# A. Phoenix and Gold

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

Phoenix has collected $n$ pieces of gold, and he wants to weigh them together so he can feel rich. The $i$-th piece of gold has weight $w_i$. All weights are **distinct**. He will put his $n$ pieces of gold on a weight scale, one piece at a time.

The scale has an unusual defect: if the total weight on it is **exactly** $x$, it will explode. Can he put all $n$ gold pieces onto the scale in some order, without the scale exploding during the process? If so, help him find some possible order.

Formally, rearrange the array $w$ so that for each $i$ $(1 \le i \le n)$, $\sum_{j=1}^{i} w_j \neq x$.

## Input

The input consists of multiple test cases. The first line contains an integer $t$ $(1 \le t \le 1000)$ — the number of test cases.

The first line of each test case contains two integers $n$ and $x$ $(1 \le n \le 100; 1 \le x \le 10^4)$ — the number of gold pieces that Phoenix has and the weight to avoid, respectively.

The second line of each test case contains $n$ space-separated integers $(1 \le w_i \le 100)$ — the weights of the gold pieces. It is guaranteed that the weights are **pairwise distinct**.

## Output

For each test case, if Phoenix cannot place all $n$ pieces without the scale exploding, print `NO`. Otherwise, print `YES` followed by the rearranged array $w$. If there are multiple solutions, print any.

**Example**

| input |
| --- |
| 3<br>3 2<br>3 2 1<br>5 3<br>1 2 3 4 8<br>1 5<br>5 |
| **output** |
| YES<br>3 2 1<br>YES<br>8 1 2 3 4<br>NO |

## Note

In the first test case, Phoenix puts the gold piece with weight $3$ on the scale first, then the piece with weight $2$, and finally the piece with weight $1$. The total weight on the scale is $3$, then $5$, then $6$. The scale does not explode because the total weight on the scale is never $2$.

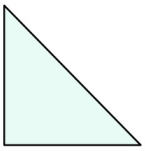
In the second test case, the total weight on the scale is $8$, $9$, $11$, $14$, then $18$. It is never $3$.

In the third test case, Phoenix must put the gold piece with weight $5$ on the scale, and the scale will always explode.

# B. Phoenix and Puzzle

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

Phoenix is playing with a new puzzle, which consists of $n$ identical puzzle pieces. Each puzzle piece is a right isosceles triangle as shown below.



A puzzle piece

The goal of the puzzle is to create a **square** using the $n$ pieces. He is allowed to rotate and move the pieces around, but none of them can overlap and all $n$ pieces must be used (of course, the square shouldn't contain any holes as well). Can he do it?

### Input

The input consists of multiple test cases. The first line contains an integer $t$ ($1 \leq t \leq 10^4$) — the number of test cases.

The first line of each test case contains an integer $n$ ($1 \leq n \leq 10^9$) — the number of puzzle pieces.

### Output

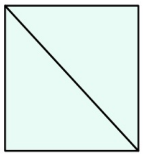
For each test case, if Phoenix can create a square with the $n$ puzzle pieces, print YES. Otherwise, print NO.

**Example**

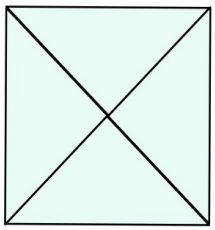
| input |
| --- |
| 3<br>2<br>4<br>6 |
| **output** |
| YES<br>YES<br>NO |

### Note

For $n = 2$, Phoenix can create a square like this:



For $n = 4$, Phoenix can create a square like this:



For $n = 6$, it is impossible for Phoenix to create a square.

## C. Phoenix and Towers

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

Phoenix has $n$ blocks of height $h_1, h_2, \ldots, h_n$, and all $h_i$ don't exceed some value $x$. He plans to stack all $n$ blocks into $m$ separate towers. The height of a tower is simply the sum of the heights of its blocks. For the towers to look beautiful, no two towers may have a height difference of strictly more than $x$.

