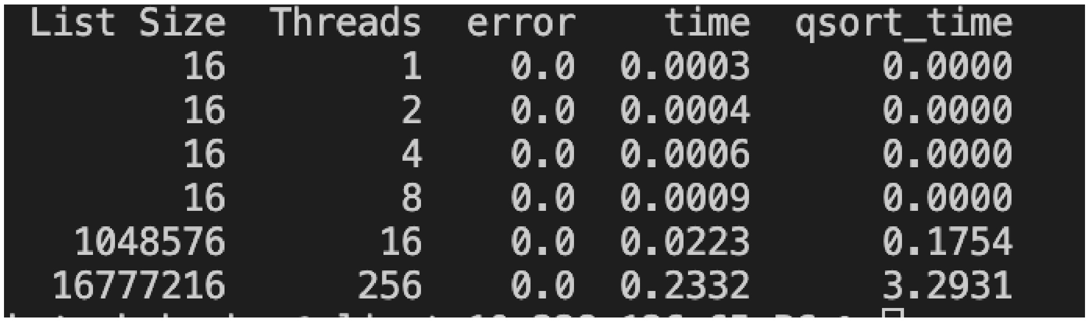
CSCE 735 Fall 2023 – HW2

Name: Ashutosh Chauhan
UIN: 232009024

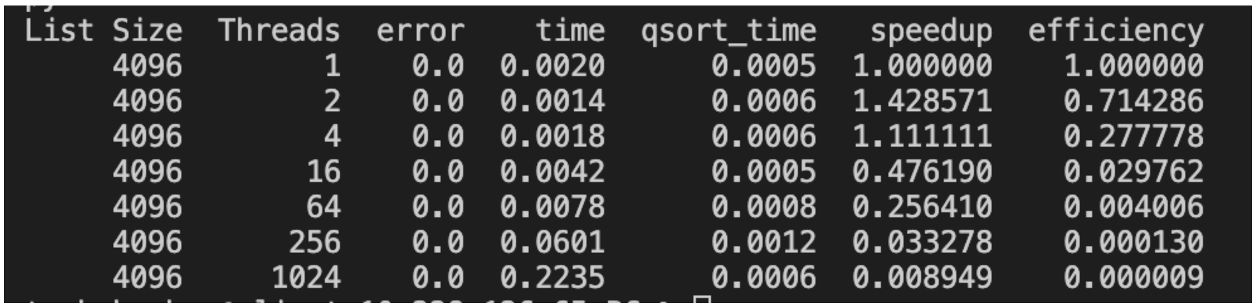
1. **(70 points) Revise the code to implement a thread-based parallel merge sort. The code should compile successfully and should report error=0 for the following instances:**
   ./sort_list.exe 4 1
   ./sort_list.exe 4 2
   ./sort_list.exe 4 3
   ./sort_list.exe 20 4
   ./sort_list.exe 24 8



