~~~
# **ICPC Dhaka Regional C++ Template 2025**

```cpp
/**************************************************
 * ICPC Dhaka Regional 2025 - Complete C++ Template
 * Created for Team: [YOUR_TEAM_NAME]
 * Target: Solve first 5 problems efficiently
 **************************************************/

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

// ==================== Optimization ====================
#pragma GCC optimize("O3,unroll-loops")
#pragma GCC target("avx2,bmi,bmi2,lzcnt,popcnt")

// ==================== Macros ====================
#define ll long long
#define ull unsigned long long
#define ld long double
#define endl "\n"
#define pb push_back
#define mp make_pair
#define all(x) (x).begin(), (x).end()
#define rall(x) (x).rbegin(), (x).rend()
#define sz(x) (int)(x).size()
#define ff first
#define ss second

// ==================== Debug ====================
#ifdef LOCAL
#define debug(x) cerr << #x << " = " << x << endl
#define debug_vec(v) cerr << #v << " = "; for(auto &x : v) cerr << x << " "; cerr << endl
#define debug_pair(p) cerr << #p << " = (" << p.ff << ", " << p.ss << ")" << endl
#else
#define debug(x)
#define debug_vec(v)
#define debug_pair(p)
#endif

// ==================== Fast I/O ====================
inline void fastIO() {
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr);
~~~

```
    cout.tie(nullptr);
}

// ==================== Math Functions ====================
ll gcd(ll a, ll b) { return b == 0 ? a : gcd(b, a % b); }
ll lcm(ll a, ll b) { return a / gcd(a, b) * b; }

ll mod_pow(ll base, ll exp, ll mod) {
    ll result = 1;
    while(exp > 0) {
        if(exp & 1) result = (result * base) % mod;
        base = (base * base) % mod;
        exp >>= 1;
    }
    return result;
}

ll mod_inv(ll a, ll mod) {
    return mod_pow(a, mod - 2, mod);
}

vector<int> sieve(int n) {
    vector<bool> isPrime(n+1, true);
    vector<int> primes;
    isPrime[0] = isPrime[1] = false;
    for(int i = 2; i <= n; i++) {
        if(isPrime[i]) {
            primes.pb(i);
            for(int j = i*i; j <= n; j += i) {
                isPrime[j] = false;
            }
        }
    }
    return primes;
}

vector<pair<ll, int>> prime_factorization(ll n) {
    vector<pair<ll, int>> factors;
    for(ll i = 2; i * i <= n; i++) {
        if(n % i == 0) {
            int cnt = 0;
            while(n % i == 0) {
                n /= i;
                cnt++;
            }
            factors.pb({i, cnt});
        }
    }
    if(n > 1) factors.pb({n, 1});
    return factors;
}

// ==================== Combinatorics ====================
const int MOD = 1e9 + 7;
const int MAX_N = 1e6 + 5;

vector<ll> fact(MAX_N), inv_fact(MAX_N);

void precompute_factorials() {
    fact[0] = 1;
    for(int i = 1; i < MAX_N; i++) {
        fact[i] = (fact[i-1] * i) % MOD;
    }
    inv_fact[MAX_N-1] = mod_inv(fact[MAX_N-1], MOD);
    for(int i = MAX_N-2; i >= 0; i--) {
        inv_fact[i] = (inv_fact[i+1] * (i+1)) % MOD;
    }
}

ll nCr(int n, int r) {
    if(r < 0 || r > n) return 0;
    return (fact[n] * inv_fact[r] % MOD) * inv_fact[n-r] % MOD;
}

ll nPr(int n, int r) {
    if(r < 0 || r > n) return 0;
    return fact[n] * inv_fact[n-r] % MOD;
}

// ==================== Graph Algorithms ====================
struct Graph {
    int n;
    vector<vector<int>> adj;

    Graph(int nodes) {
        n = nodes;
        adj.resize(n+1);
    }
    void add_edge(int u, int v, bool directed = false) {
        adj[u].pb(v);
        if(!directed) adj[v].pb(u);
    }
};

vector<int> bfs(int start, const vector<vector<int>> &adj) {
    int n = adj.size();
    vector<int> dist(n, -1);
    queue<int> q;
    q.push(start);
    dist[start] = 0;

