Problem 961: N-Repeated Element in Size 2N Array

Problem Information

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

Problem Description

You are given an integer array

nums

with the following properties:

nums.length == 2 * n

.

nums

contains

n + 1

unique

elements.

Exactly one element of

nums

is repeated

n

times.

Return

the element that is repeated

n

times

.

Example 1:

Input:

nums = [1,2,3,3]

Output:

3

Example 2:

Input:

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

Output:

2

Example 3:

Input:

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

Output:

5

Constraints:

2 <= n <= 5000

nums.length == 2 * n

0 <= nums[i] <= 10

4

nums

contains

n + 1

unique

elements and one of them is repeated exactly

n

times.

Code Snippets

C++:

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

```
}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (repeated-n-times nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: N-Repeated Element in Size 2N Array
* 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 repeatedNTimes(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: N-Repeated Element in Size 2N Array
* 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 repeatedNTimes(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: N-Repeated Element in Size 2N Array
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 repeatedNTimes(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: N-Repeated Element in Size 2N Array
* 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
* @return {number}
*/
var repeatedNTimes = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: N-Repeated Element in Size 2N Array
* 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 repeatedNTimes(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: N-Repeated Element in Size 2N Array
* 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 RepeatedNTimes(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: N-Repeated Element in Size 2N Array
* 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 repeatedNTimes(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: N-Repeated Element in Size 2N Array
// 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 repeatedNTimes(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: N-Repeated Element in Size 2N Array
// 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 repeated_n_times(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {
```

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (repeated-n-times nums)
(-> (listof exact-integer?) exact-integer?)
)
```