Problem 877: Stone Game

Problem Information

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

Problem Description

Alice and Bob play a game with piles of stones. There are an

even

number of piles arranged in a row, and each pile has a

positive

integer number of stones

piles[i]

.

The objective of the game is to end with the most stones. The

total

number of stones across all the piles is

odd

, so there are no ties.

Alice and Bob take turns, with

Alice starting first

. Each turn, a player takes the entire pile of stones either from the

beginning

or from the

end

of the row. This continues until there are no more piles left, at which point the person with the

most stones wins

.

Assuming Alice and Bob play optimally, return

true

if Alice wins the game, or

false

if Bob wins

.

Example 1:

Input:

piles = [5,3,4,5]

Output:

true

Explanation:

Alice starts first, and can only take the first 5 or the last 5. Say she takes the first 5, so that the row becomes [3, 4, 5]. If Bob takes 3, then the board is [4, 5], and Alice takes 5 to win with 10 points. If Bob takes the last 5, then the board is [3, 4], and Alice takes 4 to win with 9 points. This demonstrated that taking the first 5 was a winning move for Alice, so we return true.

Example 2:

Input:

piles = [3,7,2,3]

Output:

true

Constraints:

2 <= piles.length <= 500

piles.length

is

even

.

1 <= piles[i] <= 500

sum(piles[i])

is

odd

.

Code Snippets

C++:

```
class Solution {
public:
bool stoneGame(vector<int>& piles) {

}
};
```

Java:

```
class Solution {
public boolean stoneGame(int[] piles) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

```
}
```

TypeScript:

```
function stoneGame(piles: number[]): boolean {

};
```

C#:

```
public class Solution {
public bool StoneGame(int[] piles) {

}
}
```

C:

```
bool stoneGame(int* piles, int pilesSize) {

}
```

Go:

```
func stoneGame(piles []int) bool {

}
```

Kotlin:

```
class Solution {
fun stoneGame(piles: IntArray): Boolean {

}
}
```

Swift:

```
class Solution {
func stoneGame(_ piles: [Int]) -> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn stone_game(piles: Vec<i32>) -> bool {

}
}
```

Ruby:

```
# @param {Integer[]} piles
# @return {Boolean}
def stone_game(piles)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $piles
* @return Boolean
*/
function stoneGame($piles) {

}
}
```

Dart:

```
class Solution {
bool stoneGame(List<int> piles) {

}
}
```

Scala:

```
object Solution {
def stoneGame(piles: Array[Int]): Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec stone_game(piles :: [integer]) :: boolean
def stone_game(piles) do

end
end
```

Erlang:

```
-spec stone_game(Piles :: [integer()]) -> boolean().
stone_game(Piles) ->
.
```

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Stone Game
* Difficulty: Medium
* Tags: array, dp, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
bool stoneGame(vector<int>& piles) {

}
};
```

Java Solution:

```
/**
* Problem: Stone Game
* Difficulty: Medium
* Tags: array, dp, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public boolean stoneGame(int[] piles) {

}
}
```

Python3 Solution:

```
"""
Problem: Stone Game
Difficulty: Medium
Tags: array, dp, math

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

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

Python Solution:

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

```
"""
```

JavaScript Solution:

```
/**
* Problem: Stone Game
* Difficulty: Medium
* Tags: array, dp, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Stone Game
* Difficulty: Medium
* Tags: array, dp, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function stoneGame(piles: number[]): boolean {

};
```

C# Solution:

```
/*
* Problem: Stone Game
* Difficulty: Medium
* Tags: array, dp, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public bool StoneGame(int[] piles) {

}
}
```

C Solution:

```
/*
* Problem: Stone Game
* Difficulty: Medium
* Tags: array, dp, math
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

bool stoneGame(int* piles, int pilesSize) {

}
```

Go Solution:

```
// Problem: Stone Game
// Difficulty: Medium
// Tags: array, dp, math
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table
```

```
func stoneGame(piles []int) bool {

}
```

Kotlin Solution:

```
class Solution {
fun stoneGame(piles: IntArray): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func stoneGame(_ piles: [Int]) -> Bool {

}
}
```

Rust Solution:

```
// Problem: Stone Game
// Difficulty: Medium
// Tags: array, dp, math
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn stone_game(piles: Vec<i32>) -> bool {

}
}
```

Ruby Solution:

```
# @param {Integer[]} piles
# @return {Boolean}
def stone_game(piles)
```

```


end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $piles
* @return Boolean
*/
function stoneGame($piles) {

}
}
```

Dart Solution:

```
class Solution {
bool stoneGame(List<int> piles) {

}
}
```

Scala Solution:

```
object Solution {
def stoneGame(piles: Array[Int]): Boolean = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec stone_game(piles :: [integer]) :: boolean
def stone_game(piles) do

end
end
```

Erlang Solution:

```
-spec stone_game(Piles :: [integer()]) -> boolean().
stone_game(Piles) ->
  .
```

Racket Solution:

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