Assignment 5 (Parallel Sorting)

Program Structures and Algorithms
Spring 2023(SEC –03)

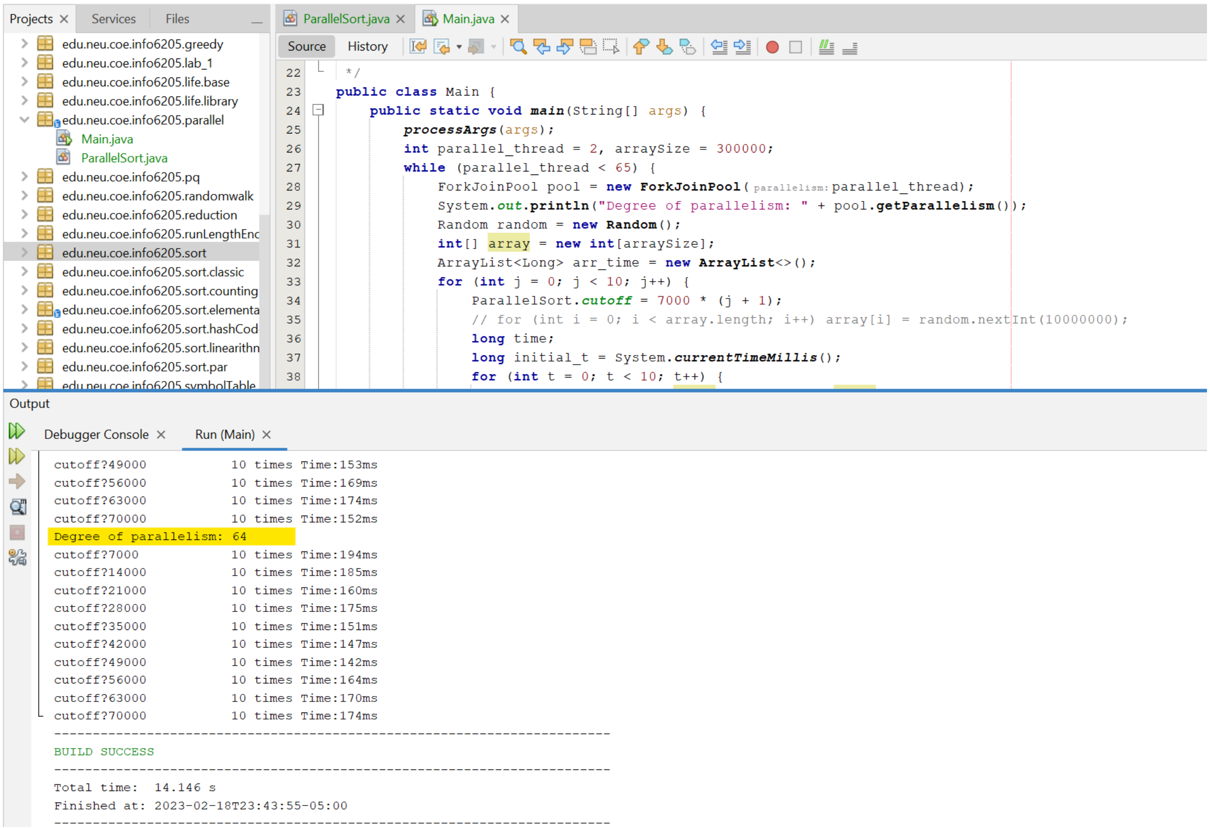
NAME:ALTAF SALIM SHAIKH
NUID:002774748

Task:

The task at hand entails the implementation of a parallel sorting algorithm that ensures the sorting of each partition in parallel. Two distinct strategies shall be evaluated for determining the suitability of parallel sorting. The first scheme involves establishing a cutoff value, which shall be updated based on the initial argument specified during program execution. The responsibility of identifying an appropriate value for this cutoff lies with the implementer, who should conduct sufficient experimentation to determine its ideal value. If the number of elements to be sorted falls below the determined cutoff value, the system sort shall be employed. The second approach involves computing the recursion depth or the available number of threads. Based on this computation, the ideal number of separate threads (which must be a power of 2) shall be determined. Thereafter, the number of partitions that should be parallelized will be arranged, with recursion ceasing after the depth of lg t is attained.

