**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>

Soln

**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)**)** [**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;

}

P. Challenge Problem 3 HISTOGRA - Largest Rectangle in a Histogram (<https://www.spoj.com/problems/HISTOGRA/cstart=40>)

#include <bits/stdc++.h>

using namespace std;

typedef long long ll;

ll largestRectangle(vector<ll>& heights) {

    stack<int> st;

    ll maxArea = 0;

    int n = heights.size();

    for (int i = 0; i <= n; ++i) {

        ll h = (i == n ? 0 : heights[i]);

        while (!st.empty() && heights[st.top()] > h) {

            ll height = heights[st.top()];

            st.pop();

            ll width = st.empty() ? i : i - st.top() - 1;

            maxArea = max(maxArea, height \* width);

        }

        st.push(i);

    }

    return maxArea;

}

int main() {

    while (true) {

        int n;

        cin >> n;

        if (n == 0) break;

        vector<ll> heights(n);

        for (int i = 0; i < n; ++i) {

            cin >> heights[i];

        }

        cout << largestRectangle(heights) << "\n";

    }

    return 0;

}

Q. Text Reverse <https://vjudge.net/problem/HDU-1062>

#include <iostream>

#include <sstream>  // for istringstream

#include <string>

#include <algorithm> // for reverse()

using namespace std;

int main() {

    int T;

    cin >> T;

    cin.ignore(); // 🔁 Ignore the newline after the integer input

    while (T--) {

        string line;

        getline(cin, line); // 🔁 Read the whole line with spaces

        istringstream iss(line); // Stream to split line into words

        string word;

        bool first = true;

        while (iss >> word) {

            reverse(word.begin(), word.end()); // 🔁 Reverse the word

            if (!first) cout << " "; // Add space if it's not the first word

            cout << word;

            first = false;

        }

        cout << '\n'; // Newline after each test case

    }

    return 0;

}

R. ACboy needs your help again! (<https://vjudge.net/problem/HDU-1702>)

My Approach:

#include <bits/stdc++.h>

using namespace std;

int main()

{

    int t;

    cin >> t;

    while (t--)

    {

        stack<int> st;

        queue<int> q;

        int n;

        cin >> n;

        string s;

        cin >> s;

        if (s == "FILO")

        {

            while (n--)

            {

                string command;

                cin >> command;

                if (command == "IN")

                {

                    int val;

                    cin >> val;

                    st.push(val);

                }

                if (command == "OUT")

                {

                    if (!st.empty())

                    {

                        cout << st.top() << endl;

                        st.pop();

                    }

                    else

                    {

                        cout << "None" << endl;

                    }

                }

            }

        }

        else

        {

            while (n--)

            {

                string command;

                cin >> command;

                if (command == "IN")

                {

                    int val;

                    cin >> val;

                    q.push(val);

                }

                if (command == "OUT")

                {

                    if (!q.empty())

                    {

                        cout << q.front() << endl;

                        q.pop();

                    }

                    else

                    {

                        cout << "None" << endl;

                    }

                }

            }

        }

    }

    return 0;

}

**GPT Approach:**

#include <bits/stdc++.h>

using namespace std;

int main() {

int T;

cin >> T; // total number of test cases

while (T--) {

int N;

string type;

cin >> N >> type; // Number of commands, and type (FIFO or FILO)

queue<int> q;

stack<int> s;

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

string cmd;

cin >> cmd;

if (cmd == "IN") {

int val;

cin >> val;

if (type == "FIFO") q.push(val); // enqueue

else s.push(val); // push to stack

} else if (cmd == "OUT") {

if (type == "FIFO") {

if (!q.empty()) {

cout << q.front() << endl;

q.pop();

} else {

cout << "None" << endl;

}

} else { // FILO

if (!s.empty()) {

cout << s.top() << endl;

s.pop();

} else {

cout << "None" << endl;

}

}

}

}

}

return 0;

}

S. Soldier and Cards (<https://vjudge.net/problem/CodeForces-546C>)

#include <iostream>

#include <queue>

#include <set>

#include <tuple>

using namespace std;

int main() {

    int n;cin >> n;

    int k1;cin >> k1;

    queue<int> p1;

    for (int i = 0; i < k1; ++i) {

        int x;

        cin >> x;

        p1.push(x);

    }

    int k2;cin >> k2;

    queue<int> p2;

    for (int i = 0; i < k2; ++i) {

        int x;

        cin >> x;

        p2.push(x);

