Multiprocessor

Time Limit : 1 Second
Memory Limit : 512 MB

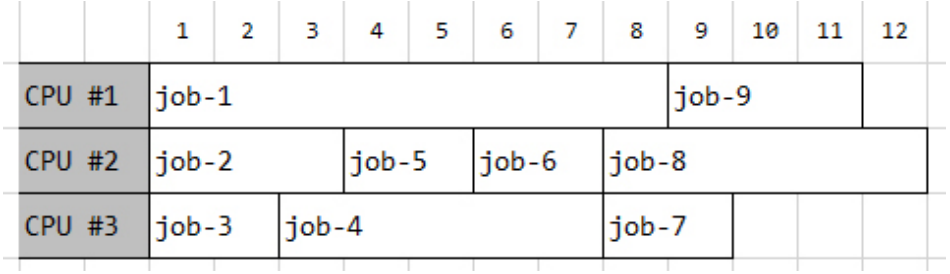
Nowadays, personal computers (workstations or laptops) are often equiped with a *multiprocessor* system. This system allows a computer to have multiple central processing units (CPUs) while sharing a common memory (RAM). In consequence, a multiprocessor system is able to run multiple processes at the same time in parallel.

Oski is a pasionate computer science student who wants to learn more about operating system. One important task in an operating system is scheduling, i.e. to determine which process (program/job) to run and how long it should run. Oski knows that scheduling is a hard problem, so he wants to start with the basic. The scheduling policy Oski considers is as follow.

- A CPU which is running a job is in a *busy* state.
- A CPU which is not running any job is in an *idle* state.
- A CPU can only run one job at a time.
- When an idle CPU accepts a job, it goes into a busy state and run the job until finished, then goes back to an idle state.
- If there is more than one idle CPU while a job is waiting to be run, the scheduler will assign the job to the idle CPU which ran the least amount of jobs so far. In case of tie, the scheduler will assign the job to the idle CPU with the smallest ID.
- The jobs will be assigned to CPUs in input order, i.e. sequentially from job-1 to job-N.

You may assume all CPUs have the same power, i.e. a job will be finished in exactly the same amount of time despite being run in which CPU.

For example, let there be K = 3 CPUs, N = 9 jobs, and the jobs need $A_{1..9}$ = {8, 3, 2, 5, 2, 2, 2, 5, 3} seconds to complete.



- At the beginning, the first three jobs will be run by CPU #1, #2, and #3, respectively.
- Job-3 which is run by CPU #3 finished in 2 seconds, after that, CPU #3 runs job-4.
- Job-2 which is run by CPU #2 finished in 3 seconds, after that, CPU #2 runs job-5.
- ...
- At the beginning of the 8^{th} second, there are two idle CPUs, i.e. CPU #2 and CPU #3, while there is a next job to run (job-7). At this time, CPU #2 already ran 3 jobs (job-2, job-5, and job-6) while CPU #3 ran 2 jobs (job-3 and job-4). Thus, job-7 is assigned to CPU #3 as CPU #3 ran the least number of jobs so far.
- Job-8 is assigned to CPU #2.
- ..
- The last job to finish is job-8, which is at the end of the 12^{th} second. So, the system needs 12 seconds to process all the jobs.

Given the number of jobs (N), number of CPUs (K), and the duration needed for each job to finish (A_i), your task in this problem is to determine the total time required by the system to finish all the jobs with the described policy.

Input

Input begins with an integer: T ($1 \le T \le 100$) denoting the number of cases.

Each case contains the following input block: Each case begins with two integers: N K ($1 \le N, K \le 100{,}000$) denoting the number of jobs and the number of CPUs, respectively. The next line contains N integers: A_i ($1 \le A_i \le 1000$) denoting the duration needed for a CPU to finish the i^{th} job for i = 1..N, respectively.

The sum of N over all test cases does not exceed 1,000,000.

Output

For each case, output in a line "`Case #X: Y`" where `X` is the case number (starts from 1) and `Y` is the output for the respective case.

Examples

input	**Example #1**

`4`
`9 1`
`8 3 2 5 2 2 2 5 3`
`9 2`
`8 3 2 5 2 2 2 5 3`
`9 3`
`8 3 2 5 2 2 2 5 3`
`5 2`
`10 1 5 200 30`	

output	

`Case #1: 32`
`Case #2: 17`
`Case #3: 12`
`Case #4: 206`	

explanation	
Case 1: There is only 1 CPU, so all jobs will be run sequentially one by one. Case 3: This is the example given in the problem statement.	

End of Problem