Codeforces Round #703 (Div. 2)

A. Shifting Stacks

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

You have n stacks of blocks. The i-th stack contains h_i blocks and it's height is the number of blocks in it. In one move you can take a block from the i-th stack (if there is at least one block) and put it to the $i+1$-th stack. Can you make the sequence of heights strictly increasing?

Note that the number of stacks always remains n: stacks don't disappear when they have 0 blocks.

Input

First line contains a single integer t $(1 \le t \le 10^4)$ — the number of test cases.

The first line of each test case contains a single integer n $(1 \le n \le 100)$. The second line of each test case contains n integers h_i $(0 \le h_i \le 10^9)$ — starting heights of the stacks.

It's guaranteed that the sum of all n does not exceed 10^4.

Output

For each test case output YES if you can make the sequence of heights strictly increasing and NO otherwise.

You may print each letter in any case (for example, YES, Yes, yes, yEs will all be recognized as positive answer).

Example

input

```
6
2
1 2
2
1 0
3
4 4 4
2
0 0
3
0 1 0
4
1000000000 1000000000 1000000000 1000000000
```

output

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

Note

In the first test case there is no need to make any moves, the sequence of heights is already increasing.

In the second test case we need to move one block from the first stack to the second. Then the heights become 0 1.

In the third test case we could move one block from the first stack to the second and then from the second to the third, which would make the heights 3 4 5.

In the fourth test case we can't make a move, but the sequence is not increasing, so the answer is NO.

In the fifth test case we can only make one move (from the second to the third stack), which would make the heights 0 0 1. Both 0 1 0 and 0 0 1 are not increasing sequences, so the answer is NO.

B. Eastern Exhibition

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

You and your friends live in n houses. Each house is located on a 2D plane, in a point with integer coordinates. There might be different houses located in the same point. The mayor of the city is asking you for places for the building of the Eastern exhibition. You have to find the number of places (points with integer coordinates), so that the summary distance from all the houses to the exhibition is minimal. The exhibition can be built in the same point as some house. The distance between two points (x_1, y_1) and (x_2, y_2) is $|x_1 - x_2| + |y_1 - y_2|$, where $|x|$ is the absolute value of x.

Input

First line contains a single integer t $(1 \le t \le 1000)$ — the number of test cases.

The first line of each test case contains a single integer n $(1 \leq n \leq 1000)$. Next n lines describe the positions of the houses (x_i, y_i) $(0 \leq x_i, y_i \leq 10^9)$.

It's guaranteed that the sum of all n does not exceed 1000.

Output

For each test case output a single integer - the number of different positions for the exhibition. The exhibition can be built in the same point as some house.

Example

input

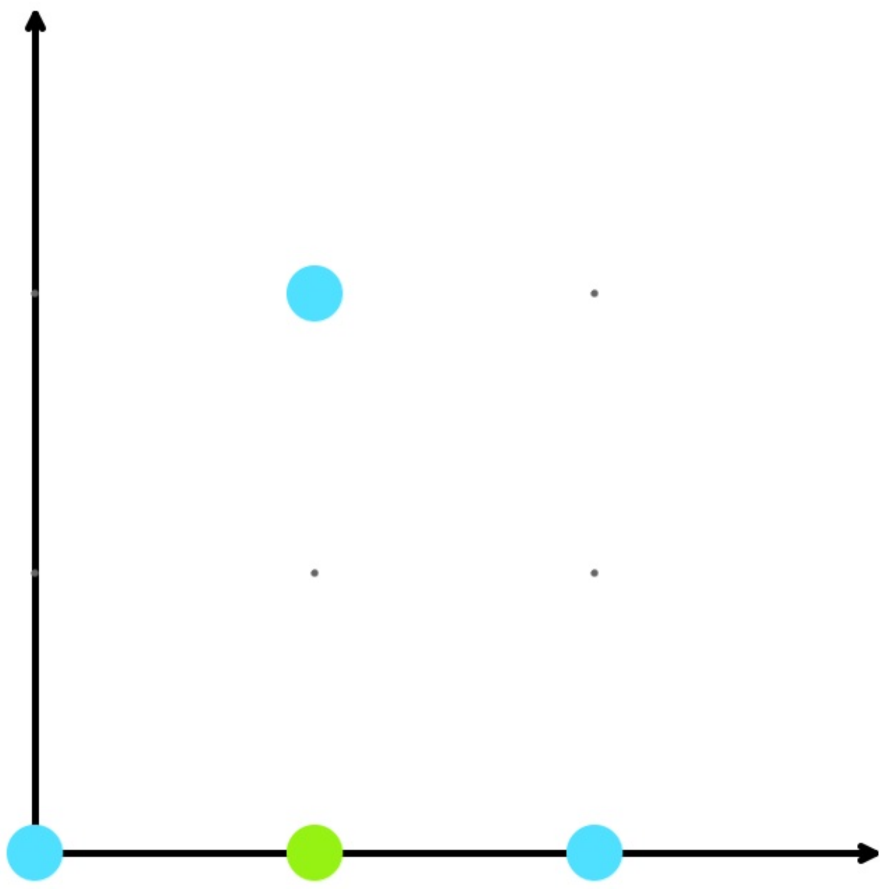
```
6
3
0 0
2 0
1 2
4
1 0
0 2
2 3
3 1
4
0 0
0 1
1 0
1 1
2
0 0
1 1
2
0 0
2 0
2
0 0
0 0
```

