LeetCode Patterns: Core Problems Guide

Table of Contents

LeetCode Patterns: Core Problems Guide

Companion solutions:

1. LeetCode Core Problems: Python and Go Solutions

How to Use This Guide

For each problem:

1. Identify the pattern first.
2. Write brute force quickly.
3. Replace with optimal pattern.
4. State time and space complexity out loud.

1. Arrays / Hashing

- `1. Two Sum` Pattern: hash map for complement lookup. Complexity: `O(n)` time, `O(n)` space.
- `49. Group Anagrams` Pattern: hash by character frequency or sorted signature. Complexity: `O(n * k log k)` with sorting, `O(n * k)` with count signature.

- `238. Product of Array Except Self` Pattern: prefix and suffix products without division. Complexity: `O(n)` time, `O(1)` extra space (excluding output).
- `560. Subarray Sum Equals K` Pattern: prefix sum + hash map of seen sums. Complexity: `O(n)` time, `O(n)` space.
- `347. Top K Frequent Elements` Pattern: frequency map + bucket sort (or heap). Complexity: `O(n)` average with buckets, `O(n log k)` with heap.

2. Sliding Window / Strings

- `3. Longest Substring Without Repeating Characters` Pattern: variable window + set/map for last seen index. Complexity: `O(n)` time, `O(min(n, charset))` space.
- `76. Minimum Window Substring` Pattern: variable window + frequency counters + valid-match count. Complexity: `O(n)` time, `O(charset)` space.
- `424. Longest Repeating Character Replacement` Pattern: window + max frequency in window. Complexity: `O(n)` time, `O(charset)` space.
- `567. Permutation in String` Pattern: fixed-size sliding window + frequency compare. Complexity: `O(n)` time, `O(charset)` space.
- `125. Valid Palindrome` Pattern: two pointers skipping non-alphanumeric. Complexity: `O(n)` time, `O(1)` extra space.

3. Binary Search

- `33. Search in Rotated Sorted Array` Pattern: binary search with sorted-half detection. Complexity: `O(log n)` time, `O(1)` space.
- `153. Find Minimum in Rotated Sorted Array` Pattern: binary search using right boundary comparison. Complexity: `O(log n)` time, `O(1)` space.
- `875. Koko Eating Bananas` Pattern: binary search on answer (minimum feasible speed). Complexity: `O(n log m)` where `m` is max pile.
- `981. Time Based Key-Value Store` Pattern: hash map key to sorted `(timestamp, value)` list + binary search. Complexity: `set O(1)`, `get O(log n)` per key.

4. Trees / Graphs

- `102. Binary Tree Level Order Traversal` Pattern: BFS with queue by level. Complexity: `O(n)` time, `O(w)` space (tree width).
- `199. Binary Tree Right Side View` Pattern: BFS by level or DFS right-first. Complexity: `O(n)` time, `O(h)` DFS or `O(w)` BFS space.
- `236. Lowest Common Ancestor of a Binary Tree` Pattern: postorder DFS returns node or null. Complexity: `O(n)` time, `O(h)` space.
- `200. Number of Islands` Pattern: grid DFS/BFS flood fill. Complexity: `O(m*n)` time, `O(m*n)` worst-case stack/queue.

- `133. Clone Graph` Pattern: DFS/BFS + old-to-new node map. Complexity: `O(V+E)` time, `O(V)` space.

5. Linked List

- `206. Reverse Linked List` Pattern: iterative pointer reversal. Complexity: `O(n)` time, `O(1)` space.
- `21. Merge Two Sorted Lists` Pattern: two-pointer merge with dummy head. Complexity: `O(n+m)` time, `O(1)` extra space.
- `143. Reorder List` Pattern: find middle + reverse second half + merge alternating. Complexity: `O(n)` time, `O(1)` space.
- `141. Linked List Cycle` Pattern: Floyd slow/fast pointers. Complexity: `O(n)` time, `O(1)` space.
- `2. Add Two Numbers` Pattern: digit-by-digit sum with carry. Complexity: `O(max(n,m))` time, `O(1)` extra space.

6. Intervals / Greedy

- `56. Merge Intervals` Pattern: sort by start, merge overlaps. Complexity: `O(n log n)` time, `O(n)` output space.
- `57. Insert Interval` Pattern: append non-overlap left, merge overlap, append right. Complexity: `O(n)` time, `O(n)` output space.
- `435. Non-overlapping Intervals` Pattern: greedy by earliest end time; count removals. Complexity: `O(n log n)` time, `O(1)` extra space after sort.
- `452. Minimum Number of Arrows to Burst Balloons` Pattern: greedy by end coordinate. Complexity: `O(n log n)` time, `O(1)` extra space after sort.

7. Dynamic Programming

- `70. Climbing Stairs` Pattern: Fibonacci-style 1D DP. Complexity: `O(n)` time, `O(1)` space optimized.
- `198. House Robber` Pattern: take/skip state transition. Complexity: `O(n)` time, `O(1)` space optimized.
- `322. Coin Change` Pattern: unbounded knapsack minimum coins. Complexity: `O(amount * coins)` time, `O(amount)` space.
- `139. Word Break` Pattern: DP over prefix breakability. Complexity: `O(n^2)` typical, depends on dictionary lookup/trie.
- `300. Longest Increasing Subsequence` Pattern: patience sorting tails + binary search. Complexity: `O(n log n)` time, `O(n)` space.

8. Backtracking

- `39. Combination Sum` Pattern: choose/recurse with repeat allowed. Complexity: exponential search space.

- `46. Permutations` Pattern: backtracking with used-set / in-place swap. Complexity: `O(n * n!)` time.
- `78. Subsets` Pattern: include/exclude recursion. Complexity: `O(n * 2^n)` time.
- `79. Word Search` Pattern: DFS + visited marking + backtrack. Complexity: `O(m*n*4^L)` worst-case.

Practice Plan

1. Week 1: Arrays/Hashing + Sliding Window.
2. Week 2: Binary Search + Linked List.
3. Week 3: Trees/Graphs + Intervals/Greedy.
4. Week 4: DP + Backtracking + timed mixed sets.

Interview prep target:

1. Solve each problem twice.
2. On second pass, explain pattern and complexity in under 60 seconds.