Algo Homework-7

Q1->

A1-> We need 2D-array and hashmap data structures.
We need to identify subproblems as function depends on 2 parameters: n and m. We need to store results for every unique pair (n, m).
Then, we need to choose data structures ('2D array' for small values of n and m and 'hashmap' for large ranges of n & m.
Then, compute solutions for smaller subproblems first (eg. when n=1 or m=1). Use these results to solve larger problems iteratively.
Then, store computed results in a table or hashmap during recursion. Check the table before performing a recursive call to avoid redundant computation.

Q2->

A2-> Input: n: positive integer
Input: m: positive integer
Input: memo: a map to store computed results.
Algorithm: MysteryRecursion
if n=1 and m=1 then
    return 1
else if (n, m) exists in memo then
    return memo[(n, m)]
else if n=1 then
    result ← m · MysteryRecursion(n, ⌊m/2⌋, memo)
else if m=1 then
    result ← n · MysteryRecursion(⌊n/2⌋, m, memo)
else
    result ← n · MysteryRecursion(⌊n/2⌋, m, memo) + m · MysteryRecursion(n, ⌊m/2⌋,
    memo[(n, m)] ← result
    return result end

Q3→

A3→ We can use nested 'for' loops to populate a 2D table (dp) where each cell dp[i][j] represents the result of mysteryRecursion(i, j). The loops traverse all combinations of n & m, starting from smallest subproblems & building upto desired dp[n][m].

For loops Explaination:-

Outer loop (for n):

Outer loop iterates through all values of n, starting from 1 to target 'n'. This ensures that smaller subproblems are solved before larger ones. (for i from 1 to n)

Inner loop (for m):-

For each value of n, inner loop iterates through all values of m, starting from 1 to m. This processes all combinations of (n, m) in increasing order. (for j from 1 to m).

Logic Inside loops:-

Base Case: If i=1 and j=1, set dp[1][1] = 1.

Case i=1: Use formula dp[1][j] = j · dp[1][⌊j/2⌋]

Case j=1: Use formula dp[i][1] = i · dp[⌊i/2⌋][1].

General Case: Use formula dp[i][j] = i · dp[⌊i/2⌋][j] + j · dp[i][⌊j/2⌋]