Program Structures and Algorithms
Spring 2023(SEC 03)

NAME: Vipul Rajderkar
NUID: 002700991

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.

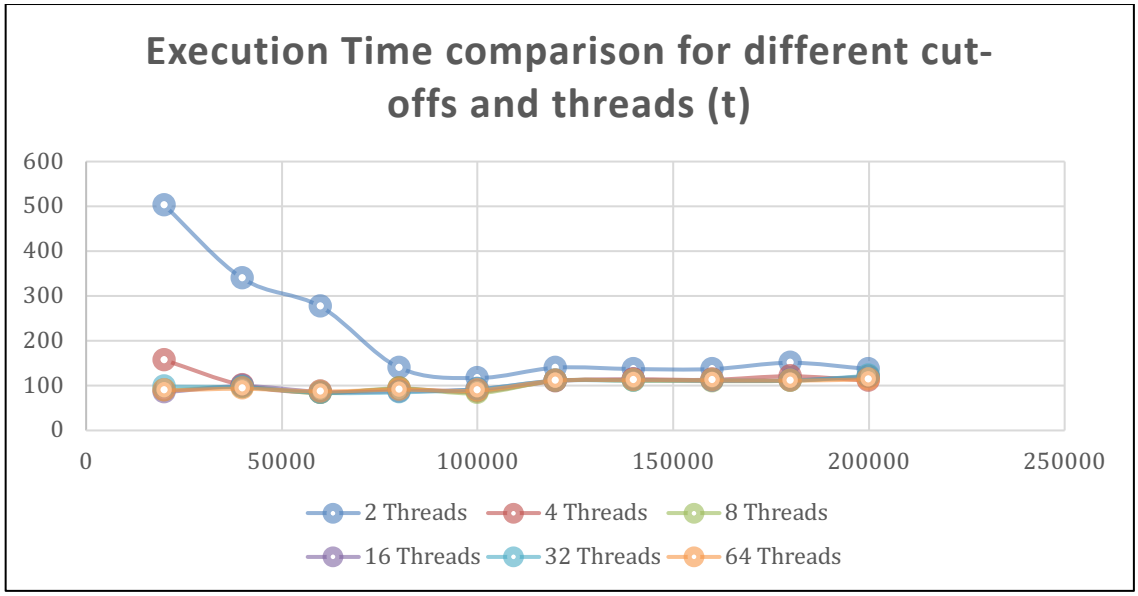
Relationship Conclusion:

It can be inferred from the evidence mentioned below, that the lowest execution time is attained for the cut-off value of approximately 25% of the array's size and with 8 threads.

Evidence to support that conclusion and graphical representation:

Array Size : 200000

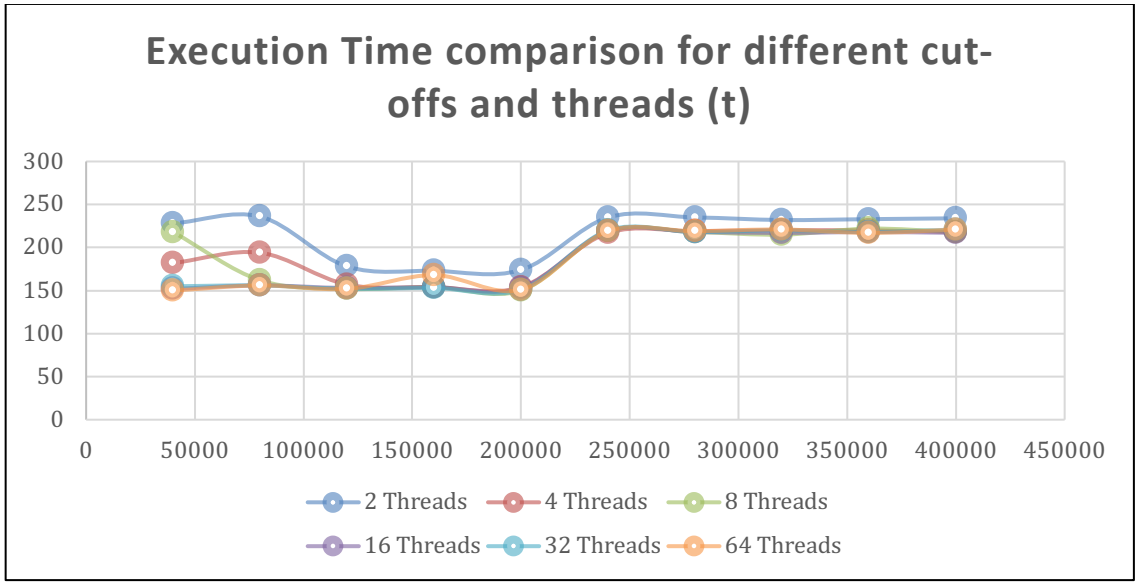
	Execution Time in milliseconds for different number of threads and cut-offs					
Cut-off	**2 Threads**	**4 Threads**	**8 Threads**	**16 Threads**	**32 Threads**	**64 Threads**
20000	502	157	89	85	98	89
40000	339	101	97	99	96	93
60000	276	84	83	86	84	87
80000	140	93	94	87	85	91
100000	117	86	83	92	92	89
120000	140	110	111	111	111	111
140000	137	114	111	113	111	112
160000	137	113	110	111	112	112
180000	151	121	111	112	111	111
200000	137	111	120	120	122	114



Array Size : 400000

Execution Time in milliseconds for different number of threads and cut-offs

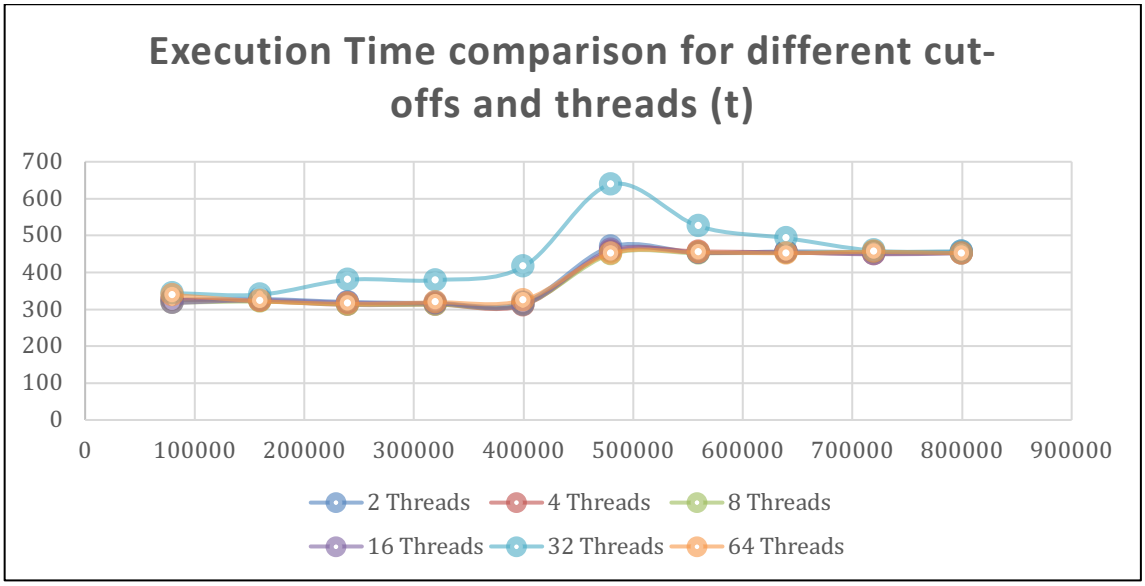
Cut-off	2 Threads	4 Threads	8 Threads	16 Threads	32 Threads	64 Threads
40000	228	182	218	152	155	150
80000	236	194	162	156	156	156
120000	178	157	152	153	152	152
160000	173	154	154	154	153	168
200000	174	154	150	153	151	151
240000	235	217	219	219	220	219
280000	235	219	218	218	218	219
320000	232	220	215	217	220	221
360000	233	220	222	218	218	217
400000	234	218	218	217	221	221



