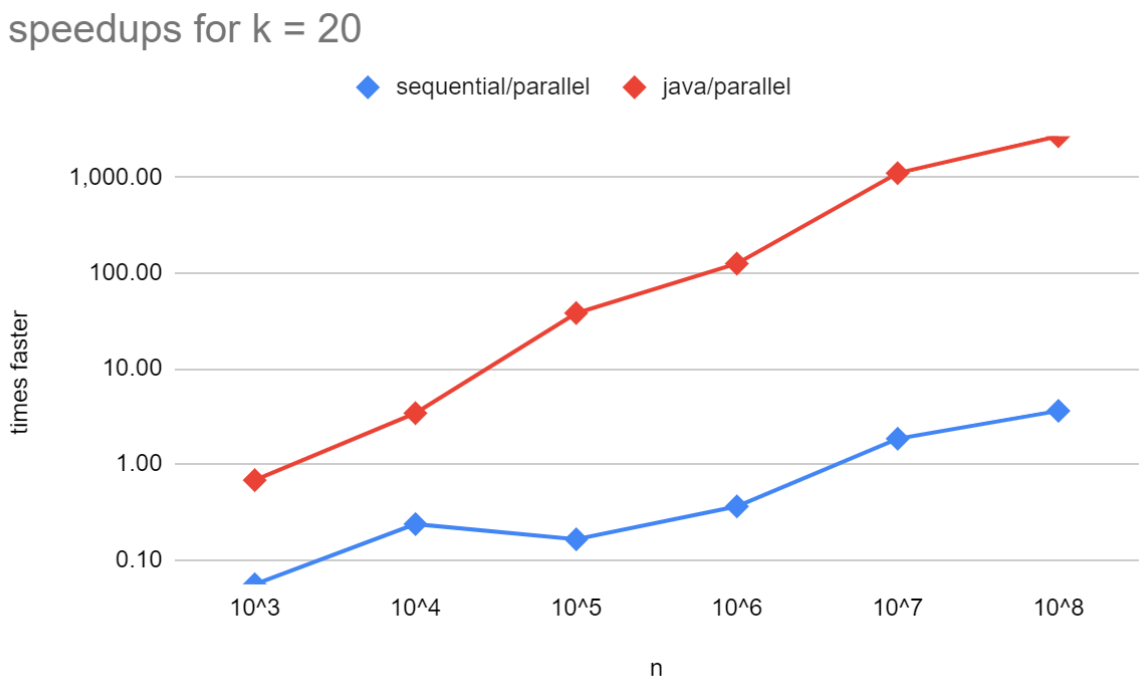
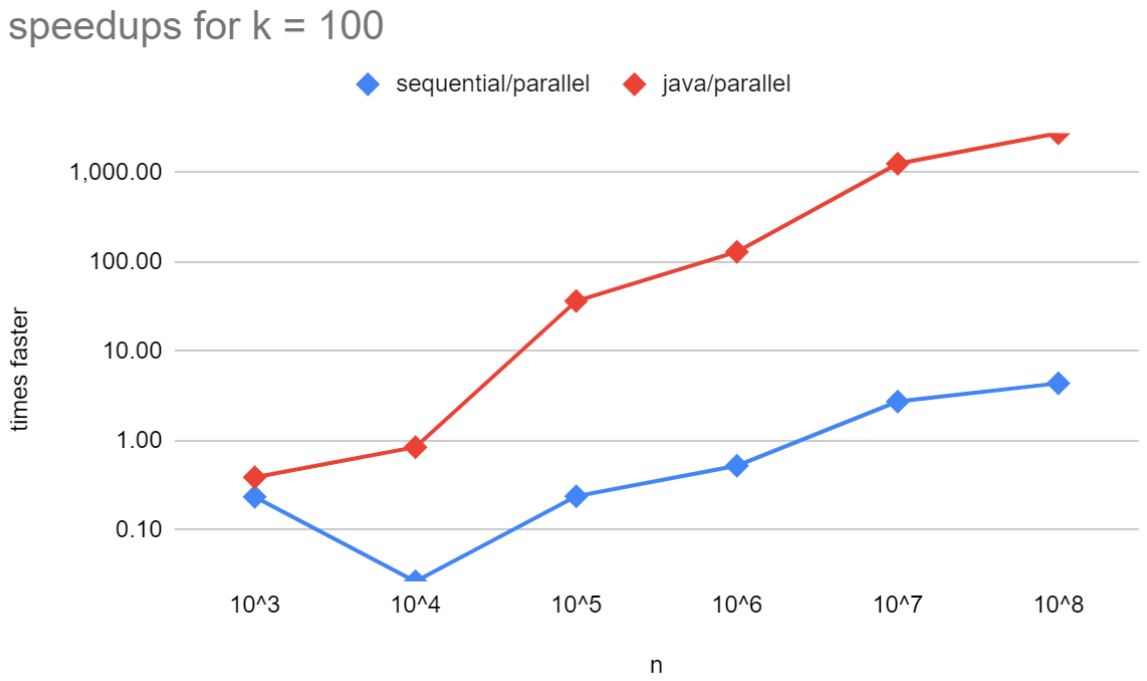
My CPU is an Intel Core i7-1165G7 2.80GHz 4 cores with 8 threads, but actually is 4.70 GHz with the boost option while it's charging. (It is a laptop)
I measured the times and did the table with seven measures for each n and k combination. The raw data can be read in the "Raw" tab in the spreadsheet file.



	medians			speedups	
n	java	sequential	parallel	sequential/parallel	java/parallel
10^3	0.64	0.05	0.94	0.06	0.68
10^4	3.50	0.24	1.02	0.24	3.42
10^5	41.11	0.18	1.08	0.16	38.07
10^6	217.45	0.63	1.73	0.36	125.34
10^7	3683.29	6.16	3.33	1.85	1,105.83
10^8	55376.17	74.22	20.56	3.61	2,693.29

When k = 20 the parallel sort is always better than the java one except for n = 1000, seeing the graph i think we should use java sort until n is less than about 3000.
On the other hand it makes sense to use the parallel solution instead of our sequential sorting algorithm only with n bigger than 10^7.
If we look at the table we can see how both my algorithms are better than the java solution, except for the parallel in n = 10^3. With n = 10^8 it shows how we pass from an entire minute to some milliseconds in both cases.



	medians			speedups	
n	java	sequential	parallel	sequential/parallel	java/parallel
10^3	0.62	0.38	1.59	0.24	0.39
10^4	3.15	0.10	3.71	0.03	0.85
10^5	36.73	0.24	1.01	0.24	36.40
10^6	214.96	0.88	1.66	0.53	129.13
10^7	3,652.65	8.04	2.93	2.74	1,245.23
10^8	51,312.46	82.42	18.79	4.39	2,731.46

When k = 100 the situation is a bit different. The parallel solution starts to beat the java one from 10^5, i think somewhere close to n = 50000. I run the test for 10^4 many times but the parallel solution is always worse when compared to 10^3 or 10^5.
Regarding the sequential solution, it is always better than the parallel until 10^7, looking at the graph should be close to n = 3*10^6.

If we compare the two ks we can see that the k value doesn't have a big effect on timing in general. Sometimes the speed is better, sometimes is worse, but the big difference is made by the n value not the k.