

Codeforces Round 967 (Div. 2)

A. Make All Equal

1 second, 256 megabytes

You are given a cyclic array a_1, a_2, \dots, a_n .

You can perform the following operation on a at most $n - 1$ times:

- Let m be the current size of a , you can choose any two adjacent elements where the previous one is no greater than the latter one (In particular, a_m and a_1 are adjacent and a_m is the previous one), and delete exactly one of them. In other words, choose an integer i ($1 \leq i \leq m$) where $a_i \leq a_{(i \bmod m)+1}$ holds, and delete exactly one of a_i or $a_{(i \bmod m)+1}$ from a .

Your goal is to find the minimum number of operations needed to make all elements in a equal.

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 first line of each test case contains a single integer n ($1 \leq n \leq 100$) — the length of the array a .

The second line of each test case contains n integers a_1, a_2, \dots, a_n ($1 \leq a_i \leq n$) — the elements of array a .

Output

For each test case, output a single line containing an integer: the minimum number of operations needed to make all elements in a equal.

input
7
1
1
3
1 2 3
3
1 2 2
5
5 4 3 2 1
6
1 1 2 2 3 3
8
8 7 6 3 8 7 6 3
6
1 1 4 5 1 4
output
0
2
1
4
4
6
3

In the first test case, there is only one element in a , so we can't do any operation.

In the second test case, we can perform the following operations to make all elements in a equal:

- choose $i = 2$, delete a_3 , then a would become $[1, 2]$.
- choose $i = 1$, delete a_1 , then a would become $[2]$.

It can be proven that we can't make all elements in a equal using fewer than 2 operations, so the answer is 2.

The problem statement has recently been changed. [View the changes.](#)

B. Generate Permutation

1.5 seconds, 256 megabytes

There is an integer sequence a of length n , where each element is initially -1 .

Misuki has two typewriters where the first one writes letters from left to right, with a pointer initially pointing to 1, and another writes letters from right to left with a pointer initially pointing to n .

Misuki would choose one of the typewriters and use it to perform the following operations until a becomes a permutation of $[1, 2, \dots, n]$

- write number: write the minimum **positive** integer that hasn't appeared in a to a_i , i is the position where the pointer points at. Such operation can be performed only when $a_i = -1$.
- carriage return: return the pointer to its initial position (i.e. 1 for the first typewriter, n for the second)
- move pointer: move the pointer to the next position, let i be the position the pointer points at before this operation, if Misuki is using the first typewriter, $i := i + 1$ would happen, and $i := i - 1$ otherwise. Such operation can be performed only if after the operation, $1 \leq i \leq n$ holds.

Your task is to construct any permutation p of length n , such that the minimum number of carriage return operations needed to make $a = p$ is the same no matter which typewriter Misuki is using.

Input

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

The first line of each test case contains a single integer n ($1 \leq n \leq 2 \cdot 10^5$) — the length of the permutation.

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 line of n integers, representing the permutation p of length n such that the minimum number of carriage return operations needed to make $a = p$ is the same no matter which typewriter Misuki is using, or -1 if it is impossible to do so.

If there are multiple valid permutations, you can output any of them.

input
3
1
2
3

output

```
1
-1
3 1 2
```

In the first testcase, it's possible to make $a = p = [1]$ using 0 carriage return operations.

In the second testcase, it is possible to make $a = p = [1, 2]$ with the minimal number of carriage returns as follows:

If Misuki is using the first typewriter:

- Write number: write 1 to a_1 , a becomes $[1, -1]$
- Move pointer: move the pointer to the next position. (i.e. 2)
- Write number: write 2 to a_2 , a becomes $[1, 2]$

If Misuki is using the second typewriter:

- Move pointer: move the pointer to the next position. (i.e. 1)
- Write number: write 1 to a_1 , a becomes $[1, -1]$
- Carriage return: return the pointer to 2.
- Write number: write 2 to a_2 , a becomes $[1, 2]$

It can be proven that the minimum number of carriage returns needed to transform a into p when using the first typewriter is 0 and it is 1 when using the second one, so this permutation is not valid.

Similarly, $p = [2, 1]$ is also not valid, so there is no solution for $n = 2$.

In the third testcase, it is possible to make $a = p = [3, 1, 2]$ with 1 carriage return with both the first and the second typewriter. It can be proven that, for both typewriters, it is impossible to write p with 0 carriage returns.

