Codeforces Round 983 (Div. 2)

A. Circuit

1 second, 256 megabytes

Alice has just crafted a circuit with n lights and $2n$ switches. Each component (a light or a switch) has two states: on or off. The lights and switches are arranged in a way that:

- Each light is connected to **exactly two** switches.
- Each switch is connected to **exactly one** light. It's **unknown** which light each switch is connected to.
- When all switches are off, all lights are also off.
- If a switch is toggled (from on to off, or vice versa), the state of the light connected to it will also toggle.

Alice brings the circuit, which shows only the states of the $2n$ switches, to her sister Iris and gives her a riddle: what is the minimum and maximum number of lights that can be turned on?

Knowing her little sister's antics too well, Iris takes no more than a second to give Alice a correct answer. Can you do the same?

Input

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

The first line of each test case contains a single integer n ($1 \le n \le 50$) — the number of lights in the circuit.

The second line of each test case contains $2n$ integers $a_1, a_2, \ldots, a_{2n}$ ($0 \le a_i \le 1$) — the states of the switches in the circuit. $a_i = 0$ means the i-th switch is off, and $a_i = 1$ means the i-th switch is on.

Output

For each test case, output two integers — the minimum and maximum number of lights, respectively, that can be turned on.

input

```
5
1
0 0
1
0 1
1
1 1
3
0 0 1 0 1 0
3
0 1 1 1 0 0
```

output

```
0 0
1 1
0 0
0 2
1 3
```

In the first test case, there is only one light in the circuit, and no switch is on, so the light is certainly off.

In the second test case, there is only one light in the circuit, but one switch connected to it is on, so the light is on.

In the third test case, there is only one light in the circuit, and both switches are on, so the light is off as it was toggled twice.

In the fourth test case, to have no lights on, the switches can be arranged in this way:

- Switch 1 and switch 4 are connected to light 1. Since both switches are off, light 1 is also off.
- Switch 2 and switch 6 are connected to light 2. Since both switches are off, light 2 is also off.
- Switch 3 and switch 5 are connected to light 3. Both switches are on, so light 3 is toggled twice from its initial off state, and thus also stays off.

And to have 2 lights on, the switches can be arranged in this way:

- Switch 1 and switch 2 are connected to light 1. Since both switches are off, light 1 is also off.
- Switch 3 and switch 4 are connected to light 2. Since switch 3 is on and switch 4 is off, light 2 is toggled once from its initial off state, so it is on.
- Switch 5 and switch 6 are connected to light 3. Since switch 5 is on and switch 6 is off, light 3 is toggled once from its initial off state, so it is on.

B. Medians

1 second, 256 megabytes

You are given an array $a = [1, 2, \ldots, n]$, where n is **odd**, and an integer k.

Your task is to choose an **odd** positive integer m and to split a into m subarrays[†] $b_1, b_2, \ldots, b_m$ such that:

- Each element of the array a belongs to exactly one subarray.
- For all $1 \le i \le m$, $|b_i|$ is **odd**, i.e., the length of each subarray is odd.
- $\operatorname{median}([\operatorname{median}(b_1), \operatorname{median}(b_2), \ldots, \operatorname{median}(b_m)]) = k$, i.e., the median[‡] of the array of medians of all subarrays must equal k. $\operatorname{median}(c)$ denotes the median of the array c.

[†] A subarray of the array a of length n is the array $[a_l, a_{l+1}, \ldots, a_r]$ for some integers $1 \le l \le r \le n$.

[‡] A median of the array of odd length is the middle element after the array is sorted in non-decreasing order. For example: $\operatorname{median}([1,2,5,4,3]) = 3$, $\operatorname{median}([3,2,1]) = 2$, $\operatorname{median}([2,1,2,1,2,2,2]) = 2$.

Input

Each test consists of multiple test cases. The first line contains a single integer t $(1 \le t \le 5000)$ — the number of test cases. The description of the test cases follows.

