A. Submission Bait

```
Input file:     standard input
Output file:    standard output
Time limit:     1 second
Memory limit:   256 megabytes
```

Alice and Bob are playing a game in an array a of size n.

They take turns to do operations, with Alice starting first. The player who can not operate will lose. At first, a variable mx is set to 0.

In one operation, a player can do:

- Choose an index i ($1 \le i \le n$) such that $a_i \ge mx$ and set mx to a_i. Then, set a_i to 0.

Determine whether Alice has a winning strategy.

Input

The first line contains an integer t ($1 \le t \le 10^3$) — the number of test cases.

For each test case:

- The first line contains an integer n ($2 \le n \le 50$) — the size of the array.
- The second line contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le n$) — the elements of the array.

Output

For each test case, if Alice has a winning strategy, output "YES". Otherwise, output "NO".

You can output the answer in any case (upper or lower). For example, the strings "yEs", "yes", "Yes", and "YES" will be recognized as positive responses.

Standard Input	Standard Output
5 2 2 1 2 1 1 3 3 3 3 4 3 3 4 4 4 1 2 2 2	YES NO YES NO YES

Note

In the first test case, Alice can choose $i = 1$ since $a_1 = 2 \ge mx = 0$.

After Alice's operation, $a = [0, 1]$ and $mx = 2$. Bob can not do any operation. Alice wins.

In the second test case, Alice doesn't have a winning strategy.

For example, if Alice chooses $i = 1$, after Alice's operation: $a = [0, 1]$ and $mx = 1$. Then, Bob can choose $i = 2$ since $a_2 = 1 \geq mx = 1$. After Bob's operation: $a = [0, 0]$ and $mx = 1$. Alice can not do any operation. Bob wins.

B. Array Craft

```
Input file:     standard input
Output file:    standard output
Time limit:     1 second
Memory limit:   256 megabytes
```

For an array b of size m, we define:

- the **maximum prefix position** of b is the **smallest** index i that satisfies $b_1 + \ldots + b_i = \max_{j=1}^{m}(b_1 + \ldots + b_j)$;
- the **maximum suffix position** of b is the **largest** index i that satisfies $b_i + \ldots + b_m = \max_{j=1}^{m}(b_j + \ldots + b_m)$.

You are given three integers n, x, and y ($x > y$). Construct an array a of size n satisfying:

- a_i is either 1 or -1 for all $1 \le i \le n$;
- the **maximum prefix position** of a is x;
- the **maximum suffix position** of a is y.

If there are multiple arrays that meet the conditions, print any. It can be proven that such an array always exists under the given conditions.

Input

The first line contains an integer t ($1 \le t \le 10^4$) — the number of test cases.

For each test case:

- The only line contains three integers n, x, and y ($2 \le n \le 10^5, 1 \le y < x \le n)$.

It is guaranteed that the sum of n over all test cases will not exceed 10^5.

Output

For each test case, output n space-separated integers $a_1, a_2, \ldots, a_n$ in a new line.

Standard Input	Standard Output
3 2 2 1 4 4 3 6 5 1	1 1 1 -1 1 1 1 1 -1 1 1 -1

Note

In the second test case,

- $i = x = 4$ is the **smallest** index that satisfies $a_1 + \ldots + a_i = \max_{j=1}^{n}(a_1 + \ldots + a_j) = 2$;
- $i = y = 3$ is the **greatest** index that satisfies $a_i + \ldots + a_n = \max_{j=1}^{n}(a_j + \ldots + a_n) = 2$.

Thus, the array $a = [1, -1, 1, 1]$ is considered correct.

,

C. Mad MAD Sum

```
Input file:     standard input
Output file:    standard output
Time limit:     2 seconds
Memory limit:   256 megabytes
```

We define the MAD (Maximum Appearing Duplicate) in an array as the largest number that appears at least twice in the array. Specifically, if there is no number that appears at least twice, the MAD value is 0.

For example, $\operatorname{MAD}([1, 2, 1]) = 1$, $\operatorname{MAD}([2, 2, 3, 3]) = 3$, $\operatorname{MAD}([1, 2, 3, 4]) = 0$.

You are given an array a of size n. Initially, a variable sum is set to 0.

The following process will be executed in a sequential loop until all numbers in a become 0:

1. Set $sum := sum + \sum_{i=1}^{n} a_i$;
2. Let b be an array of size n. Set $b_i :=\ \operatorname{MAD}([a_1, a_2, \ldots, a_i])$ for all $1 \le i \le n$, and then set $a_i := b_i$ for all $1 \le i \le n$.

Find the value of sum after the process.

Input

The first line contains an integer t ($1 \le t \le 2 \cdot 10^4$) — the number of test cases.

