Problem 2263: Make Array Non-decreasing or Non-increasing

Problem Information

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

Problem Description

You are given a

0-indexed

integer array

nums

. In one operation, you can:

Choose an index

i

in the range

0 <= i < nums.length

Set

nums[i]

to

nums[i] + 1

or

nums[i] - 1

Return

the

minimum

number of operations to make

nums

non-decreasing

or

non-increasing

.

Example 1:

Input:

nums = [3,2,4,5,0]

Output:

4

Explanation:

One possible way to turn nums into non-increasing order is to: - Add 1 to nums[1] once so that it becomes 3. - Subtract 1 from nums[2] once so it becomes 3. - Subtract 1 from nums[3] twice so it becomes 3. After doing the 4 operations, nums becomes [3,3,3,3,0] which is in non-increasing order. Note that it is also possible to turn nums into [4,4,4,4,0] in 4 operations. It can be proven that 4 is the minimum number of operations needed.

Example 2:

Input:

nums = [2,2,3,4]

Output:

0

Explanation:

nums is already in non-decreasing order, so no operations are needed and we return 0.

Example 3:

Input:

nums = [0]

Output:

0

Explanation:

nums is already in non-decreasing order, so no operations are needed and we return 0.

Constraints:

1 <= nums.length <= 1000

0 <= nums[i] <= 1000

Follow up:

Can you solve it in

O(n*log(n))

time complexity?

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Make Array Non-decreasing or Non-increasing
* Difficulty: Hard
* Tags: array, dp, greedy
*
* 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 convertArray(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Make Array Non-decreasing or Non-increasing
* Difficulty: Hard
* Tags: array, dp, greedy
*
* 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 convertArray(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Make Array Non-decreasing or Non-increasing
Difficulty: Hard
Tags: array, dp, greedy

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

```
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Make Array Non-decreasing or Non-increasing
* Difficulty: Hard
* Tags: array, dp, greedy
*
* 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 convertArray = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Make Array Non-decreasing or Non-increasing
* Difficulty: Hard
* Tags: array, dp, greedy
*
* 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 convertArray(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Make Array Non-decreasing or Non-increasing
* Difficulty: Hard
* Tags: array, dp, greedy
*
* 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 ConvertArray(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Make Array Non-decreasing or Non-increasing
* Difficulty: Hard
* Tags: array, dp, greedy
*
* 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 convertArray(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Make Array Non-decreasing or Non-increasing
// Difficulty: Hard
// Tags: array, dp, greedy
//
// 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 convertArray(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Make Array Non-decreasing or Non-increasing
// Difficulty: Hard
// Tags: array, dp, greedy
//
// 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 convert_array(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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