Atto Round 1 (Codeforces Round 1041, Div. 1 + Div. 2)

A. Mix Mex Max

1 second, 256 megabytes

You are given an array a consisting of n non-negative integers. However, some elements of a are missing, and they are represented by -1.

We define that the array a is *good* if and only if the following holds for every $1 \le i \le n-2$:

$$\operatorname{mex}([a_i, a_{i+1}, a_{i+2}]) = \max([a_i, a_{i+1}, a_{i+2}]) - \min([a_i, a_{i+1}, a_{i+2}]),$$

where $\operatorname{mex}(b)$ denotes the minimum excluded (MEX)[*] of the integers in b.

You have to determine whether you can make a *good* after replacing each -1 in a with a non-negative integer.

[*]The minimum excluded (MEX) of a collection of integers $b_1, b_2, \ldots, b_k$ is defined as the smallest non-negative integer x which does not occur in the collection b. For example, $\operatorname{mex}([2,2,1]) = 0$ because 0 does not belong to the array, and $\operatorname{mex}([0,3,1,2]) = 4$ because 0, 1, 2, and 3 appear in the array, but 4 does not.

Input

Each test contains multiple test cases. The first line contains the number of test cases t ($1 \le t \le 500$). The description of the test cases follows.

The first line of each test case contains a single integer n ($3 \le n \le 100$) — the length of a.

The second line contains n integers $a_1, a_2, \ldots, a_n$ ($-1 \le a_i \le 100$) — the elements of a. $a_i = -1$ denotes that this element is missing.

Output

For each test case, output "YES" if it is possible to make a *good*, and "NO" otherwise.

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.

input

```
8
3
-1 -1 -1
5
1 1 1 1 0
6
5 5 1 -1 -1 1
4
-1 -1 0 -1
4
-1 1 1 -1
3
3 3 -1
5
0 0 0 0 0
7
3 0 1 4 -1 2 3
```

output

```
YES
NO
NO
NO
YES
YES
NO
NO
```

In the first test case, we can put $a_1 = a_2 = a_3 = 1$. Then,

- $\operatorname{mex}([a_1, a_2, a_3]) = \operatorname{mex}([1,1,1]) = 0$;
- $\max([a_1, a_2, a_3]) = \max([1,1,1]) = 1$;
- $\min([a_1, a_2, a_3]) = \min([1,1,1]) = 1$.

And $0 = 1 - 1$. Thus, the array a is *good*.

In the second test case, none of the elements in a is missing. And we have

- $\operatorname{mex}([a_1, a_2, a_3]) = \max([a_1, a_2, a_3]) - \min([a_1, a_2, a_3])$,
- $\operatorname{mex}([a_2, a_3, a_4]) = \max([a_2, a_3, a_4]) - \min([a_2, a_3, a_4])$, but
- $\operatorname{mex}([a_3, a_4, a_5]) \ne \max([a_3, a_4, a_5]) - \min([a_3, a_4, a_5])$.

Thus, the array a cannot be *good*.

In the third test case, none of a_1, a_2, or a_3 is missing. However,

- $\operatorname{mex}([a_1, a_2, a_3]) = \operatorname{mex}([5,5,1]) = 0$;
- $\max([a_1, a_2, a_3]) = \max([5,5,1]) = 5$;
- $\min([a_1, a_2, a_3]) = \min([5,5,1]) = 1$.

And $0 \ne 5 - 1$. So the array a cannot be *good*, no matter how you replace the missing elements.

B. Hamiiid, Haaamid... Hamid?

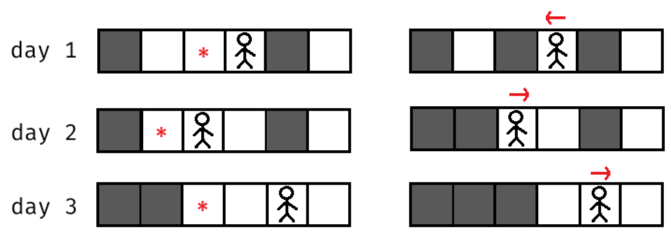
1 second, 256 megabytes

Mani has locked Hamid in a $1 \times n$ grid. Initially, some cells of the grid contain walls and the rest are empty, and Hamid is in an empty cell.

