LC 120 hours

Prefix Sum

When to use: Use the prefix sum technique when you need to compute the cumulative sum of elements in an array efficiently, especially when you have multiple queries asking for the sum of elements between different indices. It helps reduce the time complexity from O(n) per query to O(1) after preprocessing.

- **Easy**
  - 1480. Running Sum of 1d Array
  - 724. Find Pivot Index
  - 303. Range Sum Query - Immutable
- **Medium**
  - 560. Subarray Sum Equals K
  - 525. Contiguous Array
  - 523. Continuous Subarray Sum
  - 930. Binary Subarrays With Sum
  - 974. Subarray Sums Divisible by K
- **Hard**
  - 327. Count of Range Sum
  - 862. Shortest Subarray with Sum at Least K
  - 1546. Maximum Number of Non-Overlapping Subarrays With Sum Equals Target

---

Two Pointers

When to use: Use the two pointers technique when you need to process elements in a sequence from both ends or when you're looking for pairs or triplets that meet certain criteria in a sorted array. It's effective for problems where you can reduce the time complexity by moving two pointers towards each other based on certain conditions.

- **Easy**
  - 125. Valid Palindrome
  - 167. Two Sum II - Input Array Is Sorted
  - 344. Reverse String
- **Medium**
  - 15. 3Sum
  - 16. 3Sum Closest
  - 18. 4Sum
  - 259. 3Sum Smaller
- **Hard**
  - 42. Trapping Rain Water

    - 727. Minimum Window Subsequence

Sliding Window

When to use: Use the sliding window technique when you need to find a subarray or substring that satisfies certain conditions within an array or string. It's particularly useful for problems involving contiguous sequences and when you need to optimize over all possible windows (fixed or variable size).

- **Easy**
    - 219. Contains Duplicate II
    - 643. Maximum Average Subarray I
- **Medium**
    - 3. Longest Substring Without Repeating Characters
    - 438. Find All Anagrams in a String
    - 567. Permutation in String
    - 424. Longest Repeating Character Replacement
    - 209. Minimum Size Subarray Sum
- **Hard**
    - 76. Minimum Window Substring
    - 239. Sliding Window Maximum
    - 480. Sliding Window Median

Fast and Slow Pointers

When to use: Use the fast and slow pointers technique when you need to detect cycles in linked lists or arrays, find the middle of a linked list, or determine the starting point of a cycle. It's effective for problems where elements are traversed at different speeds to meet certain conditions.

- **Easy**
    - 141. Linked List Cycle
    - 876. Middle of the Linked List
- **Medium**
    - 142. Linked List Cycle II
    - 287. Find the Duplicate Number
    - 202. Happy Number
- **Hard**
    - 457. Circular Array Loop

Linked List In-place Reversal

When to use: Use linked list in-place reversal when you need to reverse a linked list or a portion of it without using extra space. It's crucial for problems that require modifying the linked list structure directly, such as reversing nodes in groups or rearranging nodes based on certain criteria.

- **Easy**
  - 206. Reverse Linked List
- **Medium**
  - 92. Reverse Linked List II
  - 24. Swap Nodes in Pairs
  - 328. Odd Even Linked List
- **Hard**
  - 25. Reverse Nodes in k-Group
  - 234. Palindrome Linked List

Monotonic Stack

When to use: Use a monotonic stack when you need to find the next or previous greater/smaller element in a sequence. This technique is particularly effective for problems that require comparing elements in a way that maintains a certain order (increasing or decreasing), enabling efficient retrieval of desired values.

- **Easy**
  - 496. Next Greater Element I
  - 739. Daily Temperatures *(Though marked as medium, it's a good starting point)*
- **Medium**
  - 901. Online Stock Span
  - 402. Remove K Digits
  - 739. Daily Temperatures
- **Hard**
  - 84. Largest Rectangle in Histogram
  - 85. Maximal Rectangle
  - 316. Remove Duplicate Letters

Top K Elements

When to use: Use top K elements techniques when you need to find the largest or smallest K elements in a collection. Heaps or priority queues are often employed to maintain the top K elements efficiently, especially when dealing with large datasets.

- **Easy**
  - 703. Kth Largest Element in a Stream
- **Medium**
  - 347. Top K Frequent Elements
  - 215. Kth Largest Element in an Array
  - 973. K Closest Points to Origin
- **Hard**
  - 295. Find Median from Data Stream
  - 480. Sliding Window Median
  - 632. Smallest Range Covering Elements from K Lists

Overlapping Intervals

When to use: Use overlapping intervals techniques when dealing with problems that involve scheduling, merging intervals, or detecting conflicts. Sorting the intervals and processing them based on start and end times is often key to solving these problems efficiently.

