## 🗓️ 7-Day Revision Plan with Curated Reading Material

All problems are handpicked from the comprehensive lists you provided.

| Day | Topic | Theory & Patterns (Read These First) | Problem 1: The Foundation | Problem 2: The Classic | Problem 3: The Variation | Problem 4: The Hard One |
| --- | --- | --- | --- | --- | --- | --- |
| **1** | **Hashing & Prefix Sums** | 1. **Hashing:** [GeeksforGeeks Tutorial](https://www.geeksforgeeks.org/hashing-data-structure/) (A great overview of the concept). <br> 2. **Prefix Sums:** [CP-Algorithms Article](https://www.google.com/search?q=https://cp-algorithms.com/data_structures/prefix-sums-and-difference-arrays.html) (The competitive programmer’s choice for this topic). | [#1 Two Sum](https://leetcode.com/problems/two-sum/) | [#560 Subarray Sum Equals K](https://leetcode.com/problems/subarray-sum-equals-k/) | [#128 Longest Consecutive Sequence](https://leetcode.com/problems/longest-consecutive-sequence/) | - |
| **2** | **Two Pointers** | 1. **Two Pointer Technique:** [GeeksforGeeks Article](https://www.geeksforgeeks.org/two-pointers-technique/) (Covers all the basic variations you need to know). | [#167 Two Sum II](https://leetcode.com/problems/two-sum-ii-input-array-is-sorted/) | [#11 Container With Most Water](https://leetcode.com/problems/container-with-most-water/) | [#15 3Sum](https://leetcode.com/problems/3sum/) | [#42 Trapping Rain Water](https://leetcode.com/problems/trapping-rain-water/) |
| **3** | **Sliding Window** | 1. **Sliding Window:** [GeeksforGeeks Article](https://www.geeksforgeeks.org/window-sliding-technique/) (Provides a solid template for both fixed and variable-size windows). | [#3 Longest Substring Without Repeating Chars](https://leetcode.com/problems/longest-substring-without-repeating-characters/) | [#209 Minimum Size Subarray Sum](https://leetcode.com/problems/minimum-size-subarray-sum/) | [#567 Permutation in String](https://leetcode.com/problems/permutation-in-string/) | [#76 Minimum Window Substring](https://leetcode.com/problems/minimum-window-substring/) |
| **4** | **Substrings & Palindromes** | 1. **Longest Palindromic Substring:** [LeetCode Solution Article](https://www.google.com/search?q=https://leetcode.com/problems/longest-palindromic-substring/solutions/127837/longest-palindromic-substring/) (Read the “Approach 4: Expand Around Center” explanation). | [#242 Valid Anagram](https://leetcode.com/problems/valid-anagram/) | [#5 Longest Palindromic Substring](https://leetcode.com/problems/longest-palindromic-substring/) | [#647 Palindromic Substrings](https://leetcode.com/problems/palindromic-substrings/) | - |
| **5** | **Subsets & Backtracking** | 1. **Backtracking:** [GeeksforGeeks Introduction](https://www.geeksforgeeks.org/introduction-to-backtracking-data-structure-and-algorithm-tutorials/) (Focus on the state-space tree concept). | [#78 Subsets](https://leetcode.com/problems/subsets/) | [#90 Subsets II](https://leetcode.com/problems/subsets-ii/) | [#494 Target Sum](https://leetcode.com/problems/target-sum/) | - |
| **6** | **Subsequences (DP)** | 1. **LIS:** [CP-Algorithms Article](https://cp-algorithms.com/sequences/longest_increasing_subsequence.html) (Explains both the O(N²) and O(N log N) solutions).<br>2. **LCS:** [GeeksforGeeks Article](https://www.geeksforgeeks.org/longest-common-subsequence-dp-4/) (A clear, standard DP explanation). | [#392 Is Subsequence](https://leetcode.com/problems/is-subsequence/) | [#300 Longest Increasing Subsequence](https://leetcode.com/problems/longest-increasing-subsequence/) | [#1143 Longest Common Subsequence](https://leetcode.com/problems/longest-common-subsequence/) | - |
| **7** | **Advanced DP on Strings** | 1. **Edit Distance:** [GeeksforGeeks Article](https://www.geeksforgeeks.org/edit-distance-dp-5/) (The classic tutorial for this must-know DP problem). | [#516 Longest Palindromic Subsequence](https://leetcode.com/problems/longest-palindromic-subsequence/) | [#72 Edit Distance](https://leetcode.com/problems/edit-distance/) | [#115 Distinct Subsequences](https://leetcode.com/problems/distinct-subsequences/) | - |

