Program Structures and Algorithms Spring 2023(SEC –1) Assignment-5

Name: Naga Venkata Nishanth Sayana

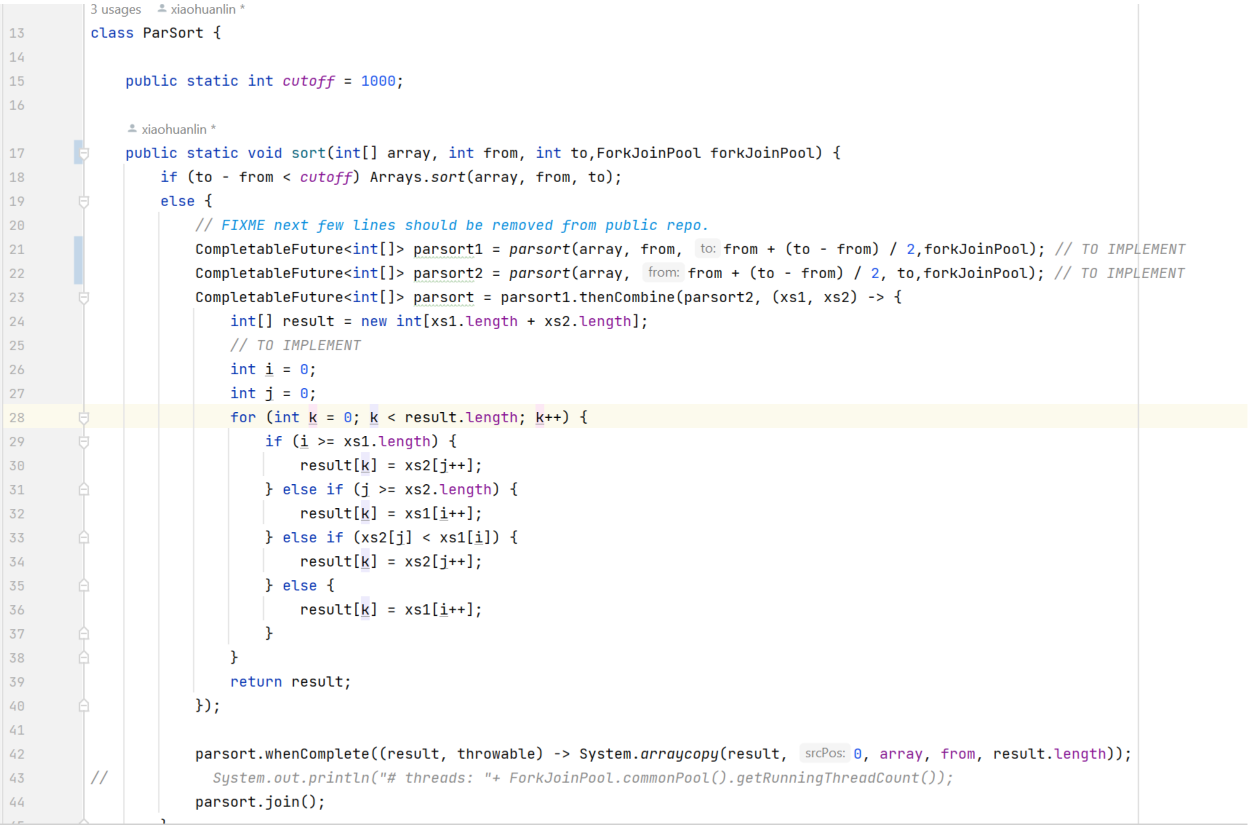
NU ID: 002930970

Task:

