Task 1

A queue is a data structure based on the principle of 'First In First Out' (FIFO). There are two ends; one end can be used only to insert an item and the other end to remove an item. A Double Ended Queue is a queue where you can insert an item in both sides as well as you can delete an item from either side. There are mainly four operations available to a double ended queue. They are:

1. **pushLeft():** inserts an item to the left end of the queue with the exception that the queue is not full.
2. **pushRight():** inserts an item to the right end of the queue with the exception that the queue is not full.
3. **popLeft():** removes an item from the left end of the queue with the exception that the queue is not empty.
4. **popRight():** removes an item from the right end of the queue with the exception that the queue is not empty.

Now you are given a queue and a list of commands, you have to report the behavior of the queue.

Input

Input starts with an integer **T (≤ 20)**, denoting the number of test cases. Each case starts with a line containing two integers **n, m (1 ≤ n ≤ 10, 1 ≤ m ≤ 100)**, where **n** denotes the size of the queue and **m** denotes the number of commands. Each of the next **m** lines contains a command which is one of:

pushLeft **x**	pushes **x (-100 ≤ x ≤ 100)** in the left end of the queue
pushRight **x**	pushes **x (-100 ≤ x ≤ 100)** in the right end of the queue
popLeft	pops an item from the left end of the queue
popRight	pops an item from the right end of the queue

Output

For each case, print the case number in a line. Then for each operation, show its corresponding output as shown in the sample. Be careful about spelling.

Sample Input	Output for Sample Input
1 3 8 pushLeft 1 pushLeft 2 pushRight -1 pushRight 1 popLeft popRight popLeft popRight	Case 1: Pushed in left: 1 Pushed in left: 2 Pushed in right: -1 The queue is full Popped from left: 2 Popped from right: -1 Popped from left: 1 The queue is empty

Task 2

The only printer in the computer science students' union is experiencing an extremely heavy workload. Sometimes there are a hundred jobs in the printer queue and you may have to wait for hours to get a single page of output.

Because some jobs are more important than others, the Hacker General has invented and implemented a simple priority system for the print job queue. Now, each job is assigned a priority between 1 and 9 (with 9 being the highest priority, and 1 being the lowest), and the printer operates as follows.

- The first job J in queue is taken from the queue.
- If there is some job in the queue with a higher priority than job J, then move J to the end of the queue without printing it.
- Otherwise, print job J (and do not put it back in the queue).

In this way, all those important muffin recipes that the Hacker General is printing get printed very quickly. Of course, those annoying term papers that others are printing may have to wait for quite some time to get printed, but that's life.

Your problem with the new policy is that it has become quite tricky to determine when your print job will actually be completed. You decide to write a program to figure this out. The program will be given the current queue (as a list of priorities) as well as the position of your job in the queue, and must then calculate how long it will take until your job is printed, assuming that no additional jobs will be added to the queue. To simplify matters, we assume that printing a job always takes exactly one minute, and that adding and removing jobs from the queue is instantaneous.

Input

One line with a positive integer: the number of test cases (at most 100). Then for each test case:

- One line with two integers n and m, where n is the number of jobs in the queue ($1 \leq n \leq 100$) and m is the position of your job ($0 \leq m \leq n-1$). The first position in the queue is number 0, the second is number 1, and so on.
- One line with n integers in the range 1 to 9, giving the priorities of the jobs in the queue. The first integer gives the priority of the first job, the second integer the priority of the second job, and so on.

Output

For each test case, print one line with a single integer; the number of minutes until your job is completely printed, assuming that no additional print jobs will arrive.

Input	Output
9 1 0 5 4 2 1 2 3 4 6 0 1 1 9 1 1 1 7 1 2 3 4 4 9 1 3 8 5 9 8 9 9 8 8 9 8 2 0 5 9 6 2 3 3 4 3 3 3 6 1 4 1 3 9 7 4 6 1 4 3 1 9 7 4	1 2 5 5 8 2 1 6 5