

# Problem 810: Chalkboard XOR Game

## Problem Information

Difficulty: **Hard**

Acceptance Rate: 0.00%

Paid Only: No

## Problem Description

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

.

Example 1:

Input:

nums = [1,1,2]

Output:

false

Explanation:

Alice has two choices: erase 1 or erase 2. If she erases 1, the nums array becomes [1, 2]. The bitwise XOR of all the elements of the chalkboard is  $1 \text{ XOR } 2 = 3$ . Now Bob can remove any element he wants, because Alice will be the one to erase the last element and she will lose. If Alice erases 2 first, now nums become [1, 1]. The bitwise XOR of all the elements of the chalkboard is  $1 \text{ XOR } 1 = 0$ . Alice will lose.

Example 2:

Input:

nums = [0,1]

Output:

true

Example 3:

Input:

```
nums = [1,2,3]
```

Output:

```
true
```

Constraints:

```
1 <= nums.length <= 1000
```

```
0 <= nums[i] < 2
```

```
16
```

## Code Snippets

**C++:**

```
class Solution {  
public:  
    bool xorGame(vector<int>& nums) {  
  
    }  
};
```

**Java:**

```
class Solution {  
    public boolean xorGame(int[] nums) {  
  
    }  
}
```

**Python3:**

```
class Solution:  
    def xorGame(self, nums: List[int]) -> bool:
```

**Python:**

```

class Solution(object):
    def xorGame(self, nums):
        """
        :type nums: List[int]
        :rtype: bool
        """

```

### JavaScript:

```

/**
 * @param {number[]} nums
 * @return {boolean}
 */
var xorGame = function(nums) {

};

```

### TypeScript:

```

function xorGame(nums: number[]): boolean {

};

```

### C#:

```

public class Solution {
    public bool XorGame(int[] nums) {

    }
}

```

### C:

```

bool xorGame(int* nums, int numsSize) {

}

```

### Go:

```

func xorGame(nums []int) bool {

}

```

## Kotlin:

```
class Solution {  
    fun xorGame(nums: IntArray): Boolean {  
  
    }  
}
```

## Swift:

```
class Solution {  
    func xorGame(_ nums: [Int]) -> Bool {  
  
    }  
}
```

## Rust:

```
impl Solution {  
    pub fn xor_game(nums: Vec<i32>) -> bool {  
  
    }  
}
```

## Ruby:

```
# @param {Integer[]} nums  
# @return {Boolean}  
def xor_game(nums)  
  
end
```

## PHP:

```
class Solution {  
  
    /**  
     * @param Integer[] $nums  
     * @return Boolean  
     */  
    function xorGame($nums) {  
  
    }  
}
```

```
}
```

### Dart:

```
class Solution {  
  bool xorGame(List<int> nums) {  
  
  }  
}
```

### Scala:

```
object Solution {  
  def xorGame(nums: Array[Int]): Boolean = {  
  
  }  
}
```

### Elixir:

```
defmodule Solution do  
  @spec xor_game(nums :: [integer]) :: boolean  
  def xor_game(nums) do  
  
  end  
end
```

### Erlang:

```
-spec xor_game(Nums :: [integer()]) -> boolean().  
xor_game(Nums) ->  
.
```

### Racket:

```
(define/contract (xor-game nums)  
  (-> (listof exact-integer?) boolean?)  
)
```

## Solutions

### C++ Solution:

```
/*
 * Problem: Chalkboard XOR Game
 * Difficulty: Hard
 * Tags: array, math
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(1) to O(n) depending on approach
 */

class Solution {
public:
    bool xorGame(vector<int>& nums) {

    }
};
```

### Java Solution:

```
/**
 * Problem: Chalkboard XOR Game
 * Difficulty: Hard
 * Tags: array, math
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(1) to O(n) depending on approach
 */

class Solution {
    public boolean xorGame(int[] nums) {

    }
}
```

### Python3 Solution:

```
"""
Problem: Chalkboard XOR Game
Difficulty: Hard
Tags: array, math
```

```

Approach: Use two pointers or sliding window technique
Time Complexity:  $O(n)$  or  $O(n \log n)$ 
Space Complexity:  $O(1)$  to  $O(n)$  depending on approach
"""

class Solution:
    def xorGame(self, nums: List[int]) -> bool:
        # TODO: Implement optimized solution
        pass

```

### Python Solution:

```

class Solution(object):
    def xorGame(self, nums):
        """
        :type nums: List[int]
        :rtype: bool
        """

```

### JavaScript Solution:

```

/**
 * Problem: Chalkboard XOR Game
 * Difficulty: Hard
 * Tags: array, math
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity:  $O(n)$  or  $O(n \log n)$ 
 * Space Complexity:  $O(1)$  to  $O(n)$  depending on approach
 */

/**
 * @param {number[]} nums
 * @return {boolean}
 */
var xorGame = function(nums) {

};

```

### TypeScript Solution:

```

/**
 * Problem: Chalkboard XOR Game
 * Difficulty: Hard
 * Tags: array, math
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(1) to O(n) depending on approach
 */

function xorGame(nums: number[]): boolean {

};

```

### C# Solution:

```

/*
 * Problem: Chalkboard XOR Game
 * Difficulty: Hard
 * Tags: array, math
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(1) to O(n) depending on approach
 */

public class Solution {
    public bool XorGame(int[] nums) {

    }
}

```

### C Solution:

```

/*
 * Problem: Chalkboard XOR Game
 * Difficulty: Hard
 * Tags: array, math
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(1) to O(n) depending on approach

```

```

*/

bool xorGame(int* nums, int numsSize) {

}

```

### Go Solution:

```

// Problem: Chalkboard XOR Game
// Difficulty: Hard
// Tags: array, math
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func xorGame(nums []int) bool {

}

```

### Kotlin Solution:

```

class Solution {
    fun xorGame(nums: IntArray): Boolean {

    }
}

```

### Swift Solution:

```

class Solution {
    func xorGame(_ nums: [Int]) -> Bool {

    }
}

```

### Rust Solution:

```

// Problem: Chalkboard XOR Game
// Difficulty: Hard
// Tags: array, math

```

```
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
    pub fn xor_game(nums: Vec<i32>) -> bool {

    }
}
```

### Ruby Solution:

```
# @param {Integer[]} nums
# @return {Boolean}
def xor_game(nums)

end
```

### PHP Solution:

```
class Solution {

    /**
     * @param Integer[] $nums
     * @return Boolean
     */
    function xorGame($nums) {

    }
}
```

### Dart Solution:

```
class Solution {
    bool xorGame(List<int> nums) {

    }
}
```

### Scala Solution:

```
object Solution {  
  def xorGame(nums: Array[Int]): Boolean = {  
  
  }  
}
```

### Elixir Solution:

```
defmodule Solution do  
  @spec xor_game(nums :: [integer]) :: boolean  
  def xor_game(nums) do  
  
  end  
end
```

### Erlang Solution:

```
-spec xor_game(Nums :: [integer()]) -> boolean().  
xor_game(Nums) ->  
.
```

### Racket Solution:

```
(define/contract (xor-game nums)  
  (-> (listof exact-integer?) boolean?)  
)
```