Bitwise XOR Pattern

The **Bitwise XOR (^)** pattern leverages properties of the XOR operator to solve problems involving:

- Finding unique/non-repeated numbers in an array
- Swapping values without a temp variable
- Bit manipulation tricks

Key XOR properties:

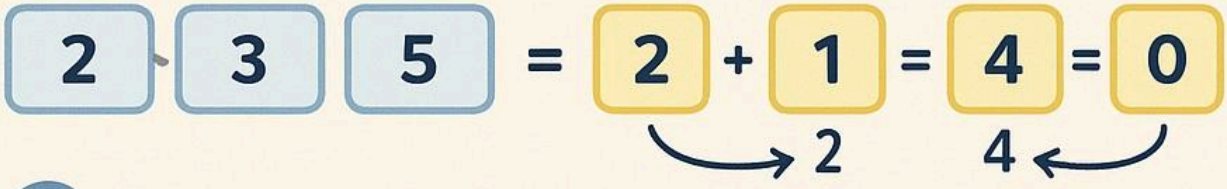
- `a ^ a = 0` (a number XOR itself is zero)
- `a ^ 0 = a` (a number XOR zero is the number)
- XOR is **commutative** (`a ^ b = b ^ a`)
- XOR is **associative** (`a ^ (b ^ c) = (a ^ b) ^ c`)

Bitwise XOR Pattern:
Find the Single Number

Every number appears twice except one. Find the number that appears only once.

1 Start with result = 0

2 XOR each element with result so far



3 Pairs cancel out → Unpaired number remains

Single Number = 2

XOR cancels duplicates:
$a \wedge a = 0$, $a \wedge 0 = a$
Use this trick for finding non-repeated numbers,swaps, and more!

Bitwise XOR is a powerful tool for detecting unique

Bitwise XOR is a powerful tool for detecting unique numbers and doing clever bit manipulation!

Classic Problem 1: Find the Single Number

Problem Statement:
In an array where every number appears **twice** except one number that appears **once**, find that single number.

Example:
`arr[] = {2, 3, 5, 4, 5, 3, 4}`
Output: `2`

Explanation

1. XOR all the numbers in the array together.
2. All pairs will cancel out (because `x ^ x = 0`).
3. Only the unique number remains.

C Code Example: Find the Single Number

```
#include <stdio.h>

int findSingleNumber(int arr[], int n) {
    int result = 0;
    for (int i = 0; i < n; i++) {
        result ^= arr[i];
    }
    return result;
}

int main() {
    int arr[] = {2, 3, 5, 4, 5, 3, 4};
    int n = sizeof(arr) / sizeof(arr[0]);
    int single = findSingleNumber(arr, n);
    printf("The single number is: %d\n", single);
    return 0;
}
```

How does it work?

- XOR all values:
  `2 ^ 3 ^ 5 ^ 4 ^ 5 ^ 3 ^ 4`
- Every pair cancels out (`3^3=0`, `4^4=0`, `5^5=0`), so only `2` remains.

Classic Problem 2: Find Two Non-Repeating Numbers

Problem:
In an array where every number appears twice except for two numbers that appear only once, find both numbers.

Example:
`arr[] = {2, 4, 7, 9, 2, 4}`
Output: `7, 9` (order may vary)

C Code Example: Find Two Unique Numbers

```
#include <stdio.h>

void findTwoUniqueNumbers(int arr[], int n, int* num1, int* num2) {
    int xor_all = 0;
    for (int i = 0; i < n; i++)
        xor_all ^= arr[i];

    // Get rightmost set bit (differs between the two unique numbers)
    int rightmost_set_bit = xor_all & (~(xor_all - 1));

    *num1 = 0;
    *num2 = 0;

    for (int i = 0; i < n; i++) {
        if (arr[i] & rightmost_set_bit)
            *num1 ^= arr[i];
        else
            *num2 ^= arr[i];
    }
}

int main() {
    int arr[] = {2, 4, 7, 9, 2, 4};
    int n = sizeof(arr) / sizeof(arr[0]);
    int num1, num2;
    findTwoUniqueNumbers(arr, n, &num1, &num2);
    printf("The two unique numbers are: %d and %d\n", num1, num2);
    return 0;
}
```

How does it work?

- XOR of all gives `7 ^ 9` (call this X).
- Find a bit where these two differ (rightmost set bit).
- Divide numbers into two groups based on that bit, and XOR within each group to find the two numbers.

Bitwise XOR Pattern is Useful For:

- **Finding single or multiple unique numbers**
- **Swapping variables without a temp:** `a ^= b; b ^= a; a ^= b;`
- **Low-level bit manipulation tasks**

Practice Challenge

- Try writing a function that finds the missing number in an array containing all numbers from `1` to `n` except one (using XOR).
- Try using XOR to swap two variables in C **without a third variable**.