1st Annual CSSA Programming Contest
January 20, 2018

General Instructions:

1. This document is printed double-sided. Please read both sides of each page.
2. Submit solutions using the PC^2 software.
3. All input should be read from standard input.
4. All output should be written to standard output.
5. Each problem has a ten (10) second time limit for CPU time when executed on the judging data.

University of Manitoba

Sponsors

A: Are we connected now?

Consider an emerging connectivity network. An example would be train stations at different locations. Say there are N cities, each independently building a train station. At first there aren't any connections between cities, but over time we add rails between them. Hopefully every station will be connected to every other station, but in the interim they may not be. At any particular point in time before the network is fully connected, we want to figure out if two cities are connected by a path in the system.

Each station can be represented as a node. Any given node X is represented by an index, which is in the range $0 \leq X < N$. There are two operations that can be performed on nodes.

The first operation is connection creation, which adds a two way connection between a pair of nodes. The second operation is a query, which determines whether or not a pair of nodes have a path between them.

You must write a program that performs these network operations and outputs the results of any queries.

Input

Each case will start with two space separated integers, $0 \leq N \leq 2000$, and $0 \leq K \leq 10^5$. The integer N represents the number of nodes in the network, and K represents the number of operations to perform.

If line begins with + followed by two nodes indices each separated by a single space (e.g. + x y), then you should perform a connection creation between nodes x and y.

If line begins with ? followed by two nodes indices each separated by a single space (e.g. ? x y), then you should perform a query operation on nodes x and y. Every case contains at least one query.

When N and K are both 0, the program should exit. N and K will never be 0 otherwise.

Output

Cases should start by outputting a line of the form "Case X:" where X is the current case number.

On the following lines, you should output the results of any queries that happen during the execution of the case. Each query output should be on its own line.

Sample Input

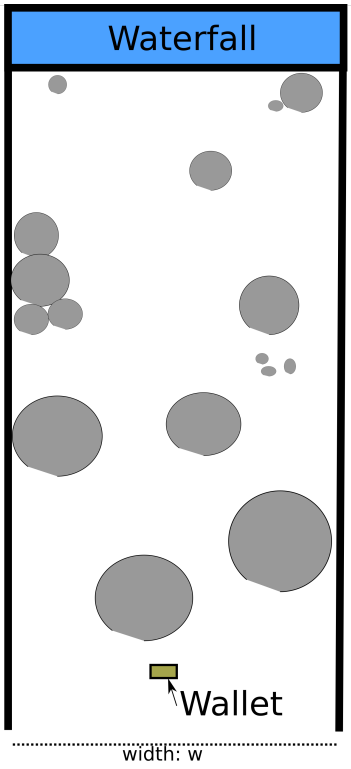
```
5 6
+ 0 2
+ 2 4
? 0 4
? 0 3
+ 3 4
? 0 3
4 4
+ 2 3
? 1 2
+ 1 2
? 1 3
0 0
```

Sample Output

```
Case 1:
true
false
true
Case 2:
false
true
```

B: Cash Flow

You're out walking across a bridge one day and accidentally drop your wallet into the river below you. To your distress, the river ends with a climactic waterfall. You know it won't be long before you can say goodbye to your fresh UPass, you also hope there is a chance your wallet will get stuck on some rocks in the river and be saved. As a computer scientist you accept that this behaviour is beyond your control; but you can determine the outcome. Let us consider the river to span a length W meters, starting from point [0, 0] to point [w,0]. In this river there are multiple rocks, which can be approximated as discs (x,y,r) where x,y are the coordinates and r is the radius of the given rock. Given that your wallet fell into the river at some point upstream (before any obstruction), you need to determine if the wallet can fall off the waterfall or not.



Note:
Assume that your wallet is capable of manoeuvring between any arbitrarily small gap formed between any two rocks and that rocks may overlap.
Assume that you wallet is capable of temporarily flowing upstream around rocks.
Assume that the rocks are tall enough that they go from the bottom of the river to above the water... IE you can't possibly go over the rocks.

Input

Each case will begin with an input containing an integer, W representing the river width and N being the number of rocks in the river. $0 \leq W \leq 100, 0 \leq N \leq 100$ The following N lines each contain 3 space separated doubles $x_i y_i r_i$, where x_i and y_i represent the x and y coordinates of the ith rock and r_i is the radius of the ith rock. Note that $0 \leq x_i \leq W$, $0 \leq y_i \leq 100$, $0 < r_i \leq \frac{W}{2}$ Input is finished once the width and number of rocks are both 0. i.e, you read a line: 0 0.

