

Robin is stuck doing a terminally boring data entry job. She has to enter a lot of numeric observations manually and report running medians after each step to her project lead. The task is sufficiently mind-numbing for her to make a lot of entry errors. Fortunately, she has a keen eye and often notices her mistakes. On the other hand, sometimes she makes mistakes while spotting - or correcting - previous mistakes, and so on.

Your task is to implement a program that will help Robin finish this boring job as soon as possible. The program will accept user input and report a running median for all the observations entered so far. It should also be capable of rolling back to any previous state (including those that were previously rolled back on their own).

Input Format

First line contains an integer, T , the total number of lines to follow.

Each of the following T lines contains either:

1. a positive integer N that will contribute in calculation of the current running median;
2. or a request to roll back to a previous state, which is represented by a negative number between -1 and $-M$, where M is the total number of queries executed so far. -1 requests a return to the immediately preceding state (which actually results in no state change), while $-M$ requests a return to the state after entering the very first number.

Output Format

Output T integers, representing current running medians after performing operations on corresponding lines in the input.

Constraints

- $1 \leq T \leq 10^5$
- $1 \leq N \leq 10^9$
- $1 \leq M \leq \text{number of queries executed so far}$.

Notes

- You will never be asked to roll back to a non-existent state, or a state for which the running median is not well-defined. In particular, the first line after T will always be a positive number.
- You should print out the median after executing each query, including rollbacks.
- For collections with even number of elements we report the smaller of the two candidate medians.

Sample Input

```
10
1
```

5
-2
3
2
5
4
-7
2
-3

Sample Output

1
1
1
1
2
2
3
1
1
3

Explanation

(This represents the collection of the numbers tracked after each step, with median in bold.)

Step 1: 1 -> [**1**]

Step 2: 5 -> [**1**, 5]

Step 3: request a return to state after step $(3 - 2) = 1$ -> [**1**]

Step 4: 3 -> [**1**, 3]

Step 5: 2 -> [1, **2**, 3]

Step 6: 5 -> [1, **2**, 3, 5]

Step 7: 4 -> [1, 2, **3**, 4, 5]

Step 8: request a return to state after step $(8 - 7) = 1$ -> [**1**]

Step 9: 2 -> [**1**, 2]

Step 10: request a return to state after step $(10 - 3) = 7$ -> [1, 2, **3**, 4, 5]

Tested by: [Abhiranjan Kumar](#)