Nikita and the Game

Nikita just came up with a new array game. The rules are as follows:

- Initially, Nikita has an array of integers.
- In each move, Nikita must partition the array into 2 non-empty contiguous parts such that the sum of the elements in the left partition is equal to the sum of the elements in the right partition. If Nikita can make such a move, she gets 1 point; otherwise, the game ends.
- After each successful move, Nikita discards either the left partition or the right partition and continues playing by using the remaining partition as array arr.

Nikita loves this game and wants your help getting the best score possible. Given arr, can you find and print the maximum number of points she can score?

For example, Nikita starts with the array $arr = [1, 2, 3, 6]$. She first splits it into $a1 = [1, 2, 3]$ and $a2 = [6]$, then discards $a2$. $arr = a1 \rightarrow a1 = [1, 2], a2 = [3]$. Discard $a2$ leaving $arr = [1, 2]$. This cannot be further split, so Nikita scored 2.

Function Description

Complete the *arraySplitting* function in the editor below. It should return an integer that reperesents the number of times Nikita can split the array.

arraySplitting has the following parameter(s):

- *arr*: an array of integers

Input Format

The first line contains an integer t, the number of test cases.

Each of the next t pairs of lines is as follows:

- The first line contains an integer n, the size of array arr.
- The next line contains n space-separated integers $arr[i]$.

Constraints

- $1 \le t \le 10$
- $1 \le n \le 2^{14}$
- $0 \le arr[i] \le 10^9$

Scoring

- $1 \le n \le 2^8$ for 30% of the test data
- $1 \le n \le 2^{11}$ for 60% of the test data
- $1 \le n \le 2^{14}$ for 100% of the test data

Output Format

For each test case, print Nikita's maximum possible score on a new line.

Sample Input

```
3
3
3 3 3
4
2 2 2 2
7
4 1 0 1 1 0 1
```

Sample Output

```
0
2
3
```

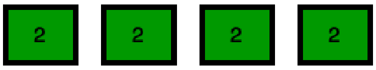
Explanation

Test Case 0:

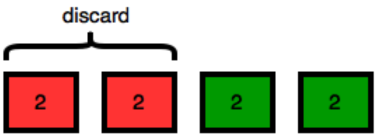
Nikita cannot partition A into 2 parts having equal sums. Therefore, her maximum possible score is 0 and we print 0 on a new line.

Test Case 1:

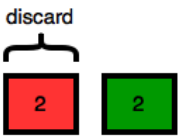
Initially, A looks like this:



She splits the array into 2 partitions having equal sums, and then discards the left partition:



She then splits the new array into 2 partitions having equal sums, and then discards the left partition:



At this point the array only has 1 element and can no longer be partitioned, so the game ends. Because Nikita successfully split the array twice, she gets 2 points and we print 2 on a new line.

Test Case 2:

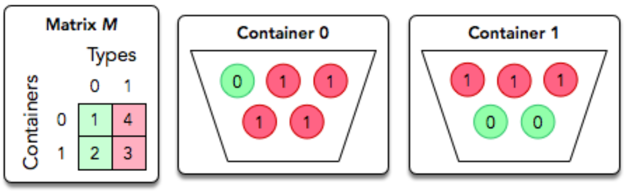
```
array  a1 a2
[4,1,0,1,1,0,1] [4] [1,0,1,1,0,1]
[1,0,1,1,0,1] [1,0,1] [1,0,1]
[1,0,1]  [1,0] [1]
```

The answer is 3.

Organizing Containers of Balls

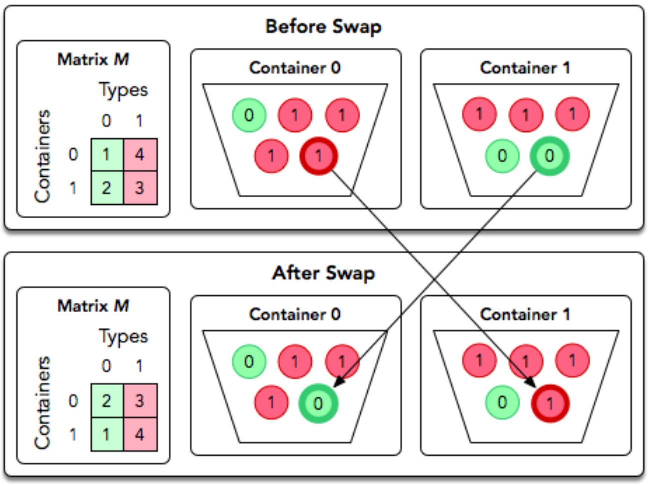
David has several containers, each with a number of balls in it. He has just enough containers to sort each type of ball he has into its own container. David wants to sort the balls using his sort method.

