Problem 3134: Find the Median of the Uniqueness Array

Problem Information

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

Problem Description

You are given an integer array

nums

. The

uniqueness array

of

nums

is the sorted array that contains the number of distinct elements of all the

subarrays

of

nums

. In other words, it is a sorted array consisting of

distinct(nums[i..j])

, for all

0 <= i <= j < nums.length

.

Here,

distinct(nums[i..j])

denotes the number of distinct elements in the subarray that starts at index

i

and ends at index

j

.

Return the

median

of the

uniqueness array

of

nums

.

Note

that the

median

of an array is defined as the middle element of the array when it is sorted in non-decreasing order. If there are two choices for a median, the

smaller

of the two values is taken.

Example 1:

Input:

nums = [1,2,3]

Output:

1

Explanation:

The uniqueness array of

nums

is

[distinct(nums[0..0]), distinct(nums[1..1]), distinct(nums[2..2]), distinct(nums[0..1]), distinct(nums[1..2]), distinct(nums[0..2])]

which is equal to

[1, 1, 1, 2, 2, 3]

. The uniqueness array has a median of 1. Therefore, the answer is 1.

Example 2:

Input:

nums = [3,4,3,4,5]

Output:

2

Explanation:

The uniqueness array of

nums

is

[1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3]

. The uniqueness array has a median of 2. Therefore, the answer is 2.

Example 3:

Input:

nums = [4,3,5,4]

Output:

2

Explanation:

The uniqueness array of

nums

is

[1, 1, 1, 1, 2, 2, 2, 3, 3, 3]

. The uniqueness array has a median of 2. Therefore, the answer is 2.

Constraints:

1 <= nums.length <= 10

5

1 <= nums[i] <= 10

5

Code Snippets

C++:

```
class Solution {
public:
int medianOfUniquenessArray(vector<int>& nums) {

}
};
```

Java:

```
class Solution {
public int medianOfUniquenessArray(int[] nums) {

}
}
```

Python3:

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

Python:

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

```
:rtype: int
"""
```

JavaScript:

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

};
```

TypeScript:

```
function medianOfUniquenessArray(nums: number[]): number {

};
```

C#:

```
public class Solution {
public int MedianOfUniquenessArray(int[] nums) {

}
}
```

C:

```
int medianOfUniquenessArray(int* nums, int numsSize) {

}
```

Go:

```
func medianOfUniquenessArray(nums []int) int {

}
```

Kotlin:

```
class Solution {
fun medianOfUniquenessArray(nums: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func medianOfUniquenessArray(_ nums: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn median_of_uniqueness_array(nums: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @return {Integer}
def median_of_uniqueness_array(nums)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @return Integer
*/
function medianOfUniquenessArray($nums) {

}
}
```

Dart:

```
class Solution {
int medianOfUniquenessArray(List<int> nums) {

}
}
```

Scala:

```
object Solution {
def medianOfUniquenessArray(nums: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec median_of_uniqueness_array(nums :: [integer]) :: integer
def median_of_uniqueness_array(nums) do

end
end
```

Erlang:

```
-spec median_of_uniqueness_array(Nums :: [integer()]) -> integer().
median_of_uniqueness_array(Nums) ->
.
```

Racket:

```
(define/contract (median-of-uniqueness-array nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find the Median of the Uniqueness Array
* Difficulty: Hard
* Tags: array, hash, sort, 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 medianOfUniquenessArray(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Find the Median of the Uniqueness Array
* Difficulty: Hard
* Tags: array, hash, sort, 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 medianOfUniquenessArray(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Find the Median of the Uniqueness Array
Difficulty: Hard
Tags: array, hash, sort, 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 medianOfUniquenessArray(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Find the Median of the Uniqueness Array
* Difficulty: Hard
* Tags: array, hash, sort, 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
* @return {number}
*/
var medianOfUniquenessArray = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Find the Median of the Uniqueness Array
* Difficulty: Hard
* Tags: array, hash, sort, 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 medianOfUniquenessArray(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Find the Median of the Uniqueness Array
* Difficulty: Hard
* Tags: array, hash, sort, 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 MedianOfUniquenessArray(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Find the Median of the Uniqueness Array
* Difficulty: Hard
* Tags: array, hash, sort, 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 medianOfUniquenessArray(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Find the Median of the Uniqueness Array
// Difficulty: Hard
// Tags: array, hash, sort, 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 medianOfUniquenessArray(nums []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun medianOfUniquenessArray(nums: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func medianOfUniquenessArray(_ nums: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Find the Median of the Uniqueness Array
// Difficulty: Hard
// Tags: array, hash, sort, 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 median_of_uniqueness_array(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @return {Integer}
def median_of_uniqueness_array(nums)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @return Integer
*/
function medianOfUniquenessArray($nums) {

}
}
```

Dart Solution:

```
class Solution {
int medianOfUniquenessArray(List<int> nums) {

}
}
```

Scala Solution:

```
object Solution {
def medianOfUniquenessArray(nums: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec median_of_uniqueness_array(nums :: [integer]) :: integer
def median_of_uniqueness_array(nums) do

end
end
```

Erlang Solution:

```
-spec median_of_uniqueness_array(Nums :: [integer()]) -> integer().
median_of_uniqueness_array(Nums) ->
.
```

Racket Solution:

```
(define/contract (median-of-uniqueness-array nums)
(-> (listof exact-integer?) exact-integer?)
)
```