With the first typewriter it is possible to:

- Move pointer: move the pointer to the next position. (i.e. 2)
- Write number: write 1 to a_2 , a becomes $[-1, 1, -1]$
- Move pointer: move the pointer to the next position. (i.e. 3)
- Write number: write 2 to a_3 , a becomes $[-1, 1, 2]$
- Carriage return: return the pointer to 1.
- Write number: write 3 to a_1 , a becomes $[3, 1, 2]$

With the second typewriter it is possible to:

- Move pointer: move the pointer to the next position. (i.e. 2)
- Write number: write 1 to a_2 , a becomes $[-1, 1, -1]$
- Carriage return: return the pointer to 3.
- Write number: write 2 to a_3 , a becomes $[-1, 1, 2]$
- Move pointer: move the pointer to the next position. (i.e. 2)
- Move pointer: move the pointer to the next position. (i.e. 1)
- Write number: write 3 to a_1 , a becomes $[3, 1, 2]$

C. Guess The Tree

2 seconds, 256 megabytes

This is an interactive problem.

Misuki has chosen a secret tree with n nodes, indexed from 1 to n , and asked you to guess it by using queries of the following type:

- " $? \ a \ b$ " — Misuki will tell you which node x minimizes $|d(a, x) - d(b, x)|$, where $d(x, y)$ is the distance between nodes x and y . If more than one such node exists, Misuki will tell you the one which minimizes $d(a, x)$.

Find out the structure of Misuki's secret tree using at most $15n$ queries!

Input

Each test consists of multiple test cases. The first line contains a single integer t ($1 \leq t \leq 200$) — the number of test cases.

Each test case consists of a single line with an integer n ($2 \leq n \leq 1000$), the number of nodes in the tree.

It is guaranteed that the sum of n across all test cases does not exceed 1000.

Interaction

The interaction begins by reading the integer n .

Then you can make up to $15n$ queries.

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

To report the answer, output a line in the format " $! \ a_1 \ b_1 \ a_2 \ b_2 \ \dots \ a_{n-1} \ b_{n-1}$ " (without quotes), meaning that there is an edge between nodes a_i and b_i , for each $1 \leq i \leq n - 1$. You can print the edges in any order.

After $15n$ queries have been made, the response to any other query will be -1 . Once you receive such a response, terminate the program to receive the `Wrong Answer` verdict.

After printing each line, do not forget to flush the output buffer. Otherwise, you will receive the `Idleness limit exceeded` verdict. To flush, 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

For hacks, use the following format: The first line contains an integer t ($1 \leq t \leq 200$) — the number of test cases.

The first line of each test contains an integer n — the number of nodes in the hidden tree.

Then $n - 1$ lines follow. The i -th of them contains two integers a_i and b_i ($1 \leq a_i, b_i \leq n$), meaning that there is an edge between a_i and b_i in the hidden tree.

The sum of n over all test cases must not exceed 1000.

input

```
1
4
1
1
1
3
```

output

```
? 1 2
? 1 3
? 1 4
! 1 2 1 3 3 4
```

A tree is an undirected acyclic connected graph. A tree with n nodes will always have $n - 1$ edges.

In the example case, the answer to " $? \ 1 \ 2$ " is 1. This means that there is an edge between nodes 1 and 2.

The answer to " $? \ 1 \ 3$ " is 1. This means that there is an edge between nodes 1 and 3.

The answer to " $? \ 1 \ 4$ " is 3. It can be proven that this can only happen if node 3 is connected to both node 1 and 4.

The edges of the tree are hence (1, 2), (1, 3) and (3, 4).

D. Longest Max Min Subsequence

2 seconds, 256 megabytes

You are given an integer sequence a_1, a_2, \dots, a_n . Let S be the set of all possible non-empty subsequences of a without duplicate elements. Your goal is to find the longest sequence in S . If there are multiple of them, find the one that minimizes lexicographical order after multiplying terms at odd positions by -1 .

For example, given $a = [3, 2, 3, 1]$, $S = \{[1], [2], [3], [2, 1], [2, 3], [3, 1], [3, 2], [2, 3, 1], [3, 2, 1]\}$. Then $[2, 3, 1]$ and $[3, 2, 1]$ would be the longest, and $[3, 2, 1]$ would be the answer since $[-3, 2, -1]$ is lexicographically smaller than $[-2, 3, -1]$.

A sequence c is a subsequence of a sequence d if c can be obtained from d by the deletion of several (possibly, zero or all) elements.

A sequence c is lexicographically smaller than a sequence d if and only if one of the following holds:

- c is a prefix of d , but $c \neq d$;
- in the first position where c and d differ, the sequence c has a smaller element than the corresponding element in d .

