sameera Oblig 1 report

The CPU is an AMD Ryzen™ 7 4700U-prosessor 2.00 GHz with 8 cores.

Tables:

Algorithm K=20	1000	10 000	100 000	1 000 000	10 000 000	100 000 000
A1(Java sort)	0.86	1,00	41,30	111,90	1366,50	8951,8
A2(Sequential)	0,25	0,30	2,40	4,80	23,50	229,3
Parallel	1,81	1,20	4,60	7,40	21,00	151,7
Speedup (A2/parallel)	0,14	0,25	0,52	0,65	1,12	1,51

Algorithm K=100	1000	10 000	100 000	1 000 000	10 000 000	100 000 000
A1(Java sort)	0.87	1,91	77,10	188,95	1989,35	22989,6
A2(Sequential)	0,41	1,53	5,64	9,99	35,68	291,2
Parallel	1,81	5,10	8,44	10,98	30,08	188,99
Speedup (A2/parallel)	0,23	0,30	0,67	0,91	1,19	1,54

As we can see for k=20, the parallel algorithm does not achieve a speedup over the sequential until n reaches 10 million. The creation and synchronization of threads introduce overhead, for small n this overhead can dominate, leading to a parallel speedup of less than 1. As n increases, the work per thread becomes substantial, and the overhead gets distributed over more significant workloads, resulting in noticeable speedup. It is at n= 10 million that the parallel method begins to show a speedup, which continues to improve, reaching 1.51 at n=100 million.

For k=100, the parallel method shows a similar pattern, with the speedup exceeding 1 starting at n=10 million. This improvement becomes more significant as n increases, achieving a speedup of 1.54 at n=100 million. But if the see at n= 1 million we can see that the sequential and parallel algorithms have a small-time difference, and that is not the case of k=20.

For both k values the Java sort is faster than the parallel algorithm when n is lower than 10 000.

Graph:

The curve for k=100 is consistently above the one for k=20, suggesting that the larger k benefits more from parallelization.