In each day, the following events happen in order:

1. Mani selects an empty cell and builds a wall in that cell. Note that he can **not** build a wall in the cell which Hamid currently is at;
2. Hamid selects a direction (left or right), then
   - If there are no walls in that direction, he will escape the grid;
   - Otherwise, he will move to the nearest wall in that direction and destroy that wall. Hamid is at the position of the destroyed wall after this day.

Here is an example of a possible sequence of actions when $n = 6$:



Hamid is **always** aware of where the walls are. He wants to minimize the number of days that he needs to escape the grid, while Mani wants to maximize it.

You have to determine the number of days Hamid needs to escape the grid if they both act optimally.

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 two integers n and x ($2 \le n \le 2 \cdot 10^5, 1 \le x \le n$) — the size of the grid and the initial position of Hamid. He is at the x-th cell from left to right initially.

The second line contains a string s of length n ($s_i =$ "#" or ".") — the initial state of the grid. The i-th cell of the grid contains a wall if $s_i =$ "#", and it is empty if $s_i =$ ".".

It is guaranteed that the x-th cell is empty, and there are at least two empty cells in the grid.

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 a single integer — the number of days Hamid needs to escape the grid if they both act optimally.

input

```
4
3 1
..#
4 2
....
5 3
##..#
6 4
#...#.
```

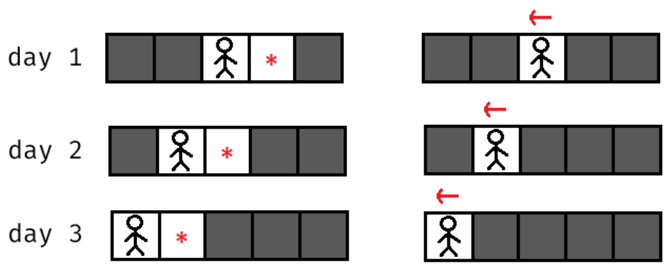
output

```
1
1
3
3
```

In the first test case, Mani must build a wall in cell 2, so Hamid can escape from the left side of the grid on the first day.

In the second test case, if Mani places the wall to the left of Hamid, Hamid can escape from the right. And if the wall is to Hamid's right, he can escape from the left. Thus, the answer is 1.

In the third test case:



It can be shown that both players acted optimally in the above illustration.

In the fourth test case, we have shown an example of actions in the statements. Note that the players did **not** act optimally in this example.

C. Trip Shopping

2 seconds, 256 megabytes

Ali and Bahamin decided to spend their summer vacation on the beautiful southern coasts of Iran. They also agreed to do some shopping during the trip — but instead of setting a fixed budget, they decided to determine how much they would spend by playing a game.

The game is played on two arrays a and b, each containing n integers.

The game will last for k rounds. In one round:

- First, Ali selects two indices i and j ($1 \le i < j \le n$);
- Then, Bahamin rearranges the four integers a_i, a_j, b_i, and b_j **arbitrarily**. Note that Bahamin can swap numbers between two arrays. He can also keep the two arrays unchanged.

After all the k rounds, the *value* of the game is defined as $v = \sum_{i=1}^{n} |a_i - b_i|$. Ali and Bahamin will spend exactly v coins during their trip.

However, their goals are quite different:

- Ali wants to spend as little as possible, that is, to minimize v;
- Bahamin wants to spend as much as possible, that is, to maximize v.

You have to find the final amount of coins they will spend if both Ali and Bahamin play optimally.

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 two integers n and k ($2 \le n \le 2 \cdot 10^5, 1 \le k \le n$) — the length of a and b, and the number of rounds.

