Codeforces Round 1045 (Div. 2)

A. Painting With Two Colors

1 second, 256 megabytes

You are given three positive integers n, a, and b ($1 \le a, b \le n$).

Consider a row of n cells, initially all white and indexed from 1 to n. You will perform the following two steps **in order**:

1. Choose an integer x such that $1 \le x \le n - a + 1$, and paint the a consecutive cells $x, x + 1, \dots, x + a - 1$ red.
2. Choose an integer y such that $1 \le y \le n - b + 1$, and paint the b consecutive cells $y, y + 1, \dots, y + b - 1$ blue.

If a cell is painted both red and blue, its final color is blue.

A coloring of the grid is considered *symmetric* if, for every integer i from 1 to n (inclusive), the color of cell i is the same as the color of cell $(n + 1 - i)$. Your task is to determine whether there exist integers x and y such that the final coloring of the grid is symmetric.

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 and the only line of each test case contains three integers n, a, and b ($1 \le n \le 10^9$, $1 \le a, b \le n$) — the number of cells of the grid and the number of cells to be painted in each step.

Output

For each testcase, output "YES" if it is possible that the final coloring of the grid is symmetric; 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.

input

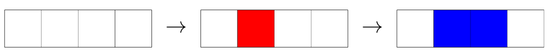
```
7
5 3 1
4 1 2
7 7 4
8 3 7
1 1 1
1000000000 1000000000 1000000000
3 2 1
```

output

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

In the first test case, the grid becomes symmetric when choosing $x = 2$ and $y = 3$, so the answer is "YES". The following figure illustrates each step of the coloring process:

In the second test case, the grid becomes symmetric when choosing $x = 2$ and $y = 2$, so the answer is "YES". The following figure illustrates each step of the coloring process:



In the third and fourth test cases, it can be proved that no choice of x and y results in a symmetric grid, so the answer is "NO".

B. Add 0 or K

1.5 seconds, 256 megabytes

You are given an array of n positive integers $a_1, a_2, \dots, a_n$ and a positive integer k.

In one operation, you may add either 0 or k to each a_i, i.e., choose another array of n integers $b_1, b_2, \dots, b_n$ where each b_i is either 0 or k, and update a_i to $a_i + b_i$ for $1 \le i \le n$. Note that you can choose different values for each element of the array b.

Your task is to perform at most k such operations to make $\gcd(a_1, a_2, \dots, a_n) > 1$ *. It can be proved that this is always possible.

Output the final array after the operations. You do **not** have to output the operations themselves.

* $\gcd(a_1, a_2, \dots, a_n)$ denotes the greatest common divisor (GCD) of $a_1, a_2, \dots, a_n$.

Input

Each test contains multiple test cases. The first line contains the number of test cases t ($1 \le t \le 1000$). The description of the test cases follows.

The first line of each test case contains two integers n and k ($1 \le n \le 10^5$, $1 \le k \le 10^9$) — the length of the array a and the given constant.

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

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

Output

For each test case, output an array of n integers in a new line — the final array after the operations. The integers in the output should be within the range from 1 to $10^9 + k^2$.

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

Note that you do **not** have to minimize the number of operations.

input

```
8
3 3
2 7 1
4 5
2 9 16 14
4 1
1 2 3 4
5 2
5 6 7 8 9
2 10
7 9
1 1000000000
1
1 371
1000000000
3 6
1 3 5
```

output

```
8 10 10
7 14 21 14
2 2 4 4
9 6 9 12 9
77 99
1000000000000000001
1000000000
25 15 5
```

In the first test case, the output $[8, 10, 10]$ is valid because $\gcd(8, 10, 10) = 2 > 1$, and the array $[2, 7, 1]$ can be transformed into $[8, 10, 10]$ using at most 3 operations. One possible sequence of operations is shown below:

Operation	b	Resulting a
1	$[3, 0, 3]$	$[5, 7, 4]$
2	$[0, 0, 3]$	$[5, 7, 7]$
3	$[3, 3, 3]$	$[8, 10, 10]$

Other outputs like $[2, 10, 4]$, $[8, 16, 4]$, and $[5, 10, 10]$ are also considered valid.

