Problem 2702: Minimum Operations to Make Numbers Non-positive

Problem Information

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

Problem Description

You are given a

0-indexed

integer array

nums

and two integers

x

and

y

. In one operation, you must choose an index

i

such that

0 <= i < nums.length

and perform the following:

Decrement

nums[i]

by

x

.

Decrement values by

y

at all indices except the

i

th

one.

Return

the minimum number of operations to make all the integers in

nums

less than or equal to zero.

Example 1:

Input:

nums = [3,4,1,7,6], x = 4, y = 2

Output:

3

Explanation:

You will need three operations. One of the optimal sequence of operations is: Operation 1: Choose i = 3. Then, nums = [1,2,-1,3,4]. Operation 2: Choose i = 3. Then, nums = [-1,0,-3,-1,2]. Operation 3: Choose i = 4. Then, nums = [-3,-2,-5,-3,-2]. Now, all the numbers in nums are non-positive. Therefore, we return 3.

Example 2:

Input:

nums = [1,2,1], x = 2, y = 1

Output:

1

Explanation:

We can perform the operation once on i = 1. Then, nums becomes [0,0,0]. All the positive numbers are removed, and therefore, we return 1.

Constraints:

1 <= nums.length <= 10

5

1 <= nums[i] <= 10

9

1 <= y < x <= 10

9

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {number[]} nums
* @param {number} x
* @param {number} y
* @return {number}
*/
```

```
var minOperations = function(nums, x, y) {

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

```
}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

```
}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Minimum Operations to Make Numbers Non-positive
```

```
* Difficulty: Hard
* Tags: array, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

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

}
};
```

Java Solution:

```
/**
* Problem: Minimum Operations to Make Numbers Non-positive
* Difficulty: Hard
* Tags: array, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

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

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Operations to Make Numbers Non-positive
Difficulty: Hard
Tags: array, search

Approach: Use two pointers or sliding window technique
```

```
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Operations to Make Numbers Non-positive
* Difficulty: Hard
* Tags: array, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Operations to Make Numbers Non-positive
* Difficulty: Hard
* Tags: array, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

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

};
```

C# Solution:

```
/*
* Problem: Minimum Operations to Make Numbers Non-positive
* Difficulty: Hard
* Tags: array, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

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

}
}
```

C Solution:

```
/*
* Problem: Minimum Operations to Make Numbers Non-positive
* Difficulty: Hard
* Tags: array, search
*
* Approach: Use two pointers or sliding window technique
```

```
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

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

}
```

Go Solution:

```
// Problem: Minimum Operations to Make Numbers Non-positive
// Difficulty: Hard
// Tags: array, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

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

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Operations to Make Numbers Non-positive
// Difficulty: Hard
// Tags: array, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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