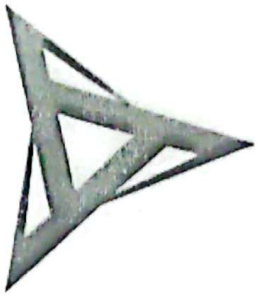
# Welcome to PUCon'24

"First, solve the problem. Then, write the code."

– John Johnson



PUCon'24

*Round 1*

*Programming Competition*

*FCIT*

*University of the Punjab*

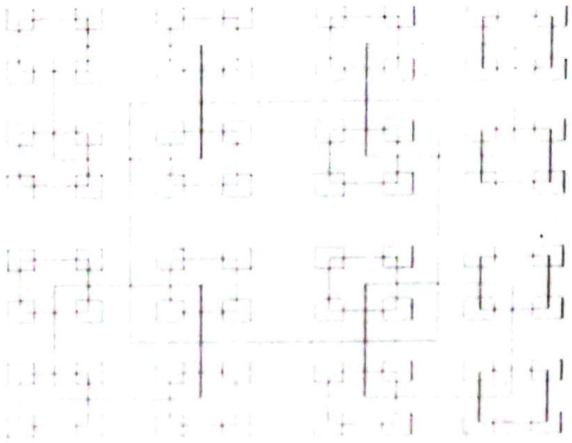
## Problem A

### Mystical land

In a mystical land filled with geometric wonders, a grid of size 1024 by 1024 holds the key to unlocking ancient secrets. Squares, each with a unique center point, dot this enchanted landscape. However, only squares with sides of odd lengths reveal their centers. A square of size k is characterized by sides of length 2k+1.

- The largest square is of size k (that is sides are of length 2k+1) and is centred in a grid of size 1024 (that is the grid sides are of length 2049).
- The smallest permissible square is of size 1 and the largest is of size 512, thus. 1<=k<=512
- All squares of size $k > 1$ have a square of size k div 2 centred on each of their 4 corners. (Div implies integer division, thus 9 div 2 = 4).
- The top left corner of the screen has coordinates (0,0), the bottom right has coordinates (2048, 2048).

Hence, given a value of k, we can draw a unique pattern of squares according to the above rules. Furthermore any point on the screen will be surrounded by zero or more squares. (If the point is on the border of a square, it is considered to be surrounded by that square). Thus if the size of the largest square is given as 15, then the following pattern would be produced.



The task is to develop a program that, given a value of k and the coordinates of a point within the grid, determines how many squares surround that point.

**Input and Output**

Input will consist of a series of lines. Each line will consist of a value of k and the coordinates of a point. The file will be terminated by a line consisting of three zeroes (0 0 0).

Output will consist of a series of lines, one for each line of the input. Each line will consist of the number of squares containing the specified point.

**Sample Input**

500 113 941
300 1024 1024
0 0 0

**Sample Output**

5
1

## Problem B

### Total Payment

An online store has an agreement with a bank for processing of online payments on its behalf. The bank doesn't necessarily have to send the money received for each individual order to the store, instead it can combine payment from multiple orders and send it in a single transaction.
For example, if the bank receives payment of 20, 50 and 80 currency units, it may send it to the store in a single transaction of 150, or two transactions of 70 and 80, or 100 and 50, or 20 and 130, or three transactions of 20, 50 and 80.

Given an integer array **orders** (containing the payment received for an individual order), and an integer **payment** (representing the amount received against multiple orders from the bank), check how many possible combinations of orders the payment can be for.

**Sample Input**
Input will consist of two lines. First representing **orders**, and second representing **payment**.

```
30 20 50 80
150
```

**Sample Output**
Output will be a single number representing the number of different order collections the given payment can relate to.

```
2
```

**Constraints**
1 <= len(orders) <= 32
1 <= orders[i] <= 1000

**Test Case 1**

**Input**
```
30 20 50 80
50
```

**Output**
```
2
```

**Explanation**
{30, 20}, {50}

## Problem C

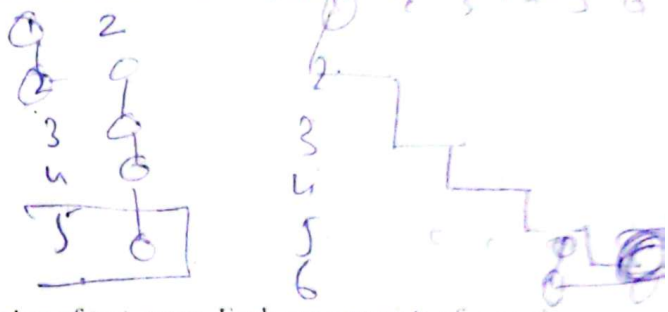
### Dumpling Dynamics

Dumpling Dynamics, a lively local eatery specializing in delectable dumplings, is staffed by two servers, Ayesha and Buraq, who share a keen interest in strategic games. The dining area is depicted as a grid with **R** rows (numbered 1 to R from top to bottom) and **C** columns (numbered 1 to C from left to right).

