**Notes: Key Coding Patterns for Problem Solving**

1. **Sliding Window Pattern**
   * **Description**: Used to process a series of data elements (e.g., lists, strings) by looking at smaller parts (windows) that slide one step at a time.
   * **When to Use**: To find a subset of elements in a linear data structure (array, string, linked list) that satisfies a given condition (e.g., longest or shortest substring/subarray).
   * **Example Problem**: "Longest Substring with K Unique Characters" (LeetCode).
2. **Subset Pattern**
   * **Description**: Used to find all possible combinations of elements from a given set. Repetitions may or may not be allowed.
   * **When to Use**: When you need to explore all possible arrangements of elements from a set.
   * **Example Problem**: Permutations problem on LeetCode.
   * **Approach**: Build subsets iteratively, level by level, similar to BFS.
3. **Modified Binary Search Pattern**
   * **Description**: The core idea of binary search remains the same (divide the search space in half repeatedly) but is adjusted to solve specific problems.
   * **When to Use**: When dealing with sorted arrays that may have been rotated or contain duplicates.
   * **Example Problem**: "Search in Rotated Sorted Array" (LeetCode).
   * **Improvement Tip**: Understand the core binary search algorithm well and practice with Python’s bisect module.
4. **Top-K Elements Pattern**
   * **Description**: Used to select the K most important elements from a larger dataset.
   * **When to Use**: When the problem asks for the top-ranking elements from a dataset.
   * **Example Problem**: Finding the Kth largest number in an array.
   * **Approach**: Use a heap to efficiently track and remove the smallest of the largest K elements encountered.
5. **Binary Tree DFS Pattern**
   * **Description**: Depth-First Search (DFS) for binary trees explores nodes one branch at a time.
   * **When to Use**: To explore a binary tree deeply, going down each branch before backtracking.
   * **Example Problem**: "Maximum Depth of Binary Tree" (LeetCode).
   * **Approach**: Use recursion to explore left and right subtrees, updating the maximum depth when a new deeper node is encountered.
6. **Topological Sort**
   * **Description**: Used to arrange elements in a specific order based on their dependencies (often in directed acyclic graphs).
   * **When to Use**: When there are prerequisites or dependency chains between elements.
   * **Example Problem**: "Course Schedule" (LeetCode).
7. **Binary Tree BFS Pattern**
   * **Description**: Breadth-First Search (BFS) explores all nodes at the same level first, using a queue to keep track.
   * **When to Use**: When you need to explore nodes level by level.
   * **Example Problem**: "Level Order Reversal of a Binary Tree" (LeetCode).
   * **Approach**: Use a queue to manage the nodes, process them, and then add their children to the queue.
8. **Two-Pointer Pattern**
   * **Description**: Used to iterate through a sorted array with two pointers, typically for efficient one-pass solutions.
   * **When to Use**: For problems involving a sorted array where you can narrow down possibilities with two pointers.
   * **Example Problem**: "Two Sum" (LeetCode).
   * **Approach**: Start one pointer at the beginning and the other at the end of the array. Move pointers toward each other based on the sum comparison with the target.

**Final Advice:**

* To master these patterns, it's essential to have a strong understanding of data structures and algorithms (DSA).
* Practice by solving problems from LeetCode that fit these patterns.
* Consider signing up for a free DSA crash course to strengthen your foundational knowledge.

# Things to study for

String

Array

Hashmap

recursion

Heap

2 pointers

Linked List

Binary search

Sliding windows pattern

Subset pattern

Time complexity

2D arrays