LeetCode: 448

✅ Problem Statement (Simplified Explanation)

We are given an **array `nums` of size `n`**, where:

- Each element in `nums` is in the range `[1, n]`.
- Some numbers in `[1, n]` might be missing, and some may appear twice.

Goal:

Find **all the numbers in `[1, n]` that do not appear in `nums`**.

Example

Input:

`nums = [4,3,2,7,8,2,3,1]`

Here, the array size is `n = 8`, so the range is `[1, 8]`.

Numbers present: `1,2,3,4,7,8`

Missing numbers: `5, 6`

Output: `[5,6]`

Constraints

- `1 <= n <= 10^5` → **O(n)** algorithm required.
- **Follow-up:** Solve without using extra space (returned list does not count as extra space).

✅ Logic Behind the Solution

We need an **O(n)** time and **O(1)** extra space approach (ignoring the result array).

Key Observation:

Each number is in the range `[1, n]`.

So ideally, if the array was perfectly arranged:

```
nums[i] = i + 1
Example: [1, 2, 3, 4, 5, 6, ...]
```

But due to duplicates and missing numbers, this won't hold true.

Approach

Use **Cyclic Sort**:

- Place each number at its correct index (`nums[i]` should go to `index = nums[i] - 1`).
- After rearrangement, the numbers that are not in the correct position indicate missing values.

✅ Algorithm Steps

1. Start from index `i = 0`.
2. While `i < n`:
   - Compute `correct_index = nums[i] - 1`.
   - If `nums[i] != nums[correct_index]` → Swap them.
   - Else → `i++` (move to next index).
3. After rearranging:
   - Traverse array from `0` to `n-1`.
   - If `nums[i] != i + 1`, then `(i + 1)` is missing → Add to result.
4. Return the result.

✅ Dry Run (Example: [4,3,2,7,8,2,3,1])

Initial array: `[4,3,2,7,8,2,3,1]`

Step 1: Cyclic sort

- `i=0`:

  `nums[0] = 4 → correct_index = 3`

  Swap(nums[0], nums[3]) → `[7,3,2,4,8,2,3,1]`

- `i=0`:

  `nums[0] = 7 → correct_index = 6`

  Swap → `[3,3,2,4,8,2,7,1]`

- `i=0`:

  `nums[0] = 3 → correct_index = 2`

  Swap → `[2,3,3,4,8,2,7,1]`

- `i=0`:

  `nums[0] = 2 → correct_index = 1`

  Swap → `[3,2,3,4,8,2,7,1]`

- `i=0`:

  `nums[0] = 3 → correct_index = 2`

  Already correct? No (duplicate), so `i++`.

Continue similarly... After full pass, array becomes:

`[1,2,3,4,3,2,7,8]`

Step 2: Find Missing

Now compare:

```
Index:  0 1 2 3 4 5 6 7
Value:  1 2 3 4 3 2 7 8
```

- At index 4: value = 3, expected 5 → Missing 5
- At index 5: value = 2, expected 6 → Missing 6

Result: `[5,6]`

✅ Time & Space Complexity

- **Time:**

- Rearrangement: O(n)
- Final scan: O(n)

  Total = **O(n)**

- **Space:**
  - In-place swaps, only result vector is extra

    O(1) (ignoring output)

✅ Key Points

✔ Cyclic Sort is ideal when numbers are in a known range `[1, n]`.

✔ We avoid using extra hash set or boolean array.

✔ Perfect for "missing number(s)" problems.
