Problem 3471: Find the Largest Almost Missing Integer

Problem Information

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

Problem Description

You are given an integer array

nums

and an integer

k

.

An integer

x

is

almost missing

from

nums

if

x

appears in

exactly

one subarray of size

k

within

nums

.

Return the

largest

almost missing

integer from

nums

. If no such integer exists, return

-1

.

A

subarray

is a contiguous sequence of elements within an array.

Example 1:

Input:

nums = [3,9,2,1,7], k = 3

Output:

7

Explanation:

1 appears in 2 subarrays of size 3:

[9, 2, 1]

and

[2, 1, 7]

.

2 appears in 3 subarrays of size 3:

[3, 9, 2]

,

[9, 2, 1]

,

[2, 1, 7]

.

3 appears in 1 subarray of size 3:

[3, 9, 2]

.

7 appears in 1 subarray of size 3:

[2, 1, 7]

.

9 appears in 2 subarrays of size 3:

[3, 9, 2]

, and

[9, 2, 1]

.

We return 7 since it is the largest integer that appears in exactly one subarray of size

k

.

Example 2:

Input:

nums = [3,9,7,2,1,7], k = 4

Output:

3

Explanation:

1 appears in 2 subarrays of size 4:

[9, 7, 2, 1]

,

[7, 2, 1, 7]

.

2 appears in 3 subarrays of size 4:

[3, 9, 7, 2]

,

[9, 7, 2, 1]

,

[7, 2, 1, 7]

.

3 appears in 1 subarray of size 4:

[3, 9, 7, 2]

.

7 appears in 3 subarrays of size 4:

[3, 9, 7, 2]

,

[9, 7, 2, 1]

,

[7, 2, 1, 7]

.

9 appears in 2 subarrays of size 4:

[3, 9, 7, 2]

,

[9, 7, 2, 1]

.

We return 3 since it is the largest and only integer that appears in exactly one subarray of size

k

.

Example 3:

Input:

nums = [0,0], k = 1

Output:

-1

Explanation:

There is no integer that appears in only one subarray of size 1.

Constraints:

1 <= nums.length <= 50

0 <= nums[i] <= 50

1 <= k <= nums.length

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

```
};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
int largestInteger(int* nums, int numsSize, int k) {

}
```

Go:

```
func largestInteger(nums []int, k int) int {

}
```

Kotlin:

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

}
}
```

Swift:

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

```
}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Find the Largest Almost Missing Integer
* Difficulty: Easy
* Tags: array, 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 largestInteger(vector<int>& nums, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Find the Largest Almost Missing Integer
* Difficulty: Easy
* Tags: array, 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 largestInteger(int[] nums, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Find the Largest Almost Missing Integer
Difficulty: Easy
Tags: array, 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 largestInteger(self, nums: List[int], k: int) -> int:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Find the Largest Almost Missing Integer
* Difficulty: Easy
* Tags: array, 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} k
* @return {number}
*/
var largestInteger = function(nums, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Find the Largest Almost Missing Integer
* Difficulty: Easy
* Tags: array, 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 largestInteger(nums: number[], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Find the Largest Almost Missing Integer
* Difficulty: Easy
* Tags: array, 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 LargestInteger(int[] nums, int k) {

}
}
```

C Solution:

```
/*
* Problem: Find the Largest Almost Missing Integer
* Difficulty: Easy
* Tags: array, 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 largestInteger(int* nums, int numsSize, int k) {

}
```

Go Solution:

```
// Problem: Find the Largest Almost Missing Integer
// Difficulty: Easy
// Tags: array, 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 largestInteger(nums []int, k int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Find the Largest Almost Missing Integer
// Difficulty: Easy
// Tags: array, 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 largest_integer(nums: Vec<i32>, k: i32) -> i32 {
```

```
}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
```

```
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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