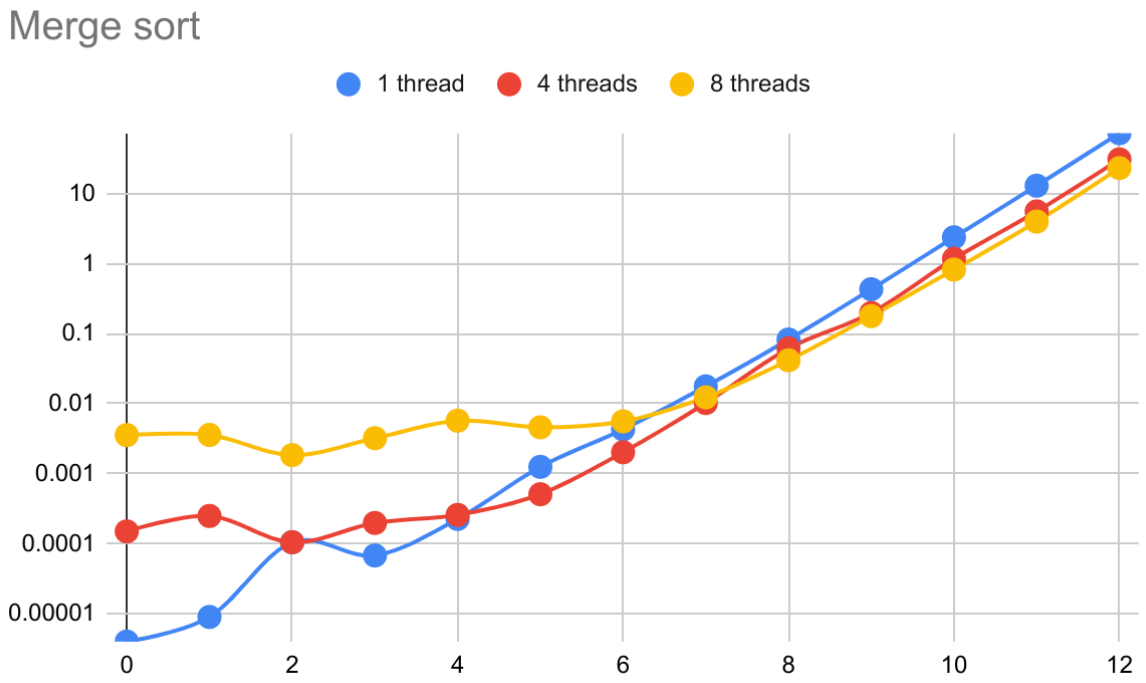
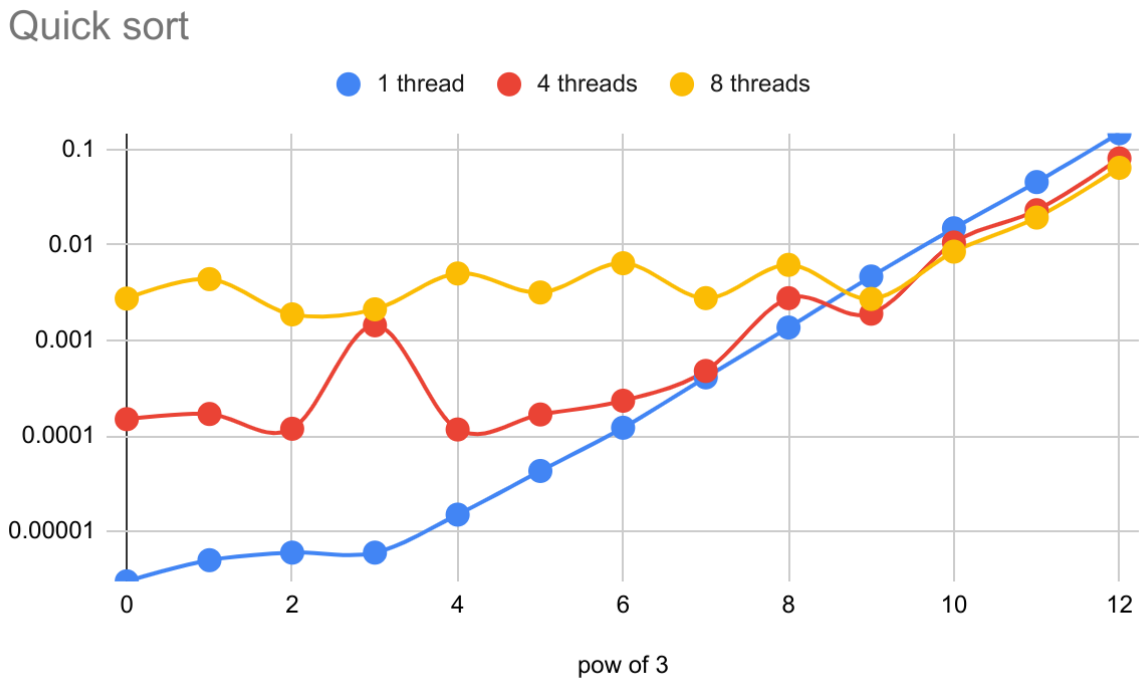
Parallel Sorting Algorithms



pow of 5	1 thread	4 threads	8 threads
0	0.000004	0.00015	0.003562
1	0.000009	0.000249	0.003581
2	0.000105	0.000105	0.001844
3	0.000068	0.000198	0.003214
4	0.000226	0.000258	0.005724
5	0.001253	0.000509	0.004598
6	0.004299	0.002024	0.005595
7	0.01768	0.010221	0.012338
8	0.082588	0.061691	0.041712
9	0.431617	0.197565	0.17834
10	2.378472	1.174473	0.824153
11	12.998823	5.576851	3.996518
12	72.989581	30.904591	23.216717



pow of 3	1 thread	4 threads	8 threads
0	0.000003	0.00015	0.00275
1	0.000005	0.00017	0.004396
2	0.000006	0.000119	0.001879
3	0.000006	0.001443	0.002122
4	0.000015	0.000117	0.005038
5	0.000043	0.000168	0.003184
6	0.000122	0.000234	0.006452
7	0.000411	0.00048	0.002781
8	0.001368	0.002777	0.006202
9	0.004684	0.00191	0.002724
10	0.014987	0.010637	0.008533
11	0.04571	0.023165	0.019415
12	0.148453	0.080873	0.06449

Conclusion

MergeSort and QuickSort show roughly the same results. At first, a single-threaded program overtakes a multi-threaded one. We think that in the beginning it takes a lot of time for a multi-threaded program to create threads, but as the array grows, the parallelism of execution compensates for the time to create threads.

We expected to see a speed increase of n times when using n threads, but our expectations were not met, on average we get a 3 times increase in speed when using 8 threads