

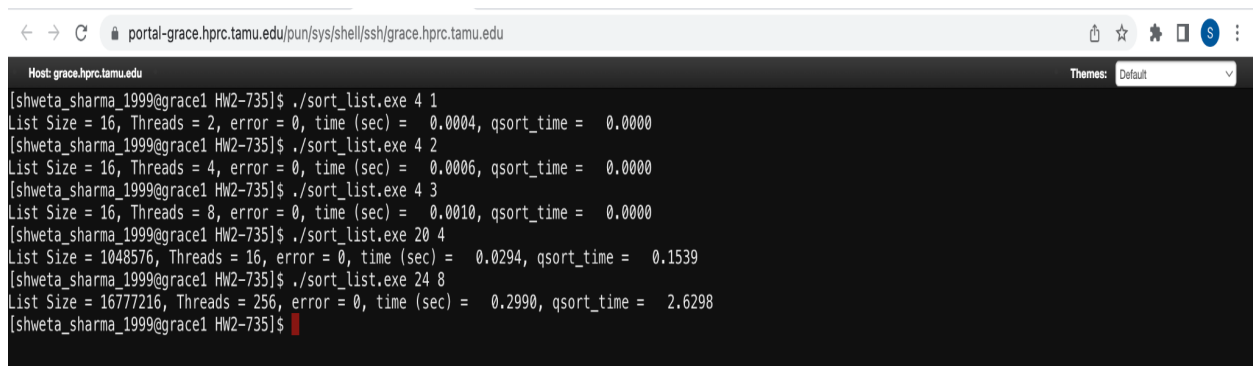
CSCE 735 Spring 2023
HW 2: Parallel Merge Sort Using Threads

Name: Shweta Sharma
UIN: 433003780

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
```

Ans: I have revised the code to execute the merge sort via multithreading in parallel. Attaching the screenshot here of the output of the code on Grace portal.



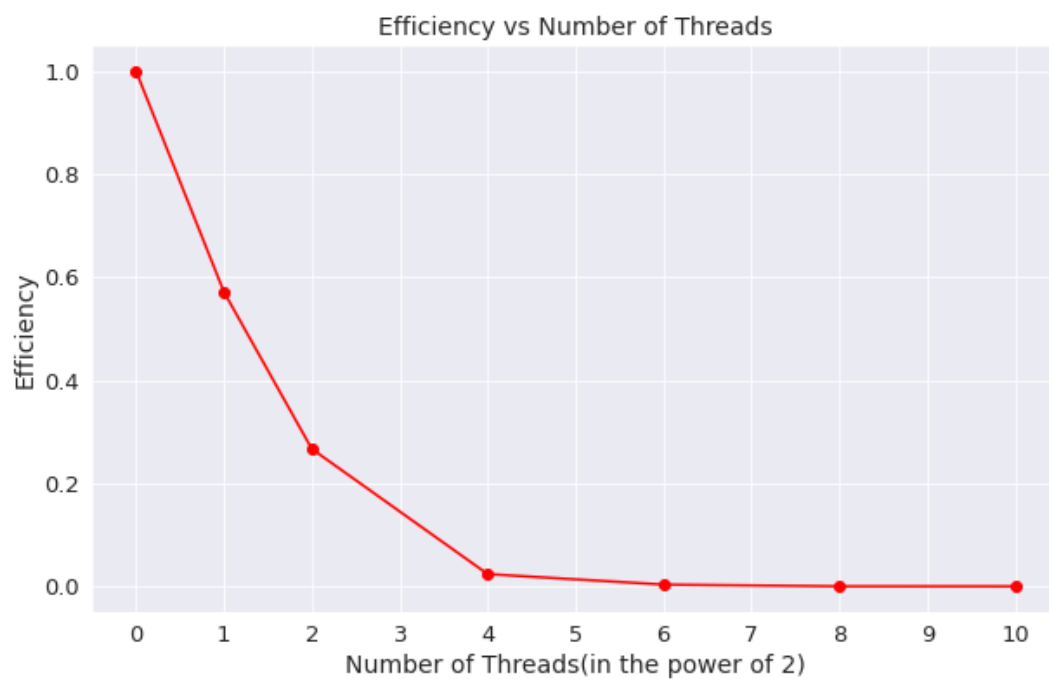
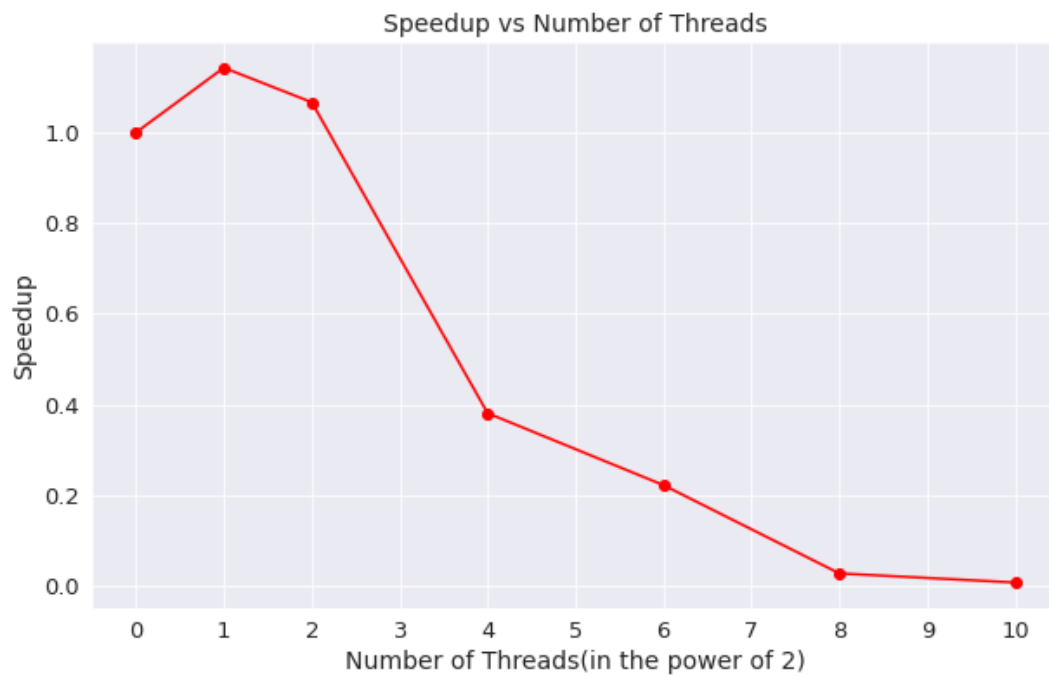
```
Host: grace.hprc.tamu.edu  
[shweta_sharma_1999@grace1 HW2-735]$ ./sort_list.exe 4 1  
List Size = 16, Threads = 2, error = 0, time (sec) = 0.0004, qsort_time = 0.0000  
[shweta_sharma_1999@grace1 HW2-735]$ ./sort_list.exe 4 2  
List Size = 16, Threads = 4, error = 0, time (sec) = 0.0006, qsort_time = 0.0000  
[shweta_sharma_1999@grace1 HW2-735]$ ./sort_list.exe 4 3  
List Size = 16, Threads = 8, error = 0, time (sec) = 0.0010, qsort_time = 0.0000  
[shweta_sharma_1999@grace1 HW2-735]$ ./sort_list.exe 20 4  
List Size = 1048576, Threads = 16, error = 0, time (sec) = 0.0294, qsort_time = 0.1539  
[shweta_sharma_1999@grace1 HW2-735]$ ./sort_list.exe 24 8  
List Size = 16777216, Threads = 256, error = 0, time (sec) = 0.2990, qsort_time = 2.6298  
[shweta_sharma_1999@grace1 HW2-735]$
```

All of these runs executed with an error 0.

