A. Permutation Warm-Up

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

For a permutation p of length n*, we define the function:

$$f(p) = \sum_{i=1}^{n} |p_i - i|$$

You are given a number n. You need to compute how many **distinct** values the function $f(p)$ can take when considering **all possible** permutations of the numbers from 1 to n.

*A permutation of length n is an array consisting of n distinct integers from 1 to n in arbitrary order. For example, $[2, 3, 1, 5, 4]$ is a permutation, but $[1, 2, 2]$ is not a permutation (2 appears twice in the array), and $[1, 3, 4]$ is also not a permutation ($n = 3$ but there is 4 in the array).

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.

The first line of each test case contains an integer n ($1 \le n \le 500$) — the number of numbers in the permutations.

Output

For each test case, output a single integer — the number of distinct values of the function $f(p)$ for the given length of permutations.

Standard Input	Standard Output
5 2 3 8 15 43	2 3 17 57 463

Note

Consider the first two examples of the input.

For $n = 2$, there are only 2 permutations — $[1, 2]$ and $[2, 1]$. $f([1, 2]) = |1 - 1| + |2 - 2| = 0$, $f([2, 1]) = |2 - 1| + |1 - 2| = 1 + 1 = 2$. Thus, the function takes 2 distinct values.

For $n = 3$, there are already 6 permutations: $[1, 2, 3]$, $[1, 3, 2]$, $[2, 1, 3]$, $[2, 3, 1]$, $[3, 1, 2]$, $[3, 2, 1]$, the function values of which will be $0, 2, 2, 4, 4$, and 4 respectively, meaning there are a total of 3 values.

,

B. SUMdamental Decomposition

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

On a recent birthday, your best friend Maurice gave you a pair of numbers n and x, and asked you to construct an array of **positive** numbers a of length n such that $a_1 \oplus a_2 \oplus \cdots \oplus a_n = x$ *.

This task seemed too simple to you, and therefore you decided to give Maurice a return gift by constructing an array among all such arrays that has the smallest sum of its elements. You immediately thought of a suitable array; however, since writing it down turned out to be too time-consuming, Maurice will have to settle for just the sum of its elements.

* $\oplus$ denotes the bitwise XOR operation.

Input

Each test contains multiple test cases. The first line contains the number of test cases t ($1 \le t \le 10^4$). The description of the test cases follows.

Each test case consists of a single line containing a pair of numbers n and x ($1 \le n \le 10^9,\ 0 \le x \le 10^9$) — the numbers given to you by Maurice.

Output

For each test case, output your gift to Maurice — the sum of the elements of the array that satisfies all the described properties. If a suitable array does not exist, output -1.

Standard Input	Standard Output
8 2 1 3 6 1 0 2 0 5 0 2 27 15 43 12345678 9101112	5 8 -1 2 8 27 55 21446778

Note

In the first test case, one of the suitable arrays is $[2, 3]$. It can be shown that it is impossible to achieve a smaller sum of array elements.

In the second case, one of the suitable arrays is $[1, 3, 4]$. It can also be shown that this is the optimal amount.

,

C. Neo's Escape

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

Neo wants to escape from the Matrix. In front of him are n buttons arranged in a row. Each button has a weight given by an integer: $a_1, a_2, \ldots, a_n$.

Neo is immobilized, but he can create and move clones. This means he can perform an unlimited number of actions of the following two types in any order:

1. Create a clone in front of a specific button.
2. Move an existing clone one position to the left or right.

As soon as a clone is in front of another button that has not yet been pressed—regardless of whether he was created or moved — he **immediately** presses it. If the button has already been pressed, a clone does nothing — buttons can only be pressed once.

For Neo to escape, he needs to press **all** the buttons in such an order that the sequence of their weights is **non-increasing** — that is, if $b_1, b_2, \ldots, b_n$ are the weights of the buttons in the order they are pressed, then it must hold that $b_1 \geq b_2 \geq \cdots \geq b_n$.

Your task is to determine the minimum number of clones that Neo needs to create in order to press all the buttons in a valid order.

Input

Each test contains multiple test cases. The first line contains the number of test cases t ($1 \leq t \leq 10^4$). The description of the test cases follows.

