Problem 442: Find All Duplicates in an Array

Problem Information

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

Problem Description

Given an integer array

nums

of length

n

where all the integers of

nums

are in the range

[1, n]

and each integer appears

at most

twice

, return

an array of all the integers that appears

twice

.

You must write an algorithm that runs in

O(n)

time and uses only

constant

auxiliary space, excluding the space needed to store the output

Example 1:

Input:

nums = [4,3,2,7,8,2,3,1]

Output:

[2,3]

Example 2:

Input:

nums = [1,1,2]

Output:

[1]

Example 3:

Input:

nums = [1]

Output:

[]

Constraints:

n == nums.length

1 <= n <= 10

5

1 <= nums[i] <= n

Each element in

nums

appears

once

or

twice

.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

```
}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

```
end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Find All Duplicates in an Array
* 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> findDuplicates(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Find All Duplicates in an Array
```

```
* 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> findDuplicates(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Find All Duplicates in an Array
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 findDuplicates(self, nums: List[int]) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Find All Duplicates in an Array
* 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 findDuplicates = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Find All Duplicates in an Array
* 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 findDuplicates(nums: number[]): number[] {

};
```

C# Solution:

```
/*
* Problem: Find All Duplicates in an Array
* 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> FindDuplicates(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Find All Duplicates in an Array
* 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* findDuplicates(int* nums, int numsSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Find All Duplicates in an Array
// 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 findDuplicates(nums []int) []int {
```

```
}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Find All Duplicates in an Array
// 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 find_duplicates(nums: Vec<i32>) -> Vec<i32> {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

```
.
```

Racket Solution:

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