Program structures and algorithms
Spring 2023 (Section-01)

BY: PRANAV KAPOOR – NUID: 002998253

Assignment 5 (Parallel sorting)

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].

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.

Solution:

```
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.Arrays;
import java.util.HashMap;
import java.util.Map;
import java.util.Random;
import java.util.concurrent.ForkJoinPool;


public class Main {
    static Random random = new Random();
    public static void main(String[] args) {
        processArgs(args);
        

        int[] array = new int[5000000];
        int index=0;
        for(int arraySize=15000; arraySize<=35000; arraySize+=10000){
            System.out.println("Array size : " + arraySize);
            array = new int[arraySize];
            
            for(int threadCount = 5; threadCount < 50; threadCount= threadCount+5) {
                
                System.out.println("The count for the thread is ::::::: " + threadCount);
                
                
                for (int j = 50; j < 100; j+=5) {
                    
                    int cutoff = 200 * (j + 1);
                    ParSort.cutoff=cutoff;
                    
                    for (int i = 0; i < array.length; i++) array[i] = random.nextInt(10000000);
                        long time;
                    
                    time=sortArray(array);
                    array[index]=(int) time;
                    reportResults(cutoff,time);

                    index++;
                }
            }
```

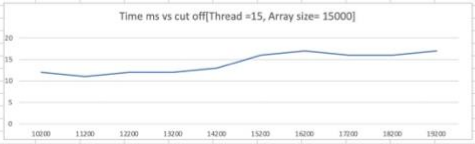
```
    }
    try {
        FileOutputStream fis = new FileOutputStream("./src/result.csv");
        OutputStreamWriter isr = new OutputStreamWriter(fis);
        BufferedWriter bw = new BufferedWriter(isr);
        int j = 0;
        for (long i : array) {
            String content = (double) 10000 * (j + 1) / 2000000 + "," + (double) i / 10 +
            j++;
            bw.write(content);
            bw.flush();
        }
        bw.close();

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

private static long sortArray(int[] array) {
    long time;
    long st = System.currentTimeMillis();
    for (int t = 0; t < 25; t++) {
        for (int i = 0; i < array.length; i++) array[i] = random.nextInt(10000000);

        ParSort.sort(array, 0, array.length);
    }
    »   long en = System.currentTimeMillis();
    »   time = (en - st);

    »   return time;
}

private static void reportResults(int cutoff, long time) {

    System.out.println("Cutoff: " + cutoff + ", Average Time: " + time + "ms");
}

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"))
            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<>();


}
```

Conclusion

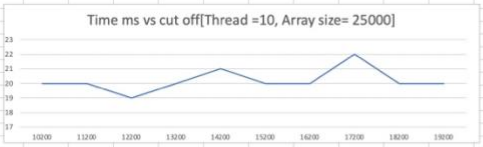
Array size 15000

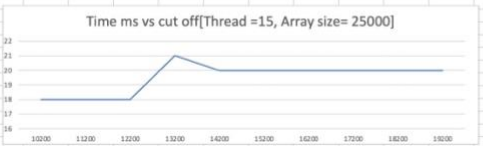
Array size= 15000		
Thread count=5		
Cut off	Time ms	
10200		63
11200		17
12200		15
13200		16
14200		22
15200		17
16200		21
17200		17
18200		23
19200		18
Thread count=10		
Cut off	Time ms	
10200		17
11200		13
12200		14
13200		17
14200		12
15200		16
16200		17
17200		17
18200		16
19200		16
Thread count=15		
Cut off	Time ms	
10200		12
11200		11
12200		12
13200		12
14200		13
15200		16
16200		17
17200		16
18200		16
19200		17



Array size 25000

Array size= 25000		
Thread count=5		
Cut off	Time ms	
10200		21
11200		19
12200		19
13200		20
14200		20
15200		20
16200		20
17200		21
18200		20
19200		20
Thread count=10		
Cut off	Time ms	
10200		20
11200		20
12200		19
13200		20
14200		21
15200		20
16200		20
17200		22
18200		20
19200		20
Thread count=15		
Cut off	Time ms	
10200		18
11200		18
12200		18
13200		21
14200		20
15200		20
16200		20
17200		20
18200		20
19200		20
Thread count=20		

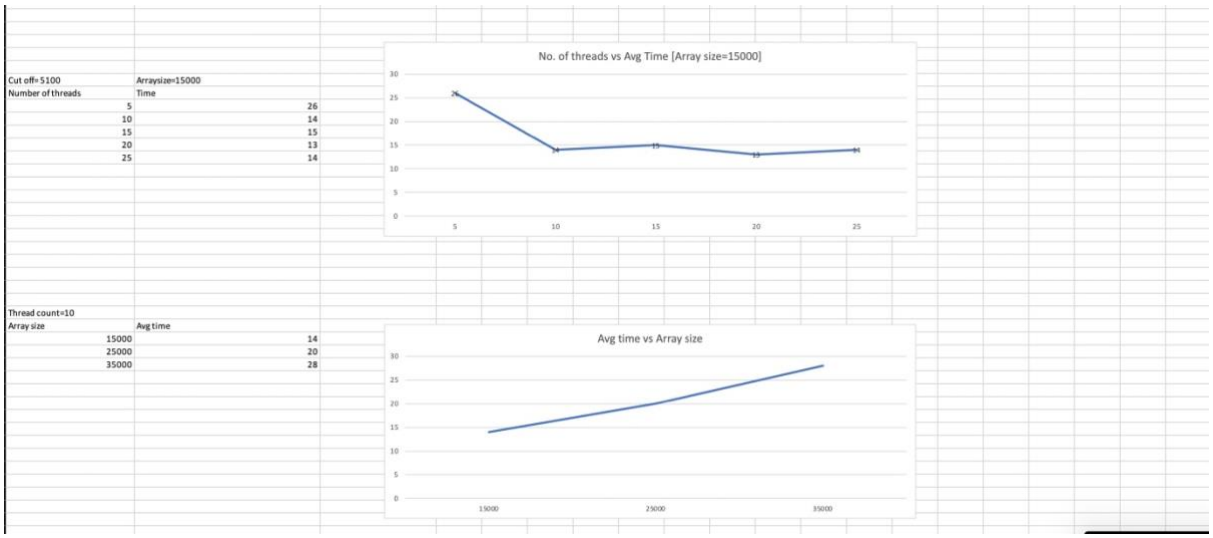




Array size 35000

Array size= 35000	
Thread count=5	
Cut off	Time ms
10200	26
11200	26
12200	26
13200	26
14200	26
15200	26
16200	26
17200	26
18200	29
19200	29
Thread count=10	
Cut off	Time ms
10200	33
11200	35
12200	27
13200	26
14200	26
15200	26
16200	26
17200	26
18200	29
19200	28
Thread count=15	
Cut off	Time ms
10200	26
11200	26
12200	26
13200	27
14200	26
15200	26
16200	25
17200	26
18200	28
19200	28
Thread count=20	

1. Graph for no of threads vs Average time when array size is 15000.
2. Graph for no of array size vs Average time when thread count is 10.



Cut off= 5100	Arraysize=15000
Number of threads	Time
5	26
10	14
15	15
20	13
25	14
Thread count=10	
Array size	Avg time
15000	14
25000	20
35000	28

Conclusion:

From the above visualisations, we can see that:

1. As the thread count increase, the time starts to decrease for each cutoff value.
2. As the array size starts to increase, the time required to run also increases.