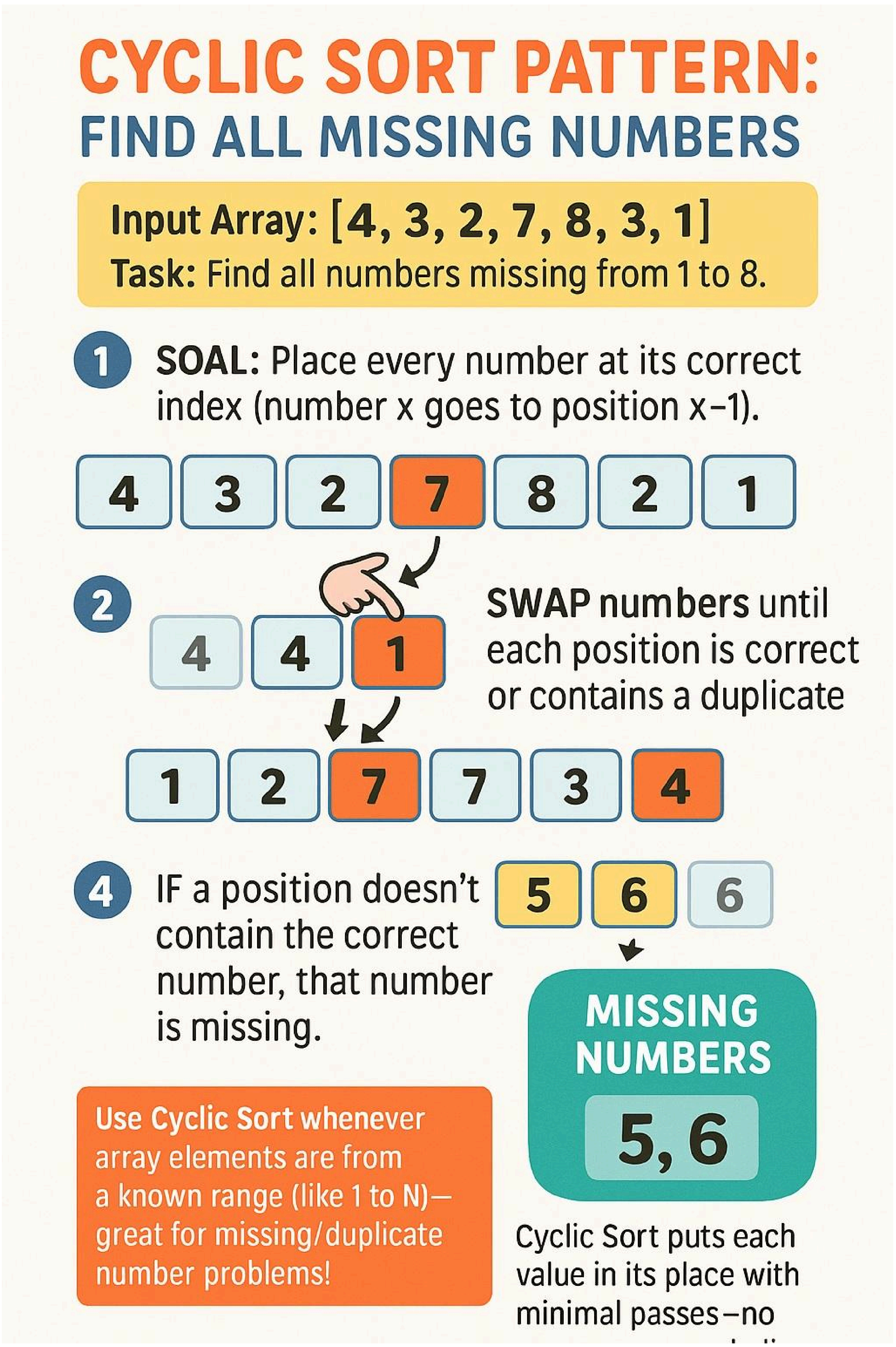
Cyclic Sort Pattern

Cyclic Sort is a clever technique for solving problems where:

- You have an array of size `n` with numbers in a *known range* (usually `1` to `n` or `0` to `n-1`).
- The goal is often to find missing or duplicate numbers, or to put every number in its correct index with minimal space and time.

Key Idea:
Place each number at its correct index by repeatedly swapping.



CYCLIC SORT PATTERN:
FIND ALL MISSING NUMBERS

Input Array: [4, 3, 2, 7, 8, 3, 1]
Task: Find all numbers missing from 1 to 8.

1 **SOAL:** Place every number at its correct index (number x goes to position x−1).

2 **SWAP numbers** until each position is correct or contains a duplicate

4 IF a position doesn't contain the correct number, that number is missing.

MISSING NUMBERS
5, 6

Use Cyclic Sort whenever array elements are from a known range (like 1 to N)—great for missing/duplicate number problems!

Cyclic Sort puts each value in its place with minimal passes—no

Classic Problem: Find Missing Number

Problem Statement:
Given an array containing `n` numbers taken from the range `1` to `n+1`, with one number missing, find the missing number.

But let's do the most classic Cyclic Sort example:

Example: Find All Missing Numbers

Problem:
Given an unsorted array of numbers taken from `1` to `n`, some numbers may be missing and some may be duplicated. Find all the missing numbers.

Input:
`arr[] = {4, 3, 2, 7, 8, 2, 3, 1}`
Output:
`[5, 6]` (since 5 and 6 are missing)

Explanation

1. **Try to place every number at its correct index.**
   - For number `x`, its correct index is `x-1`.
   - Swap numbers until each position has the right value or a duplicate.
2. **After this pass, if a position i doesn't have `i+1`, that's a missing number!**

C Code Example: Find All Missing Numbers

```
#include <stdio.h>

// Swap helper
void swap(int* a, int* b) {
    int t = *a;
    *a = *b;
    *b = t;
}

// Main function
void findMissingNumbers(int arr[], int n) {
    // Cyclic sort: place each number at its correct position if possible
    for (int i = 0; i < n; ) {
        int correctIdx = arr[i] - 1;
        // If arr[i] is not at correct position and not a duplicate
        if (arr[i] >= 1 && arr[i] <= n && arr[i] != arr[correctIdx]) {
            swap(&arr[i], &arr[correctIdx]);
        } else {
            i++;
        }
    }

    // After cyclic sort, numbers not in correct position mean missing numbers
    printf("Missing numbers: ");
    for (int i = 0; i < n; i++) {
        if (arr[i] != i + 1) {
            printf("%d ", i + 1);
        }
    }
    printf("\n");
}

int main() {
    int arr[] = {4, 3
```

Explanation

1. **Why swap?**
   You want number 1 at index 0, number 2 at index 1, ..., number n at index n-1.
2. **What happens in the loop?**
   - For each position, if the number is not at its correct spot and not a duplicate, swap it into the right place.
   - If it is already correct or a duplicate, just move to the next.

3. **How do we find missing numbers?**
    - After arranging, scan the array.
    - Any position where `arr[i] != i+1`, means `i+1` is missing.
4. **In our example:**
    After cyclic sort: `[1,2,3,4,3,2,7,8]`
    - At index 4, value is 3, so 5 is missing.
    - At index 5, value is 2, so 6 is missing.

When to Use Cyclic Sort?

- When elements are from a known, tight range (like 1 to n).
- When asked for missing or duplicate numbers.

Practice Challenge

Try modifying the code to **find duplicate numbers** in the same array.