    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);
            }
        }
    }
    return dist;
}

vector<int> topological_sort(const vector<vector<int>> &adj) {
    int n = adj.size();
    vector<int> in_degree(n, 0), topo;

    for(int u = 0; u < n; u++) {
        for(int v : adj[u]) {
            in_degree[v]++;
        }
    }

    queue<int> q;
    for(int i = 0; i < n; i++) {
        if(in_degree[i] == 0) q.push(i);
    }

    while(!q.empty()) {
        int u = q.front(); q.pop();
        topo.pb(u);
        for(int v : adj[u]) {
            in_degree[v]--;
            if(in_degree[v] == 0) q.push(v);
        }
    }

    if(topo.size() != n) return {}; // cycle exists
    return topo;
}

// Dijkstra's Algorithm
vector<ll> dijkstra(int start, vector<vector<pair<int, ll>>> &adj) {
    int n = adj.size();
    vector<ll> dist(n, LLONG_MAX);
    priority_queue<pair<ll, int>, vector<pair<ll, int>>, greater<pair<ll, int>>> pq;

    dist[start] = 0;
    pq.push({0, start});

    while(!pq.empty()) {
        auto [d, u] = pq.top(); pq.pop();
        if(d > dist[u]) continue;

        for(auto &[v, w] : adj[u]) {
            if(dist[u] + w < dist[v]) {
                dist[v] = dist[u] + w;
                pq.push({dist[v], v});
            }
        }
    }
    return dist;
}

// ==================== DSU (Union-Find) ====================
struct DSU {
    vector<int> parent, size;

    DSU(int n) {
        parent.resize(n+1);
        size.resize(n+1, 1);
        for(int i = 0; i <= n; i++) parent[i] = i;
    }

    int find(int x) {
        if(parent[x] != x) parent[x] = find(parent[x]);
        return parent[x];
    }

    bool unite(int x, int y) {
        x = find(x); y = find(y);
        if(x == y) return false;
        if(size[x] < size[y]) swap(x, y);
        parent[y] = x;
        size[x] += size[y];
        return true;
    }

    bool same(int x, int y) {
        return find(x) == find(y);
    }
};

// ==================== Segment Tree ====================
struct SegmentTree {
    int n;
    vector<ll> tree, lazy;

    SegmentTree(const vector<ll> &arr) {
        n = arr.size();
        tree.resize(4*n);
        lazy.resize(4*n, 0);
        build(1, 0, n-1, arr);
    }

    void build(int node, int l, int r, const vector<ll> &arr) {
        if(l == r) {
            tree[node] = arr[l];
            return;
        }
        int mid = (l + r) / 2;
        build(node*2, l, mid, arr);
        build(node*2+1, mid+1, r, arr);
        tree[node] = tree[node*2] + tree[node*2+1];
    }

    void update_range(int node, int l, int r, int ql, int qr, ll val) {
        if(lazy[node] != 0) {
            tree[node] += (r - l + 1) * lazy[node];
            if(l != r) {
                lazy[node*2] += lazy[node];
                lazy[node*2+1] += lazy[node];
            }
            lazy[node] = 0;
        }

        if(r < ql || l > qr) return;
        if(ql <= l && r <= qr) {
            tree[node] += (r - l + 1) * val;
            if(l != r) {
                lazy[node*2] += val;
                lazy[node*2+1] += val;
            }
            return;
        }

        int mid = (l + r) / 2;
        update_range(node*2, l, mid, ql, qr, val);
        update_range(node*2+1, mid+1, r, ql, qr, val);
        tree[node] = tree[node*2] + tree[node*2+1];
    }

    ll query_range(int node, int l, int r, int ql, int qr) {
        if(lazy[node] != 0) {
            tree[node] += (r - l + 1) * lazy[node];
            if(l != r) {
                lazy[node*2] += lazy[node];
                lazy[node*2+1] += lazy[node];
            }
            lazy[node] = 0;
        }

        if(r < ql || l > qr) return 0;
        if(ql <= l && r <= qr) return tree[node];

        int mid = (l + r) / 2;
        return query_range(node*2, l, mid, ql, qr) +
               query_range(node*2+1, mid+1, r, ql, qr);
    }

    // Helper functions
    void update(int l, int r, ll val) { update_range(1, 0, n-1, l, r, val); }
    ll query(int l, int r) { return query_range(1, 0, n-1, l, r); }
```

