DAY 20:

ASSIGNMENT 2:

Task 2: Longest Common Subsequence

Implement int LCS(string text1, string text2) to find the length of the longest common subsequence between two strings.

ANSWER:

```
public class LongestCommonSubsequence {

    public static int LCS(String text1, String text2) {
        int m = text1.length();
        int n = text2.length();
        int[] dp = new int[n + 1];

        // Build the dp array in bottom-up manner
        for (int i = 1; i <= m; i++) {
            int prev = 0; // This will be dp[j-1] from the previous row
            for (int j = 1; j <= n; j++) {
                int temp = dp[j]; // Store the current dp[j] to update it later
                if (text1.charAt(i - 1) == text2.charAt(j - 1)) {
                    dp[j] = prev + 1;
                } else {
                    dp[j] = Math.max(dp[j], dp[j - 1]);
                }
                prev = temp; // Update prev to the current dp[j] for the next iteration
            }
        }

        return dp[n];
    }
```

```
    public static void main(String[] args) {
        String text1 = "abcde"; // Example string 1
        String text2 = "ace";   // Example string 2

        System.out.println("Length of LCS = " + LCS(text1, text2));
    }
}
```

Explanation:

1. Initialization:

- dp[j] will hold the length of the LCS of the substrings text1[0..i-1] and text2[0..j-1].
- Initialize dp[j] = 0 for all j because the LCS with any empty string is 0.

2. Filling the DP Array:

- Iterate over each character of text1 (outer loop) and text2 (inner loop).
- Use a variable prev to store the value of dp[j-1] from the previous row, which is needed to calculate dp[j] for the current row.
- If text1[i-1] == text2[j-1], then the characters match, and the LCS up to i and j is prev + 1.
- If the characters do not match, then the LCS up to i and j is the maximum of the LCS without the current character of text1 (dp[j]) or without the current character of text2 (dp[j-1]).

3. Result:

- The result will be in dp[n], which represents the length of the LCS of text1 and text2.

This solution is more space-efficient, using $O(n)$ space instead of $O(m \times n)$ by leveraging a rolling array approach. The time complexity remains $O(m \times n)$, where m and n are the lengths of text1 and text2, respectively.