Basic Template & Macros

```cpp
#include <bits/stdc++.h>
using namespace std;

#define all(x) (x).begin(), (x).end()
#define rall(x) (x).rbegin(), (x).rend()
#define sz(x) (int)(x).size()
#define pb push_back
#define mp make_pair

using ll = long long;
using vi = vector<int>;
using vl = vector<ll>;
using pii = pair<int, int>;
using pll = pair<ll, ll>;
using vpii = vector<pii>;
using vpll = vector<pll>;

void solve() {

}

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr);

#ifndef ONLINE_JUDGE
    freopen("input.txt", "r", stdin);
    freopen("output.txt", "w", stdout);
#endif

    int t_case = 1;
    // cin >> t_case;

    while (t_case--) {
        solve();
    }

    return 0;
}
```

VSCode JSON

```
// --- settings.json ---
"C_Cpp.clang_format_fallbackStyle":
"{ BasedOnStyle: Google, IndentWidth: 4, ColumnLimit: 0 }",
"editor.formatOnSave": true,
"files.autoSave": "onFocusChange",

// --- c_cpp_properties.json ---
"includePath": ["${workspaceFolder}/",
"C:/msys64/mingw64/include/"],
"cppStandard": "c++17",
"compilerPath": "C:/msys64/mingw64/bin/g++.exe",
"intelliSenseMode": "windows-gcc-x64",

// --- tasks.json ---
"args": ["-std=c++17", "-O2", "-Wall", "-Wextra"],

// --- launch.json (optional) ---
{
"configurations": [{
"name": "C++ Debug",
"type": "cppdbg",
"request": "launch",
"program": "${workspaceFolder}/${fileBasenameNoExtension}.exe",
"args": [],
"stopAtEntry": false,
"cwd": "${workspaceFolder}",
"externalConsole": true,
"MIMode": "gdb",
"miDebuggerPath": "C:/msys64/mingw64/bin/gdb.exe",
"preLaunchTask": "C/C++: g++.exe build active file"
}]
}
```

String Operations

```
string s;
cin >> s; // Read word (no spaces)
getline(cin, s); // Read line (with spaces)

// Basic operations
s.length(); // Get length
s.empty(); // Check if empty
s.clear(); // Clear string
s += "text"; // Append
s.push_back('c'); // Add character
s.pop_back(); // Remove last character

// Access & modify
s[0]; s.at(0); // Access character
s.front(); s.back(); // First/last character
s.substr(pos, len); // Get substring
s.find("pattern"); // Find first occurrence
s.rfind("pattern"); // Find last occurrence
s.erase(pos, len); // Erase from position
s.insert(pos, "text"); // Insert at position

// Transform
reverse(all(s)); // Reverse string
sort(all(s)); // Sort characters

// Numeric conversion
int n = stoi(s); // String to int
string num = to_string(123); // Number to string
```

Vector Operations

```cpp
// Input vector
vector<int> v(n); for(int i = 0, i < n, i++) cin >> v[i];

// Print vector
for(int x : v) cout << x << " "; cout << endl;

// Common operations
v.size();                  // Size
v.empty();                 // Is empty?
v.push_back(x);            // Add to end
v.pop_back();              // Remove last
v.erase(v.begin()+i);      // Erase at index i
v.insert(v.begin()+i,x);   // Insert x at index i
sort(v.begin(),v.end());     // Sort ascending
sort(v.rbegin(),v.rend());   // Sort descending
reverse(v.begin(),v.end()); // Reverse
*min_element(v.begin(),v.end()); // Min element
*max_element(v.begin(),v.end()); // Max element
binary_search(v.begin(),v.end(),x); // Check if x exists
lower_bound(v.begin(),v.end(),x); // First >= x
upper_bound(v.begin(),v.end(),x); // First > x
```

Set Operations

```cpp
set<int> s;               // Init set (sorted, unique)
s.insert(x);              // Add x
s.erase(x);               // Remove x
s.count(x);               // Check if x exists (0/1)
s.find(x)!=s.end();       // Alternative check
s.size();                 // Number of elements
s.empty();                // Is empty?
s.clear();                // Clear all elements

// Print set
for(int x : s) cout << x << " "; cout << endl;

// unordered_set (faster, no order)
unordered_set<int> us;
us.insert(x);
```

Map Operations

```cpp
map<string,int> m;     // Init map
m[key] = val;          // Add/update key-val
m.count(key);          // Check if key exists
m.erase(key);          // Remove key
m.clear();             // Clear all elements
m.size();              // Size of map
m.empty();             // Check if empty
for(auto &p : m) cout << p.first << ":" << p.second << endl; // Print map

// unordered_map (faster, but no order)
unordered_map<int,int> um;
um[key] = val;
```

