Program Structures and Algorithms
Fall 2024

NAME: Sanskruti Manoria
NUID: 002643300
GITHUB LINK: https://github.com/sannskruti/INFO6205

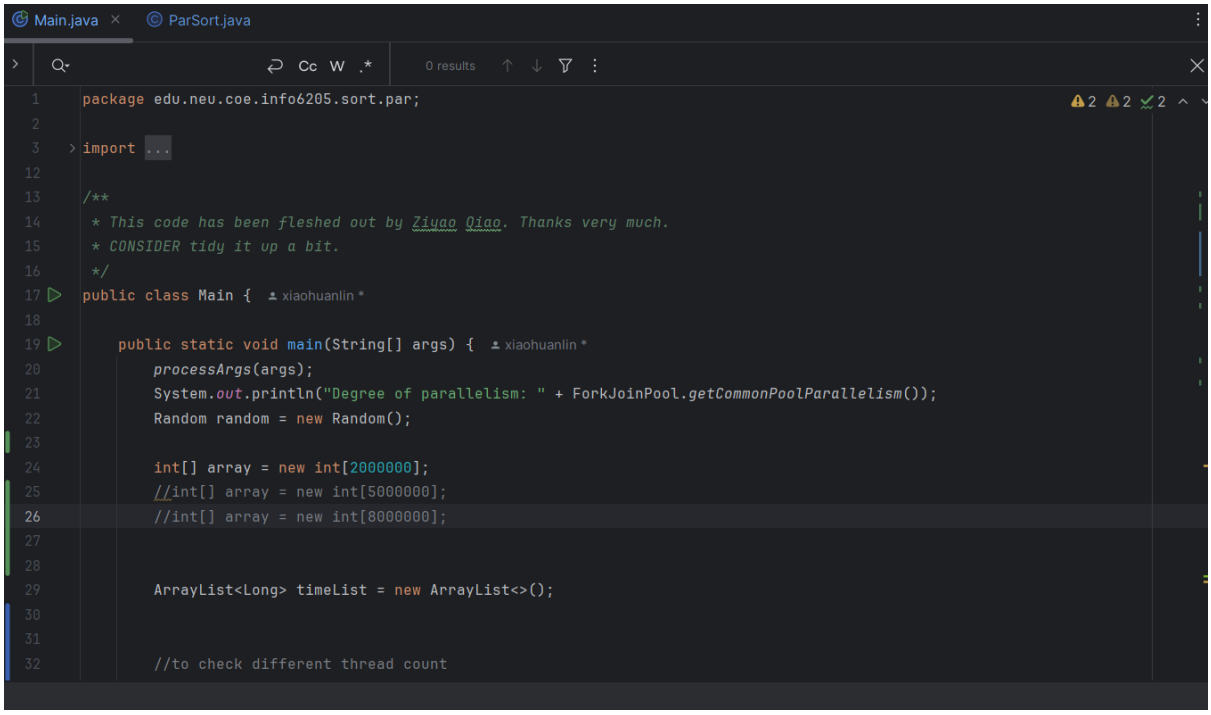
Assignment 5 Parallel sort

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.

Code:

Main.java



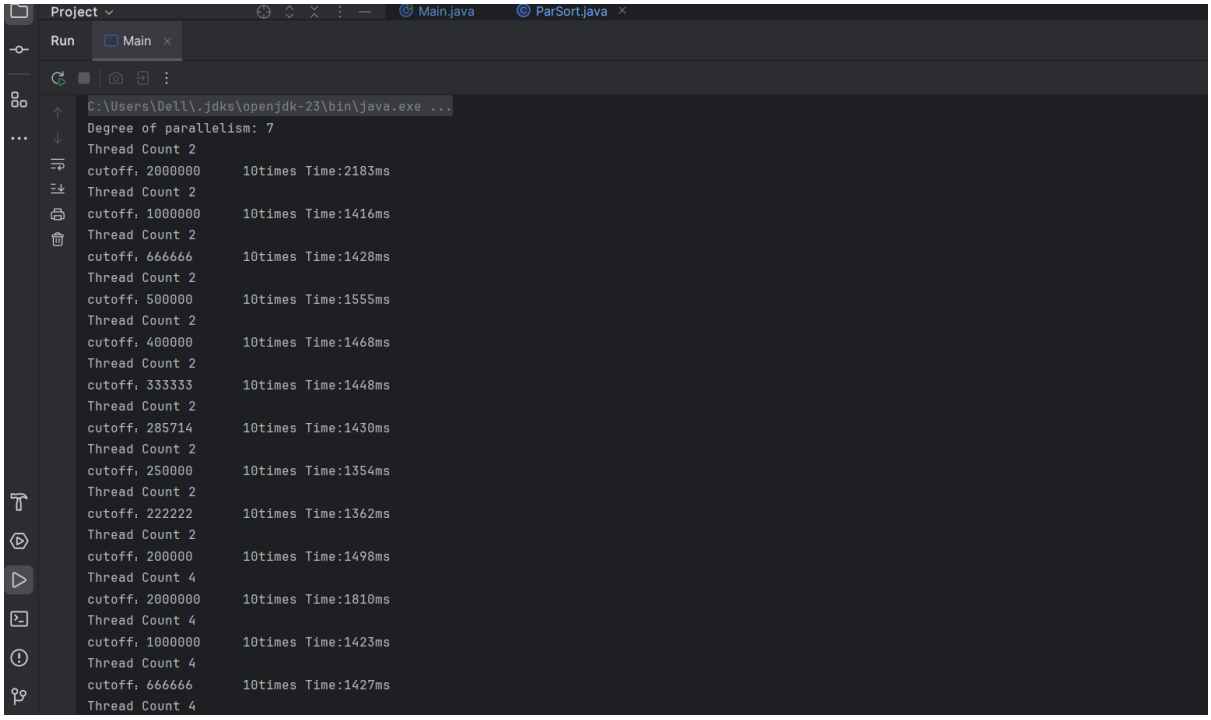
Main.java × ParSort.java

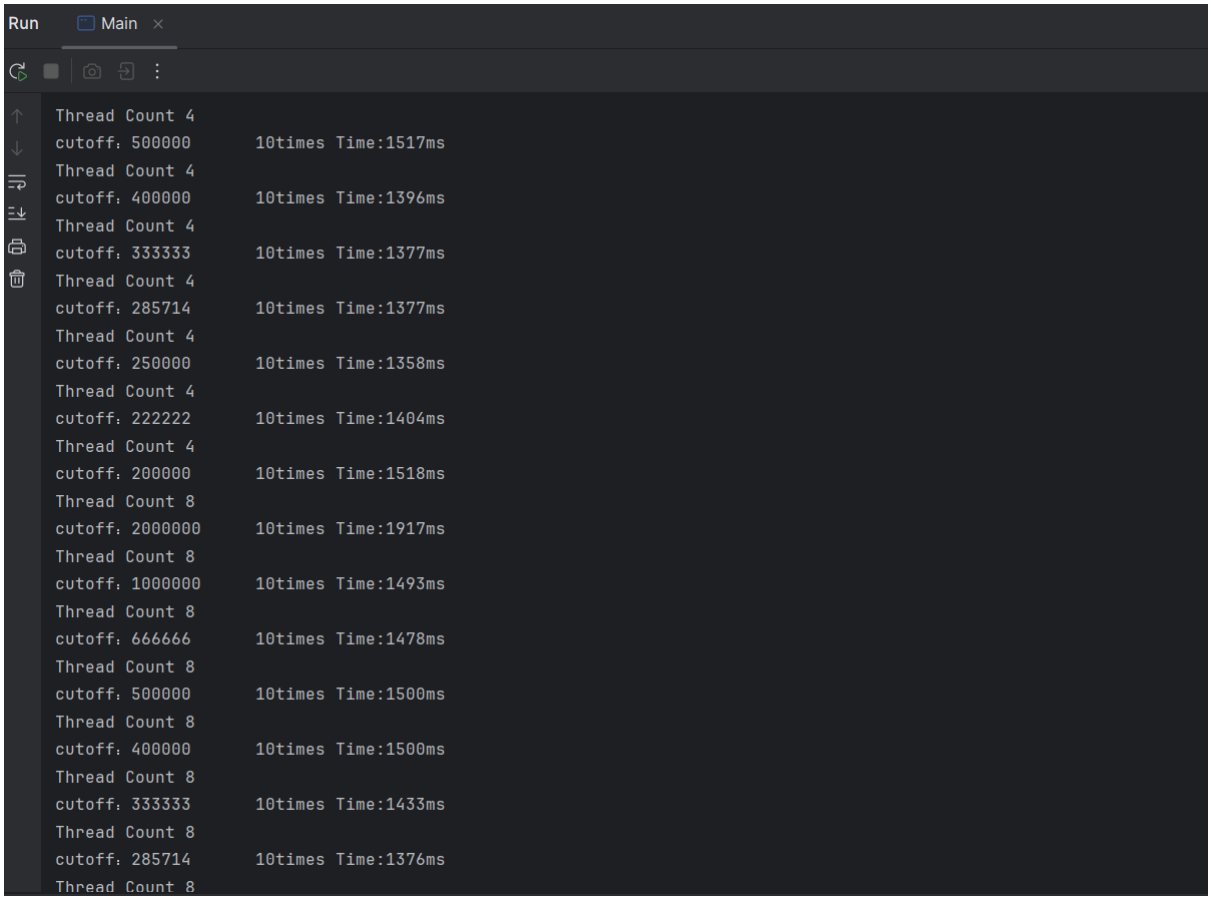
```
public class Main {  xiaohuanlin *                                                    ⚠2 ⚠2
    public static void main(String[] args) {  xiaohuanlin *
        //to check different thread count
        for (int n=2;n<=16;n=n*2){
            for (int j = 1; j <= 10; j++) {

//            for (int j = 50; j < 100; j++) {
              ParSort.cutoff = array.length/j;
              //ParSort.cutoff = 20000 * (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( bound: 10000000);
                  ParSort.sort(array,  from: 0, array.length);
              }
              long endTime = System.currentTimeMillis();
              time = (endTime - startTime);
              timeList.add(time);


              System.out.println("Thread Count " + n);
              System.out.println("cutoff: " + (ParSort.cutoff) + "\t\t10times Time:" + time + "ms");

          }
      }
          try {
```