The first line of each test case contains one integer n ($1 \leq n \leq 2 \cdot 10^5$) — the number of buttons.

The second line of each test case contains n integers $a_1, a_2, \ldots, a_n$ ($1 \leq a_i \leq 10^9$) — the weights of the buttons.

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

Output

For each test case, output one integer — the minimum number of clones that need to be created to press all the buttons in a valid order.

Standard Input	Standard Output
4 5 4 3 2 1 5 3 1 1 1 6 7 8 1 5 9 2 10 1 7 9 7 1 10 2 10 10 7	2 1 2 3

Note

In the first test case, Neo can act as follows:

1. Create a clone in front of the fifth button (with weight 5).
2. Create a clone in front of the first button (with weight 4).
3. Move the second clone from the first button to the second (with weight 3).
4. Move the second clone from the second button to the third (with weight 2).
5. Move the first clone from the fifth button to the fourth (with weight 1).

Thus, the sequence of button presses will be $5 \to 4 \to 3 \to 2 \to 1$, which meets the requirement. It can be shown that the number of clones created is the smallest possible.

In the second test case, Neo can act as follows:

1. Create a clone in front of the second button (with weight 1).
2. Move the clone from the second button to the third (with weight 1).
3. Move the clone from the third button to the second (already pressed).
4. Move the clone from the second button to the first (with weight 1).

Thus, the sequence of button presses will be $1 \to 1 \to 1$.

,

D. Needle in a Numstack

```
Input file:     standard input
Output file:    standard output
Time limit:     3 seconds
Memory limit:   256 megabytes
```

This is an interactive problem.

You found the numbers k and n in the attic, but lost two arrays A and B.

You remember that:

- $|A| + |B| = n$, the total length of the arrays is n.
- $|A| \geq k$ and $|B| \geq k$, the length of each array is at least k.
- The arrays consist only of numbers from 1 to k.
- If you take any k consecutive elements from array A, they will all be different. Also, if you take any k consecutive elements from array B, they will all be different.

Fortunately, a kind spirit that settled in the attic found these arrays and concatenated them into an array C of length n. That is, the elements of array A were first written into array C, followed by the elements of array B.

You can ask the kind spirit up to 250 questions. Each question contains an index i ($1 \leq i \leq n$). In response, you will receive the i-th element of the concatenated array C.

You need to find the lengths of arrays A and B, or report that it is impossible to determine them uniquely.

Input

Each test contains multiple test cases. The first line contains the number of test cases t ($1 \leq t \leq 300$). The description of the test cases follows.

The only line of each test case contains two integers n and k ($1 \leq k \leq 50$, $2k \leq n \leq 10^6$).

Note that the sum of n across test cases is **not limited**.

Interaction

The interaction for each test case begins with reading the integer n.

Then you can make up to 250 queries.

To make a query, output a string in the format "? x" (without quotes) ($1 \leq x \leq n$). After each query, read an integer — the answer to your query.

If you make too many queries, you will receive a verdict of `Wrong answer`.

To report your answer, output a string in the format "! a b" (without quotes), where a and b are the lengths of arrays A and B that you found, respectively. The answer is not counted when counting the number of queries.

If it is impossible to determine the lengths of the arrays uniquely, output "! -1" (without quotes). Note that if you answer -1 while there is a sequence of at most 250 queries that uniquely determines the lengths of arrays, you will get a `Wrong answer` verdict.

It is guaranteed that there are arrays A and B that do not contradict the statement, for which the interactor output is correct.

The interactor is **not** adaptive, which means that the answer is known before the participant makes queries and does not depend on the queries made by the participant.

If your program makes more than 250 queries, your program should immediately terminate to receive the verdict `Wrong answer`. Otherwise, you can get an arbitrary verdict because your solution will continue to read from a closed stream.

After outputting a query, do not forget to output a newline and flush the output buffer. Otherwise, you will receive a verdict of "`IL`" (`Idleness limit exceeded`). To flush the buffer, 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

Hacks are disabled for this problem.