```
};

// ==================== Fenwick Tree ====================
struct FenwickTree {
    int n;
    vector<ll> bit;

    FenwickTree(int size) {
        n = size;
        bit.resize(n+1, 0);
    }

    void update(int idx, ll delta) {
        for(; idx <= n; idx += idx & -idx) {
            bit[idx] += delta;
        }
    }

    ll query(int idx) {
        ll sum = 0;
        for(; idx > 0; idx -= idx & -idx) {
            sum += bit[idx];
        }
        return sum;
    }

    ll range_query(int l, int r) {
        return query(r) - query(l-1);
    }
};

// ==================== Geometry ====================
struct Point {
    ll x, y;
    Point() : x(0), y(0) {}
    Point(ll _x, ll _y) : x(_x), y(_y) {}

    bool operator<(const Point &other) const {
        return tie(x, y) < tie(other.x, other.y);
    }

    Point operator-(const Point &other) const {
        return Point(x - other.x, y - other.y);
    }
```

```
};

ll cross(const Point &a, const Point &b) {
    return a.x * b.y - a.y * b.x;
}

ll dot(const Point &a, const Point &b) {
    return a.x * b.x + a.y * b.y;
}

ll orientation(const Point &a, const Point &b, const Point &c) {
    ll val = cross(b - a, c - a);
    if(val == 0) return 0; // collinear
    return (val > 0) ? 1 : -1; // clockwise or counterclockwise
}

vector<Point> convex_hull(vector<Point> points) {
    int n = points.size();
    if(n <= 1) return points;

    sort(all(points));
    vector<Point> hull;

    // Lower hull
    for(int i = 0; i < n; i++) {
        while(hull.size() >= 2 &&
              orientation(hull[hull.size()-2], hull.back(), points[i]) <= 0) {
            hull.pop_back();
        }
        hull.pb(points[i]);
    }

    // Upper hull
    int lower_size = hull.size();
    for(int i = n-2; i >= 0; i--) {
        while(hull.size() > lower_size &&
              orientation(hull[hull.size()-2], hull.back(), points[i]) <= 0) {
            hull.pop_back();
        }
        hull.pb(points[i]);
    }

    hull.pop_back(); // Remove duplicate starting point
    return hull;
```

```
}

// ==================== String Algorithms ====================
vector<int> kmp(const string &s) {
    int n = s.size();
    vector<int> lps(n, 0);
    for(int i = 1; i < n; i++) {
        int j = lps[i-1];
        while(j > 0 && s[i] != s[j]) j = lps[j-1];
        if(s[i] == s[j]) j++;
        lps[i] = j;
    }
    return lps;
}

vector<int> z_algorithm(const string &s) {
    int n = s.size();
    vector<int> z(n, 0);
    int l = 0, r = 0;
    for(int i = 1; i < n; i++) {
        if(i <= r) z[i] = min(r-i+1, z[i-l]);
        while(i+z[i] < n && s[z[i]] == s[i+z[i]]) z[i]++;
        if(i+z[i]-1 > r) {
            l = i;
            r = i + z[i] - 1;
        }
    }
    return z;
}

// ==================== Game Theory ====================
int mex(unordered_set<int> &s) {
    int m = 0;
    while(s.count(m)) m++;
    return m;
}

// ==================== Useful Functions ====================
bool is_palindrome(const string &s) {
    int n = s.size();
    for(int i = 0; i < n/2; i++) {
        if(s[i] != s[n-i-1]) return false;
    }
    return true;
```

```
}

template<typename T>
void print_vector(const vector<T> &v, bool spaces = true) {
    for(const auto &x : v) {
        cout << x;
        if(spaces) cout << " ";
    }
    if(spaces) cout << endl;
}

// ==================== Main Solve Function ====================
void solve() {
    // Your solution here
    int n;
    cin >> n;
    vector<int> arr(n);
    for(int i = 0; i < n; i++) cin >> arr[i];

    // Example: Find sum
    ll sum = accumulate(all(arr), 0LL);
    cout << sum << endl;
}

// ==================== Main Function ====================
int main() {
    fastIO();

    // Precompute factorials if needed
    // precompute_factorials();

