

Problem 3289: The Two Sneaky Numbers of Digitville

Problem Information

Difficulty: [Easy](#)

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

In the town of Digitville, there was a list of numbers called

nums

containing integers from

0

to

$n - 1$

. Each number was supposed to appear

exactly once

in the list, however,

two

mischievous numbers sneaked in an

additional time

, making the list longer than usual.

As the town detective, your task is to find these two sneaky numbers. Return an array of size

two

containing the two numbers (in

any order

), so peace can return to Digitville.

Example 1:

Input:

nums = [0,1,1,0]

Output:

[0,1]

Explanation:

The numbers 0 and 1 each appear twice in the array.

Example 2:

Input:

nums = [0,3,2,1,3,2]

Output:

[2,3]

Explanation:

The numbers 2 and 3 each appear twice in the array.

Example 3:

Input:

nums = [7,1,5,4,3,4,6,0,9,5,8,2]

Output:

[4,5]

Explanation:

The numbers 4 and 5 each appear twice in the array.

Constraints:

$2 \leq n \leq 100$

`nums.length == n + 2`

$0 \leq \text{nums}[i] < n$

The input is generated such that

nums

contains

exactly

two repeated elements.

Code Snippets

C++:

```
class Solution {  
public:
```

```
vector<int> getSneakyNumbers(vector<int>& nums) {

}

};
```

Java:

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

}

}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

```
public class Solution {  
    public int[] GetSneakyNumbers(int[] nums) {  
  
    }  
}
```

C:

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

Go:

```
func getSneakyNumbers(nums []int) []int {  
  
}
```

Kotlin:

```
class Solution {  
    fun getSneakyNumbers(nums: IntArray): IntArray {  
  
    }  
}
```

Swift:

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

Rust:

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

```
}  
}
```

Ruby:

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

PHP:

```
class Solution {  
  
    /**  
     * @param Integer[] $nums  
     * @return Integer[]  
     */  
    function getSneakyNumbers($nums) {  
  
    }  
}
```

Dart:

```
class Solution {  
    List<int> getSneakyNumbers(List<int> nums) {  
  
    }  
}
```

Scala:

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

Elixir:

```

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

  end

  end

```

Erlang:

```

-spec get_sneaky_numbers(Nums :: [integer()]) -> [integer()].
get_sneaky_numbers(Nums) ->
.

```

Racket:

```

(define/contract (get-sneaky-numbers nums)
  (-> (listof exact-integer?) (listof exact-integer?))
  )

```

Solutions

C++ Solution:

```

/*
 * Problem: The Two Sneaky Numbers of Digitville
 * Difficulty: Easy
 * Tags: array, math, 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> getSneakyNumbers(vector<int>& nums) {

    }

};

```

Java Solution:

```

/**
 * Problem: The Two Sneaky Numbers of Digitville
 * Difficulty: Easy
 * Tags: array, math, 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 int[] getSneakyNumbers(int[] nums) {

}

}

```

Python3 Solution:

```

"""
Problem: The Two Sneaky Numbers of Digitville
Difficulty: Easy
Tags: array, math, 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 getSneakyNumbers(self, nums: List[int]) -> List[int]:
# TODO: Implement optimized solution
pass

```

Python Solution:

```

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

```

JavaScript Solution:

```
/**
 * Problem: The Two Sneaky Numbers of Digitville
 * Difficulty: Easy
 * Tags: array, math, 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 getSneakyNumbers = function(nums) {

};
```

TypeScript Solution:

```
/**
 * Problem: The Two Sneaky Numbers of Digitville
 * Difficulty: Easy
 * Tags: array, math, 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 getSneakyNumbers(nums: number[]): number[] {

};
```

C# Solution:

```
/*
 * Problem: The Two Sneaky Numbers of Digitville
 * Difficulty: Easy
 * Tags: array, math, 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 int[] GetSneakyNumbers(int[] nums) {

}

}

```

C Solution:

```

/*
* Problem: The Two Sneaky Numbers of Digitville
* Difficulty: Easy
* Tags: array, math, 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* getSneakyNumbers(int* nums, int numsSize, int* returnSize) {

}

```

Go Solution:

```

// Problem: The Two Sneaky Numbers of Digitville
// Difficulty: Easy
// Tags: array, math, 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 getSneakyNumbers(nums []int) []int {

```

```
}
```

Kotlin Solution:

```
class Solution {  
    fun getSneakyNumbers(nums: IntArray): IntArray {  
  
    }  
}
```

Swift Solution:

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

Rust Solution:

```
// Problem: The Two Sneaky Numbers of Digitville  
// Difficulty: Easy  
// Tags: array, math, 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 get_sneaky_numbers(nums: Vec<i32>) -> Vec<i32> {  
  
    }  
}
```

Ruby Solution:

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

```
end
```

PHP Solution:

```
class Solution {  
  
    /**  
     * @param Integer[] $nums  
     * @return Integer[]  
     */  
    function getSneakyNumbers($nums) {  
  
    }  
}
```

Dart Solution:

```
class Solution {  
    List<int> getSneakyNumbers(List<int> nums) {  
  
    }  
}
```

Scala Solution:

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

Elixir Solution:

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

Erlang Solution:

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

Racket Solution:

```
(define/contract (get-sneaky-numbers nums)  
  (-> (listof exact-integer?) (listof exact-integer?))  
  )
```