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.

Problem 1: Convex Hull

Link: CSES Convex Hull **Difficulty:** Advanced

C++ Solution with Explanation Comments:

```
#include <bits/stdc++.h>
using namespace std;
struct Point { long long x,y; };
long long cross(Point O, Point A, Point B){
    return (A.x-O.x)*(B.y-O.y)-(A.y-O.y)*(B.x-O.x);
}
int main(){
    int n; cin>>n;
    vector<Point> P(n);
    for(int i=0;i<n;i++) cin>>P[i].x>>P[i].y;
    sort(P.begin(),P.end(),[](Point a, Point b){ return a.x==b.x?
a.y<b.y:a.x<b.x;});
    vector<Point> H;
    // Lower hull
    for(int i=0;i<n;i++){
        while(H.size()>=2 && cross(H[H.size()-2],H[H.size()-1],P[i])<=0)
H.pop_back();
        H.push_back(P[i]);
    }
```

```
    // Upper hull
    int t=H.size()+1;
    for(int i=n-2;i>=0;i--){
        while(H.size()>=t && cross(H[H.size()-2],H[H.size()-1],P[i])<=0)
H.pop_back();
        H.push_back(P[i]);
    }
    H.pop_back(); // remove duplicate
    cout<<H.size()<<endl;
    for(auto p:H) cout<<p.x<<" "<<p.y<<endl;
}
```

Explanation Comments: - Implements Andrew's monotone chain for convex hull. - Cross product used to determine turn direction. - Constructs lower and upper hull efficiently in O(n log n).

Problem 2: Polygon Area

Link: CSES Polygon Area **Difficulty:** Intermediate

C++ Solution with Explanation Comments:

```
#include <bits/stdc++.h>
using namespace std;
struct Point { long long x,y; };
long long polygonArea(vector<Point>& P){
    long long area=0;
    int n=P.size();
    for(int i=0;i<n;i++){
        int j=(i+1)%n;
        area += (P[i].x*P[j].y - P[j].x*P[i].y);
    }
    return abs(area);
}
int main(){
    int n; cin>>n;
    vector<Point> P(n);
    for(int i=0;i<n;i++) cin>>P[i].x>>P[i].y;
    cout<<polygonArea(P)/2<<endl;
}
```

Explanation Comments: - Uses shoelace formula for polygon area. - Works for convex and concave polygons. - Handles integer coordinates precisely.

End of Week 12 - Master basic geometry operations: distance, angle, cross/dot products. - Practice convex hull and polygon operations. - Line sweep and intersection detection are key for complex geometric problems.