Array Size : 800000

Execution Time in milliseconds for different number of threads and cut-offs

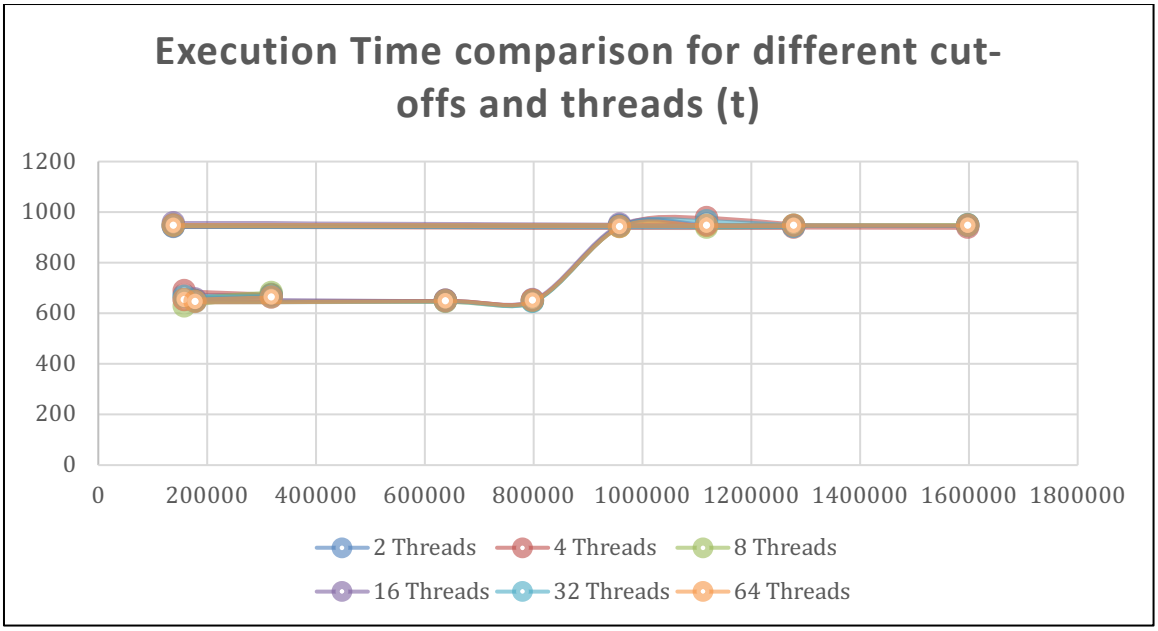
Cut-off	2 Threads	4 Threads	8 Threads	16 Threads	32 Threads	64 Threads
80000	326	326	318	317	344	339
160000	328	322	321	325	340	322
240000	320	312	313	318	380	315
320000	317	313	313	316	379	319
400000	318	311	313	313	416	325
480000	470	457	450	461	639	452
560000	453	457	452	453	525	454
640000	457	454	453	454	493	452
720000	455	450	455	449	459	456
800000	456	453	452	452	457	452