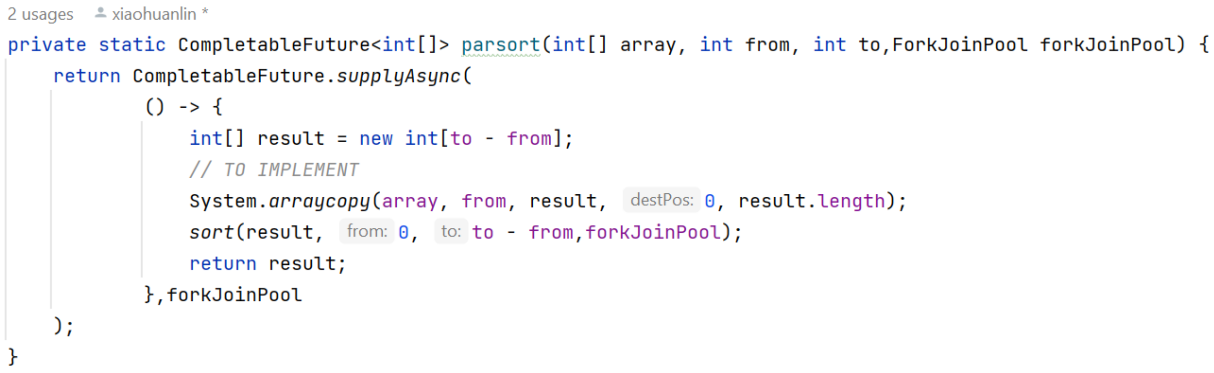
Our Task here is perform parallel sort and switch between parallel sort and system sort based on the cut-off values. We need to make a note of times taken while providing different number of threads for performing parallel sort.

Code Screen Shots:

ParaSort:

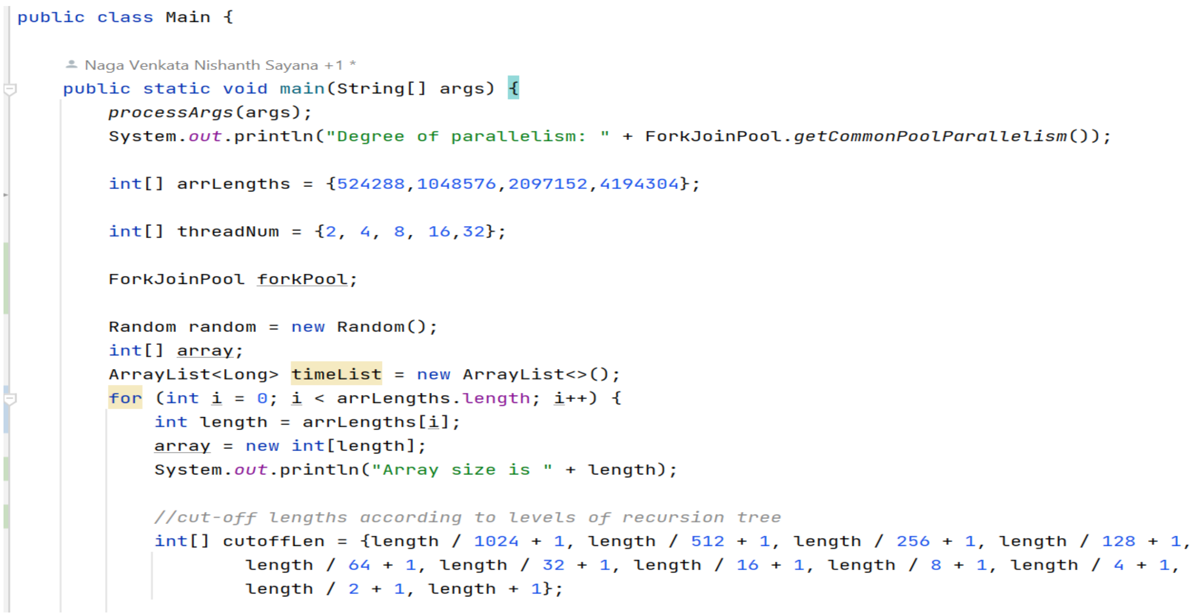


```java
3 usages  xiaohuanlin *
class ParSort {

    public static int cutoff = 1000;

    xiaohuanlin *
    public static void sort(int[] array, int from, int to,ForkJoinPool forkJoinPool) {
        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,forkJoinPool); // TO IMPLEMENT
            CompletableFuture<int[]> parsort2 = parsort(array,  from: from + (to - from) / 2, to,forkJoinPool); // 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,  srcPos: 0, array, from, result.length));
//              System.out.println("# threads: "+ ForkJoinPool.commonPool().getRunningThreadCount());
            parsort.join();
```

