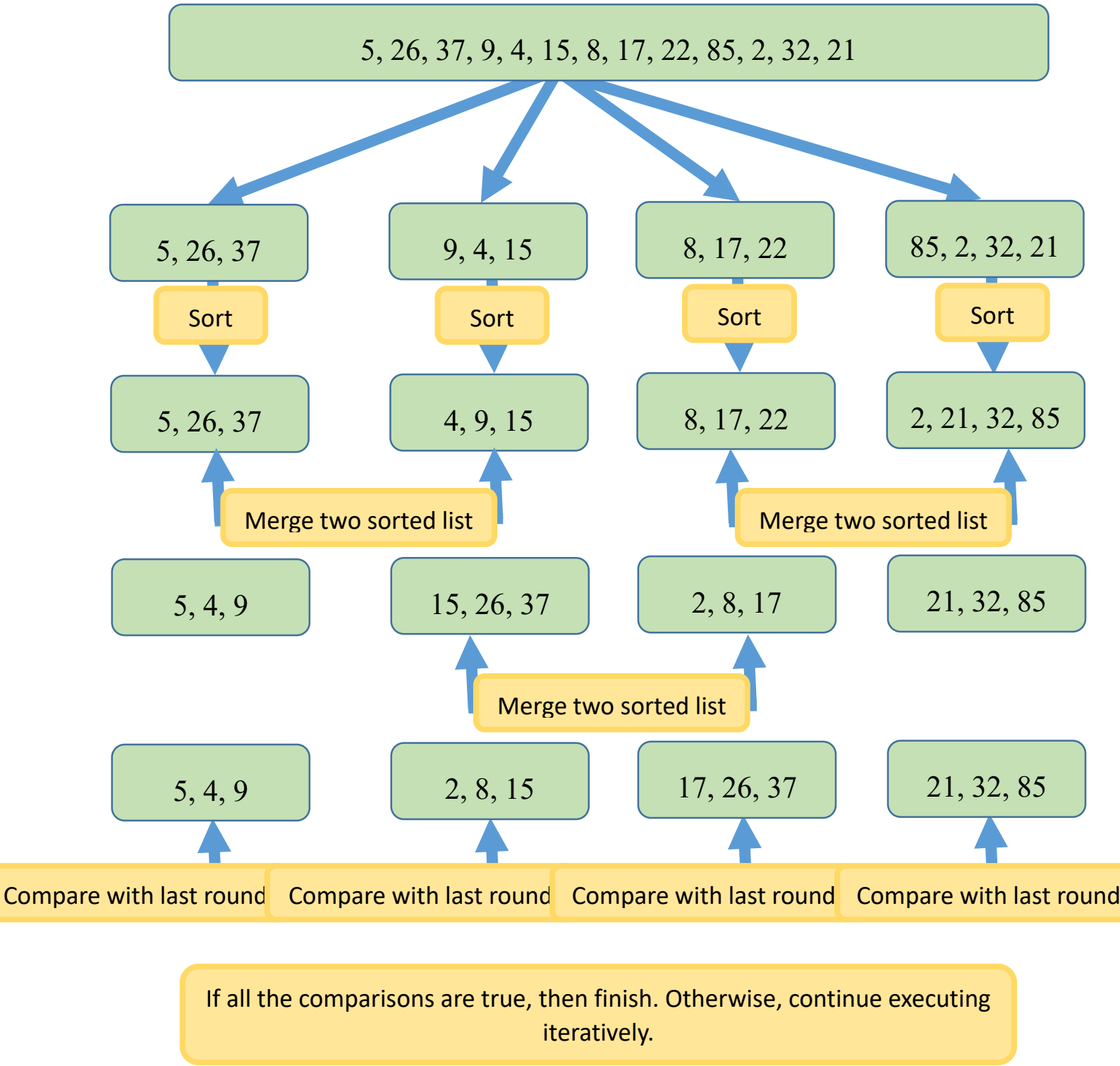
Homework 1: Odd-Even Sort

107062546 楊仲愷 資工所

Algorithm Example:

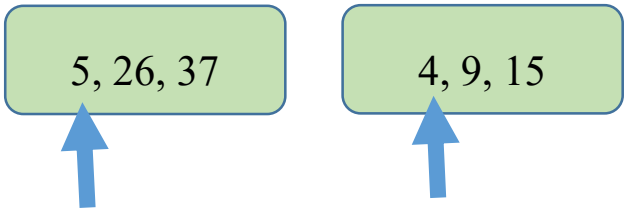


If all the comparisons are true, then finish. Otherwise, continue executing iteratively.

