Problem 229: Majority Element II

Problem Information

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

Problem Description

Given an integer array of size

n

, find all elements that appear more than

■ n/3 ■

times.

Example 1:

Input:

nums = [3,2,3]

Output:

[3]

Example 2:

Input:

nums = [1]

Output:

[1]

Example 3:

Input:

nums = [1,2]

Output:

[1,2]

Constraints:

1 <= nums.length <= 5 * 10

4

-10

9

<= nums[i] <= 10

9

Follow up:

Could you solve the problem in linear time and in

O(1)

space?

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (majority-element nums)
(-> (listof exact-integer?) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Majority Element II
* Difficulty: Medium
* Tags: array, hash, sort
*
* 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> majorityElement(vector<int>& nums) {
}
};
```

Java Solution:

```
/**
* Problem: Majority Element II
* Difficulty: Medium
* Tags: array, hash, sort
*
* 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> majorityElement(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Majority Element II
Difficulty: Medium
Tags: array, hash, sort

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Majority Element II
* Difficulty: Medium
* Tags: array, hash, sort
*
* 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 majorityElement = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Majority Element II
* Difficulty: Medium
* Tags: array, hash, sort
*
* 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 majorityElement(nums: number[]): number[] {

};
```

C# Solution:

```
/*
* Problem: Majority Element II
* Difficulty: Medium
* Tags: array, hash, sort
*
* 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> MajorityElement(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Majority Element II
* Difficulty: Medium
* Tags: array, hash, sort
*
* 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* majorityElement(int* nums, int numsSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Majority Element II
// Difficulty: Medium
// Tags: array, hash, sort
```

```
//
// 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 majorityElement(nums []int) []int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Majority Element II
// Difficulty: Medium
// Tags: array, hash, sort
//
// 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 majority_element(nums: Vec<i32>) -> Vec<i32> {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

```
end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (majority-element nums)
(-> (listof exact-integer?) (listof exact-integer?))
)
```