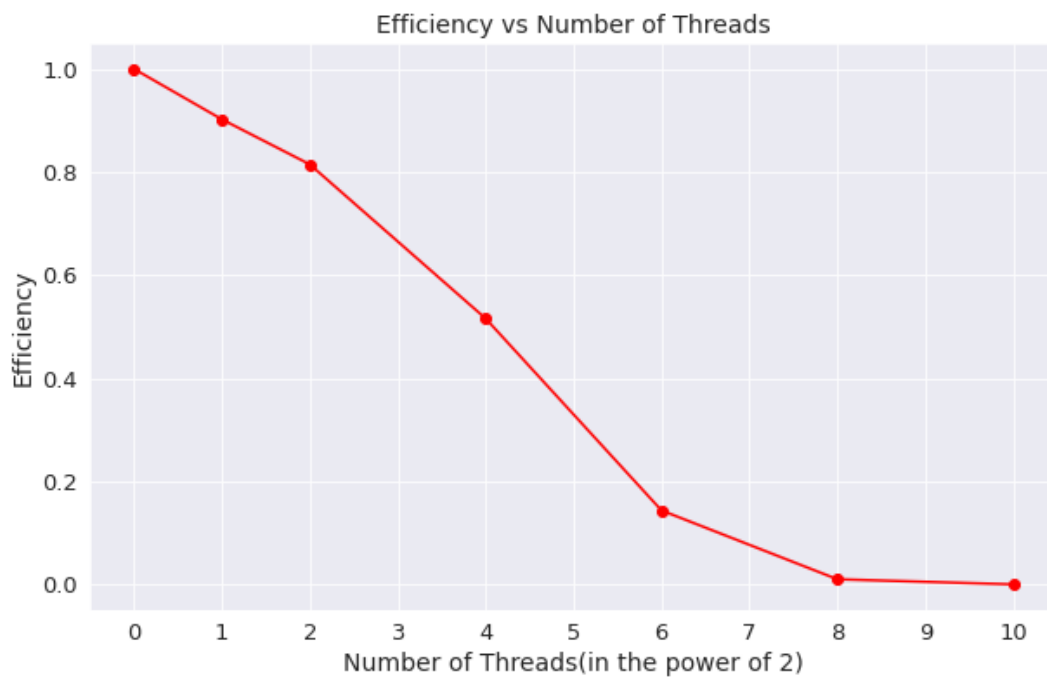
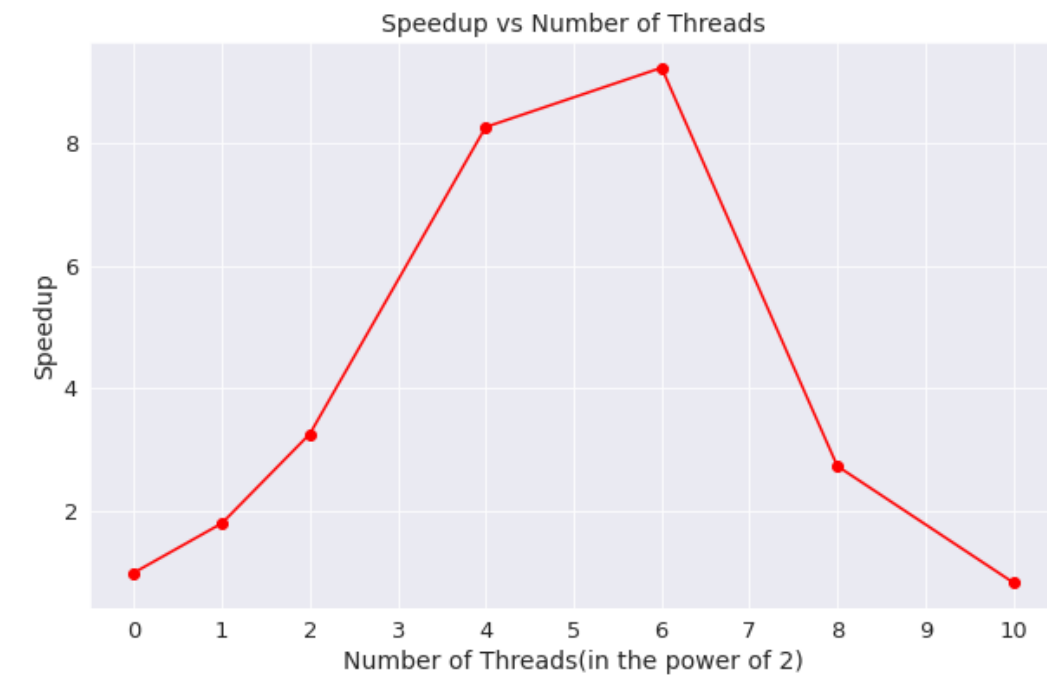
2. (20 points) Plot speedup and efficiency for all combinations of k and q chosen from the following sets: k = 12, 20, 28 ; q = 0, 1, 2, 4, 6, 8, 10. Comment on how the results of your experiments align with or diverge from your understanding of the expected behavior of the parallelized code.