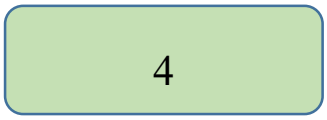
Merge two sorted list:

Type 1 (Choose the least 3 numbers):

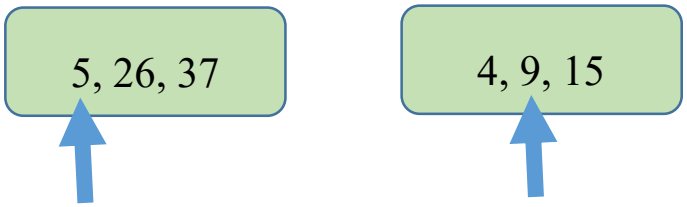
Round 1:



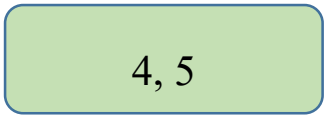
Compare 5 and 4, and choose the small one.



Round 2:



Compare 5 and 9, and choose the small one.



Iteratively, until choose the least 3 numbers.

And vice versa, type 2 (choose the biggest 3 numbers).

Time complexity: $O\left(\frac{n_1}{2}+\frac{n_2}{2}\right) = O(n_1 + n_2)$

- **Question:**

Q1. How do you handle arbitrary number of input items and processes?
A: I will check the rank whether it is the last one or not; if the process isn't the last, the item size will be $\text{total size}/_{number\ of\ process}$, and the final one should process $\text{total size}/_{number\ of\ process} +$ (total size) mod (number of size).

Q2. How do you sort in your program?
A: As I mentioned before.

Q3. Other efforts you've made in your program?
A: Actually, I have implemented several sorting algorithms which are bounded in a good time complexity, like quicksort, merge sort, etc. Furthermore, I have tried some works to improve quicksort because of its worst case. Nevertheless, I finally took advantage of a sorting method from C++ library. On the other hand, I think there are two ways to promise the array sorted. Firstly, the program can run the loop for size of processers times, and this can make the array sorted. Then, program can also iteratively execute until all the partitioned array are as same as the result from last round. Moreover, I tried to allocate the memory for array before the main function runs.

Experiment & Analysis

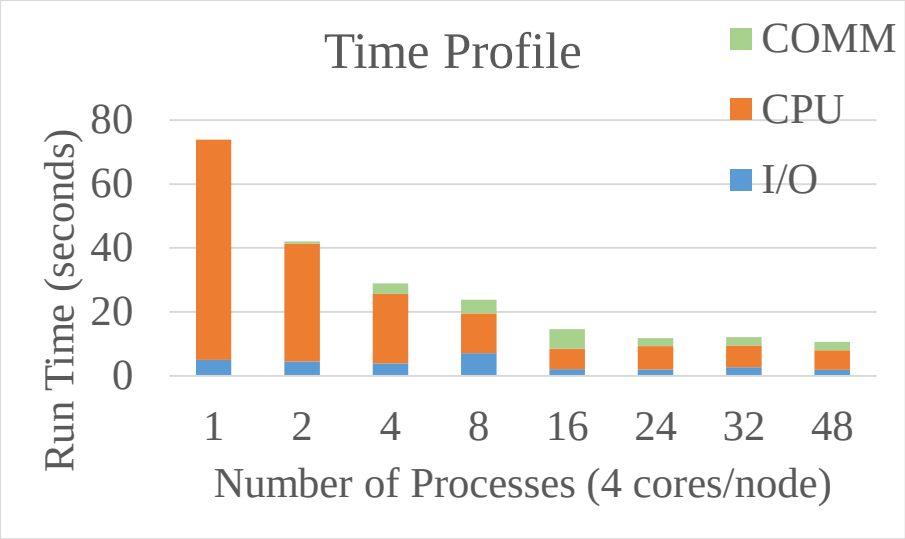
- **Methodology**

  Performance Metric:

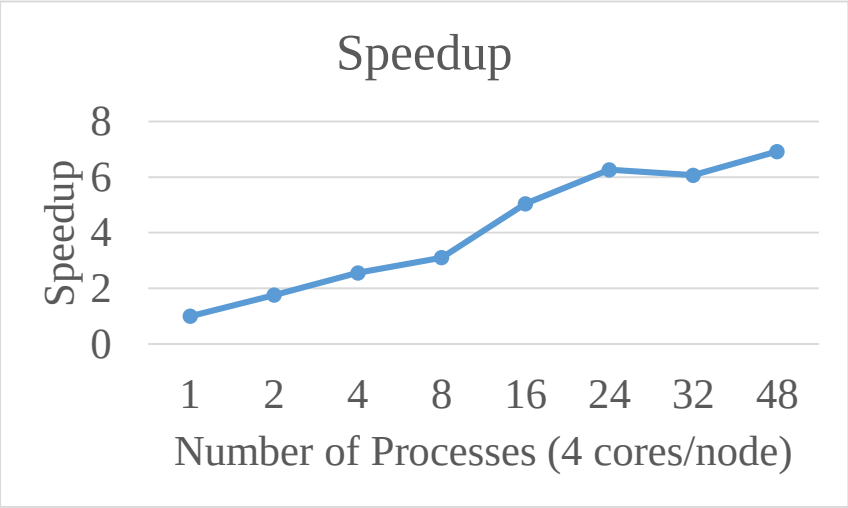
  (1) **Computing time:** the time for operating with CPU, such as, sorting algorithm, memory allocation, comparison, etc.
  (2) **Communication time:** the time for transmission with MPI function, such as, MPI_Send, MPI_Recv, etc.
  (3) **I/O time:** the time for processing I/O, such as, MPI_File_Open, MPI_File_read_at, MPI_File_write_at, etc.