In the second test case, the output $[7, 14, 21, 14]$ is valid because:

- $\gcd(7, 14, 21, 14) = 7 > 1$.
- Starting from $[2, 9, 16, 14]$, applying the operation with $b = [5, 5, 5, 0]$ yields $[7, 14, 21, 14]$, requiring no more than 5 operations.

C. Even Larger

2 seconds, 256 megabytes

An array is called *good* if, for every subarray* of length at least 2, the sum of the elements at even indices (with respect to the **original array**) is greater than or equal to the sum of the elements at odd indices. Array indexing starts from 1.

For example, the arrays $[3, 8, 4, 4]$ and $[2, 3, 1, 4, 2]$ are good. The array $[0, 2, 4, 1]$ is not because, in the subarray $[2, 4, 1]$, the elements at even indices in the original array are 2 (index 2) and 1 (index 4), while the only element at an odd index is 4 (index 3). Since $2 + 1 < 4$, the condition does **not** hold for this subarray.

You are given an array of n **non-negative** integers $a_1, a_2, \ldots, a_n$. In one operation, you can decrease any element in the array by 1, but all elements must remain **non-negative**. Your task is to find the minimum number of operations needed to make the array a good. It can be proved that it is possible to make the array good using a finite number of operations.

*An array b is a subarray of an array a if b can be obtained from a by the deletion of several (possibly, zero or all) elements from the beginning and several (possibly, zero or all) elements from the end.

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 a single integer n ($2 \le n \le 2 \cdot 10^5$) — the length of the array a.

The second line of each test case contains n non-negative integers $a_1, a_2, \ldots, a_n$ ($0 \le a_i \le 10^9$) — 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 in a new line — the minimum number of operations needed to make the array a good.

input
```
8
4
3 8 4 4
4
0 2 3 5
2
3 1
5
2 3 1 4 2
4
0 2 4 1
5
3 1 4 5 1
11
3 0 5 4 4 5 3 0 3 4 1
12
410748345 10753674 975233308 193331255 893457280 279719251
704970985 412553354 801228787 44181004 1000000000 3829103
```

output
```
0
1
2
0
3
6
14
4450984776
```

In the first test case, the array a is already good, so the answer is 0. Below are the checks for each subarray:

Subarray	Even Index Sum	Odd Index Sum	Condition met?
$[3, 8]$	8	3	Yes
$[8, 4]$	8	4	Yes
$[4, 4]$	4	4	Yes
$[3, 8, 4]$	8	$3 + 4 = 7$	Yes
$[8, 4, 4]$	$8 + 4 = 12$	4	Yes
$[3, 8, 4, 4]$	$8 + 4 = 12$	$3 + 4 = 7$	Yes

In the second test case, the array a is not good:

Subarray	Even Index Sum	Odd Index Sum	Condition met?
$[0, 2]$	2	0	Yes
$[2, 3]$	2	3	No
$[3, 5]$	5	3	Yes
$[0, 2, 3]$	2	$0 + 3 = 3$	No
$[2, 3, 5]$	$2 + 5 = 7$	3	Yes
$[0, 2, 3, 5]$	$2 + 5 = 7$	$0 + 3 = 3$	Yes

However, if we perform an operation on index 3, the array becomes $[0, 2, 2, 5]$ and is good now:

Subarray	Even Index Sum	Odd Index Sum	Condition met?
$[0, 2]$	2	0	Yes
$[2, 2]$	2	2	Yes
$[2, 5]$	5	2	Yes
$[0, 2, 2]$	2	$0 + 2 = 2$	Yes
$[2, 2, 5]$	$2 + 5 = 7$	2	Yes
$[0, 2, 2, 5]$	$2 + 5 = 7$	$0 + 2 = 2$	Yes

Therefore, the answer is 1.