output

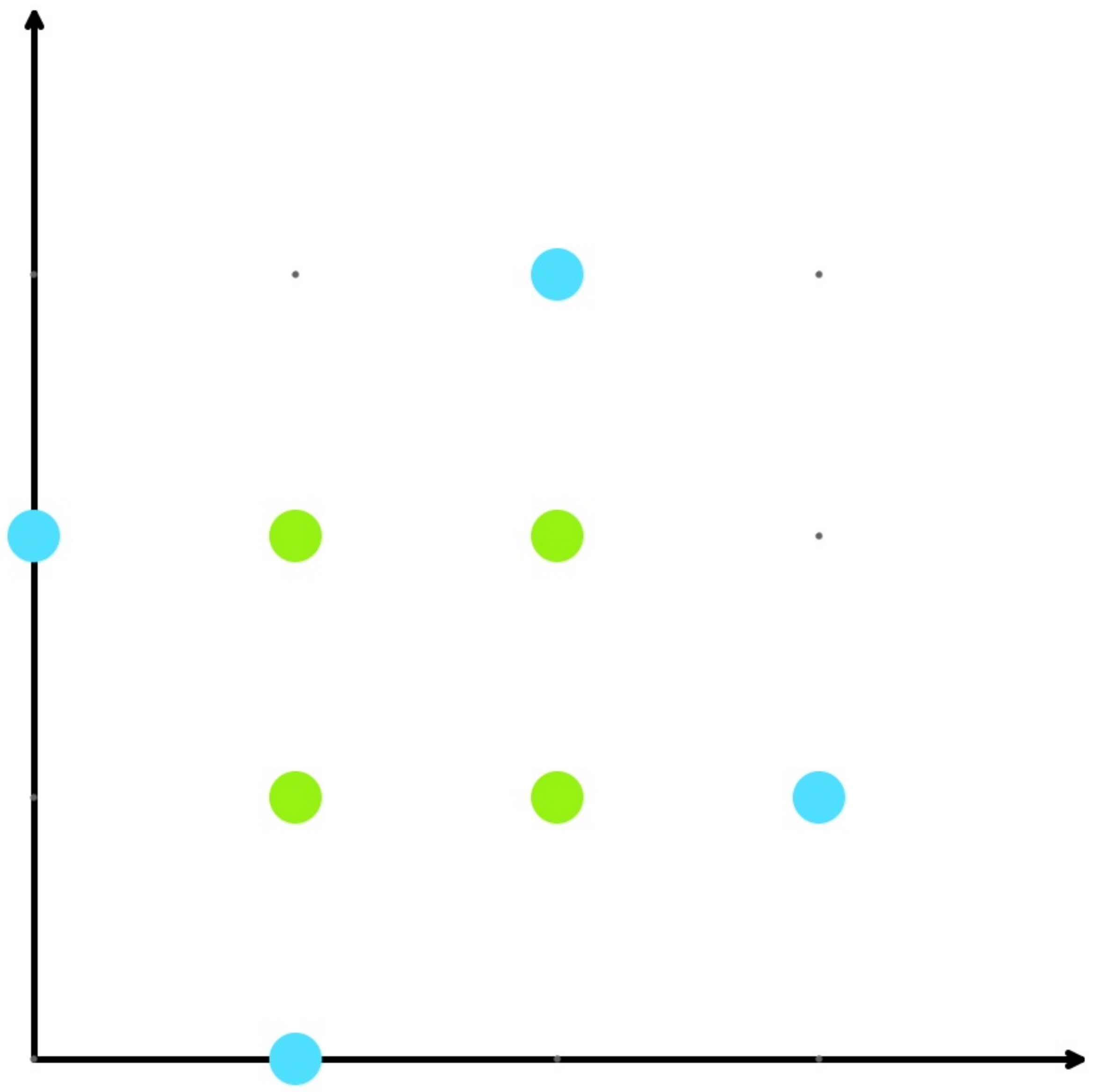
```
1
4
4
4
3
1
```

Note

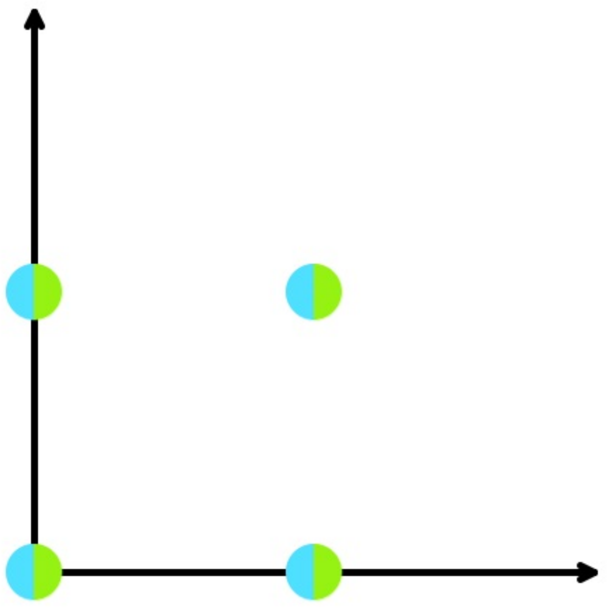
Here are the images for the example test cases. Blue dots stand for the houses, green — possible positions for the exhibition.



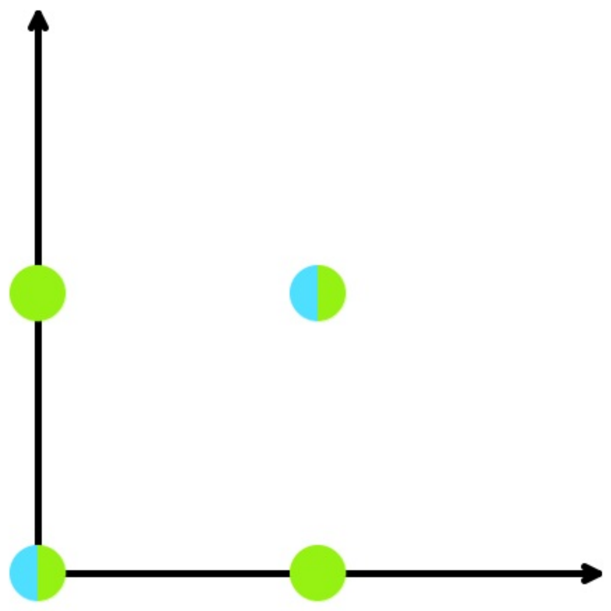
First test case.



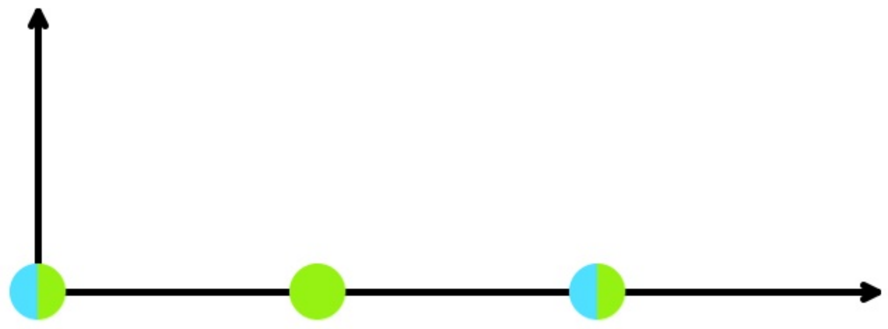
Second test case.



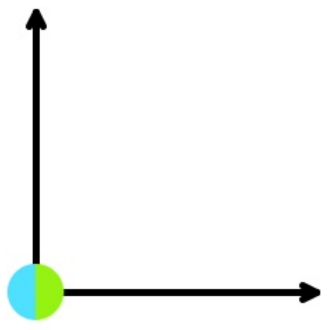
Third test case.



Fourth test case.



Fifth test case.



Sixth test case. Here both houses are located at $(0, 0)$.

C1. Guessing the Greatest (easy version)

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

The only difference between the easy and the hard version is the limit to the number of queries.

This is an interactive problem.

