Nim Game Strategy

Nim Game Description:

"Nim" is a two-player game played with sticks. The sticks are divided into piles. The player alternate turns. On each player's turn they may remove any number of sticks from one of the piles, up to the number of sticks remaining in that pile; but they can only take from a single pile on a given turn. The goal is to take the last stick. Whoever takes the last stick wins. So the goal is to make the other person clear out all but one pile, so you can take all the sticks in that last pile. Etc.

Winning Strategy / Kernel of the game:

We define the "kernel" of a game to be a set of game states including all winning states and such that if the state of the game is in the kernel, it is not possible to make a move which keeps it in the kernel, whereas if the state is not in the kernel, it IS possible to make a move which gets it into the kernel.

In other words, if I'm playing with a winning strategy based on my observation that a particular set is the kernel for the game, then when it's the opponent's move the game will be in a kernel state. Opponent's move will necessarily take it out of the kernel. My move will restore it to a kernel state because this is always possible from a non-kernel state, whereas opponent's move will always take it out because if I present my opponent with a kernel state, he cannot keep it in the kernel. And eventually, I will move it to a kernel state which is the winning state, i.e. I will win.

Example:

The rule is: in the kernel === all zeroes in parity computation (bitwise xor).

Suppose There are 3 piles which are 3, 7 ,9. Now if we represent them in binary we get :

```
0011  = 3
0111  = 7
1001  = 9
============
1101  = 13
```

Now If we do an bitwise Xor operation we will get 1101, which is not in the kernel i.e. not a winning state. Suppose here us tale 5 from the largest pile.

```
0011
0111
0100  (4)
====
0000
```

Thus we've moved into the kernel. This is the winning strategy of Nim Game.

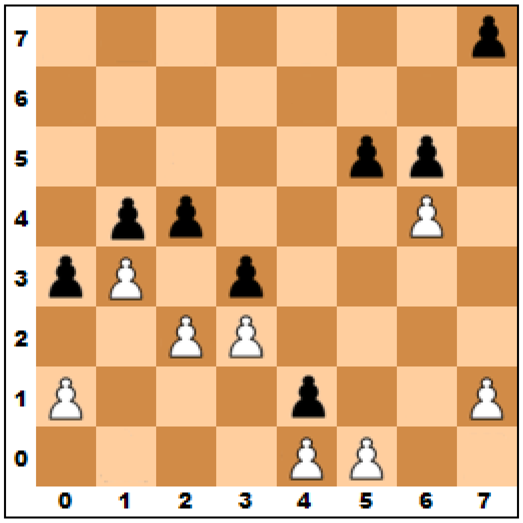
Now we will analyze some variations of Nim.

LightOJ 1186 - Incredible Chess

You are given an **n x n** chess board. Only pawn is used in the 'Incredible Chess' and they can move forward or backward. In each column there are two pawns, one white and one black. White pawns are placed in the lower part of the board and the black pawns are placed in the upper part of the board.

The game is played by two players. Initially a board configuration is given. One player uses white pieces while the other uses black. In each move, a player can move a pawn of his piece, which can go forward or backward any positive integer steps, but it cannot jump over any piece. White gives the first move.

The game ends when there is no move for a player and he will lose the game. Now you are given the initial configuration of the board. You have to write a program to determine who will be the winner.



Example of a Board

Input

Input starts with an integer **T (≤ 200)**, denoting the number of test cases.

Each case starts with an integer **n** $(3 \le n \le 100)$ denoting the dimension of the board. The next line will contain **n** integers, $W_0, W_1, \ldots, W_{n-1}$ giving the position of the white pieces. The next line will also contain **n** integers, $B_0, B_1, \ldots B_{n-1}$ giving the position of the black pieces. W_i means the row position of the white piece of **i**th column. And B_i means the row position of the black piece of **i**th column. You can assume that $(0 \le W_i < B_i < n)$for $(0 \le i < n)$ and at least one move is remaining.

Output

For each case, print the case number and **'white wins'** or **'black wins'** depending on the result.