Reports.

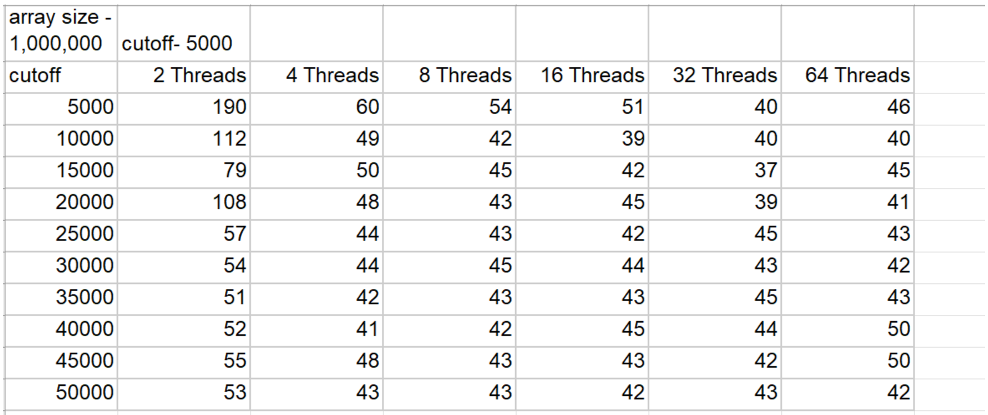
Console Output


Projects
Services
Files
edu.neu.coe.info6205.greedy
edu.neu.coe.info6205.lab_1
edu.neu.coe.info6205.life.base
edu.neu.coe.info6205.life.library
edu.neu.coe.info6205.parallel
Main.java
ParallelSort.java
edu.neu.coe.info6205.pq
edu.neu.coe.info6205.randomwalk
edu.neu.coe.info6205.reduction
edu.neu.coe.info6205.runLengthEnc
edu.neu.coe.info6205.sort
edu.neu.coe.info6205.sort.classic
edu.neu.coe.info6205.sort.counting
edu.neu.coe.info6205.sort.elementa
edu.neu.coe.info6205.sort.hashCod
edu.neu.coe.info6205.sort.linearithn
edu.neu.coe.info6205.sort.par
edu.neu.coe.info6205.symbolTable
ParallelSort.java
Main.java
Source
History
Output
Debugger Console
Run (Main)

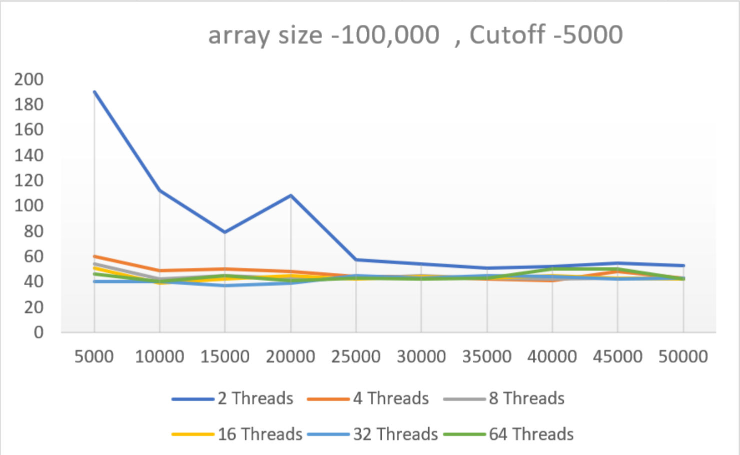
```java
 */
public class Main {
    public static void main(String[] args) {
        processArgs(args);
        int parallel_thread = 2, arraySize = 300000;
        while (parallel_thread < 65) {
            ForkJoinPool pool = new ForkJoinPool( parallelism: parallel_thread);
            System.out.println("Degree of parallelism: " + pool.getParallelism());
            Random random = new Random();
            int[] array = new int[arraySize];
            ArrayList<Long> arr_time = new ArrayList<>();
            for (int j = 0; j < 10; j++) {
                ParallelSort.cutoff = 7000 * (j + 1);
                // for (int i = 0; i < array.length; i++) array[i] = random.nextInt(10000000);
                long time;
                long initial_t = System.currentTimeMillis();
                for (int t = 0; t < 10; t++) {
```

```
cutoff?49000            10 times Time:153ms
cutoff?56000            10 times Time:169ms
cutoff?63000            10 times Time:174ms
cutoff?70000            10 times Time:152ms
Degree of parallelism: 64
cutoff?7000             10 times Time:194ms
cutoff?14000            10 times Time:185ms
cutoff?21000            10 times Time:160ms
cutoff?28000            10 times Time:175ms
cutoff?35000            10 times Time:151ms
cutoff?42000            10 times Time:147ms
cutoff?49000            10 times Time:142ms
cutoff?56000            10 times Time:164ms
cutoff?63000            10 times Time:170ms
cutoff?70000            10 times Time:174ms
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time:  14.146 s
Finished at: 2023-02-18T23:43:55-05:00
------------------------------------------------------------------------
```

