Problem 292: Nim Game

Problem Information

Difficulty: Easy
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

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

.

Example 1:

Input:

n = 4

Output:

false

Explanation:

These are the possible outcomes: 1. You remove 1 stone. Your friend removes 3 stones, including the last stone. Your friend wins. 2. You remove 2 stones. Your friend removes 2 stones, including the last stone. Your friend wins. 3. You remove 3 stones. Your friend removes the last stone. Your friend wins. In all outcomes, your friend wins.

Example 2:

Input:

n = 1

Output:

true

Example 3:

Input:

n = 2

Output:

true

Constraints:

1 <= n <= 2

31

- 1

Code Snippets

C++:

```
class Solution {
public:
bool canWinNim(int n) {

}
};
```

Java:

```
class Solution {
public boolean canWinNim(int n) {

}
}
```

Python3:

```
class Solution:
def canWinNim(self, n: int) -> bool:
```

Python:

```
class Solution(object):
def canWinNim(self, n):
"""
:type n: int
:rtype: bool
"""
```

JavaScript:

```
/**
* @param {number} n
* @return {boolean}
*/
var canWinNim = function(n) {

};
```

TypeScript:

```
function canWinNim(n: number): boolean {

};
```

C#:

```
public class Solution {
public bool CanWinNim(int n) {

}
}
```

C:

```
bool canWinNim(int n) {

}
```

Go:

```
func canWinNim(n int) bool {

}
```

Kotlin:

```
class Solution {
fun canWinNim(n: Int): Boolean {

}
}
```

Swift:

```
class Solution {
func canWinNim(_ n: Int) -> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn can_win_nim(n: i32) -> bool {

}
}
```

Ruby:

```
# @param {Integer} n
# @return {Boolean}
def can_win_nim(n)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @return Boolean
*/
function canWinNim($n) {

}
}
```

Dart:

```
class Solution {
bool canWinNim(int n) {

}
```

```
}
```

Scala:

```
object Solution {
def canWinNim(n: Int): Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec can_win_nim(n :: integer) :: boolean
def can_win_nim(n) do

end
end
```

Erlang:

```
-spec can_win_nim(N :: integer()) -> boolean().
can_win_nim(N) ->
.
```

Racket:

```
(define/contract (can-win-nim n)
(-> exact-integer? boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Nim Game
* Difficulty: Easy
* Tags: math, heap
*
```

```
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
bool canWinNim(int n) {

}
};
```

Java Solution:

```
/**
* Problem: Nim Game
* Difficulty: Easy
* Tags: math, heap
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public boolean canWinNim(int n) {

}
}
```

Python3 Solution:

```
"""
Problem: Nim Game
Difficulty: Easy
Tags: math, heap

Approach: Optimized algorithm based on problem constraints
Time Complexity: O(n) to O(n^2) depending on approach
Space Complexity: O(1) to O(n) depending on approach
"""
```

```
class Solution:
def canWinNim(self, n: int) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def canWinNim(self, n):
"""
:type n: int
:rtype: bool
"""
```

JavaScript Solution:

```
/**
* Problem: Nim Game
* Difficulty: Easy
* Tags: math, heap
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number} n
* @return {boolean}
*/
var canWinNim = function(n) {

};
```

TypeScript Solution:

```
/**
* Problem: Nim Game
* Difficulty: Easy
* Tags: math, heap
```

```
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

function canWinNim(n: number): boolean {

};
```

C# Solution:

```
/*
* Problem: Nim Game
* Difficulty: Easy
* Tags: math, heap
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public bool CanWinNim(int n) {

}
}
```

C Solution:

```
/*
* Problem: Nim Game
* Difficulty: Easy
* Tags: math, heap
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

bool canWinNim(int n) {
```

```
}
```

Go Solution:

```
// Problem: Nim Game
// Difficulty: Easy
// Tags: math, heap
//
// Approach: Optimized algorithm based on problem constraints
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

func canWinNim(n int) bool {

}
```

Kotlin Solution:

```
class Solution {
fun canWinNim(n: Int): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func canWinNim(_ n: Int) -> Bool {

}
}
```

Rust Solution:

```
// Problem: Nim Game
// Difficulty: Easy
// Tags: math, heap
//
// Approach: Optimized algorithm based on problem constraints
// Time Complexity: O(n) to O(n^2) depending on approach
```

```
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn can_win_nim(n: i32) -> bool {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @return {Boolean}
def can_win_nim(n)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @return Boolean
*/
function canWinNim($n) {

}
}
```

Dart Solution:

```
class Solution {
bool canWinNim(int n) {

}
}
```

Scala Solution:

```
object Solution {
def canWinNim(n: Int): Boolean = {
```

```
}
}
```

Elixir Solution:

```
defmodule Solution do
@spec can_win_nim(n :: integer) :: boolean
def can_win_nim(n) do

end
end
```

Erlang Solution:

```
-spec can_win_nim(N :: integer()) -> boolean().
can_win_nim(N) ->
.
```

Racket Solution:

```
(define/contract (can-win-nim n)
(-> exact-integer? boolean?)
)
```