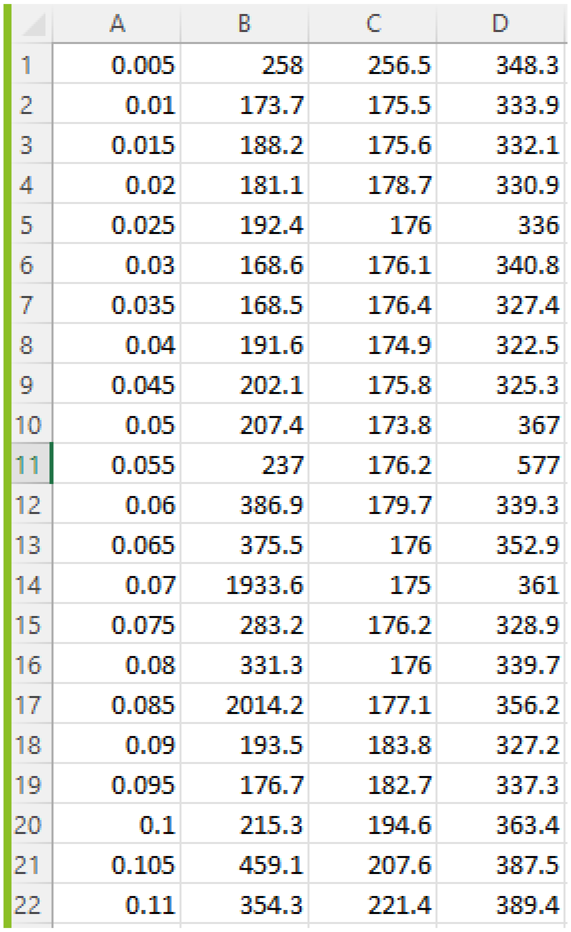
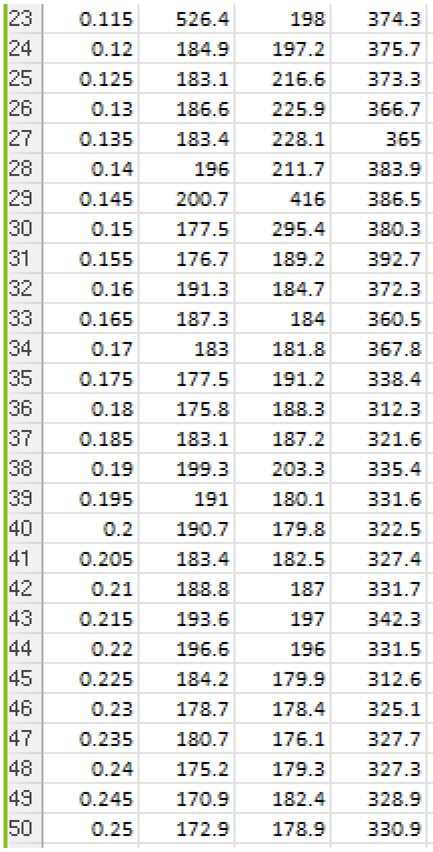
Report

There are 3 conditions to perform this task. One when the to and from means the length of array is less than the cutoff then we will simply perform Arrays.sort function. The second possibility is when threadCount is less than 1 then there is no need to do parSort algorithm we can choose any simple sorting algorithm to sort. In this case we perform merge sort for less time complexity. And the third condition when the threadCount is greater than 1 and cut off is higher than we will perform the parSort algorithm. In my code, I pass the value of cutoff = 10000 * (j+1).

Additionally, for my computer the parallelism ends up with 3 by which I have created multiple csv file and the final result/ conclusion. I tried with threadCount as 8, 4 and 2 respectively. Then when a processor splits the task into two parts and pass each one of them to another processor, the count is divided by two. When the count is 1, it means that there are 8 threads solving the smallest problem concurrently, and it is no way to divide the problem again. The results are as below.