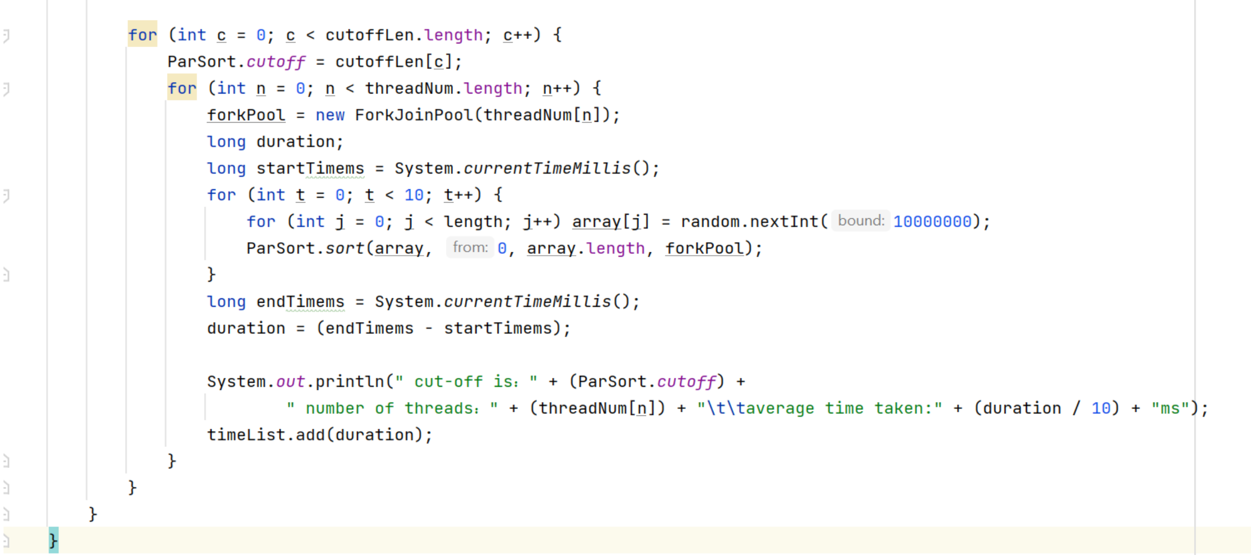


```
2 usages   xiaohuanlin *
private static CompletableFuture<int[]> parsort(int[] array, int from, int to,ForkJoinPool forkJoinPool) {
    return CompletableFuture.supplyAsync(
            () -> {
                int[] result = new int[to - from];
                // TO IMPLEMENT
                System.arraycopy(array, from, result,  destPos: 0, result.length);
                sort(result,  from: 0,  to: to - from,forkJoinPool);
                return result;
            },forkJoinPool
    );
}
```

Main Method:

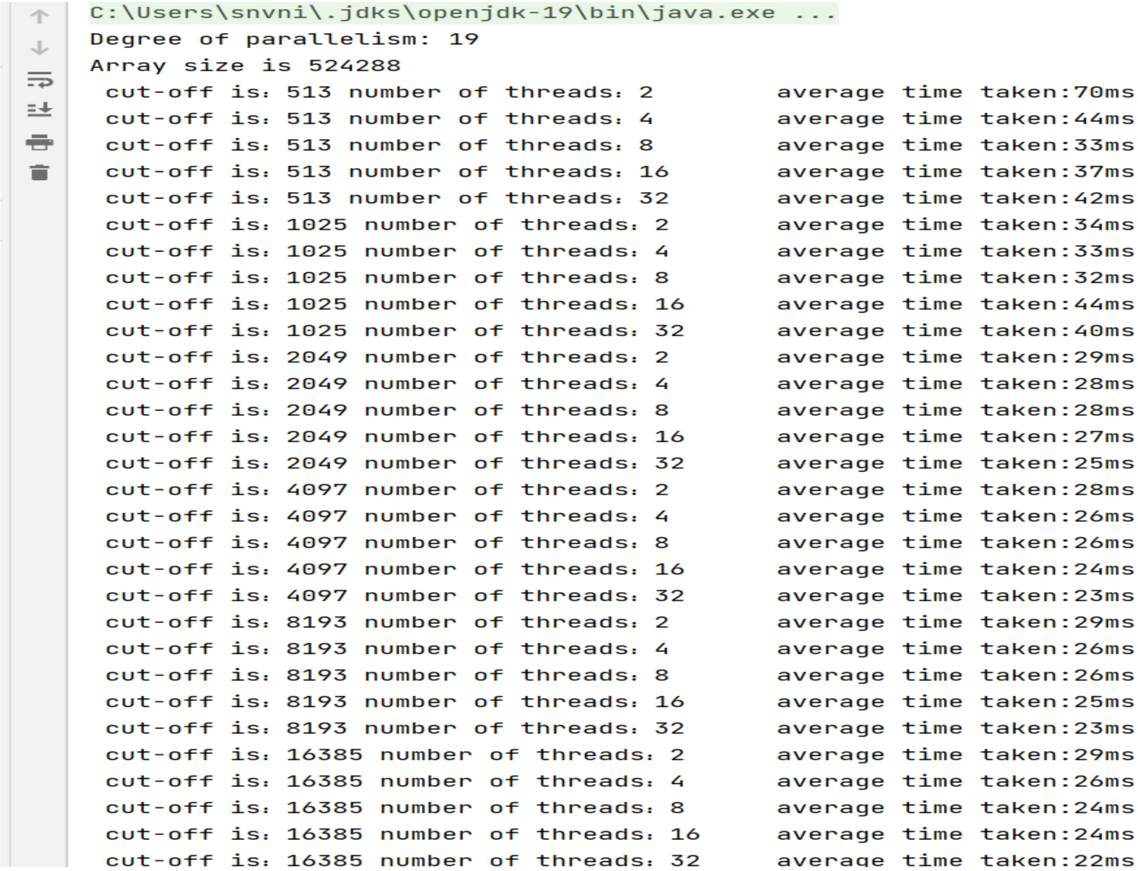


```
public class Main {

     Naga Venkata Nishanth Sayana +1 *
    public static void main(String[] args) {
        processArgs(args);
        System.out.println("Degree of parallelism: " + ForkJoinPool.getCommonPoolParallelism());

        int[] arrLengths = {524288,1048576,2097152,4194304};

        int[] threadNum = {2, 4, 8, 16,32};

        ForkJoinPool forkPool;

        Random random = new Random();
        int[] array;
        ArrayList<Long> timeList = new ArrayList<>();
        for (int i = 0; i < arrLengths.length; i++) {
            int length = arrLengths[i];
            array = new int[length];
            System.out.println("Array size is " + length);

            //cut-off lengths according to levels of recursion tree
            int[] cutoffLen = {length / 1024 + 1, length / 512 + 1, length / 256 + 1, length / 128 + 1,
                    length / 64 + 1, length / 32 + 1, length / 16 + 1, length / 8 + 1, length / 4 + 1,
                    length / 2 + 1, length + 1};
```

