Jessica Salguero, UIN: 937005006
CSCE 735-701, Summer 2025

HW2: Parallel Merge Sort Using Threads

Question 1

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

SBATCH Script

```
#!/bin/bash
##ENVIRONMENT SETTINGS; CHANGE WITH CAUTION
#SBATCH --export=NONE                #Do not propagate environment
#SBATCH --get-user-env=L             #Replicate login environment
#
##NECESSARY JOB SPECIFICATIONS
#SBATCH --job-name=CSCE735_HW2       #Set the job name to "JobExample2"
#SBATCH --time=0:30:00               #Set the wall clock limit to 6hr and
#SBATCH --nodes=1                    #Request 1 node
#SBATCH --ntasks-per-node=48         #Request 8 tasks/cores per node
#SBATCH --mem=8G                     #Request 8GB per node
#SBATCH --output=output.HW2.Q1%j     #Send stdout/err to "output.[jobID]
#
##OPTIONAL JOB SPECIFICATIONS
##SBATCH --mail-type=ALL             #Send email on all job events
##SBATCH --mail-user=email_address   #Send all emails to email_address
#
##First Executable Line
#
module load intel          # load Intel software stack
#
icx -o sort_list.exe sort_list.c -lpthread
#
./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
##
```

SBATCH Output

```
List Size = 16, Threads = 2, error = 0, time (sec) =   0.0008, qsort_time =   0.0000
List Size = 16, Threads = 4, error = 0, time (sec) =   0.0008, qsort_time =   0.0000
List Size = 16, Threads = 8, error = 0, time (sec) =   0.0010, qsort_time =   0.0000
List Size = 1048576, Threads = 16, error = 0, time (sec) =   0.0147, qsort_time =   0.1494
List Size = 16777216, Threads = 256, error = 0, time (sec) =   0.1599, qsort_time =   2.9233
```

Jessica Salguero, UIN: 937005006
CSCE 735-701, Summer 2025

Question 2

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.

SBATCH Script

```
##First Executable Line
#
module load intel          # load Intel software stack
#
icx -o sort_list.exe sort_list.c -lpthread
#
echo "k = 12, q = 0, 1, 2, 4, 6, 8, 10"
./sort_list.exe 12 0
./sort_list.exe 12 1
./sort_list.exe 12 2
./sort_list.exe 12 4
./sort_list.exe 12 6
./sort_list.exe 12 8
./sort_list.exe 12 10
echo ""
echo "k = 20, q = 0, 1, 2, 4, 6, 8, 10"
./sort_list.exe 20 0
./sort_list.exe 20 1
./sort_list.exe 20 2
./sort_list.exe 20 4
./sort_list.exe 20 6
./sort_list.exe 20 8
./sort_list.exe 20 10
echo ""
echo "k = 28, q = 0, 1, 2, 4, 6, 8, 10"
./sort_list.exe 28 0
./sort_list.exe 28 1
./sort_list.exe 28 2
./sort_list.exe 28 4
./sort_list.exe 28 6
./sort_list.exe 28 8
./sort_list.exe 28 10
##
```