Please help Phoenix build $m$ towers that look beautiful. Each tower must have *at least one block* and all blocks must be used.

### Input

The input consists of multiple test cases. The first line contains an integer $t$ ($1 \leq t \leq 1000$) — the number of test cases.

The first line of each test case contains three integers $n$, $m$, and $x$ ($1 \leq m \leq n \leq 10^5$; $1 \leq x \leq 10^4$) — the number of blocks, the number of towers to build, and the maximum acceptable height difference of any two towers, respectively.

The second line of each test case contains $n$ space-separated integers ($1 \leq h_i \leq x$) — the heights of the blocks.

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

### Output

For each test case, if Phoenix cannot build $m$ towers that look beautiful, print NO. Otherwise, print YES, followed by $n$ integers

$y_1, y_2, \ldots, y_n$, where $y_i$ ($1 \le y_i \le m$) is the index of the tower that the $i$-th block is placed in.

If there are multiple solutions, print any of them.

**Example**

| input |
| --- |
| 2<br>5 2 3<br>1 2 3 1 2<br>4 3 3<br>1 1 2 3 |
| **output** |
| YES<br>1 1 1 2 2<br>YES<br>1 2 2 3 |

### Note

In the first test case, the first tower has height $1 + 2 + 3 = 6$ and the second tower has height $1 + 2 = 3$. Their difference is $6 - 3 = 3$ which doesn't exceed $x = 3$, so the towers are beautiful.

In the second test case, the first tower has height $1$, the second tower has height $1 + 2 = 3$, and the third tower has height $3$. The maximum height difference of any two towers is $3 - 1 = 2$ which doesn't exceed $x = 3$, so the towers are beautiful.

## D. Phoenix and Socks

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

To satisfy his love of matching socks, Phoenix has brought his $n$ socks ($n$ is even) to the sock store. Each of his socks has a color $c_i$ and is either a left sock or right sock.

Phoenix can pay one dollar to the sock store to either:

- recolor a sock to any color $c'$ ($1 \le c' \le n$)
- turn a left sock into a right sock
- turn a right sock into a left sock

The sock store may perform each of these changes any number of times. Note that the color of a left sock doesn't change when it turns into a right sock, and vice versa.

A matching pair of socks is a left and right sock with the same color. What is the minimum cost for Phoenix to make $n/2$ matching pairs? Each sock must be included in exactly one matching pair.

### Input

The input consists of multiple test cases. The first line contains an integer $t$ ($1 \le t \le 1000$) — the number of test cases.

The first line of each test case contains three integers $n$, $l$, and $r$ ($2 \le n \le 2 \cdot 10^5$; $n$ is even; $0 \le l, r \le n$; $l + r = n$) — the total number of socks, and the number of left and right socks, respectively.

The next line contains $n$ integers $c_i$ ($1 \le c_i \le n$) — the colors of the socks. The first $l$ socks are left socks, while the next $r$ socks are right socks.

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

### Output

For each test case, print one integer — the minimum cost for Phoenix to make $n/2$ matching pairs. Each sock must be included in exactly one matching pair.

**Example**

| input |
| --- |
| 4<br>6 3 3<br>1 2 3 2 2 2<br>6 2 4<br>1 1 2 2 2 2<br>6 5 1<br>6 5 4 3 2 1<br>4 0 4<br>4 4 4 3 |
| **output** |
| 2<br>3<br>5<br>3 |

### Note
In the first test case, Phoenix can pay $2$ dollars to:

- recolor sock $1$ to color $2$
- recolor sock $3$ to color $2$

There are now $3$ matching pairs. For example, pairs $(1, 4)$, $(2, 5)$, and $(3, 6)$ are matching.
In the second test case, Phoenix can pay $3$ dollars to:

- turn sock $6$ from a right sock to a left sock
- recolor sock $3$ to color $1$
- recolor sock $4$ to color $1$

There are now $3$ matching pairs. For example, pairs $(1, 3)$, $(2, 4)$, and $(5, 6)$ are matching.

