**G-Hay Points** [**UVA - 10295**](https://vjudge.net/problem/UVA-10295/origin)**https://vjudge.net/problem/UVA-10295**

#include <iostream>

#include <unordered\_map>

#include <string>

#include <sstream>

#include <cstdio>

using namespace std;

int main() {

    int m, n;

    scanf("%d %d", &m, &n);

    unordered\_map<string, int> hayDict;

    char word[21];

    int value;

    // Reading the dictionary

    for (int i = 0; i < m; ++i) {

        scanf("%s %d", word, &value);

        hayDict[string(word)] = value;

    }

    // Clear newline character after last input

    cin.ignore();

    // Reading job descriptions

    string line;

    int total = 0;

    while (n--) {

        total = 0;

        while (getline(cin, line)) {

            if (line == ".") break;

            stringstream ss(line);

            string token;

            while (ss >> token) {

                if (hayDict.count(token)) {

                    total += hayDict[token];

                }

            }

        }

        printf("%d\n", total);

    }

    return 0;

}

**H-The Department of Redundancy Department**[**UVA - 484**](https://vjudge.net/problem/UVA-484/origin)**(https://vjudge.net/problem/UVA-484)**

#include <iostream>

#include <unordered\_map>

#include <vector>

using namespace std;

int main() {

    unordered\_map<int, int> freqMap;

    vector<int> order;

    int num;

    while (cin >> num) {

        if (freqMap.count(num) == 0) {

            order.push\_back(num);

        }

        freqMap[num]++;

    }

    for (int val : order) {

        cout << val << " " << freqMap[val] << "\n";

    }

    return 0;

}

**I-Babelfish** [**UVA - 10282**](https://vjudge.net/problem/UVA-10282/origin)[**https://vjudge.net/problem/UVA-10282**](https://vjudge.net/problem/UVA-10282)

#include <iostream>

#include <unordered\_map>

#include <string>

using namespace std;

int main() {

    unordered\_map<string, string> dict;

    string line, english, foreign;

    // Read dictionary entries

    while (getline(cin, line) && !line.empty()) {

        size\_t spacePos = line.find(' ');

        english = line.substr(0, spacePos);

        foreign = line.substr(spacePos + 1);

        dict[foreign] = english;

    }

    // Read and translate the foreign message

    while (getline(cin, foreign)) {

        if (dict.find(foreign) != dict.end()) {

            cout << dict[foreign] << "\n";

        } else {

            cout << "eh\n";

        }

    }

    return 0;

}

**J-Swappable**[**AtCoder - abc206\_c**](https://vjudge.net/problem/AtCoder-abc206_c/origin)[**https://vjudge.net/problem/AtCoder-abc206\_c**](https://vjudge.net/problem/AtCoder-abc206_c)

**Efficient Approach**

**Step-by-step:**

1. **Total pairs** (without conditions):

total\_pairs=N×(N−1) / 2

1. **Subtract invalid pairs (A[i] == A[j])**:
   * Count frequency of each element.
   * For every value with count f, the number of bad pairs is:

f×(f−1) / 2

* + Subtract those from total\_pairs

#include <iostream>

#include <unordered\_map>

using namespace std;

int main() {

    ios::sync\_with\_stdio(false);

    cin.tie(NULL);

    long long n;

    cin >> n;

    unordered\_map<int, long long> freq;

    int x;

    for (int i = 0; i < n; ++i) {

        cin >> x;

        freq[x]++;

    }

    long long total\_pairs = n \* (n - 1) / 2;

    for (auto it = freq.begin(); it != freq.end(); ++it) {

        long long count = it->second;

        total\_pairs -= count \* (count - 1) / 2;

    }

    cout << total\_pairs << '\n';

    return 0;

}

**Step-by-Step Complexity:**

Step 1: Reading Input and Counting Frequencies

Time: O(n)

Each element is read once and inserted/updated in the hash map.

Average-case insertion into an unordered\_map is O(1).

Space: O(k)

Where k is the number of distinct elements in the array.

Worst case: k = n (all values are unique).

Time: O(1)

Space: O(1)

Time: O(k)

We loop once over the distinct elements.

Space: O(1) extra (since we're using existing map structure).

Overall Complexity

Operation Time Complexity Space Complexity

Reading & frequency count O(n) O(k)

Compute total pairs O(1) O(1)

Subtract duplicates O(k) O(1)

Total O(n) O(k)

Where k is the number of distinct integers (k ≤ n).

Summary:

The algorithm runs in linear time relative to input size: O(n).

It uses linear space in the number of distinct integers.