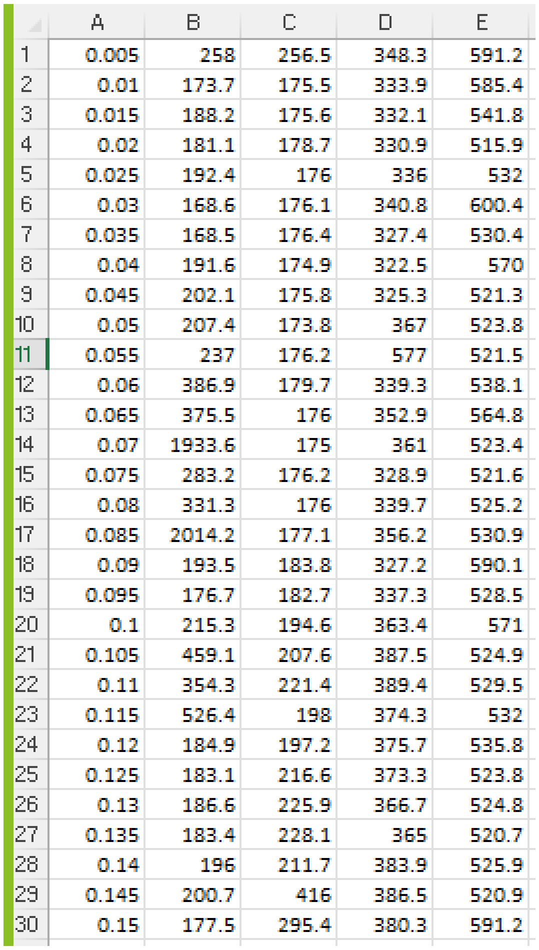
	A	B	C	D
1	0.005	258	256.5	348.3
2	0.01	173.7	175.5	333.9
3	0.015	188.2	175.6	332.1
4	0.02	181.1	178.7	330.9
5	0.025	192.4	176	336
6	0.03	168.6	176.1	340.8
7	0.035	168.5	176.4	327.4
8	0.04	191.6	174.9	322.5
9	0.045	202.1	175.8	325.3
10	0.05	207.4	173.8	367
11	0.055	237	176.2	577
12	0.06	386.9	179.7	339.3
13	0.065	375.5	176	352.9
14	0.07	1933.6	175	361
15	0.075	283.2	176.2	328.9
16	0.08	331.3	176	339.7
17	0.085	2014.2	177.1	356.2
18	0.09	193.5	183.8	327.2
19	0.095	176.7	182.7	337.3
20	0.1	215.3	194.6	363.4
21	0.105	459.1	207.6	387.5
22	0.11	354.3	221.4	389.4



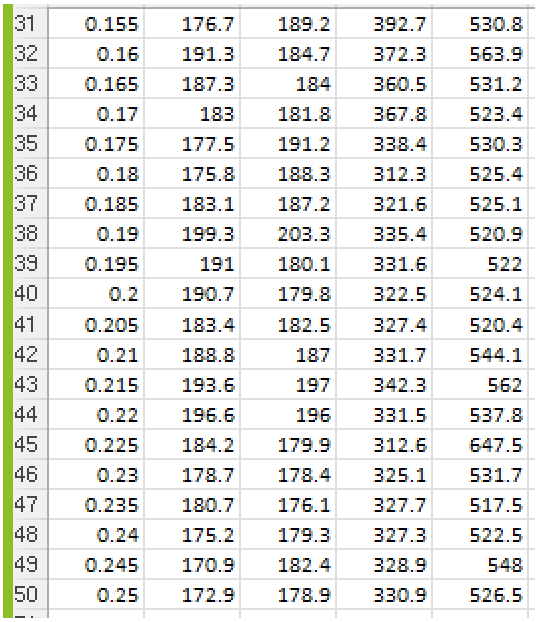
23	0.115	526.4	198	374.3
24	0.12	184.9	197.2	375.7
25	0.125	183.1	216.6	373.3
26	0.13	186.6	225.9	366.7
27	0.135	183.4	228.1	365
28	0.14	196	211.7	383.9
29	0.145	200.7	416	386.5
30	0.15	177.5	295.4	380.3
31	0.155	176.7	189.2	392.7
32	0.16	191.3	184.7	372.3
33	0.165	187.3	184	360.5
34	0.17	183	181.8	367.8
35	0.175	177.5	191.2	338.4
36	0.18	175.8	188.3	312.3
37	0.185	183.1	187.2	321.6
38	0.19	199.3	203.3	335.4
39	0.195	191	180.1	331.6
40	0.2	190.7	179.8	322.5
41	0.205	183.4	182.5	327.4
42	0.21	188.8	187	331.7
43	0.215	193.6	197	342.3
44	0.22	196.6	196	331.5
45	0.225	184.2	179.9	312.6
46	0.23	178.7	178.4	325.1
47	0.235	180.7	176.1	327.7
48	0.24	175.2	179.3	327.3
49	0.245	170.9	182.4	328.9
50	0.25	172.9	178.9	330.9