For each test case:

- The first line contains an integer n ($1 \le n \le 2 \cdot 10^5$) — the size of the array a;
- The second line contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le n$) — the elements of the array.

It is guaranteed that the sum of n over all test cases will not exceed $2 \cdot 10^5$.

Output

For each test case, output the value of sum in a new line.

Standard Input	Standard Output
4 1 1 3 2 2 3 4 2 1 1 2 4 4 4 4 4	1 13 9 40

Note

In the first test case, $a = [1]$ initially.

In the first loop:

1. Set $sum := sum + a_1 = 0 + 1 = 1$;

2. Set $b_1 := \operatorname{MAD}([a_1]) = \operatorname{MAD}([1]) = 0$, and then set $a_1 := b_1$.

After the first loop, $a = [0]$ and the process ends. The value of sum after the process is 1.

In the second test case, $a = [2, 2, 3]$ initially.

After the first loop, $a = [0, 2, 2]$ and $sum = 7$.

After the second loop, $a = [0, 0, 2]$ and $sum = 11$.

After the third loop, $a = [0, 0, 0]$ and $sum = 13$. Then the process ends.

The value of sum after the process is 13.

D. Grid Puzzle

```
Input file:     standard input
Output file:    standard output
Time limit:     2 seconds
Memory limit:   256 megabytes
```

You are given an array a of size n.

There is an $n \times n$ grid. In the i-th row, the first a_i cells are black and the other cells are white. In other words, note (i, j) as the cell in the i-th row and j-th column, cells $(i, 1), (i, 2), \ldots, (i, a_i)$ are black, and cells $(i, a_i + 1), \ldots, (i, n)$ are white.

You can do the following operations any number of times in any order:

- Dye a 2×2 subgrid white;
- Dye a whole row white. Note you can **not** dye a whole column white.

Find the minimum number of operations to dye all cells white.

Input

The first line contains an integer t $(1 \le t \le 10^4)$ — the number of test cases.

For each test case:

- The first line contains an integer n $(1 \le n \le 2 \cdot 10^5)$ — the size of the array a.
- The second line contains n integers $a_1, a_2, \ldots, a_n$ $(0 \le a_i \le n)$.

It's guaranteed that the sum of n over all test cases will not exceed $2 \cdot 10^5$.

Output

For each test case, output a single integer — the minimum number of operations to dye all cells white.

Standard Input	Standard Output
10 1 0 4 2 4 4 2 4 3 2 1 0 3 0 3 0 3 0 1 3 3 3 1 0 4 3 1 0 3 4 0 2 2 2 6	0 3 2 1 2 2 3 2 4 6

1 3 4 2 0 4 8 2 2 5 2 3 4 2 4	

Note

In the first test case, you don't need to do any operation.

In the second test case, you can do:

- Dye $(1,1), (1,2), (2,1)$, and $(2,2)$ white;
- Dye $(2,3), (2,4), (3,3)$, and $(3,4)$ white;
- Dye $(3,1), (3,2), (4,1)$, and $(4,2)$ white.

It can be proven 3 is the minimum number of operations.

In the third test case, you can do:

- Dye the first row white;
- Dye $(2,1), (2,2), (3,1)$, and $(3,2)$ white.

It can be proven 2 is the minimum number of operations.

,

E1. Catch the Mole(Easy Version)

```
Input file:     standard input
Output file:    standard output
Time limit:     4 seconds
Memory limit:   256 megabytes
```

This is the easy version of the problem. The only difference is the limit on the number of queries.

This is an interactive problem.

You are given a tree of n nodes with node 1 as its root node.

There is a hidden mole in one of the nodes. To find its position, you can pick an integer x ($1 \le x \le n$) to make an inquiry to the jury. Next, the jury will return 1 when the mole is in subtree x. Otherwise, the judge will return 0. If the judge returns 0 and the mole is not in root node 1, the mole will move to the parent node of the node it is currently on.

Use at most 300 operations to find the **current** node where the mole is located.

Input

Each test contains multiple test cases. The first line contains the number of test cases t ($1 \le t \le 100$). The description of the test cases follows.

Interaction

The first line of each test case contains one integer n ($2 \le n \le 5000$).

The following $n-1$ lines describe the edges of the tree. Each line contains two space-separated integers u_i and v_i ($1 \le u_i, v_i \le n$), indicating an edge between nodes u_i and v_i.

It is guaranteed that the input data represents a tree.

The interactor in this task is **not adaptive**. In other words, the node where the mole is located at first is fixed in every test case and does not change during the interaction.

To ask a query, you need to pick a vertex x ($1 \le x \le n$) and print the line of the following form:

- "? x"

After that, you receive:

- 0 if the mole is not in subtree x;
- 1 if the mole is in subtree x.

You can make at most 300 queries of this form for each test case.

