Problem 55: Jump Game

Problem Information

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

Problem Description

You are given an integer array

nums

. You are initially positioned at the array's

first index

, and each element in the array represents your maximum jump length at that position.

Return

true

if you can reach the last index, or

false

otherwise

.

Example 1:

Input:

nums = [2,3,1,1,4]

Output:

true

Explanation:

Jump 1 step from index 0 to 1, then 3 steps to the last index.

Example 2:

Input:

nums = [3,2,1,0,4]

Output:

false

Explanation:

You will always arrive at index 3 no matter what. Its maximum jump length is 0, which makes it impossible to reach the last index.

Constraints:

1 <= nums.length <= 10

4

0 <= nums[i] <= 10

5

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

```
};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

```
}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (can-jump nums)
(-> (listof exact-integer?) boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Jump Game
* Difficulty: Medium
* Tags: array, dp, greedy
*
* 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 canJump(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Jump Game
* Difficulty: Medium
* Tags: array, dp, greedy
*
* 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 canJump(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Jump Game
Difficulty: Medium
Tags: array, dp, greedy

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 canJump(self, nums: List[int]) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Jump Game
* Difficulty: Medium
* Tags: array, dp, greedy
*
* 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 canJump = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Jump Game
* Difficulty: Medium
* Tags: array, dp, greedy
*
* 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 canJump(nums: number[]): boolean {

};
```

C# Solution:

```
/*
* Problem: Jump Game
* Difficulty: Medium
* Tags: array, dp, greedy
*
```

```
* 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 CanJump(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Jump Game
* Difficulty: Medium
* Tags: array, dp, greedy
*
* 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 canJump(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Jump Game
// Difficulty: Medium
// Tags: array, dp, greedy
//
// 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 canJump(nums []int) bool {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Jump Game
// Difficulty: Medium
// Tags: array, dp, greedy
//
// 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 can_jump(nums: Vec<i32>) -> bool {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (can-jump nums)
(-> (listof exact-integer?) boolean?)
)
```