SBATCH Output

```
k = 12, q = 0, 1, 2, 4, 6, 8, 10
List Size = 4096, Threads = 1, error = 0, time (sec) =   0.0011, qsort_time =   0.0006
List Size = 4096, Threads = 2, error = 0, time (sec) =   0.0020, qsort_time =   0.0004
List Size = 4096, Threads = 4, error = 0, time (sec) =   0.0012, qsort_time =   0.0007
List Size = 4096, Threads = 16, error = 0, time (sec) =   0.0022, qsort_time =   0.0004
List Size = 4096, Threads = 64, error = 0, time (sec) =   0.0036, qsort_time =   0.0004
List Size = 4096, Threads = 256, error = 0, time (sec) =   0.0128, qsort_time =   0.0004
List Size = 4096, Threads = 1024, error = 0, time (sec) =   0.0551, qsort_time =   0.0004

k = 20, q = 0, 1, 2, 4, 6, 8, 10
List Size = 1048576, Threads = 1, error = 0, time (sec) =   0.1491, qsort_time =   0.1484
List Size = 1048576, Threads = 2, error = 0, time (sec) =   0.0794, qsort_time =   0.1491
List Size = 1048576, Threads = 4, error = 0, time (sec) =   0.0432, qsort_time =   0.1490
List Size = 1048576, Threads = 16, error = 0, time (sec) =   0.0145, qsort_time =   0.1488
List Size = 1048576, Threads = 64, error = 0, time (sec) =   0.0109, qsort_time =   0.1492
List Size = 1048576, Threads = 256, error = 0, time (sec) =   0.0245, qsort_time =   0.1496
List Size = 1048576, Threads = 1024, error = 0, time (sec) =   0.0694, qsort_time =   0.1495

k = 28, q = 0, 1, 2, 4, 6, 8, 10
List Size = 268435456, Threads = 1, error = 0, time (sec) =  53.3722, qsort_time =  53.3318
List Size = 268435456, Threads = 2, error = 0, time (sec) =  27.6020, qsort_time =  53.5338
List Size = 268435456, Threads = 4, error = 0, time (sec) =  14.2830, qsort_time =  53.3465
List Size = 268435456, Threads = 16, error = 0, time (sec) =   3.8791, qsort_time =  53.7885
List Size = 268435456, Threads = 64, error = 0, time (sec) =   1.8633, qsort_time =  53.5314
List Size = 268435456, Threads = 256, error = 0, time (sec) =   1.8518, qsort_time =  53.3507
List Size = 268435456, Threads = 1024, error = 0, time (sec) =   2.1220, qsort_time =  53.5300
```

