Assignment 5

Main Code Changes:

```
package edu.neu.coe.info6205.sort.par;

import java.io.BufferedWriter;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.OutputStreamWriter;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;
import java.util.Random;
import java.util.concurrent.ForkJoinPool;

/**
* This code has been fleshed out by Ziyao Qiao. Thanks very much.
* CONSIDER tidy it up a bit.
*/
public class Main {

   public static void main(String[] args) {
       int size = 1600000;
       processArgs(args);
       System.out.println("Array size: "+size);
       int thread = 2;
       while(thread < 128) {
           ForkJoinPool pool = new ForkJoinPool(thread);
           System.out.println("Degree of parallelism: " +
pool.getParallelism());
           Random random = new Random();
           int[] array = new int[size];
           ArrayList<Long> timeList = new ArrayList<>();
           for (int j = 0; j < 10; j++) {
               ParSort.cutoff = 4000 * (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, pool);
           }
           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/result.csv");
            OutputStreamWriter isr = new OutputStreamWriter(fis);
            BufferedWriter bw = new BufferedWriter(isr);
            int j = 0;
            for (long i : timeList) {
                String content = (double) 10000 * (j + 1) / 2000000 + "," +
(double) i / 10 + "\n";
                j++;
                bw.write(content);
                bw.flush();
            }
            bw.close();

        } catch (IOException e) {
            e.printStackTrace();
        }
        thread *= 2;
    }
    }

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

    private static String[] processArg(String[] xs) {
        String[] result = new String[0];
        System.arraycopy(xs, 2, result, 0, xs.length - 2);
        processCommand(xs[0], xs[1]);
        return result;
    }

    private static void processCommand(String x, String y) {
        if (x.equalsIgnoreCase("N")) setConfig(x, Integer.parseInt(y));
        else
            // TODO sort this out
            if (x.equalsIgnoreCase("P")) //noinspection
ResultOfMethodCallIgnored
                ForkJoinPool.getCommonPoolParallelism();
