Dhanraj Bhosale (dbhosal1@asu.edu)
ASU ID: 1225506620

Q. (1)

For 4 matrix, total 3 multiplications
To calculate number of ways these matrices can multiply between each other we can use **Catalan Number.**
Formula- B(n) = $\frac{1}{(n+1)}\binom{2n}{n}$
In this case n =3
Therefore, B(3) = ¼ (6C3)
= 5 ------- (1) number of combinations

Matrix chain multiplication formulation-
m[i , j] = optimal number of multiplications necessary to multiply matrices M_i through M_j
= m[i , k] + m[k+1 , j] + $p_{i-1} \cdot p_k \cdot p_j$

$$\boldsymbol{m[i,j]} = \begin{cases} \mathbf{0},\ \boldsymbol{if\ i = j} \\ \mathbf{min}\ [\boldsymbol{m[i,k]} + \boldsymbol{m[k+1,j]} + \mathbf{Pi-1\,.\,Pk\,.\,Pj}\ ],\ \boldsymbol{x \geq 0} \end{cases} \quad \text{-------------- (2)}$$

For i<=k<=j

Given, P0= 13 p1 = 5 p2 = 89 p3= 3 p4 = 34 ------------ (a)
We have to calculate m [1, 4], so From (2) –

$$\text{M[I,4]} = \min \begin{cases} m[1,1] + m[2,4] + p0 * p1 * p4 \\ m[1,2] + m[3,4] + p0 * p2 * p4 \\ m[1,3] + m[4,4] + p0 * p3 * p4 \end{cases}$$

Here we know, m[1,1] = m[4,4] = 0
M[1,2] = 13*5*89 = 5785
M [3, 4]=89*3*34 = 9078
M[2, 3] = 5*89*3 = 1335

$$\text{For [1,3]} = \min \begin{cases} m[1,1] + m[2,3] + 13 * 5 * 3 \\ m[1,2] + m[3,3] + 13 * 89 * 3 \end{cases}$$

$$= \min \begin{cases} 0 + 1335 + 195 \\ 5789 + 0 + 3471 \end{cases}$$

= 1530 ------------------------(3)

$$\text{For [2,4]} = \min \begin{cases} m[2,2] + m[3,4] + 5 * 89 * 34 \\ m[2,3] + m[4,4] + 5 * 3 * 34 \end{cases}$$

$$= \min \begin{cases} 0 + 9078 + 15130 \\ 1335 + 0 + 510 \end{cases}$$

= 1845 ------------------------(3)

$$\text{M[1, 4]} = \min \begin{cases} 0 + m[2,4] + 13 * 5 * 34 \\ 5785 + 9078 + 13 * 89 * 34 \\ m[1,3] + 0 + 13 * 3 * 34 \end{cases}$$

$$= \min \begin{cases} 0 + 1845 + 2210 \\ 5785 + 9078 + 39338 \\ 1530 + 0 + 1326 \end{cases}$$

$$= \min \begin{cases} 4055 \\ 54201 \\ 2856 \end{cases}$$

M[1, 4] = 2856
So the optimal Multiplication is 2856 at k = 3 and k = 1
So the order (A × (B × C)) × D.

Dhanraj Bhosale (dbhosal1@asu.edu)
ASU ID: 1225506620

Q. (2)

BCBA

X = ABCBDAB

Y = BDCABA

Longest Common Subsequence Length Formulation-

X = <x_1 …. x_m>

X_i = <x_1 …. x_i>

Y = <y_1 …. y_n>

Y_j = <y_1 …. y_j>

C[i, j] = length of the LCS of X_i and Y_j

$$\textbf{C [i, j]} = \begin{cases} C[i-1, j-1] + 1, \text{if } xi = yj, (i, j > 0) \\ \max[C[i, j-1], C[i-1, j]], \text{if if } xi \neq yj (i, j > 0) \\ 0, \text{ if } i = 0 \text{ or } j = 0 \end{cases}$$

Longest Common Subsequence Dynamic Programming Algorithm-

X and Y be two given sequences
Initialize a table LCS of dimension X.length * Y.length
X.label = X
Y.label = Y
LCS[0][] = 0
LCS[][0] = 0
Start from LCS[1][1]
Compare X[i] and Y[j]
    If X[i] = Y[j]
        LCS[i][j] = 1 + LCS[i-1, j-1]
        Point an arrow to LCS[i][j]
    Else
        LCS[i][j] = max(LCS[i-1][j], LCS[i][j-1])
        Point an arrow to max(LCS[i-1][j], LCS[i][j-1])

Using above formulation and above algorithm, we can find LCS length and store it in DP table. Also suing arrow we can store the char where it matches-

The DP table-

X = A**BCB**D**A**B

Y = **B**D**C**A**BA**

i\j	0	1 **B**	2 D	3 **C**	4 A	5 **B**	6 **A**
0	0	0	0	0	0	0	0
1 A	0	0	0	0	1	1	1
2 **B**	0	**1 **	<- 1	1	1	2	2
3 **C**	0	1	1	**2**	<- 2	2	2
4 **B**	0	1	1	2	2	**3**	3
5 D	0	1	2	2	2	3|	3
6 **A**	0	1	2	2	3	3	**4**
J B	0	1	2	2	3	4	4|

So, the LCS is BCBA.