As an example, David has $n = 2$ containers and 2 different types of balls, both of which are numbered from 0 to $n - 1 = 1$. The distribution of ball types per container are described by an $n \times n$ matrix of integers, $M[container][type]$. For example, consider the following diagram for $M = [[1, 4], [2, 3]]$:



In a single operation, David can *swap* two balls located in different containers.

The diagram below depicts a single swap operation:



David wants to perform some number of swap operations such that:

- Each container contains only balls of the same type.
- No two balls of the same type are located in different containers.

You must perform q queries where each query is in the form of a matrix, M. For each query, print `Possible` on a new line if David can satisfy the conditions above for the given matrix. Otherwise, print `Impossible`.

Function Description

Complete the *organizingContainers* function in the editor below. It should return a string, either `Possible` or `Impossible`.

organizingContainers has the following parameter(s):

- *containter*: a two dimensional array of integers that represent the number of balls of each color in each container

Input Format

The first line contains an integer q, the number of queries.

Each of the next q sets of lines is as follows:

1. The first line contains an integer n, the number of containers (rows) and ball types (columns).

2. Each of the next n lines contains n space-separated integers describing row $M[i]$.

Constraints

- $1 \leq q \leq 10$
- $1 \leq n \leq 100$
- $0 \leq M[container][type] \leq 10^9$

Scoring

- For 33% of score, $1 \leq n \leq 10$.
- For 100% of score, $1 \leq n \leq 100$.

Output Format

For each query, print `Possible` on a new line if David can satisfy the conditions above for the given matrix. Otherwise, print `Impossible`.

Sample Input 0

```
2
2
1 1
1 1
2
0 2
1 1
```

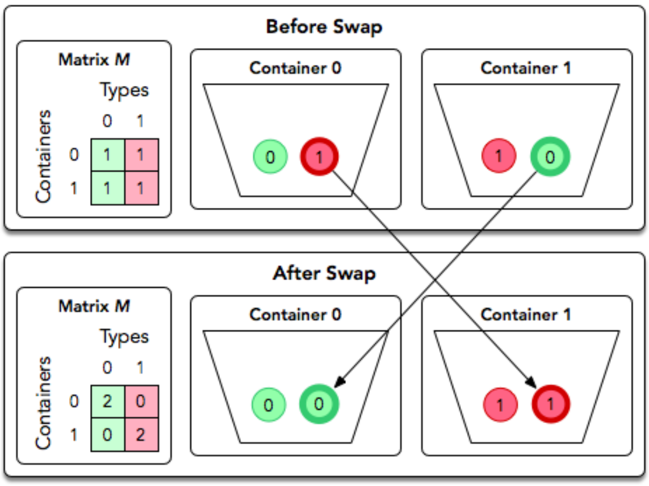
Sample Output 0

```
Possible
Impossible
```

Explanation 0

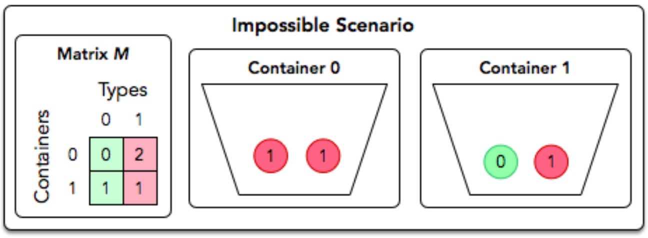
We perform the following $q = 2$ queries:

1. The diagram below depicts one possible way to satisfy David's requirements for the first query:



   Thus, we print `Possible` on a new line.

2. The diagram below depicts the matrix for the second query:



No matter how many times we swap balls of type t_0 and t_1 between the two containers, we'll never end up with one container only containing type t_0 and the other container only containing type t_1. Thus, we print `Impossible` on a new line.