Sample Input

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

Output for Sample Input

```
Case 1: black wins
Case 2: white wins
```

PROBLEM SETTER: JANE ALAM JAN

Analysis:

At First let us consider for a single column chess board. Now the game may end only if the there is no blank position between white piece and black piece . So in this case the kernel/ winning strategy of the game is all zeroes.

2	B
3	
4	
5	
6	W

So in the above example there are 3 (011) blank positions. So as I am the white player I will make 3 moves to make it zero (000).Thus I will always try to force my opponent to have the kernel state (the zero state).

Now let us consider the game having multiple columns so multiple blank position .So there bitwise Xor operations gives the state [kernel state]. If we have anything other than zero we are sure white wins vice versa.

Source Code

```
#include <stdio.h>
int arr[MAX];
int main(void)
{
    int cas,loop=0,n,sum,val;
    scanf("%d",&cas);
    while(cas--)
    {
        scanf("%d",&n);sum=0;
        for(int i=0;i<n;i++) scanf("%d",&arr[i]);
        for(int i=0;i<n;i++) scanf("%d",&val),sum^= Abs(val-arr[i] )-1 ;
        if(sum) printf("Case %d: white wins\n",++loop);
        else printf("Case %d: black wins\n",++loop);
    }
    return 0;
}
```

LightOJ 1192 - Left Right

Two players, Alice and Bob are playing a strange game in a **1 x n** board. The cells are numbered from **0** to **n-1**, where the left most cell is marked as cell **0**. Each cell can contain at most one piece.

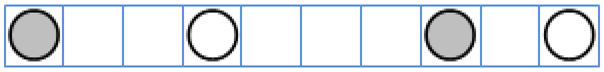


Fig 1: an example

There are two kinds of pieces, gray and white. Alice moves all gray pieces, and bob moves all white ones. The pieces alternate, that is, leftmost piece is gray, next is white, next to that is gray, then it's white again, and so on. There will always be equal number of black and gray pieces. Alice can only move pieces to the right. Bob can only move pieces to the left.

At each move, a player selects one piece and moves that piece, either to its left (Bob) or to its right (Alice), any number of cells (at least 1) but, it can neither jump over other pieces, nor it can move outside the board. The players alternate their turns.

For example, if Alice decides to move the left most gray piece, these two moves are available to her.

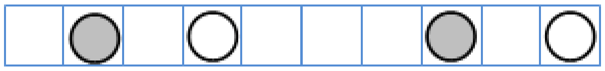


Fig 2: Moving the gray piece one cell to the right

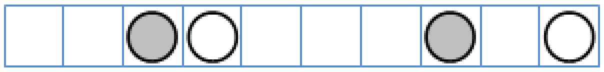


Fig 3: Moving the gray piece two cells to the right

Alice moves first. The game ends, when someone is unable to make any move, and loses the game. You can assume that, both of them play optimally (that is, if it is possible to apply a strategy that will ensure someone's win, he/she will always use that strategy).

Now you are given a configuration of a board, you have to find the winner.

Input

Input starts with an integer **T (≤ 200)**, denoting the number of test cases.

Each case starts with a line containing an integer **k (1 ≤ k ≤ 100)** denoting the number of gray pieces in the board. The next line contains **2k** distinct integers (in ascending order) denoting the position of the pieces. The first integer denotes a gray piece, the second integer denotes a white piece, the next integer denotes a gray piece and so on. All the integers will lie in the range **[0, 10^9]**. Assume that **n** is sufficiently large to contain all the pieces. And at least one move is remaining.

Output

For each case, print the case number and **'Alice'** or **'Bob'** depending on the winner of the game.

Sample Input

```
2
2
0 3 7 9
2
1 3 7 9
```

Output for Sample Input

```
Case 1: Alice
Case 2: Bob
```

PROBLEM SETTER: JANE ALAM JAN

Analysis:

Here if we make the observation that when either a white piece or a gray piece moves its position gets fixed. So now the problem is for each pair of white and gray piece we calculate the moves left. This can be considered the piles of a nim. So it has the same kernel state as Nim Game. So as Gray piece makes the move first it must always try to make its opponent fall into kernel state. We also know that it is always possible to reach from non-kernel state to kernel state.

