Taller 2
DALGO

1. Archivo 1

2. Archivo 2

3.

a) En la solución inicial basada dividir y conquistar, aunque se adopta un enfoque de esa estrategia al dividir el problema en subproblemas más pequeños, esta solución no sigue la metodología de dividir y conquistar en su sentido más eficiente, ya que los subproblemas aquí se generan a través de operaciones de inserción o eliminación, y a menudo se repiten debido a la naturaleza de las permutaciones y las operaciones permitidas. Sin embargo, sin una estrategia para combinar eficientemente las soluciones de estos subproblemas y sin evitar el cálculo redundante, el método resulta ser ineficiente, especialmente con entradas grandes.

b) La solución es bottom-up, ya que este método construye soluciones a subproblemas desde los más simples hasta resolver el problema global, llenando una tabla DP de manera iterativa para evitar cálculos, cada celda de la tabla `dp[i][j]` representa la longitud de la LCS entre las subsecuencias de las primeras `i` letras de la primera permutación y las primeras `j` letras de la segunda. Al comparar elementos de las permutaciones, se actualiza la tabla con la longitud máxima de LCS encontrada hasta el momento, asegurando que cada subproblema se solucione solo una vez. La tabla se llena de tal manera que al final, `dp[N][M]` contiene la longitud de la LCS completa, donde `N` y `M` son las longitudes de las permutaciones, permitiendo calcular el número mínimo de operaciones requeridas de manera eficiente.

c) La segunda solución es mejor que la de dividir y conquistar, debido a su eficiencia y manejo óptimo de subproblemas repetidos. Mientras que la primera solución evalúa exhaustivamente todas las posibles inserciones y eliminaciones sin aprovechar los cálculos previos, resultando en una complejidad exponencial, la solución de programación dinámica reduce significativamente la cantidad de cálculo necesario al almacenar los resultados de los subproblemas en una tabla. Esto elimina la necesidad de recalcular soluciones a subproblemas ya resueltos, lo que mejora dramáticamente la eficiencia temporal del algoritmo y permite abordar

instancias del problema de tamaño mucho mayor en un tiempo razonable, lo que la hace práctica para aplicaciones reales.