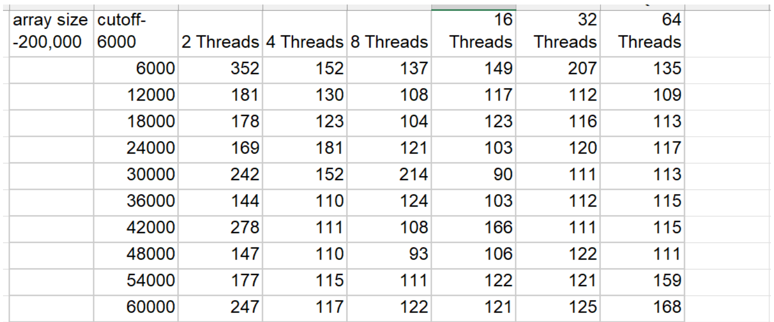
1.Array Size -100,000 cutoff-5000



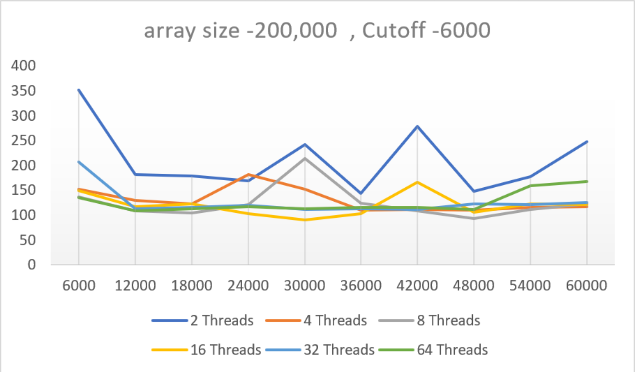
array size - 1,000,000	cutoff- 5000					
cutoff	2 Threads	4 Threads	8 Threads	16 Threads	32 Threads	64 Threads
5000	190	60	54	51	40	46
10000	112	49	42	39	40	40
15000	79	50	45	42	37	45
20000	108	48	43	45	39	41
25000	57	44	43	42	45	43
30000	54	44	45	44	43	42
35000	51	42	43	43	45	43
40000	52	41	42	45	44	50
45000	55	48	43	43	42	50
50000	53	43	43	42	43	42