The first line of each test case contains two integers n and k ($1 \le k \le n < 2 \cdot 10^5$, n is **odd**) — the length of array a and the desired median of the array of medians of all subarrays.

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

Output

For each test case:

- If there is no suitable partition, output -1 in a single line.
- Otherwise, in the first line, output an **odd** integer m $(1 \le m \le n)$, and in the second line, output m distinct integers $p_1, p_2, p_3, \ldots, p_m$ $(1 = p_1 < p_2 < p_3 < \ldots < p_m \le n)$ — denoting the left borders of each subarray.

In detail, for a valid answer $[p_1, p_2, \ldots, p_m]$:

- $b_1 = [a_{p_1}, a_{p_1+1}, \ldots, a_{p_2-1}]$
- $b_2 = [a_{p_2}, a_{p_2+1}, \ldots, a_{p_3-1}]$
- $\ldots$
- $b_m = [a_{p_m}, a_{p_m+1}, \ldots, a_n]$.

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

input

```
4
1 1
3 2
3 3
15 8
```

output

```
1
1
3
1 2 3
-1
5
1 4 7 10 13
```

In the first test case, the given partition has $m = 1$ and $b_1 = [1]$. It is obvious that $\text{median}([\text{median}([1])]) = \text{median}([1]) = 1$.

In the second test case, the given partition has $m = 3$ and:

- $b_1 = [1]$
- $b_2 = [2]$
- $b_3 = [3]$

Therefore, $\text{median}([\text{median}([1]), \text{median}([2]), \text{median}([3])]) = \text{median}([1, 2, 3]) = 2$.

In the third test case, there is no valid partition for $k = 3$.

In the fourth test case, the given partition has $m = 5$ and:

- $b_1 = [1, 2, 3]$
- $b_2 = [4, 5, 6]$
- $b_3 = [7, 8, 9]$
- $b_4 = [10, 11, 12]$
- $b_5 = [13, 14, 15]$

Therefore, $\text{median}([\text{median}([1, 2, 3]), \text{median}([4, 5, 6]), \text{median}([7, 8, 9]), \text{median}([10, 11, 12]), \text{median}([13, 14, 15])]) = \text{median}([2, 5, 8, 11, 14]) = 8$.

C. Trinity

2 seconds, 256 megabytes

You are given an array a of n elements $a_1, a_2, \ldots, a_n$.

You can perform the following operation any number (possibly 0) of times:

- Choose two integers i and j, where $1 \le i, j \le n$, and assign $a_i := a_j$.

Find the minimum number of operations required to make the array a satisfy the condition:

- For every pairwise distinct triplet of indices (x, y, z) ($1 \le x, y, z \le n$, $x \ne y$, $y \ne z$, $x \ne z$), there exists a non-degenerate triangle with side lengths a_x, a_y and a_z, i.e. $a_x + a_y > a_z$, $a_y + a_z > a_x$ and $a_z + a_x > a_y$.

Input

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

The first line of each test case contains a single integer n ($3 \le n \le 2 \cdot 10^5$) — the number of elements in the array a.

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

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 minimum number of operations required.

input

```
4
7
1 2 3 4 5 6 7
3
1 3 2
3
4 5 3
15
9 3 8 1 6 5 3 8 2 1 4 2 9 4 7
```

output

```
3
1
0
8
```

In the first test case, one of the possible series of operations would be:

- Assign $a_1 := a_4 = 4$. The array will become $[4, 2, 3, 4, 5, 6, 7]$.
- Assign $a_2 := a_5 = 5$. The array will become $[4, 5, 3, 4, 5, 6, 7]$.
- Assign $a_7 := a_1 = 4$. The array will become $[4, 5, 3, 4, 5, 6, 4]$.

It can be proven that any triplet of elements with pairwise distinct indices in the final array forms a non-degenerate triangle, and there is no possible answer using less than 3 operations.

In the second test case, we can assign $a_1 := a_2 = 3$ to make the array $a = [3, 3, 2]$.

