Week 1: Basics & Implementation

Topics: - Input/Output, Loops, Conditionals - Arrays, Strings, Basic Math - Simple sorting

Weekly Tips: - Focus on writing clean, readable code. - Always test edge cases (0, 1, negative numbers, large numbers). - Use online judge IDE or local compiler to verify behavior.

Week 2: Ad-hoc & Simulation

Topics: - Simulation - Ad-hoc logic problems - Greedy basics

Weekly Tips: - Think step by step, simulate processes on paper first. - Carefully read problem constraints to optimize loops. - Greedy approach works if problem guarantees local optimality leads to global optimality.

Week 3: Sorting & Searching

Topics: - Sorting algorithms: QuickSort, MergeSort, STL sort - Binary Search & Ternary Search - Two-pointer technique

Weekly Tips: - Always check if STL sort suffices before implementing manually. - Binary search can be applied to sorted arrays or answer space. - Two-pointer technique is useful for finding pairs, sums, or sliding windows.

Week 4: Strings & Pattern Matching

Topics: - String searching: KMP, Rabin-Karp - Palindromes & substrings - Prefix/Suffix techniques

Weekly Tips: - Understand failure function in KMP for linear-time matching. - Use rolling hash for fast substring comparison. - Practice manipulating strings efficiently with STL.

Week 5: Recursion & Backtracking

Topics: - Recursion basics - Backtracking: N-Queens, subsets, combinations - Depth-First Search (DFS) for combinatorial problems

Weekly Tips: - Draw recursion trees to understand problem flow. - Watch stack usage and avoid unnecessary deep recursion. - Memoization can be applied to optimize repetitive recursive calls.

Week 6: Graph Theory Basics

Topics: - Graph representation: adjacency list & matrix - BFS & DFS traversal - Connected components - Shortest paths (Dijkstra, BFS for unweighted)

Weekly Tips: - Always check graph type: directed, undirected, weighted, unweighted. - Use visited array to avoid revisiting nodes. - For unweighted shortest paths, BFS is sufficient.

Week 7: Dynamic Programming (DP)

Topics: - Introduction to DP: memoization & tabulation - Classic problems: Fibonacci, Knapsack, LIS - Grid DP, state compression

Weekly Tips: - Identify overlapping subproblems and optimal substructure. - Start with recursive solution, then memoize or tabulate. - Practice simple to complex DP to build intuition.

Week 8: Advanced Graph Algorithms

Topics: - Minimum Spanning Trees: Prim, Kruskal - Bellman-Ford for negative weights - Floyd-Warshall for all-pairs shortest paths - Strongly Connected Components (Kosaraju, Tarjan)

Weekly Tips: - MST: Focus on edge selection and cycle prevention. - Bellman-Ford: Detect negative cycles. - Floyd-Warshall: Use DP-like approach for all-pairs shortest path. - SCC: Identify components and condensation graph.

Week 9: Greedy & Interval Problems

Topics: - Activity Selection Problem - Interval Scheduling - Interval Covering - Fractional Knapsack

Weekly Tips: - Always sort intervals by finishing time for scheduling problems. - Greedy works when local optimum leads to global optimum. - Pay attention to edge cases where intervals overlap. - Fractional Knapsack can be solved using sorting by value/weight ratio.

Week 10: Advanced Dynamic Programming

Topics: - DP on Trees - Bitmask DP - Sequence DP with constraints (e.g., subsequences, partitions) - Optimization techniques: prefix sums, cumulative DP

Weekly Tips: - DP on trees: use DFS and store DP for subtrees. - Bitmask DP: useful for problems with small n (<=20) subsets. - Sequence DP: carefully define states and transitions. - Optimize using cumulative sums, monotonic queues when possible.

Week 11: Network Flow & Matching

Topics: - Max Flow (Ford-Fulkerson, Edmonds-Karp) - Min Cut - Bipartite Matching (Hungarian Algorithm, Hopcroft-Karp) - Flow-based problem solving

Weekly Tips: - Max Flow: Understand residual graph and augmenting paths. - Min Cut: Relates to Max Flow by MFMC theorem. - Bipartite Matching: Use flow or DFS-based approaches. - Practice transforming problems into flow networks.