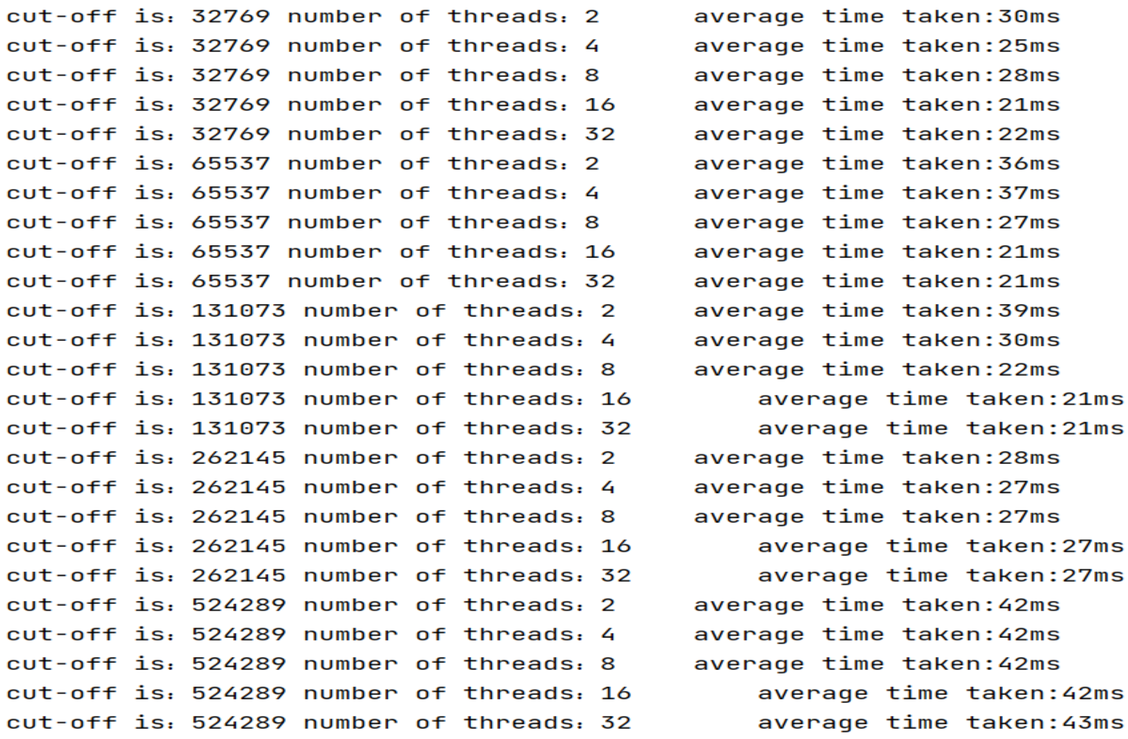


```
            for (int c = 0; c < cutoffLen.length; c++) {
                ParSort.cutoff = cutoffLen[c];
                for (int n = 0; n < threadNum.length; n++) {
                    forkPool = new ForkJoinPool(threadNum[n]);
                    long duration;
                    long startTimems = System.currentTimeMillis();
                    for (int t = 0; t < 10; t++) {
                        for (int j = 0; j < length; j++) array[j] = random.nextInt( bound: 10000000);
                        ParSort.sort(array,  from: 0, array.length, forkPool);
                    }
                    long endTimems = System.currentTimeMillis();
                    duration = (endTimems - startTimems);

                    System.out.println(" cut-off is: " + (ParSort.cutoff) +
                            " number of threads: " + (threadNum[n]) + "\t\taverage time taken:" + (duration / 10) + "ms");
                    timeList.add(duration);
                }
            }
        }
    }
```

