**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;

}

**B. Morning Jogging -** [**https://codeforces.com/contest/1517/problem/B**](https://codeforces.com/contest/1517/problem/B)

#include<bits/stdc++.h>

using  namespace  std;

int main()

{

    int t;

    cin>>t;

    while(t--)

    {

        int n,m,x;cin>>n>>m;

        vector<int>v[n+4];

        vector<int>ans[n+3];

        for(int i=0;i<n;i++)

        {

            for(int j=0;j<m;j++)

            {

                cin>>x;

                v[i].push\_back(x);

            }

        }

        for(int i=0;i<n;i++)

        {

            sort(v[i].begin(),v[i].end());

        }

        for(int i=0;i<m;i++)

        {

            int mn=INT\_MAX;

            int ind=-1;

            for(int j=0;j<n;j++)

            {

                if(v[j][0]<mn)

                {

                    mn=v[j][0];

                    ind=j;

                }

            }

            for(int j=0;j<n;j++)

            {

                if(j==ind)

                {

                    ans[j].push\_back(v[j][0]);

                    v[j].erase(v[j].begin());

                }

                else

                {

                    ans[j].push\_back(v[j][v[j].size()-1]);

                    v[j].pop\_back();

                }

            }

        }

        for(int i=0;i<n;i++)

        {

           for(int j=0;j<ans[i].size();j++)cout<<ans[i][j]<<" ";

           cout<<endl;

        }

    }

    return 0;

}

**B. TMT Document -** [**https://codeforces.com/contest/1509/problem/B**](https://codeforces.com/contest/1509/problem/B)

**🔹 Problem Restatement**

We are given a string consisting of only **T** and **M** characters. We want to check if we can partition the string into subsequences (not substrings) such that **each subsequence is exactly "TMT"**.

**Key Points:**

* Each character in the string must belong to exactly one "TMT".
* Subsequence means: we can skip characters but must keep relative order.
* String length n is always divisible by 3.
* We need to answer YES or NO for each test case.

**🔹 How to Think**

1. Each "TMT" has **2 T’s** and **1 M**.  
   ✅ So:
   * Count of T = 2 \* count(M)  
     Otherwise → immediately **NO**.
2. Order must be valid:
   * For every "M", there must be a "T" **before it** (because "TMT" starts with a T).
   * For every "M", there must also be a "T" **after it** (because "TMT" ends with a T).

That means:

* + Left-to-right check: When scanning, at any point, count(M) ≤ count(T) (otherwise we would get an M without a left T).
  + Right-to-left check: When scanning backward, same rule: count(M) ≤ count(T) (otherwise M has no right T).

**🔹 Solution Approach**

1. Check frequency condition: count(T) == 2 \* count(M).  
   If not, → NO.
2. Left-to-right pass: ensure every M has a T before it.
3. Right-to-left pass: ensure every M has a T after it.
4. If all conditions satisfied → YES.

**Code :**

#include <bits/stdc++.h>

using namespace std;

#define ll long long

bool solve(string s)

{

    int m = 0, t = 0;

    for (int i = 0; i < s.size(); i++)

    {

        if (s[i] == 'M')

            m++;

        else

            t++;

        if (m > t)

            return false;

    }

    return true;

}

int main()

{

    int t;

    cin >> t;

    while (t--)

    {

        int n;cin >> n;

        string s;

        cin >> s;

        int M = 0, T = 0;

        for (int i = 0; i < s.size(); i++)

        {

            if (s[i] == 'T')

                T++;

            else

                M++;

        }

        if (2 \* M != T)

            cout << "NO" << endl;

        else

        {

            if (solve(s) == true)

            {

                reverse(s.begin(), s.end());

                if (solve(s) == true)

                    cout << "YES" << endl;

                else

                    cout << "NO" << endl;

            }

            else

                cout << "NO" << endl;

        }

    }

    return 0;

}

**🔹 Complexity Analysis**

* Counting T and M → **O(n)**.
* Left-to-right check → **O(n)**.
* Right-to-left check → **O(n)**.
* Total per test case: **O(n)**.
* Since sum of n ≤ 10^5, the solution is efficient.