From the output of 8, 4, 2 I noticed that when I decreased the threadCount it was taking more time result was changing with good margin. Now, from the output I noticed that for threadCount 8 and 4 the result was close to each other because for both it was going to parSort. And both were dividing by 2 will give close similar result. But for threadCount 2 we can see the running time is large.

The big difference came when I used the merge sort and compare the result with ParSort and the result is ass below.



	A	B	C	D	E
1	0.005	258	256.5	348.3	591.2
2	0.01	173.7	175.5	333.9	585.4
3	0.015	188.2	175.6	332.1	541.8
4	0.02	181.1	178.7	330.9	515.9
5	0.025	192.4	176	336	532
6	0.03	168.6	176.1	340.8	600.4
7	0.035	168.5	176.4	327.4	530.4
8	0.04	191.6	174.9	322.5	570
9	0.045	202.1	175.8	325.3	521.3
10	0.05	207.4	173.8	367	523.8
11	0.055	237	176.2	577	521.5
12	0.06	386.9	179.7	339.3	538.1
13	0.065	375.5	176	352.9	564.8
14	0.07	1933.6	175	361	523.4
15	0.075	283.2	176.2	328.9	521.6
16	0.08	331.3	176	339.7	525.2
17	0.085	2014.2	177.1	356.2	530.9
18	0.09	193.5	183.8	327.2	590.1
19	0.095	176.7	182.7	337.3	528.5
20	0.1	215.3	194.6	363.4	571
21	0.105	459.1	207.6	387.5	524.9
22	0.11	354.3	221.4	389.4	529.5
23	0.115	526.4	198	374.3	532
24	0.12	184.9	197.2	375.7	535.8
25	0.125	183.1	216.6	373.3	523.8
26	0.13	186.6	225.9	366.7	524.8
27	0.135	183.4	228.1	365	520.7
28	0.14	196	211.7	383.9	525.9
29	0.145	200.7	416	386.5	520.9
30	0.15	177.5	295.4	380.3	591.2



31	0.155	176.7	189.2	392.7	530.8
32	0.16	191.3	184.7	372.3	563.9
33	0.165	187.3	184	360.5	531.2
34	0.17	183	181.8	367.8	523.4
35	0.175	177.5	191.2	338.4	530.3
36	0.18	175.8	188.3	312.3	525.4
37	0.185	183.1	187.2	321.6	525.1
38	0.19	199.3	203.3	335.4	520.9
39	0.195	191	180.1	331.6	522
40	0.2	190.7	179.8	322.5	524.1
41	0.205	183.4	182.5	327.4	520.4
42	0.21	188.8	187	331.7	544.1
43	0.215	193.6	197	342.3	562
44	0.22	196.6	196	331.5	537.8
45	0.225	184.2	179.9	312.6	647.5
46	0.23	178.7	178.4	325.1	531.7
47	0.235	180.7	176.1	327.7	517.5
48	0.24	175.2	179.3	327.3	522.5
49	0.245	170.9	182.4	328.9	548
50	0.25	172.9	178.9	330.9	526.5

As you can see mergeSort is taking more time then parSort so we can say that problems using the parallelism scheme are much faster than normal sorting algorithms. And when the array size is small, we can use simple system sort rather than any parallelism sort or normal sort.

