Program Structures & Algorithms

Spring 2022

Assignment No. 4

Name: Eswar Saladi

(NUID): 002966034

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.

GitHub Link: https://github.com/eswarsaladi/parallelsorting-assignment

Code:

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



import java.io.BufferedWriter;

import java.io.FileOutputStream;

import java.io.IOException;
```

```java
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.
* TODO tidy it up a bit.
*/
public class Main {


   public static void main(String[] args) {
       processArgs(args);
       System.out.println("Degree of parallelism: " +
ForkJoinPool.getCommonPoolParallelism());
       Random random = new Random();
       int[] array = new int[5000000];
       for (int threadCount = 1; threadCount <= 64; threadCount *= 2) {


           ArrayList<Long> timeList = new ArrayList<>();


           ForkJoinPool myPool = new ForkJoinPool(threadCount);
           for (int j = 1; j <= 10; j += 1) {
               ParSort.cutoff = 50000 * j;
               ParSort.threadPool = myPool;
               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/result" +
threadCount + ".csv");

                OutputStreamWriter isr = new OutputStreamWriter(fis);

                BufferedWriter bw = new BufferedWriter(isr);



                int j = 1;

                for (long i : timeList) {

                    String content = (double) 50000 * j / 5000000 + "," + (double) i
/ 10 + "\n";

                    j++;

                    bw.write(content);

                    bw.flush();
                }
                bw.close();

            } catch (IOException e) {
                e.printStackTrace();
            }

        }
    }

    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 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<>();



}
```

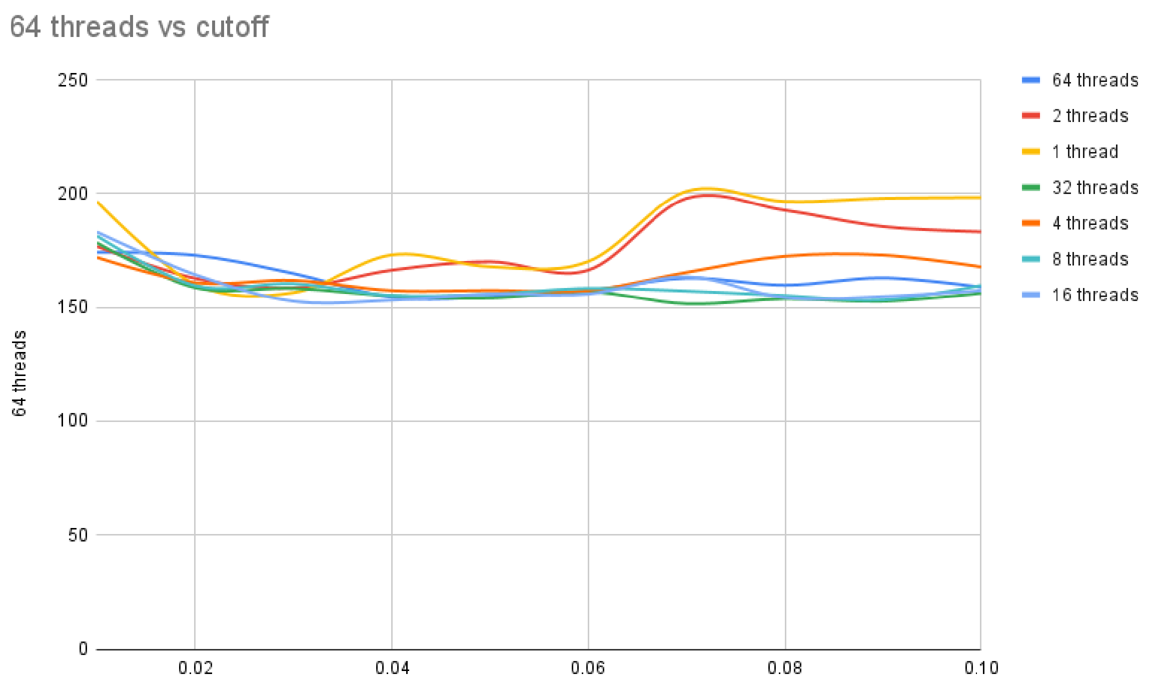
```
package edu.neu.coe.info6205.sort.par;



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.