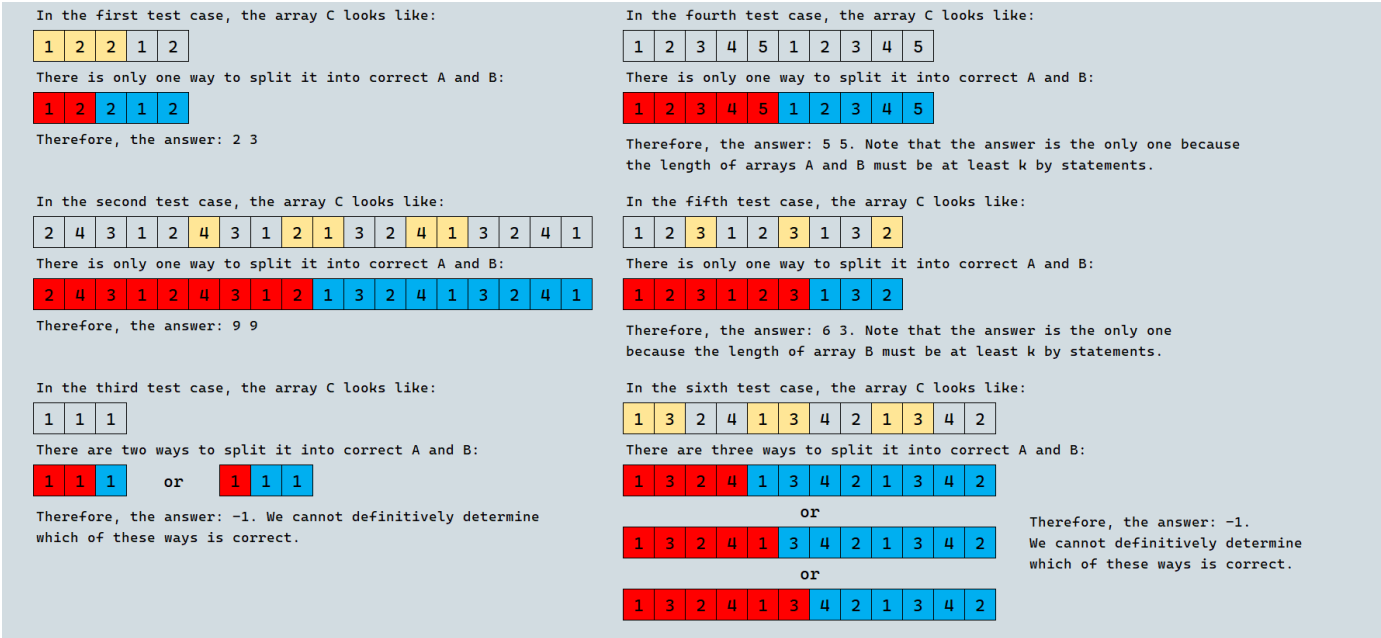
Standard Input	Standard Output
6 5 2 1 2 2 18 4 2 4 1 1 4 3 1 10 5 9 3 3 3	? 1 ? 2 ? 3 ! 2 3 ? 9 ? 13 ? 10 ? 14 ? 6 ! 9 9 ! -1 ! 5 5 ? 3 ? 6 ? 9

2 12 4 1 3 1 3 1 3	! 6 3 ? 1 ? 2 ? 5 ? 6 ? 9 ? 10 ! -1

Note

Consider the first example. We queried the first 3 elements out of 5. Now we know that the array C looks like $[1, 2, 2, ?, ?]$. We know for sure that the third element is not from array A — because according to the condition, any k consecutive elements (in our case $k = 2$) in array A are different. Thus, the third element is definitely located in array B. This means that the length of array A is 2, and the length of array B is 3.

The picture shows arrays from all test cases. The elements whose values were requested are marked in yellow.



,

E. Spruce Dispute

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

It's already a hot April outside, and Polycarp decided that this is the perfect time to finally take down the spruce tree he set up several years ago. As he spent several hours walking around it, gathering his strength, he noticed something curious: the spruce is actually a tree* — and not just any tree, but one consisting of an **odd** number of vertices n. Moreover, on $n-1$ of the vertices hang Christmas ornaments, painted in exactly $\frac{n-1}{2}$ distinct colors, with exactly two ornaments painted in each color. The remaining vertex, as tradition dictates, holds the tree's topper.