Array Size : 1600000

Execution Time in milliseconds for different number of threads and cut-offs

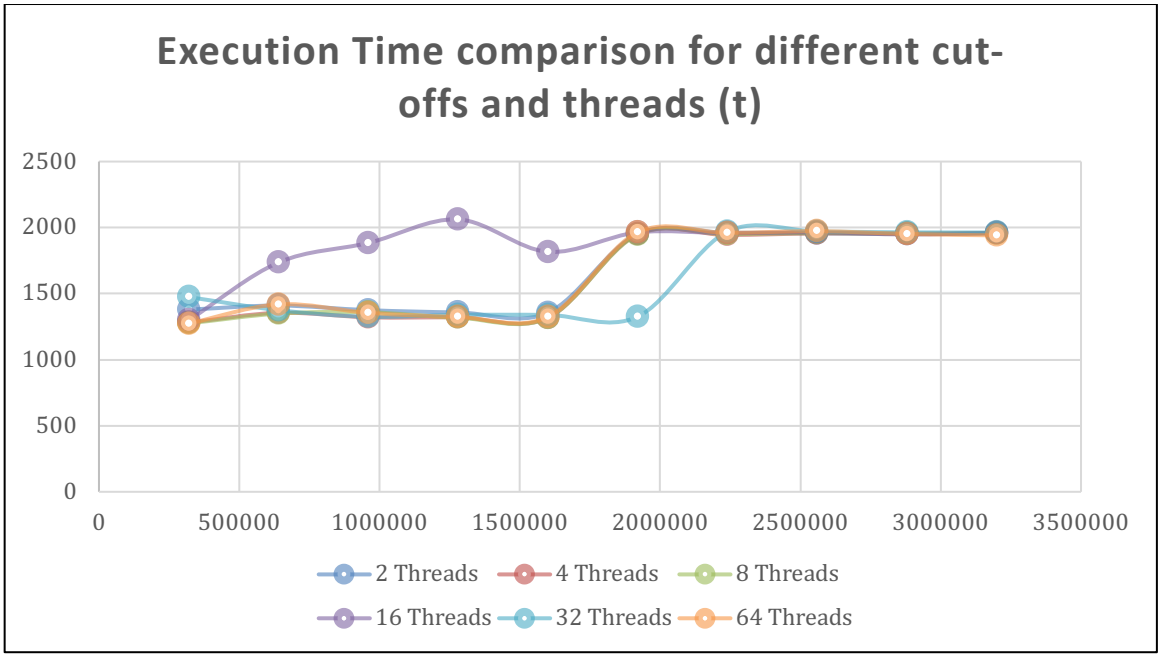
Cut-off	2 Threads	4 Threads	8 Threads	16 Threads	32 Threads	64 Threads
160000	671	687	627	653	666	653
320000	672	670	681	663	666	662
180000	646	647	651	656	645	644
640000	649	649	645	648	646	648
800000	647	649	646	652	645	649
960000	946	944	941	950	943	941
1120000	946	977	939	950	964	945
1280000	945	939	947	943	941	946
140000	942	943	948	957	944	947
1600000	948	938	949	945	946	947



Array Size : 3200000

Execution Time in milliseconds for different number of threads and cut-offs

Cut-off	2 Threads	4 Threads	8 Threads	16 Threads	32 Threads	64 Threads
320000	1379	1281	1275	1299	1474	1274
640000	1410	1355	1349	1736	1375	1419
960000	1375	1321	1359	1884	1325	1355
1280000	1358	1321	1318	2062	1339	1324
1600000	1356	1317	1317	1817	1339	1324
1920000	1952	1950	1947	1964	1326	1964
2240000	1961	1944	1950	1958	1970	1960
2560000	1957	1956	1959	1964	1970	1973
2880000	1951	1955	1954	1948	1964	1952
3200000	1956	1959	1957	1960	1963	1941



Code Snippet:

Changes in main clas:

```java
public static void main(String[] args) {

    int[] size = {200000, 400000, 800000, 1600000, 3200000};


    processArgs(args);
    for(int arrSize : size) {
        int threadCount = 2;
        System.out.println("Size of Array : " + arrSize);
        System.out.println("-----------------------------------");

        while (threadCount < 128) {

            ForkJoinPool pool = new ForkJoinPool(threadCount);
            System.out.println("Degree of parallelism : " + pool.getParallelism());

            Random random = new Random();
            int[] array = new int[arrSize];

            ArrayList<Long> timeList = new ArrayList<>();

            for (int j = 0; j < 10; j++) {
                ParSort.cutoff = (arrSize / 10) * (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("cutoff: " + (ParSort.cutoff) + "\t\t10times Time:" + time + "ms");

            }
            try {
```

Output:

For entire output logs please refer Parallel Sorting Output.docx file (Added in the repository)

Sample Output: