Problem 1658: Minimum Operations to Reduce X to Zero

Problem Information

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

Problem Description

You are given an integer array

nums

and an integer

x

. In one operation, you can either remove the leftmost or the rightmost element from the array

nums

and subtract its value from

x

. Note that this

modifies

the array for future operations.

Return

the

minimum number

of operations to reduce

x

to

exactly

0

if it is possible

, otherwise, return

-1

.

Example 1:

Input:

nums = [1,1,4,2,3], x = 5

Output:

2

Explanation:

The optimal solution is to remove the last two elements to reduce x to zero.

Example 2:

Input:

nums = [5,6,7,8,9], x = 4

Output:

-1

Example 3:

Input:

nums = [3,2,20,1,1,3], x = 10

Output:

5

Explanation:

The optimal solution is to remove the last three elements and the first two elements (5 operations in total) to reduce x to zero.

Constraints:

1 <= nums.length <= 10

5

1 <= nums[i] <= 10

4

1 <= x <= 10

9

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
```

```
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Minimum Operations to Reduce X to Zero
* Difficulty: Medium
* Tags: array, hash, search
*
* 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 minOperations(vector<int>& nums, int x) {
```

```
}
};
```

Java Solution:

```
/**
* Problem: Minimum Operations to Reduce X to Zero
* Difficulty: Medium
* Tags: array, hash, search
*
* 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 minOperations(int[] nums, int x) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Operations to Reduce X to Zero
Difficulty: Medium
Tags: array, hash, search

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], x: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Operations to Reduce X to Zero
* Difficulty: Medium
* Tags: array, hash, search
*
* 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} x
* @return {number}
*/
var minOperations = function(nums, x) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Operations to Reduce X to Zero
* Difficulty: Medium
* Tags: array, hash, search
*
* 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[], x: number): number {
```

```
};
```

C# Solution:

```
/*
* Problem: Minimum Operations to Reduce X to Zero
* Difficulty: Medium
* Tags: array, hash, search
*
* 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 MinOperations(int[] nums, int x) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Operations to Reduce X to Zero
* Difficulty: Medium
* Tags: array, hash, search
*
* 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 minOperations(int* nums, int numsSize, int x) {

}
```

Go Solution:

```
// Problem: Minimum Operations to Reduce X to Zero
// Difficulty: Medium
```

```
// Tags: array, hash, search
//
// 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, x int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Operations to Reduce X to Zero
// Difficulty: Medium
// Tags: array, hash, search
//
// 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>, x: i32) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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