Problem 3595: Once Twice

Problem Information

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

Problem Description

You are given an integer array

nums

. In this array:

Exactly one element appears

once

.

Exactly one element appears

twice

.

All other elements appear

exactly three times

.

Return an integer array of length 2, where the first element is the one that appears

once

, and the second is the one that appears

twice

.

Your solution must run in

O(n)

time and

O(1)

space.

Example 1:

Input:

nums = [2,2,3,2,5,5,5,7,7]

Output:

[3,7]

Explanation:

The element 3 appears

once

, and the element 7 appears

twice

. The remaining elements each appear

three times

.

Example 2:

Input:

nums = [4,4,6,4,9,9,9,6,8]

Output:

[8,6]

Explanation:

The element 8 appears

once

, and the element 6 appears

twice

. The remaining elements each appear

three times

.

Constraints:

3 <= nums.length <= 10

5

-2

31

<= nums[i] <= 2

31

- 1

nums.length

is a multiple of 3.

Exactly one element appears once, one element appears twice, and all other elements appear three times.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @return Integer[]
```

```
*/
function onceTwice($nums) {

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (once-twice nums)
(-> (listof exact-integer?) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Once Twice
* 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> onceTwice(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Once Twice
* 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[] onceTwice(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Once Twice
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 onceTwice(self, nums: List[int]) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Once Twice
* 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 onceTwice = function(nums) {
```

```
};
```

TypeScript Solution:

```
/**
* Problem: Once Twice
* 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 onceTwice(nums: number[]): number[] {

};
```

C# Solution:

```
/*
* Problem: Once Twice
* 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[] OnceTwice(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Once Twice
* 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* onceTwice(int* nums, int numsSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Once Twice
// 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 onceTwice(nums []int) []int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

```
}
}
```

Rust Solution:

```
// Problem: Once Twice
// 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 once_twice(nums: Vec<i32>) -> Vec<i32> {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (once-twice nums)
(-> (listof exact-integer?) (listof exact-integer?))
)
```