One repeating one missing in C++

```
#include <iostream>
#include <vector>
using namespace std;

void solution(vector<int>& arr) {
    int xor_val = 0;
    int n = arr.size();

    // XOR all elements in arr and numbers from 1 to n
    for (int i = 0; i < n; i++) {
        xor_val ^= arr[i];
        xor_val ^= (i + 1);
    }

    // Find the rightmost set bit
    int rsb = xor_val & -xor_val;

    int x = 0, y = 0;

    // Divide elements into two groups based on rsb
    for (int i = 0; i < n; i++) {
        if (arr[i] & rsb)
            x ^= arr[i];
        else
            y ^= arr[i];

        if ((i + 1) & rsb)
            x ^= (i + 1);
        else
            y ^= (i + 1);
    }

    // Check which one is repeating and which one is
missing
    for (int i = 0; i < n; i++) {
        if (arr[i] == x) {
            cout << "Missing Number -> " << y << endl;
            cout << "Repeating Number -> " << x << endl;
            break;
        } else if (arr[i] == y) {
            cout << "Missing Number -> " << x << endl;
            cout << "Repeating Number -> " << y << endl;
            break;
        }
    }
}

int main() {
    vector<int> arr = {1, 3, 4, 4, 5, 6, 7};
    solution(arr);
    return 0;
}
```

Input:

arr = {1, 3, 4, 4, 5, 6, 7}

- n = 7 → array should contain 1 to 7
- But here:
    - 4 is repeated
    - 2 is missing

Step 1: XOR all elements and numbers from 1 to n

i	arr[i]	i+1	xor_val (after arr[i])	xor_val (after i+1)
0	1	1	0 ^ 1 = 1	1 ^ 1 = 0
1	3	2	0 ^ 3 = 3	3 ^ 2 = 1
2	4	3	1 ^ 4 = 5	5 ^ 3 = 6
3	4	4	6 ^ 4 = 2	2 ^ 4 = 6
4	5	5	6 ^ 5 = 3	3 ^ 5 = 6
5	6	6	6 ^ 6 = 0	0 ^ 6 = 6
6	7	7	6 ^ 7 = 1	1 ^ 7 = 6

➞ Final xor_val = 6
Which is missing ^ repeating = 2 ^ 4 = 6

Step 2: Find rightmost set bit in xor_val

rsb = xor_val & -xor_val = 6 & -6 = 2 (binary: 10)

So we now divide numbers into **two groups** based on this bit.

Step 3: XOR within two groups

Let's categorize by whether (number & rsb) == 0 or != 0

For arr and 1 to n

Element	Binary	Group (rsb)
1	0001	y
2	0010	x
3	0011	x
4	0100	y
5	0101	y
6	0110	x
7	0111	x

Perform XOR within groups

- Group X (bit set): 2, 3, 6, 3, 6, 7 → x = 2 ^ 3

	^ 6 ^ 3 ^ 6 ^ 7 = 2 • Group Y (bit not set): 1, 4, 4, 1, 5, 7, 5 → y = 1 ^ 4 ^ 4 ^ 1 ^ 5 ^ 5 = 4 **Step 4: Determine which is missing and which is repeating** Check if x = 2 is present in arr → ✗ Not found → So x = 2 is **missing** y = 4 is found → ✓ → y = 4 is **repeating** ✓ **Final Output:** Missing Number -> 2 Repeating Number -> 4
Missing Number -> 2 Repeating Number -> 4	