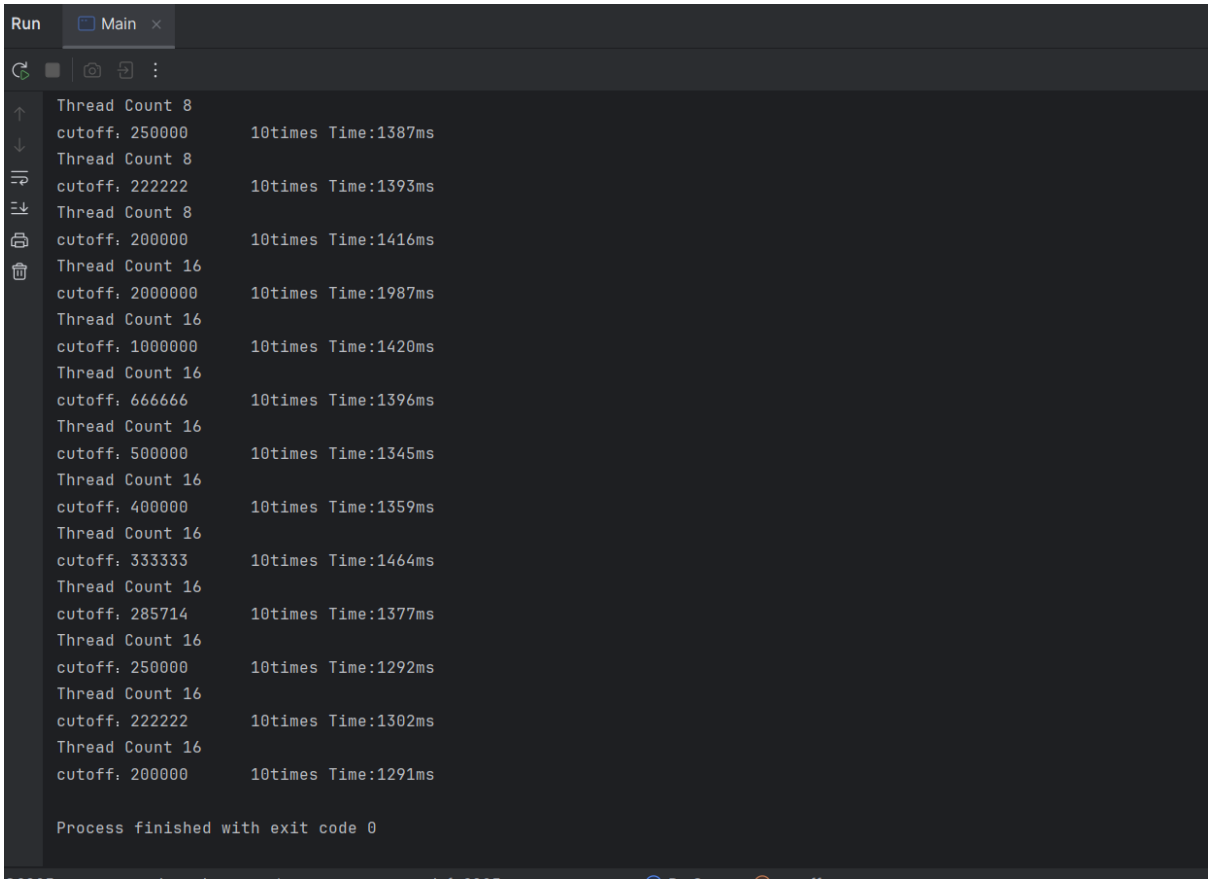
parSort.java

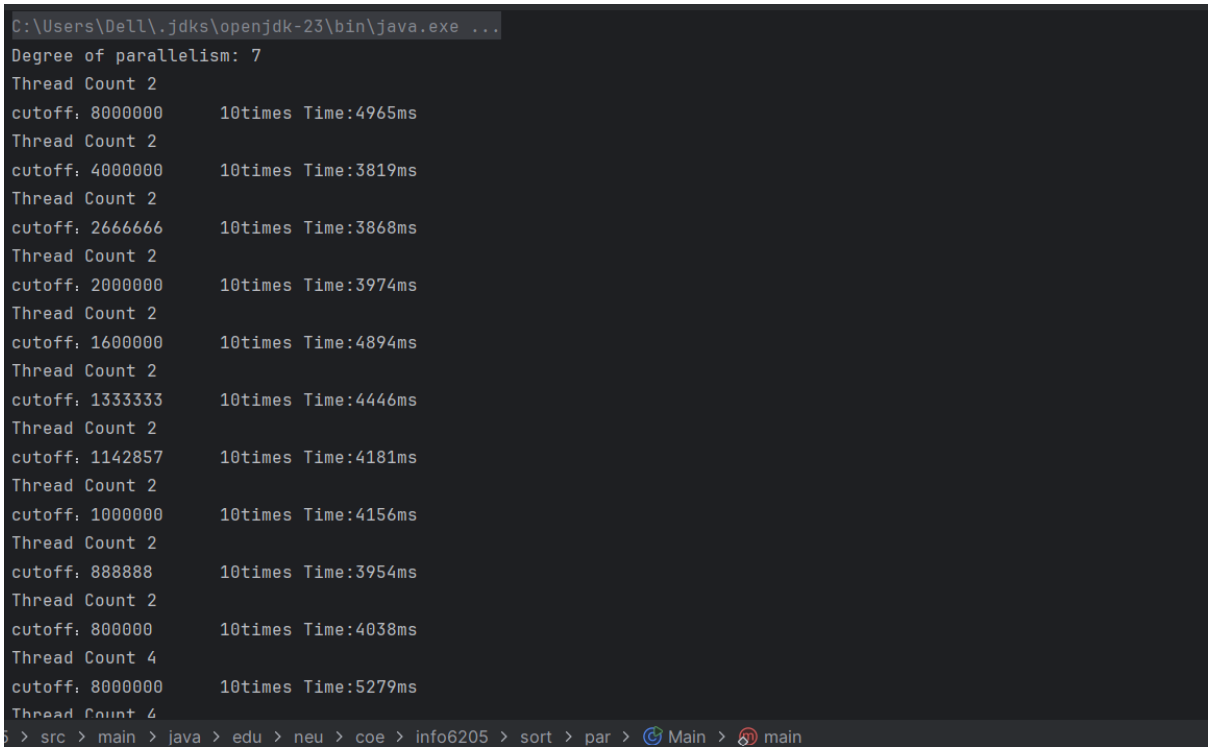
```
/**
 * This code has been fleshed out by Ziyao Qiao. Thanks very much.
 * CONSIDER tidy it up a bit.
 */
class ParSort {  3 usages  xiaohuanlin *

    public static int cutoff = 5000;
    //public static int cutoff = 1000;
    //public static int cutoff = 2000;
    //public static int cutoff = 500;

    public static void sort(int[] array, int from, int to) {  xiaohuanlin
        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,  to: from + (to - from) / 2); // TO IMPLEMENT
            CompletableFuture<int[]> parsort2 = parsort(array,  from: 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]) {
```

Output:


Project
Main.java
ParSort.java
Run
Main
C:\Users\Dell\.jdks\openjdk-23\bin\java.exe ...
Degree of parallelism: 7
Thread Count 2
cutoff: 2000000 10times Time:2183ms
Thread Count 2
cutoff: 1000000 10times Time:1416ms
Thread Count 2
cutoff: 666666 10times Time:1428ms
Thread Count 2
cutoff: 500000 10times Time:1555ms
Thread Count 2
cutoff: 400000 10times Time:1468ms
Thread Count 2