**K. Very Hard Problem** [**https://www.hackerrank.com/challenges/deque-stl/problem**](https://www.hackerrank.com/challenges/deque-stl/problem)

#include <iostream>

#include <deque>

#include <vector>

using namespace std;

void printMaxInSubarrays(const vector<int> &arr, int k)

{

    deque<int> dq;

    int n = arr.size();

    for (int i = 0; i < n; ++i)

    {

        // Remove out-of-window indices

        if (!dq.empty() && dq.front() <= i - k)

            dq.pop\_front();

        // Remove smaller values from back

        while (!dq.empty() && arr[dq.back()] <= arr[i])

            dq.pop\_back();

        dq.push\_back(i);

        // Start printing from the first complete window

        if (i >= k - 1)

            cout << arr[dq.front()] << " ";

    }

    cout << "\n";

}

int main()

{

    ios::sync\_with\_stdio(false);

    cin.tie(nullptr);

    int t;

    cin >> t;

    while (t--)

    {

        int n, k;

        cin >> n >> k;

        vector<int> arr(n);

        for (int i = 0; i < n; ++i)

        {

            cin >> arr[i];

        }

        printMaxInSubarrays(arr, k);

    }

    return 0;

}

**Time Complexity:**

For each test case:

* Every element is pushed/popped from deque **at most once**
* Total: **O(N)**

**Overall Time Complexity:**

**O(T × N)**, where N is max array length per test case

**L-Epic Transformation** [**https://codeforces.com/problemset/problem/1506/D**](https://codeforces.com/problemset/problem/1506/D)

#include <cstdio>

#include <unordered\_map>

#include <algorithm>

int main() {

int t;

scanf("%d", &t);

while (t--) {

int n;

scanf("%d", &n);

std::unordered\_map<int, int> freq;

freq.reserve(n); // Avoid rehashing

int maxFreq = 0;

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

int x;

scanf("%d", &x);

freq[x]++;

if (freq[x] > maxFreq)

maxFreq = freq[x];

}

if (maxFreq > n - maxFreq)

printf("%d\n", 2 \* maxFreq - n);

else

printf("%d\n", n % 2);

}

return 0;

}

**Why This Is Fast:**

* scanf/printf is significantly faster than cin/cout for large input/output.
* Reserving space in unordered\_map with reserve(n) avoids repeated hash table reallocation.
* Avoids use of vectors/strings/maps unnecessarily

**Time & Space Complexity**

* **Time per test case:** O(n)
* **Space:** O(k) where k is the number of distinct integers (≤ n)

**M. Compression and Expansion** <https://vjudge.net/problem/CodeForces-1523C>

**N-Challenge Problem 1 (I. Palindrome Pairs)** [**https://codeforces.com/problemset/problem/1045/I**](https://codeforces.com/problemset/problem/1045/I)

#include <bits/stdc++.h>

using namespace std;

int main() {

    ios::sync\_with\_stdio(false);

    cin.tie(nullptr);

    int n;

    cin >> n;

    unordered\_map<int, int> freq;

    vector<int> masks(n);

    for (int i = 0; i < n; ++i) {

        string s;

        cin >> s;

        int mask = 0;

        for (char c : s) {

            mask ^= (1 << (c - 'a')); // Toggle the bit for character c

        }

        masks[i] = mask;

        freq[mask]++;

    }

    long long result = 0;

    for (int i = 0; i < n; ++i) {

        int current\_mask = masks[i];

        freq[current\_mask]--; // Avoid double counting

        // 1. Pair with exact same mask (even char counts)

        result += freq[current\_mask];

        // 2. Pair with masks that differ by one bit (one odd char allowed)

        for (int bit = 0; bit < 26; ++bit) {

            int flipped = current\_mask ^ (1 << bit);

            result += freq[flipped];

        }

    }

    cout << result << '\n';

    return 0;

}

**Time Complexity**

* Processing each string: O(len)
* Total bitmask count: O(26 × n) ≈ O(n)
* Efficient because we avoid O(n²) comparisons.

**O. Challenge Problem 2 ( Balanced Brackets [HackerRank - balanced-brackets](https://vjudge.net/problem/HackerRank-balanced-brackets/origin" \t "_blank))** [**https://vjudge.net/problem/HackerRank-balanced-brackets**](https://vjudge.net/problem/HackerRank-balanced-brackets)

#include <iostream>

#include <stack>

#include <string>

using namespace std;

bool isBalanced(string s) {

stack<char> st;

for (char c : s) {

if (c == '(' || c == '{' || c == '[') {

st.push(c);

} else {

if (st.empty()) return false;

char top = st.top();

st.pop();

if ((c == ')' && top != '(') ||

(c == '}' && top != '{') ||

(c == ']' && top != '[')) {

return false;

}

}

}

return st.empty();

}

int main() {

int n;

cin >> n;

while (n--) {

string s;

cin >> s;

cout << (isBalanced(s) ? "YES" : "NO") << '\n';

}

return 0;

}

**Time Complexity**

* **O(n)** per string (where n is the length of the string).
* Efficient for up to **100,000** strings as each character is visited once.

Using Vector :

#include <bits/stdc++.h>

using namespace std;

bool isBalanced(const string& s) {

    vector<char> stack; // using vector as a stack

    for (char c : s) {

        if (c == '(' || c == '{' || c == '[') {

            stack.push\_back(c); // push to back

        } else {

            if (stack.empty()) return false;

            char top = stack.back(); // top element

            stack.pop\_back(); // remove top

            if ((c == ')' && top != '(') ||

                (c == '}' && top != '{') ||

                (c == ']' && top != '[')) {

                return false;

            }

        }

    }

    return stack.empty(); // if not empty, there are unmatched opening brackets

}

int main() {

    int t;

    cin >> t;

    while (t--) {

        string s;

        cin >> s;

        cout << (isBalanced(s) ? "YES" : "NO") << '\n';

    }

    return 0;

}