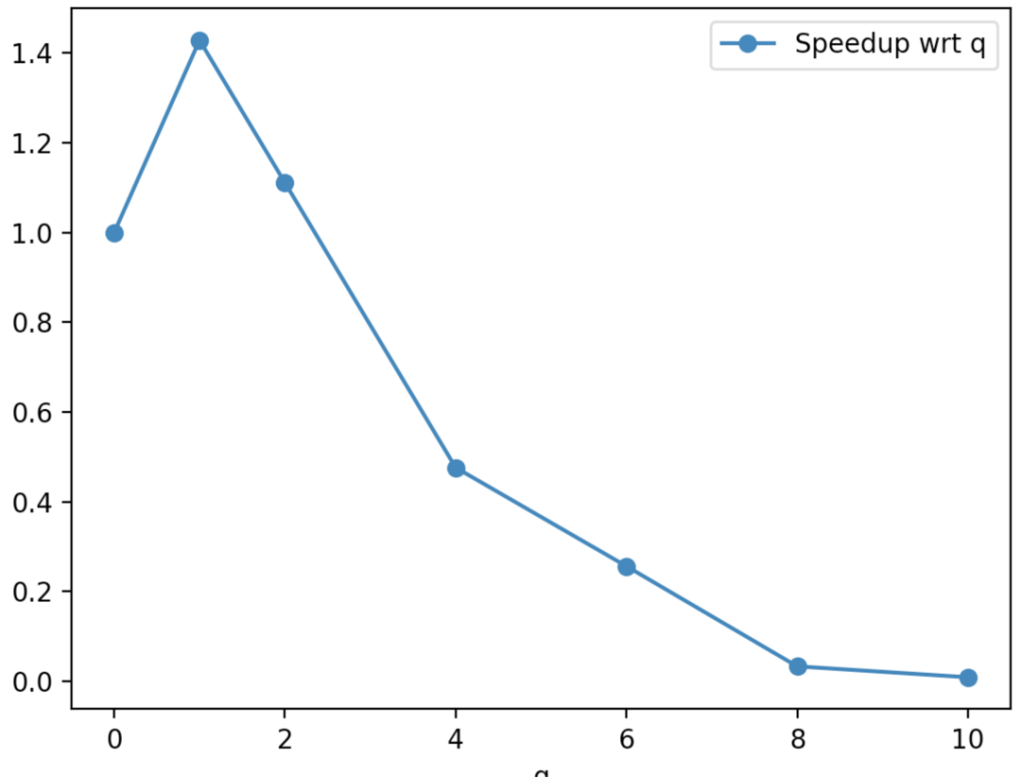
```
 List Size  Threads  error     time  qsort_time
        16        1    0.0  0.0003      0.0000
        16        2    0.0  0.0004      0.0000
        16        4    0.0  0.0006      0.0000
        16        8    0.0  0.0009      0.0000
   1048576       16    0.0  0.0223      0.1754
  16777216      256    0.0  0.2332      3.2931
```

2. **(20 points) Plot speedup and efficiency for all combinations of k and q chosen from the following sets: k = 12, 20, 28 ; q = 0, 1, 2, 4, 6, 8, 10. Comment on how the results of your experiments align with or diverge from your understanding of the expected behavior of the parallelized code.**

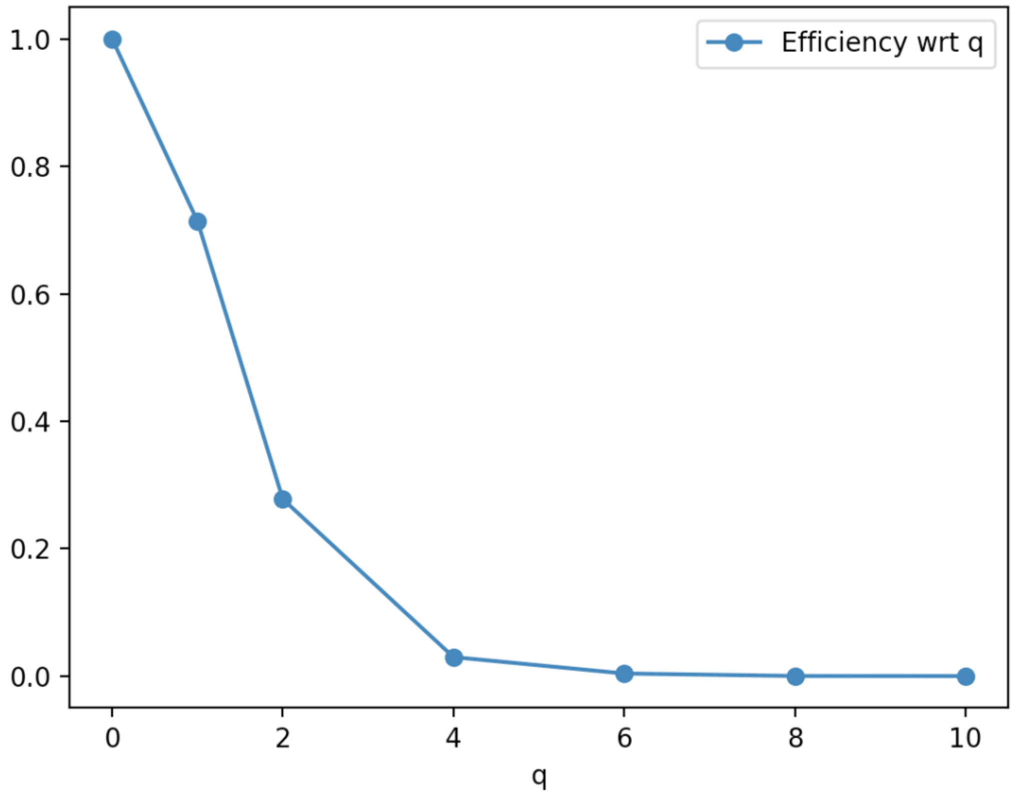
   <u>K = 12</u>



```
 List Size  Threads  error    time  qsort_time   speedup  efficiency
      4096        1    0.0  0.0020      0.0005  1.000000    1.000000
      4096        2    0.0  0.0014      0.0006  1.428571    0.714286
      4096        4    0.0  0.0018      0.0006  1.111111    0.277778
      4096       16    0.0  0.0042      0.0005  0.476190    0.029762
      4096       64    0.0  0.0078      0.0008  0.256410    0.004006
      4096      256    0.0  0.0601      0.0012  0.033278    0.000130
      4096     1024    0.0  0.2235      0.0006  0.008949    0.000009
```

