Problem 41: First Missing Positive

Problem Information

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

Problem Description

Given an unsorted integer array

nums

. Return the

smallest positive integer

that is

not present

in

nums

.

You must implement an algorithm that runs in

O(n)

time and uses

O(1)

auxiliary space.

Example 1:

Input:

nums = [1,2,0]

Output:

3

Explanation:

The numbers in the range [1,2] are all in the array.

Example 2:

Input:

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

Output:

2

Explanation:

1 is in the array but 2 is missing.

Example 3:

Input:

nums = [7,8,9,11,12]

Output:

1

Explanation:

The smallest positive integer 1 is missing.

Constraints:

1 <= nums.length <= 10

5

-2

31

<= nums[i] <= 2

31

- 1

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (first-missing-positive nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: First Missing Positive
* Difficulty: Hard
* Tags: array, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

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

}
};
```

Java Solution:

```
/**
* Problem: First Missing Positive
* Difficulty: Hard
* Tags: array, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public int firstMissingPositive(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: First Missing Positive
Difficulty: Hard
Tags: array, hash, sort

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class Solution:
def firstMissingPositive(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: First Missing Positive
* Difficulty: Hard
* Tags: array, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
```

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

};
```

TypeScript Solution:

```
/**
* Problem: First Missing Positive
* Difficulty: Hard
* Tags: array, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function firstMissingPositive(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: First Missing Positive
* Difficulty: Hard
* Tags: array, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public int FirstMissingPositive(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: First Missing Positive
* Difficulty: Hard
* Tags: array, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

int firstMissingPositive(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: First Missing Positive
// Difficulty: Hard
// Tags: array, hash, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func firstMissingPositive(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

Rust Solution:

```
// Problem: First Missing Positive
// Difficulty: Hard
// Tags: array, hash, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (first-missing-positive nums)
(-> (listof exact-integer?) exact-integer?)
)
```