Week 12: Geometry & Computational Geometry

Topics: - Points, Lines, and Vectors - Distances and Angles - Convex Hull (Graham Scan, Andrew's Algorithm) - Polygon Area and Intersection - Line Sweep and Geometric Algorithms

Weekly Tips: - Use structures/classes for points and vectors for clarity. - Pay attention to precision and rounding errors with floating points. - Start with simple geometry: distances, dot/cross product. - Convex hull is fundamental for many polygon problems. - Line sweep technique is useful for intervals and intersections.

Week 13: String Algorithms & Advanced Pattern Matching

Topics: - Suffix Arrays & LCP arrays - Trie (Prefix Tree) - Z-Algorithm for pattern matching - String Hashing (Rabin-Karp) - Aho-Corasick Algorithm

Weekly Tips: - Suffix arrays allow fast substring search and comparison. - Tries are useful for prefix-based problems and autocomplete. - Z-algorithm computes matching prefixes efficiently. - Rolling hash (Rabin-Karp) allows constant-time substring hashing. - Aho-Corasick handles multiple pattern matching efficiently.

Week 14: Advanced Graph Theory & Shortest Paths

Topics: - Dijkstra’s Algorithm with priority queue - Bellman-Ford optimizations - Floyd-Warshall for all-pairs shortest paths - Johnson’s Algorithm for sparse graphs - Shortest path variations: k-shortest paths, negative cycles

Weekly Tips: - Use priority queue (min-heap) for efficient Dijkstra. - Bellman-Ford useful for graphs with negative weights. - Floyd-Warshall computes all-pairs shortest paths in dense graphs. - Johnson’s algorithm combines Dijkstra and reweighting for sparse graphs with negative edges. - Practice different variants and modifications of shortest path problems.

Week 15: Math & Number Theory for ACM-ICPC

Topics: - Modular arithmetic & modular inverse - GCD/LCM and extended Euclidean algorithm - Prime numbers & Sieve of Eratosthenes - Combinatorics: factorial, nCr modulo prime - Fast exponentiation (binary exponentiation) - Number-theoretic functions: totient, modular exponentiation

Weekly Tips: - Modular arithmetic is crucial for large numbers and avoiding overflow. - Sieve is efficient for generating primes up to 1e7. - Extended Euclidean algorithm allows solving linear Diophantine equations. - Fast exponentiation reduces time complexity to O(log n). - Practice combinatorial calculations under modulo for counting problems.

Week 16: Advanced Data Structures

Topics: - Segment Trees (range queries and updates) - Binary Indexed Trees (Fenwick Tree) - Heaps and priority queues - Disjoint Set Union (Union-Find with path compression) - Lazy propagation in segment trees

Weekly Tips: - Segment trees efficiently handle range queries and updates in O(log n). - BITs are simpler and faster for prefix sums. - Union-Find helps with connectivity problems in graphs. - Practice using heaps for priority-based problems and Dijkstra optimization. - Lazy propagation is key for interval updates without degrading performance.

Problem 1: Range Sum Queries

Link: CSES Range Sum Queries II **Difficulty:** Intermediate

C++ Solution with Explanation Comments:

```
#include <bits/stdc++.h>
using namespace std;
struct BIT {
    vector<long long> bit;
    int n;
    BIT(int size){ n=size; bit.assign(n+1,0); }
    void update(int idx, long long val){
        for(; idx<=n; idx += idx&-idx) bit[idx]+=val;
    }
    long long query(int idx){
        long long res=0;
        for(; idx>0; idx -= idx&-idx) res+=bit[idx];
        return res;
    }
    long long range(int l,int r){ return query(r)-query(l-1); }
};
int main(){
```

```
    int n,q; cin>>n>>q;
    BIT tree(n);
    for(int i=1;i<=n;i++){
        long long x; cin>>x; tree.update(i,x);
    }
    while(q--){
        int t,a,b; cin>>t>>a>>b;
        if(t==1) tree.update(a,b); // update
        else cout<<tree.range(a,b)<<"\n"; // query
    }
}
```

Explanation Comments: - Implements BIT (Fenwick Tree) for prefix sum and point updates. - `update` adds value to a position and updates ancestors. - `query` calculates prefix sum efficiently. - `range` computes sum over an interval using prefix sums.

End of Week 16 - Master segment trees, BIT, heaps, and DSU. - Understand when to use each data structure for different problems. - Practice interval updates, connectivity, and priority-based query problems.