The second line contains n integers $a_1, a_2, \dots, a_n$ ($1 \le a_i \le 10^9$) — the elements of a.

The third line contains n integers $b_1, b_2, \dots, b_n$ ($1 \le b_i \le 10^9$) — the elements of b.

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 a single integer — the final amount of coins they will spend if both Ali and Bahamin play optimally.

input

```
5
2 1
1 7
3 5
3 2
1 5 3
6 2 4
5 4
1 16 10 10 16
3 2 2 15 15
4 1
23 1 18 4
19 2 10 3
10 10
4 3 2 100 4 1 2 4 5 5
1 200 4 5 6 1 10 2 3 4
```

output

```
8
9
30
16
312
```

In the first test case, Ali can only choose $(i, j) = (1, 2)$, and Bahamin can rearrange all four numbers. Thus, he can assign $a = [5, 1]$ and $b = [3, 7]$. And the *value* of the game will be $v = |5 - 3| + |1 - 7| = 8$. It can be shown that this is the maximum possible *value* reachable for Bahamin — Other arrangements like $a = [5, 7],\ b = [1, 3]$ are also possible, but they don't have larger values.

In the second test case, the best strategy for Bahamin is to keep the two arrays unchanged, regardless of what indices Ali selects. And the *value* of the game will be $v = |1 - 6| + |5 - 2| + |3 - 4| = 9$.

D. Root was Built by Love, Broken by Destiny

2 seconds, 256 megabytes

Heartfall River runs horizontally through Destinyland and divides it into the northern and southern sides.

Engineer Root wants to build n houses along the river, numbered from 1 to n. All houses on the northern side and all houses on the southern side must lie along straight lines parallel to Heartfall River.

There will be m bridges, with the i-th bridge connecting house u_i and house v_i ($u_i \ne v_i$). It is guaranteed that all n houses are connected by these bridges, that is, you can travel from any house to any other by crossing bridges. Also, there are no two bridges connecting the same pair of houses.

Root wants to know how many ways there are to arrange the n houses along the river, modulo $10^9 + 7$, such that the following conditions hold for the planned m bridges:

- For every bridge, the two houses it connects lie on opposite sides of the river;
- The bridges do not cross when drawn as straight lines between the houses.

A possible arrangement of the houses when $n = 5$.

Two arrangements are considered different if at least one of the following conditions holds:

- There exists a house that lies on a different side in each arrangement;
- There exist two houses a and b that are on the same side in both arrangements, but a comes before b in one arrangement and b comes before a in the other.

Since Root is distracted by his ex, whom destiny separated from him, he asks you to calculate the number of ways to arrange the houses along the river, modulo $10^9 + 7$.

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 two integers n and m ($2 \le n \le 2 \cdot 10^5$, $n - 1 \le m \le \min\left(\frac{n(n-1)}{2}, 2 \cdot 10^5\right)$) — the number of houses and the number of bridges.

Then m lines follow, the i-th line containing two integers u_i and v_i ($1 \le u_i, v_i \le n$, $u_i \ne v_i$) — the two houses that the i-th bridge connects.

It is guaranteed that all the n houses are connected by the bridges, and there are no two bridges connecting the same pair of houses.

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

Output

For each test case, output a single integer — the number of ways to arrange the n houses along the river, modulo $10^9 + 7$.

input
```
4
2 1
1 2
3 3
1 2
1 3
2 3
5 4
1 2
1 3
3 4
3 5
4 3
1 2
1 3
1 4
```
output
```
2
0
8
12
```

In the first test case, either house 1 should be built on the northern side and house 2 on the southern side, or vice versa.

In the second test case, at least two houses must be built on the same side of the river. But every pair of houses is connected by a bridge. So in every arrangement, at least one bridge will not cross the river. Thus, the answer is 0.

In the third test case, one of the possible arrangements of the houses is provided in the problem statement.

