Supermarket

This question is graded for 1.5%!

Statement

Near our Grandpa's home, there is a very "odd" supermarket. It has n queues, numbered from 1 to n for customers to line up, but only one cashier, so at most one customer can do checkout at a time, and of course, all customers must do checkout. There are q events(queries) to happen:

- *join* x: A person with an angriness level of x comes in. This customer will choose, among the n queues, the one with the fewest customers lining up and join the end of it. If there are at least two queues with that property, he will choose the one with the smallest queue number to join. Print out the number of the queue that he will choose to join.
- *serve*: The counter will choose one customer to serve. Among all customers that are standing in the front of all queues, the person with the lowest angriness level is served. If there are at least two customers with that property, the customer in the queue with the smallest queue number will be served. Print out the angriness of that person. After this query, this customer will leave the queue.
- *move* i: the person standing at the front of the ith queue will leave the current queue. At this moment, the length of the current queue will decrease by one. Then, the customer that just left will repeat the process of choosing a queue to join like in the 1st type of query (i.e: he will choose the shortest queue, and if there are ties, he will choose the one with smallest number). Print out the number of the queue that he will choose to join.

Do you think our Grandpa is able to solve this problem? Of course he can, he solved it in just 1 hour 45 minutes. Let's solve it faster than him!

Constraints

The following constraints apply for all subtasks.

- $1 \leq n, q \leq 10^5$
- In *join* queries, $0 \leq x \leq 10^9$
- In *serve* queries, it's guaranteed that there is at least one customer waiting at the moment
- In *move* queries, $1 \leq i \leq n$ and it's guaranteed that the ith queue is having at least one customer waiting

Further subtask-specific constraints will be listed below, under each subtask.

Input

The first line contains two integers n,q – the number of queues and the number of queries to process. Next q lines, each will contain a query of one of the above 3 types.

Output

For each query, print out its answer on a single line.

Examples

Sample Input	Expected Output
2 7 join 5 join 10 join 3 serve join 6 join 8 move 2	1 2 1 5 1 2 2

For the 1st example, there are 2 queues, the states of them after each query are as follows:

Time	1st Queue	2nd Queue	Explanation
Before 1st query	[ ]	[ ]	Both queues are initially empty
After 1st query	[5]	[10]	Two queues have the same length, hence the customer chooses the 1st queue
After 2nd query	[5]	[10]	Queue 2 is shorter, hence this customer joins it
After 3rd query	[5,3]	[10]	Two queues have the same length, hence the customer chooses the 1st queue
After 4th query	[3]	[10]	Among all customers at front of queues, the customer with minimum angriness (5) will be served.
After 5th query	[3,6]	[10]	Obviously this customer will join the 1st queue
After 6th query	[3,6]	[10,8]	Obviously this customer will join the 2nd queue
After 7th query	[3,6]	[8,10]	Customer with angriness value of 10 will leave the queue, after that the two queues will be [3,6] and [8]. Obviously, this customer will choose to join the 2nd queue again.

Subtasks

You may choose to solve any one of the following subtasks.
This accounts for 70% of your score for this problem.

- **Subtask 1 (60%)**:
  The complexity for each query can be up to $O(q + n \log n)$
- **Subtask 2 (75%)**:
  The complexity for each query can be up to $O(n \log n)$
- **Subtask 3 (90%)**:
  The complexity for each of *join* queries can be up to $O(\log n)$.
  The complexity of for each of *serve*, *move* queries can be up to $O(n \log n)$
- **Subtask 4 (110%)**:
  The complexity for each query can be up to $O(\log n)$

Marking Scheme

- Correctness and Efficiency (70%)
  - Refer to the Subtasks section.
- Programming Style (30%)
  - Purpose of methods and statements (5%)
  - Pre- and post- conditions (5%)
  - Modularity (5%)
  - Scoping (5%)
  - Meaningful Identifiers (4%)
  - Variable Naming (1%)
  - Identation (5%)

Notes

1. A skeleton file has been given to help you. You should not create a new file or rename the file provided. You should develop your program using this skeleton file.
2. You are free to define your own helper methods and classes (or remove existing ones) if it is suitable but you must put all the new classes, if any, in the same skeleton file provided.

Skeleton File

You are given the skeleton file `Supermarket.java`. You should see the following contents when you open the file:

```java
/**
 * Name          :
 * Matric. No    :
*/

import java.util.*;

public class Supermarket {
    private void run() {
        // implement your "main" method here
    }

    public static void main(String args[]) {
        Supermarket runner = new Supermarket();
        runner.run();
    }
}
```