cutoff: 333333 10times Time:1448ms
Thread Count 2
cutoff: 285714 10times Time:1430ms
Thread Count 2
cutoff: 250000 10times Time:1354ms
Thread Count 2
cutoff: 222222 10times Time:1362ms
Thread Count 2
cutoff: 200000 10times Time:1498ms
Thread Count 4
cutoff: 2000000 10times Time:1810ms
Thread Count 4
cutoff: 1000000 10times Time:1423ms
Thread Count 4
cutoff: 666666 10times Time:1427ms
Thread Count 4


Run
Main
Thread Count 4
cutoff: 500000 10times Time:1517ms
Thread Count 4
cutoff: 400000 10times Time:1396ms
Thread Count 4
cutoff: 333333 10times Time:1377ms
Thread Count 4
cutoff: 285714 10times Time:1377ms
Thread Count 4
cutoff: 250000 10times Time:1358ms
Thread Count 4
cutoff: 222222 10times Time:1404ms
Thread Count 4
cutoff: 200000 10times Time:1518ms
Thread Count 8
cutoff: 2000000 10times Time:1917ms
Thread Count 8
cutoff: 1000000 10times Time:1493ms
Thread Count 8
cutoff: 666666 10times Time:1478ms
Thread Count 8
cutoff: 500000 10times Time:1500ms
Thread Count 8
cutoff: 400000 10times Time:1500ms
Thread Count 8
cutoff: 333333 10times Time:1433ms
Thread Count 8
cutoff: 285714 10times Time:1376ms
Thread Count 8

