| | | | |
|---|---|---|---|
| **Course Name:** | **Competitive Programming Laboratory (216U01L401)** | **Semester:** | **IV** |
| **Date of Performance:** | **_19__ / _04__ / __25____** | **DIV/ Batch No:** | **E-2** |
| **Student Name:** | **Shreyans Tatiya** | **Roll No:** | **16010123325** |

## Experiment No: 7

**Title:** **To use a geometry-based approach to solve competitive programming problems.**

### Aim and Objective of the Experiment:

1. **Understand** the concepts of geometry-based approach.
2. **Apply** the concepts to solve the problem.
3. **Implement** the solution to given problem statement.
4. **Create** test cases for testing the solution.
5. **Analyze** the result for efficiency of the solution.

### COs to be achieved:

CO4: Apply geometry, game theory, and string algorithms in competitive programming.

### Books/ Journals/ Websites referred:

1. https://leetcode.com/problems/nim-game/
2. https://leetcode.com/problems/chalkboard-xor-game/
3. Competitive Programmer's Handbook (CPH) - Antti Laaksonen

### Theory:

Geometry-based approaches are essential in solving a wide variety of computational problems involving shapes, coordinates, distances, angles, and spatial relationships. In competitive programming, geometric algorithms are applied to problems involving points, lines, polygons, and circles, often under time and memory constraints.

The core idea is to model the given problem geometrically and use mathematical and algorithmic techniques to derive an efficient solution. These problems are typically solved using principles from **Euclidean geometry**, **analytic geometry**, and **vector algebra**.

Efficient geometric computation often involves handling **floating-point precision errors** and optimizing algorithms to run in **O(n log n)** time for large input sizes.

The **Nim Game** and **Chalkboard XOR Game** are classic problems in **combinatorial game theory**, which deals with two-player games of perfect information and no chance elements. In such games, players take turns altering a game state according to fixed rules, and the objective is usually to be the last to make a valid move. These games are often analyzed by categorizing positions as either **winning** or **losing**, depending on whether the current player can force a win with optimal play. Both games highlight the use of **mathematical invariants** and logic to determine winning strategies, and although not geometric in nature, they are essential examples of how structured thinking and discrete mathematics are applied in competitive programming.

## Problem statement

**Problem 1**

You are playing the following Nim Game with your friend:

- Initially, there is a heap of stones on the table.
- You and your friend will alternate taking turns, and you go first.
- On each turn, the person whose turn it is will remove 1 to 3 stones from the heap.
- The one who removes the last stone is the winner.

Given n, the number of stones in the heap, return true if you can win the game assuming both you and your friend play optimally, otherwise return false.

**Problem 2**

You are given an array of integers nums represents the numbers written on a chalkboard.
Alice and Bob take turns erasing exactly one number from the chalkboard, with Alice starting first. If erasing a number causes the bitwise XOR of all the elements of the chalkboard to become 0, then that player loses. The bitwise XOR of one element is that element itself, and the bitwise XOR of no elements is 0.
Also, if any player starts their turn with the bitwise XOR of all the elements of the chalkboard equal to 0, then that player wins.
Return true if and only if Alice wins the game, assuming both players play optimally.

## Code :

**Code 1-**

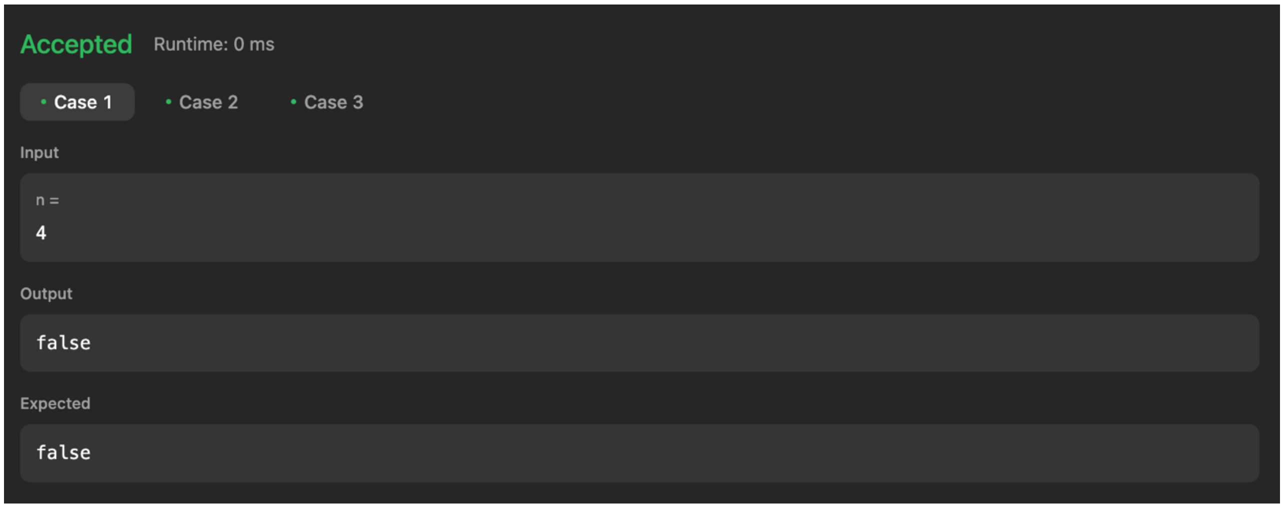
```java
class Solution {
    public boolean canWinNim(int n) {
        return n % 4 != 0;
    }
}
```

