Basic Template

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

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

    freopen("D:/File/input.txt", "r", stdin);
    freopen("D:/File/output.txt", "w", stdout);

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

    while(t_case--) {

    }

    return 0;
}
```

VSCode JSON

```json
// --- settings.json (File > Preferences > Settings) ---

// -> Add this line for Google-style formatting
"C_Cpp.clang_format_fallbackStyle": "{ BasedOnStyle: Google,
   IndentWidth: 4, ColumnLimit: 0 }",

// -> Add this line to format code every time on save
"editor.formatOnSave": true,


// --- c_cpp_properties.json (Ctrl+Shift+P > Edit Configurations) ---

// -> Add your g++ include path to fix "bits/stdc++.h" error
"includePath": ["${workspaceFolder}/**", "C:/msys64/mingw64/include/**
   "],

// -> Add this line to set C++ version for IntelliSense
"cppStandard": "c++17",


// --- tasks.json (Terminal > Configure Default Build Task) ---

// -> Add this to your build "args" to compile with C++17
"-std=c++17",
```

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
}

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

Vector Operations

```
// 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

```
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);
```

Tree (Rooted, using DFS)

```
// Tree as adjacency list
int n; cin >> n;
vector<vector<int>> tree(n+1);

for(int i = 0; i < n-1; i++) {
    int u, v; cin >> u >> v;
    tree[u].push_back(v);
    tree[v].push_back(u);
}

// DFS traversal
vector<int> visited(n+1, 0);
void dfs(int u) {
    visited[u] = 1;
    cout << u << " ";
    for(int v : tree[u])
        if(!visited[v]) dfs(v);
}

dfs(1); // Assuming 1 is root
\end{stlisting}

\newpage
\subsection*{Common Algorithms}
\begin{lstlisting}
// GCD
long long gcd(long long a, long long b) { return b ? gcd(b, a%b) : a;
   }

// Sieve of Eratosthenes
vector<bool> isPrime(n+1,true);
void sieve(int n) {
    isPrime[0] = isPrime[1] = false;
    for(int i=2;i*i<=n;++i)
        if(isPrime[i])
            for(int j=i*i;j<=n;j+=i)
                isPrime[j] = false;
}

// Factorial
long long factorial(int n) {
    long long res = 1;
    for(int i = 1; i <= n; i++) res *= i;
    return res;
}

// Sum of digits
int sumOfDigits(long long n) {
    int sum = 0;
    while (n > 0) { sum += n % 10; n /= 10; }
    return sum;
}

// Fast exponentiation (a^b)
long long power(long long a, long long b) {
```

```cpp
    long long res = 1;
    while(b) {
        if(b & 1) res *= a;
        a *= a;
        b >>= 1;
    }
    return res;
}

// Prefix Sum
vector<int> prefix(n+1,0);
for(int i=1;i<=n;i++) prefix[i]=prefix[i-1]+arr[i];
// Query sum(l..r) = prefix[r]-prefix[l-1]
```