E. Ancient Tree

4 seconds, 256 megabytes

Bahamin came from the past to visit Ali — who came from the future. He also brought an ancient tree as a gift for Ali. He noticed some of its vertices have lost their color. Bahamin needs to repaint these vertices, but he is very busy with fixing his time machine. Fortunately (or unfortunately), dinosaurs now handle such tasks — for a fee, of course. He needs your help to find the coloring with minimum cost. So he gives you the problem as follows.

You are given a rooted tree[*] of n vertices, where vertex 1 is the root. Each vertex has an integer weight w_i and a color c_i, where the colors are integers between 1 and k. However, some vertices have lost their colors, represented by $c_i = 0$.

We call vertex v *cutie* if there **exists** two vertices x and y, such that

- $\operatorname{lca}(x, y)^{\dagger} = v$,
- $c_x = c_y$, and
- $c_x \ne c_v$.

The *cost* of the tree is the sum of weights of all *cutie* vertices.

You have to assign colors between 1 and k to all the vertices which have lost their colors. Find the minimum possible *cost* among all valid colorings and provide a coloring with the minimum possible *cost*.

[*] A tree is a connected graph without cycles. A rooted tree is a tree where one vertex is special and called the root.

[†] $\operatorname{lca}(x, y)$ denotes the lowest common ancestor (LCA) of x and y.

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 two integers n and k ($3 \le n \le 2 \cdot 10^5$, $2 \le k \le n$) — the number of vertices and the number of colors.

The second line contains n integers $w_1, w_2, \ldots, w_n$ ($1 \le w_i \le 10^9$) — the weight of vertices.

The third line contains n integers $c_1, c_2, \ldots, c_n$ ($0 \le c_i \le k$) — the color of vertices. $c_i = 0$ means that vertex i has lost its color.

Then $n - 1$ lines follow, the i-th line containing two integers u and v ($1 \le u, v \le n$) — the two vertices that the i-th edge connects.

It is guaranteed that the given edges form a tree.

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

Output

For each test case, in the first line output a single integer — the minimum possible *cost* among all valid colorings.

In the second line output n integers $c'_1, c'_2, \ldots, c'_n$ — a coloring with the minimum possible *cost*. You need to guarantee that:

- $c'_i = c_i$ if $c_i \ne 0$;
- $1 \le c'_i \le k$ if $c_i = 0$.

If there are multiple colorings with the minimum possible *cost*, you can output any of them.

input

```
4
4 4
5 5 5 5
1 0 2 3
1 2
1 3
1 4
5 2
3 1 4 1 5
1 2 1 2 2
1 4
2 1
3 4
4 5
11 3
3 1 4 3 1 4 3 1 4 5 6
0 0 0 2 1 2 1 2 2 1 1
1 2
2 3
2 4
2 5
2 6
1 7
7 8
7 9
10 3
3 11
4 3
2 3 2 3
2 1 0 0
3 1
1 2
2 4
```

output

```
0
1 4 2 3
3
1 2 1 2 2
7
2 3 1 2 1 2 1 2 2 1 1
0
2 1 3 1
```

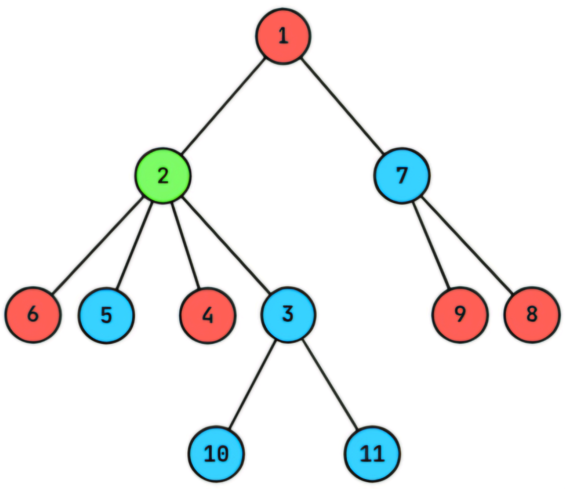
In the first test case, there are four choices for the missing color:

- $c_2 = 1$ makes no vertex *cutie*, so the *cost* is 0;
- $c_2 = 2$ makes vertex 1 *cutie*, since $c_2 = c_3 = 2$, $\operatorname{lca}(2, 3) = 1$ and $c_1 \neq 2$. Thus, the *cost* is $w_1 = 5$;
- $c_2 = 3$ makes vertex 1 *cutie*, since $c_2 = c_4 = 3$, $\operatorname{lca}(2, 4) = 1$ and $c_1 \neq 3$. Thus, the *cost* is $w_1 = 5$;
- $c_2 = 4$ makes no vertex *cutie*, so the *cost* is 0.

Thus, the minimum possible *cost* among different colorings is 0.

In the second test case, every vertex has a color. So current *cost* is not changeable. And since we have $c_5 = c_2 = 2$, $\operatorname{lca}(2, 5) = 1$ and $c_1 \neq 2$, vertex 1 is *cutie* and current *cost* is equal to $w_1 = 3$.

In the third test case, a possible coloring with the minimum cost is shown below:



Some other valid colorings are:

- $c = [3, 1, 2, 2, 1, 2, 1, 2, 2, 1, 1]$ which makes vertices $1, 2, 3, 7$ *cutie*:
  - $\operatorname{lca}(4, 8) = 1$;
  - $\operatorname{lca}(3, 4) = 2$;
  - $\operatorname{lca}(10, 11) = 3$;
  - $\operatorname{lca}(8, 9) = 7$.

  All pairs of vertices mentioned have a different color from their LCA. So the *cost* will be $w_1 + w_2 + w_3 + w_7 = 11$.
- $c = [3, 2, 1, 2, 1, 2, 1, 2, 2, 1, 1]$ which makes vertices $1, 2, 7$ *cutie*:
  - $\operatorname{lca}(5, 7) = 1$;
  - $\operatorname{lca}(5, 11) = 2$;
  - $\operatorname{lca}(8, 9) = 7$.

  All pairs of vertices mentioned have a different color from their LCA. So the *cost* will be $w_1 + w_2 + w_7 = 7$.

It can be shown that no coloring with *cost* smaller than 7 exists.

F. Hamed and AghaBalaSar

2 seconds, 512 megabytes

Hamid wrote 3014 lines of code for the a+b problem and gave 10 lines of it to Hamed; then the following problem came to Hamed's mind.

You are given two integers n and m.

An array $a_1, a_2, \ldots, a_n$ is called *snake* if and only if all of the following conditions hold:

- All elements in a are integers between 0 and m;
- $a_1 + a_2 + \cdots + a_n = m$;
- $a_n = \max\left([a_1, a_2, \ldots, a_n]\right)$.

We define $f(a)$ as in the following pseudocode:

```
function f(array a):
    pos := 1
    res := 0
    let nxt[x] be an array such that nxt[x] is the smallest
index y such that
                                         y > x and a[y] > a[x],
                               or undefined if no such y
exists
    while pos < n:
        if a[pos] < a[n]:
            res += a[nxt[pos]] - a[pos]
            pos := nxt[pos]
        else:
            pos += 1
    return res
```

Find the sum of $f(a)$ over all *snake* arrays $a_1, a_2, \ldots, a_n$, modulo $10^9 + 7$.

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 two integers n and m ($2 \le n \le 2 \cdot 10^5, 0 \le m \le 2 \cdot 10^5$) — the length of a *snake* array and the sum of all elements in a *snake* array.

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

Output

For each test case, output a single integer — the sum of $f(a)$ over all *snake* arrays $a_1, a_2, \ldots, a_n$, modulo $10^9 + 7$.

input

```
8
2 5
3 4
4 6
5 10
6 23
100 100
142857 33333
200000 0
```

output

```
9
14
76
985
142112
771227753
865580631
0
```

In the first test case, there are three *snake* arrays:

- $f([0, 5]) = 5$;
- $f([1, 4]) = 3$;
- $f([2, 3]) = 1$.

