★ Largest Common Subsequence.

Algorithm ::
// Input : Grades of 20 student in format of vector
string.

Function LCS (string a, string b)
    m ⟵ lenght of string a
    n ⟵ lenght of string b.

// create a 2D array dp of size (m+1)
    x (n+1), initialize to 0.
    int dp(m+1)[n+1] = {0}

// Fill the dp table

```
for (i from 1 to m)
 {
    for (j from 1 to n)
     {
        if (a[i-1] = b[j-1])      // char match
         {
            dp[i][j] ⟵ dp[i-1][j-1] + 1
         }
        else
         {
            dp[i][j] ← max (dp[i-1][j], dp[i][j-1])
```

```
// Backtrack to find the LCS string
   LCS ← empty string
   i ← m , j ← n.

while (i>0 && j>0)
 {
     if (a[i-1] == b[j-1])
     {
         LCS ← a[i-1] + LCS
         i ← i-1
         j ← j-1
     }
     else if (dp[i-1][j] > dp[i][j-1]
 }   {
         i ← i-1
       }
       else
       {
           j ← j-1
       }
     return LCS
   }

int main () :

   vector <string> v(20)
   String ans = " ";
    ans = LCS (v[0], v[1])
    for (i ← 2 to i<20 i++) :
     {
       ans = LCS (v[i], ans);
      }
    cout << ans;
```

PAGE No
DATE / /

Time Complexity:

Function LST:

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

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

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

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:

Test Case 1: [3, 1, 5, 8]
Expected output: 64

Test Case 2: [1, 2]
output: 0

Test case 3: [1, 1, 1, 1]
Output: 2

Test case 4: [1, 2, 3]
output: 6

Test case 5: [6, 7, 9, 2]
output: 210.

Test case 1: [2]
output: Invalid output

Test case 2: [2, 1, 0]
Output: Invalid input

Test case 3: [0]
output: Invalid Input

Test case 4: [1, 0, 2, 0]
output: Invalid input

Test case 5: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
Output: invalid.

★ Algorithm (Matrix Multiplication)

//input : Dimension of Array.

```
Matrixmul ( vector <int> arr )
{
    n ← arr.size();
    dp [nxn] = 0                          //initialize dp.

    for (int i=2 → i=n-1) {
        for (int j=0 → j=n-i-1) {
            int K = i+j ;
            dp [j][K] = INT_Max;
            for (int x=j+1 → x=K-1) {
                cost = dp[j][x] + dp[x][K] + arr[j].
                                                  arr[K]

                dp [j][K] = mid (d[j][K], cost)
            }

        }

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