Problem 3717: Minimum Operations to Make the Array Beautiful

Problem Information

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

Problem Description

You are given an integer array

nums

.

An array is called

beautiful

if for every index

i > 0

, the value at

nums[i]

is

divisible

by

nums[i - 1]

.

In one operation, you may

increment

any element

nums[i]

(with

i > 0

) by

1

.

Return the

minimum number of operations

required to make the array beautiful.

Example 1:

Input:

nums = [3,7,9]

Output:

2

Explanation:

Applying the operation twice on

nums[1]

makes the array beautiful:

[3,9,9]

Example 2:

Input:

nums = [1,1,1]

Output:

0

Explanation:

The given array is already beautiful.

Example 3:

Input:

nums = [4]

Output:

0

Explanation:

The array has only one element, so it's already beautiful.

Constraints:

1 <= nums.length <= 100

1 <= nums[i] <= 50

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) {

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Minimum Operations to Make the Array Beautiful
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/
```

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

}
};
```

Java Solution:

```
/**
* Problem: Minimum Operations to Make the Array Beautiful
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Operations to Make the Array Beautiful
Difficulty: Medium
Tags: array, dp

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

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 Beautiful
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Operations to Make the Array Beautiful
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

```
};
```

C# Solution:

```
/*
* Problem: Minimum Operations to Make the Array Beautiful
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
}
```

C Solution:

```
/*
* Problem: Minimum Operations to Make the Array Beautiful
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

int minOperations(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Minimum Operations to Make the Array Beautiful
// Difficulty: Medium
```

```
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

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 Beautiful
// Difficulty: Medium
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

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) {

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

```
end
end
```

Erlang Solution:

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

Racket Solution:

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