Yutong Wang (001530602)

Program Structures & Algorithms
FALL 2021
Assignment No. 5

Task

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.

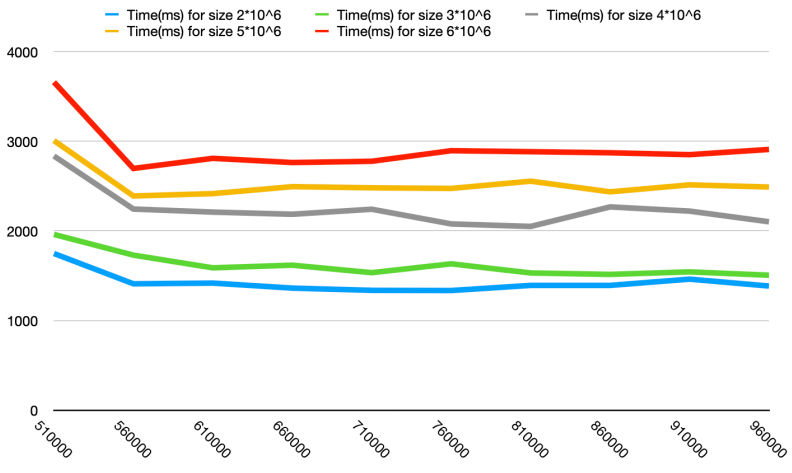
1. A cutoff (defaults to, say, 1000) which you will update according to the first argument in the command line when running. It's your job to experiment and come up with a good value for this cutoff. If there are fewer elements to sort than the cutoff, then you should use the system sort instead.
2. 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.

Outputs & Conclusion

1. find the better cutoff value with degree of parallelism: 2

Degree of Parallelism: 2

Cutoff	Time(ms) for size 2*10^6	Time(ms) for size 3*10^6	Time(ms) for size 4*10^6	Time(ms) for size 5*10^6	Time(ms) for si 6*10^6
510000	1748	1961	2836	3008	3660
560000	1410	1730	2244	2390	2697
610000	1418	1588	2210	2416	2810
660000	1362	1618	2186	2494	2763
710000	1337	1534	2242	2481	2777
760000	1335	1633	2078	2474	2895
810000	1392	1531	2050	2555	2884
860000	1392	1515	2268	2436	2872
910000	1462	1543	2221	2514	2851
960000	1385	1506	2102	2490	2909

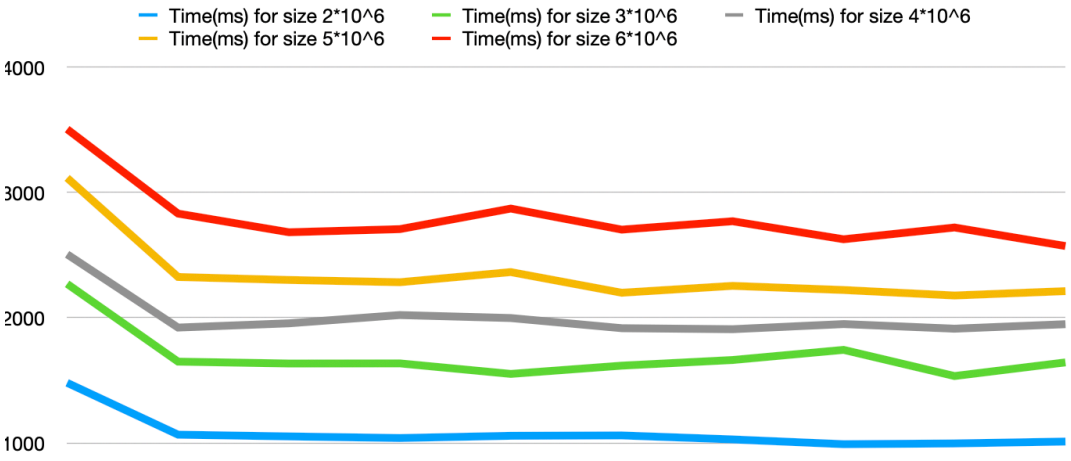


We can see that the time cost shows a significant decline between 510000 to 560000 and then flattens out. I want to narrow down to see if time declines linearly.

From the new result we can see that the significant decline happens again and the time declining is not linearly, it happens after the first cutoff.