*/

class ParSort {



   public static int cutoff = 1000;

   public static ForkJoinPool threadPool;



   public static void sort(int[] array, int from, int to) {
```

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

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

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

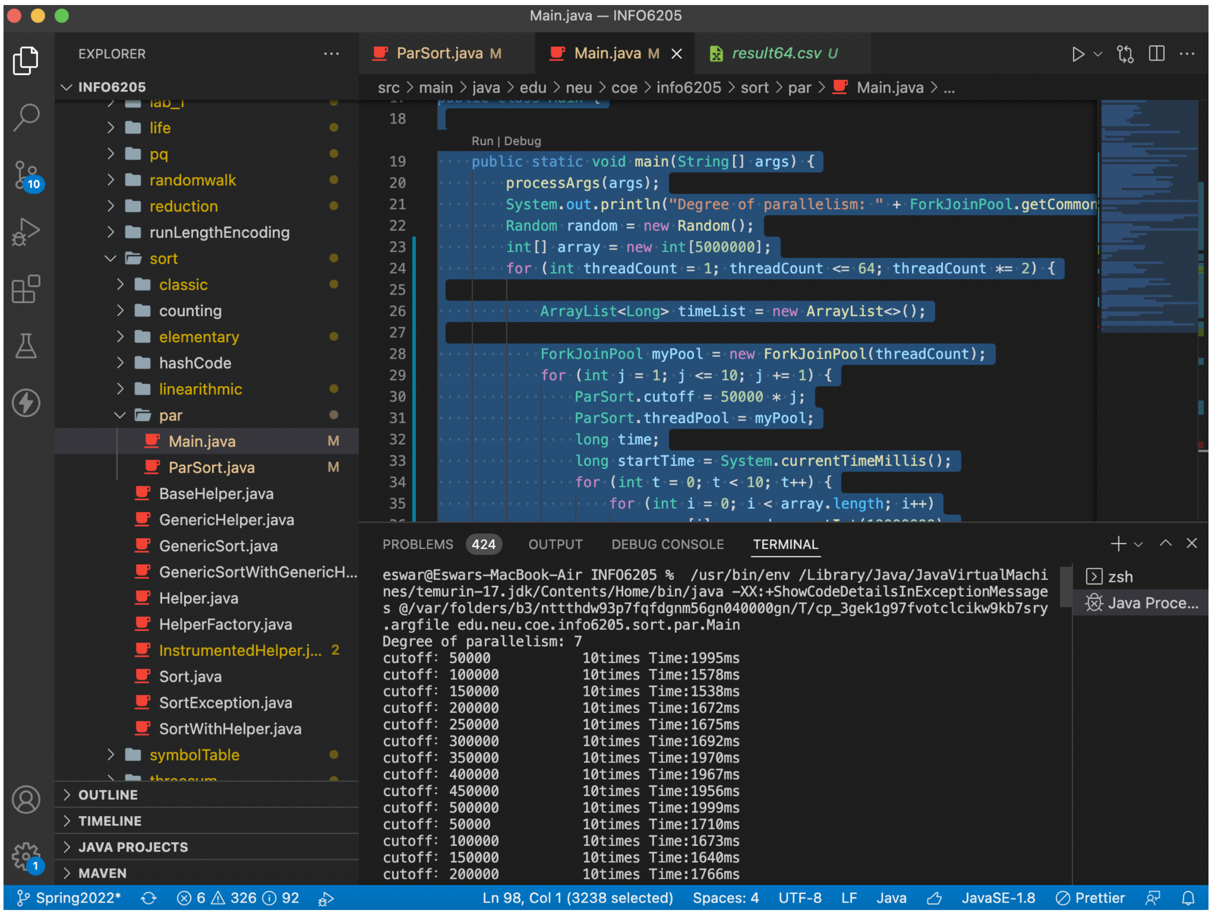
Observations: Excel Sheet and Graphs

cutoff	1 thread	2 threads	4 threads	8 threads	16 threads	32 threads	64 threads	avg
0.01	196.5	176.8	172	181.4	183.1	178.4	174.2	180.3428571
0.02	160.1	162.8	160.9	159.6	164.4	158.7	172.9	162.7714286
0.03	156.5	158.6	161.9	160.4	152.7	158.4	164.9	159.0571429
0.04	173.1	166.4	157.4	155.2	153.2	155.1	154.5	159.2714286

0.05	167.9	170.1	157.5	155.8	155.9	154.1	154.6	159.4142857
0.06	170.2	166.4	157.4	158.4	155.9	156.9	157.3	160.3571429
0.07	201	197.9	165.4	157.2	163.5	151.6	162.9	171.3571429
0.08	196.5	192.7	172.5	155.2	154.3	153.8	159.8	169.2571429
0.09	197.9	185.5	173	153.3	154.8	152.7	163	168.6
0.1	198.3	183.2	167.8	159.7	157.4	156.3	158.9	168.8
Min Time	156.5	158.6	157.4	153.3	152.7	151.6	154.5	**159.0571429**
Max Time	201	197.9	173	181.4	183.1	178.4	174.2	**180.3428571**
Avg Time	181.8	176.04	164.58	159.62	159.52	157.6	162.3	

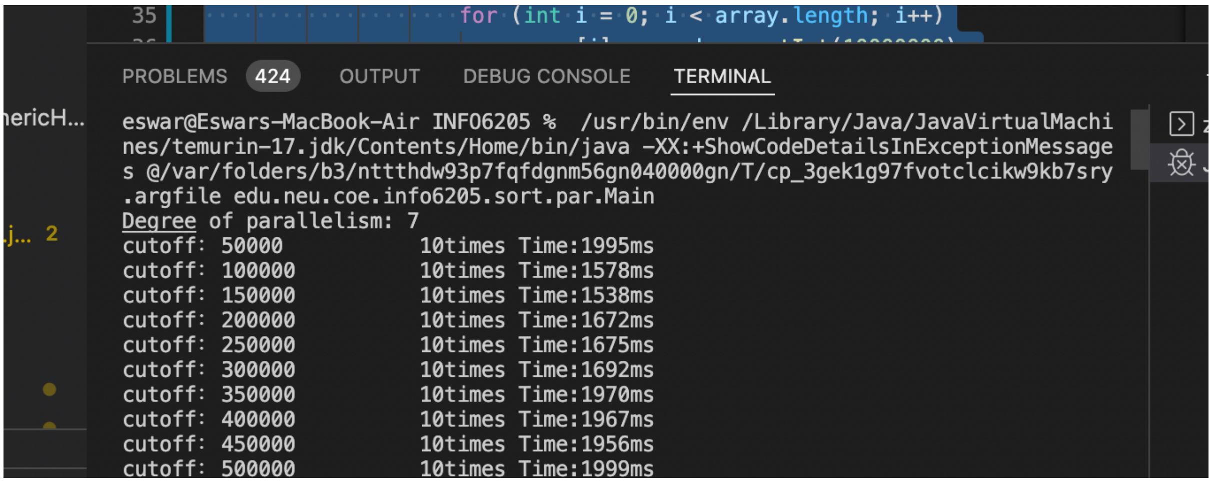


Screenshots:



```java
    public static void main(String[] args) {
        processArgs(args);
        System.out.println("Degree of parallelism: " + ForkJoinPool.getCommon
        Random random = new Random();
        int[] array = new int[5000000];
        for (int threadCount = 1; threadCount <= 64; threadCount *= 2) {

            ArrayList<Long> timeList = new ArrayList<>();

            ForkJoinPool myPool = new ForkJoinPool(threadCount);
            for (int j = 1; j <= 10; j += 1) {
                ParSort.cutoff = 50000 * j;
                ParSort.threadPool = myPool;
                long time;
                long startTime = System.currentTimeMillis();
                for (int t = 0; t < 10; t++) {
                    for (int i = 0; i < array.length; i++)
```

```
eswar@Eswars-MacBook-Air INFO6205 %  /usr/bin/env /Library/Java/JavaVirtualMachi
nes/temurin-17.jdk/Contents/Home/bin/java -XX:+ShowCodeDetailsInExceptionMessage
s @/var/folders/b3/ntthdw93p7fqfdgnm56gn040000gn/T/cp_3gek1g97fvotclcikw9kb7sry
.argfile edu.neu.coe.info6205.sort.par.Main
Degree of parallelism: 7
cutoff: 50000           10times Time:1995ms
cutoff: 100000          10times Time:1578ms
cutoff: 150000          10times Time:1538ms
cutoff: 200000          10times Time:1672ms
cutoff: 250000          10times Time:1675ms
cutoff: 300000          10times Time:1692ms
cutoff: 350000          10times Time:1970ms
cutoff: 400000          10times Time:1967ms
cutoff: 450000          10times Time:1956ms
cutoff: 500000          10times Time:1999ms
cutoff: 50000           10times Time:1710ms
cutoff: 100000          10times Time:1673ms
cutoff: 150000          10times Time:1640ms
cutoff: 200000          10times Time:1766ms
```



```
eswar@Eswars-MacBook-Air INFO6205 %  /usr/bin/env /Library/Java/JavaVirtualMachi
nes/temurin-17.jdk/Contents/Home/bin/java -XX:+ShowCodeDetailsInExceptionMessage
s @/var/folders/b3/ntthdw93p7fqfdgnm56gn040000gn/T/cp_3gek1g97fvotclcikw9kb7sry
.argfile edu.neu.coe.info6205.sort.par.Main
Degree of parallelism: 7
cutoff: 50000           10times Time:1995ms
cutoff: 100000          10times Time:1578ms
cutoff: 150000          10times Time:1538ms
cutoff: 200000          10times Time:1672ms
cutoff: 250000          10times Time:1675ms
cutoff: 300000          10times Time:1692ms
cutoff: 350000          10times Time:1970ms
cutoff: 400000          10times Time:1967ms
cutoff: 450000          10times Time:1956ms
cutoff: 500000          10times Time:1999ms
```

Observations:

1. Without cutoff, a 32 threads performs better with 157.6 ms time
2. 0.03 is the best cut off when number of threads are not considered with 159.06 ms time
3. The best performance is given by 32 threads with 0.07 cut off val with time of 151.6 ms