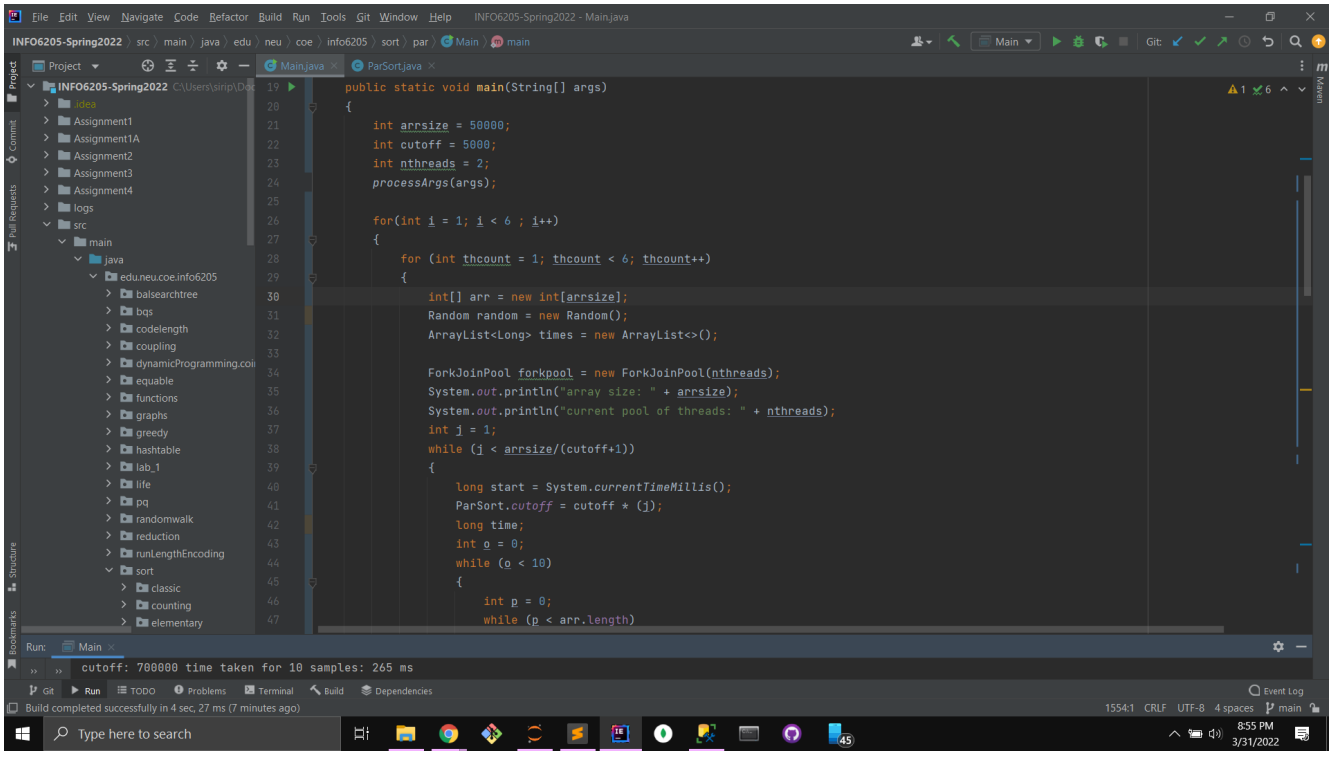
Program Structures & Algorithms
Spring 2022
Assignment No. 4

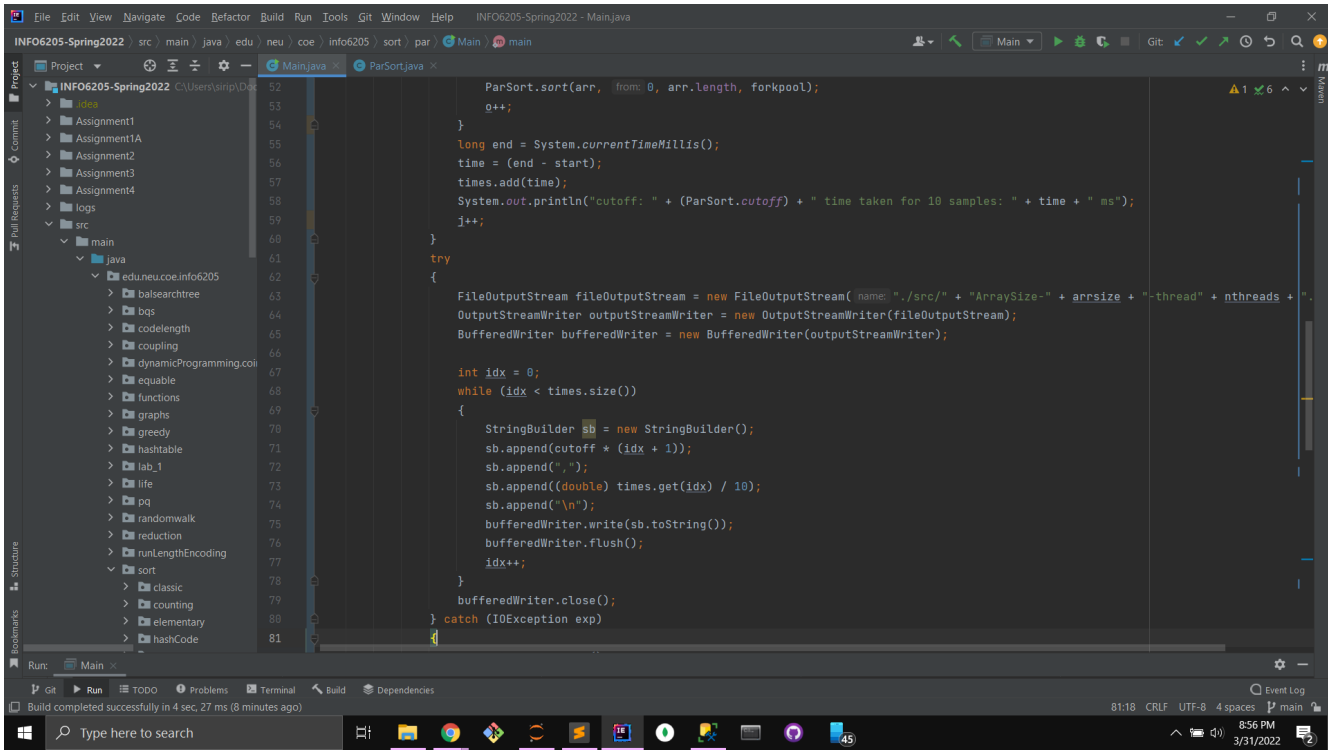
Name: Shashank Siripragada
NUID: 002193773

Task:

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.

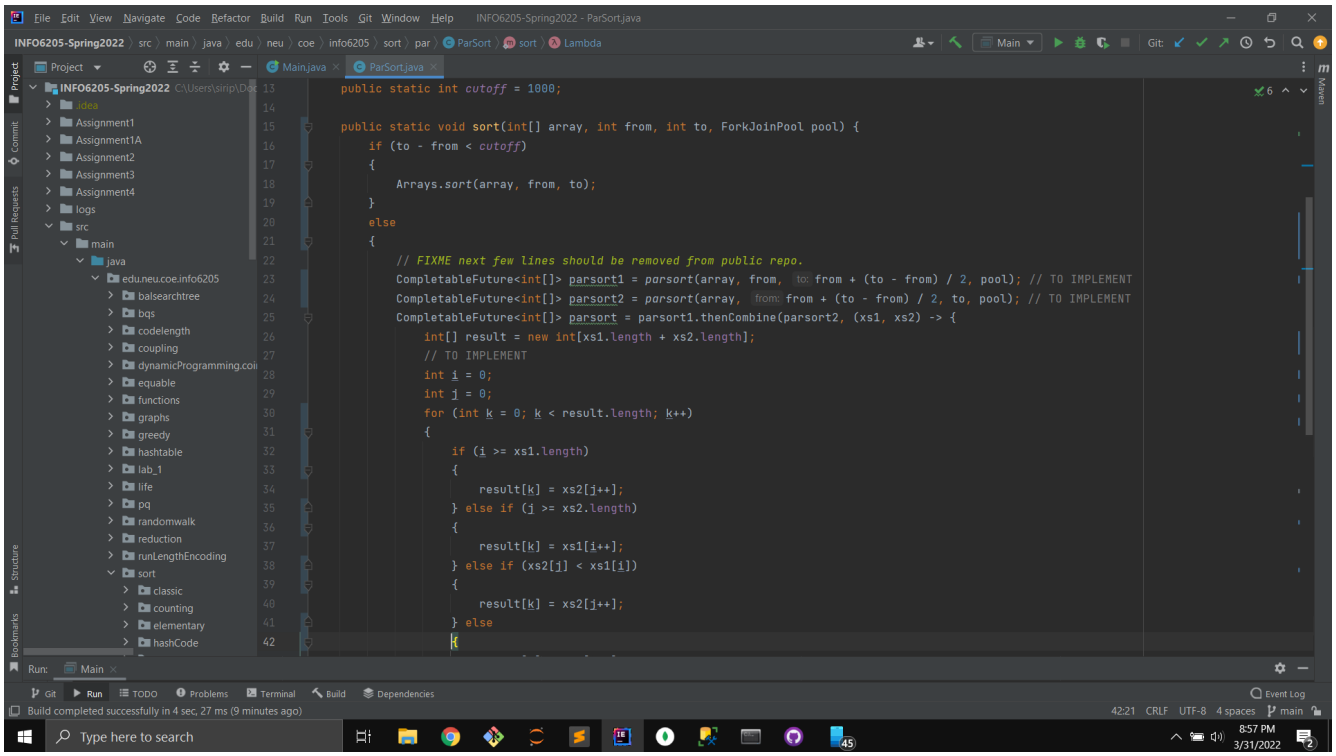
Code Changes: Main.java