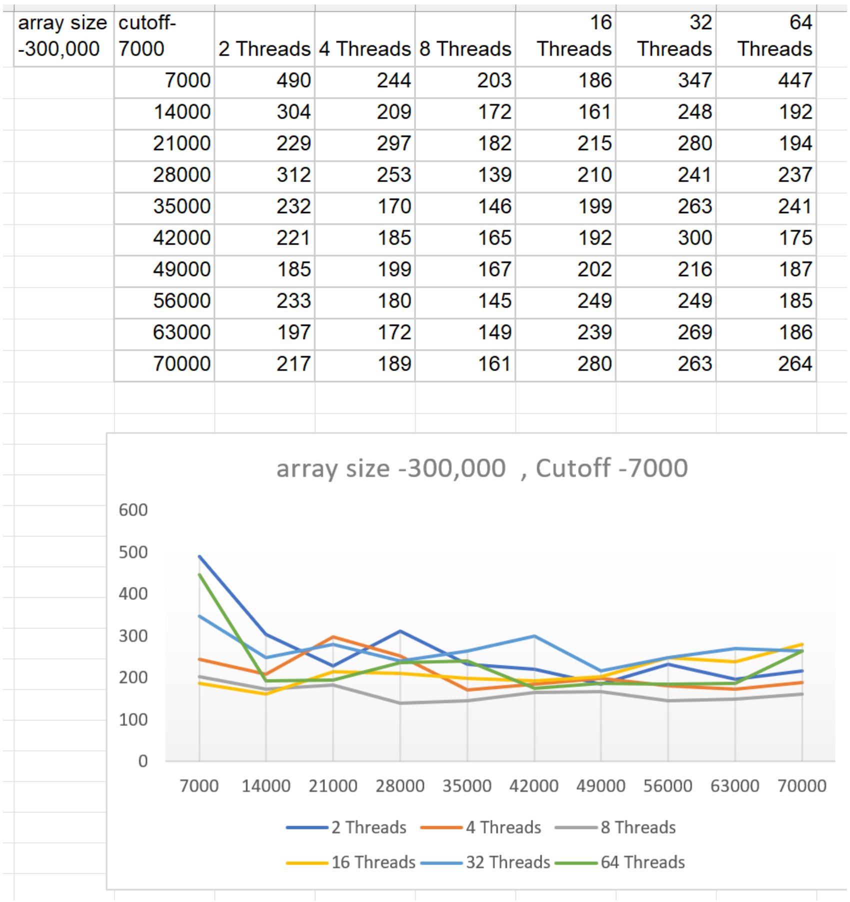
2.Array Size -200,000 cutoff-6000



array size -200,000	cutoff- 6000	2 Threads	4 Threads	8 Threads	16 Threads	32 Threads	64 Threads
	6000	352	152	137	149	207	135
	12000	181	130	108	117	112	109
	18000	178	123	104	123	116	113
	24000	169	181	121	103	120	117
	30000	242	152	214	90	111	113
	36000	144	110	124	103	112	115
	42000	278	111	108	166	111	115
	48000	147	110	93	106	122	111
	54000	177	115	111	122	121	159
	60000	247	117	122	121	125	168



3.Array Size -300,000 cutoff-7000



array size -300,000	cutoff-7000	2 Threads	4 Threads	8 Threads	16 Threads	32 Threads	64 Threads
	7000	490	244	203	186	347	447
	14000	304	209	172	161	248	192
	21000	229	297	182	215	280	194
	28000	312	253	139	210	241	237
	35000	232	170	146	199	263	241
	42000	221	185	165	192	300	175
	49000	185	199	167	202	216	187
	56000	233	180	145	249	249	185
	63000	197	172	149	239	269	186
	70000	217	189	161	280	263	264

Relationship Conclusion:

The performance of the parallel sorting algorithm is significantly influenced by the cutoff value and the number of available threads.
Using more threads generally leads to better performance, up to a certain point.
The optimal number of threads appears to be between 8 and 16 threads for most cutoff values.
The best cutoff value for achieving the best performance varies depending on the number of available threads. However, based on the presented data, it appears that a cutoff value of around 12,000 leads to the best overall performance for most thread configurations.
It is important to experiment and find the ideal combination of these factors to achieve optimal performance.
Overall, the three datasets consistently suggest that the parallel sorting algorithm's performance can be optimized through careful consideration of the cutoff value and the number of threads used.