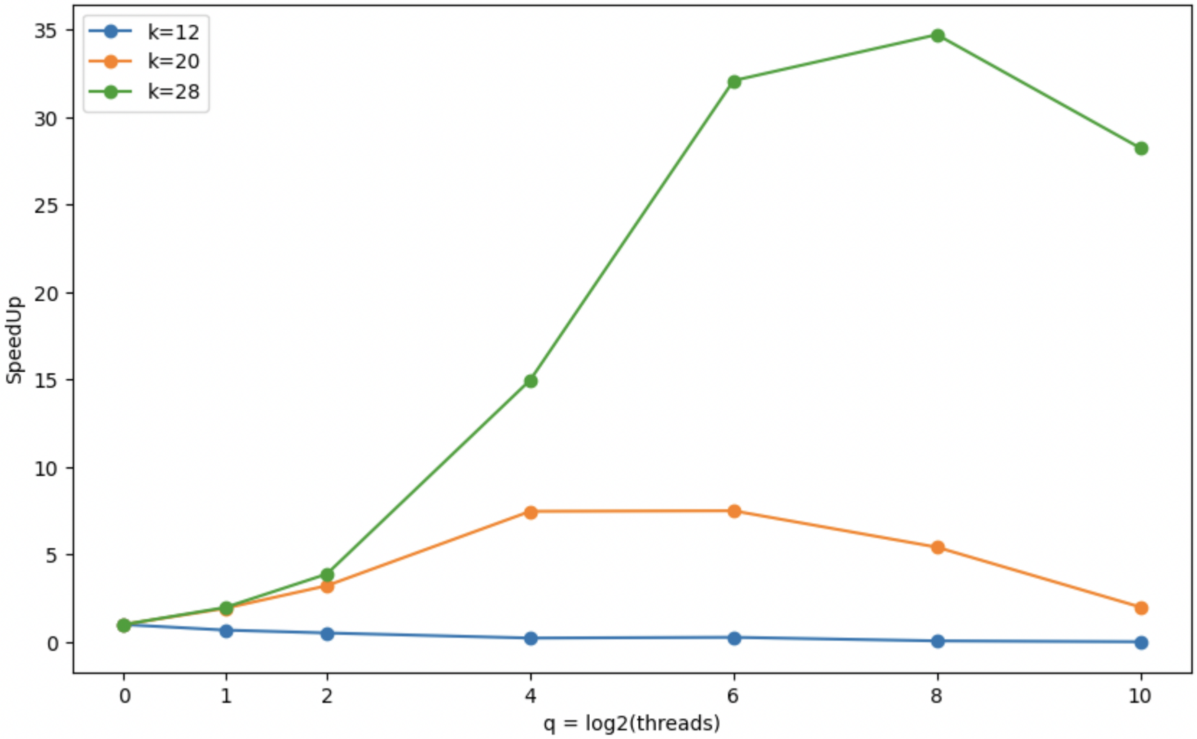
CSCE 735 Parallel Computing

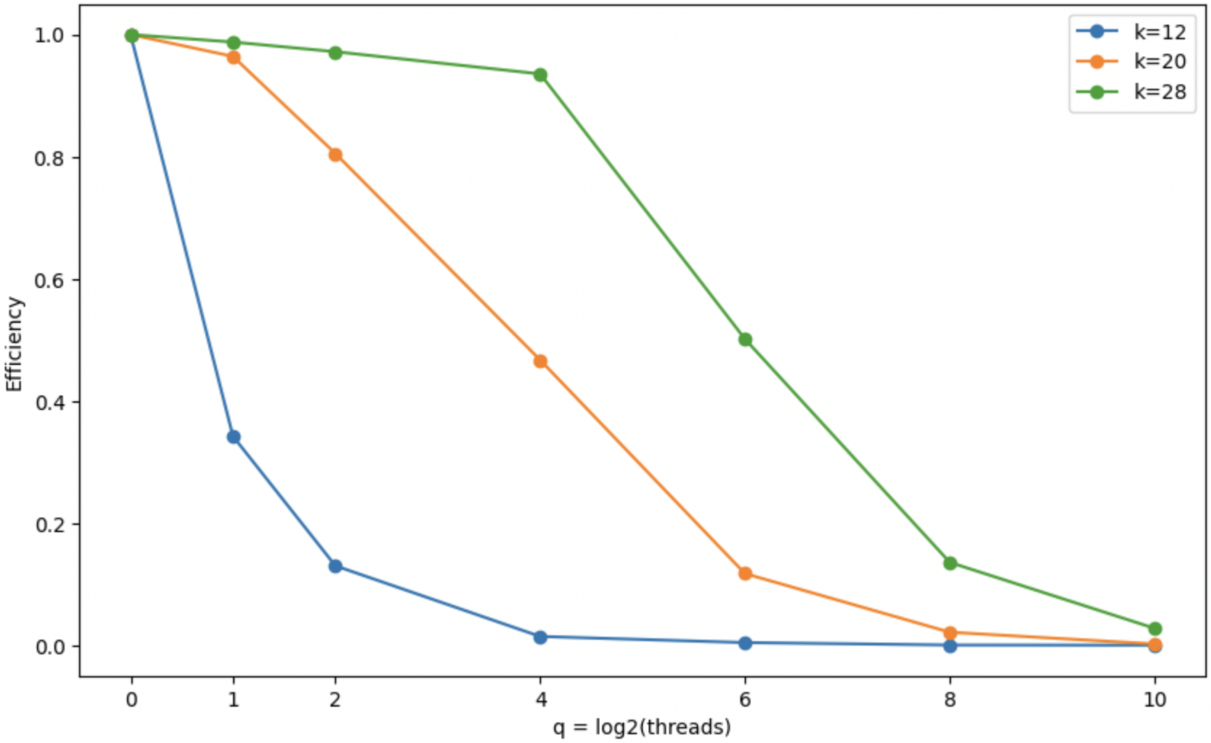
HW 2: Parallel Merge Sort Using Threads

1. **Thread-based parallel merge sort:**
   The code was edited to support multithreading. Below is the output of the program for the given samples.

