Coding Patterns: A Cheat Sheet

Here is a brief description of all the coding patterns discussed in this course:

1. Pattern: Two Pointers

Description: This method uses two pointers to traverse an array or a list from different ends or directions.

Usage: It's particularly useful for ordered data structures, where we can make intelligent decisions based on the position of the pointers.

Problems: 'Pair with Target Sum', 'Remove Duplicates', 'Squaring a Sorted Array'.

2. Pattern: Island (Matrix Traversal)

Description: It involves traversing a matrix to find 'islands' or contiguous groups of elements.

Usage: It's generally used in grid-based problems, especially when we need to group connected elements together.

Problems: 'Number of Islands', 'Max Area of Island', 'Flood Fill'.

3. Pattern: Fast & Slow Pointers

Description: In this method, two pointers move at different speeds in a data structure.

Usage: It is commonly used to detect cycles in a structure, find middle elements, or to solve other specific problems related to linked lists.

Problems: 'LinkedList Cycle', 'Middle of the LinkedList', 'Palindrome LinkedList'.

4. Pattern: Sliding Window

Description: This pattern involves creating a 'window' into the data structure and then moving that window around to gather specific information.

Problems: 'Maximum Sum Subarray of Size K', 'Smallest Subarray with a given sum', 'Longest Substring with K Distinct Characters'.

5. Pattern: Merge Intervals

Description: This pattern involves merging overlapping intervals.

Usage: Often used in problems involving time intervals, ranges, or sequences.

Problems: 'Merge Intervals', 'Insert Interval', 'Intervals Intersection'.

6. Pattern: Cyclic Sort

Description: This pattern involves sorting an array containing numbers in a given range.

Usage: It's useful in situations where the data involves a finite range of natural numbers.

Problems: 'Cyclic Sort', 'Find the Missing Number', 'Find all Duplicates'.

7. Pattern: In-place Reversal of a Linked List

Description: This pattern involves reversing elements of a linked list in-place.

Usage: It's generally used when reversing a sequence without using extra space.

Problems: 'Reverse a LinkedList', 'Reverse a Sub-list', 'Reverse Every K-element Sub-list'.

8. Pattern: Tree Breadth First Search

Description: This pattern involves level-by-level traversal of a tree.

Usage: It's used when we need to traverse a tree or graph in a level-by-level (breadth-first) manner.

Problems: 'Level Order Traversal', 'Reverse Level Order Traversal', 'Zigzag Traversal'.

9. Pattern: Tree Depth First Search

Usage: It's used when you need to search deeper into a tree/graph first before going across.

Problems: 'Binary Tree Path Sum', 'All Paths for a Sum', 'Count Paths for a Sum'.

10. Pattern: Two Heaps

Description: This pattern involves using two heaps to divide a set of numbers into two parts.

Usage: It's useful when you need to find median numbers in a sequence, or other similar problems.

Problems: 'Find the Median of a Number Stream', 'Sliding Window Median', 'Maximize Capital'.

11. Pattern: Subsets

Description: This pattern involves generating all subsets of a set.

Usage: It's helpful for solving problems that require exploring all subsets of a given set.

Problems: 'Subsets', 'Subsets With Duplicates', 'Permutations'.

12. Pattern: Modified Binary Search

Description: This is a tweaked version of the binary search algorithm.

Usage: It's used when a simple binary search isn't sufficient, like finding a number in a bitonic array.

Problems: 'Order-agnostic Binary Search', 'Ceiling of a Number', 'Next Letter'.

13. Pattern: Top 'K' Elements

Description: This pattern is used to find the top 'k' elements among a certain category.

Usage: It's commonly used in problems involving sorting, searching, and in heap data structures.

14. Pattern: Bitwise XOR

Description: This pattern involves the use of Bitwise XOR to solve various array-based problems.

Usage: It's used when we need to manipulate and compare bits directly.

Problems: 'Single Number', 'Two Single Numbers', 'Complement of Base 10 Number'.

15. Pattern: Backtracking

Description: This pattern involves exploring all possible solutions and then backtracking to correct the course whenever you're on the wrong path.

Usage: It's typically used for solving complex combinatorial problems, puzzles, and games.

Problems: 'Sudoku Solver', 'N-Queens', 'Generate Parentheses'.

16. Pattern: 0/1 Knapsack (Dynamic Programming)

Description: This pattern deals with problems where items have different values and weights, and we need to determine the maximum value we can carry.

Usage: It's typically used in optimization problems, especially those involving physical constraints.

Problems: '0/1 Knapsack', 'Equal Subset Sum Partition', 'Subset Sum'.

17. Pattern: Topological Sort (Graph)

Description: This pattern involves sorting nodes in a directed graph in a specific order where the preceding node comes before the following node.

Usage: It's used for scheduling problems and in scenarios where order needs to be imposed on how you process nodes.

Problems: 'Task Scheduling Order', 'All Tasks Scheduling Orders', 'Alien Dictionary'.

Description: This pattern involves merging 'k' sorted lists.

Usage: It's typically used in problems involving lists, where merging is required.

Problems: 'Merge K Sorted Lists', 'Kth Smallest Number in M Sorted Lists', 'Smallest Number Range'.

19. Pattern: Monotonic Stack

Description: This pattern involves using a stack to maintain a monotonic (either entirely non-increasing or non-decreasing) order of elements.

Usage: It's often used for solving problems where you need to find the next greater or smaller elements.

Problems: 'Next Greater Element', 'Next Smaller Element', 'Largest Rectangle in Histogram'.

20. Pattern: Multi-threaded

Description: This pattern involves designing algorithms that can execute multiple threads in parallel.

Usage: It's used in situations where a task can be divided into independent sub-tasks that can execute concurrently.

Problems: 'Invert Binary Tree', 'Binary Search Tree Iterator', 'Same Tree'.

21. Pattern: Union Find

Description: Union Find, also known as Disjoint Set Union (DSU), is a data structure that keeps track of a partition of a set into disjoint subsets.

Usage: This pattern is particularly useful for problems where we need to find whether 2 elements belong to the same group or need to solve connectivity-related problems in a graph or tree.

Problems: 'Graph Redundant Connection', 'Number of Provinces', 'Is Graph Bipartite'.