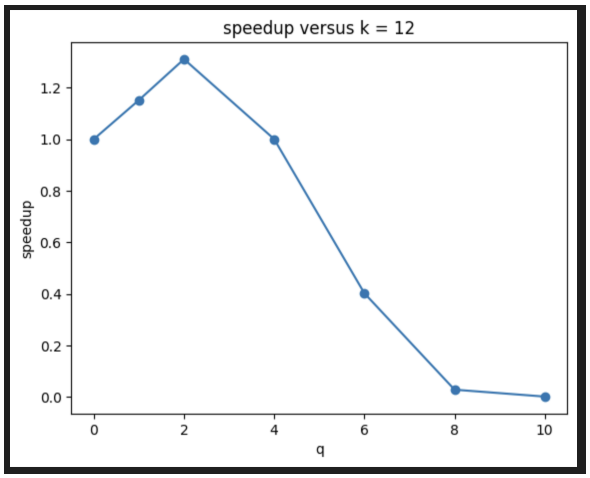
Chonglin Zhang
CSCE-735
DUE: 10-17-2023
833003072

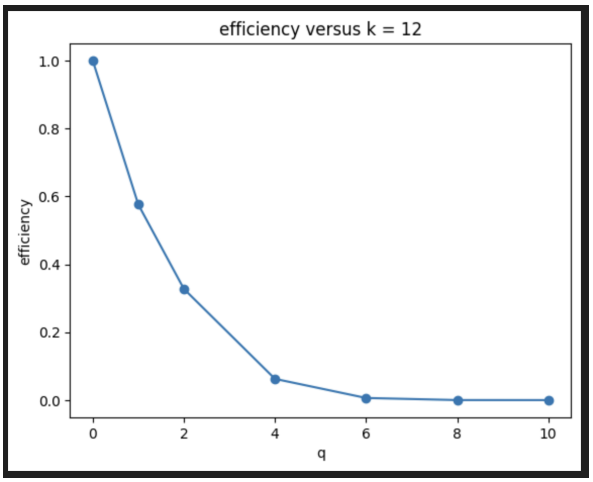
1.

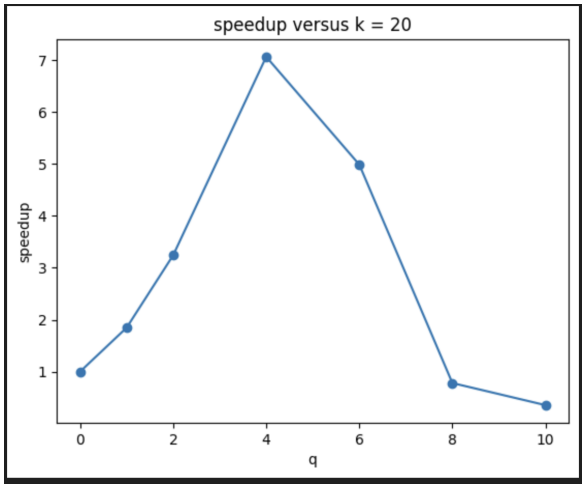
```
test for problem 1
List Size = 16, Threads = 2, error = 0, time (sec) =   0.0051, qsort_time =   0.0000
List Size = 16, Threads = 4, error = 0, time (sec) =   0.0059, qsort_time =   0.0000
List Size = 16, Threads = 8, error = 0, time (sec) =   0.0062, qsort_time =   0.0000
List Size = 1048576, Threads = 16, error = 0, time (sec) =   0.0327, qsort_time =   0.1697
List Size = 16777216, Threads = 256, error = 0, time (sec) =   0.5488, qsort_time =   3.3883
```

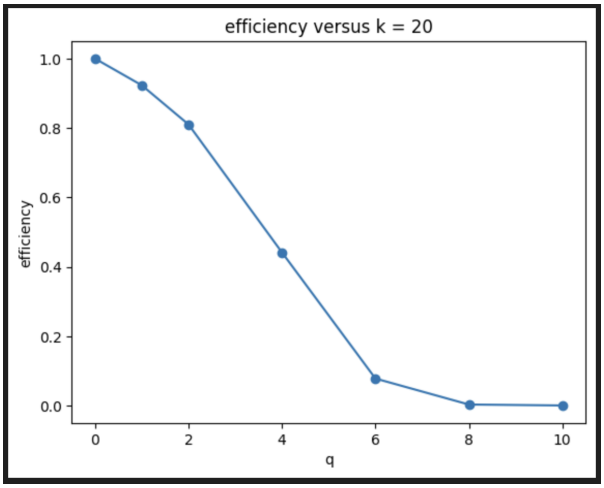
2.