Output:



```
C:\Users\snvni\.jdks\openjdk-19\bin\java.exe ...
Degree of parallelism: 19
Array size is 524288
 cut-off is: 513 number of threads: 2       average time taken:70ms
 cut-off is: 513 number of threads: 4       average time taken:44ms
 cut-off is: 513 number of threads: 8       average time taken:33ms
 cut-off is: 513 number of threads: 16      average time taken:37ms
 cut-off is: 513 number of threads: 32      average time taken:42ms
 cut-off is: 1025 number of threads: 2      average time taken:34ms
 cut-off is: 1025 number of threads: 4      average time taken:33ms
 cut-off is: 1025 number of threads: 8      average time taken:32ms
 cut-off is: 1025 number of threads: 16     average time taken:44ms
 cut-off is: 1025 number of threads: 32     average time taken:40ms
 cut-off is: 2049 number of threads: 2      average time taken:29ms
 cut-off is: 2049 number of threads: 4      average time taken:28ms
 cut-off is: 2049 number of threads: 8      average time taken:28ms
 cut-off is: 2049 number of threads: 16     average time taken:27ms
 cut-off is: 2049 number of threads: 32     average time taken:25ms
 cut-off is: 4097 number of threads: 2      average time taken:28ms
 cut-off is: 4097 number of threads: 4      average time taken:26ms
 cut-off is: 4097 number of threads: 8      average time taken:26ms
 cut-off is: 4097 number of threads: 16     average time taken:24ms
 cut-off is: 4097 number of threads: 32     average time taken:23ms
 cut-off is: 8193 number of threads: 2      average time taken:29ms
 cut-off is: 8193 number of threads: 4      average time taken:26ms
 cut-off is: 8193 number of threads: 8      average time taken:26ms
 cut-off is: 8193 number of threads: 16     average time taken:25ms
 cut-off is: 8193 number of threads: 32     average time taken:23ms
 cut-off is: 16385 number of threads: 2     average time taken:29ms
 cut-off is: 16385 number of threads: 4     average time taken:26ms
 cut-off is: 16385 number of threads: 8     average time taken:24ms
 cut-off is: 16385 number of threads: 16    average time taken:24ms
 cut-off is: 16385 number of threads: 32    average time taken:22ms
```



```
cut-off is: 32769 number of threads: 2     average time taken:30ms
cut-off is: 32769 number of threads: 4     average time taken:25ms
cut-off is: 32769 number of threads: 8     average time taken:28ms
cut-off is: 32769 number of threads: 16    average time taken:21ms
cut-off is: 32769 number of threads: 32    average time taken:22ms
cut-off is: 65537 number of threads: 2     average time taken:36ms
cut-off is: 65537 number of threads: 4     average time taken:37ms
cut-off is: 65537 number of threads: 8     average time taken:27ms
cut-off is: 65537 number of threads: 16    average time taken:21ms
cut-off is: 65537 number of threads: 32    average time taken:21ms
cut-off is: 131073 number of threads: 2    average time taken:39ms
cut-off is: 131073 number of threads: 4    average time taken:30ms
cut-off is: 131073 number of threads: 8    average time taken:22ms
cut-off is: 131073 number of threads: 16       average time taken:21ms
cut-off is: 131073 number of threads: 32       average time taken:21ms
cut-off is: 262145 number of threads: 2    average time taken:28ms
cut-off is: 262145 number of threads: 4    average time taken:27ms
cut-off is: 262145 number of threads: 8    average time taken:27ms
cut-off is: 262145 number of threads: 16       average time taken:27ms
cut-off is: 262145 number of threads: 32       average time taken:27ms
cut-off is: 524289 number of threads: 2    average time taken:42ms
cut-off is: 524289 number of threads: 4    average time taken:42ms
cut-off is: 524289 number of threads: 8    average time taken:42ms
cut-off is: 524289 number of threads: 16       average time taken:42ms
cut-off is: 524289 number of threads: 32       average time taken:43ms
```

Observations:

I have taken arrays of different lengths ranging from 524288 to 4194304(Powers of 2), and I have given number of threads as input ranging from 2 to 32(Powers of 2) and varied the cut off values based on the lengths of each array. I picked the cut off value as Cut- off=(length/2^L) +1, as L would be the number of levels of the recursion tree, this way, I could control until which level of the recursion tree, the parallel sort can be performed.