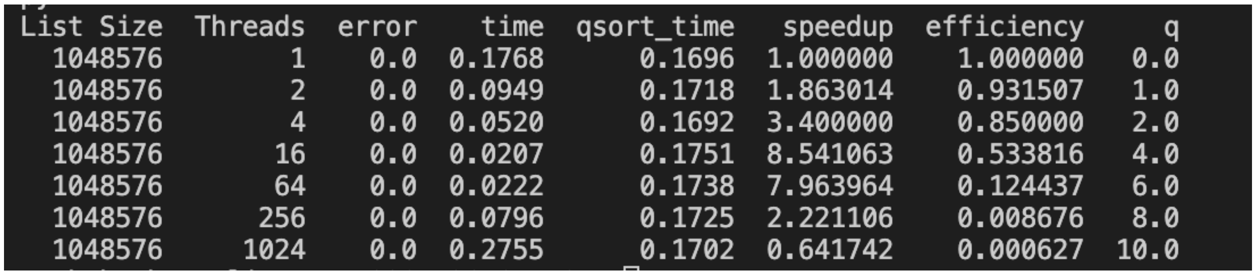


Speedup wrt q
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
8
10

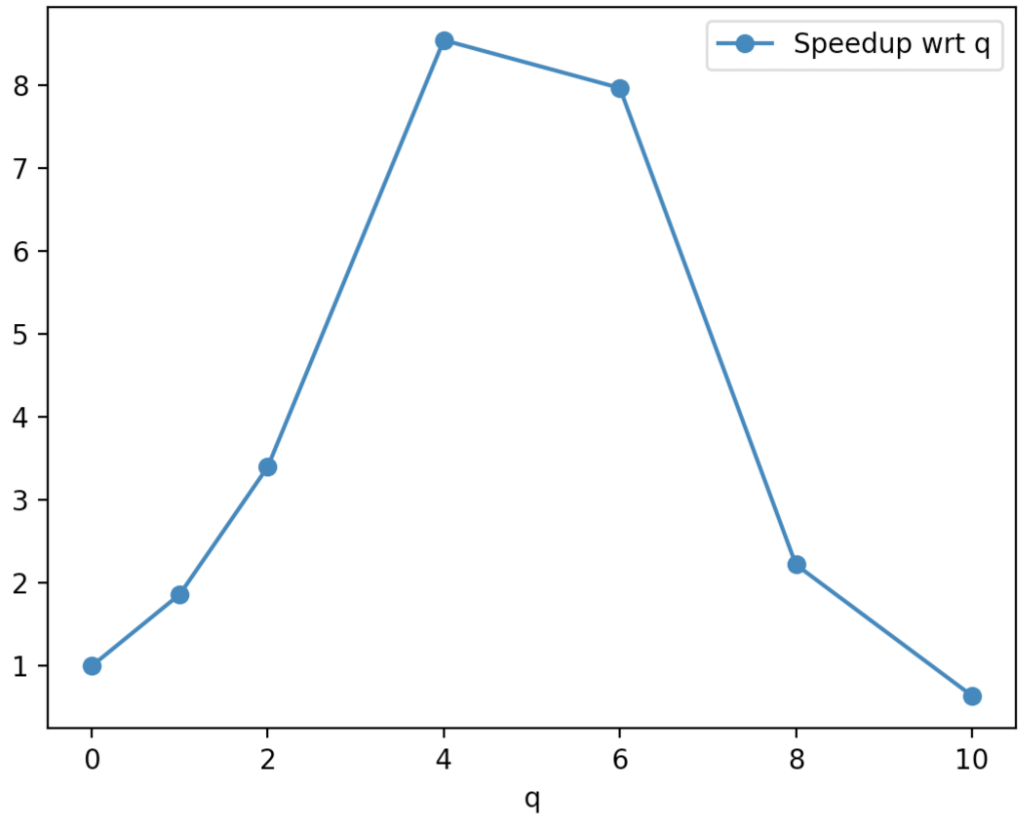


