Problem 2598: Smallest Missing Non-negative Integer After Operations

Problem Information

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

Problem Description

You are given a

0-indexed

integer array

nums

and an integer

value

.

In one operation, you can add or subtract

value

from any element of

nums

.

For example, if

nums = [1,2,3]

and

value = 2

, you can choose to subtract

value

from

nums[0]

to make

nums = [-1,2,3]

.

The MEX (minimum excluded) of an array is the smallest missing

non-negative

integer in it.

For example, the MEX of

[-1,2,3]

is

0

while the MEX of

[1,0,3]

is

2

.

Return

the maximum MEX of

nums

after applying the mentioned operation

any number of times

.

Example 1:

Input:

nums = [1,-10,7,13,6,8], value = 5

Output:

4

Explanation:

One can achieve this result by applying the following operations: - Add value to nums[1] twice to make nums = [1,

0

,7,13,6,8] - Subtract value from nums[2] once to make nums = [1,0,

2

,13,6,8] - Subtract value from nums[3] twice to make nums = [1,0,2,

3

,6,8] The MEX of nums is 4. It can be shown that 4 is the maximum MEX we can achieve.

Example 2:

Input:

nums = [1,-10,7,13,6,8], value = 7

Output:

2

Explanation:

One can achieve this result by applying the following operation: - subtract value from nums[2] once to make nums = [1,-10,

0

,13,6,8] The MEX of nums is 2. It can be shown that 2 is the maximum MEX we can achieve.

Constraints:

1 <= nums.length, value <= 10

5

-10

9

<= nums[i] <= 10

9

Code Snippets

C++:

```
class Solution {
public:
int findSmallestInteger(vector<int>& nums, int value) {

}
};
```

Java:

```
class Solution {
public int findSmallestInteger(int[] nums, int value) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

```
};
```

TypeScript:

```
function findSmallestInteger(nums: number[], value: number): number {

};
```

C#:

```
public class Solution {
public int FindSmallestInteger(int[] nums, int value) {

}
}
```

C:

```
int findSmallestInteger(int* nums, int numsSize, int value) {

}
```

Go:

```
func findSmallestInteger(nums []int, value int) int {

}
```

Kotlin:

```
class Solution {
fun findSmallestInteger(nums: IntArray, value: Int): Int {

}
}
```

Swift:

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

```
}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer} value
# @return {Integer}
def find_smallest_integer(nums, value)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $value
* @return Integer
*/
function findSmallestInteger($nums, $value) {

}
}
```

Dart:

```
class Solution {
int findSmallestInteger(List<int> nums, int value) {

}
}
```

Scala:

```
object Solution {
def findSmallestInteger(nums: Array[Int], value: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec find_smallest_integer(nums :: [integer], value :: integer) :: integer
def find_smallest_integer(nums, value) do

end
end
```

Erlang:

```
-spec find_smallest_integer(Nums :: [integer()], Value :: integer()) ->
integer().
find_smallest_integer(Nums, Value) ->
.
```

Racket:

```
(define/contract (find-smallest-integer nums value)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Smallest Missing Non-negative Integer After Operations
* Difficulty: Medium
* Tags: array, greedy, math, 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 findSmallestInteger(vector<int>& nums, int value) {

}
};
```

Java Solution:

```
/**
* Problem: Smallest Missing Non-negative Integer After Operations
* Difficulty: Medium
* Tags: array, greedy, math, 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 findSmallestInteger(int[] nums, int value) {

}
}
```

Python3 Solution:

```
"""
Problem: Smallest Missing Non-negative Integer After Operations
Difficulty: Medium
Tags: array, greedy, math, 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 findSmallestInteger(self, nums: List[int], value: int) -> int:
```

```
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Smallest Missing Non-negative Integer After Operations
* Difficulty: Medium
* Tags: array, greedy, math, 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} value
* @return {number}
*/
var findSmallestInteger = function(nums, value) {

};
```

TypeScript Solution:

```
/**
* Problem: Smallest Missing Non-negative Integer After Operations
* Difficulty: Medium
* Tags: array, greedy, math, 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 findSmallestInteger(nums: number[], value: number): number {

};
```

C# Solution:

```
/*
* Problem: Smallest Missing Non-negative Integer After Operations
* Difficulty: Medium
* Tags: array, greedy, math, 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 FindSmallestInteger(int[] nums, int value) {

}
}
```

C Solution:

```
/*
* Problem: Smallest Missing Non-negative Integer After Operations
* Difficulty: Medium
* Tags: array, greedy, math, 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 findSmallestInteger(int* nums, int numsSize, int value) {
```

```
}
```

Go Solution:

```
// Problem: Smallest Missing Non-negative Integer After Operations
// Difficulty: Medium
// Tags: array, greedy, math, 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 findSmallestInteger(nums []int, value int) int {

}
```

Kotlin Solution:

```
class Solution {
fun findSmallestInteger(nums: IntArray, value: Int): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Smallest Missing Non-negative Integer After Operations
// Difficulty: Medium
// Tags: array, greedy, math, 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 find_smallest_integer(nums: Vec<i32>, value: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer} value
# @return {Integer}
def find_smallest_integer(nums, value)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $value
* @return Integer
*/
function findSmallestInteger($nums, $value) {

}
}
```

Dart Solution:

```
class Solution {
int findSmallestInteger(List<int> nums, int value) {

}
}
```

Scala Solution:

```
object Solution {
def findSmallestInteger(nums: Array[Int], value: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec find_smallest_integer(nums :: [integer], value :: integer) :: integer
def find_smallest_integer(nums, value) do

end
end
```

Erlang Solution:

```
-spec find_smallest_integer(Nums :: [integer()], Value :: integer()) ->
integer().
find_smallest_integer(Nums, Value) ->
.
```

Racket Solution:

```
(define/contract (find-smallest-integer nums value)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```