Problem 448: Find All Numbers Disappeared in an Array

Problem Information

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

Problem Description

Given an array

nums

of

n

integers where

nums[i]

is in the range

[1, n]

, return

an array of all the integers in the range

[1, n]

that do not appear in

nums

.

Example 1:

Input:

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

Output:

[5,6]

Example 2:

Input:

nums = [1,1]

Output:

[2]

Constraints:

n == nums.length

1 <= n <= 10

5

1 <= nums[i] <= n

Follow up:

Could you do it without extra space and in

O(n)

runtime? You may assume the returned list does not count as extra space.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {number[]} nums
* @return {number[]}
*/
```

```
var findDisappearedNumbers = function(nums) {

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Find All Numbers Disappeared in an Array
* Difficulty: Easy
* 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> findDisappearedNumbers(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Find All Numbers Disappeared in an Array
* Difficulty: Easy
* 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> findDisappearedNumbers(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Find All Numbers Disappeared in an Array
Difficulty: Easy
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 findDisappearedNumbers(self, nums: List[int]) -> List[int]:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Find All Numbers Disappeared in an Array
* Difficulty: Easy
* 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 findDisappearedNumbers = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Find All Numbers Disappeared in an Array
* Difficulty: Easy
* 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 findDisappearedNumbers(nums: number[]): number[] {

};
```

C# Solution:

```
/*
* Problem: Find All Numbers Disappeared in an Array
* Difficulty: Easy
* 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> FindDisappearedNumbers(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Find All Numbers Disappeared in an Array
* Difficulty: Easy
* 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* findDisappearedNumbers(int* nums, int numsSize, int* returnSize) {
```

```
}
```

Go Solution:

```
// Problem: Find All Numbers Disappeared in an Array
// Difficulty: Easy
// 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 findDisappearedNumbers(nums []int) []int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Find All Numbers Disappeared in an Array
// Difficulty: Easy
// 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_disappeared_numbers(nums: Vec<i32>) -> Vec<i32> {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

```
}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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