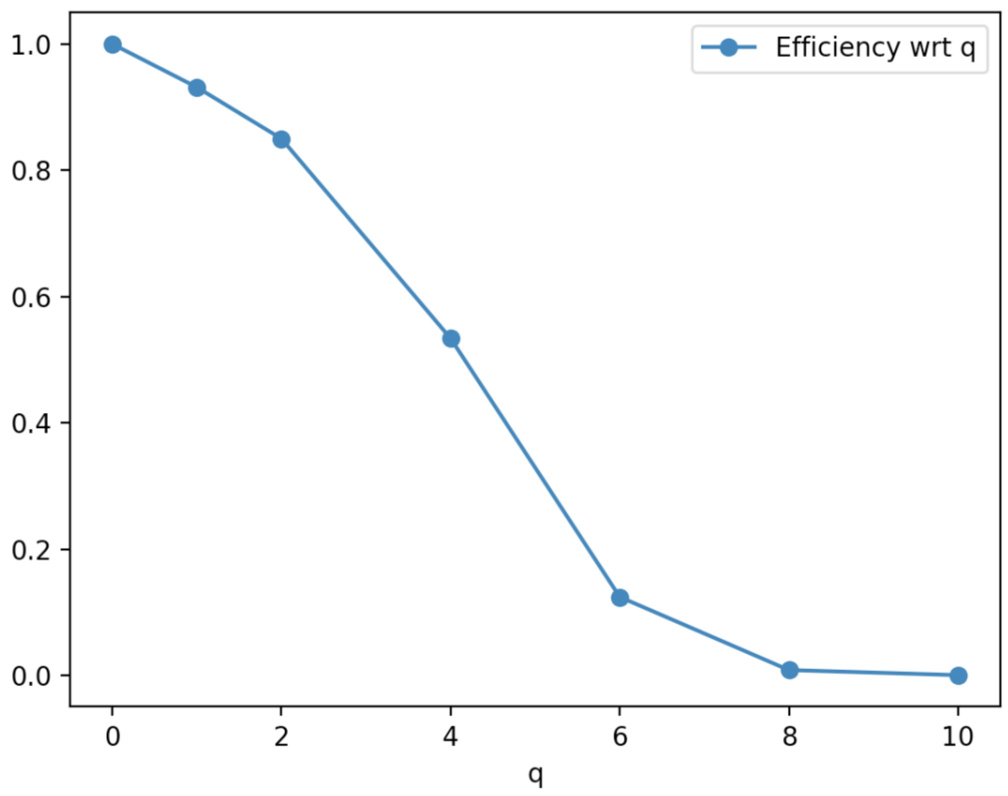
Efficiency wrt q
1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
8
10
q