## E. Phoenix and Computers

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

There are $n$ computers in a row, all originally off, and Phoenix wants to turn all of them on. He will manually turn on computers one at a time. At any point, if computer $i-1$ and computer $i+1$ are both on, computer $i$ $(2 \leq i \leq n-1)$ will turn on automatically if it is not already on. Note that Phoenix cannot manually turn on a computer that already turned on automatically.

If we only consider the sequence of computers that Phoenix turns on manually, how many ways can he turn on all the computers? Two sequences are distinct if either the set of computers turned on manually is distinct, or the order of computers turned on manually is distinct. Since this number may be large, please print it modulo $M$.

### Input
The first line contains two integers $n$ and $M$ ($3 \leq n \leq 400$; $10^8 \leq M \leq 10^9$) — the number of computers and the modulo. It is guaranteed that $M$ is prime.

### Output
Print one integer — the number of ways to turn on the computers modulo $M$.

#### Examples

| input |
| --- |
| 3 100000007 |
| **output** |
| 6 |

| input |
| --- |
| 4 100000007 |
| **output** |
| 20 |

| input |
| --- |
| 400 234567899 |
| **output** |
| 20914007 |

### Note
In the first example, these are the $6$ orders in which Phoenix can turn on all computers:

- $[1, 3]$. Turn on computer $1$, then $3$. Note that computer $2$ turns on automatically after computer $3$ is turned on manually, but we only consider the sequence of computers that are turned on manually.
- $[3, 1]$. Turn on computer $3$, then $1$.
- $[1, 2, 3]$. Turn on computer $1$, $2$, then $3$.
- $[2, 1, 3]$
- $[2, 3, 1]$
- $[3, 2, 1]$

## F. Phoenix and Earthquake

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

Phoenix's homeland, the Fire Nation had $n$ cities that were connected by $m$ roads, but the roads were all destroyed by an earthquake. The Fire Nation wishes to repair $n-1$ of these roads so that all the cities are connected again.

The $i$-th city has $a_i$ tons of asphalt. $x$ tons of asphalt are used up when repairing a road, and to repair a road between $i$ and $j$, cities $i$ and $j$ must have at least $x$ tons of asphalt between them. In other words, if city $i$ had $a_i$ tons of asphalt and city $j$ had $a_j$ tons, there would remain $a_i + a_j - x$ tons after repairing the road between them. Asphalt can be moved between cities if the road between them is already repaired.

Please determine if it is possible to connect all the cities, and if so, output any sequence of roads to repair.

## Input

The first line contains integers $n$, $m$, and $x$ ($2 \leq n \leq 3 \cdot 10^5$; $n-1 \leq m \leq 3 \cdot 10^5$; $1 \leq x \leq 10^9$) — the number of cities, number of roads, and amount of asphalt needed to repair one road.

The next line contains $n$ space-separated integer $a_i$ ($0 \leq a_i \leq 10^9$) — the amount of asphalt initially at city $i$.

The next $m$ lines contains two integers $x_i$ and $y_i$ ($x_i \neq y_i$; $1 \leq x_i, y_i \leq n$) — the cities connected by the $i$-th road. It is guaranteed that there is at most one road between each pair of cities, and that the city was originally connected before the earthquake.

## Output

If it is not possible to connect all the cities, print NO. Otherwise, print YES followed by $n-1$ integers $e_1, e_2, \ldots, e_{n-1}$, the order in which the roads should be repaired. $e_i$ is the index of the $i$-th road to repair. If there are multiple solutions, print any.

