2 Parallel processing

Problem Introduction

In this problem you will simulate a program that processes a list of jobs in parallel. Operating systems such as Linux, MacOS or Windows all have special programs in them called schedulers which do exactly this with the programs on your computer.

Problem Description

Task. You have a program which is parallelized and uses n independent threads to process the given list of m jobs. Threads take jobs in the order they are given in the input. If there is a free thread, it immediately takes the next job from the list. If a thread has started processing a job, it doesn't interrupt or stop until it finishes processing the job. If several threads try to take jobs from the list simultaneously, the thread with smaller index takes the job. For each job you know exactly how long will it take any thread to process this job, and this time is the same for all the threads. You need to determine for each job which thread will process it and when will it start processing.

Input Format. The first line of the input contains integers n and m.
The second line contains m integers t_i — the times in seconds it takes any thread to process i-th job. The times are given in the same order as they are in the list from which threads take jobs.
Threads are indexed starting from 0.

Constraints. $1 \le n \le 10^5$; $1 \le m \le 10^5$; $0 \le t_i \le 10^9$.

Output Format. Output exactly m lines. i-th line (0-based index is used) should contain two space-separated integers — the 0-based index of the thread which will process the i-th job and the time in seconds when it will start processing that job.

Time Limits. `C`: 1 sec, `C++`: 1 sec, `Java`: 4 sec, `Python`: 6 sec. `C#`: 1.5 sec, `Haskell`: 2 sec, `JavaScript`: 6 sec, `Ruby`: 6 sec, `Scala`: 6 sec.

Memory Limit. 512MB.

Sample 1.

Input:

```
2 5
1 2 3 4 5
```

Output:

```
0 0
1 0
0 1
1 2
0 4
```

1. The two threads try to simultaneously take jobs from the list, so thread with index 0 actually takes the first job and starts working on it at the moment 0.
2. The thread with index 1 takes the second job and starts working on it also at the moment 0.
3. After 1 second, thread 0 is done with the first job and takes the third job from the list, and starts processing it immediately at time 1.
4. One second later, thread 1 is done with the second job and takes the fourth job from the list, and starts processing it immediately at time 2.
5. Finally, after 2 more seconds, thread 0 is done with the third job and takes the fifth job from the list, and starts processing it immediately at time 4.

Sample 2.

Input:

```
4 20
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

Output:

```
0 0
1 0
2 0
3 0
0 1
1 1
2 1
3 1
0 2
1 2
2 2
3 2
0 3
1 3
2 3
3 3
0 4
1 4
2 4
3 4
```

Explanation:
Jobs are taken by 4 threads in packs of 4, processed in 1 second, and then the next pack comes. This happens 5 times starting at moments 0, 1, 2, 3 and 4. After that all the $5 \times 4 = 20$ jobs are processed.

Starter Files

The starter solutions for C++, Java and Python3 in this problem read the input, apply an $\Theta(n^2)$ algorithm to solve the problem and write the output. You need to replace the $\Theta(n^2)$ algorithm with a faster one. If you use other languages, you need to implement the solution from scratch.

What to Do

Think about the sequence of events when one of the threads becomes free (at the start and later after completing some job). How to apply priority queue to simulate processing of these events in the required order? Remember to consider the case when several threads become free simultaneously.

Beware of integer overflow in this problem: use type `long long` in C++ and type `long` in Java wherever the regular type `int` can overflow given the restrictions in the problem statement.

Need Help?

Ask a question or see the questions asked by other learners at this forum thread.