K = 20

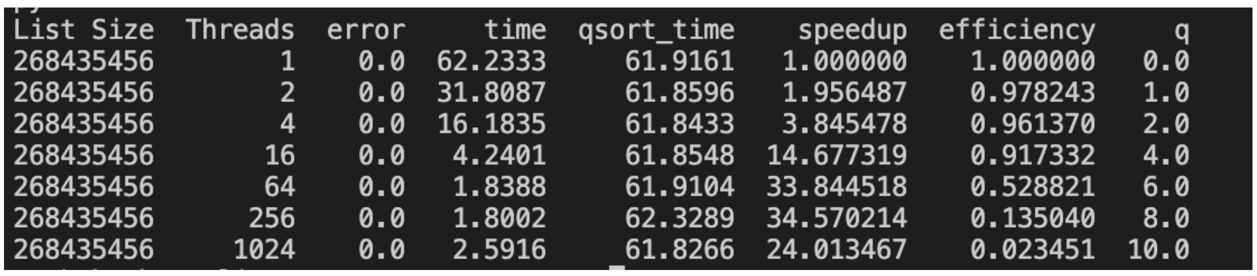


```
List Size  Threads  error    time  qsort_time   speedup  efficiency     q
  1048576        1    0.0  0.1768      0.1696  1.000000    1.000000   0.0
  1048576        2    0.0  0.0949      0.1718  1.863014    0.931507   1.0
  1048576        4    0.0  0.0520      0.1692  3.400000    0.850000   2.0
  1048576       16    0.0  0.0207      0.1751  8.541063    0.533816   4.0
  1048576       64    0.0  0.0222      0.1738  7.963964    0.124437   6.0
  1048576      256    0.0  0.0796      0.1725  2.221106    0.008676   8.0
  1048576     1024    0.0  0.2755      0.1702  0.641742    0.000627  10.0
```