**Examples**

| input |
| --- |
| 5 4 1<br>0 0 0 4 0<br>1 2<br>2 3<br>3 4<br>4 5 |
| **output** |
| YES<br>3<br>2<br>1<br>4 |

| input |
| --- |
| 2 1 2<br>1 1<br>1 2 |
| **output** |
| YES<br>1 |

| input |
| --- |
| 2 1 2<br>0 1<br>1 2 |
| **output** |
| NO |

| input |
| --- |
| 5 6 5<br>0 9 4 0 10<br>1 2<br>1 3<br>2 3<br>3 4<br>1 4<br>4 5 |
| **output** |
| YES<br>6<br>4<br>1<br>2 |

## Note

In the first example, the roads are repaired in the following order:

- Road $3$ is repaired, connecting cities $3$ and $4$. City $4$ originally had $4$ tons of asphalt. After this road is constructed, $3$ tons remain.
- Road $2$ is repaired, connecting cities $2$ and $3$. The asphalt from city $4$ can be transported to city $3$ and used for the road. $2$ tons remain.

- Road $1$ is repaired, connecting cities $1$ and $2$. The asphalt is transported to city $2$ and used for the road. $1$ ton remain.
- Road $4$ is repaired, connecting cities $4$ and $5$. The asphalt is transported to city $4$ and used for the road. No asphalt remains.

All the cities are now connected.
In the second example, cities $1$ and $2$ use all their asphalt together to build the road. They each have $1$ ton, so together they have $2$ tons, which is enough.

In the third example, there isn't enough asphalt to connect cities $1$ and $2$.

## G. Phoenix and Odometers

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

In Fire City, there are $n$ intersections and $m$ one-way roads. The $i$-th road goes from intersection $a_i$ to $b_i$ and has length $l_i$ miles.

There are $q$ cars that may only drive along those roads. The $i$-th car starts at intersection $v_i$ and has an odometer that begins at $s_i$, increments for each mile driven, and resets to $0$ whenever it reaches $t_i$. Phoenix has been tasked to drive cars along some roads (possibly none) and **return them to their initial intersection** with the odometer showing $0$.

For each car, please find if this is possible.

A car may visit the same road or intersection an arbitrary number of times. The odometers don't stop counting the distance after resetting, so odometers may also be reset an arbitrary number of times.

### Input
The first line of the input contains two integers $n$ and $m$ ($2 \le n \le 2 \cdot 10^5$; $1 \le m \le 2 \cdot 10^5$) — the number of intersections and the number of roads, respectively.

Each of the next $m$ lines contain three integers $a_i$, $b_i$, and $l_i$ ($1 \le a_i, b_i \le n$; $a_i \ne b_i$; $1 \le l_i \le 10^9$) — the information about the $i$-th road. The graph is not necessarily connected. It is guaranteed that between any two intersections, there is at most one road for each direction.

The next line contains an integer $q$ ($1 \le q \le 2 \cdot 10^5$) — the number of cars.

Each of the next $q$ lines contains three integers $v_i$, $s_i$, and $t_i$ ($1 \le v_i \le n$; $0 \le s_i < t_i \le 10^9$) — the initial intersection of the $i$-th car, the initial number on the $i$-th odometer, and the number at which the $i$-th odometer resets, respectively.

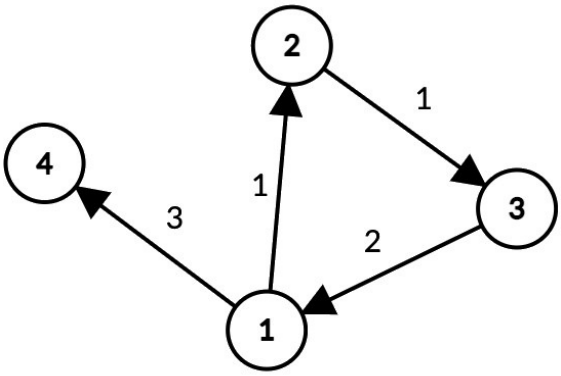
### Output
Print $q$ answers. If the $i$-th car's odometer may be reset to $0$ by driving through some roads (possibly none) and returning to its starting intersection $v_i$, print YES. Otherwise, print NO.