So we Bitwise Xor the moves for each pair of white and gray piece. If it is zero first player loses or vice versa.

Source Code

```
#include <stdio.h>
int main(void)
{
    int cas,loop=0,k,val,sum,val2;
    scanf("%d",&cas);
    while(cas--)
    {
        sum=0;scanf("%d",&k);for(int i=0;i<k;i++) scanf("%d
%d",&val,&val2) , sum^=(val2-val)-1;
        if(sum) printf("Case %d: Alice\n",++loop);
        else printf("Case %d: Bob\n",++loop);
    }
    return 0;
}
```

LightOJ 1247 - Matrix Game

Given an **m x n** matrix, where **m** denotes the number of rows and **n** denotes the number of columns and in each cell a pile of stones is given. For example, let there be a **2 x 3** matrix, and the piles are

2 3 8
5 2 7

That means that in cell(1, 1) there is a pile with 2 stones, in cell(1, 2) there is a pile with 3 stones and so on.

Now Alice and Bob are playing a strange game in this matrix. Alice starts first and they alternate turns. In each turn a player selects a row, and can draw any number of stones from any number of cells in that row. But he/she must draw at least one stone. For example, if Alice chooses the 2nd row in the given matrix, she can pick 2 stones from cell(2, 1), 0 stones from cell (2, 2), 7 stones from cell(2, 3). Or she can pick 5 stones from cell(2, 1), 1 stone from cell(2, 2), 4 stones from cell(2, 3). There are many other ways but she must pick at least one stone from all piles. The player who can't take any stones loses.

Now if both play optimally who will win?

Input

Input starts with an integer **T (≤ 100)**, denoting the number of test cases.

Each case starts with a line containing two integers: **m** and **n** **(1 ≤ m, n ≤ 50)**. Each of the next **m** lines contains **n** space separated integers that form the matrix. All the integers will be between **0** and **10^9**(inclusive).

Output

For each case, print the case number and **'Alice'** if Alice wins, or **'Bob'** otherwise.

Sample Input	Output for Sample Input

2
2 3
2 3 8
5 2 7
2 3
1 2 3
3 2 1
	Case 1: Alice
	Case 2: Bob

PROBLEM SETTER: JANE ALAM JAN

Analysis:

Here if we observe that the entire row can be considered as a pile because internally it doesn't matter which one was picked which one was not. So we sum the value of an entire row and then do bitwise Xor for each row. Now like Nim it has the same kernel state i.e. if we can force the opponent to have zero state then it ensures our victory.

For the first player if bitwise Xor gives kernel state then he has no choice but to make it non-kernel state. And again the second player can make it kernel state for him. So the first player will lose. But if we have some value in bitwise Xor then it will be the other way around and first player will win. Thus the problem turns out to be very simple and straightforward Nim game theory.

Source Code

```
#include <stdio.h>

int main(void)
{
    int cas,loop=0,row,col,ans,val,sum;
    scanf("%d",&cas);
    while(cas--)
    {
        scanf("%d %d",&row,&col);
        ans=0;
        for(int i=0;i<row;i++,sum=0)
        {
            for(int j=0;j<col;j++)
            {
                scanf("%d",&val);
                sum+=val;
            }
            ans^=sum;
        }
        if(ans) printf("Case %d: Alice\n",++loop);
        else printf("Case %d: Bob\n",++loop);
    }
    return 0;
}
```

Misère Nim Game Strategy

SPOJ Problem Set (classical)

3969. M&M Game

Problem code: MMMGAME

Little John is playing very funny game with his younger brother. There is one big box filled with M&Ms of different colors. At first John has to eat several M&Ms of the same color. Then his opponent has to make a turn. And so on. Please note that each player has to eat at least one M&M during his turn. If John (or his brother) will eat the last M&M from the box he will be considered as a looser and he will have to buy a new candy box.

Both of players are using optimal game strategy. John starts first always. You will be given information about M&Ms and your task is to determine a winner of such a beautiful game.