Ans:

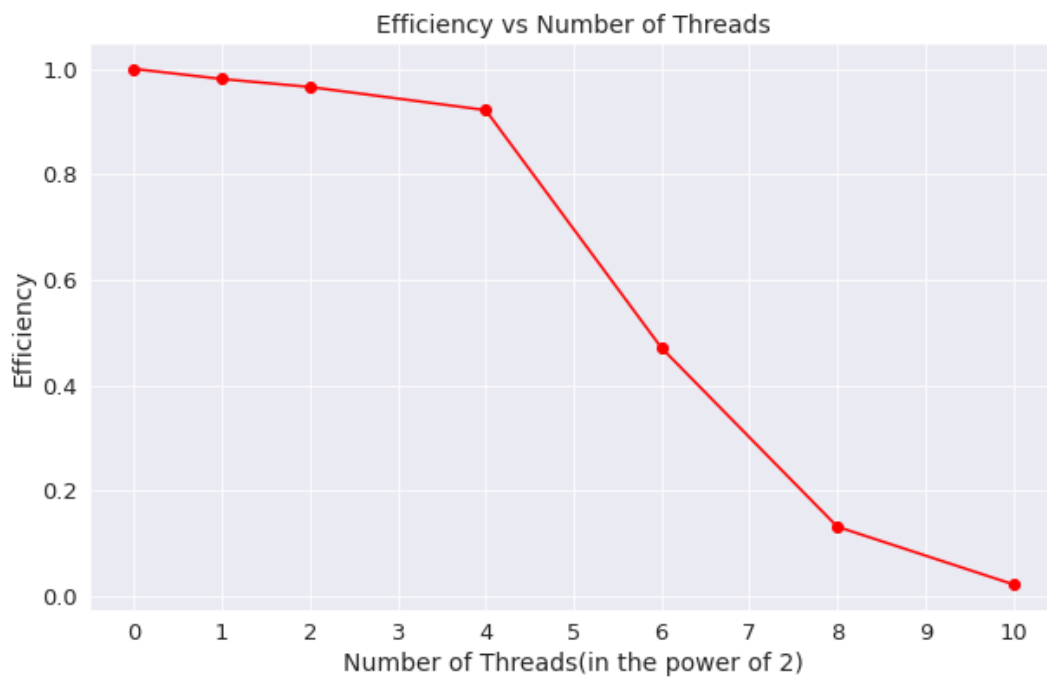
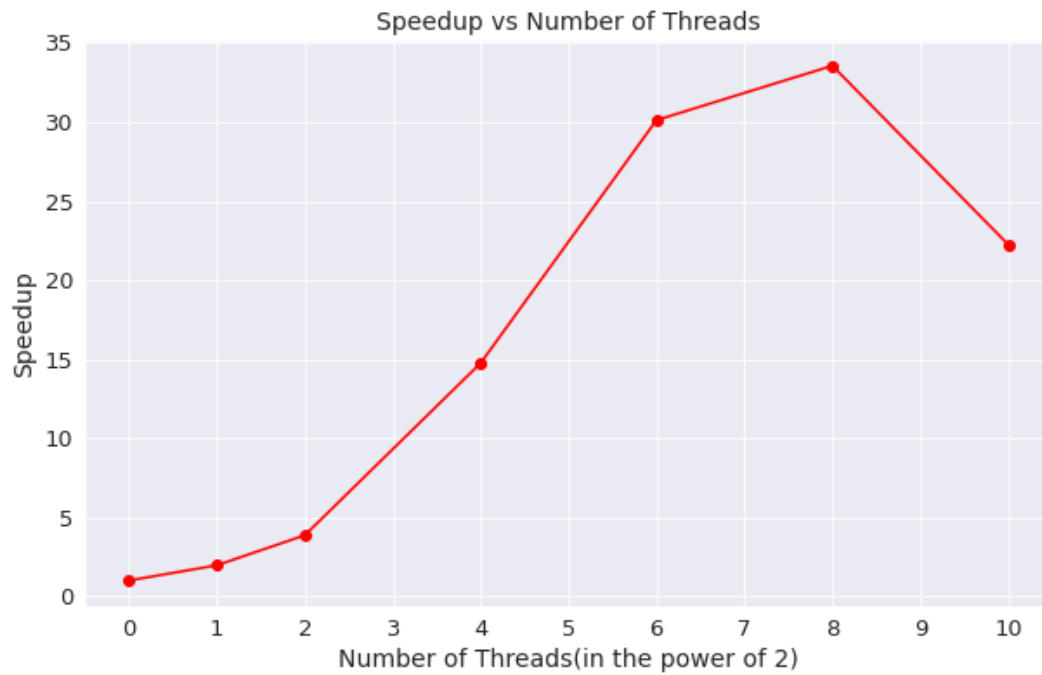
For k=12, we have the following curves:



For $k=20$, we have the following curves:



For $k=28$, we have the following curves:



It is surprising that only parallelising with 2 threads had maximum speedup when we tried sorting a list with $k = 12$. This suggests that the overhead of creating threads and merging lists by using multiple threads and to call hardware resources is a lot higher and takes more time than directly sorting the list conventionally.

For $k=20$, the speedup curve showed the general trend of the increase and decrease with respect to the number of threads, but still was not able to utilize the threads to the fullest. Ideally, the peak should be at 48 or more threads representing