**Day 3: Recursion - Core Patterns**

**🧮 1. Factorial**

**Problem Statement:**  
Calculate the total number of ways to arrange n distinct items — useful in probability models and permutation tasks.  
**Sample Input:**  
n = 5  
**Expected Output:**  
120  
*Explanation: 5! = 5 × 4 × 3 × 2 × 1*

**🐇 2. Fibonacci**

**Problem Statement:**  
Model population growth, where each generation gives rise to the next according to Fibonacci patterns — applicable in biology and finance.  
**Sample Input:**  
n = 6  
**Expected Output:**  
8  
*Explanation: Sequence is [0,1,1,2,3,5,8] — 6th value is 8.*

**🔋 3. Power(x, n)**

**Problem Statement:**  
Simulate exponential growth, like compounding interest or energy decay — commonly used in physics and finance calculators.  
**Sample Input:**  
x = 2, n = 10  
**Expected Output:**  
1024

**🧷 4. Generate Parentheses**

**Problem Statement:**  
Design a code formatter to wrap operations in valid parentheses — useful in compiler design and syntax generation.  
**Sample Input:**  
n = 3  
**Expected Output:**  
["((()))","(()())","(())()","()(())","()()()"]

**🔄 5. Permutations**

**Problem Statement:**  
Generate all ways to assign tasks to employees or shuffle a deck — applicable in scheduling and simulations.  
**Sample Input:**  
nums = [1,2,3]  
**Expected Output:**  
[[1,2,3],[1,3,2],[2,1,3],[2,3,1],[3,1,2],[3,2,1]]

**🔁 6. Permutations II (With Duplicates)**

**Problem Statement:**  
Create unique seating arrangements even if some guests share names — used in event planning and layout logic.  
**Sample Input:**  
nums = [1,1,2]  
**Expected Output:**  
[[1,1,2],[1,2,1],[2,1,1]]

**📦 7. Subsets**

**Problem Statement:**  
List all possible combinations of selected features in a product — essential in configuration tools or split testing.  
**Sample Input:**  
nums = [1,2,3]  
**Expected Output:**  
[[],[1],[2],[1,2],[3],[1,3],[2,3],[1,2,3]]

**📦 8. Subsets II (With Duplicates)**

**Problem Statement:**  
Design feature bundles while avoiding duplicate combinations — helpful in product variants and UI controls.  
**Sample Input:**  
nums = [1,2,2]  
**Expected Output:**  
[[],[1],[1,2],[1,2,2],[2],[2,2]]

**🎯 9. Combination Sum**

**Problem Statement:**  
Find all ways to purchase items where total cost exactly meets budget — allows repeated use of same item.  
**Sample Input:**  
candidates = [2,3,6,7], target = 7  
**Expected Output:**  
[[2,2,3],[7]]

**🛍️ 10. Combination Sum II**

**Problem Statement:**  
Identify all unique order combinations to match a budget — with each item used only once.  
**Sample Input:**  
candidates = [10,1,2,7,6,1,5], target = 8  
**Expected Output:**  
[[1,1,6],[1,2,5],[1,7],[2,6]]

**☎️ 11. Letter Combinations of a Phone Number**

**Problem Statement:**  
Simulate predictive text on keypads — generate all possible word combinations based on digit inputs, just like T9 on old phones.  
**Sample Input:**  
digits = "23"  
**Expected Output:**  
["ad", "ae", "af", "bd", "be", "bf", "cd", "ce", "cf"]

**🔤 12. Word Search**

**Problem Statement:**  
Implement a puzzle solver that scans a grid to detect if a hidden word can be traced through adjacent cells — great for gaming and educational apps.  
**Sample Input:**

board = [

["A","B","C","E"],

["S","F","C","S"],

["A","D","E","E"]

]

word = "ABCCED"

**Expected Output:**  
True

**🪞 13. Palindrome Partitioning**

**Problem Statement:**  
Split a word into all possible segments where each piece is a palindrome — helpful in text mining and linguistic pattern detection.  
**Sample Input:**  
s = "aab"  
**Expected Output:**  
[["a","a","b"], ["aa","b"]]

**👑 14. N-Queens**

**Problem Statement:**  
Design seating charts or architectural layouts where no two agents (queens) interfere — like conflict-free drone deployments or surveillance grids.  
**Sample Input:**  
n = 4  
**Expected Output:**  
2  
*Explanation: Two distinct ways to place queens without conflict.*

**🧠 15. Sudoku Solver**

**Problem Statement:**  
Create a game engine or automation tool that completes a Sudoku puzzle based on existing clues — key for educational tools or puzzle generators.  
**Sample Input:** *(simplified for context)*  
board = 9×9 grid with some "." and numbers  
**Expected Output:**  
Modified board filled with digits that obey Sudoku rules.

**🧺 16. Flatten Nested List Iterator**

**Problem Statement:**  
Build a content feed that unfolds nested folders, playlists, or collections recursively — used in file systems and media apps.  
**Sample Input:**  
[[1,1], 2, [1,1]]  
**Expected Output:**  
[1, 1, 2, 1, 1]

**⚙️ 17. Gray Code**

**Problem Statement:**  
Generate signal encoding sequences where only one bit changes at a time — crucial in hardware circuits and error-correcting protocols.  
**Sample Input:**  
n = 2  
**Expected Output:**  
[0, 1, 3, 2]

**🪜 18. Climbing Stairs**

**Problem Statement:**  
Count different ways a person (or algorithm) can climb a staircase using 1- or 2-step moves — helpful in movement simulations or health apps.  
**Sample Input:**  
n = 5  
**Expected Output:**  
8

**🌲 19. Recursive Tree Traversal**

**Problem Statement:**  
Navigate a decision tree, like a chatbot or flow chart, to understand or manipulate branching logic — used in compilers and diagnostics.  
**Sample Input:**  
Binary tree:

1

\

2

/

3

**Expected Output:**

* Inorder: [1, 3, 2]
* Preorder: [1, 2, 3]
* Postorder: [3, 2, 1]

**💰 20. Coin Change (Memoization & DP)**

**Problem Statement:**  
Build an ATM or change dispenser to return the least number of coins to reach a requested amount — with performance optimization for scale.  
**Sample Input:** coins = [1,3,4], amount = 6  
**Expected Output:**  
2  
\_Explanation: [3 + 3] or [2 + 4] depending on coin options.