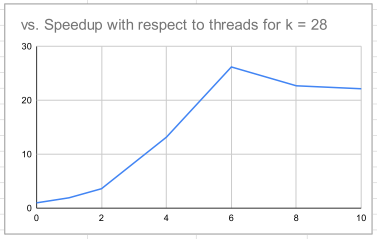
# List Size	# List Size (k)	# Threads	# Threads (q)	# Error	# Time	# qsort_time	# Speedup	# Efficiency	Speedup vs qsort
268435456	28	1	0	0	38.7103	38.6755	1	1	0.9991010145
268435456	28	2	1	0	19.9865	38.6143	1.936822355	0.9684111775	1.932019113
268435456	28	4	2	0	10.6951	38.7942	3.619442548	0.9048606371	3.627287262
268435456	28	16	4	0	2.9381	38.7122	13.17528335	0.8234552091	13.17593002
268435456	28	64	6	0	1.4782	38.7987	26.18745772	0.4091790269	26.24726018
268435456	28	256	8	0	1.7055	39.0104	22.69733216	0.08866145375	22.87329229
268435456	28	1024	10	0	1.7479	39.0302	22.14674753	0.02162768313	22.32976715
1048576	20	1	0	0	0.1084	0.1077	1	1	0.9935424354
1048576	20	2	1	0	0.0583	0.1086	1.859348199	0.9296740995	1.862778731
1048576	20	4	2	0	0.0319	0.1079	3.398119122	0.8495297806	3.382445141
1048576	20	16	4	0	0.0115	0.1099	9.426086957	0.5891304348	9.556521739
1048576	20	64	6	0	0.0096	0.1095	11.29166667	0.1764322917	11.40625
1048576	20	256	8	0	0.0245	0.1097	4.424489796	0.01728316327	4.47755102
1048576	20	1024	10	0	0.0679	0.1089	1.59646539	0.001559048233	1.603829161
4096	12	1	0	0	0.0009	0.0003	1	1	0.3333333333
4096	12	2	1	0	0.0008	0.0003	1.125	0.5625	0.375
4096	12	4	2	0	0.0014	0.0003	0.6428571429	0.1607142857	0.2142857143
4096	12	16	4	0	0.0016	0.0003	0.5625	0.03515625	0.1875
4096	12	64	6	0	0.003	0.0003	0.3	0.0046875	0.1
4096	12	256	8	0	0.0102	0.0003	0.08823529412	0.0003446691176	0.02941176471
4096	12	1024	10	0	0.0496	0.0003	0.01814516129	0.00001771988407	0.006048387097

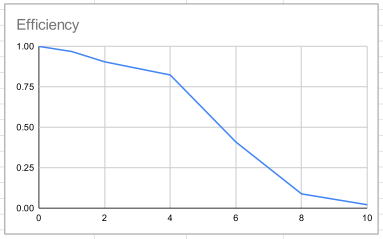
Speedup with respect to threads for k = 28

0	1
1	1.936822355
2	3.619442548
4	13.17528335
6	26.18745772
8	22.69733216
10	22.14674753

Efficiency

0	1
1	0.9684111775
2	0.9048606371
4	0.8234552091
6	0.4091790269
8	0.08866145375
10	0.02162768313





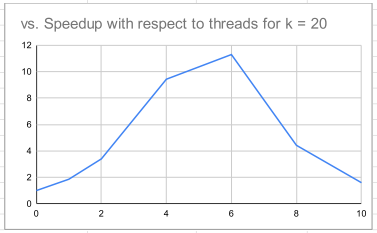
Here, the algorithm scales well, with peak speedup reaching 26x with 64 threads. After this point, there is diminishing returns, which is expected.
Efficiency degrades as thread count increases, which is also expected.

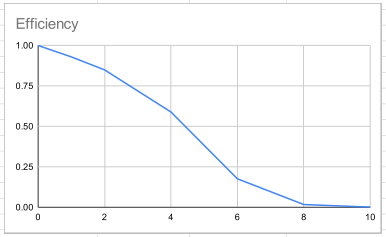
Speedup with respect to threads for k = 20

0	1
1	1.859348199
2	3.398119122
4	9.426086957
6	11.29166667
8	4.424489796
10	1.59646539

Efficiency

0	1
1	0.9296740995
2	0.8495297806
4	0.5891304348
6	0.1764322917
8	0.01728316327
10	0.001559048233





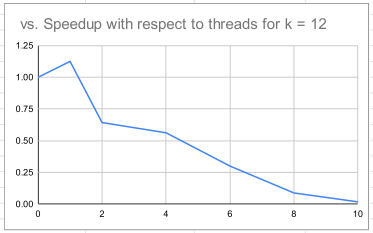
Here, the algorithm doesn't scale quite as well as the previous problem, because there is less work to do. The max speed up still reaches a respectable 11x at 64 threads
Efficiency degrades similarly.

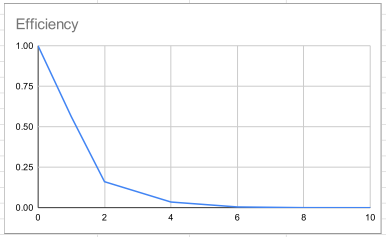
Speedup with respect to threads for k = 12

0	1
1	1.125
2	0.6428571429
4	0.5625
6	0.3
8	0.08823529412
10	0.01814516129

Efficiency

0	1
1	0.5625
2	0.1607142857
4	0.03515625
6	0.0046875
8	0.0003446691176
10	0.00001771988407





Here, performances is bad for anything more than two threads. Evidently when the list is so small, the sorting is trivial and multithreading overhead far exceeds single thread sort time
Efficiency expectedly drops off a cliff.

# List Size	# List Size (k)	# Threads	# Threads (q)	# Error	# Time	# qsort_time	# Speedup	# Efficiency	Speedup vs qsort
part 3.									
q = 6 seems to be the cut off point for diminishing returns, asm demonstrated in both k = 20 and k = 28.									
In terms of varying q, it is up to the application. going beyond q seems to still yield good speed up in large lists, and in much larger lists the scaling will likely be much better, with larger q values being the cut off rather than q = 6									
Based on the plots above and my accompanying analysis, the implementation scales well with large lists, while performing badly against small lists, which is all very expected behavior.									