There is an array a of n **different** numbers. In one query you can ask the position of the second maximum element in a subsegment $a[l..r]$. Find the position of the maximum element in the array in no more than **40** queries.

A subsegment $a[l..r]$ is all the elements $a_l, a_{l+1}, ..., a_r$. After asking this subsegment you will be given the position of the second maximum from this subsegment **in the whole** array.

Input

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

Interaction

You can ask queries by printing "? l r" $(1 \le l < r \le n)$. The answer is the index of the second maximum of all elements $a_l, a_{l+1}, ..., a_r$. Array a is fixed beforehand and can't be changed in time of interaction.

You can output the answer by printing "! p", where p is the index of the maximum element in the array.

You can ask no more than **40** queries. **Printing the answer doesn't count as a query**.

After printing a query do not forget to output end of 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;
- `flush(output)` in Pascal;
- `stdout.flush()` in Python;
- see documentation for other languages

Hacks

To make a hack, use the following test format.

In the first line output a single integer n $(2 \le n \le 10^5)$. In the second line output a permutation of n integers 1 to n. The position of n in the permutation is the position of the maximum

Example

input

```
5

3

4
```

output

```
? 1 5

? 4 5

! 1
```

Note

In the sample suppose a is $[5, 1, 4, 2, 3]$. So after asking the $[1..5]$ subsegment 4 is second to max value, and it's position is 3. After asking the $[4..5]$ subsegment 2 is second to max value and it's position in the whole array is 4.

Note that there are other arrays a that would produce the same interaction, and the answer for them might be different. Example output is given in purpose of understanding the interaction.

C2. Guessing the Greatest (hard version)

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

The only difference between the easy and the hard version is the limit to the number of queries.

This is an interactive problem.

There is an array a of n **different** numbers. In one query you can ask the position of the second maximum element in a subsegment $a[l..r]$. Find the position of the maximum element in the array in no more than **20** queries.

A subsegment $a[l..r]$ is all the elements $a_l, a_{l+1}, ..., a_r$. After asking this subsegment you will be given the position of the second maximum from this subsegment **in the whole** array.

Input

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

Interaction

You can ask queries by printing "? l r" $(1 \le l < r \le n)$. The answer is the index of the second maximum of all elements $a_l, a_{l+1}, ..., a_r$. Array a is fixed beforehand and can't be changed in time of interaction.

You can output the answer by printing "! p", where p is the index of the maximum element in the array.

You can ask no more than **20** queries. **Printing the answer doesn't count as a query**.

After printing a query do not forget to output end of 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;
- `flush(output)` in Pascal;
- `stdout.flush()` in Python;
- see documentation for other languages

Hacks

To make a hack, use the following test format.

In the first line output a single integer n $(2 \le n \le 10^5)$. In the second line output a permutation of n integers 1 to n. The position of n in the permutation is the position of the maximum

Example

input

```
5

3

4
```

output

```
? 1 5

? 4 5

! 1
```

Note

In the sample suppose a is $[5, 1, 4, 2, 3]$. So after asking the $[1..5]$ subsegment 4 is second to max value, and it's position is 3. After asking the $[4..5]$ subsegment 2 is second to max value and it's position in the whole array is 4.

Note that there are other arrays a that would produce the same interaction, and the answer for them might be different. Example output is given in purpose of understanding the interaction.

D. Max Median

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

You are a given an array a of length n. Find a subarray $a[l..r]$ with length at least k with the largest median.

A median in an array of length n is an element which occupies position number $\lfloor \frac{n+1}{2} \rfloor$ after we sort the elements in non-decreasing order. For example: $median([1, 2, 3, 4]) = 2$, $median([3, 2, 1]) = 2$, $median([2, 1, 2, 1]) = 1$.

Subarray $a[l..r]$ is a contiguous part of the array a, i. e. the array $a_l, a_{l+1}, \dots, a_r$ for some $1 \le l \le r \le n$, its length is $r - l + 1$.

Input

The first line contains two integers n and k $(1 \le k \le n \le 2 \cdot 10^5)$.

The second line contains n integers $a_1, a_2, \dots, a_n$ $(1 \le a_i \le n)$.

Output

Output one integer m — the maximum median you can get.

Examples

input

```
5 3
1 2 3 2 1
```

output

```
2
```

input

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

output

```
3
```

Note

In the first example all the possible subarrays are $[1..3]$, $[1..4]$, $[1..5]$, $[2..4]$, $[2..5]$ and $[3..5]$ and the median for all of them is 2, so the maximum possible median is 2 too.

In the second example $median([3..4]) = 3$.