**Code 2-**

```java
class Solution {
    public boolean xorGame(int[] nums) {
        int xor = 0;
        for (int num : nums) {
            xor ^= num;
        }
        return xor == 0 || nums.length % 2 == 0;
    }
}
```

**Output:**

**Code 1-**



Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3

Input

```
n =
4
```

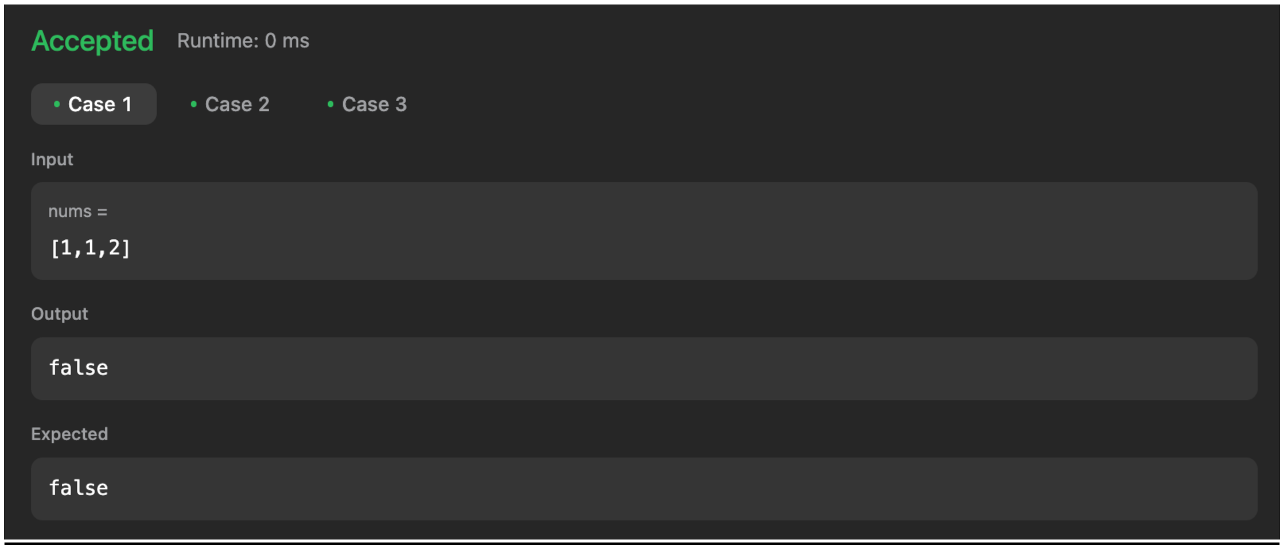
Output

```
false
```

Expected

```
false
```

**Code 2-**



Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3

Input

```
nums =
[1,1,2]
```

Output

```
false
```

Expected

```
false
```

**Post Lab Subjective/Objective type Questions:**

1. Explain Branch and Bound approach in 15-Puzzle Problem and 0/1 knapsack Problem?

The **Branch and Bound** approach is an optimization technique used to solve problems by exploring all possible solutions in a structured way while pruning paths that cannot yield better results. It uses a bound function to estimate the best possible outcome from a given state, helping to avoid unnecessary computations.

In the **15-Puzzle Problem**, each state represents a configuration of the puzzle. The algorithm branches by moving the blank tile in different directions. To guide the search, a bounding function such as the Manhattan Distance is used, which adds the actual cost (moves so far) and the estimated cost (tiles' distances to goal positions). States with lower costs are explored first, and less promising ones are pruned.

In the **0/1 Knapsack Problem**, the algorithm explores two choices for each item: include or exclude it. It calculates an upper bound on the total value using the fractional knapsack approach. If the bound is less than the current best solution or if the total weight exceeds the capacity, that branch is pruned. This helps find the optimal solution efficiently without checking every combination.

**Conclusion:**

Therefore, the above experiment highlights use of game-theory and geometry based approaches for the above two problem statements.