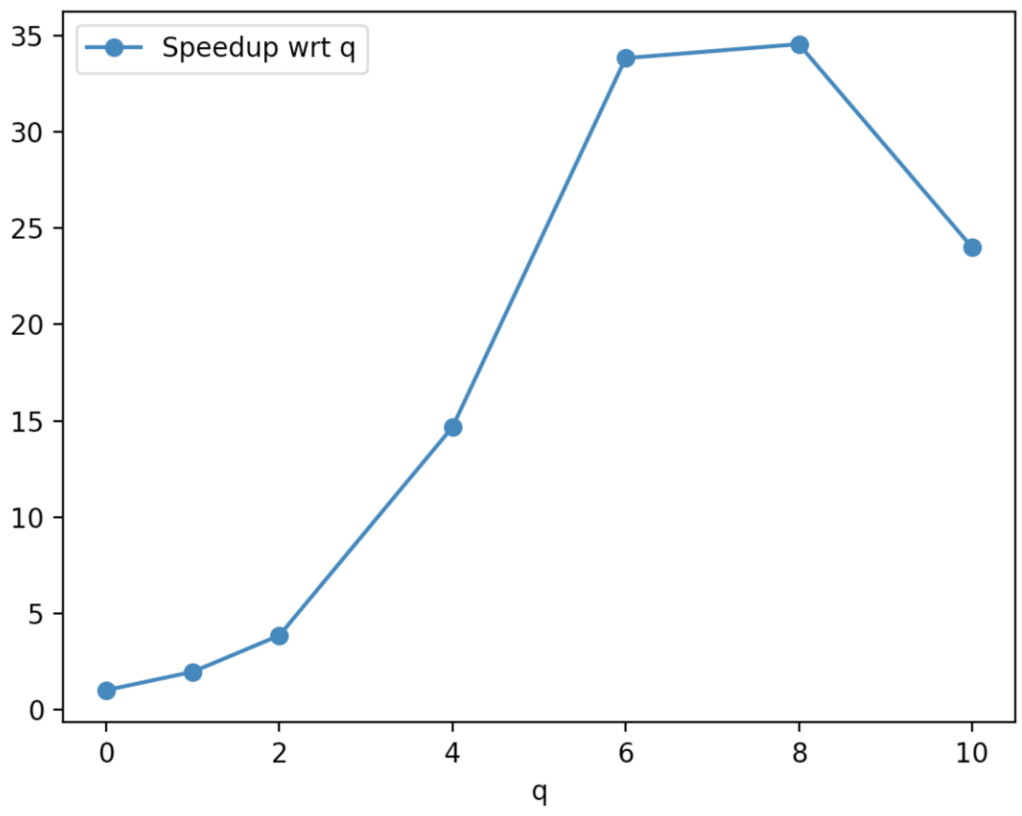


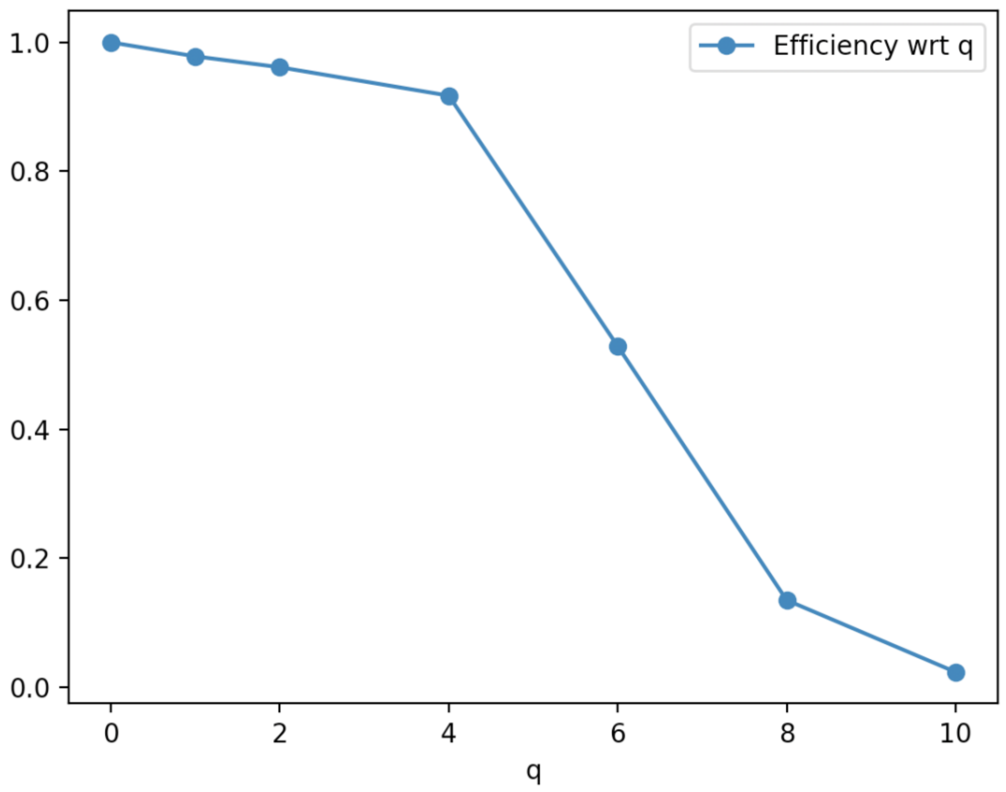


<u>K = 28</u>



```
List Size  Threads  error     time  qsort_time    speedup  efficiency     q
268435456        1    0.0  62.2333     61.9161   1.000000    1.000000   0.0
268435456        2    0.0  31.8087     61.8596   1.956487    0.978243   1.0
268435456        4    0.0  16.1835     61.8433   3.845478    0.961370   2.0
268435456       16    0.0   4.2401     61.8548  14.677319    0.917332   4.0
268435456       64    0.0   1.8388     61.9104  33.844518    0.528821   6.0
268435456      256    0.0   1.8002     62.3289  34.570214    0.135040   8.0
268435456     1024    0.0   2.5916     61.8266  24.013467    0.023451  10.0
```


