Q2.

1.

# of threads	1	2	4	8
Running Time(μs)	2625	1686	1269	1349

```
-bash-4.2$ g++ -std=c++11 -pthread merge_sort.c -o mergesort && ./mergesort 1 && rm -f mergesort
Using 1 threads...
Time taken: 2625[μs]
Sorted array: 295 50340 99727 148320 194732 246835 299273 348544 394246 447475 499160 548164 599481 653038 700007 7
48889 800025 849470 898557 950697
-bash-4.2$ g++ -std=c++11 -pthread merge_sort.c -o mergesort && ./mergesort 2 && rm -f mergesort
Using 2 threads...
Time taken: 1686[μs]
Sorted array: 145 51412 97473 151327 201871 251653 302807 353640 402345 450864 503542 550978 598590 648671 704246 7
50484 803483 853044 904842 951539
-bash-4.2$ g++ -std=c++11 -pthread merge_sort.c -o mergesort && ./mergesort 4 && rm -f mergesort
Using 4 threads...
Time taken: 1269[μs]
Sorted array: 50 51636 104614 156509 208703 256592 306331 357555 404490 451704 500671 554451 605898 652338 701866 7
53901 802878 851437 901190 951897
-bash-4.2$ g++ -std=c++11 -pthread merge_sort.c -o mergesort && ./mergesort 8 && rm -f mergesort
Using 8 threads...
Time taken: 1349[μs]
Sorted array: 301 47555 93721 145317 192752 239926 291543 342056 398112 448234 497122 544424 592634 643790 695654 7
48185 797678 847628 900298 951031
-bash-4.2$
```

2. The biggest challenge I met on merge sort is the number of items assigned to each thread. There is a tricky thing that if the number of items cannot be divided by the number of threads then there will be some remaining items. So, I have to consider those remaining items in my last thread. The second challenge is that I tried to parallel the final merge part using pthread. For example, as for 8 threads sorting, I tried to use 4 threads to merge 4 parts of the array parallelly. Then I use 2 threads to merge the first half part and the second half part. Under these circumstances, I will have more overload for creating threads. I am wondering whether the overload is worthy rather than merging sequentially because as we can see the running time for 8 threads is not better than 4 threads in my code.

3. For **strong scaling:** we test using the fixed problem set, which is 7500 in our sorting. As for the result in the above form, we could tell that the speedup is not linear. We even found speedup less than 1 when we use 8 threads. So we conclude that the speedup is limited by the series code in the sorting, which is illustrated in Amdahl's law.

# of items	# of threads	Running time (μs)
10,000	1	3482
20,000	2	4331
40,000	4	5980
80,000	8	10067

For **Weak scaling:** we increase the problem size based on the number of the threads. Based on the form above, we could find that the scaled speedup is linear of the number of threads, which is the reflection of Gustafson's law.

References

1. Multithreading merge sort: https://geeksforgeeks.org/merge-sort-using-multi-threading/
2. Generate uniformly distributed pseudo random integers: https://en.cppreference.com/w/cpp/numeric/random/uniform_int_distribution
3. Measure cpp elapsed time: https://stackoverflow.com/questions/2808398/easily-measure-elapsed-time