10g Longest common subsequence.

Algorithm.

// Input :- grades of 20 students in the format of vector of string.

// Output :- longest common subsequence between 20 strings.

function LCS (string x, string y)

m ← length (x)

n ← length (y)

// Create a 2D array dp of size (m+1) x (n+1), initialize to 0

dp [m+1] [n+1] = {0}

// fill the dp table

for i = 1 → m do

for j = 1 → n

if x[i-1] = y[j-1]

dp[i] [j] ← dp[i-1] [j-1] + 1

else :

dp[i] [j] ← max(dp[i-1] [j], dp[i] [j-1])

// Using backtracking to find LCS string

i ← m, j ← n

```
while i > 0 and j > 0 do
    if x[i-1] == x[j-1] do
        lcs ⇐ x[i-1] + lcs
        i <- i+i  i--
        j--
    else if dp[i-1][j] > dp[i][j-1]
        i--
    else
        j--
return LCS
```

Time Complexity

Function LST :

$O(n^2)$ for dp matrix formation.

$O(n^2)$ for backtracking answer string.

Total time complexity

Time complexity $= O(n^2) + O(n^2) = \cdots$

$= 2 \times O(n^2) = O(n^2)$

LCS for 20 string $= 20 \times$ function LCS

$= 20 \times 2 \times O(n^2)$

$= 40 \times O(n^2)$

$= O(n^2)$

Matrix Chain Multiplication.

```
int matrixMultiplication (arr)

// input :- an array
// output :- a int as a cost.
// objective :- To find the minimum cost for
                matrixMultiplication.

n = arr.size
vector < vector <int>> dp (n, vector<int>(n, 0));
        dp <= {0}
for { len <- 2 --> n do
    for i <- 0 ---> n - len do
        j = i + len
        dp[i][j] = INT_MIN
        for k <- i+1 ---> j do
        cost = dp[i][k] + dp[k][j] + arr[i] * arr[k] * arr[j]
        dp[i][j] = min (dp[i][j], cost)

return dp[0][n-1]
```

Test case.

1) dimension :- [3, 1, 5, 8]
output :- 64

2) dimension [1, 2]
output :- 0

3) dimension [1, 2, 3]
output :- 6

4) dimension [2]
output :- invalid input

5) dimension [1, 0, 2, 0]
output :- invalid input : dimension cannot be zero.

Time complexity

As we iterate over the function we get a time complexity of $O(n^3)$ as 3 for loop are nested.