    }

    private static void setConfig(String x, int i) {
```

```
        configuration.put(x, i);
    }

    @SuppressWarnings("MismatchedQueryAndUpdateOfCollection")
    private static final Map<String, Integer> configuration = new HashMap<>();


}
```

ParSort Code Changes:

```
package edu.neu.coe.info6205.sort.par;

import java.util.Arrays;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.ForkJoinPool;

/**
* This code has been fleshed out by Ziyao Qiao. Thanks very much.
* CONSIDER tidy it up a bit.
*/
class ParSort {

    public static int cutoff = 1000;

    public static void sort(int[] array, int from, int to, ForkJoinPool pool) {
        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, pool); // TO IMPLEMENT
            CompletableFuture<int[]> parsort2 = parsort(array, from + (to -
from) / 2, to, pool); // TO IMPLEMENT
            CompletableFuture<int[]> parsort = parsort1.thenCombine(parsort2,
(xs1, xs2) -> {
                int[] result = new int[xs1.length + xs2.length];
                // TO IMPLEMENT
                int i = 0;
                int j = 0;
                for (int k = 0; k < result.length; k++) {
                    if (i >= xs1.length) {
                        result[k] = xs2[j++];
                    } else if (j >= xs2.length) {
                        result[k] = xs1[i++];
                    } else if (xs2[j] < xs1[i]) {
                        result[k] = xs2[j++];
                    } else {
                        result[k] = xs1[i++];
                    }
```

```
                }
                return result;
            });

            parsort.whenComplete((result, throwable) -> System.arraycopy(result,
0, array, from, result.length));
//            System.out.println("# threads: "+
ForkJoinPool.commonPool().getRunningThreadCount());
            parsort.join();
        }
    }

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

Output:

Array size: 400000
Degree of parallelism: 2
cutoff:4000 10times Time:794ms
cutoff:8000 10times Time:525ms
cutoff:12000 10times Time:357ms
cutoff:16000 10times Time:318ms
cutoff:20000 10times Time:239ms
cutoff:24000 10times Time:210ms
cutoff:28000 10times Time:210ms