Input

The first line of input will contain a single integer T – the number of test cases. Next T pairs of lines will describe tests in a following format. The first line of each test will contain an integer N –
the amount of
different M&M colors in a box. Next line will contain N integers Ai, separated by spaces – amount of M&Ms of i-th color.
Constrains:
1 <= T <= 474,
1 <= N <= 47,
1 <= Ai <= 4747

Output

Output T lines each of them containing information about game winner. Print “John” if John will win the game or “Brother” in other case.

Sample

```
Sample input:
2
3
3 5 1
1
1
Sample output:
John
Brother
```

Analysis:

This type of game is known as Misère Nim in which in which the player to take the last object loses.Thus slightly different technique is followed here.

Winning Strategy:

Here the first player to give the opponent one (0001) state wins(after bitwise Xor)   because this means that the other player needs to pick it up making him the last person thus ensuring his defeat.

So we can devise a strategy such that we will play normal nim until a situation arises where there is only One board of some tiles such that if we pick all tiles except one then the opponent must loose.
Exception: If there are some boards all of size 1 then if odd number of boards are availbale then first player loses and if there are even number of boards then first player wins.

Source Code

```
#include <stdio.h>
int main(void)
{
    int cas,k,sum,val,flag;
    scanf("%d",&cas);
    while(cas--)
    {
        sum=0;flag=0;
        scanf("%d",&k);
        for(int i=0;i<k;i++)
        {
            scanf("%d",&val);
            if(val>1) flag=1;
            sum^=val;
        }
        if(flag)
        {
            if(sum) puts("John");
            else puts("Brother");
        }
        else if(k%2) puts("Brother");
        else puts("John");
    }
    return 0;
}}
```

Sprague–Grundy theorem

Impartial:

A combinatorial game G is impartial if for any position, both players have the same set of possible moves.

Positions in Impartial Games:

For any position H in an impartial game, there is a set of possible positions which either player can move to $\{H_1, H_2, \dots\dots, H_n\}$ and we write $H=\{H_1, H_2, \dots\dots, H_n\}$ to indicate this. Note that the original game is then $G = \{G_1, G_2, \dots\dots, G_n\}$.

Minimal Excluded Principle:

Let $G \equiv \{ *a_1, *a_2, \dots\dots, *a_n \}$ and let b be the smallest nonnegative integer not in $\{ *a_1, *a_2, \dots\dots, *a_n \}$ Then $G \equiv *b$.

Proof:

Let H be an arbitrary combinatorial game. To prove the proposition, it suffices to show that H+G and H+ *b always has the same winning player. To see this, let us suppose that player i has a winning strategy in H + *b and let's take the role of this player in H+G. We shall now pretend that G is actually *b and follow our winning strategy in this pretend game. If we are the first player to play in the game G , then whatever move we made while pretending G was *b is available in G , and afterward our pretend game is becomes the true game , so following our strategy gets us a win. Similarly, if the other player plays first in G, but moves G to a position which was available in *b , then our pretend game becomes the actual game , and following our strategy gets us a win. The only complication is if the other player is first to play in G and chooses to move G to the position $*a_i$ where $a_i > b$. This move was not available in our pretend game. However, we can counter it on our next move by returning this Nim heap to *b , now , each player has lost a turn, but the pretend game has turned into the true one, so our strategy will get us a win.

Sprague-Grundy Analysis:

For every finite impartial game G there exist ($a \sum N$) so that $G \equiv *a$.

Proof:

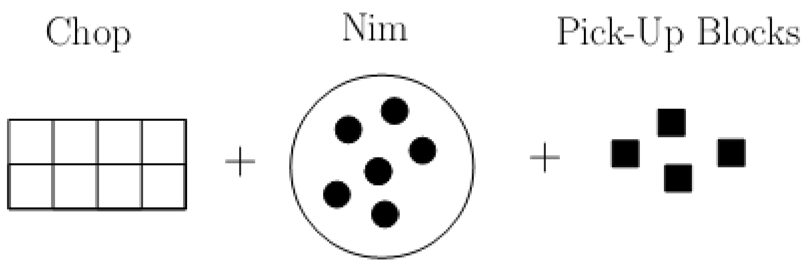
We proceed by induction on the depth of the game (the length of the longest possible game) . As a base case, if there are no legal moves , then $G \equiv *0$. For the inductive step, suppose that $G = \{ G_1, G_2, \dots\dots, G_n \}$. Then in each of the games $G_1, G_2, \dots\dots, G_n$ has smaller depth than G , so by induction each G_i is equivalent to $*a_i$ for some $a_i \sum N$. Thus $G =\{ G_1, G_2, \dots \dots , G_n \} \equiv \{ *a_1, *a_2, \dots \dots , *a_n \}$. It now follows from the previous lemma that $G\equiv *b$ where b is the smallest integer not in $\{ a_1, a_2, \dots \dots , a_n \}$.

