CodeCraft-21 and Codeforces Round #711 (Div. 2)

A. GCD Sum

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

The $gcdSum$ of a positive integer is the gcd of that integer with its sum of digits. Formally, $gcdSum$ x gcd x x for a positive integer x. gcd a b denotes the greatest common divisor of a and b — the largest integer d such that both integers a and b are divisible by d.

For example: $gcdSum$ gcd gcd .

Given an integer n, find the smallest integer x n such that $gcdSum$ x .

Input

The first line of input contains one integer t t — the number of test cases.

Then t lines follow, each containing a single integer n n .

All test cases in one test are different.

Output

Output t lines, where the i-th line is a single integer containing the answer to the i-th test case.

Example

input

```
3
11
31
75
```

output

```
12
33
75
```

Note

Let us explain the three test cases in the sample.

Test case 1: n :

$gcdSum$ gcd gcd .

$gcdSum$ gcd gcd .

So the smallest number whose $gcdSum$ is .

Test case 2: n :

$gcdSum$ gcd gcd .

$gcdSum$ gcd gcd .

$gcdSum$ gcd gcd .

So the smallest number whose $gcdSum$ is .

Test case 3: n :

$gcdSum$ gcd gcd .

The $gcdSum$ of is already . Hence, it is the answer.

B. Box Fitting

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

You are given n rectangles, each of height . Each rectangle's width is a power of (i. e. it can be represented as ^{x} for some non-negative integer x).

You are also given a two-dimensional box of width W. Note that W may or may not be a power of . Moreover, W is at least as large as the width of the largest rectangle.

You have to find the smallest height of this box, such that it is able to fit all the given rectangles. It is allowed to have some empty space left in this box after fitting all the rectangles.

You cannot rotate the given rectangles to make them fit into the box. Moreover, any two distinct rectangles must not overlap, i. e., any two distinct rectangles must have zero intersection area.

See notes for visual explanation of sample input.

Input

The first line of input contains one integer t (t) — the number of test cases. Each test case consists of two lines.

For each test case:

- the first line contains two integers n (n) and W (W);
- the second line contains n integers w w w_n (w_i), where w_i is the width of the i-th rectangle. Each w_i is a power of ;
- additionally, $^{n}_{i} \quad w_i \quad W$.

The sum of n over all test cases does not exceed .

Output

Output t integers. The i-th integer should be equal to the answer to the i-th test case — the smallest height of the box.

Example

input

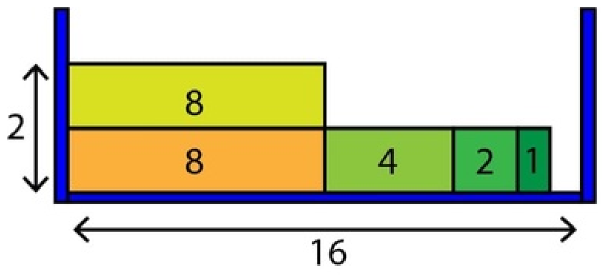
```
2
5 16
1 2 8 4 8
6 10
2 8 8 2 2 8
```

output

```
2
3
```

Note

For the first test case in the sample input, the following figure shows one way to fit the given five rectangles into the 2D box with minimum height:



In the figure above, the number inside each rectangle is its width. The width of the 2D box is (indicated with arrow below). The minimum height required for the 2D box in this case is (indicated on the left).

In the second test case, you can have a minimum height of three by keeping two blocks (one each of widths eight and two) on each of the three levels.

C. Planar Reflections

time limit per test: 1 second
memory limit per test: 256 megabytes
input: standard input
output: standard output

Gaurang has grown up in a mystical universe. He is faced by n consecutive 2D planes. He shoots a particle of decay age k at the planes.

A particle can pass through a plane directly, however, every plane produces an identical copy of the particle going in the opposite direction with a decay age k . If a particle has decay age equal to , it will NOT produce a copy.

For example, if there are two planes and a particle is shot with decay age (towards the right), the process is as follows: (here, $D \; x$ refers to a single particle with decay age x)

1. the first plane produces a D to the left and lets D continue on to the right;
2. the second plane produces a D to the left and lets D continue on to the right;
3. the first plane lets D continue on to the left and produces a D to the right;
4. the second plane lets D continue on to the right (D cannot produce any copies).

In total, the final multiset S of particles is D D D D . (See notes for visual explanation of this test case.)

Gaurang is unable to cope up with the complexity of this situation when the number of planes is too large. Help Gaurang find the size of the multiset S, given n and k.

Since the size of the multiset can be very large, you have to output it modulo .

Note: Particles can go back and forth between the planes without colliding with each other.

Input

The first line of the input contains the number of test cases t (t). Then, t lines follow, each containing two integers n and k (n k).

Additionally, the sum of n over all test cases will not exceed , and the sum of k over all test cases will not exceed . All test cases in one test are different.

Output

Output t integers. The i-th of them should be equal to the answer to the i-th test case.