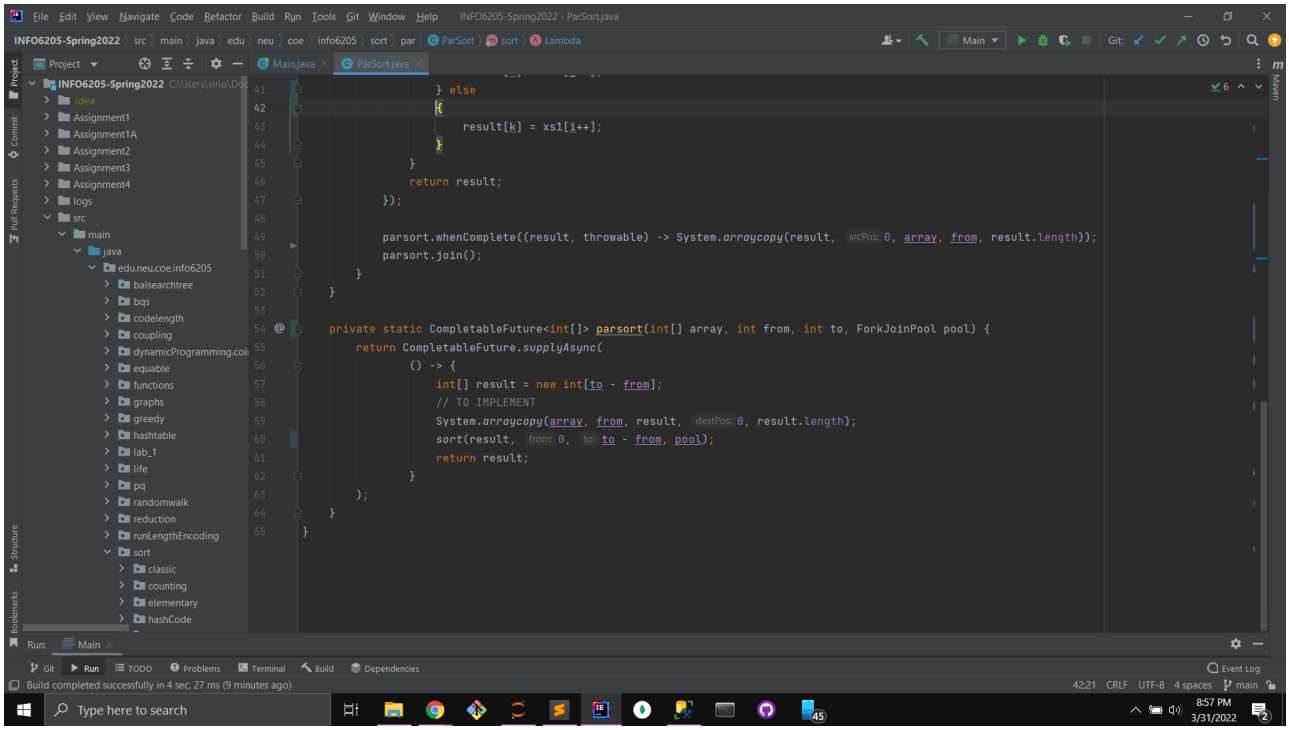


```
                    ParSort.sort(arr, from: 0, arr.length, forkpool);
                    o++;
                }
                long end = System.currentTimeMillis();
                time = (end - start);
                times.add(time);
                System.out.println("cutoff: " + (ParSort.cutoff) + " time taken for 10 samples: " + time + " ms");
                j++;
            }
            try
            {
                FileOutputStream fileOutputStream = new FileOutputStream( name: "./src/" + "ArraySize-" + arrsize + "-thread" + nthreads + ".