In the third test case, since 3, 4 and 5 are valid side lengths of a triangle, we don't need to perform any operation to the array.

D. Genokraken

2 seconds, 256 megabytes

This is an interactive problem.

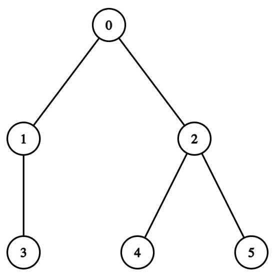
Upon clearing the Waterside Area, Gretel has found a monster named Genokraken, and she's keeping it contained for her scientific studies.

The monster's nerve system can be structured as a tree[†] of n nodes *(really, everything should stop resembling trees all the time…)*, numbered from 0 to $n - 1$, with node 0 as the root.

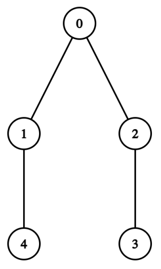
Gretel's objective is to learn the exact structure of the monster's nerve system — more specifically, she wants to know the values $p_1, p_2, \ldots, p_{n-1}$ of the tree, where p_i ($0 \le p_i < i$) is the direct parent node of node i ($1 \le i \le n - 1$).

She doesn't know exactly how the nodes are placed, but she knows a few convenient facts:

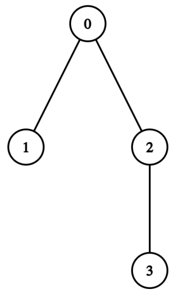
- If we remove root node 0 and all adjacent edges, this tree will turn into a forest consisting of only paths[‡]. Each node that was initially adjacent to the node 0 **will be the end of some path**.
- The nodes are indexed in a way that if $1 \le x \le y \le n - 1$, then $p_x \le p_y$.
- Node 1 has **exactly two** adjacent nodes (including the node 0).



The tree in this picture **does not** satisfy the condition, because if we remove node 0, then node 2 (which was initially adjacent to the node 0) will not be the end of the path $4 - 2 - 5$.



The tree in this picture **does not** satisfy the condition, because $p_3 \le p_4$ must hold.



The tree in this picture **does not** satisfy the condition, because node 1 has only one adjacent node.

Gretel can make queries to the containment cell:

- "`? a b`" ($1 \le a, b < n$, $a \ne b$) — the cell will check if the simple path between nodes a and b contains the node 0.

However, to avoid unexpected consequences by overstimulating the creature, Gretel wants to query at most $2n - 6$ times. Though Gretel is gifted, she can't do everything all at once, so can you give her a helping hand?

[†] A tree is a connected graph where every pair of distinct nodes has exactly one simple path connecting them.

[‡] A path is a tree whose vertices can be listed in the order $v_1, v_2, \ldots, v_k$ such that the edges are (v_i, v_{i+1}) $(1 \le i < k)$.

Input

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

The first line of each test case contains a single integer n ($4 \le n \le 10^4$) — the number of nodes in Genokraken's nerve system.

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

Interaction

For each test case, interaction starts by reading the integer n.

Then you can make queries of the following type:

- "`? a b`" (without quotes) $(1 \le a, b < n, a \ne b)$.

After the query, read an integer r — the answer to your query. You are allowed to use at most $2n - 6$ queries of this type.

- If the simple path between nodes a and b does not contain node 0, you will get $r = 0$.
- If the simple path between nodes a and b contains node 0, you will get $r = 1$.
- In case you make more than $2n - 6$ queries or make an invalid query, you will get $r = -1$. You will need to terminate after this to get the "`Wrong answer`" verdict. Otherwise, you can get an arbitrary verdict because your solution will continue to read from a closed stream.

When you find out the structure, output a line in the format "! $p_1\ p_2 \ldots p_{n-1}$" (without quotes), where p_i $(0 \le p_i < i)$ denotes the index of the direct parent of node i. This query is not counted towards the $2n - 6$ queries limit.

After solving one test case, the program should immediately move on to the next one. After solving all test cases, the program should be terminated immediately.

