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.

Problem 1: DP on Trees

Link: CSES Tree Distances **Difficulty:** Advanced

C++ Solution with Explanation Comments:

```
#include <bits/stdc++.h>
using namespace std;
vector<vector<int>> adj;
vector<int> dp;
void dfs(int u, int p){
    dp[u]=0;
    for(int v:adj[u]) if(v!=p){
        dfs(v,u);
        dp[u]+=dp[v]+1; // count distance to children
    }
}
int main(){
    int n; cin>>n;
    adj.resize(n+1); dp.resize(n+1);
    for(int i=0;i<n-1;i++){ int u,v; cin>>u>>v; adj[u].push_back(v);
adj[v].push_back(u); }
    dfs(1,0);
    for(int i=1;i<=n;i++) cout<<dp[i]<<" ";
}
```

Explanation Comments: - DFS to calculate sum of distances in subtrees. - DP[u] stores cumulative info of children. - Classic tree DP pattern.

Problem 2: Bitmask DP

Link: CSES Travelling Salesman Problem **Difficulty:** Advanced

C++ Solution with Explanation Comments:

```
#include <bits/stdc++.h>
using namespace std;
int main(){
    int n; cin>>n;
    vector<vector<long long>> cost(n,vector<long long>(n));
    for(int i=0;i<n;i++) for(int j=0;j<n;j++) cin>>cost[i][j];
    int N=1<<n;
    vector<vector<long long>> dp(N,vector<long long>(n,1e18));
    dp[1][0]=0;
    for(int mask=1;mask<N;mask++){
        for(int u=0;u<n;u++){
            if(!(mask&(1<<u))) continue;
            for(int v=0;v<n;v++){
                if(mask&(1<<v)) continue;
                dp[mask|(1<<v)][v]=min(dp[mask|(1<<v)][v], dp[mask][u]+cost[u]
[v]);
            }
        }
    }
    long long ans=1e18;
    for(int i=1;i<n;i++) ans=min(ans, dp[N-1][i]+cost[i][0]);
    cout<<ans<<endl;
}
```

Explanation Comments: - DP[mask][u] stores min cost visiting subset mask ending at u. - Iterate over all subsets, add new nodes. - Bitmask allows subset representation for TSP-like problems.

Problem 3: Sequence DP

Link: CSES Increasing Subsequence II **Difficulty:** Advanced

C++ Solution with Explanation Comments:

```
#include <bits/stdc++.h>
using namespace std;
int main(){
    int n; cin>>n;
    vector<long long> a(n);
    for(int i=0;i<n;i++) cin>>a[i];
    vector<long long> dp;
    for(int i=0;i<n;i++){
        auto it=lower_bound(dp.begin(),dp.end(),a[i]);
        if(it==dp.end()) dp.push_back(a[i]);
        else *it=a[i];
    }
    cout<<dp.size()<<endl;
}
```

Explanation Comments: - Efficient LIS using binary search. - DP stores the smallest ending element for increasing subsequences of each length. - O(n log n) solution for large sequences.

End of Week 10 - Focus on tree and bitmask DP. - Sequence DP requires careful state management. - Practice both recursive and iterative DP implementations.