**B. AGAGA XOOORRR -** [**https://codeforces.com/contest/1516/problem/B**](https://codeforces.com/contest/1516/problem/B)

**1. Problem Statement (in simple words)**

You have:

* An array a of length n.
* You can **pick two adjacent elements**, replace them with their **bitwise XOR**, which reduces the array size by 1.
* You must **stop** when at least **2 elements remain**.
* Goal: Check if you can make **all remaining elements equal**.

**Output:** "YES" if possible, otherwise "NO".

**Example**

**Example 1:**

n = 3

a = [0, 2, 2]

Step:

Pick (0, 2) → XOR = 2 → array becomes [2, 2]

All equal → YES

**Example 2:**

n = 4

a = [2, 3, 1, 10]

No matter how you XOR adjacent pairs, you can’t end with ≥2 elements all equal → NO

**2. How to Think About It**

**Key observation about XOR**

* XOR is **associative** and **commutative**:
* a ^ b ^ c = c ^ a ^ b
* If all remaining elements must be equal to some value X, then:
  + Every segment we create must XOR to the same value.

**Case 1:**

If totalXor = a[0] ^ a[1] ^ ... ^ a[n-1] is **0** →  
We can always split the array into at least 2 equal parts (since 0 means we can balance segments easily). → Answer is **YES**.

**Case 2:**

If totalXor ≠ 0 →  
We can still succeed if we can split the array into **at least 3 parts** such that:

XOR of each part = totalXor

Why 3 parts?

* If you split into 3 parts, you can merge first part to 1 element, merge second part to 1 element, and the third will also be same because XOR matches.

**3. Solution Approach**

1. Compute totalXor of the array.
2. If totalXor == 0 → print "YES".
3. Else:
   * Traverse the array, keep cumulative XOR.
   * Whenever cumulative XOR equals totalXor, reset cumulative XOR and increase segment count.
   * If count >= 2 by the end, print "YES".
   * Else, "NO".

**4. C++ Implementation**

#include <bits/stdc++.h>

using namespace std;

bool canMakeEqual(vector<int>& arr) {

int totalXor = 0;

for (int num : arr) {

totalXor ^= num;

}

if (totalXor == 0) return true;

int currentXor = 0, count = 0;

for (int num : arr) {

currentXor ^= num;

if (currentXor == totalXor) {

count++;

currentXor = 0;

}

}

return count >= 2; // need at least 3 segments → 2 cuts

}

int main() {

int t;

cin >> t;

while (t--) {

int n;

cin >> n;

vector<int> arr(n);

for (int i = 0; i < n; i++) {

cin >> arr[i];

}

cout << (canMakeEqual(arr) ? "YES" : "NO") << "\n";

}

return 0;

}

**5. Complexity Analysis**

* **Time Complexity:**
  + Computing totalXor: **O(n)**
  + Traversal to find segments: **O(n)**
  + Overall per test case: **O(n)**  
    With n ≤ 2000 and t ≤ 15 → safe.
* **Space Complexity:**
  + Only storing the array: **O(n)**

**Another solution :**

#include <bits/stdc++.h>

using namespace std;

int main()

{

    int t;

    cin >> t;

    while (t--)

    {

        int n;

        cin >> n;

        int ar[n + 3];

        for (int i = 1; i <= n; i++)

        {

            cin >> ar[i];

        }

        int pre[n + 3];

        pre[0] = 0;

        for (int i = 1; i <= n; i++)

        {

            pre[i] = pre[i - 1] ^ ar[i];

        }

        int ans = 0;

        // p 2

        for (int i = 1; i < n; i++)

        {

            int a = pre[i];

            int b = pre[n] ^ pre[i];

            if (a == b)

            {

                ans = 1;

                break;

            }

        }

        // p - 3

        for (int i = 1; i < n - 1; i++)

        {

            int a = pre[i];

            for (int j = i + 1; j < n; j++)

            {

                int b = pre[j] ^ pre[i];

                int c = pre[n] ^ pre[j];

                if (a == b && b == c)

                {

                    ans = 1;

                    break;

                }

            }

        }

        if (ans == 1)

            cout << "YES" << endl;

        else

            cout << "NO" << endl;

    }

    return 0;

}