Speedup wrt q
q
0
5
10
15
20
25
30
35
0
2
4
6
8
10


Efficiency wrt q
q
0.0
0.2
0.4
0.6
0.8
1.0
0
2
4
6
8
10

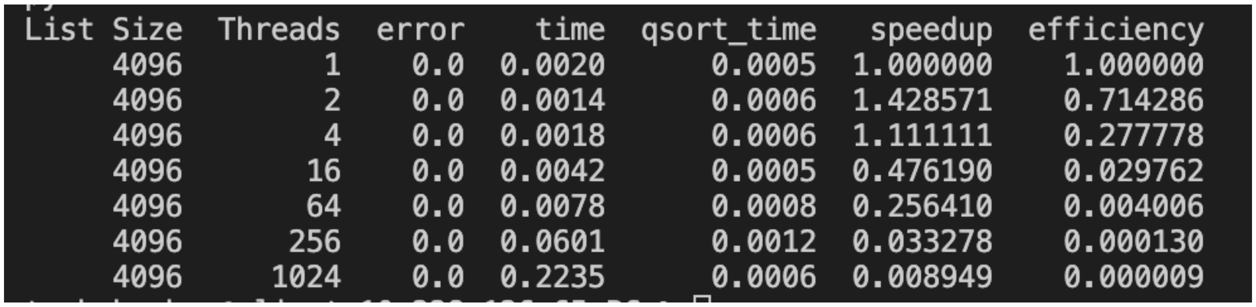
Comments:

We can observe that in all the above cases speedup increases until it reaches a threshold and then starts decreasing after a certain value of threads. This is because every thread requires some system resources like stack, memory etc. for context switching. As we increase the number of threads the overhead associated with managing and switching between threads also increases. This outweighs the benefits that we get because of multiple threads after a certain value. Hence, we see this behavior.

Also, we can observe that efficiency keeps on decreasing with an increase in the number of threads in all the above cases. This can be attributed to the growing number of underutilized threads. With the increase in the number of threads the amount of work shared by each thread is less compared to the overhead it takes to manage multiple threads.

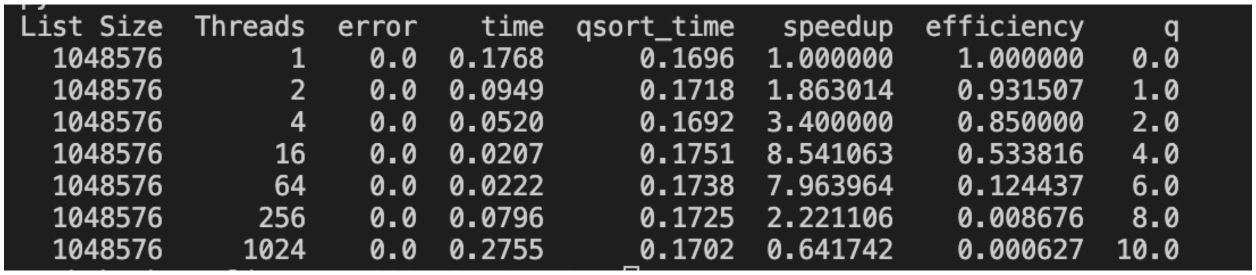
3. **(10 points) Your code should demonstrate speedup when sorting lists of appropriate sizes. Determine two values of k for which your code shows speedup as q is varied. Present the timing results for your code along with speedup and efficiency obtained to convince the reader that you have a well-designed parallel merge sort. You may use results from experiments in previous problems or identify new values k and q to illustrate how well your code has been parallelized.**

