Problem 3224: Minimum Array Changes to Make Differences Equal

Problem Information

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

Problem Description

You are given an integer array

nums

of size

n

where

n

is

even

, and an integer

k

.

You can perform some changes on the array, where in one change you can replace

any

element in the array with

any

integer in the range from

0

to

k

.

You need to perform some changes (possibly none) such that the final array satisfies the following condition:

There exists an integer

X

such that

abs(a[i] - a[n - i - 1]) = X

for all

(0 <= i < n)

.

Return the

minimum

number of changes required to satisfy the above condition.

Example 1:

Input:

nums = [1,0,1,2,4,3], k = 4

Output:

2

Explanation:

We can perform the following changes:

Replace

nums[1]

by 2. The resulting array is

nums = [1,

2

,1,2,4,3]

.

Replace

nums[3]

by 3. The resulting array is

nums = [1,2,1,

3

,4,3]

.

The integer

X

will be 2.

Example 2:

Input:

nums = [0,1,2,3,3,6,5,4], k = 6

Output:

2

Explanation:

We can perform the following operations:

Replace

nums[3]

by 0. The resulting array is

nums = [0,1,2,

0

,3,6,5,4]

.

Replace

nums[4]

by 4. The resulting array is

nums = [0,1,2,0,

4

,6,5,4]

.

The integer

X

will be 4.

Constraints:

2 <= n == nums.length <= 10

5

n

is even.

0 <= nums[i] <= k <= 10

5

Code Snippets

C++:

```
class Solution {
public:
int minChanges(vector<int>& nums, int k) {
```

```
}
};
```

Java:

```
class Solution {
public int minChanges(int[] nums, int k) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function minChanges(nums: number[], k: number): number {

};
```

C#:

```
public class Solution {
public int MinChanges(int[] nums, int k) {

}
}
```

C:

```
int minChanges(int* nums, int numsSize, int k) {

}
```

Go:

```
func minChanges(nums []int, k int) int {

}
```

Kotlin:

```
class Solution {
fun minChanges(nums: IntArray, k: Int): Int {

}
}
```

Swift:

```
class Solution {
func minChanges(_ nums: [Int], _ k: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn min_changes(nums: Vec<i32>, k: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer} k
# @return {Integer}
def min_changes(nums, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $k
* @return Integer
*/
function minChanges($nums, $k) {

}
}
```

Dart:

```
class Solution {
int minChanges(List<int> nums, int k) {

}
}
```

Scala:

```
object Solution {
def minChanges(nums: Array[Int], k: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_changes(nums :: [integer], k :: integer) :: integer
```

```
def min_changes(nums, k) do

end
end
```

Erlang:

```
-spec min_changes(Nums :: [integer()], K :: integer()) -> integer().
min_changes(Nums, K) ->
.
```

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Minimum Array Changes to Make Differences Equal
* Difficulty: Medium
* Tags: array, 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 minChanges(vector<int>& nums, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Array Changes to Make Differences Equal
* Difficulty: Medium
* Tags: array, 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 minChanges(int[] nums, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Array Changes to Make Differences Equal
Difficulty: Medium
Tags: array, 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 minChanges(self, nums: List[int], k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Array Changes to Make Differences Equal
* Difficulty: Medium
* Tags: array, 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
* @param {number} k
* @return {number}
*/
var minChanges = function(nums, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Array Changes to Make Differences Equal
* Difficulty: Medium
* Tags: array, 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 minChanges(nums: number[], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Array Changes to Make Differences Equal
* Difficulty: Medium
```

```
* Tags: array, 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 MinChanges(int[] nums, int k) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Array Changes to Make Differences Equal
* Difficulty: Medium
* Tags: array, 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 minChanges(int* nums, int numsSize, int k) {

}
```

Go Solution:

```
// Problem: Minimum Array Changes to Make Differences Equal
// Difficulty: Medium
// Tags: array, 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 minChanges(nums []int, k int) int {
```

```
}
```

Kotlin Solution:

```
class Solution {
fun minChanges(nums: IntArray, k: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func minChanges(_ nums: [Int], _ k: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Array Changes to Make Differences Equal
// Difficulty: Medium
// Tags: array, 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 min_changes(nums: Vec<i32>, k: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer} k
# @return {Integer}
def min_changes(nums, k)
```

```
end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $k
* @return Integer
*/
function minChanges($nums, $k) {

}
}
```

Dart Solution:

```
class Solution {
int minChanges(List<int> nums, int k) {

}
}
```

Scala Solution:

```
object Solution {
def minChanges(nums: Array[Int], k: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_changes(nums :: [integer], k :: integer) :: integer
def min_changes(nums, k) do

end
end
```

Erlang Solution:

```
-spec min_changes(Nums :: [integer()], K :: integer()) -> integer().
min_changes(Nums, K) ->
.
```

Racket Solution:

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