Degree of Parallelism: 2

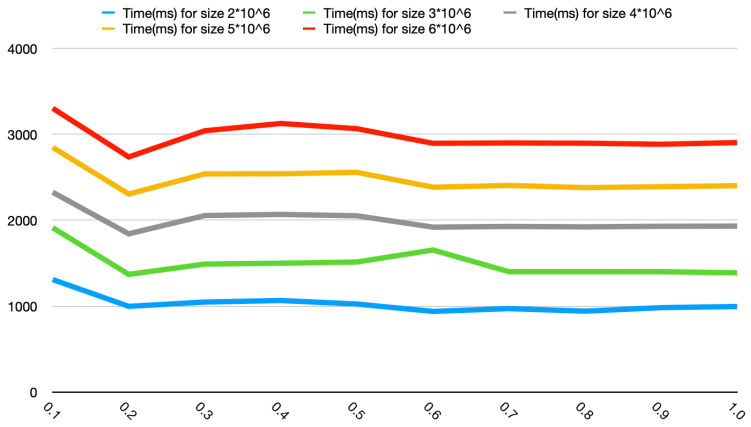
Cutoff(*10000)	Time(ms) for size 2*10^6	Time(ms) for size 3*10^6	Time(ms) for size 4*10^6	Time(ms) for size 5*10^6	Time(ms) for size 6*10^6
50	1480	2270	2506	3116	3505
51	1066	1649	1921	2325	2831
52	1052	1634	1955	2301	2682
53	1039	1635	2021	2283	2707
54	1057	1551	1997	2364	2871
55	1060	1617	1916	2199	2703
56	1028	1662	1908	2254	2770
57	990	1743	1949	2221	2626
58	996	1534	1912	2177	2720
59	1011	1643	1948	2212	2572



From the above results, it is difficult to see the relationship between cutoff and arraysize, so I want to try to change the cutoff with the length of the array.

Degree of Parallelism: 2

Cutoff(*array.length)	Time(ms) for size 2*10^6	Time(ms) for size 3*10^6	Time(ms) for size 4*10^6	Time(ms) for size 5*10^6	Time(ms) for size 6*10^6
0.1	1309	1912	2325	2847	3303
0.2	997	1369	1841	2304	2736
0.3	1047	1489	2054	2539	3041
0.4	1065	1499	2067	2540	3126
0.5	1024	1513	2051	2557	3065
0.6	938	1653	1919	2384	2895
0.7	971	1400	1927	2404	2899
0.8	941	1400	1922	2379	2896
0.9	982	1400	1929	2390	2884
1.0	995	1387	1931	2401	2903