cutoff:32000 10times Time:213ms
cutoff:36000 10times Time:236ms
cutoff:40000 10times Time:225ms
Degree of parallelism: 4
cutoff:4000 10times Time:223ms
cutoff:8000 10times Time:260ms
cutoff:12000 10times Time:217ms
cutoff:16000 10times Time:218ms
cutoff:20000 10times Time:263ms

cutoff:24000 10times Time:328ms
cutoff:28000 10times Time:285ms
cutoff:32000 10times Time:221ms
cutoff:36000 10times Time:193ms
cutoff:40000 10times Time:210ms
Degree of parallelism: 8
cutoff:4000 10times Time:259ms
cutoff:8000 10times Time:249ms
cutoff:12000 10times Time:182ms
cutoff:16000 10times Time:168ms
cutoff:20000 10times Time:170ms
cutoff:24000 10times Time:192ms
cutoff:28000 10times Time:203ms
cutoff:32000 10times Time:183ms
cutoff:36000 10times Time:170ms
cutoff:40000 10times Time:166ms
Degree of parallelism: 16
cutoff:4000 10times Time:194ms
cutoff:8000 10times Time:181ms
cutoff:12000 10times Time:229ms
cutoff:16000 10times Time:197ms
cutoff:20000 10times Time:172ms
cutoff:24000 10times Time:168ms
cutoff:28000 10times Time:171ms
cutoff:32000 10times Time:170ms
cutoff:36000 10times Time:171ms
cutoff:40000 10times Time:186ms
Degree of parallelism: 32
cutoff:4000 10times Time:227ms
cutoff:8000 10times Time:190ms
cutoff:12000 10times Time:205ms
cutoff:16000 10times Time:236ms
cutoff:20000 10times Time:224ms
cutoff:24000 10times Time:180ms
cutoff:28000 10times Time:181ms
cutoff:32000 10times Time:211ms
cutoff:36000 10times Time:178ms
cutoff:40000 10times Time:167ms
Degree of parallelism: 64
cutoff:4000 10times Time:194ms
cutoff:8000 10times Time:208ms
cutoff:12000 10times Time:232ms
cutoff:16000 10times Time:189ms
cutoff:20000 10times Time:196ms

cutoff:24000 10times Time:198ms
cutoff:28000 10times Time:172ms
cutoff:32000 10times Time:178ms
cutoff:36000 10times Time:189ms
cutoff:40000 10times Time:171ms

—----------

Array size: 800000
Degree of parallelism: 2
cutoff:4000 10times Time:1254ms
cutoff:8000 10times Time:847ms
cutoff:12000 10times Time:488ms
cutoff:16000 10times Time:483ms
cutoff:20000 10times Time:437ms
cutoff:24000 10times Time:404ms
cutoff:28000 10times Time:445ms
cutoff:32000 10times Time:461ms
cutoff:36000 10times Time:431ms
cutoff:40000 10times Time:412ms
Degree of parallelism: 4