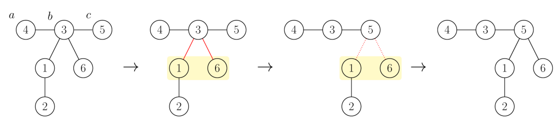
D. Sliding Tree

2 seconds, 256 megabytes

You are given a tree* with n vertices numbered from 1 to n. You can modify its structure using the following multi-step operation, called a *sliding operation*:

1. Choose three **distinct** vertices a, b, and c such that b is directly connected to both a and c.
2. Then, for every neighbor d of b (**excluding** a and c), remove the edge between b and d, and instead connect d directly to c.

For example, the figure below illustrates this operation with $a = 4$, $b = 3$, and $c = 5$ in the leftmost tree.



It can be proved that after a sliding operation, the resulting graph is still a tree.

Your task is to find a sequence of sliding operations that transforms the tree into a path graph†, while **minimizing** the total number of operations. You only need to output the **first sliding operation** in an optimal sequence if at least one operation is required. It can be proved that it is possible to transform the tree into a path graph using a finite number of operations.

*A tree is a connected graph without cycles.

†A path graph is a tree where every vertex has a degree of at most 2. Note that a graph with only 1 vertex and no edges is also a path graph.

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 a single integer n ($1 \le n \le 2 \cdot 10^5$) — the number of vertices in the tree.

The i-th of the following $n - 1$ lines contains two integers u_i and v_i ($1 \le u_i, v_i \le n$, $u_i \ne v_i$) — the ends of the i-th edge.

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:

- If no operations are required (that is, the input tree is already a path graph), output -1.
- Otherwise, output three **distinct** integers a, b, and c ($1 \le a, b, c \le n$) — the chosen vertices for the first sliding operation in an optimal sequence.

If there are multiple valid choices for the first operation, you can output any of them.

input

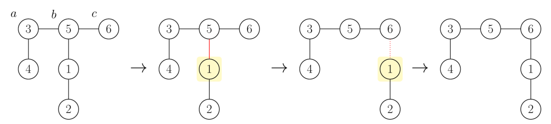
```
4
6
4 3
3 5
3 1
1 2
3 6
1
2
1 2
5
5 4
2 3
4 2
1 4
```

output

```
4 3 5
-1
-1
2 4 1
```

The first test case matches the example provided in the problem statement. It can be proved that we cannot transform the given tree into a path graph using less than 2 operations.

However, we can transform the given tree into a path graph using 2 operations: Initially, we perform an operation with $a = 4$, $b = 3$, and $c = 5$, as shown in the example. Next, we apply an operation with $a = 3$, $b = 5$, and $c = 6$. After this, the tree becomes a path graph. The second operation is illustrated in the figure below.



Thus, we obtain a sequence of sliding operations with the total number of operations minimized. Note, however, that only the first operation must be in the output; the number of operations and the second operation should **not** appear in the output.

In the second and third test cases, the tree is already a path graph, so no operations are required.

E. Power Boxes

2 seconds, 256 megabytes

This is an interactive problem.

You are given n boxes, indexed from 1 to n. The boxes look identical, but each one has a hidden power value a_i, which is either 1 or 2.

You want to determine the power value of each box. To do so, you conduct the following experiment. Initially, the i-th box is placed at coordinate i on a number line ($1 \le i \le n$).

You are allowed to perform the following two types of queries:

- "`swap` x" ($1 \le x \le n-1$): Swap the boxes currently located at **coordinates** x and $x+1$. Note that this change is permanent and affects all subsequent queries.
- "`throw` x" ($1 \le x \le n$): Throw a ball at the box located at **coordinate** x. The ball travels p units forward to coordinate $x+p$ if the power value of the box is p. If there is a box at the new coordinate, the ball jumps again using the power of that box. This continues until the ball lands on a coordinate without a box. As a response, you are given the total number of jumps the ball made before stopping.