## 🗂️ Arrays & String Topics

1. **Basic Array Operations**  
   – Remove Duplicates, Merge, Watermelon, Majority Element
2. **Prefix Sums & Range Queries**  
   – Range Sum, Subarray Sum K, Pivot Index, Chef/Subarray
3. **Array Rotation & Cyclic Operations**  
   – Rotate Array, Rotate Function, Target Practice, First Missing Positive
4. **Frequency Counting & Hashing**  
   – Two Sum, Disappeared Numbers, Triple, Longest Consecutive Sequence
5. **Matrix Operations**  
   – Rotate Image, Spiral Matrix, String Solitaire, Set Matrix Zeroes
6. **Two Pointers (Basic → Advanced → String Variants)**  
   – Two Sum II, Valid/Reverse/Vowels Palindrome, Move Zeroes  
   – 3-Sum, 4-Sum & Variants  
   – Dutch Flag, Container With Most Water, Trapping Rain Water  
   – Nth Node From End, Interval Intersections, Sorted Squares  
   – Subsequence checks, Longest Word delete, Valid Palindrome II
7. **Sliding Window (Fixed-Size → Variable → Advanced)**  
   – Max Avg Subarray, Max Vowels, Find Anagrams  
   – Longest Substring w/o Repeats, Min-Size Subarray, K-Distinct  
   – Min-Window Substring, Permutation in String, Repeating Char Replacement  
   – Sliding Window Maximum/Median, Subarrays w/ K Different
8. **Hybrid Window + Two-Pointer Patterns**  
   – Fruit Into Baskets, Min Swaps to Group 1s, Beat The Odds
9. **String Matching & Transformation**  
   – strStr(), Repeated Substring, Rotate String, Anagram Checks  
   – Palindromic Substrings, Partitioning, Longest Palindromic Substring  
   – Group Anagrams, Min Steps to Anagram, Valid Anagram
10. **Subsets & Bitmask-DP**  
    – Subsets I/II, Target Sum, Subset-Sum Variants
11. **Subsequences & Sequence-DP**  
    – Is Subsequence, LCS, LIS, Number of LIS  
    – Distinct Subsequences I/II, Palindromic Subsequence, Edit-Distance–style
12. **Advanced Substring/DP Integration**  
    – Edit Distance, Interleaving, Shortest Common Supersequence  
    – Longest Valid Parentheses, String Compression, Valid Palindrome III  
    – Longest Duplicate Substring, Delete Operation Two Strings
13. **Mixed-Pattern & Contest-Style Combinations**  
    – Problem sets that blend multiple above patterns under contest time pressure
14. **Final Mixed Challenge**  
    – Hard mash-ups (e.g., Job Scheduling, Super Egg Drop, CodeChef Mixes)

**Pattern Recognition**

**1. Two-Pointers**

**When to spot it:**

* You need to find pairs (or triples) in an **array/string** where you can move two indices from ends or a left/right boundary.
* **Array is sorted** (or you can sort it without breaking index requirements).
* Typical tasks:
  + Sum to a target (two-sum II, 3-sum, closest sum)
  + Container/trapping water (max area, trapping rain water)
  + Merging intervals or checking palindromes on strings

**Rule of Thumb:**

If you see “find two indices i < j such that …” **and** either the input is sorted or can be sorted, reach for two-pointers.

**2. Sliding Window**

**When to spot it:**

* You’re looking for a **contiguous subarray or substring** that optimizes/minimizes something (sum, length, count).
* The problem asks for “longest,” “smallest,” or “exact” window satisfying a condition.
* You can maintain counts or sums incrementally by expanding/contracting a window.

**Rule of Thumb:**

If you see “subarray” or “substring” **and** a question about sum/count/unique-characters over it (“at most k,” “exactly k,” “max/min size”), use sliding window.

**3. Subsets / Backtracking**

**When to spot it:**

* You need to **generate** all combinations/subsets/permutations.
* The output asks for “all possible …” or “count of subsets,” or you need to test each subset for some property.
* Constraints are small enough (n ≤ 15–20).

**Rule of Thumb:**

If the prompt says “find all subsets/combinations” or “choose any k out of n,” lean on backtracking/bitmask DP.

**4. Substrings vs Subsequences**

| **Pattern** | **Definition** | **When to use** |
| --- | --- | --- |
| **Substring** | Contiguous sequence of characters | Sliding window, KMP, two-pointer on strings |
| **Subsequence** | Not necessarily contiguous (order only) | Greedy scan (isSubsequence), LCS, DP on sequences |

**Rule of Thumb:**

“Substring” → contiguous → sliding window or KMP.  
“Subsequence” → skip allowed → greedy two-pointer scan or DP (LCS-style).

**5. Dynamic Programming (DP)**

**When to spot it:**

* Problem asks for **optimal value** (max/min count, way counts) under overlapping subproblems.
* You see phrases like “maximum sum,” “longest,” “count ways,” “edit distance,” or constraints up to n ≈ 10^3–10^5 but with DP-friendly structure.
* State can be defined by indices (i, j), masks, or lengths.

**Rule of Thumb:**

If you can define dp[i], dp[i][j], or dp[mask] with smaller subproblems, go DP.

**6. Hashing & Frequency Counting**

**When to spot it:**

* You need to detect duplicates, anagrams, or count occurrences.
* Queries about “most frequent,” “pairs with equal x,” or “exists two with same difference.”

**Rule of Thumb:**

If the core operation is “counting” rather than “ordering,” a hash map (or frequency array) is your friend.

**7. Graph / Tree Patterns**

**When to spot it:**

* Input describes nodes/edges or a binary tree string.
* You’re asked about connectivity, shortest paths, spanning trees, or tree traversals.

**Rule of Thumb:**

Node-edge input → BFS/DFS. Weighted paths → Dijkstra/Bellman-Ford. Tree → recursion/tree DP.

**8. Greedy Algorithms**

**When to spot it:**

* You need a locally optimal choice leading to global optimum (“minimum number of intervals,” “maximize number of events attended”).
* Sort + scan pattern.

**Rule of Thumb:**

If a sorted order and a “take-it-or-leave-it” choice at each step works, try greedy.

**🧰 Quick Pattern-Recognition Checklist**

1. **Pairs in sorted data?** → Two Pointers
2. **Contiguous subarray/substring optimize?** → Sliding Window
3. **Generate all combinations?** → Backtracking / Subsets
4. **Skip allowed?** → Subsequences / LCS DP
5. **Optimal value with overlapping subproblems?** → DP
6. **Count or frequency?** → Hashing
7. **Nodes & edges?** → Graph/Tree Algo
8. **Sort + simple selection?** → Greedy

Use this checklist when you read the problem statement: highlight keywords (“subarray,” “k distinct,” “choose,” “max/min,” “path,” “interval”), match them to the rules above, and pick your pattern. With practice, pattern identification becomes almost instinctive. Good luck!