List Size (k)	Threads (q)	Error	Time (sec)	QSort Time (sec)
16 (4)	2 (1)	0	0.0004	0.0000
16 (4)	4 (2)	0	0.0005	0.0000
16 (4)	8 (3)	0	0.0007	0.0000
1048576 (20)	16 (4)	0	0.0222	0.1776
16777216 (24)	256 (8)	0	0.1866	3.3079

2. **Efficiency and SpeedUp plots for k = 12,20,28 and q = 0,1,2,4,6,8,10**





1. From the speedUp graph, we can infer that there is not much improvement in speedup for k=12. This might be because the thread creation overhead takes more time than the task itself. This can be also observed when we run the program without multithreading(see below). The execution time is lesser for non-parallel code for smaller values of k.
2. The efficiency of execution does not improve much after 6 threads. As we can see, speedup and efficiency at 8 and 10 threads is almost the same for all k. This shows that, even if we increase it to execute in more threads, the speed up is not significant.
3. The efficiency graph for k=12 has a different curve than k=20 and k=28.

Normal sort:

```
List Size = 16, Threads = 2, error = 0, time (sec) =   0.0000, qsort_time =   0.0000
List Size = 16, Threads = 4, error = 0, time (sec) =   0.0000, qsort_time =   0.0000
List Size = 16, Threads = 8, error = 0, time (sec) =   0.0000, qsort_time =   0.0000
List Size = 1048576, Threads = 16, error = 0, time (sec) =   0.1820, qsort_time =   0.1715
List Size = 16777216, Threads = 256, error = 0, time (sec) =   3.6665, qsort_time =   3.3095
```

3. From the experiments of the previous question, we don’t see much improvement with k=12, but we can see significant improvement for k = 20 and k = 28.

Log for the experiments:

```
List Size = 4096, Threads = 1, error = 0, time (sec) =   0.0013, qsort_time =   0.0008
List Size = 4096, Threads = 2, error = 0, time (sec) =   0.0019, qsort_time =   0.0010
List Size = 4096, Threads = 4, error = 0, time (sec) =   0.0025, qsort_time =   0.0006
List Size = 4096, Threads = 16, error = 0, time (sec) =   0.0057, qsort_time =   0.0006
List Size = 4096, Threads = 64, error = 0, time (sec) =   0.0048, qsort_time =   0.0011
List Size = 4096, Threads = 256, error = 0, time (sec) =   0.0194, qsort_time =   0.0009
List Size = 4096, Threads = 1024, error = 0, time (sec) =   0.0727, qsort_time =   0.0008

List Size = 1048576, Threads = 1, error = 0, time (sec) =   0.1778, qsort_time =   0.1734
List Size = 1048576, Threads = 2, error = 0, time (sec) =   0.0922, qsort_time =   0.1720
List Size = 1048576, Threads = 4, error = 0, time (sec) =   0.0552, qsort_time =   0.1759
List Size = 1048576, Threads = 16, error = 0, time (sec) =   0.0238, qsort_time =   0.1775
List Size = 1048576, Threads = 64, error = 0, time (sec) =   0.0237, qsort_time =   0.1739
List Size = 1048576, Threads = 256, error = 0, time (sec) =   0.0328, qsort_time =   0.1742
List Size = 1048576, Threads = 1024, error = 0, time (sec) =   0.0889, qsort_time =   0.1767

List Size = 268435456, Threads = 1, error = 0, time (sec) =  63.0070, qsort_time =  62.6086
List Size = 268435456, Threads = 2, error = 0, time (sec) =  31.8949, qsort_time =  62.6108
List Size = 268435456, Threads = 4, error = 0, time (sec) =  16.2072, qsort_time =  62.5559
List Size = 268435456, Threads = 16, error = 0, time (sec) =   4.2093, qsort_time =  62.6065
List Size = 268435456, Threads = 64, error = 0, time (sec) =   1.9638, qsort_time =  62.5492
List Size = 268435456, Threads = 256, error = 0, time (sec) =   1.8151, qsort_time =  62.6168
List Size = 268435456, Threads = 1024, error = 0, time (sec) =   2.2318, qsort_time =  62.9990
```

We can see the improvement by comparing the execution time with the normal qsort algorithm.

For k=20, and q = 0, both take almost the same time:

```
List Size = 1048576, Threads = 1, error = 0, time (sec) =   0.1778, qsort_time =   0.1734
```

Whereas, as the number of threads increase, for k=20 and q=6, we can see a significant increase in speed:

```
List Size = 1048576, Threads = 64, error = 0, time (sec) =   0.0237, qsort_time =   0.1739
```

And, As q increases, the execution time reduces(only until a certain extent).

Similarly, for k=28, the difference is even more prominent. For k=28 and q=8 has the least execution time:

```
List Size = 268435456, Threads = 256, error = 0, time (sec) =   1.8151, qsort_time =  62.6168
```

The normal quicksort takes 62 seconds whereas the parallelized sort takes less than 2 seconds to complete.