- Time Profile & Speedup Factor

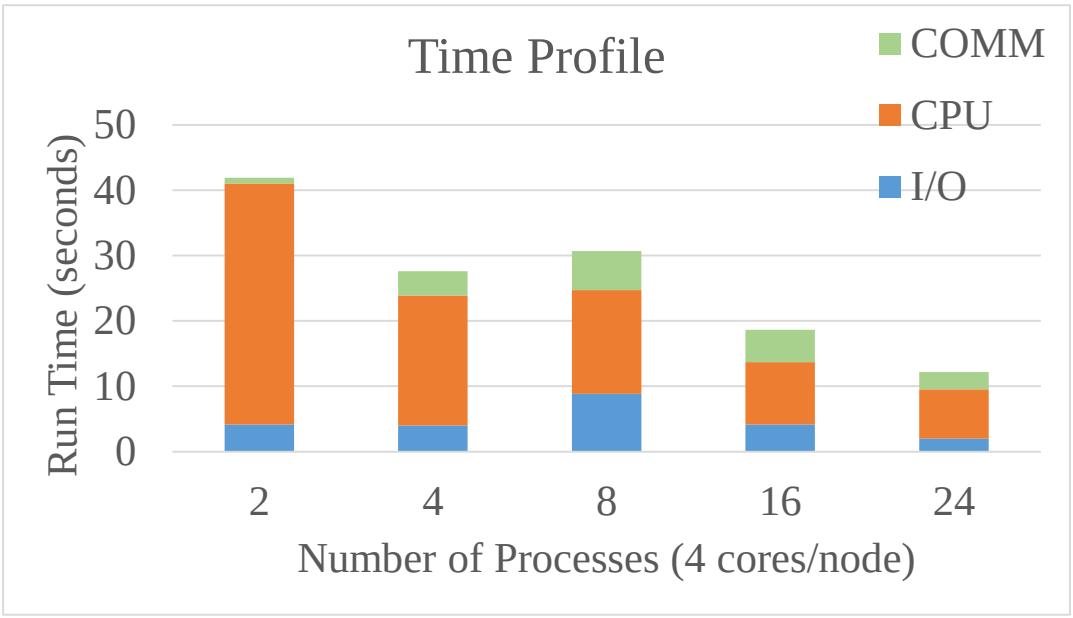
(1) Single Node



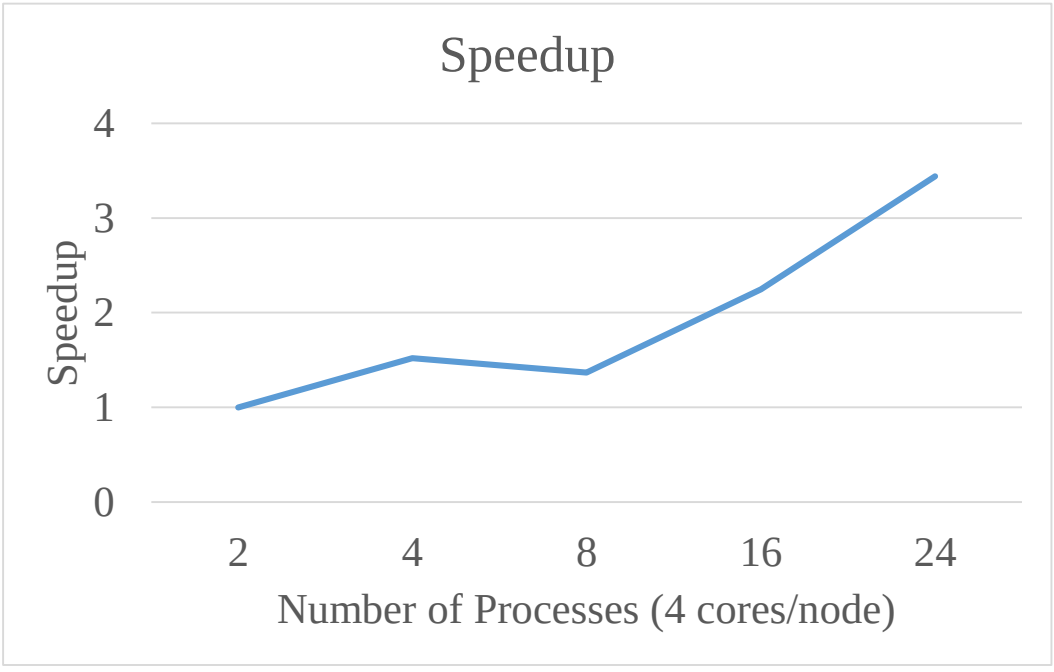
	I/O	CPU	COMM	Total Time	Speedup
1	5.065272	68.826932	0	**73.892204**	**1**
2	4.482421	36.893812	0.63109	**42.007323**	**1.759031**
4	3.933729	21.73155	3.226203	**28.891482**	**2.557577**
8	7.133818	12.433429	4.25672	**23.823967**	**3.101591**
16	2.139225	6.412036	6.10941	**14.660671**	**5.040165**
24	1.975035	7.353185	2.467591	**11.795811**	**6.264275**
32	2.706294	6.793404	2.676042	**12.17574**	**6.068806**
48	1.932623	6.045185	2.698963	**10.676771**	**6.920838**



