Algorithm CompSci 260P Project 2

Student 1: Wei Lan
UCI id #: 26324230
Student 2: Yenkai Wang
UCI id #: 14161906

Algorithm:

We find the length of the LCS strings and the distinct strings at the same time by using dynamic programming.
We defend a 2D (n+1) * (n+1) dp array. For each position of dp[i+1][j+1], it represents the length of the LCS between x[0:i] and y[0:j].
We also define similar 2D (n+1)*(n+1) set<string> setStr array. It keeps the distinct strings in corresponding dp bucket.

We can observe the following relations:
if x[i] == y[j], then dp[i][j] = dp[i-1][j-1] + 1.
The equality between x[i] and y[j] ensures they can form one character in LCS.
We put all of the distinct LCS appending the matched char from setStr[i-1][j-1] into setStr[i][j]

if x[i] != y[j], then dp[i][j] = max(dp[i-1][j], dp[i][j-1])
If x[i] and y[j] mismatches, then we have to remove either x[i] or y[j] to find out the recursive LCS length. We pick up the larger one which can generate the longest LCS. We put this set strings into setStr[i][j] without adding any characters.

We do bottom up from i=0 to n and j=0 to n. The length of LCS is saved in dp[n][n]. The distinct set strings are saved in setStr[n][n]

Analysis:

Because we do from bottom up, the total operations are N^2. Another complexity comes from inserting into set. Since the set size doesn't very large, we can treat it as nearly constant. The runtime complexity is almost O(N^2).