**Examples**

| input |
| --- |
| 4 4<br>1 2 1<br>2 3 1<br>3 1 2<br>1 4 3<br>3<br>1 1 3<br>1 2 4<br>4 0 1 |
| **output** |
| YES<br>NO<br>YES |

| input |
| --- |
| 4 5<br>1 2 1<br>2 3 1<br>3 1 2<br>1 4 1<br>4 3 2<br>2<br>1 2 4<br>4 3 5 |
| **output** |
| YES<br>YES |

### Note
The illustration for the first example is below:

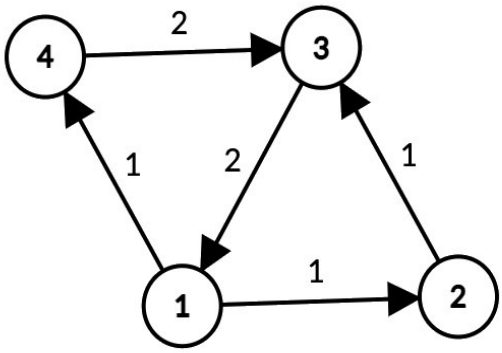



In the first query, Phoenix can drive through the following cities: $1 \to 2 \to 3 \to 1 \to 2 \to 3 \to 1$. The odometer will have reset $3$ times, but it displays $0$ at the end.

In the second query, we can show that there is no way to reset the odometer to $0$ and return to intersection $1$.

In the third query, the odometer already displays $0$, so there is no need to drive through any roads.

Below is the illustration for the second example:




## H. Phoenix and Bits

time limit per test: 4 seconds
memory limit per test: 512 megabytes
input: standard input
output: standard output

Phoenix loves playing with bits — specifically, by using the bitwise operations AND, OR, and XOR. He has $n$ integers $a_1, a_2, \ldots, a_n$, and will perform $q$ of the following queries:

1. replace all numbers $a_i$ where $l \leq a_i \leq r$ with $a_i$ AND $x$;
2. replace all numbers $a_i$ where $l \leq a_i \leq r$ with $a_i$ OR $x$;
3. replace all numbers $a_i$ where $l \leq a_i \leq r$ with $a_i$ XOR $x$;
4. output how many **distinct** integers $a_i$ where $l \leq a_i \leq r$.

For each query, Phoenix is given $l$, $r$, and $x$. Note that he is considering the values of the numbers, not their indices.

### Input

The first line contains two integers $n$ and $q$ ($1 \leq n \leq 2 \cdot 10^5$; $1 \leq q \leq 10^5$) — the number of integers and the number of queries, respectively.

The second line contains $n$ integers $a_1, a_2, \ldots, a_n$ ($0 \leq a_i < 2^{20}$) — the integers that Phoenix starts with.

The next $q$ lines contain the queries. For each query, the first integer of each line is $t$ ($1 \leq t \leq 4$) — the type of query.

If $t \in \{1, 2, 3\}$, then three integers $l_i$, $r_i$, and $x_i$ will follow ($0 \leq l_i, r_i, x_i < 2^{20}$; $l_i \leq r_i$).

Otherwise, if $t = 4$, two integers $l_i$ and $r_i$ will follow ($0 \leq l_i \leq r_i < 2^{20}$).

It is guaranteed that there is at least one query where $t = 4$.

### Output

Print the answer for each query where $t = 4$.