                OutputStreamWriter outputStreamWriter = new OutputStreamWriter(fileOutputStream);
                BufferedWriter bufferedWriter = new BufferedWriter(outputStreamWriter);

                int idx = 0;
                while (idx < times.size())
                {
                    StringBuilder sb = new StringBuilder();
                    sb.append(cutoff * (idx + 1));
                    sb.append(",");
                    sb.append((double) times.get(idx) / 10);
                    sb.append("\n");
                    bufferedWriter.write(sb.toString());
                    bufferedWriter.flush();
                    idx++;
                }
                bufferedWriter.close();
            } catch (IOException exp)
            {
```

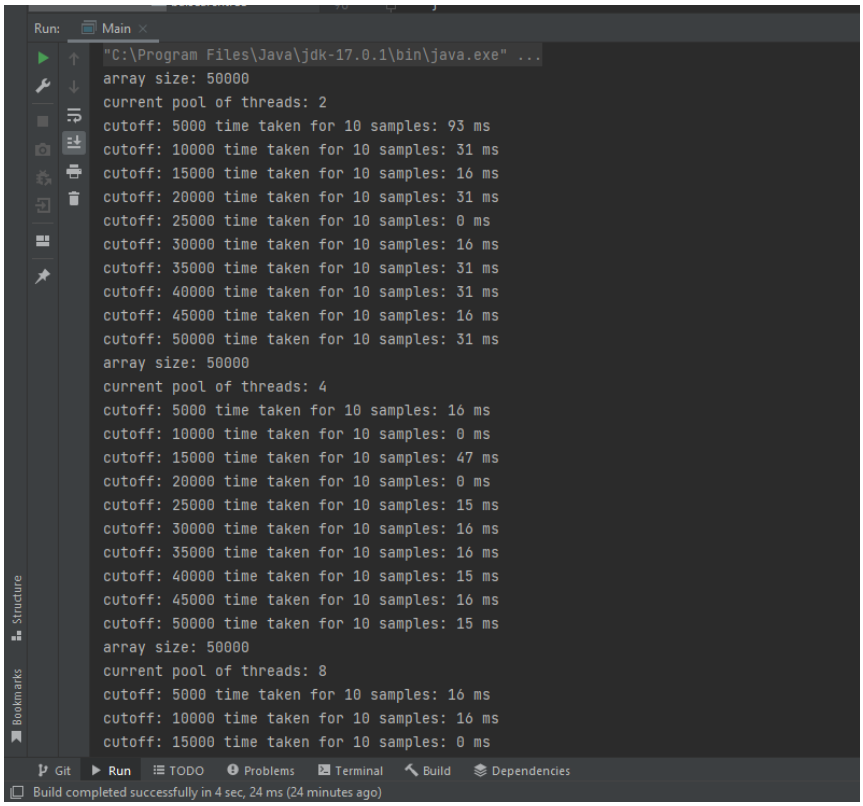
ParSort.java:



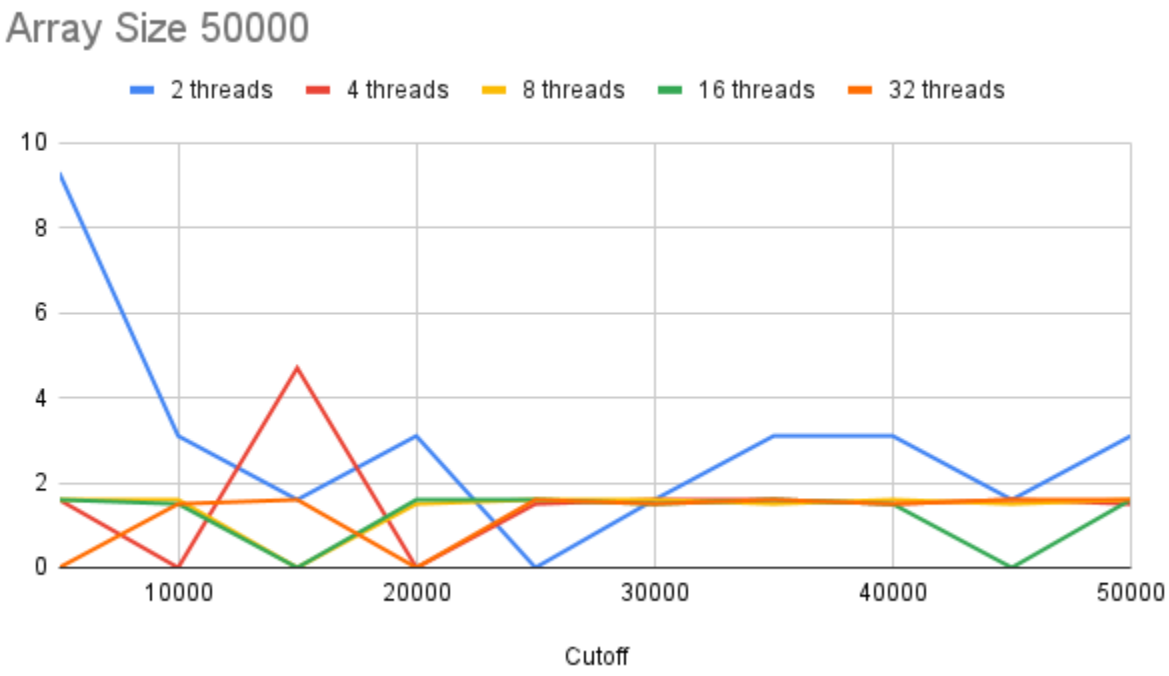
```
public static int cutoff = 1000;

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

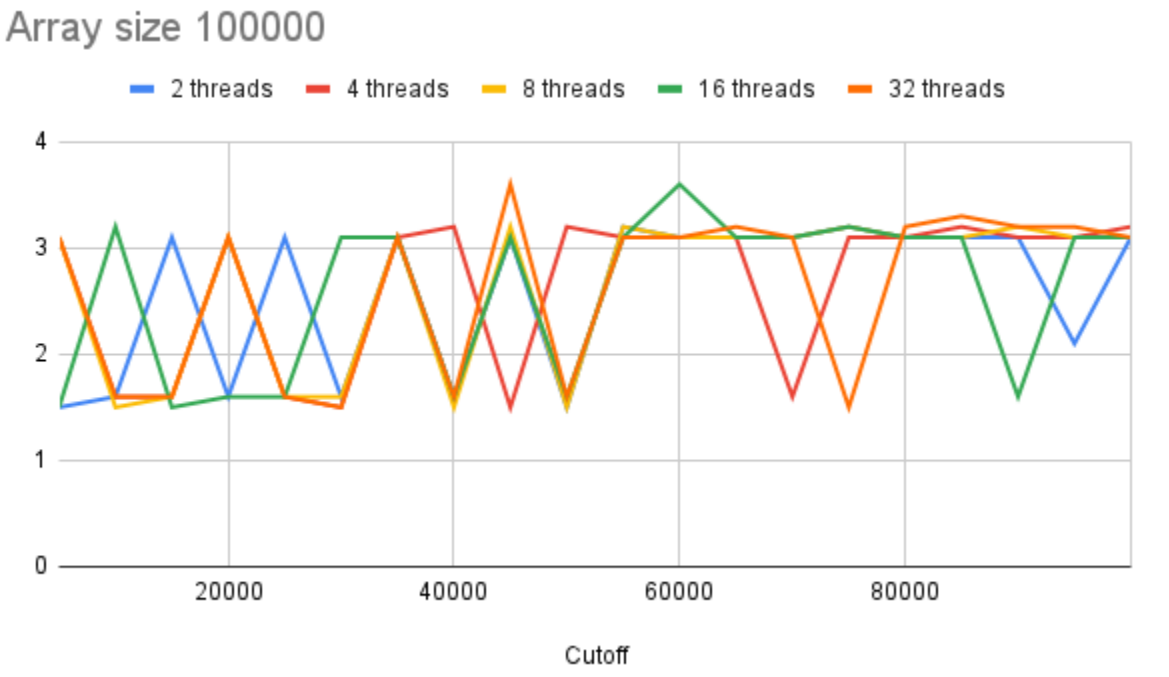


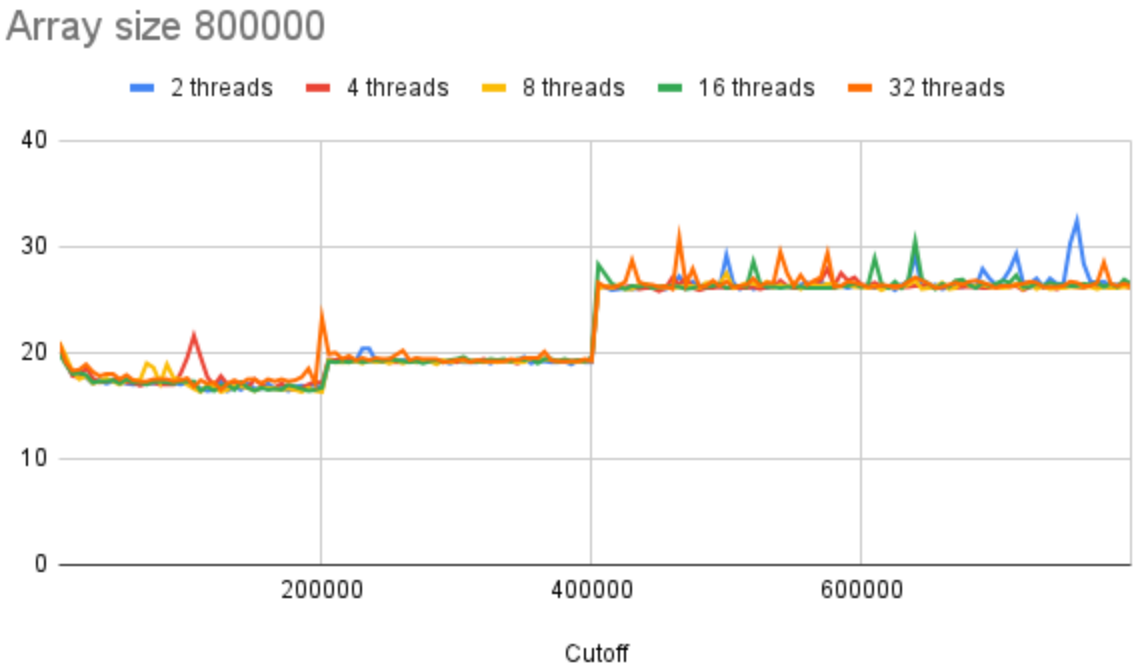
Console Output and CSV files: The csvs containing the observations are added to the src folder.



Plots:


Array Size 50000
2 threads
4 threads
8 threads
16 threads
32 threads
10
8
6
4
2
0
10000
20000
30000
40000
50000
Cutoff


Array size 100000
2 threads
4 threads
8 threads
16 threads
32 threads
4
3
2
1
0
20000
40000
60000
80000
Cutoff



Observations:

- The plots are generated from the csv files containing different values of the cutoffs and threads.
- It can be concluded that 4 will be the optimal number of threads as there is no change in the performance as we increase the threads.
- The lowest performance is when the cutoff is ¼ size of the array.