cutoff:4000 10times Time:369ms
cutoff:8000 10times Time:342ms
cutoff:12000 10times Time:431ms
cutoff:16000 10times Time:345ms
cutoff:20000 10times Time:326ms
cutoff:24000 10times Time:347ms
cutoff:28000 10times Time:361ms
cutoff:32000 10times Time:357ms
cutoff:36000 10times Time:349ms
cutoff:40000 10times Time:357ms
Degree of parallelism: 8
cutoff:4000 10times Time:504ms
cutoff:8000 10times Time:424ms
cutoff:12000 10times Time:401ms
cutoff:16000 10times Time:405ms
cutoff:20000 10times Time:377ms
cutoff:24000 10times Time:385ms
cutoff:28000 10times Time:376ms
cutoff:32000 10times Time:366ms
cutoff:36000 10times Time:360ms
cutoff:40000 10times Time:373ms
Degree of parallelism: 16
cutoff:4000 10times Time:468ms

cutoff：8000 10times Time:405ms
cutoff：12000 10times Time:391ms
cutoff：16000 10times Time:365ms
cutoff：20000 10times Time:391ms
cutoff：24000 10times Time:386ms
cutoff：28000 10times Time:360ms
cutoff：32000 10times Time:362ms
cutoff：36000 10times Time:362ms
cutoff：40000 10times Time:397ms
Degree of parallelism: 32
cutoff：4000 10times Time:494ms
cutoff：8000 10times Time:411ms
cutoff：12000 10times Time:429ms
cutoff：16000 10times Time:364ms
cutoff：20000 10times Time:389ms
cutoff：24000 10times Time:394ms
cutoff：28000 10times Time:375ms
cutoff：32000 10times Time:376ms
cutoff：36000 10times Time:361ms
cutoff：40000 10times Time:368ms
Degree of parallelism: 64
cutoff：4000 10times Time:467ms
cutoff：8000 10times Time:421ms
cutoff：12000 10times Time:380ms
cutoff：16000 10times Time:385ms
cutoff：20000 10times Time:392ms
cutoff：24000 10times Time:384ms
cutoff：28000 10times Time:359ms
cutoff：32000 10times Time:361ms
cutoff：36000 10times Time:412ms
cutoff：40000 10times Time:345ms

—----------

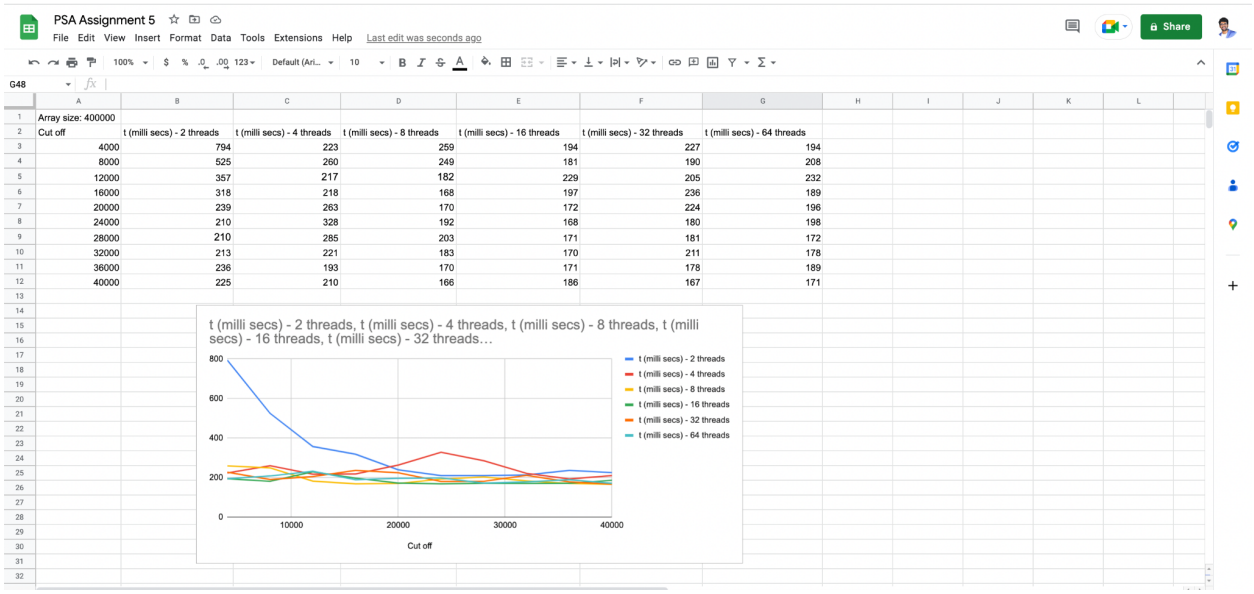
Array size: 1600000
Degree of parallelism: 2
cutoff：4000 10times Time:2870ms
cutoff：8000 10times Time:984ms
cutoff：12000 10times Time:758ms
cutoff：16000 10times Time:818ms
cutoff：20000 10times Time:770ms
cutoff：24000 10times Time:788ms
cutoff：28000 10times Time:786ms
cutoff：32000 10times Time:798ms

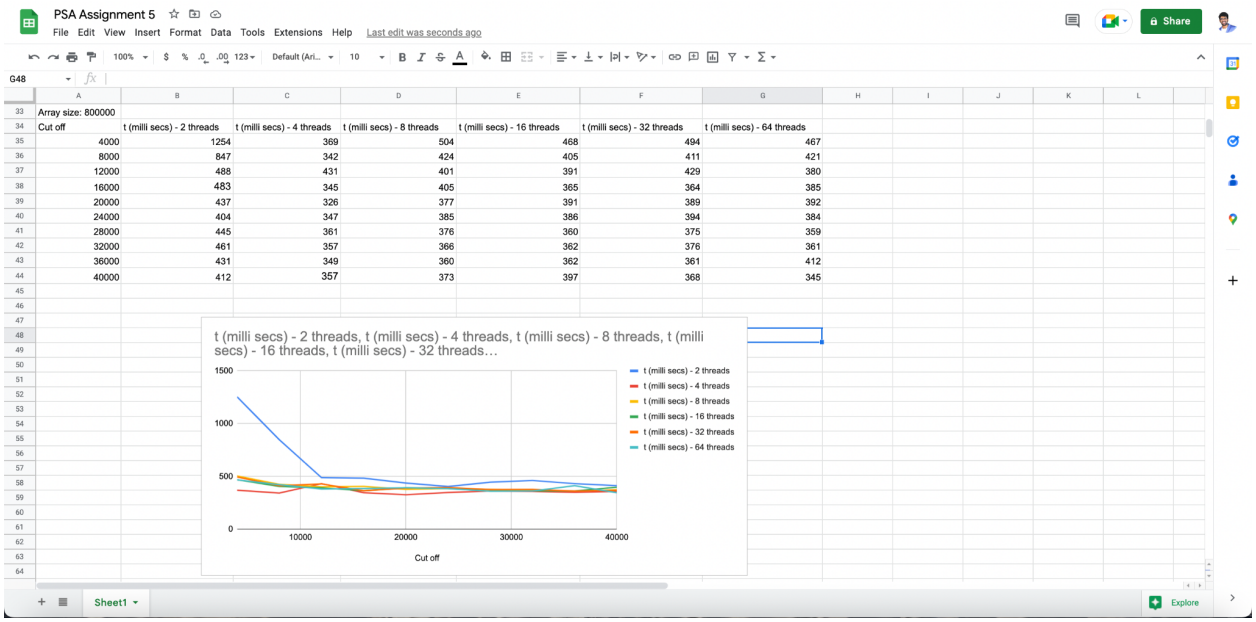
cutoff:36000 10times Time:794ms
cutoff:40000 10times Time:786ms
Degree of parallelism: 4
cutoff:4000 10times Time:757ms