E. Paired Payment

time limit per test: 4 seconds
memory limit per test: 512 megabytes

input: standard input
output: standard output

There are n cities and m bidirectional roads in the country. The roads in the country form an undirected weighted graph. The graph **is not guaranteed to be connected**. Each road has it's own parameter w. You can travel through the roads, but the government made a new law: you can only go through two roads at a time (go from city a to city b and then from city b to city c) and you will have to pay $(w_{ab} + w_{bc})^2$ money to go through those roads. Find out whether it is possible to travel from city 1 to every other city t and what's the minimum amount of money you need to get from 1 to t.

Input

First line contains two integers n, m ($2 \leq n \leq 10^5$, $1 \leq m \leq min(\frac{n \cdot (n-1)}{2}, 2 \cdot 10^5)$).

Next m lines each contain three integers v_i, u_i, w_i ($1 \leq v_i, u_i \leq n$, $1 \leq w_i \leq 50$, $u_i \neq v_i$). It's guaranteed that there are no multiple edges, i.e. for any edge (u_i, v_i) there are no other edges (u_i, v_i) or (v_i, u_i).

Output

For every city t print one integer. If there is no correct path between 1 and t output -1. Otherwise print out the minimum amount of money needed to travel from 1 to t.

Examples

input

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

output

```
0 98 49 25 114
```

input

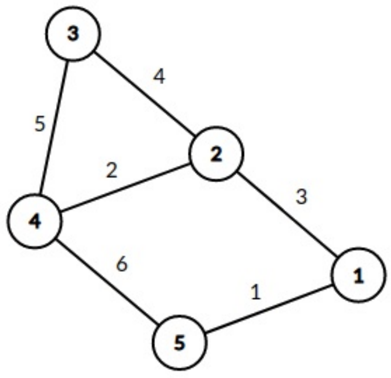
```
3 2
1 2 1
2 3 2
```

output

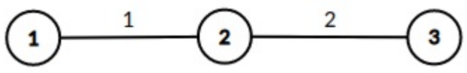
```
0 -1 9
```

Note

The graph in the first example looks like this.



In the second example the path from 1 to 3 goes through 2, so the resulting payment is $(1 + 2)^2 = 9$.



F. Pairs of Paths

time limit per test: 6 seconds
memory limit per test: 512 megabytes

input: standard input
output: standard output

You are given a tree consisting of n vertices, and m simple vertex paths. Your task is to find how many pairs of those paths intersect at exactly one vertex. More formally you have to find the number of pairs (i, j) $(1 \le i < j \le m)$ such that $path_i$ and $path_j$ have exactly one vertex in common.

Input

First line contains a single integer n $(1 \le n \le 3 \cdot 10^5)$.

Next $n-1$ lines describe the tree. Each line contains two integers u and v $(1 \le u, v \le n)$ describing an edge between vertices u and v.

Next line contains a single integer m $(1 \le m \le 3 \cdot 10^5)$.

Next m lines describe paths. Each line describes a path by it's two endpoints u and v $(1 \le u, v \le n)$. The given path is all the vertices on the shortest path from u to v (including u and v).

Output

Output a single integer — the number of pairs of paths that intersect at exactly one vertex.

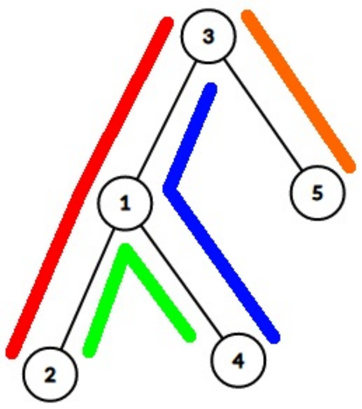
Examples

input
```
5
1 2
1 3
1 4
3 5
4
2 3
2 4
3 4
3 5
```
output
```
2
```

input
```
1
3
1 1
1 1
1 1
```
output
```
3
```

input
```
5
1 2
1 3
1 4
3 5
6
2 3
2 4
3 4
3 5
1 1
1 2
```
output
```
7
```

Note



The tree in the first example and paths look like this. Pairs $(1, 4)$ and $(3, 4)$ intersect at one vertex.

In the second example all three paths contain the same single vertex, so all pairs $(1, 2)$, $(1, 3)$ and $(2, 3)$ intersect at one vertex.

The third example is the same as the first example with two additional paths. Pairs $(1, 4)$, $(1, 5)$, $(2, 5)$, $(3, 4)$, $(3, 5)$, $(3, 6)$ and $(5, 6)$ intersect at one vertex.