all the threads (cores) on the board executing, not idling. Its efficiency dropped linearly with more threads involved and down to almost 0 after 256 threads.

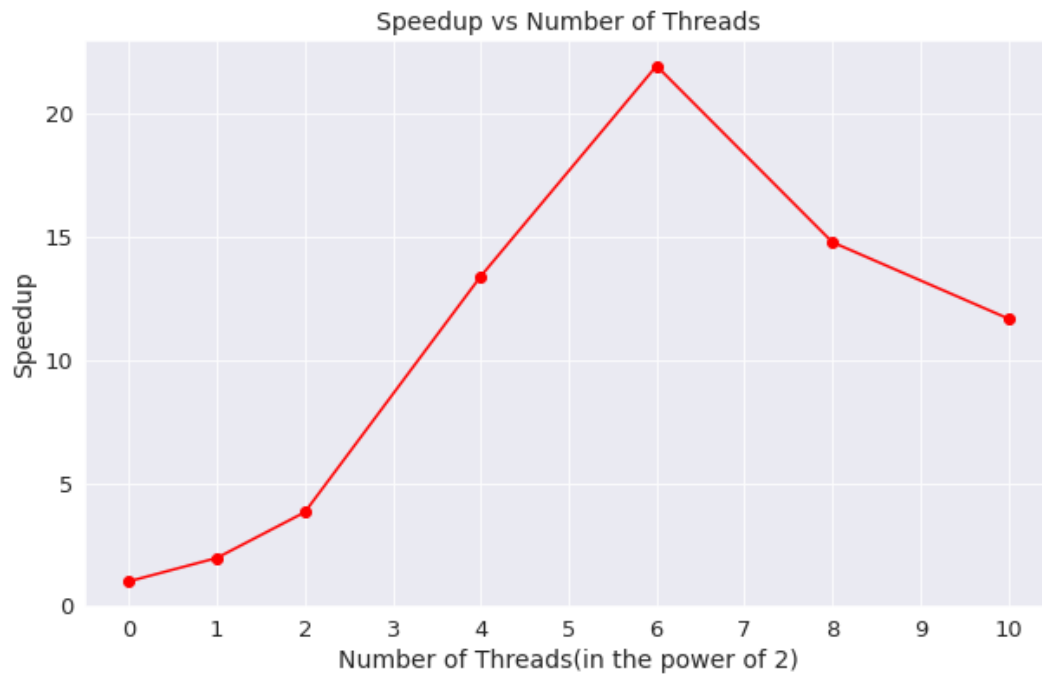
The speedup and efficiency curves best aligned with our expectation with $k = 28$. The maximum speedup (minimum execution time) was achieved when $q = 8$. This result was in conformity with the trend we observed in HW1. We had seen that for larger datasets, as the size increases, the maximum speedup is achieved when the number of threads increases as well until a point after which it decreases. This number of threads required for maximum speedup increases as the size of the data grows, consequently the efficiency decreases, because each thread does less amount of work if there are more threads.