Jessica Salguero, UIN: 937005006
CSCE 735-701, Summer 2025

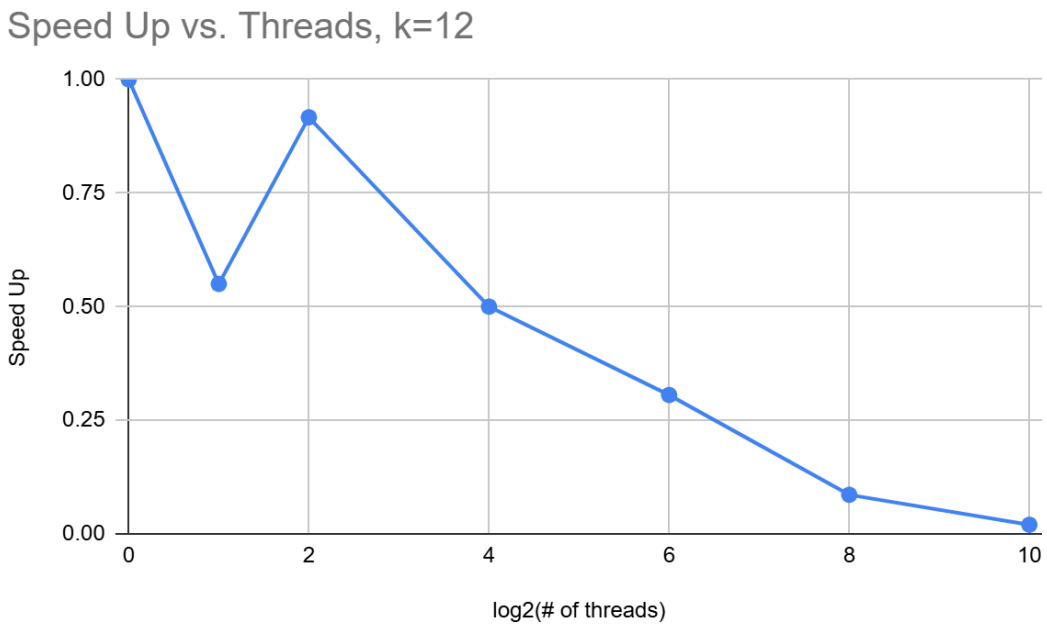
Question 2 Table

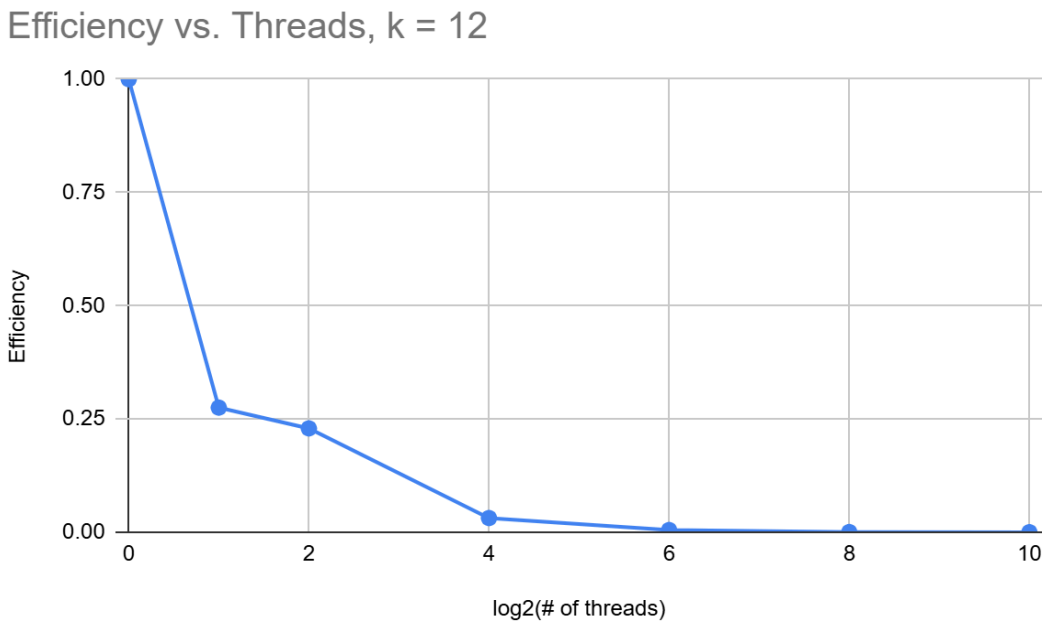
k	q	log2(# of threads)	Execution Time	Speed Up	Efficiency
12	1	0	0.0011	1	1
12	2	1	0.002	0.55	0.275
12	4	2	0.0012	0.9166666667	0.2291666667
12	16	4	0.0022	0.5	0.03125
12	64	6	0.0036	0.3055555556	0.004774305556
12	256	8	0.0128	0.0859375	0.000335693394
12	1024	10	0.0551	0.01996370236	0.000019495803 09
20	1	0	0.1491	1	1
20	2	1	0.0794	1.877833753	0.9389168766
20	4	2	0.0432	3.451388889	0.8628472222
20	16	4	0.0145	10.28275862	0.6426724138
20	64	6	0.0109	13.67889908	0.2137327982
20	256	8	0.0245	6.085714286	0.02377232143
20	1024	10	0.0694	2.148414986	0.002098061509
28	1	0	53.3722	1	1
28	2	1	27.602	1.933635244	0.9668176219
28	4	2	14.283	3.736763985	0.9341909963
28	16	4	3.8791	13.75891315	0.8599320719
28	64	6	1.8633	28.64391134	0.4475611147
28	256	8	1.8518	28.82179501	0.1125851368
28	1024	10	2.122	25.15183789	0.02456234169

Jessica Salguero, UIN: 937005006
CSCE 735-701, Summer 2025

K = 12

For k = 12, the speedup and efficiency tended to decrease as the number of threads increased. This behavior is expected because the size of the list is only 4096 items, which may seem like a big number but the overhead of managing a large number of threads and splitting work effectively ends up being a large time burden. Setting up parallel computing for parallel computation is less efficient than just using one thread to sort the list.