Sample Input 1

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

Sample Output 1

```
Impossible
Possible
```

Save the Prisoner!

A jail has a number of prisoners and a number of treats to pass out to them. Their jailer decides the fairest way to divide the treats is to seat the prisoners around a circular table in sequentially numbered chairs. A chair number will be drawn from a hat. Beginning with the prisoner in that chair, one candy will be handed to each prisoner sequentially around the table until all have been distributed.

The jailer is playing a little joke, though. The last piece of candy looks like all the others, but it tastes awful! Determine the chair number occupied by the prisoner who will receive that candy.

For example, there are 4 prisoners and 6 pieces of candy. The prisoners arrange themselves in seats numbered 1 to 4. Let's suppose two is drawn from the hat. Prisoners receive candy at positions $2, 3, 4, 1, 2, 3$. The prisoner to be warned sits in chair number 3.

Function Description

Complete the saveThePrisoner function in the editor below. It should return an integer representing the chair number of the prisoner to warn.

saveThePrisoner has the following parameter(s):

- n an integer, the number of prisoners
- m an integer, the number of sweets
- s an integer, the chair number to begin passing out sweets from

Input Format

The first line contains an integer, t, denoting the number of test cases.
The next t lines each contain 3 space-separated integers:
- n: the number of prisoners
- m: the number of sweets
- s: the chair number to start passing out treats at

Constraints

- $1 \le t \le 100$
- $1 \le n \le 10^9$
- $1 \le m \le 10^9$
- $1 \le s \le n$

Output Format

For each test case, print the chair number of the prisoner who receives the awful treat on a new line.

Sample Input 0

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

Sample Output 0

```
2
3
```

Explanation 0

In first query, there are $n = 5$ prisoners and $m = 2$ sweets. Distribution starts at seat number $s = 1$. Prisoners in seats numbered 1 and 2 get sweets. Warn prisoner 2.
In the second query, distribution starts at seat 2 so prisoners in seats 2 and 3 get sweets. Warn prisoner 3.

Sample Input 1

```
2
7 19 2
3 7 3
```

Sample Output 1

```
6
3
```

Explanation 1

In the first test case, there are $n = 7$ prisoners, $m = 19$ sweets and they are passed out starting at chair $s = 2$. The candies go all around twice and there are 5 more candies passed to each prisoner from seat 2 to seat 6.

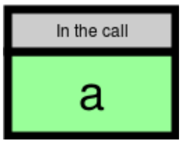
In the second test case, there are $n = 3$ prisoners, $m = 7$ candies and they are passed out starting at seat $s = 3$. They go around twice, and there is one more to pass out to the prisoner at seat 3.

Video Conference

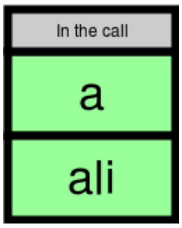
Bob is making a video conference software. Whenever a new person joins the conference, Bob displays the person's name in the interface.

But displaying full name is boring and takes a lots of space. So he decided to display the shortest prefix which doesn't match with any prefix of any person who have joined earlier.

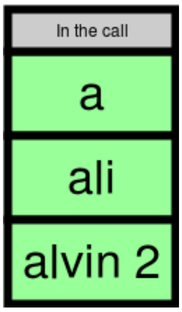
Let's suppose the first person to enter the conference is `alvin`.



Now suppose next person to join is `alice`. The shortest prefix of `alice` that doesn't match with any prefix of `alvin` is `ali`.



If the full name of the a new person matches completely with any person who have joined earlier, he will display the full name and add a suffix which indicates how many times the same name occurs in the list so far. For example if another person name `alvin` joins, the list will look like this:



You are given the list of the person who have joined the call in chronological order. Your task is to figure out how the final list will look like.

Input Format

The first line contains an integer n.

The subsequent n line contains a string s_i denoting the name of the i^{th} person to join the call.

Constraints

Video Conference

Output Format

Print n lines. The i^{th} line should contain the prefix of name of the i^{th} person which doesn't match with any other person who have joined earlier.

Sample Input 0

```
3
```

```
alvin
alice
alvin
```

Sample Output 0

```
a
ali
alvin 2
```

Sample Input 1

```
6
mary
stacy
sam
samuel
sam
miguel
```

Sample Output 1

```
m
s
sa
samu
sam 2
mi
```