Currently stationed at coordinates **(1,1)**, beside a cart loaded with savory dumplings, Ayesha and Buraq are on a mission to deliver the cart to a customer situated at coordinates **(R,C)**. To add an element of challenge, they turn the task into a game.

Here's how the game unfolds: Ayesha and Buraq alternate turns pushing the cart. During Ayesha's turn, the cart must move between 1 and **A** units downward. On Buraq's turn, the cart must shift between 1 and **B** units to the right. It's crucial that the cart stays within the grid; if, on Ayesha's turn, the cart is already at row **R** or, on Buraq's turn, it's at column **C**, the respective server forfeits their turn.

The "victor" is the one who successfully guides the cart to coordinates **(R,C)** and receives praise from the customer. The initial push is made by Ayesha. The question at hand is: does Ayesha possess a foolproof winning strategy?

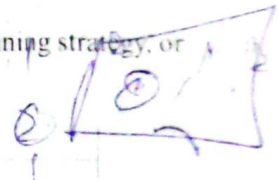


**Constraints:**

- $1 \leq \mathbf{T} \leq 600$
- $2 \leq \mathbf{R}, \mathbf{C} \leq 10^9$
- $1 \leq \mathbf{A} < \mathbf{R}$
- $1 \leq \mathbf{B} < \mathbf{C}$

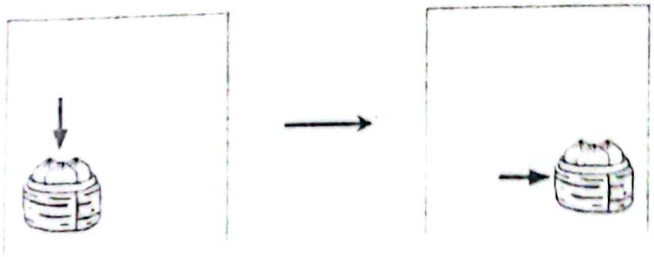
**Input Format:**
The input starts with an integer T, representing the number of test cases. Each case consists of a single line with four space-separated integers: **R**, **C**, **A**, and **B**.

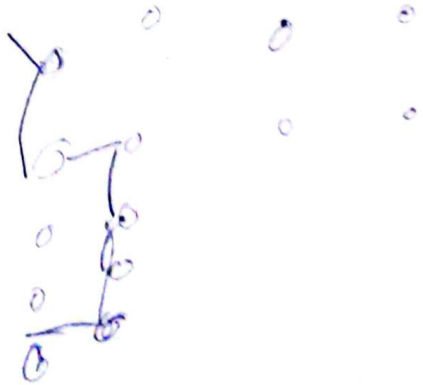
**Output Format:**
For the i-th test case, print "Case #i: " followed by "**YES**" if Ayesha has a guaranteed winning strategy, or "**NO**" otherwise.



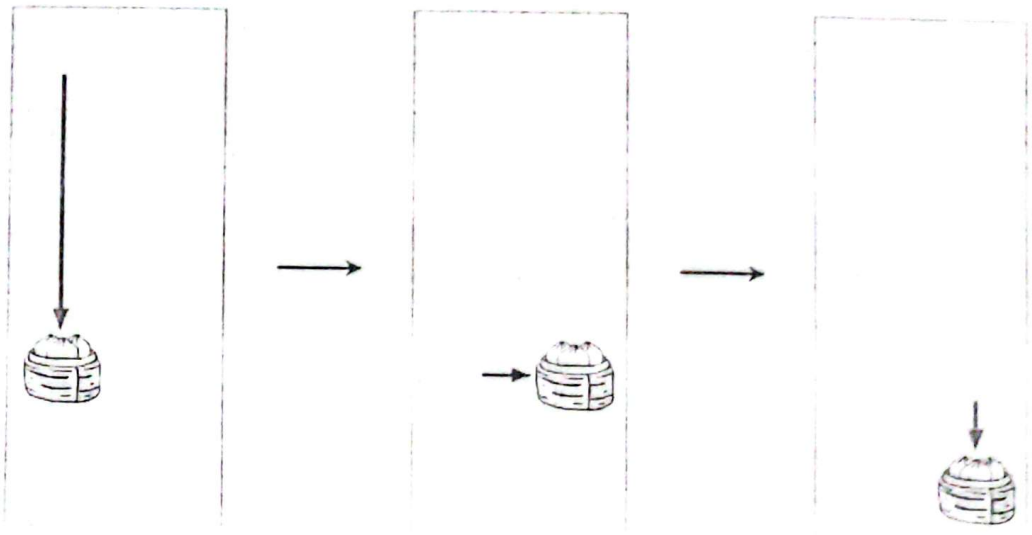
| Sample Input | Sample Output |
|---|---|
| 3<br>2 2 1 1<br>5 2 3 1<br>6 6 4 4 | Case #1: NO<br>Case #2: YES<br>Case #3: NO |

