Problem 287: Find the Duplicate Number

Problem Information

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

Problem Description

Given an array of integers

nums

containing

n + 1

integers where each integer is in the range

[1, n]

inclusive.

There is only

one repeated number

in

nums

, return

this repeated number

.

You must solve the problem

without

modifying the array

nums

and using only constant extra space.

Example 1:

Input:

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

Output:

2

Example 2:

Input:

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

Output:

3

Example 3:

Input:

nums = [3,3,3,3,3]

Output:

3

Constraints:

1 <= n <= 10

5

nums.length == n + 1

1 <= nums[i] <= n

All the integers in

nums

appear only

once

except for

precisely one integer

which appears

two or more

times.

Follow up:

How can we prove that at least one duplicate number must exist in

nums

?

Can you solve the problem in linear runtime complexity?

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {number[]} nums
* @return {number}
*/
```

```
var findDuplicate = function(nums) {

};
```

TypeScript:

```
function findDuplicate(nums: number[]): number {

};
```

C#:

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

}
}
```

C:

```
int findDuplicate(int* nums, int numsSize) {

}
```

Go:

```
func findDuplicate(nums []int) int {

}
```

Kotlin:

```
class Solution {
fun findDuplicate(nums: IntArray): Int {

}
}
```

Swift:

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

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
int findDuplicate(List<int> nums) {

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Find the Duplicate Number
* Difficulty: Medium
* Tags: array, search
*
* 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 findDuplicate(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Find the Duplicate Number
* Difficulty: Medium
* Tags: array, search
*
* 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 findDuplicate(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Find the Duplicate Number
Difficulty: Medium
Tags: array, search

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Find the Duplicate Number
* Difficulty: Medium
* Tags: array, search
*
* 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[]} nums
* @return {number}
*/
var findDuplicate = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Find the Duplicate Number
* Difficulty: Medium
* Tags: array, search
*
* 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 findDuplicate(nums: number[]): number {
};
```

C# Solution:

```
/*
* Problem: Find the Duplicate Number
* Difficulty: Medium
* Tags: array, search
*
* 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 FindDuplicate(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Find the Duplicate Number
* Difficulty: Medium
* Tags: array, search
*
* 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 findDuplicate(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Find the Duplicate Number
// Difficulty: Medium
// Tags: array, search
//
// 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 findDuplicate(nums []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun findDuplicate(nums: IntArray): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Find the Duplicate Number
// Difficulty: Medium
// Tags: array, search
//
// 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 find_duplicate(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
int findDuplicate(List<int> nums) {

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

```
end
end
```

Erlang Solution:

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

Racket Solution:

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