# Problem 645: Set Mismatch

## Problem Information

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

## Problem Description

You have a set of integers

s

, which originally contains all the numbers from

1

to

n

. Unfortunately, due to some error, one of the numbers in

s

got duplicated to another number in the set, which results in

repetition of one

number and

loss of another

number.

You are given an integer array

nums

representing the data status of this set after the error.

Find the number that occurs twice and the number that is missing and return

them in the form of an array

.

Example 1:

Input:

nums = [1,2,2,4]

Output:

[2,3]

Example 2:

Input:

nums = [1,1]

Output:

[1,2]

Constraints:

2 <= nums.length <= 10

4

1 <= nums[i] <= 10

## Code Snippets

**C++:**

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

}
};
```

**Java:**

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

}
}
```

**Python3:**

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

**Python:**

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

**JavaScript:**

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

```
var findErrorNums = function(nums) {

};
```

**TypeScript:**

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

};
```

**C#:**

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

}
}
```

**C:**

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

}
```

**Go:**

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

}
```

**Kotlin:**

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

}
}
```

**Swift:**

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

}
}
```

**Rust:**

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

}
}
```

**Ruby:**

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

end
```

**PHP:**

```
class Solution {

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

}
}
```

**Dart:**

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

}
}
```

### Scala:

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

}
}
```

### Elixir:

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

end
end
```

### Erlang:

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

### Racket:

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

## Solutions

### C++ Solution:

```
/*
* Problem: Set Mismatch
* Difficulty: Easy
* 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> findErrorNums(vector<int>& nums) {

}
};
```

### Java Solution:

```
/**
* Problem: Set Mismatch
* Difficulty: Easy
* 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 int[] findErrorNums(int[] nums) {

}
}
```

### Python3 Solution:

```
"""
Problem: Set Mismatch
Difficulty: Easy
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 findErrorNums(self, nums: List[int]) -> List[int]:
# TODO: Implement optimized solution
```

```
pass
```

### Python Solution:

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

### JavaScript Solution:

```
/**
* Problem: Set Mismatch
* Difficulty: Easy
* 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 findErrorNums = function(nums) {

};
```

### TypeScript Solution:

```
/**
* Problem: Set Mismatch
* Difficulty: Easy
* 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 findErrorNums(nums: number[]): number[] {

};
```

### C# Solution:

```
/*
* Problem: Set Mismatch
* Difficulty: Easy
* 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 int[] FindErrorNums(int[] nums) {

}
}
```

### C Solution:

```
/*
* Problem: Set Mismatch
* Difficulty: Easy
* 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* findErrorNums(int* nums, int numsSize, int* returnSize) {
```

```
}
```

### Go Solution:

```
// Problem: Set Mismatch
// Difficulty: Easy
// 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 findErrorNums(nums []int) []int {

}
```

### Kotlin Solution:

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

}
}
```

### Swift Solution:

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

}
}
```

### Rust Solution:

```
// Problem: Set Mismatch
// Difficulty: Easy
// 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_error_nums(nums: Vec<i32>) -> Vec<i32> {

}
}
```

### Ruby Solution:

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

end
```

### PHP Solution:

```
class Solution {

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

}
}
```

### Dart Solution:

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

}
}
```

### Scala Solution:

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

```
}
}
```

### Elixir Solution:

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

end
end
```

### Erlang Solution:

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

### Racket Solution:

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