Below are the results for the quicksort program, running sequentially on a single thread:

Sequential	
Results are time in nanoseconds.	
Size \ Thread Count	1
1,000	278400
5,000	1457000
10,000	3303200
100,000	42144200
1,000,000	824186500
2,000,000	2415587000
5,000,000	11617635800
10,000,000	42357152100

The following are the results of 1-5 threads using OpenMP and quicksorting random numbers:

OpenMP				
Results are time in nanoseconds.				
Size \ Thread Count	1	2	4	6
1,000	464700	280400	599900	3273600
5,000	1561200	1428500	1605800	5112900
10,000	3638900	2657000	3885100	7668700
100,000	44332500	43526700	28720700	48475200
1,000,000	829311800	489686800	546305900	620153100
2,000,000	2342612100	2002485900	1457168600	1763231800
5,000,000	11541250500	5833330100	7119770600	8777353000
10,000,000	42041381800	25362875000	22942362700	19123291500

These results show that multithreading definitely reduces the time taken for quicksorting overall, but only becomes really efficient when the dataset arrives at the 10 million mark. The overhead for OpenMP to create and manage threads is less efficient with the smaller datasets and increases the time taken below the 5 million mark.