Bitly

As a new employer at Intel, you are faced with a notoriously repetitive job of manipulating register bits for configuring processors. The CS brain inside of you just wants an easier way to manipulate these bits with style and efficiency. You are generally faced with the following tasks in bit manipulation: setting and clearing a specific bit, and performing the logical OR/AND operation of any 2 bits. The register always initially contains all zeros.

To streamline your work, you are going to write a program that can *understand* the following commands:

- `ONE i` - puts a 1 at bit position i
- `ZERO i` - puts a 0 at bit position i
- `EITHER i j` - stores at i the logical OR operation result of bit i and bit j
- `BOTH i j` - stores at i the logical AND operation result of bit i and bit j

Since this is a 32-bit register, the values of i and j are always $0 <= i,j < 32$. Within the register the leftmost bit is the most significant bit (index 31) and the rightmost bit is the least significant bit (index 0).

Input

The first line of input contains the number of commands that follow, $0 < n <= 100$. The next n lines contain commands as indicated above each on a separate line.

Output

Content of the register by itself (no newline character at the end). The least significant bit should be on the right and the most significant bit should be on the left.

Example 1

```
Input:
2
ONE 20
EITHER 1 20

Output:
00000000000100000000000000000010
```

In the first operation bit position 20 was flipped to 1. In the second operation the result of logical OR between bits 20 and 1 is written to bit at position 1.

Example 2

```
Input:
4
BOTH 0 1
ONE 0
EITHER 3 0
ZERO 3

Output:
00000000000000000000000000000001
```

Line Up

Queuing, it's what the British are renowned for doing - and doing very well. Better than anyone else in the world, if reputation is to be believed. Today, queues can be found just about anywhere (even in the virtual world).

Standard *Queues* are well known to most computer science students. It's a first-in-first-out data structure. Kobayashi is already familiar with it. But she is assigned to write some code to maintain a variant of Standard Queues: **Group Queues**.

In a group queue each element belongs to a group. If an element is pushed to the queue and there is no element that belongs to the same group in the queue, it will be placed behind the last element in the queue. Otherwise, if there are already some elements that belong to the same group, it should be placed immediately behind them.

Dequeuing works the same as int the standard queue: We will remove the first element according to the current order from head to tail in the queue.

Kanna is preparing for her school's Sports Festival, and she wants Kobayashi to come to this important event. But Kobayashi will not be able to attend unless she can finish her program early. As Kobayashi's colleague and close friend, you decide to help her. You need to write a program that simulates such a group queue.

Input

The 1st line contains an integer `n` (`1 <= n <= 1,000`) equal to the number of groups, as described above.

Then `n` group descriptions follow. Each group is described on one line by an integer `k` (the number of elements belonging to the group) followed by `k` integers (the indexes of the elements in this group).

Elements are indexed with integer in the range `[ 0 , 999,999 ]`. And a group may consist of up to `1,000` elements.

Finally, a list of commands follows. There are 3 different kinds of commands:

- `Push ind` : Push the element with index `ind` into the group queue.
- `Pop` : Output the first element and remove it from the queue.
- `Shutdown` : Stop your program (end of input).

There may be up to `200,000` commands in the input file, so the implementation of the group queue should be efficient: both enqueuing and dequeuing of an element should only take a constant time.

Output

For each `Pop` command, print an integer, the index of the element which is dequeued, followed by a newline.

Example 1

```
Input:
2
3 1 2 3
3 4 5 6
Push 1
Push 4
Push 2
Push 5
Push 3
Push 6
Pop
Pop
Pop
Pop
Pop
Pop
Shutdown

Output:
1
```

```
2
3
4
5
6
```

Example 2

```
Input:
2
3 1 2 3
3 4 5 6
Push 1
Push 2
Push 4
Pop
Pop
Push 3
Push 5
Pop
Pop
Push 6
Pop
Pop
Shutdown

Output:
1
2
4
5
3
6
```

Giant Squad

Conflict Empire is fighting a war against Light Kingdom. The commander of Conflict Army is building a squad of giants. Giants are huge creatures, typically a dozen feet tall, inhabiting the territory of Conflict Empire. Giants are so tall that their height differences are also very large, preventing orders from being effectively transmitted. To address this issue, the commander decides to form squads with only odd number of giants with different heights and select the captain in such a way so that the number of giants in the squad who are shorter than that captain is equal to the number of giants who are taller than that captain. You are given heights of all giants in that squad and you are asked to determine the height of the captain.

Input

The input consists of a single line, starting with an integer N ($1 < N < 11$, N is odd), the number of giants in that squad, followed by N integers representing heights of the giants of the squad. Those N integers will be between 11 and 20 (inclusive). Additionally, heights will be given in strictly increasing or decreasing order.

Output

You should print a single integer x, where x is the height of the captain.

Example 1

```
Input:
5 19 17 16 14 12

Output:
16
```

Example 2

```
Input:
5 12 14 16 17 18

Output:
16
```

Running Median

Professor Stats has a new theory of how useful the median value could be in describing a real-life data set. She is especially interested in analyzing the running medians. As her research assistant, you are tasked with writing a program that calculates the running median as the data is received, i.e., for each subsequence starting at data point 1 up to data point K (where K ranges from 1 up to the largest index of any data point), you need to compute the median of that subsequence.

If the current sequence has an odd number of elements, then the median is the middle element when the values are in sorted order. For example, for the sequence {1, 3, 6, 2, 7}, the median is 3.

If the current sequence has an even number of elements, then the median is the average of the two middle elements. For example, for the sequence {1, 3, 6, 2, 7, 8}, the median is (3+6)/2 = 4.5. Since Professor Stats lives in the Integer World, you need to report only the integer part of the median, so in this case the program should report 4 (not 4.5).

Input

The input consists of series of integers `X` (`0 <= X <= 2^31`). The total number of integers `N` is less than 100,000. Each line of input contains only one integer, but the numbers may have leading or trailing spaces.

Output

For each input line (each new value), print the current value of the running median.

Example 1

```
Input:
1
3
4
60
70
50
2

Output:
1
2
3
3
4
27
4
```

Unique Subarray

A subarray is the sequence of consecutive elements in an array. A *unique* subarray is a subarray in which all elements are unique (i.e., no repeated values). Given an array, your task is to calculate the maximum length of a *unique* subarray.

For example [4, 3, 2, 2, 1], the maximum length of of a *unique* subarray is 3. The *unique* subarray is [4,3,2]. [3,2,2,1] is a subarray of length 4, but since 2 appears twice in this subarray, it is not a *unique* subarray.

Input

The 1st line contains an integer n (1 <= n <= 10^6), specifying the length of the array.

The following line contains n integers, in range [1, 10^9], denoting the elements of the array.

Output

The maximum length of a *unique* subarray.

Example 1

```
Input:
5
4 3 2 2 1

Output:
3
```

Example 2

```
Input:
6
1 1 1 1 1 1

Output:
1
```