Playing Sums of Combinatorial Games:

If $G_1, G_2, \dots \dots , G_k$ are impartial combinatorial games, and $b_1 , b_2 , \dots \dots , b_k$ are nonnegative integers so that $G_i \equiv *b_i$ for every $1\leq i \leq k$ then the game

$$G_1+ G_2+ \dots \dots +G_k \equiv *b_1 + *b_2 + \dots \dots + *b_k \equiv *(b_1 \text{ XOR } b_2 \text{ XOR} \dots \dots \text{ XOR } b_k)$$ where XOR means bitwise Xclusive OR.

Furthermore, we can use this information to find optimal strategies in the game $G_1+ G_2+ \dots \dots +G_k$. For instance, consider the game below



Suppose that game of Chop is equivalent to *2,

The Nim heap is equivalent to *6, and

The game of pick-Up Blocks is equivalent to *1 . Thus, this game is equivalent to $*2 + *6 + *1 \equiv *5$. It follows that an optimal move for the first player is to pick up 3 tokens from the Nim game , which will move this position to one equivalent to $*2 + *3 + *1 \equiv 0$.

LightOJ 1199 - Partitioning Game

Alice and Bob are playing a strange game. The rules of the game are:

1. Initially there are **n** piles.
2. A pile is formed by some cells.
3. Alice starts the game and they alternate turns.
4. In each tern a player can pick any pile and divide it into two unequal piles.
5. If a player cannot do so, he/she loses the game.

Now you are given the number of cells in each of the piles, you have to find the winner of the game if both of them play optimally.

Input

Input starts with an integer **T (≤ 1000)**, denoting the number of test cases.

Each case starts with a line containing an integer **n (1 ≤ n ≤ 100)**. The next line contains **n** integers, where the ith integer denotes the number of cells in the ith pile. You can assume that the number of cells in each pile is between **1** and **10000**.

Output

For each case, print the case number and **'Alice'** or **'Bob'** depending on the winner of the game.

Sample Input	Output for Sample Input

3
1
4
3
1 2 3
1
7
	Case 1: Bob
	Case 2: Alice
	Case 3: Bob

Explanation

In case 1, Alice has only 1 move, she divides the pile with 4 cells into two unequal piles, where one pile has 1 cell and the other pile has 3 cells. Now it's Bob's turn. Bob divides the pile with 3 cells into two piles, where one pile has 1 cell and another pile has 2 cells. So, now there are three piles having cells 1, 1, 2. And Alice loses, since she doesn't have any moves now.

PROBLEM SETTER: JANE ALAM JAN

Analysis:

In this problem each pile can be divided in many ways.Now For N , if we can find out the nimvalue aka nimber of each pair of these new 2 piles which makes N and xor them then we will get all possible *a. Now we iterate and find the minimum *b which is not among these { $*a_1$, $*a_2$, … … , $*a_n$ } . Note that *b has to be the minimum .

So what we are basically trying to do is :