Graph (Adjacency List)

```cpp
// Undirected graph with n nodes
int n, m;
cin >> n >> m;
vector<vector<int>> adj(n+1);

// Input edges
for(int i = 0; i < m; i++) {
    int u, v; cin >> u >> v;
    adj[u].push_back(v);
    adj[v].push_back(u); // Remove if directed
}

// Weighted graph
vector<vector<pii>> wadj(n+1); // pair<node, weight>
wadj[u].push_back({v, w});

// Common operations
adj[u].size(); // Degree of node u
adj[u].clear(); // Clear all edges from u
adj[u].empty(); // Check if node has no edges

// Check if edge u-v exists
bool edge_exists = find(all(adj[u]), v) != adj[u].end();

// Remove edge u-v
adj[u].erase(remove(all(adj[u]), v), adj[u].end());

// Print adjacency list
for(int u = 1; u <= n; u++) {
    cout << u << ": ";
    for(int v : adj[u]) cout << v << " ";
    cout << endl;
}
```

Tree (Adjacency List)

```cpp
// Tree with n nodes (1-based indexing)
vector<vector<int>> tree(n+1);

// Add undirected edge
tree[u].pb(v);
tree[v].pb(u);

// Common operations
tree[u].size(); // Degree of node u
tree[u].clear(); // Remove all edges from u
tree[u].erase(find(all(tree[u]), v)); // Remove edge u-v
auto it = find(all(tree[u]), v); // Check if edge exists

// DFS traversal (iterative)
vector<int> visited(n+1,0);
stack<int> st;
st.push(1); // root

while(!st.empty()) {
    int u = st.top(); st.pop();
    visited[u] = 1;
    for(int v : tree[u])
        if(!visited[v]) st.push(v);
}

// BFS traversal
queue<int> q;
vector<int> dist(n+1,-1);
q.push(1); dist[1] = 0;

while(!q.empty()) {
    int u = q.front(); q.pop();
    for(int v : tree[u])
        if(dist[v] == -1) {
            dist[v] = dist[u] + 1;
            q.push(v);
        }
}
```

Breadth-First Search (BFS)

```cpp
vector<int> dist; // To store shortest distance
void bfs(int start_node, int V) {
    dist.assign(V + 1, -1);
    queue<int> q;

    dist[start_node] = 0;
    q.push(start_node);

    while (!q.empty()) {
        int u = q.front();
        q.pop();

        for (int v : adj[u]) {
            if (dist[v] == -1) {
                dist[v] = dist[u] + 1;
                q.push(v);
            }
        }
    }
}
```

Depth-First Search (DFS)

```cpp
vector<bool> visited;

void dfs(int u) {
    visited[u] = true;
    // cout << u << " "; // Pre-order action

    for (int v : adj[u]) {
        if (!visited[v]) {
            dfs(v);
        }
    }
    // cout << u << " "; // Post-order action
}

void traverse_graph(int V) {
    visited.assign(V + 1, false);
    for(int i = 1; i <= V; i++) {
        if (!visited[i]) {
            // start DFS for a new component
            dfs(i);
        }
    }
}
```

Binary Search on Answer Range

```
// Finds smallest X satisfying a monotonic check(X)
ll bs_answer(ll low, ll high) {
    ll ans = high;
    while (low <= high) {
        ll mid = low + (high - low) / 2;
        if (check(mid)) { // Problem-specific boolean function
            ans = mid;
            high = mid - 1; // Try smaller
        } else {
            low = mid + 1;  // Too small
        }
    }
    return ans;
}
```

Two Pointers Technique

```
// For two-sum on sorted array 'a'
void two_pointers_sum(vector<int>& a, int target) {
    int left = 0;
    int right = sz(a) - 1;

    while (left < right) {
        int current_sum = a[left] + a[right];
        if (current_sum == target) {
            // Success logic
            left++; right--;
        } else if (current_sum < target) {
            left++;
        } else {
            right--;
        }
    }
}
```

Mathematical Functions

```

// Prime check
bool is_prime(ll n) {
    if(n < 2) return false;
    for(ll i = 2; i*i <= n; i++)
        if(n % i == 0) return false;
    return true;
}

// LCM
ll lcm(ll a, ll b) {
    return a / gcd(a, b) * b;
}

// Sum of first n natural numbers
ll sum_n(ll n) {
    return n * (n + 1) / 2;
}

// Check even/odd
bool is_even(ll n) {
    return n % 2 == 0;
}
bool is_odd(ll n) {
    return n % 2 == 1;
}

// Absolute value
ll abs(ll x) {
    return x < 0 ? -x : x;
}
```