

# Problem 45: Jump Game II

## Problem Information

**Difficulty:** Medium

**Acceptance Rate:** 0.00%

**Paid Only:** No

## Problem Description

You are given a

0-indexed

array of integers

nums

of length

n

. You are initially positioned at index 0.

Each element

nums[i]

represents the maximum length of a forward jump from index

i

. In other words, if you are at index

i

, you can jump to any index

$(i + j)$

where:

$0 \leq j \leq \text{nums}[i]$

and

$i + j < n$

Return

the minimum number of jumps to reach index

$n - 1$

. The test cases are generated such that you can reach index

$n - 1$

.

Example 1:

Input:

$\text{nums} = [2,3,1,1,4]$

Output:

2

Explanation:

The minimum number of jumps to reach the last index is 2. Jump 1 step from index 0 to 1, then 3 steps to the last index.

Example 2:

Input:

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

Output:

2

Constraints:

$1 \leq \text{nums.length} \leq 10$

4

$0 \leq \text{nums}[i] \leq 1000$

It's guaranteed that you can reach

nums[n - 1]

## Code Snippets

C++:

```
class Solution {
public:
    int jump(vector<int>& nums) {
        }
};
```

Java:

```
class Solution {
public int jump(int[] nums) {
```

```
}
```

```
}
```

### Python3:

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

### Python:

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

### JavaScript:

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

### TypeScript:

```
function jump(nums: number[]): number {  
  
};
```

### C#:

```
public class Solution {  
    public int Jump(int[] nums) {  
  
    }  
}
```

**C:**

```
int jump(int* nums, int numsSize) {  
  
}
```

**Go:**

```
func jump(nums []int) int {  
  
}
```

**Kotlin:**

```
class Solution {  
    fun jump(nums: IntArray): Int {  
  
    }  
}
```

**Swift:**

```
class Solution {  
    func jump(_ nums: [Int]) -> Int {  
  
    }  
}
```

**Rust:**

```
impl Solution {  
    pub fn jump(nums: Vec<i32>) -> i32 {  
  
    }  
}
```

**Ruby:**

```
# @param {Integer[]} nums  
# @return {Integer}  
def jump(nums)  
  
end
```

**PHP:**

```
class Solution {  
  
    /**  
     * @param Integer[] $nums  
     * @return Integer  
     */  
    function jump($nums) {  
  
    }  
}
```

**Dart:**

```
class Solution {  
    int jump(List<int> nums) {  
  
    }  
}
```

**Scala:**

```
object Solution {  
    def jump(nums: Array[Int]): Int = {  
  
    }  
}
```

**Elixir:**

```
defmodule Solution do  
  @spec jump(list :: [integer]) :: integer  
  def jump(nums) do  
  
  end  
end
```

**Erlang:**

```
-spec jump(list :: [integer()]) -> integer().  
jump(List) ->  
.
```

## Racket:

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

## Solutions

### C++ Solution:

```
/*
 * Problem: Jump Game II
 * 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:
    int jump(vector<int>& nums) {
}
```

### Java Solution:

```
/**
 * Problem: Jump Game II
 * 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 int jump(int[] nums) {
```

```
}
```

```
}
```

### Python3 Solution:

```
"""
Problem: Jump Game II
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 jump(self, nums: List[int]) -> int:
        # TODO: Implement optimized solution
        pass
```

### Python Solution:

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

### JavaScript Solution:

```
/**
 * Problem: Jump Game II
 * 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 {number}
 */
var jump = function(nums) {
};


```

### TypeScript Solution:

```

/**
 * Problem: Jump Game II
 * 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 jump(nums: number[]): number {

};


```

### C# Solution:

```

/*
 * Problem: Jump Game II
 * 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 int Jump(int[] nums) {
    }
}


```

```
}
```

### C Solution:

```
/*
 * Problem: Jump Game II
 * 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
 */

int jump(int* nums, int numsSize) {

}
```

### Go Solution:

```
// Problem: Jump Game II
// 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 jump(nums []int) int {

}
```

### Kotlin Solution:

```
class Solution {
    fun jump(nums: IntArray): Int {
        }

    }
}
```

### Swift Solution:

```
class Solution {  
func jump(_ nums: [Int]) -> Int {  
}  
}  
}
```

### Rust Solution:

```
// Problem: Jump Game II  
// 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 jump(nums: Vec<i32>) -> i32 {  
}  
}  
}
```

### Ruby Solution:

```
# @param {Integer[]} nums  
# @return {Integer}  
def jump(nums)  
  
end
```

### PHP Solution:

```
class Solution {  
  
/**  
 * @param Integer[] $nums  
 * @return Integer  
 */  
function jump($nums) {  
  
}  
}
```

### Dart Solution:

```
class Solution {  
    int jump(List<int> nums) {  
  
    }  
}
```

### Scala Solution:

```
object Solution {  
    def jump(nums: Array[Int]): Int = {  
  
    }  
}
```

### Elixir Solution:

```
defmodule Solution do  
    @spec jump([integer]) :: integer  
    def jump(nums) do  
  
    end  
end
```

### Erlang Solution:

```
-spec jump([integer()]) -> integer().  
jump([_]) ->  
.
```

### Racket Solution:

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