Problem 945: Minimum Increment to Make Array Unique

Problem Information

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

Problem Description

You are given an integer array

nums

. In one move, you can pick an index

i

where

0 <= i < nums.length

and increment

nums[i]

by

1

.

Return

the minimum number of moves to make every value in

nums

unique

.

The test cases are generated so that the answer fits in a 32-bit integer.

Example 1:

Input:

nums = [1,2,2]

Output:

1

Explanation:

After 1 move, the array could be [1, 2, 3].

Example 2:

Input:

nums = [3,2,1,2,1,7]

Output:

6

Explanation:

After 6 moves, the array could be [3, 4, 1, 2, 5, 7]. It can be shown that it is impossible for the array to have all unique values with 5 or less moves.

Constraints:

1 <= nums.length <= 10

5

0 <= nums[i] <= 10

5

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
```

```
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (min-increment-for-unique nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Increment to Make Array Unique
* Difficulty: Medium
* Tags: array, greedy, sort
*
```

```
* 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 minIncrementForUnique(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Increment to Make Array Unique
* Difficulty: Medium
* Tags: array, greedy, sort
*
* 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 minIncrementForUnique(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Increment to Make Array Unique
Difficulty: Medium
Tags: array, greedy, sort

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Increment to Make Array Unique
* Difficulty: Medium
* Tags: array, greedy, sort
*
* 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 minIncrementForUnique = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Increment to Make Array Unique
* Difficulty: Medium
* Tags: array, greedy, sort
```

```
*
* 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 minIncrementForUnique(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Increment to Make Array Unique
* Difficulty: Medium
* Tags: array, greedy, sort
*
* 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 MinIncrementForUnique(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Increment to Make Array Unique
* Difficulty: Medium
* Tags: array, greedy, sort
*
* 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 minIncrementForUnique(int* nums, int numsSize) {
```

```
}
```

Go Solution:

```
// Problem: Minimum Increment to Make Array Unique
// Difficulty: Medium
// Tags: array, greedy, sort
//
// 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 minIncrementForUnique(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Increment to Make Array Unique
// Difficulty: Medium
// Tags: array, greedy, sort
//
// 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 min_increment_for_unique(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

```
}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (min-increment-for-unique nums)
(-> (listof exact-integer?) exact-integer?)
)
```