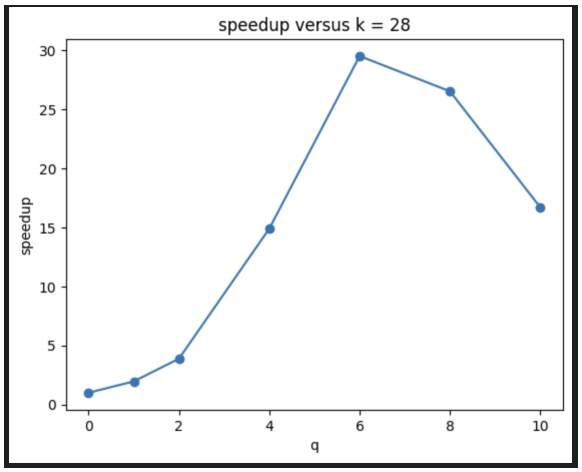
```
k = 12, q = 0,1,2,4,6,8,10
List Size = 4096, Threads = 1, error = 0, time (sec) =   0.0076, qsort_time =   0.0010
List Size = 4096, Threads = 2, error = 0, time (sec) =   0.0066, qsort_time =   0.0010
List Size = 4096, Threads = 4, error = 0, time (sec) =   0.0058, qsort_time =   0.0007
List Size = 4096, Threads = 16, error = 0, time (sec) =   0.0076, qsort_time =   0.0008
List Size = 4096, Threads = 64, error = 0, time (sec) =   0.0189, qsort_time =   0.0004
List Size = 4096, Threads = 256, error = 0, time (sec) =   0.2727, qsort_time =   0.0004
List Size = 4096, Threads = 1024, error = 0, time (sec) =   7.2213, qsort_time =   0.0004
k = 20, q = 0,1,2,4,6,8,10
List Size = 1048576, Threads = 1, error = 0, time (sec) =   0.1815, qsort_time =   0.1684
List Size = 1048576, Threads = 2, error = 0, time (sec) =   0.0983, qsort_time =   0.1693
List Size = 1048576, Threads = 4, error = 0, time (sec) =   0.0560, qsort_time =   0.1691
List Size = 1048576, Threads = 16, error = 0, time (sec) =   0.0257, qsort_time =   0.1691
List Size = 1048576, Threads = 64, error = 0, time (sec) =   0.0364, qsort_time =   0.1702
List Size = 1048576, Threads = 256, error = 0, time (sec) =   0.2324, qsort_time =   0.2942
List Size = 1048576, Threads = 1024, error = 0, time (sec) =   0.5099, qsort_time =   0.2615
k = 28, q = 0,1,2,4,6,8,10
List Size = 268435456, Threads = 1, error = 0, time (sec) =  61.9290, qsort_time =  61.8746
List Size = 268435456, Threads = 2, error = 0, time (sec) =  31.5582, qsort_time =  61.8389
List Size = 268435456, Threads = 4, error = 0, time (sec) =  15.9552, qsort_time =  61.9186
List Size = 268435456, Threads = 16, error = 0, time (sec) =   4.1596, qsort_time =  61.7967
List Size = 268435456, Threads = 64, error = 0, time (sec) =   2.0992, qsort_time =  62.2604
List Size = 268435456, Threads = 256, error = 0, time (sec) =   2.3356, qsort_time =  62.0624
List Size = 268435456, Threads = 1024, error = 0, time (sec) =   3.7091, qsort_time =  62.5115
```

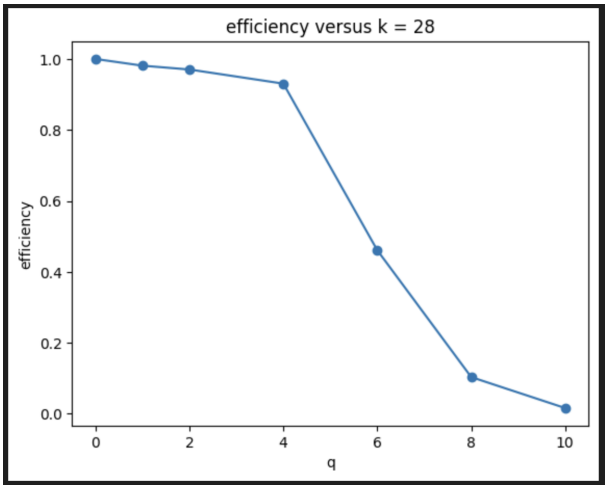












For these speedup graph, when there are more and more threads to omp program, we see the speedup increase as shown above graphs. It means more thread can handle large list size. However, when it got a certain number of threads, the speedup starts to decrease. This is because there are enough threads to handle the size of lists. For the efficiency graphs, we can see all of graphs are decreasing. As increasing threads for omp, it may cause some overhead which will lead efficiency decreasing.

3. Using export OMP_PLACES and export OMP_PROC_BIND

PLACES = threads, and BIND = master, close, spread
BIND = master, time (sec) = 66.6790
BIND = close,time (sec) = 2.1960
BIND = spread, time (sec) = 2.1620

PLACES = cores, and BIND = master, close, spread
BIND = master, time (sec) = 67.0245
BIND = close, time (sec) = 2.2024
BIND = spread,time (sec) = 2.1822

PLACES = sockets, and BIND = master, close, spread
BIND = master, time (sec) = 3.9570

BIND = close, time (sec) = 2.1771
BIND = spread, time (sec) = 2.1762

As these results, we can see most of time are around 2-4 second, but there are only two results are about 66-67 seconds when the OMP_PROC_BIND is master and OMP_PLACES are threads and cores. When the OMP_PROC_BIND is master, it means all threads will be assigned to the same location as the master thread. However, when OMP_PLACES is socket and OMP_PROC_BIND is master, we can see the time is around 4 second because sockets have a little bit different on threads and cores. Sockets can map each location to a unique socket (comprising one or more cores) in the local memory. Therefore, the socket may help the omp to reduce the time.