Problem 137: Single Number II

Problem Information

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

Problem Description

Given an integer array

nums

where every element appears

three times

except for one, which appears

exactly once

.

Find the single element and return it

.

You must implement a solution with a linear runtime complexity and use only constant extra space.

Example 1:

Input:

nums = [2,2,3,2]

Output:

3

Example 2:

Input:

nums = [0,1,0,1,0,1,99]

Output:

99

Constraints:

1 <= nums.length <= 3 * 10

4

-2

31

<= nums[i] <= 2

31

- 1

Each element in

nums

appears exactly

three times

except for one element which appears

once

.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (single-number nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Single Number II
* Difficulty: Medium
* Tags: array
*
* 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 singleNumber(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Single Number II
* Difficulty: Medium
* Tags: array
*
* 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 singleNumber(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Single Number II
Difficulty: Medium
Tags: array

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 singleNumber(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Single Number II
* Difficulty: Medium
* Tags: array
*
* 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
* @return {number}
*/
var singleNumber = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Single Number II
* Difficulty: Medium
* Tags: array
*
* 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 singleNumber(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Single Number II
* Difficulty: Medium
* Tags: array
*
* 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 SingleNumber(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Single Number II
* Difficulty: Medium
* Tags: array
*
* 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 singleNumber(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Single Number II
// Difficulty: Medium
// Tags: array
//
// 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 singleNumber(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Single Number II
// Difficulty: Medium
// Tags: array
//
// 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 single_number(nums: Vec<i32>) -> i32 {

}
```

```
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (single-number nums)
(-> (listof exact-integer?) exact-integer?)
)
```