Problem 540: Single Element in a Sorted Array

Problem Information

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

Problem Description

You are given a sorted array consisting of only integers where every element appears exactly twice, except for one element which appears exactly once.

Return

the single element that appears only once

.

Your solution must run in

O(log n)

time and

O(1)

space.

Example 1:

Input:

nums = [1,1,2,3,3,4,4,8,8]

Output:

2

Example 2:

Input:

nums = [3,3,7,7,10,11,11]

Output:

10

Constraints:

1 <= nums.length <= 10

5

0 <= nums[i] <= 10

5

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Single Element in a Sorted Array
* Difficulty: Medium
* Tags: array, sort, search
*
* 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 singleNonDuplicate(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Single Element in a Sorted Array
* Difficulty: Medium
* Tags: array, sort, search
*
* 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 singleNonDuplicate(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Single Element in a Sorted Array
Difficulty: Medium
Tags: array, sort, search

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Single Element in a Sorted Array
* Difficulty: Medium
* Tags: array, sort, search
*
* 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 singleNonDuplicate = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Single Element in a Sorted Array
* Difficulty: Medium
* Tags: array, sort, search
*
* 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 singleNonDuplicate(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Single Element in a Sorted Array
* Difficulty: Medium
* Tags: array, sort, search
*
* 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 SingleNonDuplicate(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Single Element in a Sorted Array
* Difficulty: Medium
* Tags: array, sort, search
*
* 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 singleNonDuplicate(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Single Element in a Sorted Array
// Difficulty: Medium
// Tags: array, sort, search
//
// 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 singleNonDuplicate(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

Rust Solution:

```
// Problem: Single Element in a Sorted Array
// Difficulty: Medium
// Tags: array, sort, search
//
// 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_non_duplicate(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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