Output

Output TRUE if your wallet will go off the waterfall (in any possible way) Output: FALSE if your wallet is completely safe, that is there is no possible way for it to reach the waterfall.

Sample Input

```
10 5
1 3 1
8 2 1
5 5 2
2 8 2
9 8 1.5
10 3
0 5 2
4 5 2
8 5 2
0 0
```

Sample Output

```
TRUE
FALSE
```

C: Oar Factory

You're an expert oar-maker working in a high quality oar factory. The factory is sometimes referred to as "The LabOARatOARy" by other oar-makers, and by people who make bad jokes. The oar factory manufactures oars with a specific laminate that cannot be found at any other factory. That being said, the oars from this factory are exclusive oars.

The laminate being applied in this factory is what separates their oars from their competitors. There is a large conveyor belt in the factory that carries oars from the storage area where partially laminated oars are kept, to the laminating section of the factory where the final layer of laminate can be applied. Due to the nature of the oar manufacturing process, and general shape of oars, an oar can be resting in one of two ways. Either the side that needs laminating is facing up, or it is facing down. It's really quite beautiful.

When oars are loaded onto the conveyor belt from storage, there isn't any guarantee about whether the laminated side will be facing up or down when it rests on the belt. All of the oars on the conveyor belt are equally distanced from each other so that they can be easily flipped by the arms from the machines overhead. Additionally, the conveyor belt is configured to move left and right within your control. The only constraint being that there must always be an oar resting underneath each arm of the machine, for safety reasons.

Your job as an oar-maker who had their job automated by the overhead machines, is to find a way to flip the oars such that all of them have their laminate facing down in preparation for the next lamination. This won't always be possible to achieve without human intervention, which is why our mechanical overlords have allowed you to keep your job.

The reason it won't always be possible to make all of the oars face laminate-side down is that the arms of the machine cannot act independently of each other. If you choose to make the machine flip the oars, it will flip every oar that is currently resting underneath one of the machine arms.

Input

The first line contains only an integer $0 \leq n \leq 10000$, representing the number of oar configurations to be tested. The following n lines contain two space separated binary strings, V and K.

The string V is a vector representing the configuration of the oars on the conveyor belt, where a 0 represents an oar with its laminate facing down, and a 1 represents an oar with its laminate facing up. The length of V is in the range $1 \leq |V| \leq 64$.

The string K is a vector representing where the overhead flippers are aligned, where a 1 represents one of the arms that can flip the oar below it, and a 0 represents a gap between the flipping arms (the oar below it will NOT be flipped). K will always start and end with a 1, and the length of K is in the range $1 \leq |K| < |V|$.

Output

For each case, output "YUP" if it is possible to zero the vector, or "NO CAN DO" if it is impossible. Each case should be output on its own line.

Sample Input

```
3
1000101011 1
10001010001 1000001
111001100000000 11
```

Sample Output

```
YUP
YUP
NO CAN DO
```

D: The Adventures of Jumping Janice

Jumping Janice has the incredibly useful super power of being able to jump massive distances ahead on sidewalks. She enjoys waking up to do a morning exercise where she goes to the longest sidewalk in her home town to do a series of jump circuits. The sidewalk is always a great enough length for her entire exercise, and is composed of a series of $1m^2$ concrete tiles.

Each circuit consists of her starting at some tile on the sidewalk, where she then jumps along the sidewalk skipping some number of tiles forward or backward with each jump.

Every tile in the sidewalk has a tile number relative to the circuit's start position. The starting tile will always be tile 0, the next one forward is tile 1, then tile 2, and so on. Take note that Janice can also jump backward, which means it is possible for her to land on tile -10, for example.

Janice has become curious about which tile she most frequently jumps to within each circuit, and wants a way to determine which tile that is.

At the start of the circuit, the starting tile has a single visit because she is standing on it. After that, every jump in the circuit will cause some other tile to be visited.

Input

Each circuit will start with a line of input containing only an integer, $0 \leq n \leq 100$, representing the number of jumps in the circuit.

The following line contains n space separated integers $-2^{31} \leq k_1, k_2, \ldots, k_n \leq 2^{31} - 1$, where k_i represents the distance of the ith jump in a given circuit, and $k_i \neq 0$.

When you reach a circuit that has $n = 0$ jumps, then there are no more cases, and your program should exit.

Output

For each circuit, you should print a line of the form "Circuit #X: Y". X will be the case number, starting from 1, and increasing by one every case. Y will be the tile number of the most frequently visited tile.

If there are multiple tiles that have the highest frequency, you should only output the tile with the lowest tile number.

