Problem 2660: Determine the Winner of a Bowling Game

Problem Information

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

Problem Description

You are given two

0-indexed

integer arrays

player1

and

player2

, representing the number of pins that player 1 and player 2 hit in a bowling game, respectively.

The bowling game consists of

n

turns, and the number of pins in each turn is exactly 10.

Assume a player hits

x

i

pins in the i

th

turn. The value of the i

th

turn for the player is:

2x

i

if the player hits 10 pins

in either (i - 1)

th

or (i - 2)

th

turn

.

Otherwise, it is

x

i

.

The

score

of the player is the sum of the values of their

n

turns.

Return

1 if the score of player 1 is more than the score of player 2,

2 if the score of player 2 is more than the score of player 1, and

0 in case of a draw.

Example 1:

Input:

player1 = [5,10,3,2], player2 = [6,5,7,3]

Output:

1

Explanation:

The score of player 1 is 5 + 10 + 2*3 + 2*2 = 25.

The score of player 2 is 6 + 5 + 7 + 3 = 21.

Example 2:

Input:

player1 = [3,5,7,6], player2 = [8,10,10,2]

Output:

2

Explanation:

The score of player 1 is 3 + 5 + 7 + 6 = 21.

The score of player 2 is 8 + 10 + 2*10 + 2*2 = 42.

Example 3:

Input:

player1 = [2,3], player2 = [4,1]

Output:

0

Explanation:

The score of player1 is 2 + 3 = 5.

The score of player2 is 4 + 1 = 5.

Example 4:

Input:

player1 = [1,1,1,10,10,10,10], player2 = [10,10,10,10,1,1,1]

Output:

2

Explanation:

The score of player1 is 1 + 1 + 1 + 10 + 2*10 + 2*10 + 2*10 = 73.

The score of player2 is 10 + 2*10 + 2*10 + 2*10 + 2*1 + 2*1 + 1 = 75.

Constraints:

n == player1.length == player2.length

1 <= n <= 1000

0 <= player1[i], player2[i] <= 10

Code Snippets

C++:

```
class Solution {
public:
int isWinner(vector<int>& player1, vector<int>& player2) {

}
};
```

Java:

```
class Solution {
public int isWinner(int[] player1, int[] player2) {

}
}
```

Python3:

```
class Solution:
def isWinner(self, player1: List[int], player2: List[int]) -> int:
```

Python:

```
class Solution(object):
def isWinner(self, player1, player2):
"""
:type player1: List[int]
```

```
:type player2: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} player1
* @param {number[]} player2
* @return {number}
*/
var isWinner = function(player1, player2) {

};
```

TypeScript:

```
function isWinner(player1: number[], player2: number[]): number {

};
```

C#:

```
public class Solution {
public int IsWinner(int[] player1, int[] player2) {

}
}
```

C:

```
int isWinner(int* player1, int player1Size, int* player2, int player2Size) {

}
```

Go:

```
func isWinner(player1 []int, player2 []int) int {

}
```

Kotlin:

```
class Solution {
fun isWinner(player1: IntArray, player2: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func isWinner(_ player1: [Int], _ player2: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn is_winner(player1: Vec<i32>, player2: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} player1
# @param {Integer[]} player2
# @return {Integer}
def is_winner(player1, player2)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $player1
* @param Integer[] $player2
* @return Integer
*/
function isWinner($player1, $player2) {

}
}
```

Dart:

```
class Solution {
int isWinner(List<int> player1, List<int> player2) {

}
}
```

Scala:

```
object Solution {
def isWinner(player1: Array[Int], player2: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec is_winner(player1 :: [integer], player2 :: [integer]) :: integer
def is_winner(player1, player2) do

end
end
```

Erlang:

```
-spec is_winner(Player1 :: [integer()], Player2 :: [integer()]) -> integer().
is_winner(Player1, Player2) ->
.
```

Racket:

```
(define/contract (is-winner player1 player2)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Determine the Winner of a Bowling Game
* Difficulty: Easy
* Tags: array
*
* 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:
int isWinner(vector<int>& player1, vector<int>& player2) {

}
};
```

Java Solution:

```
/**
* Problem: Determine the Winner of a Bowling Game
* Difficulty: Easy
* Tags: array
*
* 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 int isWinner(int[] player1, int[] player2) {

}
}
```

Python3 Solution:

```
"""
Problem: Determine the Winner of a Bowling Game
Difficulty: Easy
Tags: array
```

```
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 isWinner(self, player1: List[int], player2: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def isWinner(self, player1, player2):
"""
:type player1: List[int]
:type player2: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Determine the Winner of a Bowling Game
* Difficulty: Easy
* Tags: array
*
* 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[]} player1
* @param {number[]} player2
* @return {number}
*/
var isWinner = function(player1, player2) {

};
```

TypeScript Solution:

```
/**
* Problem: Determine the Winner of a Bowling Game
* Difficulty: Easy
* Tags: array
*
* 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 isWinner(player1: number[], player2: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Determine the Winner of a Bowling Game
* Difficulty: Easy
* Tags: array
*
* 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 int IsWinner(int[] player1, int[] player2) {

}
}
```

C Solution:

```
/*
* Problem: Determine the Winner of a Bowling Game
* Difficulty: Easy
* Tags: array
*
* 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
*/

int isWinner(int* player1, int player1Size, int* player2, int player2Size) {

}
```

Go Solution:

```
// Problem: Determine the Winner of a Bowling Game
// Difficulty: Easy
// Tags: array
//
// 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 isWinner(player1 []int, player2 []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun isWinner(player1: IntArray, player2: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func isWinner(_ player1: [Int], _ player2: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Determine the Winner of a Bowling Game
// Difficulty: Easy
// Tags: array
//
// 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 is_winner(player1: Vec<i32>, player2: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} player1
# @param {Integer[]} player2
# @return {Integer}
def is_winner(player1, player2)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $player1
* @param Integer[] $player2
* @return Integer
*/
function isWinner($player1, $player2) {

}
}
```

Dart Solution:

```
class Solution {
int isWinner(List<int> player1, List<int> player2) {
}
}
```

Scala Solution:

```
object Solution {
def isWinner(player1: Array[Int], player2: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec is_winner(player1 :: [integer], player2 :: [integer]) :: integer
def is_winner(player1, player2) do

end
end
```

Erlang Solution:

```
-spec is_winner(Player1 :: [integer()], Player2 :: [integer()]) -> integer().
is_winner(Player1, Player2) ->
.
```

Racket Solution:

```
(define/contract (is-winner player1 player2)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```