Dhanraj Bhosale (dbhosal1@asu.edu)
ASU ID: 1225506620

Q. (3)

Interleaving String Formulation-

Given String

$S1 = <s_1 \ldots . s_m>$
$S2 = <s2_1 \ldots . s2_n>$
$S1_i = <s1_1 \ldots . s1_i>$
$S2_j = <s2_1 \ldots . s2_j>$
$S3 = <s3_1 \ldots . s3_k>$
$S3_{i+j} = <s3_1 \ldots . s3_{i+j}>$

C [i, j] = **True** if $S3_{i+j}$ is an interleaving string of $S1_i$ and $S2_j$, **False** otherwise.

Case 1: If i=j=0:
Base case- no string
C[i, j] = True

Case 2: If i=0:
C[0, j] = C[0, j-1]

Case 3: If j=0:
C[i, 0] = C[i-1, 0]

Case 4: $s3_{i+j} = s1_i$
Look if $S3_{i+j-1}$ is interleaving $S1_{i-1}$ and $S2_j$, i.e. look for value c [i-1, j]
c[i, j] = c [i-1, j]

Case 5: $s3_{i+j} = s2_j$
Look if $S3_{i+j-1}$ is interleaving $S1_i$ and $S2_{j-1}$, i.e. Look for value c [i, j-1]
c[i, j] = c [i, j-1]

Case 6: $s3_{i+j} \neq s1_i \text{ and } s3_{i+j} \neq s2_j$:
Not interleaving, Assign False
C[i, j] = False

$$\textbf{C [i, j]} = \begin{cases} \boldsymbol{C[i-1, j], if\ s3_{i+j} = s1_i, (i, j > 0)} \\ \boldsymbol{C[i, j-1], if\ s3_{i+j} = s2_j, (i, j > 0)} \\ \boldsymbol{c[0, j-1], if\ i = 0\ and\ s3_j = s2_j} \\ \boldsymbol{c[i-1, 0], if\ j = 0\ and\ s3_i = s1_i} \\ \boldsymbol{True,\ if\ i = 0\ and\ j = 0} \end{cases}$$

- **Pseudo-code –**

```
IS(S1, S2, S3):
   r ← Length (S1)
   c ← Length (S2)

   dp[0][0] = True
   for i ← 1 to r do dp[i][0] ← s3[i] == s1[i] and dp[i-1][0]
   for j ← 1 to c do dp[0][j] ← s3[j] == s1[j] and dp[0][j=1]

   for i ← 1 to r do
     for j ← 1 to c do
```

Dhanraj Bhosale (dbhosal1@asu.edu)
ASU ID: 1225506620

```
        if s1[i] == s3[i+j] then
            dp[i][j] ← dp[i-1][j]
        else if s2[j] == s3[i+j] then
            dp[i][j] ← dp[i][j-1]
    return dp[r-1][c-1]
```

Q. (4)

Longest Palindromic Subsequence Formulation-

Given String S = < s_0, s_1, …, s_m>

C [i, j] = length of the Longest Palindromic Subsequence of substring starting from i to j index.
i.e. < s_i, s_{i+1}, …, s_j>

Case 1: If i=j:
single char is palindromic
return 1

Case 2: $xi = xj$
Add these 2 to length to the inside window c[i+1, j-1]
2 + c [i+1, j-1]

Case 3: $xi \neq xj$:
Take max of inner windows
Max(c[i+1, j], c[I, j-1])

$$\textbf{C [i, j]} = \begin{cases} \boldsymbol{C[i+1, j-1] + 2, if\ xi = xj, (i, j > 0)} \\ \boldsymbol{\max[C[i+1, j], C[i, j-1]], if\ if\ xi \neq xj(i, j > 0)} \\ \boldsymbol{1,\ if\ i = j} \end{cases}$$

- **Pseudo-code –**

```
LPS(S):
  n ← Length (S)
  for i ← 0 to n-1 do c[i, i] ← 1
  // cl is window length
  // now loop window from 2 to n length
  for cl ← 2 to  cl<=n do
    for i ← 0 to i<n-cl+1 do
      j = i+cl-1
      if xi = xj and cl==2 then
        c[i, j] ← 2
      else if xi = xj then
        c[i, j] ← c[i+1, j-1] + 2
      Else:
        c[i, j] ← max(c[i+1, j], c[i, j-1])
  return  c[0,n-1]
```