Code LCS:

```
def lcs(s1, s2, m, n):
    if m==0 or n==0:
        return 0

    if s1[m-1] == s2[n-1]:
        return 1 + lcs(s1, s2, m-1, n-1)

    else:
        return max(lcs(s1, s2, m-1, n), lcs(s1, s2, m, n-1))

def lcs(X, Y):
    m = len(X)
    n = len(Y)

    mm = [[0] * (n + 1) for _ in range(m + 1)]

    for i in range(1, m + 1):
        for j in range(1, n + 1):
            if X[i - 1] == Y[j - 1]:
                mm[i][j] = mm[i - 1][j - 1] + 1
            else:
                mm[i][j] = max(mm[i - 1][j], mm[i][j - 1])

    return mm[m][n]
```

```
def lcs_print(x, y, m, n):
    if m == 0 or n == 0:
        return ""

    if x[m-1] == y[n-1]:
        return lcs_print(x, y, m-1, n-1) + x[m-1]

    else:
        lcs1 = lcs_print(x, y, m-1, n)
        lcs2 = lcs_print(x, y, m, n-1)

        if len(lcs1) > len(lcs2):
            return lcs1
        else:
            return lcs2
```

Code Matrix_chain_multiplication:

```
def mat_chain_multiply(arr, i, j):
    if j - i <= 1:
        return 0

    min_cost = sys.maxsize

    for k in range(i + 1, j):
        cost = (mat_chain_multiply(arr, i, k) +
                mat_chain_multiply(arr, k, j) +
                arr[i] * arr[k] * arr[j])
```

```
            min_cost = min(min_cost, cost)

    return min_cost

def mat_chain_mult(p):
    n = len(p) - 1
    m = [[0 for _ in range(n)] for _ in range(n)]

    for l in range(2, n + 1):
        for i in range(n - l + 1):
            j = i + l - 1
            m[i][j] = float('inf')

            for k in range(i, j):
                q = m[i][k] + m[k + 1][j] + p[i] * p[k + 1] * p[j + 1]

                if q < m[i][j]:
                    m[i][j] = q

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