# Problem 2150: Find All Lonely Numbers in the Array

## Problem Information

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

## Problem Description

You are given an integer array

nums

. A number

x

is

lonely

when it appears only

once

, and no

adjacent

numbers (i.e.

x + 1

and

x - 1)

appear in the array.

Return

all

lonely numbers in

nums

. You may return the answer in

any order

.

Example 1:

Input:

nums = [10,6,5,8]

Output:

[10,8]

Explanation:

- 10 is a lonely number since it appears exactly once and 9 and 11 does not appear in nums. - 8 is a lonely number since it appears exactly once and 7 and 9 does not appear in nums. - 5 is not a lonely number since 6 appears in nums and vice versa. Hence, the lonely numbers in nums are [10, 8]. Note that [8, 10] may also be returned.

Example 2:

Input:

nums = [1,3,5,3]

Output:

[1,5]

Explanation:

- 1 is a lonely number since it appears exactly once and 0 and 2 does not appear in nums. - 5 is a lonely number since it appears exactly once and 4 and 6 does not appear in nums. - 3 is not a lonely number since it appears twice. Hence, the lonely numbers in nums are [1, 5]. Note that [5, 1] may also be returned.

Constraints:

1 <= nums.length <= 10

5

0 <= nums[i] <= 10

6

## Code Snippets

**C++:**

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

}
};
```

**Java:**

```
class Solution {
public List<Integer> findLonely(int[] nums) {
```

```
}
}
```

**Python3:**

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

**Python:**

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

**JavaScript:**

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

};
```

**TypeScript:**

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

};
```

**C#:**

```
public class Solution {
public IList<int> FindLonely(int[] nums) {

}
}
```

**C:**

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

}
```

**Go:**

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

}
```

**Kotlin:**

```
class Solution {
fun findLonely(nums: IntArray): List<Int> {

}
}
```

**Swift:**

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

}
}
```

**Rust:**

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

}
}
```

**Ruby:**

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

```
end
```

**PHP:**

```
class Solution {

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

}
}
```

**Dart:**

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

}
}
```

**Scala:**

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

}
}
```

**Elixir:**

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

end
end
```

**Erlang:**

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

**Racket:**

```
(define/contract (find-lonely nums)
(-> (listof exact-integer?) (listof exact-integer?))
)
```

## Solutions

### C++ Solution:

```
/*
* Problem: Find All Lonely Numbers in the Array
* Difficulty: Medium
* 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:
vector<int> findLonely(vector<int>& nums) {

}
};
```

### Java Solution:

```
/**
* Problem: Find All Lonely Numbers in the Array
* Difficulty: Medium
* 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 List<Integer> findLonely(int[] nums) {

}
}
```

### Python3 Solution:

```
"""
Problem: Find All Lonely Numbers in the Array
Difficulty: Medium
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 findLonely(self, nums: List[int]) -> List[int]:
# TODO: Implement optimized solution
pass
```

### Python Solution:

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

### JavaScript Solution:

```
/**
* Problem: Find All Lonely Numbers in the Array
* Difficulty: Medium
* 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 findLonely = function(nums) {

};
```

### TypeScript Solution:

```
/**
* Problem: Find All Lonely Numbers in the Array
* Difficulty: Medium
* 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 findLonely(nums: number[]): number[] {

};
```

### C# Solution:

```
/*
* Problem: Find All Lonely Numbers in the Array
* Difficulty: Medium
* 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 IList<int> FindLonely(int[] nums) {

}
}
```

### C Solution:

```
/*
* Problem: Find All Lonely Numbers in the Array
* Difficulty: Medium
* 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
*/

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

}
```

### Go Solution:

```
// Problem: Find All Lonely Numbers in the Array
// Difficulty: Medium
// 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 findLonely(nums []int) []int {

}
```

### Kotlin Solution:

```
class Solution {
fun findLonely(nums: IntArray): List<Int> {

}
}
```

**Swift Solution:**

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

}
}
```

**Rust Solution:**

```
// Problem: Find All Lonely Numbers in the Array
// Difficulty: Medium
// 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 find_lonely(nums: Vec<i32>) -> Vec<i32> {

}
}
```

**Ruby Solution:**

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

end
```

**PHP Solution:**

```
class Solution {
```

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

}
}
```

**Dart Solution:**

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

}
}
```

**Scala Solution:**

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

}
}
```

**Elixir Solution:**

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

end
end
```

**Erlang Solution:**

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

**Racket Solution:**

```
(define/contract (find-lonely nums)
(-> (listof exact-integer?) (listof exact-integer?))
)
```