At last, after several days of mental preparation, Polycarp began dismantling the spruce. First, he removed the topper and had already started taking apart some branches when suddenly a natural question struck him: how can he remove one of the tree's edges and rearrange the ornaments in such a way that the sum of the lengths of the simple paths between ornaments of the same color is as large as possible?

In this problem, removing an edge from the tree is defined as follows: choose a pair of adjacent vertices a and b ($a < b$), then remove vertex b from the tree and reattach all of b's adjacent vertices (except for a) directly to a.

Polycarp cannot continue dismantling his spruce until he gets an answer to this question. However, checking all possible options would take him several more years. Knowing your experience in competitive programming, he turned to you for help. But can you solve this dispute?

*A tree is a connected graph without cycles.

Input

Each test contains multiple test cases. The first line contains the number of test cases t ($1 \le t \le 10^4$). The description of the test cases follows.

The first line of each test case contains an **odd** number n ($3 \le n < 2 \cdot 10^5$) — the number of vertices in the tree.

The following $n-1$ lines describe the tree, given by pairs of adjacent vertices u, v ($1 \le u, v \le n, u \neq v$).

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

Output

For each test case, you need to output two lines.

In the first line, output the pair of vertices u, v, the edge between which Polycarp is going to remove.

In the next line, output the array c of n numbers from 0 to $\frac{n-1}{2}$, where $c[i]$ — the positive color number assigned to vertex i. Note that $c[\max(u, v)] = 0$, since this vertex has been removed.

Standard Input	Standard Output
3 5 1 2	1 2 2 0 1 1 2 1 5

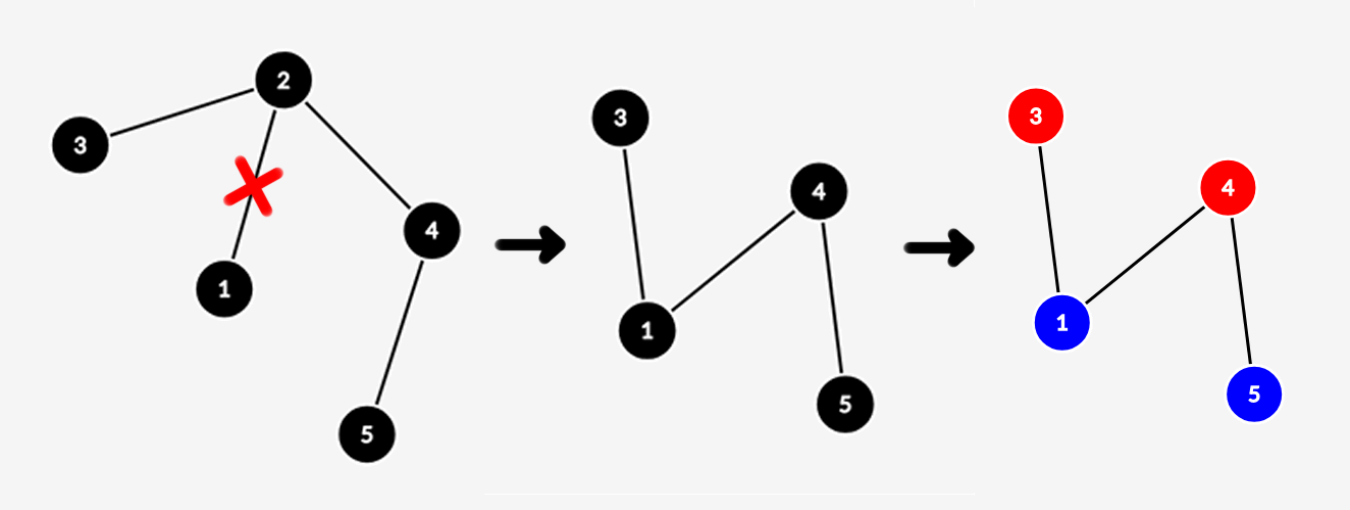
2 3 2 4 4 5 5 1 2 1 3 1 4 1 5 7 1 5 5 4 4 3 3 2 2 6 6 7	1 1 2 2 0 4 3 1 3 3 0 2 2 1

Note

Consider the first test case.

