Find All Duplicates in an Array LeetCode

Given an integer array nums of length n where all the integers of nums are in the range [1, n] and each integer appears once or 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 extra space.

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

Output: [2,3]

Input: nums = [1,1,2]

Output: [1]

Input: nums = [1]

Output: []

Note:

You should not modify the original array.

The order of elements in the output vector does not matter.

Your solution should have a time complexity of O(n) and a space complexity of O(1) (excluding the output vector).

Approach 1: Using Nested Loop to compare array elements with each other.

iterates through the input array nums using two nested loops. It compares each element with the remaining elements to count the occurrences of each element. If an element occurs exactly twice, it is considered a duplicate and added to the ans vector.

Time Complexity: The outer loop runs n times, where n is the size of the input array. The inner loop runs n - i - 1 times for each outer loop iteration. Overall, this approach has a **time complexity of O(n^2), where n is the size of the input array.**

Space Complexity: The ans vector stores the duplicate elements, **which can have a maximum size of n/2 (assuming every element appears twice). Therefore, the space complexity is O(n).**

Approach 2: Using Hash map to keep track of the count of the array elements.

uses an unordered map, count, to store the frequency of each element in the input array nums. It then iterates through the map and adds the elements with a frequency of 2 to the ans vector.

Time Complexity: The first loop that counts the frequency of each element in the input array takes O(n) time, where n is the size of the input array. The second loop iterates through the map, which contains at most n/2 elements (assuming every element appears twice), **resulting in a time complexity of O(n).**

Space Complexity: **The space complexity is O(n) because the unordered map, count**, can store at most n/2 elements (assuming every element appears twice), and the ans vector can also have a maximum size of n/2.

Approach 3: Using "Negation Marking" or the "Index as a Hash" approach.

Note: it comes with the constraint that the elements in the array must be within a specific range and allows only non-negative values. If the range or the non-negative constraint is not guaranteed, this approach may not be applicable.

The optimized approach takes advantage of the fact that the elements in the array are in the range of 1 to n, where n is the size of the array. It modifies the array itself to keep track of the duplicates.

Consider the example array nums = {4, 3, 2, 7, 8, 2, 3, 1}.

Step 1:

Initialize an empty vector to store the duplicate elements: duplicates = {}.

Step 2:

Iterate through each element in the array:

For the first element, num = 4, the corresponding index is abs(4) - 1 = 3. Since the value at index 3 is positive (nums[3] = 7), we multiply it by -1: nums[3] = -7.

```
nums = {4, 3, 2, -7, 8, 2, 3, 1}
```

For the second element, num = 3, the corresponding index is abs(3) - 1 = 2. Since the value at index 2 is positive (nums[2] = 2), we multiply it by -1: nums[2] = -2.

```
nums = {4, 3, -2, -7, 8, 2, 3, 1}
```

For the third element, num = -2, the corresponding index is abs(-2) - 1 = 1. Since the value at index 1 is positive (nums[1] = 3), we multiply it by -1: nums[1] = -3.

```
nums = {4, -3, -2, -7, 8, 2, 3, 1}
```

Continuing this process, we encounter the element 7, which has a corresponding index of 6. The value at index 6 is positive (nums[6] = 3), so we multiply it by -1: nums[6] = -3.

```
nums = {4, -3, -2, -7, 8, 2, -3, 1}
```

The element 8 corresponds to index 7, which is positive. We multiply it by -1: nums[7] = -1.

```
nums = {4, -3, -2, -7, 8, 2, -3, -1}
```

For the element 2, the corresponding index is 1. The value at index 1 is already negative (nums[1] = -3), indicating that 2 is a duplicate. So we add its absolute value to the duplicates vector: duplicates = {2}.

Finally, for the element 3, the corresponding index is 2. The value at index 2 is already negative (nums[2] = -2), indicating that 3 is a duplicate. So we add its absolute value to the duplicates vector: duplicates = {2, 3}.

```
duplicates = {2, 3}
```

This problem with a time complexity of O(n) and constant space complexity (O(1)).