**Examples**

| input |
| --- |
| 5 6<br>5 4 3 2 1 |

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

**output**

```
3
2
1
```

**input**

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

**output**

```
5
1
2
```

### Note

In the first example:

- For the first query, $2$ is replaced by $2$ AND $2 = 2$ and $3$ is replaced with $3$ AND $2 = 2$. The set of numbers is $\{1, 2, 4, 5\}$.
- For the second query, there are $3$ distinct numbers between $2$ and $5$: $2$, $4$, and $5$.
- For the third query, $2$ is replaced by $2$ XOR $3 = 1$, $4$ is replaced by $4$ XOR $3 = 7$, and $5$ is replaced by $5$ XOR $3 = 6$. The set of numbers is $\{1, 6, 7\}$.
- For the fourth query, there are $2$ distinct numbers between $1$ and $6$: $1$ and $6$.
- For the fifth query, $1$ is replaced by $1$ OR $8 = 9$. The set of numbers is $\{6, 7, 9\}$.
- For the sixth query, there is one distinct number between $8$ and $10$: $9$.

## I. Phoenix and Diamonds

time limit per test: 5 seconds
memory limit per test: 1024 megabytes
input: standard input
output: standard output

Phoenix wonders what it is like to rob diamonds from a jewelry store!

There are $n$ types of diamonds. The $i$-th type has weight $w_i$ and value $v_i$. The store initially has $a_i$ diamonds of the $i$-th type.

Each day, for $q$ days, one of the following will happen:

1. A new shipment of $k_i$ diamonds of type $d_i$ arrive.
2. The store sells $k_i$ diamonds of type $d_i$.
3. Phoenix wonders what will happen if he robs the store using a bag that can fit diamonds with total weight not exceeding $c_i$. If he greedily takes diamonds of the largest value that fit, how much value would be taken? If there are multiple diamonds with the largest value, he will take the one with minimum weight. If, of the diamonds with the largest value, there are multiple with the same minimum weight, he will take any of them.

Of course, since Phoenix is a law-abiding citizen, this is all a thought experiment and he never actually robs any diamonds from the store. This means that queries of type $3$ do not affect the diamonds in the store.

### Input

The first line contains two integers $n$ and $q$ ($1 \le n \le 2 \cdot 10^5$; $1 \le q \le 10^5$) — the number of types of diamonds and number of days, respectively.

The next $n$ lines describe each type of diamond. The $i$-th line will contain three integers $a_i$, $w_i$, and $v_i$ ($0 \le a_i \le 10^5$; $1 \le w_i, v_i \le 10^5$) — the initial number of diamonds of the $i$-th type, the weight of diamonds of the $i$-th type, and the value of diamonds of the $i$-th type, respectively.

The next $q$ lines contain the queries. For each query, the first integer of each line is $t$ ($1 \le t \le 3$) — the type of query.

If $t = 1$, then two integers $k_i$, $d_i$ follow ($1 \le k_i \le 10^5$; $1 \le d_i \le n$). This means that a new shipment of $k_i$ diamonds arrived, each of type $d_i$.

If $t = 2$, then two integers $k_i$, $d_i$ follow ($1 \le k_i \le 10^5$; $1 \le d_i \le n$). This means that the store has sold $k_i$ diamonds, each of type $d_i$. It is guaranteed that the store had the diamonds before they sold them.

If $t = 3$, an integer $c_i$ will follow ($1 \le c_i \le 10^{18}$) — the weight capacity of Phoenix's bag.

It is guaranteed that there is at least one query where $t = 3$.

## Output

Print the answer for each query of the third type ($t = 3$).

**Example**

| input |
| --- |
| 3 5<br>2 3 4<br>1 5 1<br>0 2 4<br>3 6<br>1 3 3<br>3 10<br>2 2 3<br>3 30 |
| **output** |
| 8<br>16<br>13 |

## Note

For the first query where $t = 3$, Phoenix can fit $2$ diamonds of type $1$, with total weight $6$ and value $8$.

For the second query where $t = 3$, Phoenix will first fit in $3$ diamonds of type $3$, then one diamond of type $1$ for a total weight of $9$ and a value of $16$. Note that diamonds of type $3$ are prioritized over type $1$ because type $3$ has equal value but less weight.

For the final query where $t = 3$, Phoenix can fit every diamond for a total value of $13$.