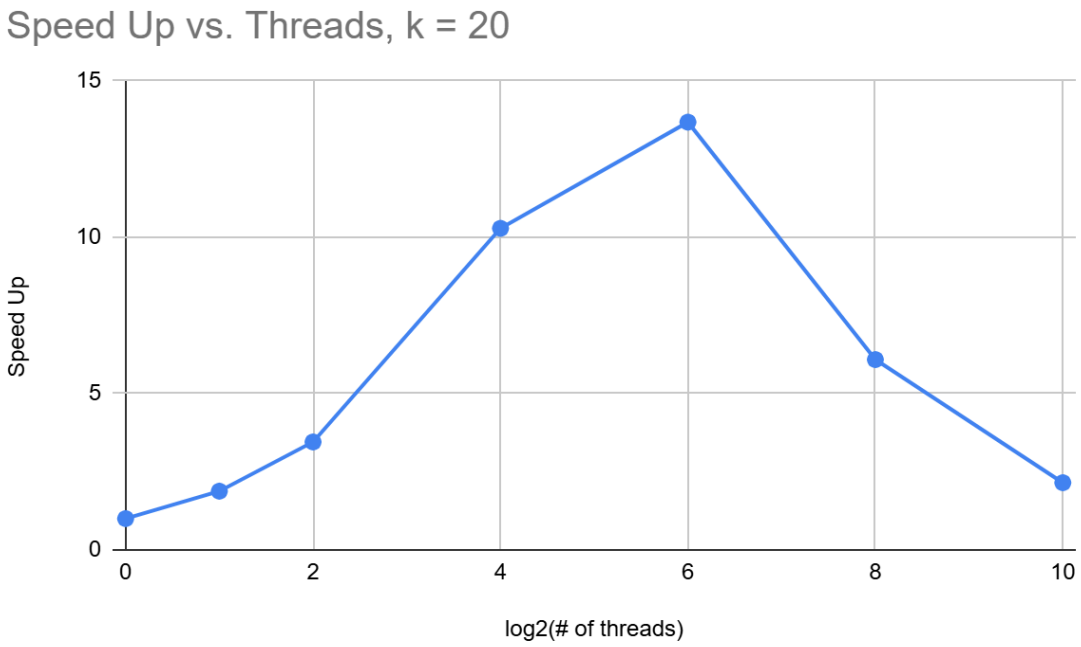


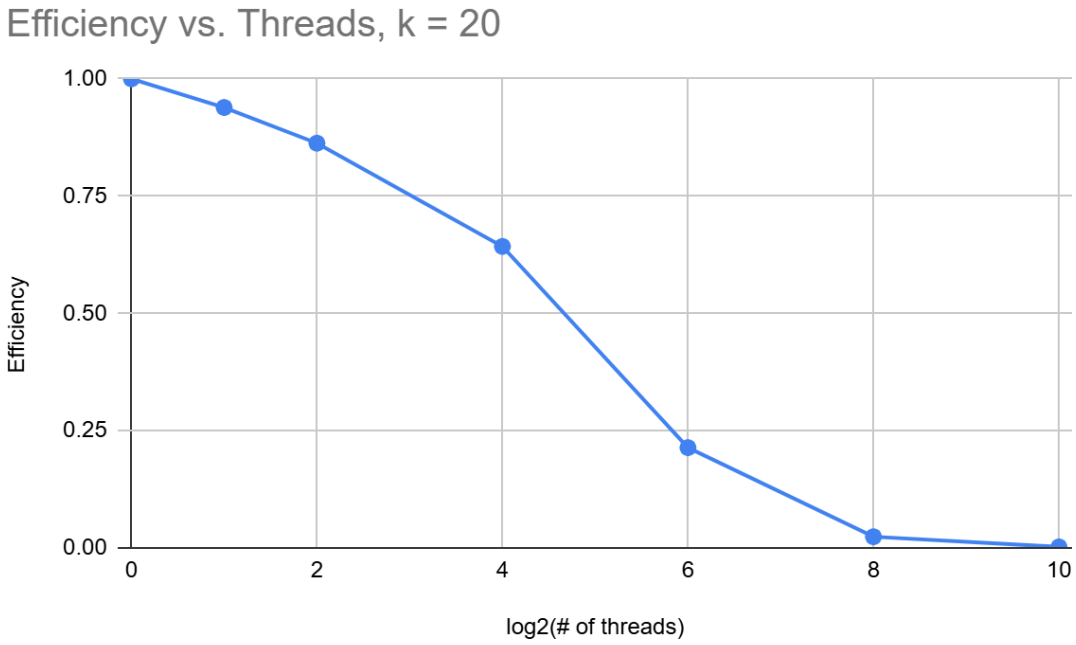


Jessica Salguero, UIN: 937005006
CSCE 735-701, Summer 2025

K = 20

For k = 20, there is some improvement seen when using parallel computing for sorting the list. Speed up increases until 64 threads, but higher amounts of threads begin to decrease speed up again. This is expected because the larger list size of over a million items is a lot for only one thread to handle, so the overhead of managing multiple threads is worthwhile. However, after a certain amount of threads (64), the overhead of managing threads ends up being a hindrance to performance.