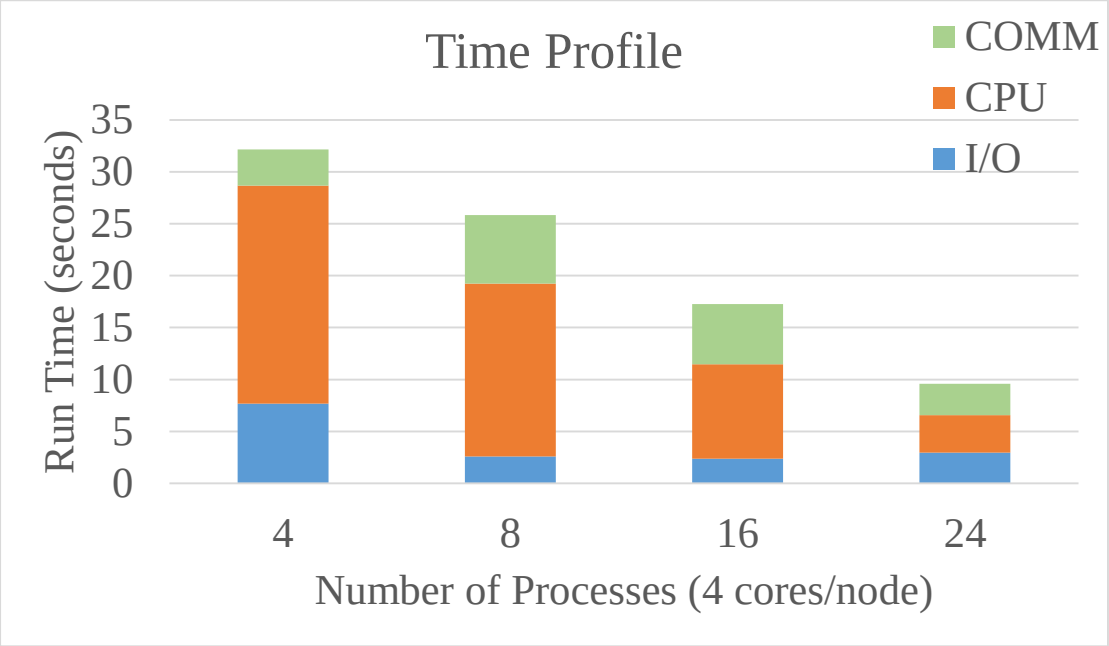
(2) 2-Node



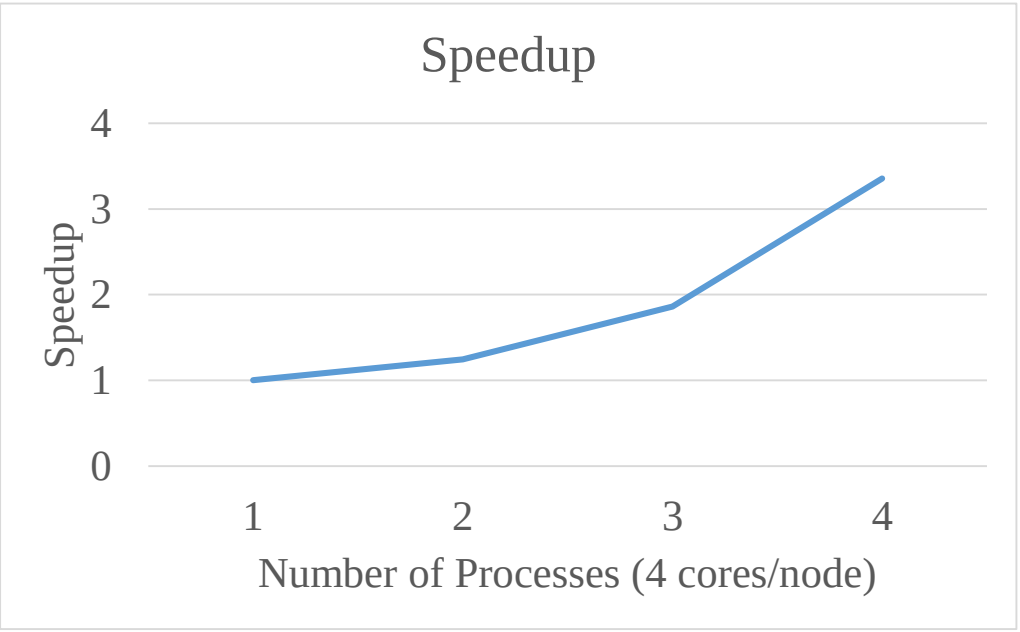
	I/O	CPU	COMM	Total Time	Speedup
2	4.129371	36.899843	0.884479	**41.913693**	**1**
4	3.998645	19.88915	3.70977	**27.597565**	**1.518746**
8	8.863936	15.876964	5.945667	**30.686567**	**1.365865**
16	4.16574	9.509277	4.974366	**18.649383**	**2.247457**
24	2.010109	7.521404	2.647392	**12.178905**	**3.441499**



(3) 4-Node



	I/O	CPU	COMM	Total Time	Speedup
4	7.671628	21.005816	3.487747	**32.165191**	**1**
8	2.571934	16.662314	6.591378	**25.825626**	**1.245476**
16	2.382809	9.094232	5.796128	**17.273169**	**1.862148**
24	2.956992	3.609236	3.021835	**9.588063**	**3.354712**



- Discussion

(1) According to the figures I demonstrated before, time for CPU processing is always the longest one. Therefore, if I want to accelerate it, I think the most important point is the sorting algorithm; because I have tried a lot of different sorting algorithm, and it always takes the most time.

(2) **Strong Scaling:** According to the figures I demonstrated before, with the number of processes arising, the time will decrease because the program executes parallel.

(3) **Weak Scaling:** According to the figures I demonstrate below, there are 100,000 samples and 400,000 samples in case 08 and case 19, respectively. Therefore, we can see the time is close.

```
[ppl9s33@apollo31 hw1]$ srun -n 2 ./hw1 100000 ./cases/19.in 19.out
total time: 0.022669
total time: 0.022946
[ppl9s33@apollo31 hw1]$ srun -n 8 ./hw1 400000 ./cases/19.in 19.out
total time: 0.095531
total time: 0.095377
```

- **Conclusion**

I learned a lot about how to communicate between processes with MPI function. I think the most difficult part is I have never touched anything about MPI; consequently, this is my first time to process such an unknown area. Moreover, there are a lot of program logic and bugs I have to get used to.