    }

    int rounds = 0;

    set<pair<queue<int>, queue<int>>> seen;

    while (!p1.empty() && !p2.empty()) {

        if (seen.count({p1, p2})) {

            cout << -1 << endl;

            return 0;

        }

        seen.insert({p1, p2});

        int c1 = p1.front(); p1.pop();

        int c2 = p2.front(); p2.pop();

        if (c1 > c2) {

            p1.push(c2);

            p1.push(c1);

        } else {

            p2.push(c1);

            p2.push(c2);

        }

        rounds++;

    }

    if (p1.empty()) {

        cout << rounds << " 2\n";

    } else {

        cout << rounds << " 1\n";

    }

    return 0;

}

**T. Misha and Changing Handles(**[**https://vjudge.net/problem/codeforces-501b**](https://vjudge.net/problem/codeforces-501b)**)**

#include <iostream>

#include <map>

#include <string>

using namespace std;

int main() {

    int q;

    cin >> q;

    map<string, string> current\_to\_original;

    for (int i = 0; i < q; i++) {

        string old\_handle, new\_handle;

        cin >> old\_handle >> new\_handle;

        string original;

        if (current\_to\_original.find(old\_handle) != current\_to\_original.end()) {

            // old\_handle was already a changed handle

            original = current\_to\_original[old\_handle];

            current\_to\_original.erase(old\_handle);

        } else {

            // old\_handle is an original handle

            original = old\_handle;

        }

        // Map new\_handle to the original

        current\_to\_original[new\_handle] = original;

    }

    // Count users who changed handles (excluding those who didn't change)

    int count = 0;

    for (auto& pair : current\_to\_original) {

        if (pair.first != pair.second) {

            count++;

        }

    }

    cout << count << endl;

    // Output the mappings

    for (auto& pair : current\_to\_original) {

        if (pair.first != pair.second) {

            cout << pair.second << " " << pair.first << endl;

        }

    }

    return 0;

}

**Time Complexity:** O(q log q) due to map operations

Space Complexity: O(q) for storing the mappings

**U-TKTKTKKT - Team Queue -** [**https://vjudge.net/problem/UVA-540**](https://vjudge.net/problem/UVA-540)

**(**[**https://www.spoj.com/problems/TKTKTKKT/**](https://www.spoj.com/problems/TKTKTKKT/)**)**

#include <iostream>

#include <queue>

#include <map>

#include <string>

using namespace std;

int main() {

    int t;

    int scenario = 1;

    while (cin >> t && t != 0) {

        cout << "Scenario #" << scenario << endl;

        // Map element to team number

        map<int, int> elementToTeam;

        // Read team descriptions

        for (int i = 0; i < t; i++) {

            int teamSize;

            cin >> teamSize;

            for (int j = 0; j < teamSize; j++) {

                int element;

                cin >> element;

                elementToTeam[element] = i;

            }

        }

        // Main queue to track order of teams

        queue<int> teamOrder;

        // Individual queues for each team

        vector<queue<int>> teamQueues(t);

        // Track which teams are currently in the main queue

        vector<bool> teamInQueue(t, false);

        string command;

        while (cin >> command && command != "STOP") {

            if (command == "ENQUEUE") {

                int x;

                cin >> x;

                int team = elementToTeam[x];

                teamQueues[team].push(x);

                // If this team is not already in the main queue, add it

                if (!teamInQueue[team]) {

                    teamOrder.push(team);

                    teamInQueue[team] = true;

                }

            }

            else if (command == "DEQUEUE") {

                // Get the team at the front of the main queue

                int frontTeam = teamOrder.front();

                // Dequeue from that team's queue

                int element = teamQueues[frontTeam].front();

                teamQueues[frontTeam].pop();

                cout << element << endl;

                // If this team's queue is now empty, remove it from main queue

                if (teamQueues[frontTeam].empty()) {

                    teamOrder.pop();

                    teamInQueue[frontTeam] = false;

                }

            }

        }

        cout << endl;

        scenario++;

    }

    return 0;

}

**Time Complexity**: Both ENQUEUE and DEQUEUE are O(1) operations as required.

Space Complexity: O(total number of elements + number of teams)

**Andy's First Dictionary UVA - 10815** [**https://vjudge.net/problem/UVA-10815**](https://vjudge.net/problem/UVA-10815)

#include <bits/stdc++.h>

using namespace std;

int main() {

    set<string> dictionary;

    string word;

    string line;

    while (getline(cin, line)) {

        for (char ch : line) {

            if (isalpha(ch)) {

                word += tolower(ch);

            } else {

                if (!word.empty()) {

                    dictionary.insert(word);

                    word.clear();

                }

            }

        }

        // In case the line ends with a word

        if (!word.empty()) {

            dictionary.insert(word);

            word.clear();

        }

    }

    for (const string& w : dictionary) {

        cout << w << '\n';

    }

    return 0;

}

**⏱️ Time Complexity**

Let’s assume:

* N = total number of characters in the input (up to 5000 lines × 200 chars = **~1,000,000 chars**).
* W = number of distinct words (maximum 5000).