Next, if your program has found the **current** node where the mole is located, print the line of the following form:

- "! x"

Note that this line is **not** considered a query and is **not** taken into account when counting the number of queries asked.

After this, proceed to the next test case.

If you make more than 300 queries during an interaction, your program must terminate immediately, and you will receive the `Wrong Answer` verdict. Otherwise, you can get an arbitrary verdict because your solution will continue to read from a closed stream.

After printing a query or the answer for a test case, do not forget to output the end of line and flush the output. Otherwise, you will get the verdict `Idleness Limit Exceeded`. To do this, use:

- `fflush(stdout)` or `cout.flush()` in C++;
- `System.out.flush()` in Java;
- `flush(output)` in Pascal;
- `stdout.flush()` in Python;
- see the documentation for other languages.

Hacks

To hack, follow the test format below.

The first line contains the number of test cases t ($1 \le t \le 100$). The description of the test cases follows.

The first line of each test case contains two integers n and x ($2 \le n \le 5000$, $1 \le x \le n$) — the size of the tree and the initial position of the mole.

The following $n-1$ lines describe the edges of the tree. Each line contains two space-separated integers u_i and v_i ($1 \le u_i, v_i \le n$), indicating an edge between nodes u_i and v_i.

The input data must represent a tree.

Standard Input	Standard Output
2 2 1 2 1 6 1 2 1 3 1 4 4 5 5 6 0 0 1	 ? 2 ! 2 ? 2 ? 6 ? 4 ! 4

Note

In the first test case, the mole is in node 2 initially.

For the query "? 2", the jury returns 1 because the mole is in subtree 2. After this query, the mole does not move.

The answer 2 is the **current** node where the mole is located, so the answer is considered correct.

In the second test case, the mole is in node 6 initially.

For the query "? 2", the jury returns 0 because the mole is not in subtree 2. After this query, the mole moves from node 6 to node 5.

For the query "? 6", the jury returns 0 because the mole is not in subtree 6. After this query, the mole moves from node 5 to node 4.

For the query "? 4", the jury returns 1 because the mole is in subtree 4. After this query, the mole does not move.

The answer 4 is the **current** node where the mole is located, so the answer is considered correct.

Please note that the example is only for understanding the statement, and the queries in the example do **not** guarantee to determine the unique position of the mole.

,

E2. Catch the Mole(Hard Version)

```
Input file:     standard input
Output file:    standard output
Time limit:     4 seconds
Memory limit:   256 megabytes
```

This is the hard version of the problem. The only difference is the limit on the number of queries.

This is an interactive problem.

You are given a tree of n nodes with node 1 as its root node.

There is a hidden mole in one of the nodes. To find its position, you can pick an integer x ($1 \le x \le n$) to make an inquiry to the jury. Next, the jury will return 1 when the mole is in subtree x. Otherwise, the judge will return 0. If the judge returns 0 and the mole is not in root node 1, the mole will move to the parent node of the node it is currently on.

Use at most 160 operations to find the **current** node where the mole is located.

Input

Each test contains multiple test cases. The first line contains the number of test cases t ($1 \le t \le 100$). The description of the test cases follows.

Interaction

The first line of each test case contains one integer n ($2 \le n \le 5000$).

The following $n-1$ lines describe the edges of the tree. Each line contains two space-separated integers u_i and v_i ($1 \le u_i, v_i \le n$), indicating an edge between nodes u_i and v_i.

It is guaranteed that the input data represents a tree.

The interactor in this task is **not adaptive**. In other words, the node where the mole is located at first is fixed in every test case and does not change during the interaction.

To ask a query, you need to pick a vertex x ($1 \le x \le n$) and print the line of the following form:

- "? x"

After that, you receive:

- 0 if the mole is not in subtree x;
- 1 if the mole is in subtree x.

You can make at most 160 queries of this form for each test case.

Next, if your program has found the **current** node where the mole is located, print the line of the following form:

- "! x"

Note that this line is **not** considered a query and is **not** taken into account when counting the number of queries asked.

After this, proceed to the next test case.

If you make more than 160 queries during an interaction, your program must terminate immediately, and you will receive the `Wrong Answer` verdict. Otherwise, you can get an arbitrary verdict because your solution will continue to read from a closed stream.

After printing a query or the answer for a test case, do not forget to output the end of line and flush the output. Otherwise, you will get the verdict `Idleness Limit Exceeded`. To do this, use:

- `fflush(stdout)` or `cout.flush()` in C++;
- `System.out.flush()` in Java;
- `flush(output)` in Pascal;
- `stdout.flush()` in Python;
- see the documentation for other languages.

Hacks

To hack, follow the test format below.

The first line contains the number of test cases t ($1 \le t \le 100$). The description of the test cases follows.

The first line of each test case contains one integer n and x ($2 \le n \le 5000$, $1 \le x \le n$) — the size of the tree and the initial position of the mole.