cutoff:8000 10times Time:682ms
cutoff:12000 10times Time:683ms
cutoff:16000 10times Time:693ms
cutoff:20000 10times Time:695ms
cutoff:24000 10times Time:697ms
cutoff:28000 10times Time:713ms
cutoff:32000 10times Time:696ms
cutoff:36000 10times Time:741ms
cutoff:40000 10times Time:683ms
Degree of parallelism: 8
cutoff:4000 10times Time:895ms
cutoff:8000 10times Time:1231ms
cutoff:12000 10times Time:1053ms
cutoff:16000 10times Time:866ms
cutoff:20000 10times Time:838ms
cutoff:24000 10times Time:814ms
cutoff:28000 10times Time:752ms
cutoff:32000 10times Time:846ms
cutoff:36000 10times Time:754ms
cutoff:40000 10times Time:778ms
Degree of parallelism: 16
cutoff:4000 10times Time:957ms
cutoff:8000 10times Time:822ms
cutoff:12000 10times Time:817ms
cutoff:16000 10times Time:789ms
cutoff:20000 10times Time:808ms
cutoff:24000 10times Time:1025ms
cutoff:28000 10times Time:857ms
cutoff:32000 10times Time:615ms
cutoff:36000 10times Time:614ms
cutoff:40000 10times Time:618ms
Degree of parallelism: 32
cutoff:4000 10times Time:753ms
cutoff:8000 10times Time:728ms
cutoff:12000 10times Time:667ms
cutoff:16000 10times Time:682ms
cutoff:20000 10times Time:662ms
cutoff:24000 10times Time:658ms
cutoff:28000 10times Time:642ms
cutoff:32000 10times Time:773ms

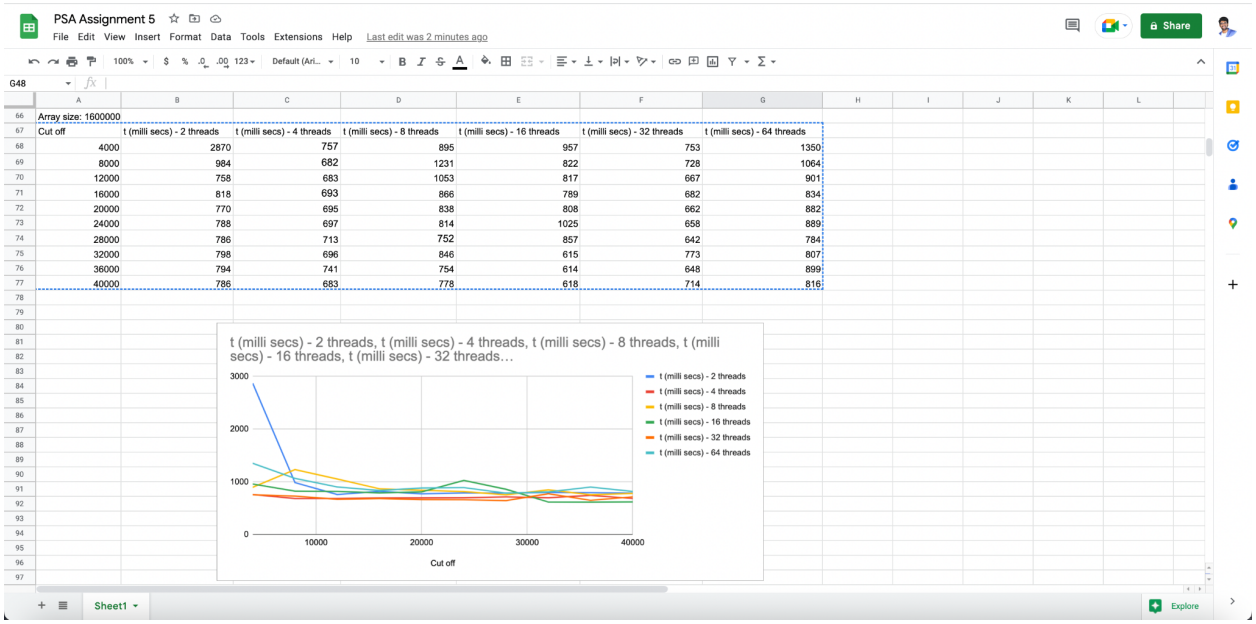
cutoff：36000　　10times Time:648ms
cutoff：40000　　10times Time:714ms
Degree of parallelism: 64
cutoff：4000　　10times Time:1350ms
cutoff：8000　　10times Time:1064ms
cutoff：12000　　10times Time:901ms
cutoff：16000　　10times Time:834ms
cutoff：20000　　10times Time:882ms
cutoff：24000　　10times Time:889ms
cutoff：28000　　10times Time:784ms
cutoff：32000　　10times Time:807ms
cutoff：36000　　10times Time:899ms
cutoff：40000　　10times Time:816ms

——————————

Screenshots:







Observations:

a) It can be inferred from the results presented above and the graphs that: For different sizes of arrays, changing the cutoff value and the number of threads does not enhance performance. Hence, keeping 4 threads is the best option.
b) With reference to the graph, it can be said that the lowest performance time is attained for the cutoff value of 25% of the array's size.
c) Hence, the best results can be seen when the cutoff value is 25% and there are 4 threads.