Your task is to determine the power value of each box using no more than $\lceil \frac{3n}{2} \rceil$ queries in total, counting both `swap` and `throw` queries.

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 and the only line of each test case contains a single integer n ($2 \le n \le 1000$) — the number of boxes.

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

Interaction

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

To make a query, output a line in one of the following formats:

- "`swap` x" (without quotes) ($1 \le x \le n-1$): Swap the boxes currently located at **coordinates** x and $x+1$.
- "`throw` x" (without quotes) ($1 \le x \le n$): Throw a ball at the box located at **coordinate** x. The jury will respond with an integer representing the number of jumps the ball made before stopping.

Note that queries are **case sensitive**.

Once you have determined the power value of each box, output a line in the following format:

- "`!` $a_1\ a_2\ \ldots\ a_n$" (without quotes): Here, a_i is the power value of the box **initially located** at **coordinate** i ($1 \le i \le n$). After submitting the final answer, proceed on to the next test case.

Note that submitting the final answer with the previous query does **not** count towards the limit of $\lceil \frac{3n}{2} \rceil$ queries.

If your program exceeds $\lceil \frac{3n}{2} \rceil$ queries in one test case, your program must terminate immediately to receive the verdict `Wrong Answer`. Otherwise, it may receive any other verdict.

After outputting a query, do not forget to output the end of the line and flush the output. Otherwise, you will get `Idleness limit exceeded`. To do this, use:

- `fflush(stdout)` or `cout.flush()` in C++;
- `System.out.flush()` in Java;
- `sys.stdout.flush()` in Python;
- `std::io::stdout().flush()` in Rust;
- see the documentation for other languages.

The interactor is **non-adaptive**; the power values of the boxes remain constant throughout the interaction.

Hacks

To hack, use the following format.

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

The first line of each test case contains a single integer n ($2 \le n \le 1000$) — the number of boxes.

The second line of each test case contains n integers $a_1, a_2, \ldots, a_n$ ($1 \le a_i \le 2$) — the power value of each box.

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

input

```
2
4

2


3

3

2

2


1
```

output

```

throw 2

swap 3
throw 2

throw 1

! 2 1 2 1

throw 1

swap 1
throw 1

! 1 2
```

Below is the interaction process in the example:

Solution	Jury	Explanation
	2	There are 2 test cases.
	4	There are 4 boxes in the first test case. The hidden power values are $a = [2, 1, 2, 1]$.
throw 2	2	Throw a ball at the box located at coordinate 2. The ball travels through coordinates $2 \to 3 \to 5$ and stops at coordinate 5, so the response is 2.
swap 3		Swap the boxes located at coordinate 3 and 4. Now box 3 is located at coordinate 4 and box 4 is located at coordinate 3.
throw 2	3	Throw a ball at the box located at coordinate 2. The ball travels through coordinates $2 \to 3 \to 4 \to 6$ and stops at coordinate 6, so the response is 3. Note that the response is different because of the swap.
throw 1	3	Throw a ball at the box located at coordinate 1. The ball travels through coordinates $1 \to 3 \to 4 \to 6$ and stops at coordinate 6, so the response is 3.
! 2 1 2 1		The solution concludes that the power values are $[2, 1, 2, 1]$.
	2	There are 2 boxes in the second test case. The hidden power values are $a = [1, 2]$.
throw 1	2	Throw a ball at the box located at coordinate 1. The ball travels through coordinates $1 \to 2 \to 4$ and stops at coordinate 4, so the response is 2.
swap 1		Swap the boxes located at coordinate 1 and 2. Now box 1 is located at coordinate 2 and box 2 is located at coordinate 1.
throw 1	1	Throw a ball at the box located at coordinate 1. The ball travels through coordinates $1 \to 3$ and stops at coordinate 3, so the response is 1.
! 1 2		The solution concludes that the power values are $[1, 2]$.