**B. The Cake Is a Lie -** [**https://codeforces.com/contest/1519/problem/B**](https://codeforces.com/contest/1519/problem/B)

#include <bits/stdc++.h>

using namespace std;

int main()

{

    int t;

    cin >> t;

    while (t--)

    {

        int n, m, k;

        cin >> n >> m >> k;

        int result = (n - 1) + (m - 1) \* n;

        if (result != k)

            cout << "NO" << endl;

        else

            cout << "YES" << endl;

    }

    return 0;

}

**B. Phoenix and Puzzle -** [**https://codeforces.com/contest/1515/problem/B**](https://codeforces.com/contest/1515/problem/B)

**🔹 Problem Statement (in my words)**

You are given n puzzle pieces, where each piece is a **right isosceles triangle** (two equal sides, one right angle).  
You must decide if it is possible to arrange **all n pieces** to form a **perfect square** (without gaps or overlaps).

For each test case, print **YES** if possible, otherwise **NO**.

**🔹 How to Think**

1. **Understand the piece**:
   * Each piece is a **right isosceles triangle**.
   * Two such triangles can form a **square of side length 1** (like joining them along the hypotenuse).
   * So, **n must be even** (otherwise, one triangle will always be left unused).

✅ First condition: n % 2 == 0

1. **Can every even n work?**  
   Let’s test:
   * n = 2 → Yes (forms a 1x1 square).
   * n = 4 → Yes (forms a √2 x √2 square).
   * n = 6 → ❌ Not possible.

So not all even n work.

1. **Key Insight**
   * If you take **2 triangles**, they form a square of area 1.
   * So, n triangles can form an area = n/2.
   * For this to be arranged as a **big square**, that area (n/2) must itself be a **perfect square**.

✅ Condition: (n / 2) should be a perfect square.

Example:

* + n = 2 → n/2 = 1 → perfect square → YES
  + n = 4 → n/2 = 2 → not square ❌ … but wait example says YES. Why?

1. **Careful Observation**
   * Another way: sometimes 4 triangles form a 1x1 square (not 2 triangles).
   * Actually, two possibilities exist:
     + Case 1: n / 2 is a perfect square.
     + Case 2: n / 4 is a perfect square.

Let’s check again:

* + n = 2 → n/2 = 1 (perfect square) → YES
  + n = 4 → n/4 = 1 (perfect square) → YES
  + n = 6 → n/2 = 3 (not square), n/4 = 1.5 (not integer) → NO ✅
  + n = 8 → n/2 = 4 (perfect square) → YES

**🔹 Final Solution Approach**

1. If n is **odd** → NO
2. Else check:
   * If (n / 2) is a perfect square → YES
   * Or (n / 4) is a perfect square → YES
   * Otherwise → NO

**🔹 C++ Implementation**

#include <bits/stdc++.h>

using namespace std;

bool isPerfectSquare(long long x) {

long long r = sqrt(x);

return r \* r == x;

}

int main() {

ios::sync\_with\_stdio(false);

cin.tie(nullptr);

int t;

cin >> t;

while (t--) {

long long n;

cin >> n;

if (n % 2 == 0 && isPerfectSquare(n / 2)) {

cout << "YES\n";

}

else if (n % 4 == 0 && isPerfectSquare(n / 4)) {

cout << "YES\n";

}

else {

cout << "NO\n";

}

}

return 0;

}

**🔹 Complexity**

* **Square root check**: O(1)
* For each test case → O(1)
* For t test cases → O(t)

Memory usage is minimal (O(1)).