Examples

input

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

output

```
4
3
1
2
```

input

```
3
1 1
1 500
500 250
```

output

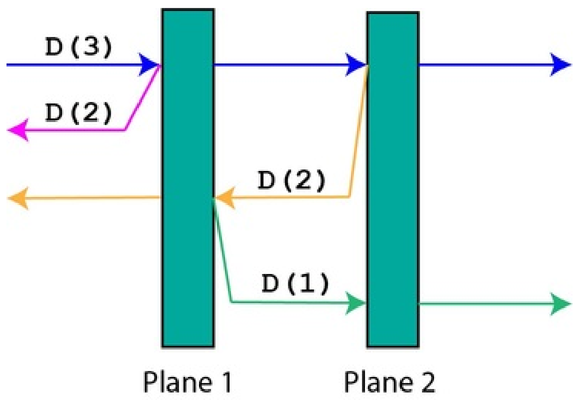
```
1
2
257950823
```

Note

Let us explain the first example with four test cases.

Test case 1: (n , k) is already explained in the problem statement.

See the below figure of this simulation. Each straight line with a different color represents the path of a different particle. As you can see, there are four distinct particles in the multiset. Note that the vertical spacing between reflected particles is for visual clarity only (as mentioned before, no two distinct particles collide with each other)



Test case 2: (n , k) is explained as follows:

1. the first plane produces a D to the left and lets D continue on to the right;
2. the second plane produces a D to the left and lets D continue on to the right;
3. the first plane lets D continue on to the left (D cannot produce any copies).

Total size of multiset obtained D D D is equal to three.

Test case 3: (n , k), there are three planes, but decay age is only one. So no new copies are produced while the one particle passes through the planes. Hence, the answer is one.

Test case 4: (n , k) there is only one plane. The particle produces a new copy to the left. The multiset D D is of size two.

D. Bananas in a Microwave

time limit per test: 3 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

You have a malfunctioning microwave in which you want to put some bananas. You have n time-steps before the microwave stops working completely. At each time-step, it displays a new operation.

Let k be the number of bananas in the microwave currently. Initially, k . In the i-th operation, you are given three parameters t_i , x_i, y_i in the input. Based on the value of t_i, you must do one of the following:

Type 1: (t_i , x_i, y_i) — pick an a_i, such that a_i y_i, and perform the following update a_i times: k k x_i .

Type 2: (t_i , x_i, y_i) — pick an a_i, such that a_i y_i, and perform the following update a_i times: k k x_i .

Note that x_i **can be a fractional value**. See input format for more details. Also, x is the smallest integer x.

At the i-th time-step, you must apply the i-th operation exactly once.

For each j such that j m, output the earliest time-step at which you can create **exactly** j bananas. If you cannot create **exactly** j bananas, output .

Input

The first line contains two space-separated integers n n and m m .

Then, n lines follow, where the i-th line denotes the operation for the i-th timestep. Each such line contains three space-separated integers t_i, x_i and y_i (t_i , y_i m).

Note that you are given x_i, which is x_i. Thus, to obtain x_i, use the formula x_i $\frac{x_i}{}$.

For **type 1** operations, x_i m, and for **type 2** operations, x_i m.

Output

Print m integers, where the i-th integer is the earliest time-step when you can obtain **exactly** i bananas (or if it is impossible).

Examples

input

```
3 20
1 300000 2
2 400000 2
1 1000000 3
```

output

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

input

```
3 20
1 399999 2
2 412345 2
1 1000001 3
```

output

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

Note

In the **first sample input**, let us see how to create number of bananas in three timesteps. Initially, k .

- In timestep 1, we choose a , so we apply the type 1 update — k k — two times. Hence, k is now 6.
- In timestep 2, we choose a , hence value of k remains unchanged.
- In timestep 3, we choose a , so we are applying the type 1 update k k once. Hence, k is now 16.

It can be shown that k cannot be reached in fewer than three timesteps with the given operations.

In the **second sample input**, let us see how to create number of bananas in two timesteps. Initially, k .

- In timestep 1, we choose a , so we apply the type 1 update — k k — once. Hence, k is now 4.
- In timestep 2, we choose a , so we apply the type 2 update — k k — once. Hence, k is now 17.

It can be shown that k cannot be reached in fewer than two timesteps with the given operations.

E. Two Houses

time limit per test: 3.5 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

This is an interactive problem. Remember to flush your output while communicating with the testing program. You may use `fflush(stdout)` in C++, `system.out.flush()` in Java, `stdout.flush()` in Python or `flush(output)` in Pascal to flush the output. If you use some other programming language, consult its documentation. You may also refer to the guide on interactive problems: https://codeforces.com/blog/entry/45307.

There is a city in which Dixit lives. In the city, there are n houses. There is **exactly one directed road between every pair of houses.** For example, consider two houses A and B, then there is a directed road either from A to B or from B to A but not both. The number of roads leading to the i-th house is k_i.

Two houses A and B are *bi-reachable* if A is reachable from B **and** B is reachable from A. We say that house B is reachable from house A when there is a path from house A to house B.