Thus, the answer is $5 + 3 + 1 = 9$.

In the second test case, there are six *snake* arrays:

- $f([0, 0, 4]) = 4$;
- $f([0, 1, 3]) = 3$;
- $f([1, 0, 3]) = 2$;
- $f([1, 1, 2]) = 1$;
- $f([0, 2, 2]) = 2$;
- $f([2, 0, 2]) = 2$.

Thus, the answer is $4 + 3 + 2 + 1 + 2 + 2 = 14$.

In the fifth test case, a possible *snake* array is:

- $f([3, 1, 4, 1, 5, 9]) = 6$.

G1. Inter Active (Easy Version)

2 seconds, 512 megabytes

This is the easy version of the problem. The difference between the versions is that in this version, you can make at most $15 \cdot n$ queries. You can hack only if you solved all versions of this problem.

Ali loved Bahamin's gift (from problem E) so much that he illegally traveled from Qazvin to Liverpool to have the gift signed by football players. Now Interpol is searching for him, but they've offered a deal: solve a problem, and he can stay in Liverpool. But since he's currently at the stadium, he can't solve it — so he asked you to do it.

This is an interactive problem.

There is a hidden permutation* p of length $n \geq 4$, where $p_i \neq i$ for each $1 \leq i \leq n$.

Initially, you should give the jury a positive integer $k \leq n$, which will be **constant** through future queries. Then you need to find permutation p using some queries.

In each query, you give a permutation $q_1, q_2, \ldots, q_n$ to the jury. In response, you will receive the number of pairs (i, j) such that all of the following conditions hold:

- $i < j$;
- $p_{q_i} = q_j$;
- $i \neq k$. (k is the constant you have given to the jury)

You are given n, and you need to find the permutation p in at most $15 \cdot n$ queries.

*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 ($1 \leq t \leq 500$). The description of the test cases follows.

The only line of each test case contains a single integer n ($4 \leq n \leq 100$) — the length of p.

It is guaranteed that the sum of n^2 over all test cases does not exceed 10^4.

Interaction

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

Then you should output the integer k ($1 \leq k \leq n$). This is not considered as a query.

Then you can ask up to $15 \cdot n$ queries.

To make a query, output a line in the following format:

- **?** $q_1\ q_2\ \ldots\ q_n$

The jury will return the answer to the query.

When you find the permutation p, output a single line in the following format:

- **!** $p_1\ p_2\ \ldots\ p_n$

This is also not considered as a query.

After that, proceed to process the next test case or terminate the program if it is the last test case.

The interactor is **not** adaptive, which means that the permutation is determined before the participant outputs k.

If your program makes more than $15 \cdot n$ 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 printing each query do not forget to output the end of line and flush* the output. Otherwise, you will get `Idleness limit exceeded` verdict.

If, at any interaction step, you read -1 instead of valid data, your solution must exit immediately. This means that your solution will receive `Wrong answer` because of an invalid query or any other mistake. Failing to exit can result in an arbitrary verdict because your solution will continue to read from a closed stream.

Hacks

To perform a hack, use the following format:

Each test contains multiple test cases. The first line contains the number of test cases t ($1 \leq t \leq 500$). The description of the test cases follows.

The first line of each test case contains a single integer n ($4 \leq n \leq 100$) — the length of p.

The second line contains n integers $p_1, p_2, \ldots, p_n$ ($1 \leq p_i \leq n$, $p_i \neq i$, all p_i-s are distinct) — the permutation p.

It is guaranteed that the sum of n^2 over all test cases does not exceed 10^4.

*To flush, use:

- `fflush(stdout)` or `cout.flush()` in C++;
- `sys.stdout.flush()` in Python;
- see the documentation for other languages.

input

```
2
4


1

5


1

2
```

output

```

1
? 1 2 3 4

! 3 1 4 2

3
? 1 2 5 4 3

? 2 1 4 3 5

! 3 1 2 5 4
```

In the first test case, $p = [3, 1, 4, 2]$. The solution selected $k = 1$, then it asked permutation $q = [1, 2, 3, 4]$. Only pair $(3, 4)$ satisfies the conditions.

