A - Ferry Loading *IV*

Source file name: `ferry.py`
Time limit: 1 second

Before bridges were common, ferries were used to transport cars across rivers. River ferries, unlike their larger cousins, run on a guide line and are powered by the river's current. Cars drive onto the ferry from one end, the ferry crosses the river, and the cars exit from the other end of the ferry.

There is an l-meter long ferry that crosses the river. A car may arrive at either river bank to be transported by the ferry to the opposite bank. The ferry travels continuously back and forth between the banks so long as it is carrying a car or there is at least one car waiting at either bank. Whenever the ferry arrives at one of the banks, it unloads its cargo and loads up cars that are waiting to cross as long as they fit on its deck. The cars are loaded in the order of their arrival; ferry's deck accommodates only one lane of cars. The ferry is initially on the left bank where it broke and it took quite some time to fix it. In the meantime, lines of cars formed on both banks that await to cross the river.

Input

The first line of input contains c, the number of test cases. Each test case begins with a line containing l and m separated by a blank. Then m lines follow describing the cars that arrive in this order to be transported. Each line gives the length of a car (in centimeters), and the bank at which the car arrives ("`left`" or "`right`").

The input must be read from standard input.

Output

For each test case, output the number of times the ferry has to cross the river in order to serve all cars.

The output must be written to standard output.

Sample Input	**Sample Output**
4 20 4 380 left 720 left 1340 right 1040 left 15 4 380 left 720 left 1340 right 1040 left 15 4 380 left 720 left 1340 left 1040 left 15 4 380 right 720 right 1340 right 1040 right	3 3 5 6

B - I Can Guess the Data Structure!

Source file name: `guess.py`
Time limit: 1 second

There is a bag-like data structure, supporting two types of commands:

`1 x` throws element x into the bag, and

`2` takes out an element from the bag.

Given a sequence of operations with return values, you're going to guess the data structure. It is a stack (last-in, first-out), a queue (first-in, first-out), a priority-queue (always take out larger elements first), or something else that you can hardly imagine!

Input

There are several test cases. Each test case begins with a line containing a single integer n ($1 \leq n \leq 1000$). Each of the next n lines is either a type-1 command or an integer 2 followed by an integer x, meaning that after executing a type-2 command, we get an element x *without* error. The value of x is always a positive integer not larger than 100. The input is terminated by end-of-input.

The input must be read from standard input.

Output

For each test case, output one of the following:

`stack` if it is definitely a stack;

`queue` if it is definitely a queue;

`priority queue` if it is definitely a priority queue;

`impossible` if it is none of the above; and

`not sure` if it can be more than one of the data structures mentioned above.

The output must be written to standard output.

Sample Input	**Sample Output**
6 1 1 1 2 1 3 2 1 2 2 2 3 6 1 1 1 2 1 3 2 3 2 2 2 1 2 1 1 2 2 4 1 2 1 1 2 1 2 2 7 1 2 1 5 1 1 1 3 2 5 1 4 2 4	queue not sure impossible stack priority queue

C - Generalized Matrioshkas

Source file name: `matrioshkas.py`
Time limit: 1 second

Vladimir worked for years making *matrioshkas*, those nesting dolls that certainly represent truly Russian craft. A matrioshka is a doll that may be opened in two halves, so that one finds another doll inside. Then this doll may be opened to find another one inside it. This can be repeated several times, till a final doll –that cannot be opened- is reached.

Recently, Vladimir realized that the idea of nesting dolls might be generalized to nesting toys. Indeed, he has designed toys that contain toys but in a more general sense. One of these toys may be opened in two halves and it may have more than one toy inside it. That is the new feature that Vladimir wants to introduce in his new line of toys.

Vladimir has developed a notation to describe how nesting toys should be constructed. A toy is represented with a positive integer, according to its size. More precisely: if when opening the toy represented by m we find the toys represented by $n_1, n_2, \ldots, n_r$, it must be true that $n_1 + n_2 + \cdots + n_r < m$. And if this is the case, we say that toy m contains directly the toys $n_1, n_2, \ldots, n_r$. It should be clear that toys that may be contained in any of the toys $n_1, n_2, \ldots, n_r$ are not considered as directly contained in the toy m.

A *generalized matrioshka* is denoted with a non-empty sequence of non zero integers of the form:

$$a_1, a_2, \ldots, a_N$$

such that toy k is represented in the sequence with two integers $-k$ and k, with the negative one occurring in the sequence first that the positive one.

For example, the sequence $-9, -7, -2, 2, -3, -2, -1, 1, 2, 3, 7, 9$ represents a generalized matrioshka conformed by six toys, namely, 1, 2 (twice), 3, 7, and 9. Note that toy 7 contains directly toys 2 and 3. Note that the first copy of toy 2 occurs left from the second one and that the second copy contains directly a toy 1. It would be wrong to understand that the first -2 and the last 2 should be paired.

On the other hand, the following sequences do not describe generalized matrioshkas:

- $-9, -7, -2, 2, -3, -1, -2, 2, 1, 3, 7, 9$ because toy 2 is bigger than toy 1 and cannot be allocated inside it.
- $-9, -7, -2, 2, -3, -2, -1, 1, 2, 3, 7, -2, 2, 9$ because 7 and 2 may not be allocated together inside 9.
- $-9, -7, -2, 2, -3, -1, -2, 3, 2, 1, 7, 9$ because there is a nesting problem within toy 3.

Your problem is to write a program to help Vladimir telling good designs from bad ones.

Input

The input file contains several test cases, each one of them in a separate line. Each test case is a sequence of non zero integers, each one with an absolute value less than 10^7.

The input must be read from standard input.

Output

Output texts for each input case are presented in the same order that input is read.

For each test case the answer must be a line of the form

```
:-) Matrioshka!
```

if the design describes a generalized matrioshka. In other case, the answer should be of the form

```
:-( Try again.
```

The output must be written to standard output.

Sample Input	**Sample Output**
-9 -7 -2 2 -3 -2 -1 1 2 3 7 9	:-) Matrioshka!
-9 -7 -2 2 -3 -1 -2 2 1 3 7 9	:-(Try again.
-9 -7 -2 2 -3 -1 -2 3 2 1 7 9	:-(Try again.
-100 -50 -6 6 50 100	:-) Matrioshka!
-100 -50 -6 6 45 100	:-(Try again.
-10 -5 -2 2 5 -4 -3 3 4 10	:-) Matrioshka!
-9 -5 -2 2 5 -4 -3 3 4 9	:-(Try again.