Dixit wants to buy two houses in the city, that is, one for living and one for studying. Of course, he would like to travel from one house to another. So, he wants to find a pair of bi-reachable houses A and B. Among all such pairs, he wants to choose one with the maximum value of k_A k_B, where k_i is the number of roads leading to the house i. If more than one optimal pair exists, any of them is suitable.

Since Dixit is busy preparing CodeCraft, can you help him find the desired pair of houses, or tell him that no such houses exist?

In the problem input, you are **not** given the direction of each road. You are given — for each house — only the number of incoming roads to that house (k_i).

You are allowed to ask only one type of query from the judge: give two houses A and B, and the judge answers whether B is reachable from A. There is **no upper limit on the number of queries**. But, **you cannot ask more queries after the judge answers "Yes" to any of your queries.** Also, you cannot ask the same query twice.

Once you have exhausted all your queries (or the judge responds "`Yes`" to any of your queries), your program must output its guess for the two houses and quit.

See the Interaction section below for more details.

Input

The first line contains a single integer n (n) denoting the number of houses in the city. The next line contains n space-separated integers k k k_n (k_i n), the i-th of them represents the number of incoming roads to the i-th house.

Interaction

To ask a query, print "`? A B`" A B N A B. The judge will respond "`Yes`" if house B is reachable from house A, or "`No`" otherwise.

To output the final answer, print "`! A B`", where A and B are bi-reachable with the maximum possible value of k_A k_B. If there does not exist such pair of houses A and B, output "`! 0 0`".

After outputting the final answer, your program must terminate immediately, otherwise you will receive Wrong Answer verdict.

You cannot ask the same query twice. **There is no upper limit to the number of queries you ask, but, you cannot ask more queries after the judge answers "Yes" to any of your queries.** Your program must now output the final answer ("`! A B`" or "`! 0 0`") and terminate.

If you ask a query in incorrect format or repeat a previous query, you will get Wrong Answer verdict.

After printing a query do not forget to output the end of the line and flush the output. Otherwise, you will get the Idleness limit exceeded error. 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.

Examples

input

```
3
1 1 1
Yes
```

output

```
? 1 2
! 1 2
```

input

```
4
1 2 0 3
No
No
No
No
No
No
```

output

```
? 2 1
? 1 3
? 4 1
? 2 3
? 4 2
? 4 3
! 0 0
```

Note

In the first sample input, we are given a city of three houses with one incoming road each. The user program asks one query: "`? 1 2`": asking whether we can reach from house to house . The judge responds with "`Yes`". The user program now concludes that this is sufficient information to determine the correct answer. So, it outputs "`! 1 2`" and quits.

In the second sample input, the user program queries for six different pairs of houses, and finally answers "`! 0 0`" as it is convinced that no two houses as desired in the question exist in this city.

F. Christmas Game

time limit per test: 2 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

Alice and Bob are going to celebrate Christmas by playing a game with a tree of presents. The tree has n nodes (numbered to n, with some node r as its root). There are a_i presents are hanging from the i-th node.

Before beginning the game, a special integer k is chosen. The game proceeds as follows:

- Alice begins the game, with moves alternating each turn;
- in any move, the current player may choose some node (for example, i) which has depth at least k. Then, the player picks some positive number of presents hanging from that node, let's call it m m a_i ;
- the player then places these m presents on the k-th ancestor (let's call it j) of the i-th node (the k-th ancestor of vertex i is a vertex j such that i is a descendant of j, and the difference between the depth of j and the depth of i is exactly k). Now, the number of presents of the i-th node a_i is decreased by m, and, correspondingly, a_j is increased by m;
- Alice and Bob both play optimally. The player unable to make a move loses the game.

For each possible root of the tree, find who among Alice or Bob wins the game.

Note: The depth of a node i in a tree with root r is defined as the number of edges on the simple path from node r to node i. The depth of root r itself is zero.

Input

The first line contains two space-separated integers n and k n k .

The next n lines each contain two integers x and y x y n x y , denoting an undirected edge between the two nodes x and y. These edges form a tree of n nodes.

The next line contains n space-separated integers denoting the array a a_i .

Output

Output n integers, where the i-th integer is if Alice wins the game when the tree is rooted at node i, or otherwise.

Example

input

```
5 1
1 2
1 3
5 2
4 3
0 3 2 4 4
```

output

```
1 0 0 1 1
```

Note

Let us calculate the answer for sample input with root node as 1 and as 2.

Root node 1

Alice always wins in this case. One possible gameplay between Alice and Bob is:

- Alice moves one present from node 4 to node 3.
- Bob moves four presents from node 5 to node 2.
- Alice moves four presents from node 2 to node 1.
- Bob moves three presents from node 2 to node 1.
- Alice moves three presents from node 3 to node 1.
- Bob moves three presents from node 4 to node 3.
- Alice moves three presents from node 3 to node 1.

Bob is now unable to make a move and hence loses.

Root node 2

Bob always wins in this case. One such gameplay is:

- Alice moves four presents from node 4 to node 3.
- Bob moves four presents from node 5 to node 2.
- Alice moves six presents from node 3 to node 1.
- Bob moves six presents from node 1 to node 2.

Alice is now unable to make a move and hence loses.