Input

Each test contains multiple test cases. The first line contains the number of test cases t ($1 \leq t \leq 5 \cdot 10^4$). The description of the test cases follows.

The first line of each test case contains an integer n ($1 \leq n \leq 3 \cdot 10^5$) — the length of a .

The second line of each test case contains n integers a_1, a_2, \dots, a_n ($1 \leq a_i \leq n$) — the sequence a .

It is guaranteed that the sum of n over all test cases does not exceed $3 \cdot 10^5$.

Output

For each test case, output the answer in the following format:

Output an integer m in the first line — the length of b .

Then output m integers b_1, b_2, \dots, b_m in the second line — the sequence b .

input
4
4
3 2 1 3
4
1 1 1 1
9
3 2 1 3 2 1 3 2 1
1
1

output
3
3 2 1
1
1
3
3 1 2
1
1

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

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

In the first example, $S = \{[1], [2], [3], [1, 3], [2, 1], [2, 3], [3, 1], [3, 2], [2, 1, 3], [3, 2, 1]\}$. Among them, $[2, 1, 3]$ and $[3, 2, 1]$ are the longest and $[-3, 2, -1]$ is lexicographical smaller than $[-2, 1, -3]$, so $[3, 2, 1]$ is the answer.

In the second example, $S = \{[1]\}$, so $[1]$ is the answer.

E1. Deterministic Heap (Easy Version)

3 seconds, 512 megabytes

This is the easy version of the problem. The difference between the two versions is the definition of deterministic max-heap, time limit, and constraints on n and t . You can make hacks only if both versions of the problem are solved.

Consider a perfect binary tree with size $2^n - 1$, with nodes numbered from 1 to $2^n - 1$ and rooted at 1. For each vertex v ($1 \leq v \leq 2^{n-1} - 1$), vertex $2v$ is its left child and vertex $2v + 1$ is its right child. Each node v also has a value a_v assigned to it.

Define the operation **pop** as follows:

1. initialize variable v as 1;
2. repeat the following process until vertex v is a leaf (i.e. until $2^{n-1} \leq v \leq 2^n - 1$):
 - a. among the children of v , choose the one with the larger value on it and denote such vertex as x ; if the values on them are equal (i.e. $a_{2v} = a_{2v+1}$), you can choose any of them;
 - b. assign a_x to a_v (i.e. $a_v := a_x$);
 - c. assign x to v (i.e. $v := x$);
3. assign -1 to a_v (i.e. $a_v := -1$).

Then we say the **pop** operation is deterministic if there is a unique way to do such operation. In other words, $a_{2v} \neq a_{2v+1}$ would hold whenever choosing between them.

A binary tree is called a max-heap if for every vertex v ($1 \leq v \leq 2^{n-1} - 1$), both $a_v \geq a_{2v}$ and $a_v \geq a_{2v+1}$ hold.

A max-heap is deterministic if the **pop** operation is deterministic to the heap when we do it **for the first time**.

Initially, $a_v := 0$ for every vertex v ($1 \leq v \leq 2^n - 1$), and your goal is to count the number of different deterministic max-heaps produced by applying the following operation **add** exactly k times:

- Choose an integer v ($1 \leq v \leq 2^n - 1$) and, for every vertex x on the path between 1 and v , add 1 to a_x .

Two heaps are considered different if there is a node which has different values in the heaps.

Since the answer might be large, print it modulo p .

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 first line of each test case contains three integers n, k, p ($1 \leq n, k \leq 500, 10^8 \leq p \leq 10^9, p$ is a prime).

It is guaranteed that the sum of n and the sum of k over all test cases does not exceed 500.

Output

For each test case, output a single line containing an integer: the number of different deterministic max-heaps produced by applying the aforementioned operation **add** exactly k times, modulo p .

input
7
1 13 998244353
2 1 998244353
3 2 998244853
3 3 998244353
3 4 100000037
4 2 100000039
4 3 100000037

output
1
2
12
52
124
32
304

input
1
500 500 100000007
output
76297230

input
6
87 63 100000037
77 77 100000039
100 200 998244353
200 100 998244353
32 59 998244853
1 1 998244353
output
26831232
94573603
37147649
847564946
727060898
1

For the first testcase, there is only one way to generate a , and such sequence is a deterministic max-heap, so the answer is 1.

For the second testcase, if we choose $v = 1$ and do the operation, we would have $a = [1, 0, 0]$, and since $a_2 = a_3$, we can choose either of them when doing the first **pop** operation, so such heap is not a deterministic max-heap.