Empty lines in the example input and output are given only for better readability; you don't need to output them in your solution.

Note that in the first test case, the given queries are in fact insufficient to uniquely determine the power values; they are given only to illustrate the input/output format.

F. Permutation Oddness

5 seconds, 256 megabytes

You are given four positive integers c_0, c_1, c_2, and c_3.

Let $n = c_0 + c_1 + c_2 + c_3$. Consider an array a of n integers with x ($0 \le x \le 3$) appearing c_x times. For any **distinct permutation**[*] b of the array a, define its *oddness* as[†‡]:

$$\sum_{i=1}^{n-1} \operatorname{lowbit}(b_i \oplus b_{i+1})$$

Your task is to count, for each integer k from 0 to $2 \cdot (n-1)$ (inclusive), the number of distinct permutations of a with an oddness equal to k.

Since the numbers might be too large, you are only required to find them modulo $10^9 + 7$.

[*] A permutation of the array is an arrangement of its elements into any order. For example, $[1, 2, 2]$ is a permutation of $[2, 2, 1]$, but $[1, 1, 2]$ is not. Two permutations are considered distinct if they differ in at least one position.

[†] $\oplus$ denotes the bitwise XOR operation.

[‡] $\operatorname{lowbit}(x)$ is the value of the lowest binary bit of x, e.g. $\operatorname{lowbit}(12) = 4$, $\operatorname{lowbit}(8) = 8$. Specifically, we define $\operatorname{lowbit}(0) = 0$.

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 and the only line of each test case contains four positive integers c_0, c_1, c_2, and c_3 ($1 \le c_0, c_1, c_2, c_3 < 800$, $4 \le c_0 + c_1 + c_2 + c_3 \le 800$).

Let $n = c_0 + c_1 + c_2 + c_3$. It is guaranteed that the sum of n over all test cases does not exceed 800.

Output

For each test case, output $2 \cdot (n-1) + 1$ integers in one line — the number of distinct permutations of a with an oddness equal to $0, 1, \ldots, 2 \cdot (n-1)$ respectively. Output the answers modulo $10^9 + 7$.

input

```
3
1 1 1 1
1 2 4 1
3 3 3 3
```

output

```
0 0 0 8 8 8 0
0 0 0 8 32 126 184 244 156 72 18 0 0 0 0
0 0 0 8 56 424 1472 5760 12128 29376 40384 65232 59920 65232
40384 29376 12128 5760 1472 424 56 8 0
```

In the first test case, the array a has 24 distinct permutations. The table below shows the oddness values for some selected permutations:

Permutation	Oddness
$[0, 1, 2, 3]$	$\operatorname{lowbit}(0 \oplus 1) + \operatorname{lowbit}(1 \oplus 2) + \operatorname{lowbit}(2 \oplus 3) = 1 + 1 + 1 =$
$[0, 2, 1, 3]$	$\operatorname{lowbit}(0 \oplus 2) + \operatorname{lowbit}(2 \oplus 1) + \operatorname{lowbit}(1 \oplus 3) = 2 + 1 + 2 =$
$[0, 1, 3, 2]$	$\operatorname{lowbit}(0 \oplus 1) + \operatorname{lowbit}(1 \oplus 3) + \operatorname{lowbit}(3 \oplus 2) = 1 + 2 + 1 =$

Overall, among the 24 permutations:

- 8 permutations have oddness 3.
- 8 permutations have oddness 4.
- 8 permutations have oddness 5.

In the second test case, the array a has 840 distinct permutations. The distribution of oddness values is as follows:

- 8 permutations have oddness 3.
- 32 permutations have oddness 4.
- 126 permutations have oddness 5.
- 184 permutations have oddness 6.
- 244 permutations have oddness 7.
- 156 permutations have oddness 8.
- 72 permutations have oddness 9.
- 18 permutations have oddness 10.