K = 12



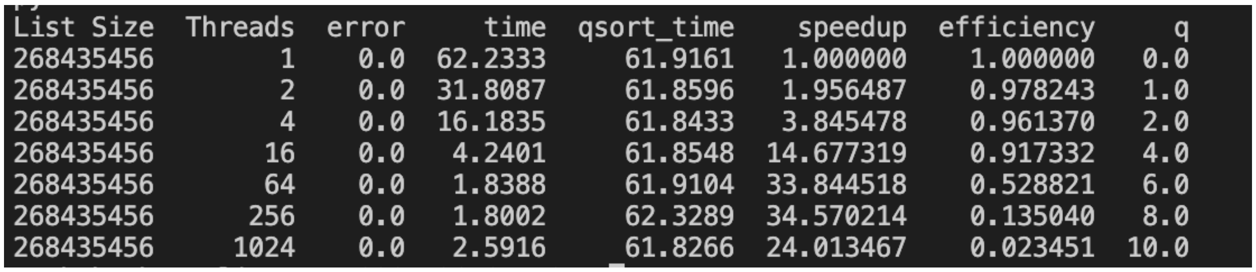
```
 List Size  Threads  error    time  qsort_time   speedup  efficiency
      4096        1    0.0  0.0020      0.0005  1.000000    1.000000
      4096        2    0.0  0.0014      0.0006  1.428571    0.714286
      4096        4    0.0  0.0018      0.0006  1.111111    0.277778
      4096       16    0.0  0.0042      0.0005  0.476190    0.029762
      4096       64    0.0  0.0078      0.0008  0.256410    0.004006
      4096      256    0.0  0.0601      0.0012  0.033278    0.000130
      4096     1024    0.0  0.2235      0.0006  0.008949    0.000009
```

K = 20



```
List Size  Threads  error    time  qsort_time   speedup  efficiency     q
  1048576        1    0.0  0.1768      0.1696  1.000000    1.000000   0.0
  1048576        2    0.0  0.0949      0.1718  1.863014    0.931507   1.0
  1048576        4    0.0  0.0520      0.1692  3.400000    0.850000   2.0
  1048576       16    0.0  0.0207      0.1751  8.541063    0.533816   4.0
  1048576       64    0.0  0.0222      0.1738  7.963964    0.124437   6.0
  1048576      256    0.0  0.0796      0.1725  2.221106    0.008676   8.0
  1048576     1024    0.0  0.2755      0.1702  0.641742    0.000627  10.0
```

K = 28


```
List Size  Threads  error     time  qsort_time    speedup  efficiency     q
268435456        1    0.0  62.2333     61.9161   1.000000    1.000000   0.0
268435456        2    0.0  31.8087     61.8596   1.956487    0.978243   1.0
268435456        4    0.0  16.1835     61.8433   3.845478    0.961370   2.0
268435456       16    0.0   4.2401     61.8548  14.677319    0.917332   4.0
268435456       64    0.0   1.8388     61.9104  33.844518    0.528821   6.0
268435456      256    0.0   1.8002     62.3289  34.570214    0.135040   8.0
268435456     1024    0.0   2.5916     61.8266  24.013467    0.023451  10.0
```

From the above data, we can observe that the time required to sort the list reduces with an increase in number of threads until a certain limit. This is evident from all the above three tables. In k = 20 table we can we the time needed to sort an array of size 1048576 reduced from 0.1768 sec to 0.0207 sec when the number of threads were increased to 16. Similarly, in k = 28 table the time was reduced from 62.2333 sec to 1.8002 when the number of threads were increased to 256 for the same input. This can prove that our parallel merge sort algorithm is working fine.
With a further increase in the number of threads, the time starts to increase which is because of the overhead associated with managing and switching between threads also increases. This outweighs the benefits that we get because of multiple threads after a certain value.