- **Easy**
  - 252. Meeting Rooms
- **Medium**
  - 56. Merge Intervals
  - 57. Insert Interval
  - 253. Meeting Rooms II
  - 435. Non-overlapping Intervals
- **Hard**
  - 759. Employee Free Time
  - 352. Data Stream as Disjoint Intervals

Modified Binary Search

When to use: Use modified binary search when dealing with sorted or partially sorted arrays where standard binary search needs adjustments, such as rotated arrays or arrays with unknown properties. It's ideal for finding elements under specific conditions that standard binary search doesn't cover.

- **Easy**
  - 704. Binary Search
  - 744. Find Smallest Letter Greater Than Target
- **Medium**
  - 33. Search in Rotated Sorted Array
  - 81. Search in Rotated Sorted Array II

    - 162. Find Peak Element
    - 153. Find Minimum in Rotated Sorted Array
- **Hard**
    - 154. Find Minimum in Rotated Sorted Array II
    - 302. Smallest Rectangle Enclosing Black Pixels

Binary Tree Traversal

When to use: Use binary tree traversal techniques when you need to visit all nodes in a binary tree in a specific order (preorder, inorder, postorder, level-order). Traversals are fundamental for solving tree-related problems involving data processing or tree construction.

- **Easy**
    - 94. Binary Tree Inorder Traversal
    - 144. Binary Tree Preorder Traversal
    - 145. Binary Tree Postorder Traversal
- **Medium**
    - 102. Binary Tree Level Order Traversal
    - 103. Binary Tree Zigzag Level Order Traversal
    - 105. Construct Binary Tree from Preorder and Inorder Traversal
- **Hard**
    - 297. Serialize and Deserialize Binary Tree
    - 124. Binary Tree Maximum Path Sum
    - 1457. Pseudo-Palindromic Paths in a Binary Tree

Depth First Search

When to use: Use Depth First Search (DFS) when you need to explore all possible paths in a graph or tree, especially when you want to traverse as deep as possible before backtracking. It's suitable for problems involving connectivity, pathfinding, and detecting cycles.

- **Easy**
    - 104. Maximum Depth of Binary Tree
    - 101. Symmetric Tree
- **Medium**
    - 200. Number of Islands
    - 695. Max Area of Island
    - 130. Surrounded Regions
    - 547. Number of Provinces
- **Hard**
    - 329. Longest Increasing Path in a Matrix

    - 212. Word Search II
    - 124. Binary Tree Maximum Path Sum

Breadth First Search

When to use: Use Breadth First Search (BFS) when you need to explore nodes in a graph or tree level by level, particularly useful for finding the shortest path in unweighted graphs or when the problem requires processing nodes in order of their distance from the source.

- **Easy**
    - 111. Minimum Depth of Binary Tree
    - 104. Maximum Depth of Binary Tree *(Can also be solved using BFS)*
- **Medium**
    - 102. Binary Tree Level Order Traversal
    - 133. Clone Graph
    - 207. Course Schedule
    - 994. Rotting Oranges
- **Hard**
    - 126. Word Ladder II
    - 317. Shortest Distance from All Buildings
    - 1293. Shortest Path in a Grid with Obstacles Elimination

---

Matrix Traversal Pattern

When to use: Use matrix traversal techniques when dealing with grid-based problems, often involving visiting each cell and exploring its neighbors. This pattern is common in problems that require searching, pathfinding, or region identification in 2D grids.

- **Easy**
    - 733. Flood Fill
    - 542. 01 Matrix
- **Medium**
    - 79. Word Search
    - 200. Number of Islands
    - 994. Rotting Oranges
    - 130. Surrounded Regions
- **Hard**
    - 212. Word Search II
    - 329. Longest Increasing Path in a Matrix
    - 85. Maximal Rectangle

Backtracking

When to use: Use backtracking when you need to generate all possible solutions and need to explore all potential options, especially when the problem involves permutations, combinations, or subsets. It's effective for problems where you can prune paths that don't lead to a valid solution.

- **Easy**
  - 784. Letter Case Permutation
- **Medium**
  - 46. Permutations
  - 47. Permutations II
  - 39. Combination Sum
  - 40. Combination Sum II
  - 78. Subsets
  - 79. Word Search
- **Hard**
  - 51. N-Queens
  - 212. Word Search II
  - 37. Sudoku Solver

Dynamic Programming

When to use: Use dynamic programming when a problem can be broken down into overlapping subproblems, and when you can store the results of these subproblems to avoid redundant calculations. It's ideal for optimization problems where you're looking for the maximum, minimum, or total number of ways to do something.

- **Easy**
  - 70. Climbing Stairs
  - 121. Best Time to Buy and Sell Stock
  - 198. House Robber
- **Medium**
  - 322. Coin Change
  - 300. Longest Increasing Subsequence
  - 139. Word Break
  - 152. Maximum Product Subarray
  - 62. Unique Paths
- **Hard**
  - 72. Edit Distance
  - 312. Burst Balloons

- 115. Distinct Subsequences
- 132. Palindrome Partitioning II