Sample Input

```
4
-5 -10 5 5
2
10 5
0
```

Sample Output

```
Circuit #1: -5
Circuit #2: 0
```

E: Coconut Calamity

Honeycomb Havoc is a mini-game from Mario Party 2, where you and your friends are trying to catch some fruits, and strangely coins, from a tree. Each player gets to take some items from the tree, then move to the back of the line to wait their turn again. The number of items you catch doesn't really matter, as long as none of them are the dreaded honeycomb. Catching the honeycomb causes the bees to swarm and attack you, which is totally not a good time.

The original Honeycomb Havoc game

In the classic Honeycomb Havoc game, you are allowed to choose between taking either one or two items. In our variant of the game, Coconut Calamity, you must take at least one item from the tree, with a maximum of K items that can be taken in a single turn.

The tree in Coconut Calamity consists of some number of fruits, followed by a series of coconuts. Taking fruits is totally safe, but if you take even just one coconut, they were all topple over on your head, eliminating you from the game and giving you a serious headache.

After some number of turns, all but one of your friends have gotten hit on the head with a coconut. You're entering the final round, and this time it's for all the marbles. There are exactly N fruits remaining in the tree, followed by several coconuts.

It's your turn to take items from the tree. Given that your friend has learned the best strategy possible for any game of Coconut Calamity, and they will always make the best move possible, you must determine whether it is possible for you to win the game or not.

Input

The first line contains only an integer $0 \leq t \leq 10^5$, representing the number of games to play. The following t lines contain two integers, $0 \leq N \leq 10^5$ and $1 \leq K \leq 10^5$, separated by a space.

As mentioned before, the integer N represents the number of fruits remaining before the coconuts are being taken from the tree, and the integer K is the maximum number of items that can be taken from the tree in any one turn.

Output

If you can win the game, output "Winner", or if you're guaranteed to lose the game, output "Loser". Each game's output should be on its own line.

Sample Input

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

Sample Output

```
Winner
Loser
Winner
```

F: Stack It Up

You're working at your part time job - box tower building. As we know box towers are extremely popular with the kids - basically the new fidget spinners. Your job consists of standing by a conveyor belt which brings you perfectly cubed boxes of various sizes and building the tallest possible box stack you can during your shift.

Unfortunately there isn't any order to the size of the boxes and as we all know box towers are built by stacking smaller boxes on top of larger boxes. To make matters worse, due to safety regulations any box you take from the conveyor belt has to be placed directly on top of your current stack and can't be stored for later. You also cannot remove boxes from the stack once placed.

Luckily each day when you go to work you receive a list indicating the size of all the boxes and the order thay the will arrive in on the conveyor belt during your shift. Using your skills as a computer scientist you want to ensure you are building the tallest stack of boxes.

Input

Each case will begin with a positive integer $1 \leq N \leq 1000$ indicating the number of boxes that will be on the conveyor belt during your shift.

The following N lines each contain an integer $1 \leq D \leq 100$ representing the dimensions of one box ordered by occurence.

Any occurrence of 0 indicates that there are no more cases to be executed.

Output

For each case, output the height of the tallest box tower you can build. Each case should be output on its own line.

