Problem 2170: Minimum Operations to Make the Array Alternating

Problem Information

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

Problem Description

You are given a

0-indexed

array

nums

consisting of

n

positive integers.

The array

nums

is called

alternating

if:

nums[i - 2] == nums[i]

, where

2 <= i <= n - 1

.

nums[i - 1] != nums[i]

, where

1 <= i <= n - 1

.

In one

operation

, you can choose an index

i

and

change

nums[i]

into

any

positive integer.

Return

the

minimum number of operations

required to make the array alternating

.

Example 1:

Input:

nums = [3,1,3,2,4,3]

Output:

3

Explanation:

One way to make the array alternating is by converting it to [3,1,3,

1

,

3

,

1

]. The number of operations required in this case is 3. It can be proven that it is not possible to make the array alternating in less than 3 operations.

Example 2:

Input:

nums = [1,2,2,2,2]

Output:

2

Explanation:

One way to make the array alternating is by converting it to [1,2,

1

,2,

1

]. The number of operations required in this case is 2. Note that the array cannot be converted to [

2

,2,2,2,2] because in this case nums[0] == nums[1] which violates the conditions of an alternating array.

Constraints:

1 <= nums.length <= 10

5

1 <= nums[i] <= 10

5

Code Snippets

C++:

```
class Solution {
public:
```

```
int minimumOperations(vector<int>& nums) {

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

```
end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Minimum Operations to Make the Array Alternating
* Difficulty: Medium
* Tags: array, greedy, hash
*
* 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 minimumOperations(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Operations to Make the Array Alternating
* Difficulty: Medium
* Tags: array, greedy, hash
*
* 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 minimumOperations(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Operations to Make the Array Alternating
Difficulty: Medium
Tags: array, greedy, hash

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Operations to Make the Array Alternating
* Difficulty: Medium
* Tags: array, greedy, hash
*
* 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 minimumOperations = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Operations to Make the Array Alternating
* Difficulty: Medium
* Tags: array, greedy, hash
*
* 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 minimumOperations(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Operations to Make the Array Alternating
* Difficulty: Medium
* Tags: array, greedy, hash
*
* 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 MinimumOperations(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Operations to Make the Array Alternating
* Difficulty: Medium
* Tags: array, greedy, hash
*
* 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 minimumOperations(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Minimum Operations to Make the Array Alternating
// Difficulty: Medium
// Tags: array, greedy, hash
//
// 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 minimumOperations(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Operations to Make the Array Alternating
// Difficulty: Medium
// Tags: array, greedy, hash
//
// 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 minimum_operations(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {
```

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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