Here are the values that I have tabulated:

a) For Array of length=524288

Cutoff	2 threads	4 threads	8 threads	16 threads	32 threads
513	70ms	44ms	33ms	37ms	42ms
1025	34ms	33ms	32ms	44ms	40ms
2049	29ms	28ms	28ms	27ms	25ms
4097	28ms	26ms	26ms	24ms	23ms
8193	29ms	26ms	26ms	25ms	23ms
16385	29ms	26ms	24ms	24ms	22ms
32769	30ms	25ms	28ms	21ms	22ms
65537	36ms	37ms	27ms	21ms	21ms
131073	39ms	30ms	22ms	21ms	21ms
262145	28ms	27ms	27ms	27ms	27ms
524289	42ms	42ms	42ms	42ms	43ms

b) For Array of length= 1048576

Cutoff	2 threads	4 threads	8 threads	16 threads	32 threads
1025	82ms	62ms	56ms	55ms	56ms
2049	61ms	55ms	51ms	49ms	59ms
4097	61ms	55ms	49ms	47ms	49ms
8193	55ms	50ms	48ms	50ms	43ms
16385	56ms	49ms	49ms	47ms	45ms
32769	58ms	51ms	53ms	51ms	40ms
65537	62ms	65ms	68ms	49ms	46ms
131073	80ms	70ms	56ms	45ms	39ms
262145	85ms	62ms	44ms	44ms	44ms
524289	58ms	57ms	57ms	57ms	58ms
1048577	91ms	89ms	90ms	92ms	92ms

c) For Array of length= 2097152

Cutoff	2 threads	4 threads	8 threads	16 threads	32 threads
2049	135ms	108ms	114ms	99ms	93ms
4097	122ms	96ms	107ms	91ms	87ms
8193	108ms	100ms	88ms	103ms	97ms
16385	110ms	94ms	96ms	104ms	89ms
32769	108ms	101ms	96ms	96ms	105ms
65537	115ms	98ms	131ms	112ms	77ms
131073	138ms	143ms	126ms	95ms	77ms
262145	145ms	151ms	114ms	81ms	71ms
524289	164ms	130ms	92ms	131ms	173ms
1048577	247ms	250ms	242ms	244ms	242ms
2097153	396ms	395ms	400ms	404ms	408ms

d) For Array of length= 4194304

Cutoff	2 threads	4 threads	8 threads	16 threads	32 threads
4097	491ms	416ms	410ms	407ms	419ms
8193	447ms	400ms	409ms	404ms	381ms
16385	447ms	383ms	393ms	392ms	388ms
32769	443ms	398ms	379ms	385ms	374ms
65537	476ms	406ms	438ms	392ms	390ms
131073	515ms	513ms	524ms	410ms	315ms
262145	501ms	556ms	496ms	387ms	332ms
524289	623ms	646ms	475ms	318ms	302ms
1048577	704ms	265ms	187ms	184ms	191ms
2097153	470ms	504ms	500ms	502ms	500ms
4194305	834ms	838ms	835ms	413ms	398ms

Based on the above observations, I concluded that,

The performance gains from increasing the number of threads are not consistent across different cutoff values. In some cases, increasing the number of threads leads to a significant reduction in execution time, while in other cases the reduction is relatively small or even negligible.

In some cases, there appears to be significant variability in execution times across multiple runs, as indicated by the standard deviations of the measurements. This could be due to factors such as system load or other processes running on the computer.

Overall, the data suggests that increasing the number of threads can lead to improved performance for lower cutoff values, but there is a diminishing return to using more threads as the cutoff value increases. Therefore, choosing the optimal number of threads for a given task requires careful consideration of factors such as the size of the problem and the available computing resources.