The following $n - 1$ lines describe the edges of the tree. Each line contains two space-separated integers u_i and v_i ($1 \le u_i, v_i \le n$), indicating an edge between nodes u_i and v_i.

The input data must represent a tree.

Standard Input	Standard Output
2	
2	
1 2	? 2
1	! 2
6	
1 2	
1 3	
1 4	
4 5	
5 6	? 2
0	? 6
0	? 4
1	! 4

Note

In the first test case, the mole is in node 2 initially.

For the query "? 2", the jury returns 1 because the mole is in subtree 2. After this query, the mole does not move.

The answer 2 is the **current** node where the mole is located, so the answer is considered correct.

In the second test case, the mole is in node 6 initially.

For the query "? 2", the jury returns 0 because the mole is not in subtree 2. After this query, the mole moves from node 6 to node 5.

For the query "? 6", the jury returns 0 because the mole is not in subtree 6. After this query, the mole moves from node 5 to node 4.

For the query "? 4", the jury returns 1 because the mole is in subtree 4. After this query, the mole does not move.

The answer 4 is the **current** node where the mole is located, so the answer is considered correct.

Please note that the example is only for understanding the statement, and the queries in the example do **not** guarantee to determine the unique position of the mole.

,

F. Polygonal Segments

```
Input file:     standard input
Output file:    standard output
Time limit:     8 seconds
Memory limit:   512 megabytes
```

You are given an array a of size n.

A segment $[l, r](1 \le l < r \le n)$ is called a **polygonal** segment only if the following conditions hold:

- $(r - l + 1) \ge 3$;
- Considering $a_l, a_{l+1}, \ldots, a_r$ as side lengths, these sides can form a polygon with $(r - l + 1)$ sides.

Process q queries of two types:

- "`1 l r`": find the length of the longest segment among all **polygonal** segments $[l_0, r_0]$ satisfying $l \le l_0 \le r_0 \le r$. If there is no such **polygonal** segment, output -1 instead;
- "`2 i x`": assign $a_i := x$.

Input

The first line contains an integer t ($1 \le t \le 10^4$) — the number of test cases.

For each test case:

- The first line of each testcase contains two integers n, q ($4 \le n \le 2 \cdot 10^5$, $1 \le q \le 10^5$);
- The second line of each testcase contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 10^{12}$);
- The following q lines contain the description of queries. Each line is of one of two types:
  - "`1 l r`" ($1 \le l < r \le n$, $r - l + 1 \ge 3$);
  - "`2 i x`" ($1 \le i \le n$, $1 \le x \le 10^{12}$).

It is guaranteed that the sum of n over all test cases will not exceed $2 \cdot 10^5$, and the sum of q over all test cases will not exceed 10^5.

Output

For each query, if there is no suitable segment, output -1 in a new line. Otherwise, output the length of the longest segment satisfying the condition above in a new line.

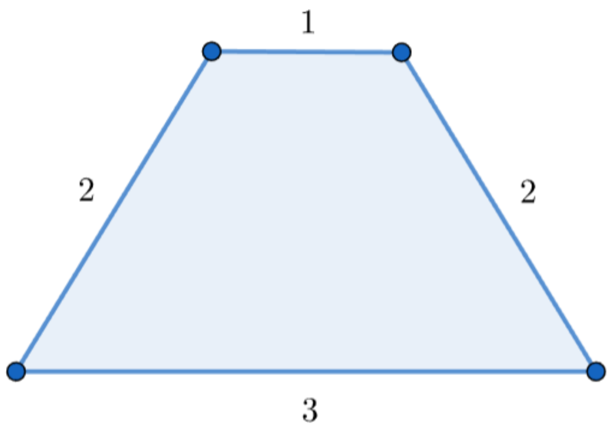
Standard Input	Standard Output
2 5 6 3 1 2 2 8 1 1 3 1 1 4 1 1 5 2 1 5 1 1 4 1 1 5 4 10	-1 4 4 3 5 -1 -1 4 -1 3

500000000000 500000000000 1000000000000 500000000000 1 1 3 1 2 4 1 1 4 2 1 499999999999 2 3 999999999999 1 1 3 1 2 4 1 1 4 2 3 1000000000000 1 1 3	4 -1

Note

In the first query of the first test case, there is no **polygonal** segment under the given condition. For example, considering segment $[1, 3]$, you can not form a triangle with side lengths of $a_1 = 3$, $a_2 = 1$, and $a_3 = 2$.

In the second query of the first test case, the longest **polygonal** segment is $[1, 4]$. You can form a quadrilateral with side lengths of $a_1 = 3$, $a_2 = 1$, $a_3 = 2$, and $a_4 = 2$.



An example of a quadrilateral with side lengths of 3, 1, 2, and 2.