In the second test case, $p = [3, 1, 2, 5, 4]$. The solution selected $k = 3$.

- For permutation $q = [1, 2, 5, 4, 3]$, only pair $(1, 5)$ satisfies the conditions.
- For permutation $q = [2, 1, 4, 3, 5]$, pairs $(1, 2)$ and $(2, 4)$ satisfy the conditions.

G2. Inter Active (Hard Version)

2 seconds, 512 megabytes

This is the hard version of the problem. The difference between the versions is that in this version, you can make at most $10 \cdot n$ queries. You can hack only if you solved all versions of this problem.

Ali loved Bahamin's gift (from problem E) so much that he illegally traveled from Qazvin to Liverpool to have the gift signed by football players. Now Interpol is searching for him, but they've offered a deal: solve a problem, and he can stay in Liverpool. But since he's currently at the stadium, he can't solve it — so he asked you to do it.

This is an interactive problem.

There is a hidden permutation* p of length $n \geq 4$, where $p_i \neq i$ for each $1 \leq i \leq n$.

Initially, you should give the jury a positive integer $k \leq n$, which will be **constant** through future queries. Then you need to find permutation p using some queries.

In each query, you give a permutation $q_1, q_2, \ldots, q_n$ to the jury. In response, you will receive the number of pairs (i, j) such that all of the following conditions hold:

- $i < j$;
- $p_{q_i} = q_j$;
- $i \neq k$. (k is the constant you have given to the jury)

You are given n, and you need to find the permutation p in at most $10 \cdot n$ queries.

*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 ($1 \leq t \leq 500$). The description of the test cases follows.

The only line of each test case contains a single integer n ($4 \leq n \leq 100$) — the length of p.

It is guaranteed that the sum of n^2 over all test cases does not exceed 10^4.

Interaction

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

Then you should output the integer k ($1 \leq k \leq n$). This is not considered as a query.

Then you can ask up to $10 \cdot n$ queries.

To make a query, output a line in the following format:

- **?** $q_1\ q_2\ \ldots\ q_n$

The jury will return the answer to the query.

When you find the permutation p, output a single line in the following format:

- **!** $p_1\ p_2\ \ldots\ p_n$

This is also not considered as a query.

After that, proceed to process the next test case or terminate the program if it is the last test case.

The interactor is **not** adaptive, which means that the permutation is determined before the participant outputs k.

If your program makes more than $10 \cdot n$ 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 printing each query do not forget to output the end of line and flush* the output. Otherwise, you will get `Idleness limit exceeded` verdict.

If, at any interaction step, you read -1 instead of valid data, your solution must exit immediately. This means that your solution will receive `Wrong answer` because of an invalid query or any other mistake. Failing to exit can result in an arbitrary verdict because your solution will continue to read from a closed stream.

Hacks

To perform a hack, use the following format:

Each test contains multiple test cases. The first line contains the number of test cases t ($1 \leq t \leq 500$). The description of the test cases follows.

The first line of each test case contains a single integer n ($4 \leq n \leq 100$) — the length of p.

The second line contains n integers $p_1, p_2, \ldots, p_n$ ($1 \leq p_i \leq n$, $p_i \neq i$, all p_i-s are distinct) — the permutation p.

It is guaranteed that the sum of n^2 over all test cases does not exceed 10^4.

*To flush, use:

- `fflush(stdout)` or `cout.flush()` in C++;
- `sys.stdout.flush()` in Python;
- see the documentation for other languages.

input

```
2
4


1

5



1

2
```

output

```

1
? 1 2 3 4

! 3 1 4 2

3
? 1 2 5 4 3

? 2 1 4 3 5

! 3 1 2 5 4
```

In the first test case, $p = [3, 1, 4, 2]$. The solution selected $k = 1$, then it asked permutation $q = [1, 2, 3, 4]$. Only pair $(3, 4)$ satisfies the conditions.