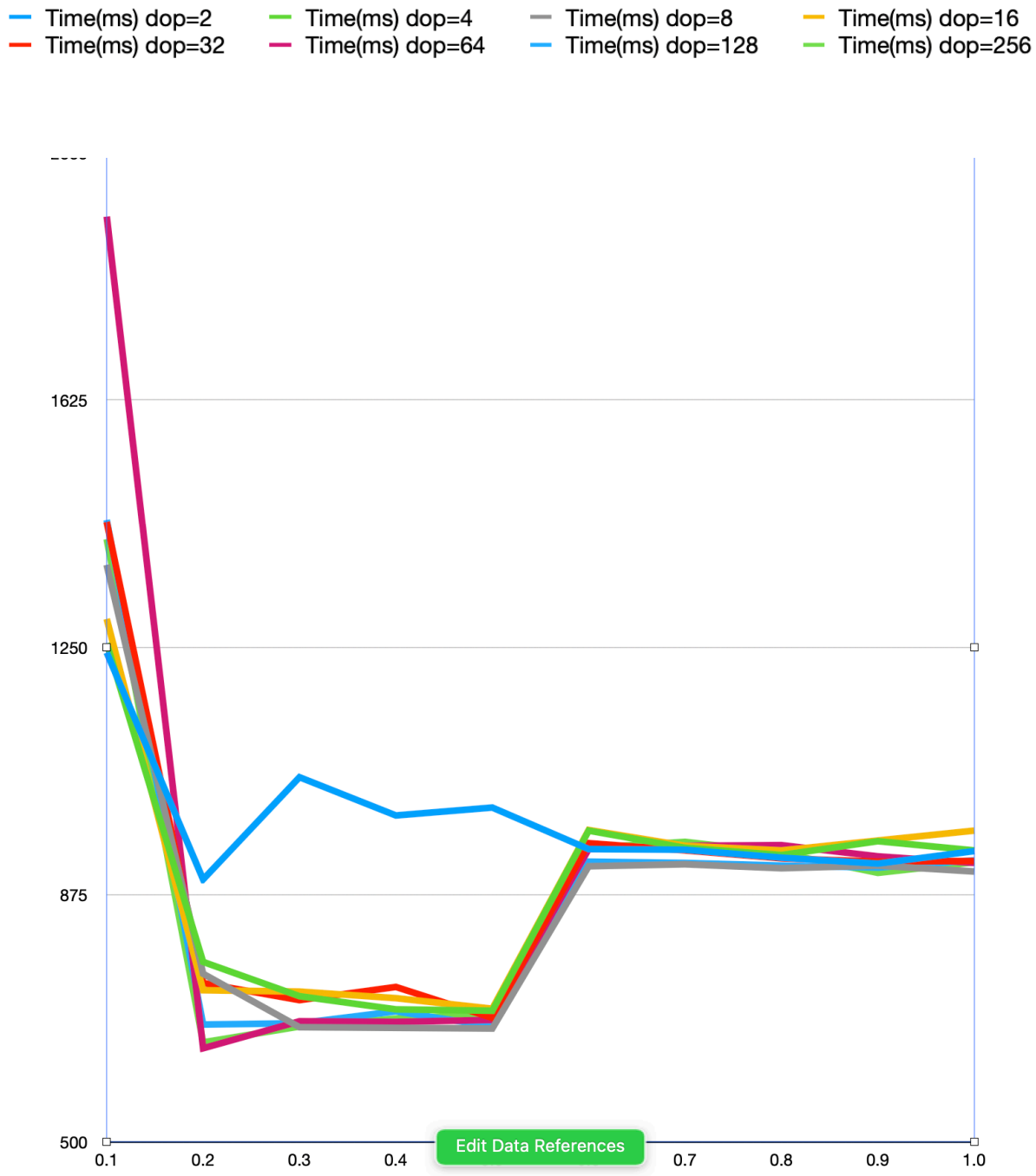


Conclusion: This time the trend has become obvious. We can still see a significant drop after the first cutoff, but this time we can get a minimum point at 0.2, then rise again at 0.3 and then slowly drop and flatten out.

2. find the better thread value with array size: 2000000

Degree of Parallelism(dop)

Cutoff(*array.length)	Time(ms) dop=2	Time(ms) dop=4	Time(ms) dop=8	Time(ms) dop=16	Time(ms) dop=32	Time(ms) dop=64	Time(ms) dop=128	Time(ms) dop=256
0.1	1242	1256	1375	1293	1440	1903	1443	1414
0.2	898	773	755	730	741	642	678	651
0.3	1053	721	674	728	715	683	680	675
0.4	995	701	673	718	735	682	698	687
0.5	1007	699	672	702	688	685	675	694
0.6	944	972	918	973	953	944	925	946
0.7	943	946	921	949	942	949	923	955
0.8	931	935	915	942	930	950	919	938
0.9	922	956	919	957	925	933	916	908
1.0	941	942	910	972	926	923	927	924

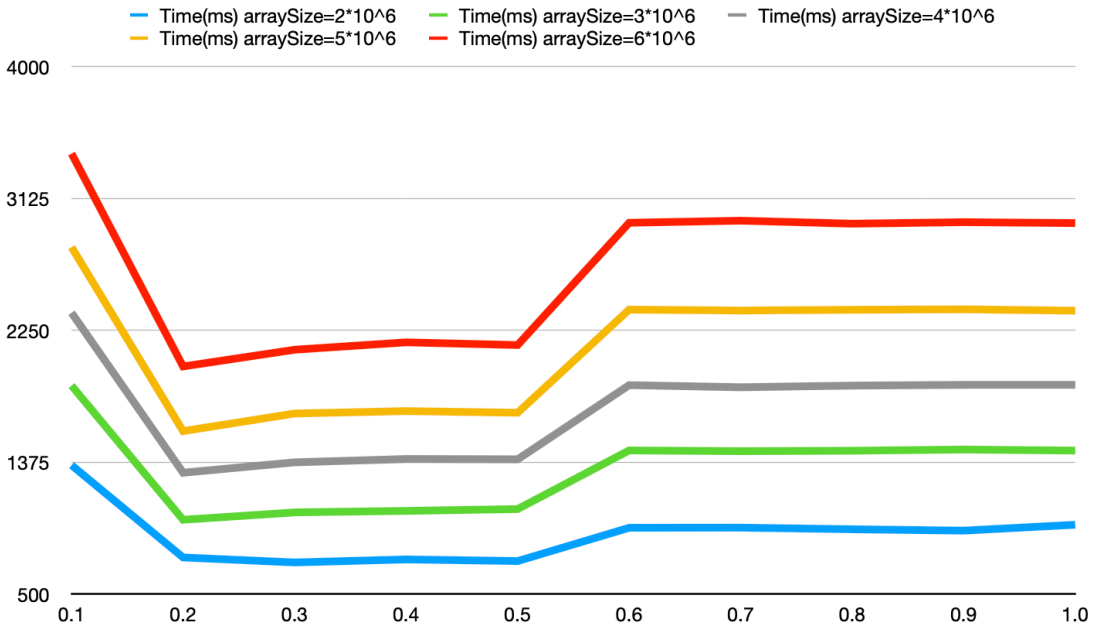


Conclusion: overall we can see that we got smaller time cost when the degree of parallelism or thread is equal to 8.

So I tried to test again with degree of parallelism fixed as 8.

Degree of Parallelism(dop) = 8

Cutoff(*array.length)	Time(ms) arraySize=2*10^6	Time(ms) arraySize=3*10^6	Time(ms) arraySize=4*10^6	Time(ms) arraySize=5*10^6	Time(ms) arraySize=6*10^6
0.1	1356	1884	2367	2802	3423
0.2	744	994	1306	1582	2011
0.3	711	1043	1375	1699	2122
0.4	731	1053	1397	1715	2171
0.5	720	1065	1395	1704	2153
0.6	941	1454	1887	2388	2964
0.7	942	1449	1873	2382	2978
0.8	931	1452	1884	2387	2958
0.9	922	1460	1889	2390	2968
1.0	961	1453	1889	2381	2962



Conclusion: The experimental results verify the conclusion of question 1: When the cutoff point is around 0.2 of the array length, we can get the minimum time cost.

Conclusion

We get the minimum time cost when cutoff = array.length*0.2 and thread = 8.