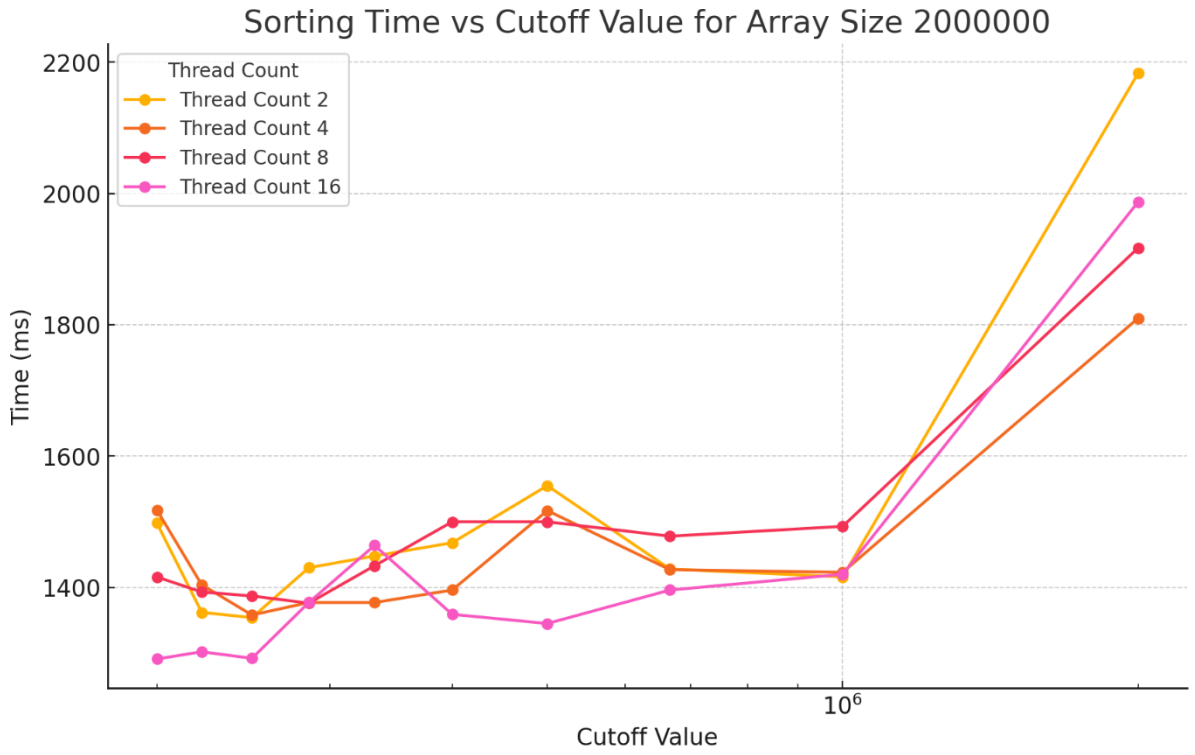

Run
Main
Thread Count 8
cutoff: 250000 10times Time:1387ms
Thread Count 8
cutoff: 222222 10times Time:1393ms
Thread Count 8
cutoff: 200000 10times Time:1416ms
Thread Count 16
cutoff: 2000000 10times Time:1987ms
Thread Count 16
cutoff: 1000000 10times Time:1420ms
Thread Count 16
cutoff: 666666 10times Time:1396ms
Thread Count 16
cutoff: 500000 10times Time:1345ms
Thread Count 16
cutoff: 400000 10times Time:1359ms
Thread Count 16
cutoff: 333333 10times Time:1464ms
Thread Count 16
cutoff: 285714 10times Time:1377ms
Thread Count 16
cutoff: 250000 10times Time:1292ms
Thread Count 16
cutoff: 222222 10times Time:1302ms
Thread Count 16
cutoff: 200000 10times Time:1291ms
Process finished with exit code 0

