Problem E. Binary Search Algorithm

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2 seconds
Memory limit:	256 mebibytes

This is an interactive problem.

I have a hidden permutation $p_1, p_2, \ldots, p_n$. You are not to guess it. Your task is to devise a *data structure* (that's against the rules!) that supports the following operations on a set S, which is initially empty:

- "`add` x" — put element x in S,
- "`delete` x" — delete element x from S,
- "`getMin`" — print the element x from S such that p_x is the smallest among x in S.

You will have to perform "`getMin`" after each operation of other types.

You don't know the permutation, but you can make queries. In one query you can choose k distinct indices x_1, x_2, $\ldots$, x_k for some value of k, and in return I will tell you the permutation of these indices y_1, y_2, $\ldots$, y_k such that $p_{y_1} < p_{y_2} < \ldots < p_{y_k}$. In other words, I will sort the indices according to p.

Note that all x_i should be present in S at the moment of query.

It is easy to perform "`getMin`" in 1 query — just sort everything in S. It is also not hard to perform it using several queries with sum of k up to $O(\log n)$. Can you flex your *algorithm* (this is lame) muscles and satisfy both?

Note that since you don't know p and my task is to make your solution fail, I **can** change p depending on your queries, but only in such a way that all my previous responses are correct. I **can** also choose the order of operations you have to perform depending on your queries.

Input

Initially you are given a single line with one integer n ($1 \le n \le 8000$) — the number of elements. Each element will be inserted and deleted exactly once.

Interaction Protocol

Then there will be exactly $2n$ rounds of interaction.

Each round of interaction consists of 4 phases:

1. you read the next operation on a separate line: either "`add` x" or "`delete` x" for some $1 \le x \le n$;
2. you choose some $0 \le k \le \min(|S|, 30)$ and print $k+1$ numbers on a separate line: k first, then x_1, x_2, $\ldots$, x_k: the k elements you want to sort. Elements you choose should be between 1 and n, should be **distinct**, and should be in S at this time. Note that S is already changed according to phase 1;
3. you read k integers y_1, y_2, $\ldots$, y_k on a separate line: y is a permutation of x you just printed, and $p_{y_1} < p_{y_2} < \ldots < p_{y_k}$;
4. you print a single integer x on a separate line, such that x is in S, and p_x is the smallest possible. Print -1 if S is empty.

It is guaranteed that all $2n$ possible operations ("`add` x" and "`delete` x" for all $1 \le x \le n$) will occur exactly once, and for each x operation "`add` x" will precede "`delete` x".

Do not forget to print end of line and flush your output before you read anything.

Example

standard input	standard output
3	
add 1	
	1 1
1	
	1
add 3	
	2 1 3
3 1	
	3
delete 1	
	1 3
3	
	3
add 2	
	2 2 3
3 2	
	3
delete 3	
	1 2
2	
	2
delete 2	
	0
	-1

Note

In the example $p = [2, 3, 1]$.