Problem 2996: Smallest Missing Integer Greater Than Sequential Prefix Sum

Problem Information

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

Problem Description

You are given a

0-indexed

array of integers

nums

.

A prefix

nums[0..i]

is

sequential

if, for all

1 <= j <= i

,

nums[j] = nums[j - 1] + 1

. In particular, the prefix consisting only of

nums[0]

is

sequential

.

Return

the

smallest

integer

x

missing from

nums

such that

x

is greater than or equal to the sum of the

longest

sequential prefix.

Example 1:

Input:

nums = [1,2,3,2,5]

Output:

6

Explanation:

The longest sequential prefix of nums is [1,2,3] with a sum of 6. 6 is not in the array, therefore 6 is the smallest missing integer greater than or equal to the sum of the longest sequential prefix.

Example 2:

Input:

nums = [3,4,5,1,12,14,13]

Output:

15

Explanation:

The longest sequential prefix of nums is [3,4,5] with a sum of 12. 12, 13, and 14 belong to the array while 15 does not. Therefore 15 is the smallest missing integer greater than or equal to the sum of the longest sequential prefix.

Constraints:

1 <= nums.length <= 50

1 <= nums[i] <= 50

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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



}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Smallest Missing Integer Greater Than Sequential Prefix Sum
* Difficulty: Easy
* 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 missingInteger(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Smallest Missing Integer Greater Than Sequential Prefix Sum
* Difficulty: Easy
* 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 missingInteger(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Smallest Missing Integer Greater Than Sequential Prefix Sum
Difficulty: Easy
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 missingInteger(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Smallest Missing Integer Greater Than Sequential Prefix Sum
* Difficulty: Easy
* 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 missingInteger = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Smallest Missing Integer Greater Than Sequential Prefix Sum
* Difficulty: Easy
* 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 missingInteger(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Smallest Missing Integer Greater Than Sequential Prefix Sum
* Difficulty: Easy
* 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 MissingInteger(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Smallest Missing Integer Greater Than Sequential Prefix Sum
* Difficulty: Easy
* 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 missingInteger(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Smallest Missing Integer Greater Than Sequential Prefix Sum
// Difficulty: Easy
// 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 missingInteger(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Smallest Missing Integer Greater Than Sequential Prefix Sum
// Difficulty: Easy
// 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 missing_integer(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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