**Another Approach :**

**Code :**

#include<bits/stdc++.h>

using  namespace  std;

#define ll long long

int main()

{

    map<ll,ll>mp;

    ll ans=4;

    mp[2]=1;

    mp[4]=1;

    ll i=1;

    while(ans<=1e9)

    {

        ll an=i+i+1;

        an\*=4;

        ans+=an;

        mp[ans]=1;

        i++;

    }

    ans=2;

    i=1;

    while(ans<=1e9)

    {

        ll an=i+i+1;

        an\*=2;

        ans+=an;

        mp[ans]=1;

        i++;

    }

    int t;

    cin>>t;

    while(t--)

    {

        int n;

        cin>>n;

        if(mp[n]==1)cout<<"YES"<<endl;

        else cout<<"NO"<<endl;

    }

}

**B. Ordinary Numbers -** [**https://codeforces.com/contest/1520/problem/B**](https://codeforces.com/contest/1520/problem/B)

**🔎 Problem Understanding**

We are asked to count **ordinary numbers** between 1 and n.

👉 **Ordinary number**: A number where **all digits are the same**.  
Examples:

* ✅ 1, 2, 3, ..., 9, 11, 22, 33, ..., 99, 111, 222, ... are ordinary.
* ❌ 10, 12, 101, 2021 are not.

We are given t test cases.  
For each n, we need to output how many ordinary numbers exist between 1 and n.

**🔎 Example Walkthrough**

Input:

6

1

2

3

4

5

100

Output:

1

2

3

4

5

18

* For n = 1: Only {1} → **1**.
* For n = 2: {1, 2} → **2**.
* For n = 5: {1, 2, 3, 4, 5} → **5**.
* For n = 100:  
  Ordinary numbers are:
  + 1-digit: 1..9 (9 numbers).
  + 2-digit: 11, 22, 33, ..., 99 (9 numbers).
  + 3-digit ordinary numbers ≤ 100: Only 111 is > 100, so we stop.  
    ✅ Total = 9 + 9 = **18**.

**💡 How to Think About It**

1. Any **ordinary number** is of the form:  
   d, dd, ddd, dddd, ... where d ∈ [1..9].
   * For example: digit 7 generates → 7, 77, 777, 7777, ...
2. To count ordinary numbers ≤ n:
   * Check how many "digit-repeated" numbers exist that are ≤ n.
3. Key insight:
   * Count of ordinary numbers depends on:
     + The number of digits in n.
     + The first digit of n.

**✅ Solution Approach**

1. Convert n into string (to get length and first digit).
2. If n has k digits:
   * All ordinary numbers with **fewer digits** (1..k-1) are automatically valid.  
     → That gives (k-1) \* 9 ordinary numbers.  
     (For each digit 1..9, we can form exactly 1 ordinary number with len < k.)
   * For length = k, check how many exist:
     + Form a number like ddd...d (length k) for each digit 1..first\_digit.
     + Count only those ≤ n.

**🖥️ C++ Implementation**

#include <bits/stdc++.h>

using namespace std;

int main() {

ios::sync\_with\_stdio(false);

cin.tie(nullptr);

int t;

cin >> t;

while (t--) {

long long n;

cin >> n;

string s = to\_string(n);

int len = s.size(); // number of digits

int first\_digit = s[0] - '0';

// (len - 1) \* 9 covers all ordinary numbers with fewer digits

int ans = (len - 1) \* 9;

// Build the number "ddd...d" with same length as n

long long candidate = 0;

for (int i = 0; i < len; i++) candidate = candidate \* 10 + first\_digit;

// If candidate <= n, we can count it; else only first\_digit-1

if (candidate <= n) ans += first\_digit;

else ans += (first\_digit - 1);

cout << ans << "\n";

}

return 0;

}

**📊 Complexity Analysis**

* **Time Complexity**:  
  For each test case:
  + Conversion to string → O(log n)
  + Simple arithmetic → O(1)  
    ✅ Total = O(t \* log n) (but since n ≤ 1e9, log n ≤ 10 → effectively O(t)).
* **Memory Complexity**: O(1) (just a few variables).

**✔ Example Dry Run**

Input:

n = 100

* len = 3, first\_digit = 1.
* Numbers with length < 3 → (3-1)\*9 = 18.
* Candidate with length 3 → 111. But 111 > 100, so don’t count.
* Add first\_digit - 1 = 0.
* ✅ Answer = 18.