**Sample Explanation:**
For the first scenario, illustrated below, the moves executed by Ayesha are marked in red, while Buraq's actions are denoted in blue. Ayesha initiates with a downward move, and Buraq promptly responds with a rightward push, achieving an immediate triumph. No other valid sequence of moves exists, leading to the conclusion that Ayesha does not possess a foolproof winning strategy.



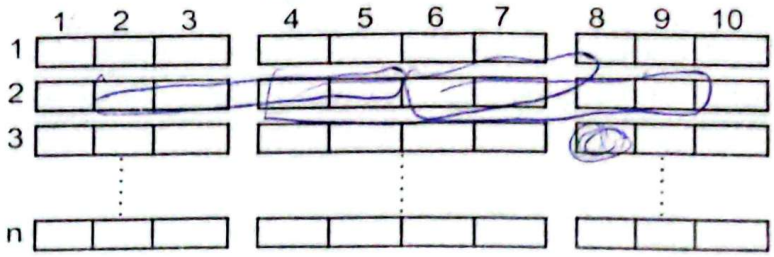


In the second case, as depicted in the visualization below, a potential winning strategy for Ayesha involves the initial movement of 3 units downwards. Subsequently, Buraq's options are limited to a single unit to the right, enabling Aeysha to secure victory with a decisive final move of 1 unit.



## Problem D

**Cinema Seat Allocation**




A cinema has n rows of seats, numbered from 1 to n and there are ten seats in each row, labelled from 1 to 10 as shown in the figure above.

Given the array reservedSeats containing the numbers of seats already reserved, for example, reservedSeats[i] = [3,8] means the seat located in row **3** and labelled with **8** is already reserved.

Return the maximum number of four-person groups you can assign on the cinema seats. A four-person group occupies four adjacent seats **in one single row**. Seats across an aisle (such as [3,3] and [3,4]) are not considered to be adjacent, but there is an exceptional case on which an aisle split a four-person group. in that case, the aisle split a four-person group in the middle, which means to have two people on each side.

**Constraints:**

- $1 <= n <= 10^9$
- $1 <= reservedSeats.length <= min(10*n, 10^4)$
- reservedSeats[i].length == 2
- 1 <= reservedSeats[i][0] <= n
- 1 <= reservedSeats[i][1] <= 10
- All reservedSeats[i] are distinct.

**Input**

The first line of the input shows the number of rows 'n', followed by count of reserved seats 's' followed by s lines, where each line consist of two integers, shows the row number and the reserved seat number.

**Output**

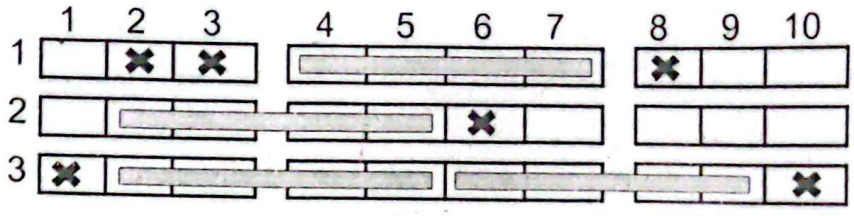
The single integers shows the maximum number of four-person groups you can assign on the cinema seats.

**Sample Input**

```
3
6
1 2
1 3
1 8
2 6
3 1
3 10
```

**Sample Output**

```
4
```




**Input:** n = 3, reservedSeats = [[1,2],[1,3],[1,8],[2,6],[3,1],[3,10]]

**Output:** 4

**Explanation:** The figure above shows the optimal allocation for four groups, where seats mark with blue are already reserved and contiguous seats mark with orange are for one group.