    #ifdef LOCAL
    freopen("input.txt", "r", stdin);
    freopen("output.txt", "w", stdout);
    freopen("error.txt", "w", stderr);
    #endif

    int t = 1;
    cin >> t; // Comment if single test case
    for(int tc = 1; tc <= t; tc++) {
        // cout << "Case " << tc << ": ";
        solve();
    }
    #ifdef LOCAL
    cerr << "Time elapsed: " << 1.0 * clock() / CLOCKS_PER_SEC << " seconds" << endl;
    #endif

    return 0;
}

// ==================== Additional Templates ====================
/*
// Matrix Exponentiation
struct Matrix {
    int n, m;
    vector<vector<ll>> mat;

    Matrix(int _n, int _m) : n(_n), m(_m), mat(_n, vector<ll>(_m, 0)) {}

    Matrix operator*(const Matrix &other) const {
        Matrix result(n, other.m);
        for(int i = 0; i < n; i++) {
            for(int j = 0; j < other.m; j++) {
                for(int k = 0; k < m; k++) {
                    result.mat[i][j] = (result.mat[i][j] + mat[i][k] * other.mat[k][j]) % MOD;
                }
            }
        }
        return result;
    }
};

Matrix mat_pow(Matrix base, ll exp) {
    Matrix result(base.n, base.n);
    for(int i = 0; i < base.n; i++) result.mat[i][i] = 1;

    while(exp > 0) {
        if(exp & 1) result = result * base;
        base = base * base;
        exp >>= 1;
    }
    return result;
}
*/

// ==================== Template Usage Tips ====================
/*
```

1. For problems A-B: Use simple implementations
2. For problem C: Use BFS/DFS or basic DP
3. For problem D: Use Segment Tree or Number Theory
4. For problem E: Use advanced DS or Graph algorithms

Always check constraints first!
Time Complexity Estimation:
- n ≤ 10^5: O(n log n) usually safe
- n ≤ 10^3: O(n^2) usually safe
- n ≤ 20: O(2^n) may work
*/
```

📦 টেমপ্লেটে যা যা আছে:

1. বেসিক অপটিমাইজেশন
- Fast I/O
- GCC optimization pragmas

2. গাণিতিক ফাংশন
- GCD, LCM
- Modular exponentiation
- Sieve of Eratosthenes
- Prime factorization

3. কম্বিনেটরিক্স
- nCr, nPr modulo 1e9+7
- Precomputed factorials

4. গ্রাফ অ্যালগরিদম
- BFS, DFS
- Topological Sort
- Dijkstra's Algorithm
- DSU (Union-Find)

5. ডেটা স্ট্রাকচার
- Segment Tree (with lazy propagation)
- Fenwick Tree (Binary Indexed Tree)

6. জ্যামিতি
- Point operations
- Convex Hull
- Cross and Dot products
```

### **7. স্ট্রিং অ্যালগরিদম**
- KMP (prefix function)
- Z-algorithm

### **8. গেম থিওরি**
- MEX function

## **🚀 ব্যবহারের উপায়:**

1. **কোডটাকে `template.cpp` নামে সেভ করুন**
2. **প্রতিটি কনটেস্টের আগে কপি করুন**
3. **প্রয়োজনমতো ফাংশন কল করুন**
4. **solve() ফাংশনে আপনার সমাধান লিখুন**

## **📌 গুরুত্বপূর্ণ নোট:**

1. **ঢাকা রিজিওনালে সাধারণত 256MB মেমোরি** limit থাকে
2. **স্ট্যাক সাইজ** বাড়াতে `-Wl,--stack,268435456` flag ব্যবহার করুন
3. **লোকাল টেস্টিং** এর জন্য LOCAL define ব্যবহার করুন
4. **Big Integer** লাগলে Python ব্যবহার করুন

## **💡 সমস্যা টাইপ অনুযায়ী কোন টেমপ্লেট ব্যবহার করবেন:**

- **Problem A/B:** Basic math, sorting, implementation
- **Problem C:** Graph (BFS/DFS), simple DP
- **Problem D:** Segment Tree, Number Theory
- **Problem E:** Advanced Graph, Geometry, Game Theory

**শুভকামনা!** 🏆 ঢাকা রিজিওনাল ২০২৫-এ সাফল্য কামনা করছি!