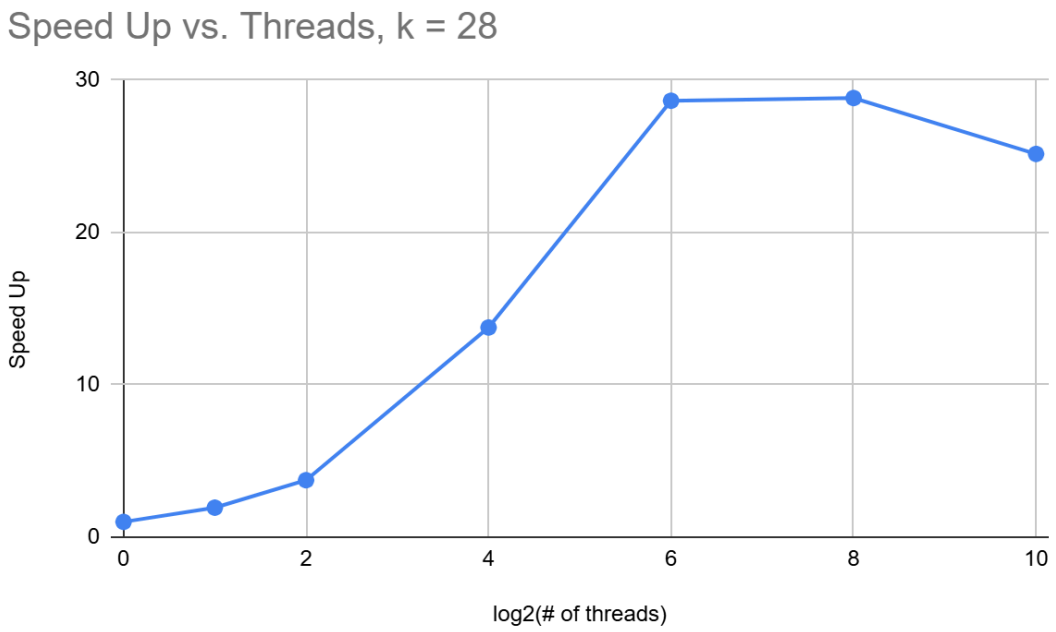


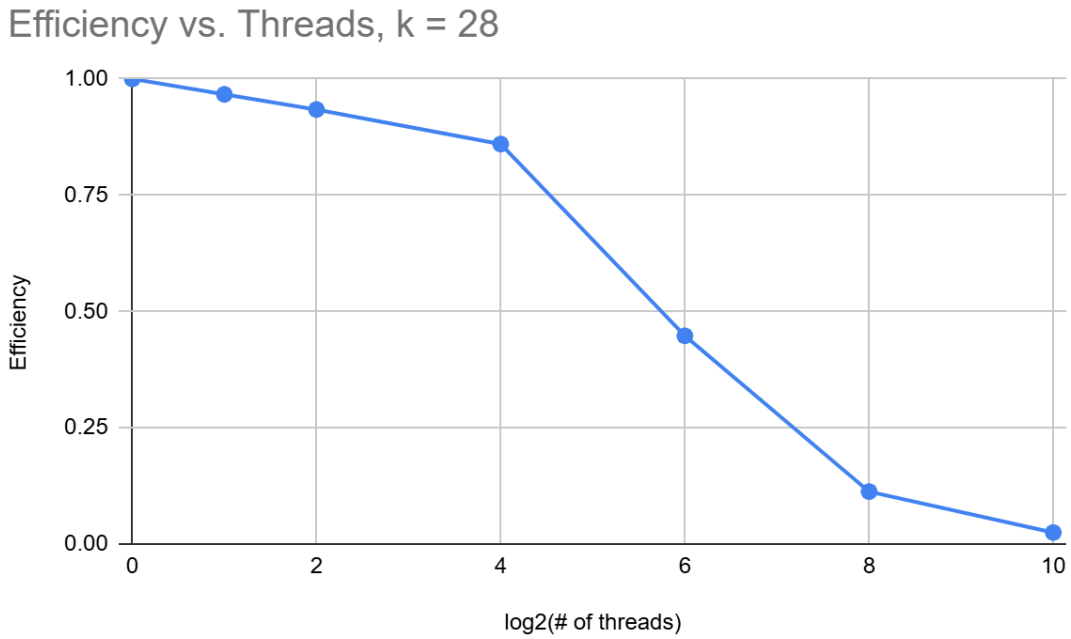


Jessica Salguero, UIN: 937005006
CSCE 735-701, Summer 2025

K = 28

For k = 28, the speed up increases drastically & efficiency decreases more gradually when using more than one thread. This is as expected due to the extremely large list size of over 200 million items. This scenario benefited greatly from using parallel threads, and performance improved a lot. Speed up only began to decrease again at 1024 threads, suggesting that 256 was the ideal number of threads to be used for this experiment. Overall, the parallelized code works better with larger list sizes.





Jessica Salguero, UIN: 937005006
CSCE 735-701, Summer 2025

Question 3

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.

SBATCH Script

```
##First Executable Line
#
module load intel        # load Intel software stack
#
icx -o sort_list.exe sort_list.c -lpthread
#
echo "k = 24, q = 0, 1, 2, 4, 6, 8, 10"
./sort_list.exe 24 0
./sort_list.exe 24 1
./sort_list.exe 24 2
./sort_list.exe 24 4
./sort_list.exe 24 6
./sort_list.exe 24 8
./sort_list.exe 24 10
echo ""
echo "k = 28, q = 0, 1, 2, 4, 6, 8, 10"
./sort_list.exe 28 0
./sort_list.exe 28 1
./sort_list.exe 28 2
./sort_list.exe 28 4
./sort_list.exe 28 6
./sort_list.exe 28 8
./sort_list.exe 28 10
```

SBATCH Output

```
