Problem Description

The problem is to find the Longest Common Subsequence (LCS) of two given strings. A subsequence is a sequence that appears in the same relative order but not necessarily contiguous. The Longest Common Subsequence is the longest subsequence that is common to both strings.

Recursive Solution:

The given code uses a recursive approach to find the LCS. The function `recursionLCS` takes two strings (`str1` and `str2`) and their sizes (`size1` and `size2`) as input. It returns the length of the LCS and updates the `ans` string with the LCS.

Dynamic Programming Solution:

The algorithm uses a two-dimensional array (dp) to store intermediate results.The cell dp[i][j] represents the length of the LCS of the substrings s1[0...i-1] and s2[0...j-1].

Filling the DP Table:

The algorithm iterates over the lengths of substrings s1 and s2 and fills in the DP table based on character matches.If s1[i-1] == s2[j-1], it increments the LCS length by 1.Otherwise, it takes the maximum LCS length from the adjacent cells in the DP table.

Backtracking:

After constructing the DP table, the algorithm performs backtracking to reconstruct the actual LCS string.It starts from the bottom-right corner of the DP table and moves towards the top-left, choosing characters that contribute to the LCS.

Asymptotic Upper Bound

Recursive Solution:

The recursive solution has an exponential time complexity. Let n be the maximum of `size1` and `size2` (the length of the longer string). The recurrence relation for the time complexity is **T(n) = 2T(n-1) + O(1).** This results from the fact that for each character in the strings, the recursive function makes two calls (one excluding the character from `str1`, the other excluding it from `str2`). The base case takes constant time. **The overall time complexity is O(2^n), which is exponential.**

The space complexity is also significant due to the recursive calls, and it is O(n) due to the recursion stack depth.

Dynamic Programming Solution:

The dynamic programming approach to solving the Longest Common Subsequence problem is more efficient. It has a time complexity of **O(size1 times size2)** and a space complexity of **O(size1 times size2)**. This improvement is achieved by avoiding recomputation of overlapping subproblems, storing the solutions in a matrix and building up the solution iteratively.

Conclusion:

In conclusion, while the recursive solution provides a simple and intuitive approach to solving the problem, it suffers from an exponential time complexity. The dynamic programming solution is more efficient and practical for larger inputs, providing a polynomial time complexity.