Sample Input

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

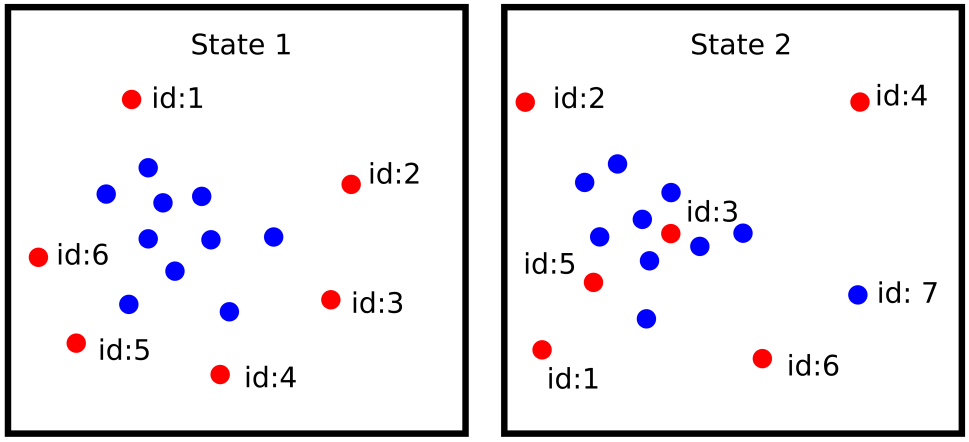
Sample Output

```
2
5
6
```

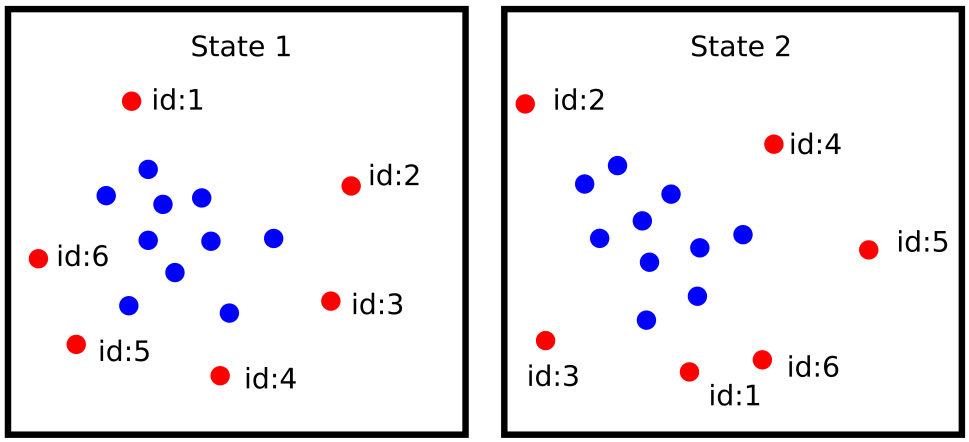
G: Swarms as a Service

It's the year 2050, and SaaS (Swarms as a Service) has become the norm. SaaS involves large groups of robots (called swarms) that deliver consumer goods throughout the skies. As a stationary observer, you want to ensure that all of the robots flying above you have stable dynamics. You do this by sampling the swarms state at two different times (t1, t2). When we look at the swarm between any two states, we first want to determine if the swarm has maintained formation, or if it entering unstable dynamics.

A swarm exhibits stable dynamics if the robots forming the outer boundary of the swarm of state 1, are the same robots in the outer boundary of state 2 (i.e, if you took an elastic band and wrapped it around the robots, the outer boundary is formed by all robots touching the elastic band.)



example 1: Unstable swarm dynamics. Robot 3 has left the outer boundary of the swarm. Furthermore robot 7 becomes part of the outer boundary.



example 2: Stable swarm dynamics. The robots [1,2,3,4,5,6] are surrounding the swarm in both states, despite changing their positions.

Input

Each case will begin with an input containing an integer N being the number of robots in the swarm. $0 \leq N \leq 100$

The following 2N lines contain space separated integers $p_i x_i y_i$, where p_i is the id of the ith robot and x_i and y_i represent the x and y coordinates of the ith robot. The first N of these values represent the state of the N robots at time t_1. The second batch of N lines represents the state of swarm at time t_2

Input is complete once you read in a single "0" indicating a swarm with no robots.

Output

Output TRUE If the swarm is stable between states 1 and 2. Output: FALSE If the swarm is not stable between states 1 and 2.

Sample Input

```
4
1 5 2
2 1 3
3 1 5
4 4 4
1 2 4
2 1 3
3 1 5
4 4 4
3
1 2 4
2 1 3
3 1 5
1 333 666
2 420 69
3 391 5
0
```

Sample Output

```
FALSE
TRUE
```

H: Mean Girls

The girls are back because Hollywood producers are running out of ideas and it is about time they reboot this franchise. Cady, Regina, Gretchen and Karen started cheer leading this year and are very good at it. They have been practicing two formations that they have to get perfectly at nationals.

In the first formation they are positioned in such a way that they form a square. In the second formation they are positioned in such a way that they form a rectangle.

Given all the girls positions, you must output whether they are constructing a square, rectangle, or neither.

Input

The first line contains an integer $0 \leq n \leq 1000$, representing the number of position sets to be tested. The following n lines contains 8 integers separated by spaces, $-10000 \leq x1, y1, x2, y2, x3, y3, x4, y4 \leq 10000$. Each (x_i, y_i) pair represents the position of the cheerleaders in the formation.

Output

If their positions can form a square, then you must output "SQUARE". Otherwise, if the positions can form a rectangle, then you must output "RECTANGLE". In all other cases, you must output "NEITHER".

Sample Input

```
3
-2 0 0 2 2 0 0 -2
-3 0 0 2 3 0 0 -2
1 1 2 3 1 2 3 10
```

Sample Output

```
SQUARE
NEITHER
NEITHER
```