Phase 1: Python Fundamentals for DSA

Before diving into complex structures, ensure you're comfortable with these Python essentials. They are the building blocks you'll use constantly.

- **Core Syntax:** Variables, loops (`for`, `while`), conditional statements (`if-elif-else`), functions.
- **Built-in Data Structures:**
  - **Lists:** `.append()`, `.pop()`, `.sort()`, slicing, list comprehensions.
  - **Dictionaries:** `.get()`, `.keys()`, `.values()`, `.items()`, iteration.
  - **Sets:** Unique elements, fast lookups using `in`.
  - **Strings:** Manipulation, slicing, `.join()`, `.split()`.
- **Big O Notation:** Analyze time and space complexity to evaluate algorithm efficiency.
- **Recursion:** Functions that call themselves. Crucial for tree traversals, backtracking, etc.

Phase 2: Core Data Structures & Algorithms

Learn these topics in the order presented. Each builds upon the previous one.

1. Arrays & Hashing

- **Concepts:** Static vs. dynamic arrays, dictionary/set for efficient lookups.
- **Algorithms:**
  - **Two Pointers**
  - **Sliding Window**
- **Practice Problems:**
  - Two Sum (LeetCode #1)
  - Contains Duplicate (LeetCode #217)
  - Valid Anagram (LeetCode #242)
  - Group Anagrams (LeetCode #49)
  - Maximum Subarray (LeetCode #53)
  - Container With Most Water (LeetCode #11)

2. Stacks & Queues

- **Concepts:** LIFO (Stack), FIFO (Queue), use `list` or `collections.deque`.
- **Practice Problems:**
  - Valid Parentheses (LeetCode #20)
  - Min Stack (LeetCode #155)
  - Implement Queue using Stacks (LeetCode #232)
  - Daily Temperatures (LeetCode #739)

3. Linked Lists

- **Concepts:** Singly/Doubly Linked List, fast & slow pointers (Floyd's algorithm).
- **Practice Problems:**
  - Reverse Linked List (LeetCode #206)
  - Merge Two Sorted Lists (LeetCode #21)
  - Linked List Cycle (LeetCode #141)
  - Remove Nth Node From End of List (LeetCode #19)
  - Reorder List (LeetCode #143)

4. Binary Search

- **Concepts:** Divide and conquer, search spaces beyond arrays.
- **Practice Problems:**
  - Binary Search (LeetCode #704)
  - Search a 2D Matrix (LeetCode #74)
  - Find Minimum in Rotated Sorted Array (LeetCode #153)
  - Search in Rotated Sorted Array (LeetCode #33)

5. Trees

- **Concepts:**
  - Binary Tree
  - Binary Search Tree (BST)
  - Traversals: DFS (Inorder, Preorder, Postorder), BFS (Level Order)
- **Practice Problems:**
  - Invert Binary Tree (LeetCode #226)
  - Maximum Depth of Binary Tree (LeetCode #104)
  - Same Tree (LeetCode #100)
  - Lowest Common Ancestor of BST (LeetCode #235)
  - Binary Tree Level Order Traversal (LeetCode #102)
  - Validate BST (LeetCode #98)

6. Heaps (Priority Queues)

- **Concepts:** Min-Heap, Max-Heap, `heapify`, `heappush`, `heappop` using `heapq`.
- **Practice Problems:**
  - Kth Largest Element in a Stream (LeetCode #703)
  - Last Stone Weight (LeetCode #1046)
  - Kth Largest Element in an Array (LeetCode #215)
  - Top K Frequent Elements (LeetCode #347)

7. Backtracking

- **Concepts:** "Choose, explore, unchoose" pattern. Recursive brute-force with pruning.
- **Practice Problems:**
  - Subsets (LeetCode #78)
  - Combination Sum (LeetCode #39)

  - Permutations (LeetCode #46)
  - Word Search (LeetCode #79)

8. Graphs

- **Concepts:**
  - Representation: Adjacency List vs. Matrix
  - Traversals: BFS (shortest path), DFS (pathfinding, cycle detection)
- **Practice Problems:**
  - Number of Islands (LeetCode #200)
  - Clone Graph (LeetCode #133)
  - Course Schedule (LeetCode #207)
  - Pacific Atlantic Water Flow (LeetCode #417)

9. Dynamic Programming (DP) - Introduction

- **Concepts:** Overlapping subproblems, optimal substructure, memoization (top-down), tabulation (bottom-up).
- **Practice Problems:**
  - Climbing Stairs (LeetCode #70)
  - Coin Change (LeetCode #322)
  - Longest Increasing Subsequence (LeetCode #300)
  - House Robber (LeetCode #198)