**Day 2: Strings - Patterns and Sliding Window**

**🔐 1. Longest Substring Without Repeating Characters**

**Problem Statement:**  
Track the longest sequence of keystrokes a user enters without typing any repeated character — useful for password strength or behavior analysis.  
**Sample Input:**  
s = "abcabcbb"  
**Expected Output:**  
3  
*Explanation: The substring "abc" is the longest without repeating characters.*

**🔤 2. Valid Anagram**

**Problem Statement:**  
Validate whether two usernames or entries are rearrangements of the same letters — useful in spelling check or secure aliasing.  
**Sample Input:**  
s = "anagram", t = "nagaram"  
**Expected Output:**  
True

**🔁 3. Group Anagrams**

**Problem Statement:**  
Cluster search queries or user aliases that are scrambled versions of each other — useful in NLP or search optimization.  
**Sample Input:**  
strs = ["eat","tea","tan","ate","nat","bat"]  
**Expected Output:**  
[["eat","tea","ate"], ["tan","nat"], ["bat"]]

**🪞 4. Longest Palindromic Substring**

**Problem Statement:**  
Detect the longest symmetric pattern in a DNA sequence or user input — often used in bioinformatics or UI analysis.  
**Sample Input:**  
s = "babad"  
**Expected Output:**  
"bab" or "aba"

**🔍 5. Palindromic Substrings**

**Problem Statement:**  
Count how many palindromic snippets exist in a text body — useful in linguistics or pattern discovery.  
**Sample Input:**  
s = "abc"  
**Expected Output:**  
3  
*Explanation: "a", "b", and "c" are each palindromic.*

**🧭 6. Minimum Window Substring**

**Problem Statement:**  
Identify the shortest section of a document that contains all required keywords — useful for search or content extraction.  
**Sample Input:**  
s = "ADOBECODEBANC", t = "ABC"  
**Expected Output:**  
"BANC"

**🧬 7. Isomorphic Strings**

**Problem Statement:**  
Determine if two identifier mappings follow the same structural pattern — valuable in language compilers or encryption.  
**Sample Input:**  
s = "egg", t = "add"  
**Expected Output:**  
True

**📦 8. String Compression**

**Problem Statement:**  
Compress repeated logs or character sequences in telemetry data — crucial for memory-efficient processing.  
**Sample Input:**  
chars = ["a","a","b","b","c","c","c"]  
**Expected Output:**  
6, chars = ["a","2","b","2","c","3"]

**🧮 9. Multiply Strings**

**Problem Statement:**  
Simulate multiplication of two extremely large numbers stored as strings — essential in financial or scientific computing.  
**Sample Input:**  
num1 = "123", num2 = "456"  
**Expected Output:**  
"56088"

**🔎 10. Implement strStr()**

**Problem Statement:**  
Find the first occurrence of a keyword in a paragraph or code snippet — like Ctrl+F behavior.  
**Sample Input:**  
haystack = "hello", needle = "ll"  
**Expected Output:**  
2

**🔍 11. Rabin-Karp Algorithm**

**Problem Statement:**  
Quickly scan large documents to locate a plagiarism match or keyword occurrence using rolling hash — perfect for search engines and text indexing.  
**Sample Input:**  
text = "ABABDABACDABABCABCABCABCABC", pattern = "ABABCABCABCABC"  
**Expected Output:**  
15  
*Explanation: Pattern found starting at index 15.*

**🔎 12. KMP Algorithm**

**Problem Statement:**  
Efficiently locate repeating template phrases in legal documents or logs using prefix optimization — ideal for streamlining content search.  
**Sample Input:**  
text = "ABABDABACDABABCABCABCABCABC", pattern = "ABABCABCABCABC"  
**Expected Output:**  
15

**🏛️ 13. Roman to Integer**

**Problem Statement:**  
Convert legacy data formats or archaeological notations (Roman numerals) into standard numeric form for analysis.  
**Sample Input:**  
s = "MCMXC"  
**Expected Output:**  
1990

**🏺 14. Integer to Roman**

**Problem Statement:**  
Format modern values into Roman numeral form for certificate generation or themed branding (e.g. "Version IV").  
**Sample Input:**  
num = 1994  
**Expected Output:**  
"MCMXCIV"

**🔓 15. Decode Ways**

**Problem Statement:**  
Determine how many possible interpretations exist for a numerical code, such as SMS-based encoding or secret unlocking patterns.  
**Sample Input:**  
s = "226"  
**Expected Output:**  
3  
*Explanation: Interpretations include "2-2-6", "22-6", "2-26".*

**🪡 16. Zigzag Conversion**

**Problem Statement:**  
Transform a message into zigzag pattern for stylized display or transmission layout — common in data encoding or banners.  
**Sample Input:**  
s = "PAYPALISHIRING", numRows = 3  
**Expected Output:**  
"PAHNAPLSIIGYIR"

**🔁 17. Reverse Words in a String**

**Problem Statement:**  
Reorder a sentence’s words to reverse their meaning or create stylized content for quotations or tweets.  
**Sample Input:**  
s = " hello world "  
**Expected Output:**  
"world hello"

**📏 18. Longest Common Prefix**

**Problem Statement:**  
Extract shared prefixes in product IDs or code snippets for categorization or grouping.  
**Sample Input:**  
strs = ["flower","flow","flight"]  
**Expected Output:**  
"fl"

**🔄 19. Check if One String is Rotation of Another**

**Problem Statement:**  
Validate cyclic permutations of passcodes or user inputs — useful in authentication and puzzle logic.  
**Sample Input:**  
s1 = "abcde", s2 = "cdeab"  
**Expected Output:**  
True

**🔢 20. Count and Say**

**Problem Statement:**  
Generate sequential sound-based encodings for versioning or puzzle generation — mimicking auditory patterns.  
**Sample Input:**  
n = 4  
**Expected Output:**  
"1211"  
*Explanation: "1" → "11" → "21" → "1211"*