k = 24, q = 0, 1, 2, 4, 6, 8, 10
List Size = 16777216, Threads = 1, error = 0, time (sec) =   2.8461, qsort_time =   2.8442
List Size = 16777216, Threads = 2, error = 0, time (sec) =   1.4891, qsort_time =   2.8713
List Size = 16777216, Threads = 4, error = 0, time (sec) =   0.7727, qsort_time =   2.8612
List Size = 16777216, Threads = 16, error = 0, time (sec) =   0.2177, qsort_time =   2.8664
List Size = 16777216, Threads = 64, error = 0, time (sec) =   0.1291, qsort_time =   2.8429
List Size = 16777216, Threads = 256, error = 0, time (sec) =   0.1654, qsort_time =   2.8454
List Size = 16777216, Threads = 1024, error = 0, time (sec) =   0.1792, qsort_time =   2.8703

k = 28, q = 0, 1, 2, 4, 6, 8, 10
List Size = 268435456, Threads = 1, error = 0, time (sec) =  53.3620, qsort_time =  53.3772
List Size = 268435456, Threads = 2, error = 0, time (sec) =  27.5988, qsort_time =  53.3466
List Size = 268435456, Threads = 4, error = 0, time (sec) =  14.2894, qsort_time =  53.3821
List Size = 268435456, Threads = 16, error = 0, time (sec) =   3.8876, qsort_time =  53.4824
List Size = 268435456, Threads = 64, error = 0, time (sec) =   1.8032, qsort_time =  53.4403
List Size = 268435456, Threads = 256, error = 0, time (sec) =   1.8432, qsort_time =  53.5967
List Size = 268435456, Threads = 1024, error = 0, time (sec) =   2.1084, qsort_time =  53.5038
```

