### 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.

**Problem 1: Minimum Spanning Tree (Kruskal)** **Link:** [CSES Minimum Spanning Tree](https://cses.fi/problemset/task/1671/) **Difficulty:** Intermediate

**C++ Solution with Explanation Comments:**

#include <bits/stdc++.h>  
using namespace std;  
struct Edge { int u,v,w; };  
int parent[100005];  
int find(int x){ return parent[x]==x?x:parent[x]=find(parent[x]); }  
int main(){  
 int n,m; cin>>n>>m;  
 vector<Edge> edges(m);  
 for(int i=0;i<m;i++) cin>>edges[i].u>>edges[i].v>>edges[i].w;  
 sort(edges.begin(),edges.end(),[](Edge a,Edge b){return a.w<b.w;});  
 for(int i=1;i<=n;i++) parent[i]=i;  
 long long mst=0;  
 for(auto e:edges){  
 int u=find(e.u), v=find(e.v);  
 if(u!=v){ mst+=e.w; parent[u]=v; }  
 }  
 cout<<mst<<endl;  
}

**Explanation Comments:** - Kruskal’s algorithm sorts edges by weight and adds them if they don’t form a cycle. - Union-Find (find) is used to detect cycles efficiently. - Time complexity: O(m log m) due to sorting.

**Problem 2: Bellman-Ford Shortest Path** **Link:** [CSES Shortest Routes I](https://cses.fi/problemset/task/1671/) **Difficulty:** Intermediate

**C++ Solution with Explanation Comments:**

#include <bits/stdc++.h>  
using namespace std;  
struct Edge{int u,v,w;};  
int main(){  
 int n,m; cin>>n>>m;  
 vector<Edge> edges(m);  
 for(int i=0;i<m;i++) cin>>edges[i].u>>edges[i].v>>edges[i].w;  
 vector<long long> dist(n+1,LLONG\_MAX);  
 dist[1]=0;  
 for(int i=1;i<n;i++){  
 for(auto e:edges){  
 if(dist[e.u]!=LLONG\_MAX && dist[e.u]+e.w<dist[e.v]) dist[e.v]=dist[e.u]+e.w;  
 }  
 }  
 for(int i=1;i<=n;i++) cout<<dist[i]<<" ";  
}

**Explanation Comments:** - Bellman-Ford handles graphs with negative weights. - Relax all edges n-1 times to find shortest distances. - Can detect negative cycles if relaxation is still possible after n-1 iterations.

**End of Week 8** - Practice MST, shortest paths, and SCC problems. - Understand differences between graph traversal algorithms and when to use each. - Pay attention to cycle detection and negative weight handling.