After printing any query do not forget to output an end of line and flush the output buffer. Otherwise, you will get `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 documentation for other languages.

The interactor is **non-adaptive**. The tree **does not change** during the interaction.

Hacks

For hack, use the following format:

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

The first line of each test case contains a single integer n ($4 \le n \le 10^4$) — the number of nodes in Genokraken's nerve system.

The second line of each test case contains $n - 1$ integers $p_1, p_2, \ldots, p_{n-1}$ $(0 \le p_1 \le p_2 \le \ldots \le p_{n-1} \le n-2$, $0 \le p_i < i)$ — the direct parents of node $1, 2, \ldots, n-1$ in the system, respectively.

In each test case, the values $p_1, p_2, \ldots, p_{n-1}$ must ensure the following in the tree:

- If we remove root node 0 and all adjacent edges, this tree will turn into a forest consisting of only paths. Each node that was initially adjacent to the node 0 will be the end of some path.
- Node 1 has exactly two adjacent nodes (including the node 0).

The sum of n over all test cases must not exceed 10^4.

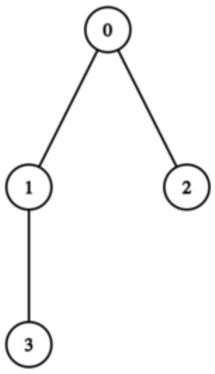
input

```
3
4

1

5

1

0

9
```

output

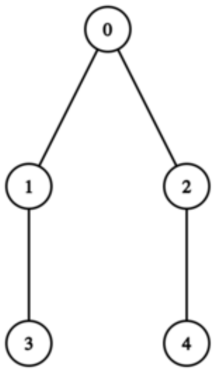
```


? 2 3

! 0 0 1

? 2 3

? 2 4

! 0 0 1 2

! 0 0 0 1 3 5 6 7
```

In the first test case, Genokraken's nerve system forms the following tree:



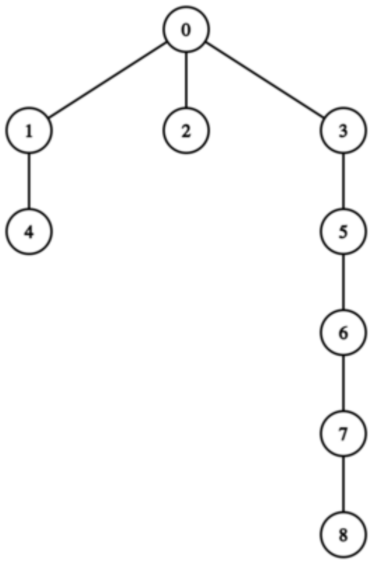
- The answer to "`? 2 3`" is 1. This means that the simple path between nodes 2 and 3 contains node 0.

In the second test case, Genokraken's nerve system forms the following tree:



- The answer to "`? 2 3`" is 1. This means that the simple path between nodes 2 and 3 contains node 0.
- The answer to "`? 2 4`" is 0. This means that the simple path between nodes 2 and 4 doesn't contain node 0.

In the third test case, Genokraken's nerve system forms the following tree:



E. Balanced

2 seconds, 256 megabytes

You are given a **cyclic** array a with n elements, where n is **odd**. In each operation, you can do the following:

- Choose an index $1 \le i \le n$ and increase a_{i-1} by 1, a_i by 2, and a_{i+1} by 1. The element before the first element is the last element because this is a cyclic array.

A cyclic array is called *balanced* if all its elements are equal to each other.

Find any sequence of operations to make this cyclic array balanced or determine that it is impossible. Please note that you **do not** have to minimize the number of operations.

Input

Each test consists of multiple test cases. The first line contains a single integer t ($1 \le t \le 2 \cdot 10^5$) — the number of test cases. The description of the test cases follows.

The first line of each test case contains a single integer n ($1 \le n < 2 \cdot 10^5$, n is **odd**) — the length of the array a.