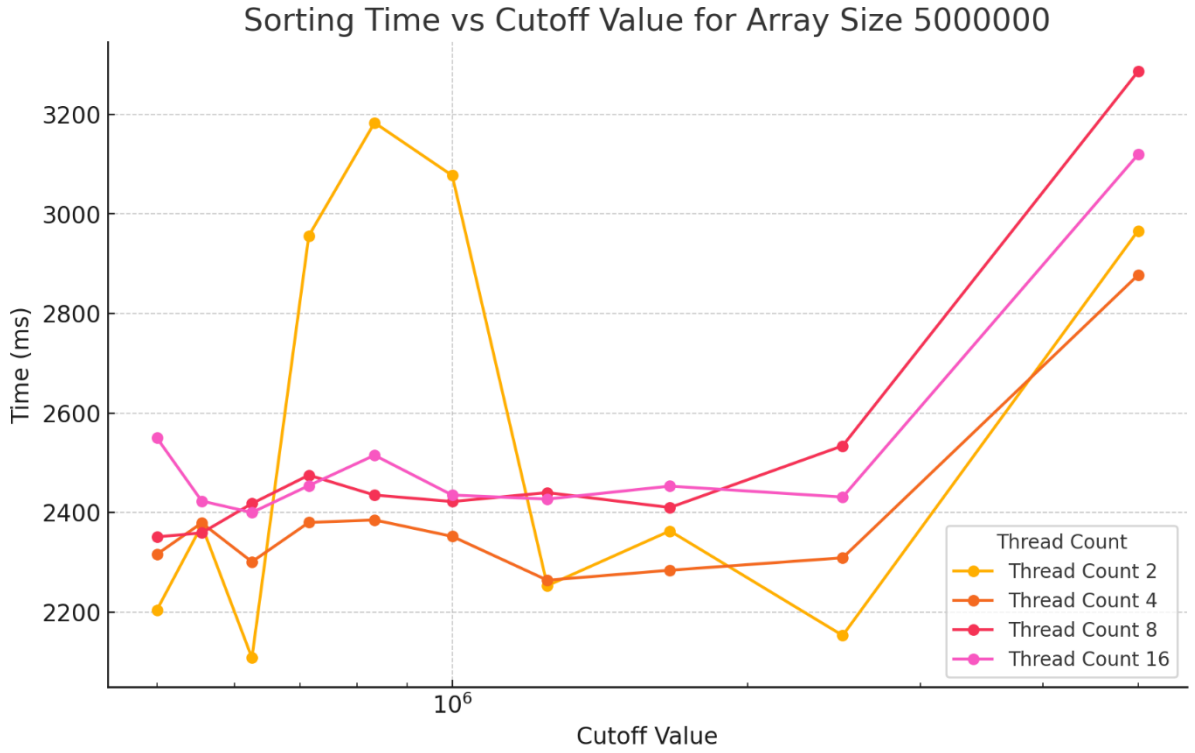


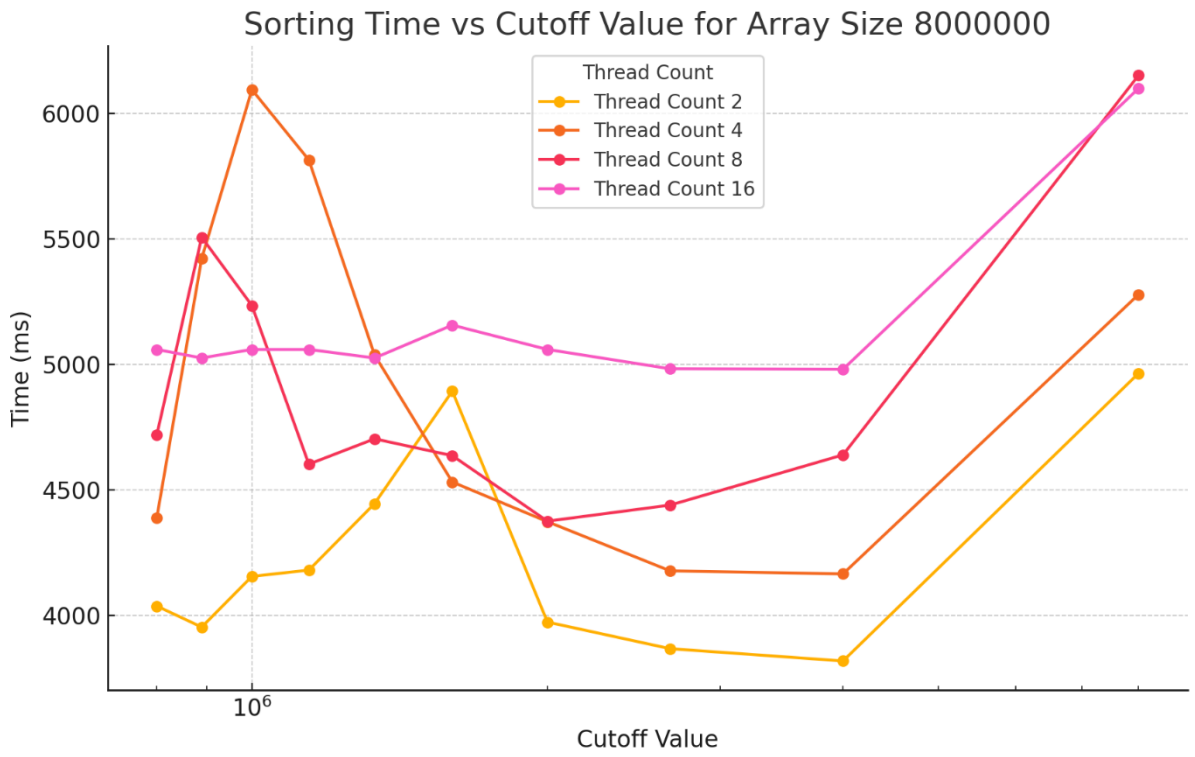
Conclusion: Experimented with cutoff values, thread count and different array sizes given in excel to perform experiment and observe results and created graphs for same mentioned below using python-

parallel_sort_data_points.xlsx

Here are the graphs showing sorting time versus cutoff values for different array sizes (2,000,000, 5,000,000, and 8,000,000). Each line represents a different thread count, allowing for a clear comparison of performance across cutoff values for each array size. The logarithmic scale on the x-axis provides better visibility of cutoff variations.


Sorting Time vs Cutoff Value for Array Size 2000000
Thread Count
Thread Count 2
Thread Count 4
Thread Count 8
Thread Count 16
Time (ms)
2200
2000
1800
1600
1400
10^6
Cutoff Value


Sorting Time vs Cutoff Value for Array Size 5000000
Time (ms)
3200
3000
2800
2600
2400
2200
10^6
Cutoff Value
Thread Count
Thread Count 2
Thread Count 4
Thread Count 8
Thread Count 16



Optimal Cutoff: Cutoff values between 250,000 and 500,000 yield the best balance between task partitioning and overhead.

Thread Count Efficiency: 4 to 8 threads offer the best performance across all array sizes, with diminishing returns beyond 8 threads.

Scalability: As array size increases, parallel sorting provides more significant performance benefits.

Solid Conclusion
Parallelizing sort with optimal cutoff values and thread counts proves highly effective for large arrays, significantly reducing sorting time. The best results are achieved with 4–8 threads and a cutoff of 250,000–500,000, making this method ideal for high-performance sorting of large datasets.