Problem 486: Predict the Winner

Problem Information

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

Problem Description

You are given an integer array

nums

. Two players are playing a game with this array: player 1 and player 2.

Player 1 and player 2 take turns, with player 1 starting first. Both players start the game with a score of

0

. At each turn, the player takes one of the numbers from either end of the array (i.e.,

nums[0]

or

nums[nums.length - 1]

) which reduces the size of the array by

1

. The player adds the chosen number to their score. The game ends when there are no more elements in the array.

Return

true

if Player 1 can win the game. If the scores of both players are equal, then player 1 is still the winner, and you should also return

true

. You may assume that both players are playing optimally.

Example 1:

Input:

nums = [1,5,2]

Output:

false

Explanation:

Initially, player 1 can choose between 1 and 2. If he chooses 2 (or 1), then player 2 can choose from 1 (or 2) and 5. If player 2 chooses 5, then player 1 will be left with 1 (or 2). So, final score of player 1 is 1 + 2 = 3, and player 2 is 5. Hence, player 1 will never be the winner and you need to return false.

Example 2:

Input:

nums = [1,5,233,7]

Output:

true

Explanation:

Player 1 first chooses 1. Then player 2 has to choose between 5 and 7. No matter which number player 2 choose, player 1 can choose 233. Finally, player 1 has more score (234) than player 2 (12), so you need to return True representing player1 can win.

Constraints:

1 <= nums.length <= 20

0 <= nums[i] <= 10

7

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

```
class Solution(object):
def predictTheWinner(self, nums):
```

```
"""
:type nums: List[int]
:rtype: bool
"""
```

JavaScript:

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

};
```

TypeScript:

```
function predictTheWinner(nums: number[]): boolean {

};
```

C#:

```
public class Solution {
public bool PredictTheWinner(int[] nums) {

}
}
```

C:

```
bool predictTheWinner(int* nums, int numsSize) {

}
```

Go:

```
func predictTheWinner(nums []int) bool {

}
```

Kotlin:

```
class Solution {
fun predictTheWinner(nums: IntArray): Boolean {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (predict-the-winner nums)
(-> (listof exact-integer?) boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Predict the Winner
* 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 predictTheWinner(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Predict the Winner
* 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 predictTheWinner(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Predict the Winner
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 predictTheWinner(self, nums: List[int]) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Predict the Winner
* 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[]} nums
* @return {boolean}
*/
var predictTheWinner = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Predict the Winner
* 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 predictTheWinner(nums: number[]): boolean {

};
```

C# Solution:

```
/*
* Problem: Predict the Winner
* 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 PredictTheWinner(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Predict the Winner
* 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 predictTheWinner(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Predict the Winner
// 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 predictTheWinner(nums []int) bool {

}
```

Kotlin Solution:

```
class Solution {
fun predictTheWinner(nums: IntArray): Boolean {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Predict the Winner
// 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 predict_the_winner(nums: Vec<i32>) -> bool {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (predict-the-winner nums)
(-> (listof exact-integer?) boolean?)
)
```