The second line of each test case contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 10^6$) — the elements of the array a.

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

Output

For each test case:

- If it is impossible to make the cyclic array balanced, output -1.
- Otherwise, output n integers $v_1, v_2, \ldots, v_n$ ($0 \le v_i \le 10^{18}$) — where v_i denotes the number of operations applied to index i. It can be proved that if any solution exists, then there exists a solution under the given constraints. If there are several solutions under the given constraints, output any of them.

input
```
6
3
2 1 2
3
1 2 3
5
1 2 1 2 1
7
1 2 1 2 1 3 1
9
10000 10000 10000 10000 10000 10001 10002 10001 10000
1
10
```

output
```
0 1 0
2 1 0
2 0 3 0 2
4 2 7 0 8 0 6
1 1 1 1 1 1 0 1 1
0
```

In the first test case:

- After 1 operation applied to index $i = 2$, the array $a = [3, 3, 3]$.

In the second test case:

- After 2 operations applied to index $i = 1$, the array $a = [5, 4, 5]$.
- After 1 operation applied to index $i = 2$, the array $a = [6, 6, 6]$.

In the third test case:

- After 2 operations applied to index $i = 1$, the array $a = [5, 4, 1, 2, 3]$.
- After 3 operations applied to index $i = 3$, the array $a = [5, 7, 7, 5, 3]$.
- After 2 operations applied to index $i = 5$, the array $a = [7, 7, 7, 7, 7]$.

F. Peanuts

4 seconds, 256 megabytes

Having the magical beanstalk, Jack has been gathering a lot of peanuts lately. Eventually, he has obtained n pockets of peanuts, conveniently numbered 1 to n from left to right. The i-th pocket has a_i peanuts.

Jack and his childhood friend Alice decide to play a game around the peanuts. First, Alice divides the pockets into some boxes; each box will have a non-zero number of **consecutive** pockets, and each pocket will, obviously, belong to exactly one box. At the same time, Alice does not change the order of the boxes, that is, the boxes are numbered in ascending order of the indices of the pockets in them.

After that, Alice and Jack will take turns alternately, with Alice going first.

At each turn, the current player will remove a positive number of peanuts from **exactly one** pocket which belongs to the **leftmost non-empty box** (i.e., the leftmost box containing at least one non-empty pocket). In other words, if we number the boxes from left to right, then each player can only pick peanuts from the pocket in the j-th box ($j \ge 2$) only if the $(j-1)$-th box has no peanuts left. The player who cannot make a valid move loses.

Alice is sure she will win since she has the advantage of dividing the pockets into boxes herself. Thus, she wanted to know how many ways there are for her to divide the peanuts into boxes at the start of the game so that she will win, assuming both players play optimally. Can you help her with the calculation?

As the result can be very large, output it modulo $998\,244\,353$.

Input

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

The first line of each test case contains a single integer n ($1 \le n \le 10^6$) — the number of pockets.

The second line of each test case contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 10^9$) — the number of peanuts in each pocket.

It is guaranteed that the sum of n over all test cases does not exceed 10^6.

Output

For each test case, output a single integer — the number of ways for Alice to divide the pockets into boxes at the start of the game to guarantee her win, assuming both players play optimally, modulo $998\,244\,353$.

input

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

output

```
1
4
16
0
205
```

In the first test case, the only way for Alice to win is to divide the pockets into two boxes as follows: $([1,2],[3])$ (the first box contains the first two pockets and the second box contains the third pocket). Alice wins by taking both peanuts from the second pocket, leaving Jack with $([1],[3])$. Jack is forced to take the only peanut left in the first box, which allows Alice to take the remaining ones in the second box.

In the second test case, the winning divisions for Alice are $([1],[2,3,1])$, $([1,2,3,1])$, $([1,2],[3],[1])$, and $([1,2],[3,1])$.

In the third test case, Alice always wins no matter how she divides the pockets into boxes.

In the fourth test case, Alice always loses no matter how she divides the pockets into boxes.