This experiment showed results that are in sync with what we have learnt about parallelising code so far.

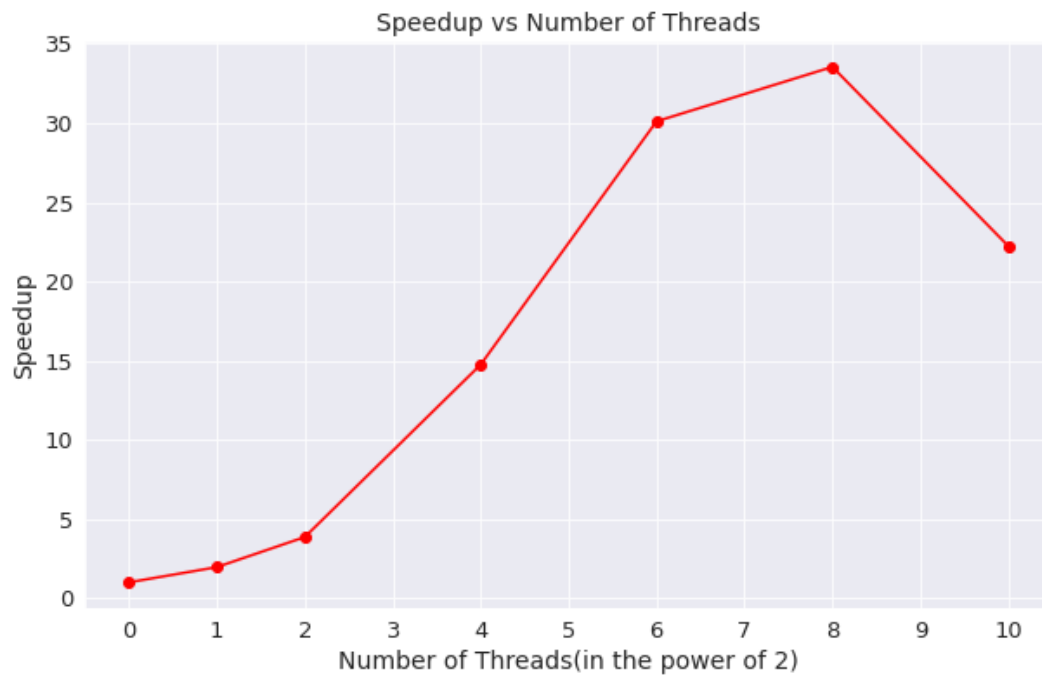
3. (10 points) Your code should demonstrate speedup when sorting lists of appropriate sizes. Determine two values of k for which your code shows speedup as q is varied. Present the timing results for your code along with speedup and efficiency obtained to convince the reader that you have a well-designed 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.

Ans:

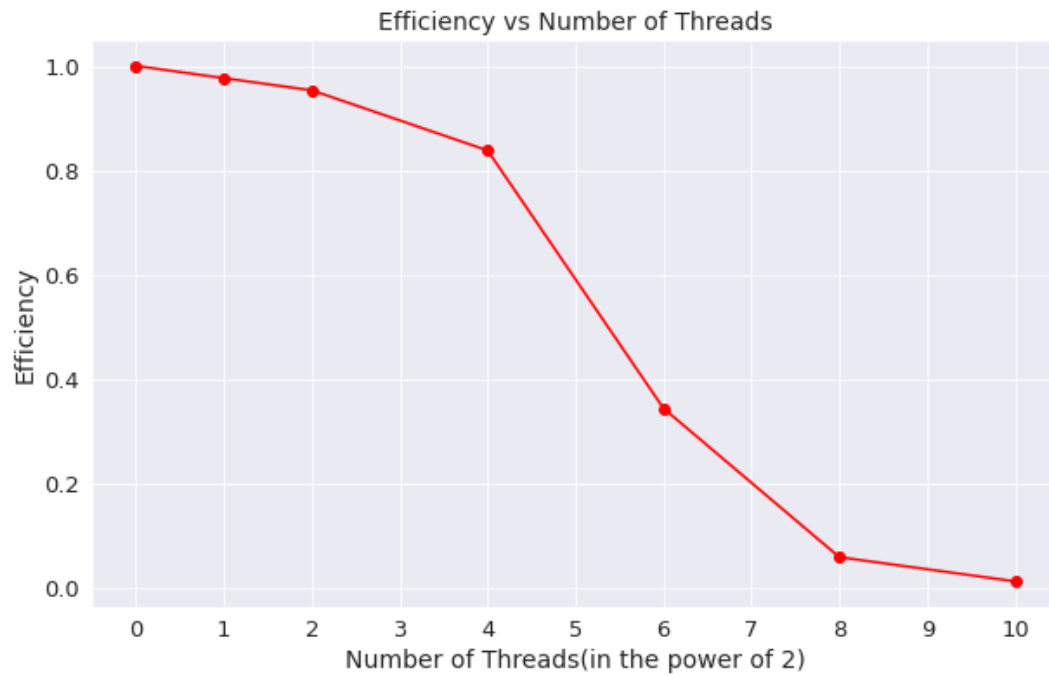
Speedup when $k=24$:



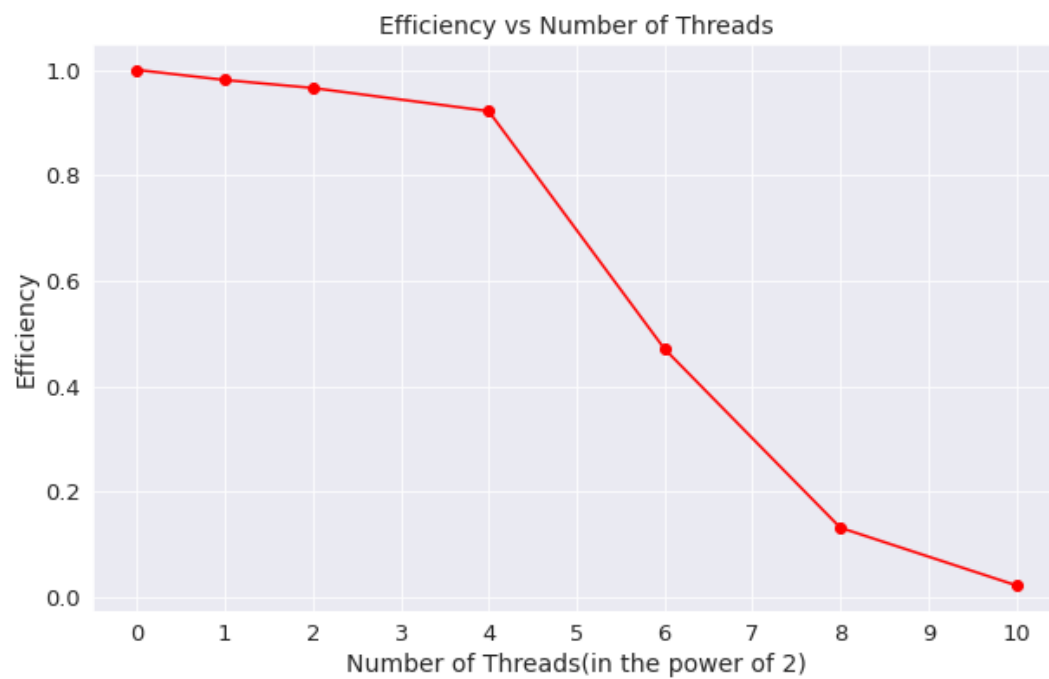
Speedup when $k=28$:



Efficiency when $k=24$:



Efficiency when $k=28$:



I selected two k values, $k = 24$ & 28 to demonstrate the parallelization of my code. Because a large dataset size could better exhibit the parallel utilization of threads. Both speedup and efficiency graphs showed that this code was under adequate exploitation of threads with maximum speedup at 64 and 128 threads respectively, which indicated that it employed 48 cores (threads) on the node efficiently. We also see that we need more threads for optimal speedup if our list size increases, which is exactly as expected out of an optimally parallelized code. Therefore, this clearly demonstrates that my code for parallelising merge sort works as expected.