# 题目

给你一个长度为n的整数数组nums，其中nums的所有整数都在范围[1, n]内，且每个整数出现一次或两次 。请你找出所有出现两次的整数，并以数组形式返回。

你必须设计并实现一个时间复杂度为 O(n) 且仅使用常量额外空间的算法解决此问题。

示例 1：

输入：nums = [4,3,2,7,8,2,3,1]

输出：[2,3]

示例 2：

输入：nums = [1,1,2]

输出：[1]

示例 3：

输入：nums = [1]

输出：[]

提示：

n == nums.length

1 <= n <= 10^5

1 <= nums[i] <= n

nums 中的每个元素出现 一次 或 两次

# 分析

## 方法一：使用正负号作为标记

思路：

为了设计一个时间复杂度为O(n)且仅使用常量额外空间的算法来解决这个问题，我们可以利用数组索引和值之间的对应关系。因为数组中的每个整数都在范围[1, n]内，我们可以将每个整数对应的索引位置的数变为负数（或者加上n，只要保证不会与原数冲突即可），以此标记该整数已经出现过一次。当我们再次遇到这个整数时，由于它对应的索引位置的数已经是负数（或者大于n），我们就可以确定这个整数出现了两次。

说明：这种方法会改变原来数组的值。

代码：

class Solution {

public:

std::vector<int> findDuplicates(std::vector<int>& nums) {

std::vector<int> result; // 用于存储出现两次的整数的结果数组

// 遍历数组中的每个元素

for (int i = 0; i < nums.size(); ++i) {

// 获取当前元素的绝对值，并转换为数组索引（因为数组元素范围是[1, n]，所以减1得到正确的索引）

int index = abs(nums[i]) - 1;

// 如果当前索引处的元素为正数，说明该数只出现了一次，将其变为负数作为标记

if (nums[index] > 0) {

nums[index] = -nums[index];

} else {

// 如果当前索引处的元素已经是负数（或者其他标记），说明该数已经出现过一次，现在是第二次出现

// 将该数添加到结果数组中

result.push\_back(abs(nums[i]));

}

}

// 返回结果数组

return result;

}

};

代码解释：

1、我们首先定义了一个result数组，用于存储最终找到的出现两次的整数。

2、然后，我们遍历输入数组nums中的每个元素。

3、对于每个元素，我们通过计算其绝对值并减去1来得到对应的数组索引。因为数组元素范围是[1, n]，我们需要减1来将其转换为有效的数组索引（从0开始）。

4、接下来，我们检查当前索引处的元素。如果它是正数，说明这个数只出现了一次，我们将该索引处的元素变为负数作为标记。

5、如果当前索引处的元素已经是负数（或者其他标记），则说明这个数已经出现过一次，现在是第二次出现。我们将这个数（取绝对值，因为可能已经被标记为负数）添加到结果数组中。

6、最后，我们返回结果数组，它包含了所有出现两次的整数。

请注意，这个算法假设输入的数组nums中的每个元素都在范围[1, n]内，并且每个整数只出现一次或两次。此外，算法使用了原数组nums进行标记，因此不需要额外的空间（除了结果数组外），满足了题目要求的常量额外空间。然而，算法修改了原数组，这在实际应用中可能不是期望的行为，但在题目要求的范围内是允许的。

## 方法二：将元素交换到对应的位置