Problem 260: Single Number III

Problem Information

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

Problem Description

Given an integer array

nums

, in which exactly two elements appear only once and all the other elements appear exactly twice. Find the two elements that appear only once. You can return the answer in

any order

.

You must write an algorithm that runs in linear runtime complexity and uses only constant extra space.

Example 1:

Input:

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

Output:

[3,5]

Explanation:

[5, 3] is also a valid answer.

Example 2:

Input:

nums = [-1,0]

Output:

[-1,0]

Example 3:

Input:

nums = [0,1]

Output:

[1,0]

Constraints:

2 <= nums.length <= 3 * 10

4

-2

31

<= nums[i] <= 2

31

- 1

Each integer in

nums

will appear twice, only two integers will appear once.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* singleNumber(int* nums, int numsSize, int* returnSize) {

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}

}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```cpp
/*
* Problem: Single Number III
* 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:
vector<int> singleNumber(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Single Number III
* 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 III
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]) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Single Number III
* 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 III
* 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 III
* 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 III
* 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
*/

/**
```

```
* Note: The returned array must be malloced, assume caller calls free().
*/
int* singleNumber(int* nums, int numsSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Single Number III
// 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): IntArray {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Single Number III
// 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>) -> Vec<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 {
List<int> singleNumber(List<int> nums) {

}
}
```

Scala Solution:

```
object Solution {
def singleNumber(nums: Array[Int]): Array[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?) (listof exact-integer?))
)
```