The graph for ThreadCount 8,4,2 and merge sort: (Series1: cutoff, Series2: 8 TC, Series3: 4 TC, Series4: 2 TC, Series5: MergeSort)

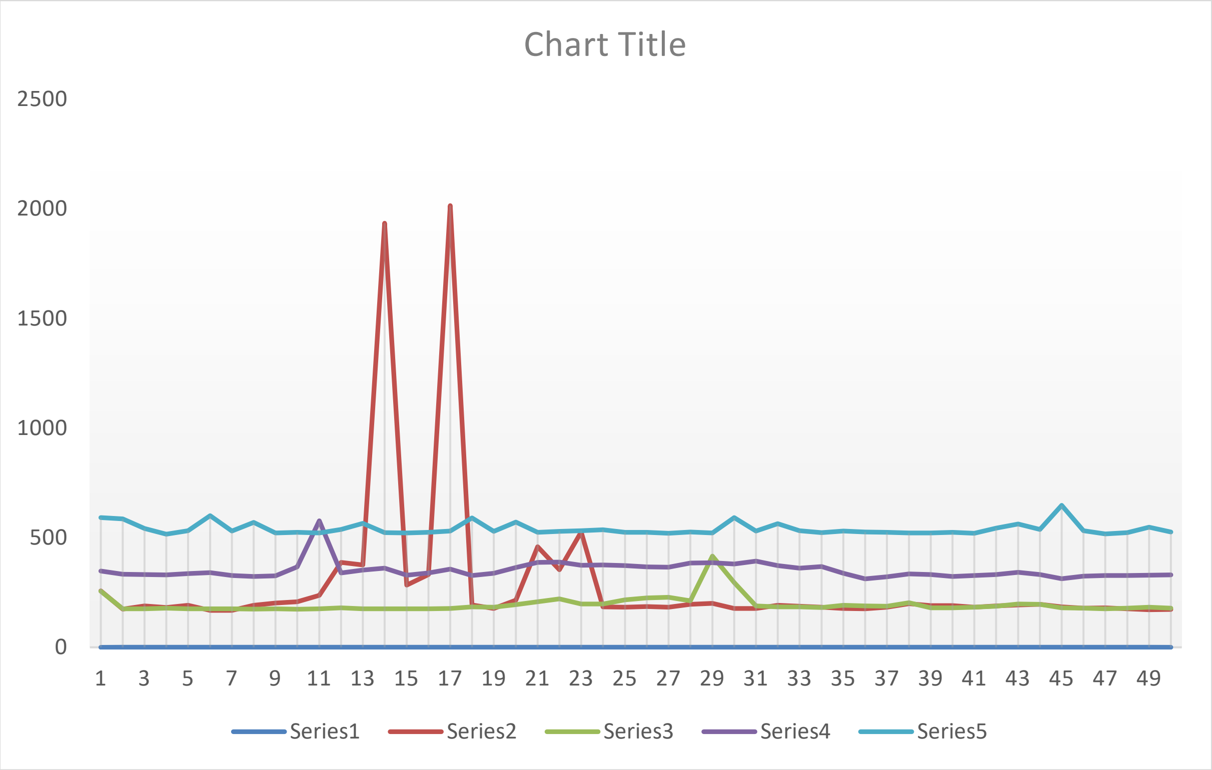


Chart Title
Series1
Series2
Series3
Series4
Series5