Problem 3505: Minimum Operations to Make Elements Within K Subarrays Equal

Problem Information

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

Problem Description

You are given an integer array

nums

and two integers,

x

and

k

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

including zero

):

Increase or decrease any element of

nums

by 1.

Return the

minimum

number of operations needed to have

at least

k

non-overlapping

subarrays

of size

exactly

x

in

nums

, where all elements within each subarray are equal.

Example 1:

Input:

nums = [5,-2,1,3,7,3,6,4,-1], x = 3, k = 2

Output:

8

Explanation:

Use 3 operations to add 3 to

nums[1]

and use 2 operations to subtract 2 from

nums[3]

. The resulting array is

[5, 1, 1, 1, 7, 3, 6, 4, -1]

.

Use 1 operation to add 1 to

nums[5]

and use 2 operations to subtract 2 from

nums[6]

. The resulting array is

[5, 1, 1, 1, 7, 4, 4, 4, -1]

.

Now, all elements within each subarray

[1, 1, 1]

(from indices 1 to 3) and

[4, 4, 4]

(from indices 5 to 7) are equal. Since 8 total operations were used, 8 is the output.

Example 2:

Input:

nums = [9,-2,-2,-2,1,5], x = 2, k = 2

Output:

3

Explanation:

Use 3 operations to subtract 3 from

nums[4]

. The resulting array is

[9, -2, -2, -2, -2, 5]

.

Now, all elements within each subarray

[-2, -2]

(from indices 1 to 2) and

[-2, -2]

(from indices 3 to 4) are equal. Since 3 operations were used, 3 is the output.

Constraints:

2 <= nums.length <= 10

5

-10

6

<= nums[i] <= 10

6

2 <= x <= nums.length

1 <= k <= 15

2 <= k * x <= nums.length

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

```
:type x: int
:type k: int
:rtype: int
"""
```

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $x
* @param Integer $k
* @return Integer
```

```
*/
function minOperations($nums, $x, $k) {

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Minimum Operations to Make Elements Within K Subarrays Equal
* Difficulty: Hard
* Tags: array, dp, 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) or O(n * m) for DP table
*/

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

}
};
```

Java Solution:

```
/**
* Problem: Minimum Operations to Make Elements Within K Subarrays Equal
* Difficulty: Hard
* Tags: array, dp, 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) or O(n * m) for DP table
*/

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

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Minimum Operations to Make Elements Within K Subarrays Equal
Difficulty: Hard
Tags: array, dp, 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) or O(n * m) for DP table
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Operations to Make Elements Within K Subarrays Equal
* Difficulty: Hard
* Tags: array, dp, 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) or O(n * m) for DP table
*/
```

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

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Operations to Make Elements Within K Subarrays Equal
* Difficulty: Hard
* Tags: array, dp, 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) or O(n * m) for DP table
*/

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

};
```

C# Solution:

```
/*
* Problem: Minimum Operations to Make Elements Within K Subarrays Equal
* Difficulty: Hard
* Tags: array, dp, 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) or O(n * m) for DP table
*/

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

```
}
}
```

C Solution:

```
/*
* Problem: Minimum Operations to Make Elements Within K Subarrays Equal
* Difficulty: Hard
* Tags: array, dp, 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) or O(n * m) for DP table
*/

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

}
```

Go Solution:

```
// Problem: Minimum Operations to Make Elements Within K Subarrays Equal
// Difficulty: Hard
// Tags: array, dp, 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) or O(n * m) for DP table

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

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Operations to Make Elements Within K Subarrays Equal
// Difficulty: Hard
// Tags: array, dp, 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) or O(n * m) for DP table

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $x
* @param Integer $k
```

```
* @return Integer
*/
function minOperations($nums, $x, $k) {

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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