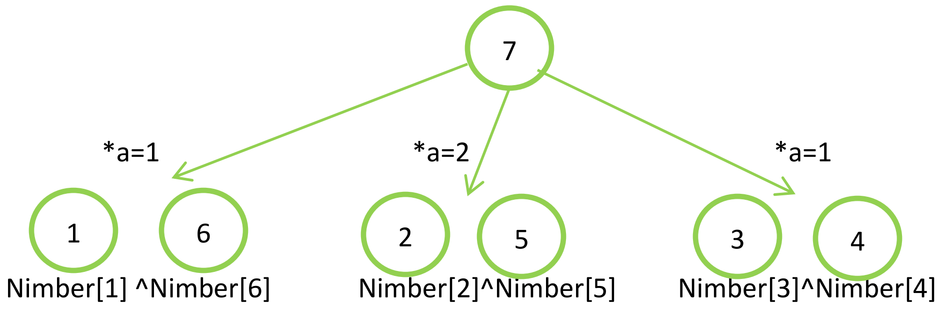
1. For N find all possible pair of piles such that pile i and pile N-i where i not equal to N-i that is i!=N-i .
2. Let us assume we know nimber for each of these piles i where i<N .
3. We do Xor operation on each of these pair i.e nimber[i]^nimber[N-i] and track the value *a
4. Now we find the minimum *b not in { $*a_1$, $*a_2$, … … , $*a_n$ }.
5. This *b is the nimber of N i.e. nimber[N] = b.
6. Now according to Sums of combinatorial Game for each N if we xor all their nim values we will get the answer.
7. If we find anything other than zero then first player wins else second player wins.

Example:

Suppose we have two piles 7 and 11.First we try to figure out the nimvalue of 7 . How we get this is given below :

found value of *a		
	1	2

Nimber[7]=0



Assume we know the nimber of the following piles. We want to find nimber of 7.

Nimber[1] = 0

Nimber[2] = 0

Nimber[3] = 1

Nimber[4] = 0

Nimber[5] = 2

Nimber[6] = 1

Now if there were another pile of value 11 we can find its nimvalue the same way. Here if the this new piles nimber value was 2 i.e. nimber [11] =2

Then we get nimber [7] =0 and nimber [11] =2 .

Nimber[7]^Nimber[11]=2.

Xor them both yields 2 which gives us the winning move. So the first player wins.

Source Code

```
#include <map>
#include <queue>
#include <stack>
#include <cmath>
#include <cctype>
#include <set>
#include <bitset>
#include <algorithm>
#include <list>
#include <vector>
#include <sstream>
#include <iostream>

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
#include <ctype.h>

using namespace std;

typedef long long ll;
typedef pair<int,int> paii;
typedef pair< ll, ll > pall;


#define PI (2.0*acos(0))
#define ERR 1e-5
#define mem(a,b) memset(a,b,sizeof a)
#define pb push_back
#define popb pop_back
#define all(x) (x).begin(),(x).end()
#define mp make_pair
#define SZ(x) (int)x.size()
#define oo (1<<25)
#define FOREACH(it,x) for(__typeof((x).begin()) it=(x.begin());
it!=(x).end(); ++it)
#define Contains(X,item)        ((X).find(item) != (X).end())
#define popc(i) (__builtin_popcount(i))
#define fs      first
#define sc      second
#define EQ(a,b)     (fabs(a-b)<ERR)
#define MAX 10050
```

```
template<class T> T Abs(T x) {return x > 0 ? x : -x;}
template<class T> inline T sqr(T x){return x*x;}
ll Pow(ll B,ll P){      ll R=1; while(P>0)      {if(P%2==1)
R=(R*B);P/=2;B=(B*B);}return R;}
int BigMod(ll B,ll P,ll M){     ll R=1; while(P>0)
{if(P%2==1){R=(R*B)%M;}P/=2;B=(B*B)%M;} return (int)R;} /// (B^P)%M

int nimber[MAX];
int col[MAX];

void generate(int MAX_value)
{
    int range;
    for(int i=1;i<MAX;i++)
    {
        range=( (i&1) ? i/2 : i/2 -1 );
        for(int j=1;j<=range;j++)
        {
            col[ nimber[j]^nimber[i-j] ]=i;
        }
        for(int p=0;p<MAX;p++) if(col[p]!=i) {nimber[i]=p;break;}
    }
}

int main(void)
{
    generate(10005);
    int cas,loop=0,ans,n,val;
    scanf("%d",&cas);
    while(cas--)
    {
        ans=0;
        scanf("%d",&n);
        while(n--) scanf("%d",&val) , ans^=nimber[val];
        if(ans) printf("Case %d: Alice\n",++loop);
        else printf("Case %d: Bob\n",++loop);
    }
    return 0;
}
```