**Step-by-step:**

1. **Reading Input & Extracting Words**:
   * Iterate through each character once: **O(N)**
   * Extract words using character checks: **O(N)**
2. **Converting to lowercase**:
   * Each word is converted to lowercase: assume **average length L**, for W words → **O(W × L)**  
     (Since L is small, treated as constant → O(W))
3. **Inserting into set<string>**:
   * Insertion in a set (Red-Black Tree) is **O(log W)** per word.
   * Insert W words → **O(W log W)**
4. **Outputting sorted words**:
   * Iterating through the set: **O(W)**

**✅ Total Time Complexity:**

scss

CopyEdit

O(N) + O(W × log W) ≈ O(N + W log W)

Since W ≤ 5000 and N ≤ 1,000,000 — this is very efficient.

**🧠 Space Complexity**

1. **Word Storage**:
   * Up to W words (each up to ~100 chars max) → **O(W × L)**  
     → Treated as **O(W)** in practice.
2. **Set Structure Overhead**:
   * std::set maintains a tree → extra overhead for tree pointers  
     → Still within **O(W)** space.
3. **Temporary Strings**:
   * A few extra strings during processing, negligible.

**Total Space Complexity:**

scss

CopyEdit

O(W)

**Summary Table**

| **Component** | **Complexity** |
| --- | --- |
| Time Complexity | O(N + W log W) |
| Space Complexity | O(W) |

**101 The Blocks Problem -** [**https://vjudge.net/problem/POJ-1208**](https://vjudge.net/problem/POJ-1208)

#include <bits/stdc++.h>

#include <algorithm>

using namespace std;

const int MAX = 25;

vector<vector<int>> blocks(MAX);

int position[MAX]; // position[i] = stack index where block i currently is

// Return blocks stacked above 'block' to their original positions

void returnAbove(int block) {

    int pos = position[block];

    while (blocks[pos].back() != block) {

        int top = blocks[pos].back();

        blocks[pos].pop\_back();

        blocks[top].push\_back(top);

        position[top] = top;

    }

}

// Move block a onto block b (after returning blocks above both a and b)

void moveOnto(int a, int b) {

    returnAbove(a);

    returnAbove(b);

    blocks[position[a]].pop\_back();

    blocks[position[b]].push\_back(a);

    position[a] = position[b];

}

// Move block a over block b (return blocks above a, then place on top of b's stack)

void moveOver(int a, int b) {

    returnAbove(a);

    blocks[position[a]].pop\_back();

    blocks[position[b]].push\_back(a);

    position[a] = position[b];

}

// Pile blocks starting from a onto block b (return blocks above b first)

void pileOnto(int a, int b);

// Pile blocks starting from a over block b

void pileOver(int a, int b) {

    int from = position[a], to = position[b];

    vector<int> &src = blocks[from];

    vector<int> &dest = blocks[to];

    auto it = find(src.begin(), src.end(), a);

    vector<int> moving(it, src.end());

    src.erase(it, src.end());

    for (int block : moving) {

        dest.push\_back(block);

        position[block] = to;

    }

}

void pileOnto(int a, int b) {

    returnAbove(b);

    pileOver(a, b);

}

int main() {

    int n;

    cin >> n;

    string line;

    getline(cin, line); // consume newline after n

    // Initialize each block in its own position

    for (int i = 0; i < n; ++i) {

        blocks[i].clear();

        blocks[i].push\_back(i);

        position[i] = i;

    }

    while (getline(cin, line)) {

        if (line == "quit") break;

        stringstream ss(line);

        string cmd1, cmd2;

        int a, b;

        ss >> cmd1 >> a >> cmd2 >> b;

        if (a == b || position[a] == position[b]) continue;

        if (cmd1 == "move" && cmd2 == "onto") moveOnto(a, b);

        else if (cmd1 == "move" && cmd2 == "over") moveOver(a, b);

        else if (cmd1 == "pile" && cmd2 == "onto") pileOnto(a, b);

        else if (cmd1 == "pile" && cmd2 == "over") pileOver(a, b);

    }

    // Print final state

    for (int i = 0; i < n; ++i) {

        cout << i << ":";

        for (int b : blocks[i]) {

            cout << " " << b;

        }

        cout << "\n";

    }

    return 0;

}

**Notation**

* Let **n** = number of blocks (≤ 25, very small).
* Let **m** = number of commands (could be large, but typically ≤ 1000 or so).

