Problem 2029: Stone Game IX

Problem Information

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

Problem Description

Alice and Bob continue their games with stones. There is a row of n stones, and each stone has an associated value. You are given an integer array

stones

, where

stones[i]

is the

value

of the

i

th

stone.

Alice and Bob take turns, with

Alice

starting first. On each turn, the player may remove any stone from

stones

. The player who removes a stone

loses

if the

sum

of the values of

all removed stones

is divisible by

3

. Bob will win automatically if there are no remaining stones (even if it is Alice's turn).

Assuming both players play

optimally

, return

true

if Alice wins and

false

if Bob wins

.

Example 1:

Input:

stones = [2,1]

Output:

true

Explanation:

The game will be played as follows: - Turn 1: Alice can remove either stone. - Turn 2: Bob removes the remaining stone. The sum of the removed stones is 1 + 2 = 3 and is divisible by 3. Therefore, Bob loses and Alice wins the game.

Example 2:

Input:

stones = [2]

Output:

false

Explanation:

Alice will remove the only stone, and the sum of the values on the removed stones is 2. Since all the stones are removed and the sum of values is not divisible by 3, Bob wins the game.

Example 3:

Input:

stones = [5,1,2,4,3]

Output:

false

Explanation:

Bob will always win. One possible way for Bob to win is shown below: - Turn 1: Alice can remove the second stone with value 1. Sum of removed stones = 1. - Turn 2: Bob removes the fifth stone with value 3. Sum of removed stones = 1 + 3 = 4. - Turn 3: Alices removes the fourth stone with value 4. Sum of removed stones = 1 + 3 + 4 = 8. - Turn 4: Bob removes the third stone with value 2. Sum of removed stones = 1 + 3 + 4 + 2 = 10. - Turn 5: Alice removes the first stone with value 5. Sum of removed stones = 1 + 3 + 4 + 2 + 5 = 15. Alice loses the game because the sum of the removed stones (15) is divisible by 3. Bob wins the game.

Constraints:

1 <= stones.length <= 10

5

1 <= stones[i] <= 10

4

Code Snippets

C++:

```
class Solution {
public:
bool stoneGameIX(vector<int>& stones) {

}
};
```

Java:

```
class Solution {
public boolean stoneGameIX(int[] stones) {

}
}
```

Python3:

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

Python:

```
class Solution(object):
def stoneGameIX(self, stones):
"""
:type stones: List[int]
:rtype: bool
"""
```

JavaScript:

```
/**
* @param {number[]} stones
* @return {boolean}
*/
var stoneGameIX = function(stones) {

};
```

TypeScript:

```
function stoneGameIX(stones: number[]): boolean {

};
```

C#:

```
public class Solution {
public bool StoneGameIX(int[] stones) {

}
}
```

C:

```
bool stoneGameIX(int* stones, int stonesSize) {

}
```

Go:

```
func stoneGameIX(stones []int) bool {

}
```

Kotlin:

```
class Solution {
fun stoneGameIX(stones: IntArray): Boolean {

}
}
```

Swift:

```
class Solution {
func stoneGameIX(_ stones: [Int]) -> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn stone_game_ix(stones: Vec<i32>) -> bool {

}
}
```

Ruby:

```
# @param {Integer[]} stones
# @return {Boolean}
def stone_game_ix(stones)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $stones
* @return Boolean
```

```
*/
function stoneGameIX($stones) {

}
}
```

Dart:

```
class Solution {
bool stoneGameIX(List<int> stones) {

}
}
```

Scala:

```
object Solution {
def stoneGameIX(stones: Array[Int]): Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec stone_game_ix(stones :: [integer]) :: boolean
def stone_game_ix(stones) do

end
end
```

Erlang:

```
-spec stone_game_ix(Stones :: [integer()]) -> boolean().
stone_game_ix(Stones) ->
.
```

Racket:

```
(define/contract (stone-game-ix stones)
(-> (listof exact-integer?) boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Stone Game IX
* Difficulty: Medium
* Tags: array, greedy, 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 stoneGameIX(vector<int>& stones) {

}
};
```

Java Solution:

```
/**
* Problem: Stone Game IX
* Difficulty: Medium
* Tags: array, greedy, 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 stoneGameIX(int[] stones) {

}
}
```

Python3 Solution:

```
"""
Problem: Stone Game IX
Difficulty: Medium
Tags: array, greedy, 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 stoneGameIX(self, stones: List[int]) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def stoneGameIX(self, stones):
"""
:type stones: List[int]
:rtype: bool
"""
```

JavaScript Solution:

```
/**
* Problem: Stone Game IX
* Difficulty: Medium
* Tags: array, greedy, 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[]} stones
* @return {boolean}
*/
var stoneGameIX = function(stones) {
```

```
};
```

TypeScript Solution:

```
/**
* Problem: Stone Game IX
* Difficulty: Medium
* Tags: array, greedy, 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 stoneGameIX(stones: number[]): boolean {

};
```

C# Solution:

```
/*
* Problem: Stone Game IX
* Difficulty: Medium
* Tags: array, greedy, 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 StoneGameIX(int[] stones) {

}
}
```

C Solution:

```
/*
* Problem: Stone Game IX
* Difficulty: Medium
```

```
* Tags: array, greedy, 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 stoneGameIX(int* stones, int stonesSize) {

}
```

Go Solution:

```
// Problem: Stone Game IX
// Difficulty: Medium
// Tags: array, greedy, 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 stoneGameIX(stones []int) bool {

}
```

Kotlin Solution:

```
class Solution {
fun stoneGameIX(stones: IntArray): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func stoneGameIX(_ stones: [Int]) -> Bool {

}
}
```

Rust Solution:

```
// Problem: Stone Game IX
// Difficulty: Medium
// Tags: array, greedy, 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 stone_game_ix(stones: Vec<i32>) -> bool {

}
}
```

Ruby Solution:

```
# @param {Integer[]} stones
# @return {Boolean}
def stone_game_ix(stones)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $stones
* @return Boolean
*/
function stoneGameIX($stones) {

}
}
```

Dart Solution:

```
class Solution {
bool stoneGameIX(List<int> stones) {
```

```
}
}
```

Scala Solution:

```
object Solution {
def stoneGameIX(stones: Array[Int]): Boolean = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec stone_game_ix(stones :: [integer]) :: boolean
def stone_game_ix(stones) do

end
end
```

Erlang Solution:

```
-spec stone_game_ix(Stones :: [integer()]) -> boolean().
stone_game_ix(Stones) ->
.
```

Racket Solution:

```
(define/contract (stone-game-ix stones)
(-> (listof exact-integer?) boolean?)
)
```