And if we choose $v = 2$, we would have $a = [1, 1, 0]$, during the first **pop**, the following would happen:

- initialize v as 1
- since $a_{2v} > a_{2v+1}$, choose $2v$ as x , then $x = 2$
- assign a_x to a_v , then $a = [1, 1, 0]$
- assign x to v , then $v = 2$
- since v is a leaf, assign -1 to a_v , then $a = [1, -1, 0]$

Since the first **pop** operation is deterministic, this is a deterministic max-heap. Also, if we choose $v = 3$, a would be a deterministic max-heap, so the answer is 2.

E2. Deterministic Heap (Hard Version)

4 seconds, 512 megabytes

This is the hard version of the problem. The difference between the two versions is the definition of deterministic max-heap, time limit, and constraints on n and t . You can make hacks only if both versions of the problem are solved.

Consider a perfect binary tree with size $2^n - 1$, with nodes numbered from 1 to $2^n - 1$ and rooted at 1. For each vertex v ($1 \leq v \leq 2^{n-1} - 1$), vertex $2v$ is its left child and vertex $2v + 1$ is its right child. Each node v also has a value a_v assigned to it.

Define the operation **pop** as follows:

1. initialize variable v as 1;
2. repeat the following process until vertex v is a leaf (i.e. until $2^{n-1} \leq v \leq 2^n - 1$);
 - a. among the children of v , choose the one with the larger value on it and denote such vertex as x ; if the values on them are equal (i.e. $a_{2v} = a_{2v+1}$), you can choose any of them;
 - b. assign a_x to a_v (i.e. $a_v := a_x$);
 - c. assign x to v (i.e. $v := x$);
3. assign -1 to a_v (i.e. $a_v := -1$).

Then we say the **pop** operation is deterministic if there is a unique way to do such operation. In other words, $a_{2v} \neq a_{2v+1}$ would hold whenever choosing between them.

A binary tree is called a max-heap if for every vertex v ($1 \leq v \leq 2^{n-1} - 1$), both $a_v \geq a_{2v}$ and $a_v \geq a_{2v+1}$ hold.

A max-heap is deterministic if the **pop** operation is deterministic to the heap when we do it **for the first and the second time**.

Initially, $a_v := 0$ for every vertex v ($1 \leq v \leq 2^n - 1$), and your goal is to count the number of different deterministic max-heaps produced by applying the following operation **add** exactly k times:

- Choose an integer v ($1 \leq v \leq 2^n - 1$) and, for every vertex x on the path between 1 and v , add 1 to a_x .

Two heaps are considered different if there is a node which has different values in the heaps.

Since the answer might be large, print it modulo p .

Input

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

The first line of each test case contains three integers n, k, p ($2 \leq n \leq 100, 1 \leq k \leq 500, 10^8 \leq p \leq 10^9, p$ is a prime).

It is guaranteed that the sum of n does not exceed 100 and the sum of k over all test cases does not exceed 500.

Output

For each test case, output a single line containing an integer: the number of different deterministic max-heaps produced by applying the aforementioned operation **add** exactly k times, modulo p .

input	
6	
2	1 998244353
3	2 998244853
3	3 998244353
3	4 100000037
4	2 100000039
4	3 100000037
output	
2	
12	
40	
100	
32	
224	

input	
1	
100	500 100000037

output	
66681128	
input	
2	
87	63 100000037
13	437 100000039
output	
83566569	
54517140	

For the first testcase, if we choose $v = 1$ and do the operation, we would have $a = [1, 0, 0]$, and since $a_2 = a_3$, we can choose either of them when doing the first **pop** operation, so such heap is not a deterministic max-heap.

And if we choose $v = 2$, we would have $a = [1, 1, 0]$, during the first **pop**, the following would happen:

- initialize v as 1
- since $a_{2v} > a_{2v+1}$, choose $2v$ as x , then $x = 2$
- assign a_x to a_v , then $a = [1, 1, 0]$
- assign x to v , then $v = 2$
- since v is a leaf, assign -1 to a_v , then $a = [1, -1, 0]$

And during the second **pop**, the following would happen:

- initialize v as 1
- since $a_{2v} < a_{2v+1}$, choose $2v + 1$ as x , then $x = 3$
- assign a_x to a_v , then $a = [0, -1, 0]$
- assign x to v , then $v = 3$
- since v is a leaf, assign -1 to a_v , then $a = [0, -1, -1]$

Since both the first and the second **pop** operation are deterministic, this is a deterministic max-heap. Also, if we choose $v = 3$, a would be a deterministic max-heap, so the answer is 2.

