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.

Problem 1: Division

Link: UVa 725 **Difficulty:** Intermediate

C++ Solution with Explanation Comments:

```
#include <iostream>
#include <iomanip>
using namespace std;

int main() {
    int N;
    while (cin >> N && N >= 0) {
        for (int num = 1234; num <= 98765; num++) {
            int denom = num * N;
            if (denom > 98765) break;
            int digits[10] = {0};
            int tempNum = num, tempDenom = denom;

            // Count digits in numerator
            digits[tempNum / 10000]++;
            tempNum %= 10000;
            digits[tempNum / 1000]++;
            tempNum %= 1000;
            digits[tempNum / 100]++;
            tempNum %= 100;
            digits[tempNum / 10]++;
            tempNum %= 10;
            digits[tempNum]++;

            // Count digits in denominator
            for (int i = 0; i < 5; i++) {
                digits[tempDenom % 10]++;
                tempDenom /= 10;
            }

            // Check if all digits 0-9 appear exactly once
            bool valid = true;
            for (int i = 0; i <= 9; i++) {
                if (digits[i] != 1) { valid = false; break; }
            }

            if (valid) {
                cout << setw(5) << denom << " / " << setw(5) << num << " = " <<
N << endl;
            }
        }
        cout << endl;
    }
    return 0;
}
```

Explanation Comments: - Uses recursion-like iteration to explore all valid pairs. - Counts digits to satisfy unique 0-9 usage. - Demonstrates careful state management similar to backtracking.

Problem 2: Walking on the Safe Side

Link: UVa 825 **Difficulty:** Intermediate

C++ Solution with Explanation Comments:

```cpp
#include <iostream>
#include <vector>
using namespace std;

int countPaths(int x, int y, vector<vector<int>>& grid, vector<vector<int>>&
memo) {
    if (x < 0 || y < 0) return 0;
    if (x == 0 && y == 0) return 1;
    if (memo[x][y] != -1) return memo[x][y];
    int paths = countPaths(x-1, y, grid, memo) + countPaths(x, y-1, grid, memo);
    return memo[x][y] = paths;
}

int main() {
    int n, m;
    cin >> n >> m;
    vector<vector<int>> grid(n, vector<int>(m, 0));
    vector<vector<int>> memo(n, vector<int>(m, -1));
    cout << countPaths(n-1, m-1, grid, memo) << endl;
    return 0;
}
```

Explanation Comments: - Recursive DFS with memoization to count paths in a grid. - Avoids recomputation with `memo` array. - Demonstrates recursion and dynamic programming combination.

Problem 3: Sudoku

Link: Kattis Sudoku **Difficulty:** Intermediate

C++ Solution with Explanation Comments:

```cpp
#include <iostream>
#include <vector>
using namespace std;

bool isSafe(vector<vector<int>>& board, int row, int col, int num) {
    for (int x = 0; x < 9; x++) {
        if (board[row][x] == num || board[x][col] == num) return false;
    }
    int startRow = row - row%3, startCol = col - col%3;
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            if (board[startRow+i][startCol+j] == num) return false;
        }
    }
    return true;
}

bool solveSudoku(vector<vector<int>>& board, int row, int col) {
    if (row == 9) return true;
    if (col == 9) return solveSudoku(board, row+1, 0);
    if (board[row][col] != 0) return solveSudoku(board, row, col+1);
    for (int num = 1; num <= 9; num++) {
        if (isSafe(board, row, col, num)) {
            board[row][col] = num;
            if (solveSudoku(board, row, col+1)) return true;
            board[row][col] = 0;
        }
    }
    return false;
}

int main() {
    vector<vector<int>> board(9, vector<int>(9));
    for (int i = 0; i < 9; i++)
        for (int j = 0; j < 9; j++)
            cin >> board[i][j];
    if (solveSudoku(board, 0, 0)) {
        for (int i = 0; i < 9; i++) {
            for (int j = 0; j < 9; j++) {
                cout << board[i][j] << " ";
            }
            cout << endl;
        }
    }
    return 0;
}
```

Explanation Comments: - Uses recursive backtracking to fill the Sudoku board. - `isSafe` checks row, column, and 3x3 box constraints. - Demonstrates DFS and constraint checking in combinatorial search.

End of Week 5 - Practice recursion trees and memoization. - Apply backtracking to combinatorial problems like puzzles, subsets, and pathfinding. - Understand pruning unnecessary branches to optimize recursion.