**Time Complexity**

**1. Initialization**

* Initializing stacks and positions takes **O(n)**.

**2. Processing commands**

For each command (move or pile), the complexity depends on how many blocks get moved or returned:

* **returnAbove(block)**:
  + Returns any blocks above block to their original positions.
  + In the worst case, this could move **up to n** blocks one by one.
  + Each block moved updates position and moves between stacks.
  + So worst case **O(n)** per call.
* **moveOnto(a, b)**:
  + Calls returnAbove(a) and returnAbove(b) → each up to **O(n)**.
  + Then moves block a → O(1).
  + Total: **O(n)**.
* **moveOver(a, b)**:
  + Calls returnAbove(a) → O(n).
  + Moves block a → O(1).
  + Total: **O(n)**.
* **pileOver(a, b)**:
  + Finds block a in its stack → in worst case **O(n)**.
  + Moves sub-stack from a to b → up to **O(n)** blocks moved.
  + Total: **O(n)**.
* **pileOnto(a, b)**:
  + Calls returnAbove(b) → O(n).
  + Calls pileOver(a, b) → O(n).
  + Total: **O(n)**.

**Overall per command:**

**O(n)** in the worst case.

**Total Complexity**

For **m** commands, total worst case time complexity:

O(m×n)O(m \times n)O(m×n)

Since n≤25n \leq 25n≤25, this is effectively linear in mmm, very efficient.

**Space Complexity**

* Storing positions: **O(n)**
* Storing stacks: at most all blocks in one stack → **O(n)**
* Temporary vectors for pile operations: **O(n)**

Total space complexity:

O(n)O(n)O(n)

**Summary**

| **Complexity** | **Value** |
| --- | --- |
| Time Complexity | O(m×n)O(m \times n)O(m×n) |
| Space Complexity | O(n)O(n)O(n) |

**Since n is very small (max 25), this solution runs very efficiently even for thousands of commands.**

**X. Lala Land and Apple Trees -** [**https://vjudge.net/problem/CodeForces-558A**](https://vjudge.net/problem/CodeForces-558A)

#include <bits/stdc++.h>

using namespace std;

int main() {

    int n;

    cin >> n;

    vector<pair<int, int>> left, right;

    for (int i = 0; i < n; i++) {

        int x, a;

        cin >> x >> a;

        if (x < 0) {

            left.push\_back({x, a});

        } else {

            right.push\_back({x, a});

        }

    }

    // Sort left (from closest to 0, so descending order)

    sort(left.begin(), left.end(), greater<>());

    // Sort right (from closest to 0, so ascending order)

    sort(right.begin(), right.end());

    int total = 0;

    int len = min(left.size(), right.size());

    // Pick equal number from both sides

    for (int i = 0; i < len; i++) {

        total += left[i].second;

        total += right[i].second;

    }

    // Take one extra from the side that has more trees

    if (left.size() > right.size()) {

        total += left[len].second;

    } else if (right.size() > left.size()) {

        total += right[len].second;

    }

    cout << total << endl;

    return 0;

}

**Time and Space Complexity:**

* **Time:** O(n log n) → due to sorting left and right.
* **Space:** O(n) → for storing the two vectors.

**Y. A problem of sorting -** [**https://vjudge.net/problem/HDU-5427**](https://vjudge.net/problem/HDU-5427)

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

struct Person {

    string name;

    int birthYear;

};

// Comparator to sort from youngest to oldest (larger birthYear first)

bool compare(Person a, Person b) {

    return a.birthYear > b.birthYear;

}

int main() {

    int T;

    cin >> T;

    while (T--) {

        int n;

        cin >> n;

        vector<Person> people(n);

        // Read input

        for (int i = 0; i < n; ++i) {

            cin >> ws; // consume any leading whitespaces

            getline(cin, people[i].name, ' ');

            cin >> people[i].birthYear;

        }

        // Sort from youngest to oldest

        sort(people.begin(), people.end(), compare);

        // Output names only

        for (int i = 0; i < n; ++i) {

            cout << people[i].name << endl;

        }

    }

    return 0;

}