Problem 3422: Minimum Operations to Make Subarray Elements Equal

Problem Information

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

Problem Description

You are given an integer array

nums

and an integer

k

. You can perform the following operation any number of times:

Increase or decrease any element of

nums

by 1.

Return the

minimum

number of operations required to ensure that

at least

one

subarray

of size

k

in

nums

has all elements equal.

Example 1:

Input:

nums = [4,-3,2,1,-4,6], k = 3

Output:

5

Explanation:

Use 4 operations to add 4 to

nums[1]

. The resulting array is

[4, 1, 2, 1, -4, 6]

.

Use 1 operation to subtract 1 from

nums[2]

. The resulting array is

[4, 1, 1, 1, -4, 6]

.

The array now contains a subarray

[1, 1, 1]

of size

k = 3

with all elements equal. Hence, the answer is 5.

Example 2:

Input:

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

Output:

0

Explanation:

The subarray

[-2, -2]

of size

k = 2

already contains all equal elements, so no operations are needed. Hence, the answer is 0.

Constraints:

2 <= nums.length <= 10

5

-10

6

<= nums[i] <= 10

6

2 <= k <= nums.length

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

```
func minOperations(nums []int, k int) int64 {
}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

```
impl Solution {
pub fn min_operations(nums: Vec<i32>, k: i32) -> i64 {

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Minimum Operations to Make Subarray Elements Equal
* Difficulty: Medium
* Tags: array, math, hash, queue, heap
*
* 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:
long long minOperations(vector<int>& nums, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Operations to Make Subarray Elements Equal
* Difficulty: Medium
* Tags: array, math, hash, queue, heap
*
* 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 long minOperations(int[] nums, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Operations to Make Subarray Elements Equal
Difficulty: Medium
Tags: array, math, hash, queue, heap

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Operations to Make Subarray Elements Equal
* Difficulty: Medium
* Tags: array, math, hash, queue, heap
*
* 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 minOperations = function(nums, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Operations to Make Subarray Elements Equal
* Difficulty: Medium
* Tags: array, math, hash, queue, heap
*
* 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 minOperations(nums: number[], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Operations to Make Subarray Elements Equal
* Difficulty: Medium
* Tags: array, math, hash, queue, heap
*
* 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 long MinOperations(int[] nums, int k) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Operations to Make Subarray Elements Equal
* Difficulty: Medium
* Tags: array, math, hash, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

long long minOperations(int* nums, int numsSize, int k) {

}
```

Go Solution:

```
// Problem: Minimum Operations to Make Subarray Elements Equal
// Difficulty: Medium
// Tags: array, math, hash, queue, heap
//
// 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 minOperations(nums []int, k int) int64 {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Operations to Make Subarray Elements Equal
// Difficulty: Medium
// Tags: array, math, hash, queue, heap
//
// 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_operations(nums: Vec<i32>, k: i32) -> i64 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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