✅ Problem Summary LeetCode:268

We are given an array `nums` of length `n` containing **distinct numbers** in the range `[0, n]`. Our task is to **find the missing number** in this range.

Example:

`nums = [3,0,1]`

`n = 3` (length of array)

Range → `[0, 3]`

Missing number → `2`

✅ Your Code (Method 1)

```
class Solution {
public:
    int missingNumber(vector<int>& nums) {
        // method 1 using extra space actually O(n)
        int n = nums.size();
        vector<bool> flag_vector(n+1, false);

        // Step 1: Mark presence
        for (int i = 0; i < n; i++) {
            flag_vector[nums[i]] = true;
        }

        // Step 2: Find the missing number
        for (int i = 0; i <= n; i++) {
            if (flag_vector[i] == false) return i;
        }

        return 100; // fallback (not really needed)
    }
};
```

✅ Step-by-Step Explanation

1. Idea Behind the Code

- We know that numbers should be in the range `[0, n]`.
- There are `n+1` possible numbers, but we have only `n` numbers in the array → so exactly **1 number is missing**.
- To find which one is missing, we mark which numbers exist.

2. Code Logic

Step 1: Create a flag array

```
vector<bool> flag_vector(n+1, false);
```

- Size = `n+1` because numbers range from `0` to `n` inclusive.
- Initialize all positions as `false` (means "not present").

Step 2: Mark the present numbers

```
for (int i = 0; i < n; i++) {
    flag_vector[nums[i]] = true;
}
```

- For every element in `nums`, mark its index in `flag_vector` as `true`.

Step 3: Find the missing number

```
for (int i = 0; i <= n; i++) {
    if (flag_vector[i] == false) return i;
}
```

- The first index that is still `false` is the missing number.

3. Time & Space Complexity

- **Time:**
  - O(n) for marking
  - O(n) for scanning

    → Total = **O(n)**
- **Space:**
  - Extra array of size `n+1` → **O(n)**

✅ Example Dry Run

Input: `nums = [3,0,1]`

`n = 3`

`flag_vector = [false, false, false, false]`

- Marking step:

  `nums[0]=3 → flag_vector[3]=true`

  `nums[1]=0 → flag_vector[0]=true`

  `nums[2]=1 → flag_vector[1]=true`

`flag_vector = [true, true, false, true]`

Index `2` is false → Missing number = `2`

✅ Why `return 100`?

This is just a **fallback return**, but logically **it will never execute** because one number is always missing as per the problem statement.

✅ Follow-Up (O(1) Space, O(n) Time)

The problem asks for a solution with **constant space**. Two better approaches:

Approach 1: Sum Formula

- Sum of numbers from `0` to `n` = `n*(n+1)/2`
- Compute actual sum of array → Missing number = `expectedSum - actualSum`

Code:

```
int missingNumber(vector<int>& nums) {
    int n = nums.size();
    int expectedSum = n * (n + 1) / 2;
    int actualSum = 0;
    for (int num : nums) actualSum += num;
    return expectedSum - actualSum;
}
```

Approach 2: XOR Trick

- XOR of all numbers `0` to `n` and all numbers in array.
- The duplicate parts cancel out, leaving the missing number.

Code:

```
int missingNumber(vector<int>& nums) {
    int n = nums.size();
    int xorAll = 0;
    for (int i = 0; i <= n; i++) xorAll ^= i;
    for (int num : nums) xorAll ^= num;
    return xorAll;
}
```

✅ Key Points for Interview

- Your method is correct but uses **O(n) extra space**.
- Best solution: **O(n) time, O(1) space** → Sum method or XOR method.
- Both handle large `n` easily and no extra memory overhead.