86000	72ms
87000	71ms
88000	82ms
89000	76ms
90000	80ms
91000	80ms
92000	85ms
93000	78ms
94000	80ms

- CSV files attached on github for a better view of the cut-off values and time.

Graphical Representation:





Time vs. Input size