In the second test case, $p = [3, 1, 2, 5, 4]$. The solution selected $k = 3$.

- For permutation $q = [1, 2, 5, 4, 3]$, only pair $(1, 5)$ satisfies the conditions.
- For permutation $q = [2, 1, 4, 3, 5]$, pairs $(1, 2)$ and $(2, 4)$ satisfy the conditions.

H. 23 Rises Again

5 seconds, 1024 megabytes

Kiarash is picking strawberries to take home...

A graph is called *candy* if and only if the degree of every vertex in it is at most 2.

You are given a simple, undirected, and connected graph G of $n \le 30$ vertices, with a special property: **each vertex belongs to at most 5 simple cycles**[*].

What is the maximum number of edges among all subgraphs[†] of G that are *candy*?

[*] A simple cycle is a connected subgraph such that each vertex has a degree of exactly 2

[†] A subgraph of G is a graph whose vertices and edges are subsets of G.

Input

Each test contains multiple test cases. The first line contains the number of test cases t ($1 \le t \le 50$). The description of the test cases follows.

The first line of each test case contains two integers n and m ($3 \le n \le 30, n - 1 \le m \le \frac{n(n-1)}{2}$) — the number of vertices and the number of edges.

Then m lines follow, the i-th line containing two integers u and v ($1 \le u, v \le n$) — the two vertices that the i-th edge connects.

It is guaranteed that the given graph is simple and connected, and each vertex belongs to at most 5 simple cycles.

It is guaranteed that the sum of n^2 over all test cases does not exceed 900.

Output

For each test case, output a single integer — the maximum number of edges among all subgraphs that are *candy graphs*.

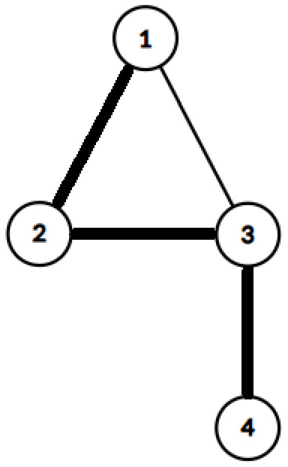
input

```
3
4 4
1 2
1 3
2 3
3 4
7 10
1 2
1 3
1 4
2 4
3 4
4 5
4 6
5 6
5 7
6 7
9 10
1 2
1 3
3 4
3 7
4 5
4 6
5 6
7 8
7 9
8 9
```

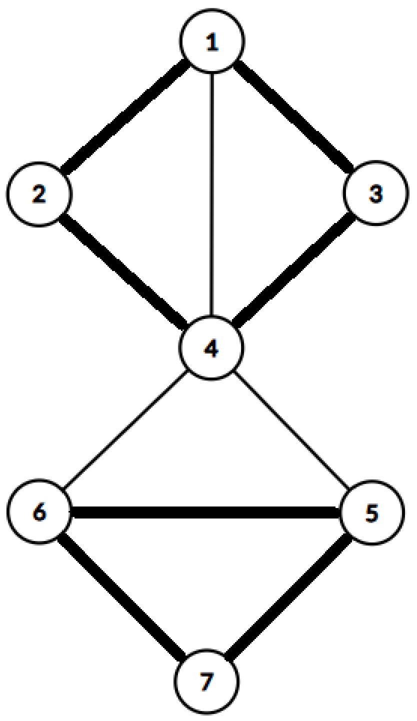
output

```
3
7
8
```

In the first test case, you can select the edges marked in the image below. On the other hand, you can't select all the edges because the degree of vertex 3 would exceed 2. So the maximum number of edges among all subgraphs that are *candy* is 3.



In the second test case, you can select the edges marked in the image below. It can be proven that any subgraph that is *candy* has at most 7 edges.



In the third test case, the image below shows one of the subgraphs with the maximum possible number of edges that is *candy*.