Longest Common Subsequence

Problem Statement:

A subsequence is formed by removing zero or more characters from a given string while maintaining the original order of the remaining characters. For example, the subsequences of “ABC” include “A”, “B”, “C”, “AB”, “AC”, “BC”, and “ABC”. Given two sequences, we need to find the length of their longest subsequence that appears in both sequences in the same order but not necessarily contiguously. Given two sequences X and Y, the goal is to determine the longest sequence Z such that Z is a subsequence of both X and Y.

Example:
Input: X="ACDBE", Y="ABCDE"
Output: LCS = "ACDE"

Algorithm 1: Recursive Approach

This approach uses recursion to find the LCS by checking whether characters of the two sequences match or not.

- If the last characters of both sequences match, the LCS length is incremented by 1, and the problem is solved for the remaining substrings.
- Otherwise, the function is called recursively for both cases: excluding the last character from one sequence at a time.

Algorithm 2: Memoization Approach

To optimize the recursive approach, we use memoization by storing the results of subproblems in a table. This approach improves the recursive method by storing previously computed results to avoid redundant calculations.

- A table is used to store LCS values for different prefixes of the sequences.
- If the LCS for a given subproblem is already computed, the stored value is returned instead of recalculating it.

Algorithm 3: Dynamic Programming Approach

This approach builds the LCS solution iteratively using a table instead of recursion.

- The LCS values for smaller subproblems are computed first and used to solve larger ones.
- The table is filled row by row until the final LCS value is obtained.