Remove the edge connecting vertices 1 and 2. After this, vertex 2 will be removed from the tree, and vertices 3 and 4 will be connected to vertex 1.

Color vertices 3 and 4 with the first color, and vertices 1 and 5 with the second. The sum of the lengths of simple paths between ornaments of the same color is equal to $2 + 2 = 4$. It can be shown that this value is the largest possible.



In the second and third examples, the maximum sum of path lengths will be 3 and 9, respectively.

,

F. Fallen Towers

Input file: standard input
Output file: standard output
Time limit: 3 seconds
Memory limit: 256 megabytes

Pizano built an array a of n towers, each consisting of $a_i \geq 0$ blocks.

Pizano can knock down a tower so that the next a_i towers grow by 1. In other words, he can take the element a_i, increase the next a_i elements by one, and then set a_i to 0. The blocks that fall outside the array of towers disappear. If Pizano knocks down a tower with 0 blocks, nothing happens.

Pizano wants to knock down all n towers in any order, **each exactly once**. That is, for each i from 1 to n, he will knock down the tower at position i exactly once.

Moreover, the resulting array of tower heights **must be non-decreasing**. This means that after he knocks down all n towers, for any $i < j$, the tower at position i must not be taller than the tower at position j.

You are required to output the maximum MEX of the resulting array of tower heights.

The MEX of an array is the smallest non-negative integer that is not present in the array.

Input

Each test contains multiple test cases. The first line contains the number of test cases t ($1 \leq t \leq 10^4$). The description of the test cases follows.

The first line of each test case contains an integer n ($1 \leq n \leq 10^5$) — the number of towers.

The second line of each test case contains n integers — the initial heights of the towers $a_1, \ldots, a_n$ ($0 \leq a_i \leq 10^9$).

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

Output

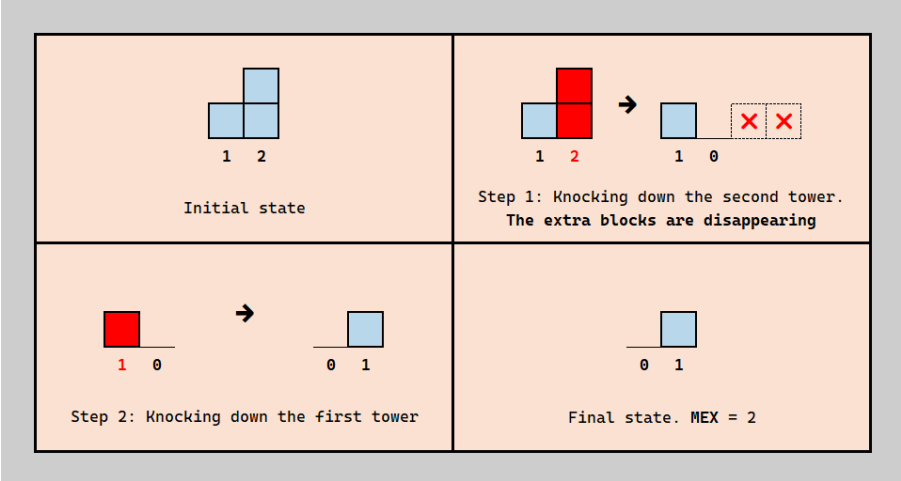
For each test case, output a single integer — the maximum MEX of the final array.

Standard Input	Standard Output
8 2 1 2 4 2 1 0 0 10 5 9 3 7 1 5 1 5 4 3 10 1 1 1 1 1 1 1 1 1 1 10 3 2 1 0 3 2 1 0 3 2 5 5 2 0 5 5 1	2 3 7 4 5 4 1 3

1000000000 7 4 0 1 0 2 7 7	

Note

Explanation for the first test case.



Explanation for the second test case. Note that all towers were knocked down exactly once, and the final array of heights is non-decreasing.