Problem 1827: Minimum Operations to Make the Array Increasing

Problem Information

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

Problem Description

You are given an integer array

nums

(

0-indexed

). In one operation, you can choose an element of the array and increment it by

1

.

For example, if

nums = [1,2,3]

, you can choose to increment

nums[1]

to make

nums = [1,

3

,3]

.

Return

the

minimum

number of operations needed to make

nums

strictly

increasing

.

An array

nums

is

strictly increasing

if

nums[i] < nums[i+1]

for all

0 <= i < nums.length - 1

. An array of length

1

is trivially strictly increasing.

Example 1:

Input:

nums = [1,1,1]

Output:

3

Explanation:

You can do the following operations: 1) Increment nums[2], so nums becomes [1,1,

2

]. 2) Increment nums[1], so nums becomes [1,

2

,2]. 3) Increment nums[2], so nums becomes [1,2,

3

].

Example 2:

Input:

nums = [1,5,2,4,1]

Output:

14

Example 3:

Input:

nums = [8]

Output:

0

Constraints:

1 <= nums.length <= 5000

1 <= nums[i] <= 10

4

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```


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

```
}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Scala:

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

}
}
```

Racket:

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

)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Operations to Make the Array Increasing
* Difficulty: Easy
* Tags: array, greedy
*
* 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 minOperations(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Operations to Make the Array Increasing
* Difficulty: Easy
* Tags: array, greedy
*
* 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 minOperations(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Operations to Make the Array Increasing
Difficulty: Easy
Tags: array, greedy

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Operations to Make the Array Increasing
* Difficulty: Easy
* Tags: array, greedy
*
* 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 minOperations = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Operations to Make the Array Increasing
* Difficulty: Easy
* Tags: array, greedy
*
* 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 minOperations(nums: number[]): number {
```

```
};
```

C# Solution:

```
/*
* Problem: Minimum Operations to Make the Array Increasing
* Difficulty: Easy
* Tags: array, greedy
*
* 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 MinOperations(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Operations to Make the Array Increasing
* Difficulty: Easy
* Tags: array, greedy
*
* 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 minOperations(int* nums, int numsSize){

}
```

Go Solution:

```
// Problem: Minimum Operations to Make the Array Increasing
// Difficulty: Easy
// Tags: array, greedy
//
// 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 minOperations(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Operations to Make the Array Increasing
// Difficulty: Easy
// Tags: array, greedy
//
// 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_operations(nums: Vec<i32>) -> i32 {

}
```

```
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Scala Solution:

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

}
}
```

Racket Solution:

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

)
```