Jessica Salguero, UIN: 937005006
CSCE 735-701, Summer 2025

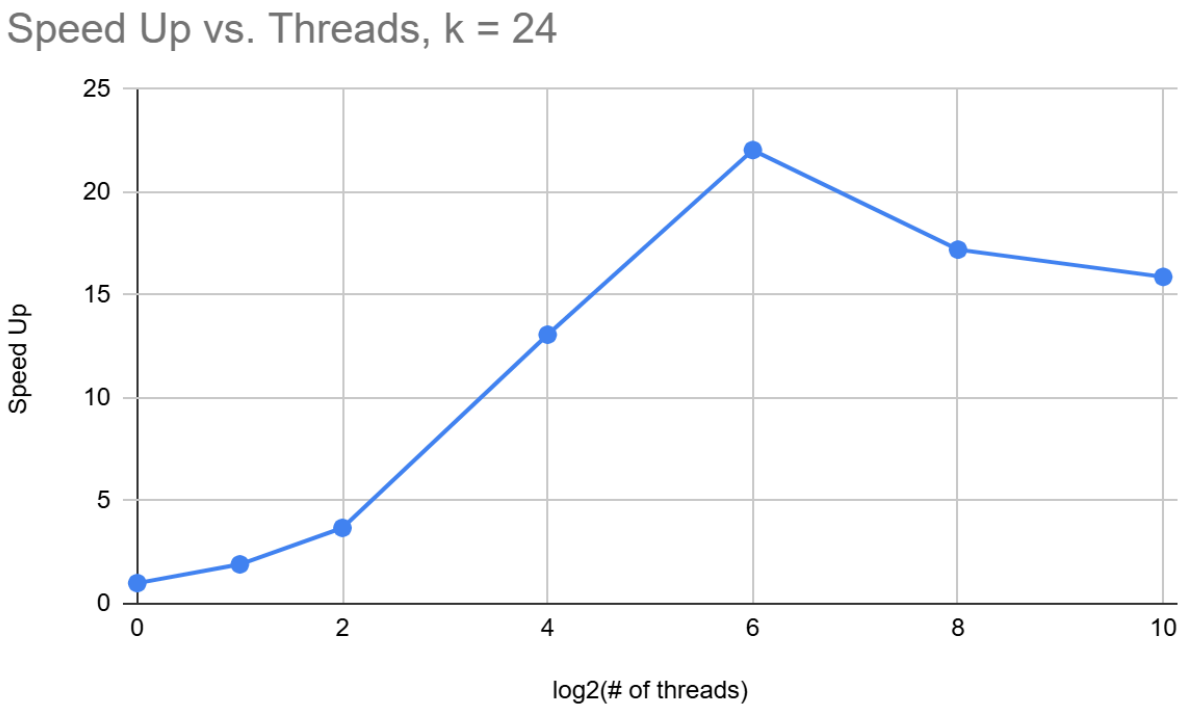
Question 3 Table

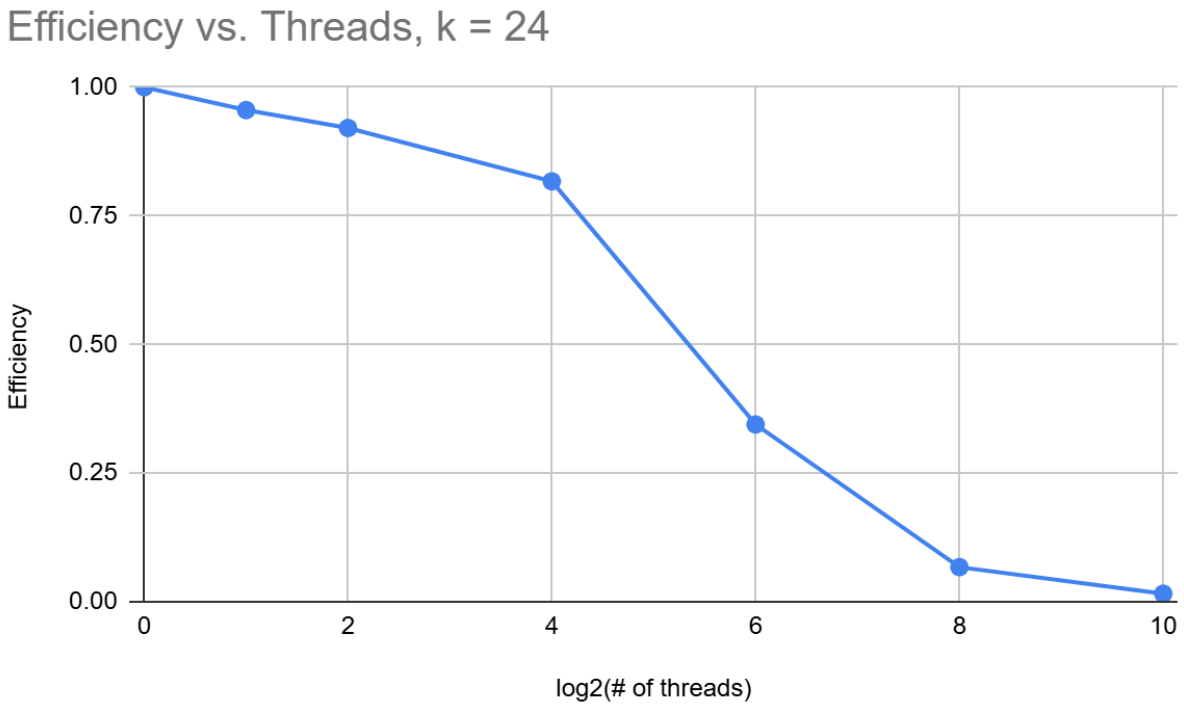
k	q	log2(# of threads)	Execution Time	Speed Up	Efficiency
24	1	0	2.8461	1	1
24	2	1	1.4891	1.911288698	0.9556443489
24	4	2	0.7727	3.683318235	0.9208295587
24	16	4	0.2177	13.07349564	0.8170934773
24	64	6	0.1291	22.04570101	0.3444640782
24	256	8	0.1654	17.20737606	0.06721631273
24	1024	10	0.1792	15.88225446	0.01551001413
28	1	0	53.362	1	1
28	2	1	27.5988	1.933489862	0.9667449309
28	4	2	14.2894	3.734376531	0.9335941327
28	16	4	3.8876	13.7262064	0.8578879
28	64	6	1.8032	29.59294587	0.4623897793
28	256	8	1.8432	28.95073785	0.1130888197
28	1024	10	2.1084	25.30923923	0.02471605394

For question 3, I wanted to compare the results for k = 24 and k =28. For k = 24, the highest speed up was approximately 22x and seen with 64 threads. Although performance began to decrease with additional threads, it still showed vast improvement when compared to sorting this large list with only one singular thread. For k = 28, the highest performance improvement was also seen with 64 threads, but the improvement was almost 30x more than when it was sorted with one singular thread. Overall, we can conclude that the parallelized code does increase speed up as q varies; with these experimental runs, improvement is seen up to 64 threads, but larger amounts of threads are too high and the overhead of managing threads begins to hinder performance.

Jessica Salguero, UIN: 937005006
CSCE 735-701, Summer 2025

K = 24





Jessica Salguero, UIN: 937005006
CSCE 735-701, Summer 2025

K = 28