Part A

Aim:

1. Dynamic programming
2. Matrix Chain Multiplication

Prerequisite: Any programming language

Outcome: Algorithms and their implementation

Theory:

Given a sequence of matrices, find the most efficient way to multiply these matrices together. The problem is not actually to perform the multiplications, but merely to decide in which order to perform the multiplications.

Procedure:

1. Design algorithm and find best, average and worst-case complexity
2. Implement algorithm in any programming language.
3. Paste output

Practice Exercise:

S.no	Statement
1	Applying dynamic programming methodology to find minimum number of multiplications to multiply the chain of matrices. Also find the order in which we should multiply the algorithm in order to minimize the no of multiplication.
2	Find the run time complexity of the above algorithm

Instructions:

1. Design, analysis and implement the algorithms.
2. Paste the snapshot of the output in input & output section.

Part B

Matrix chain multiplication

Input: Order of some matrices

Output: Minimum number of operations(multiplications) required and the order of matrices.

Algorithm:

Finding the minimum number of multiplications required and finding split matrix

```
def min_matrix(mat,split):
    l=len(mat)
    for i in range(1,l):
        split[i][i] = i
```

```
    for d in range(1,l-1):
        for i in range(1,l-d):
            j=i+d
            min=999999999
            for k in range(i,j):
                val=mat[i][k]+mat[k+1][j]+arr_d[i-1]*arr_d[k]*arr_d[j]
                if(val<min):
                    min=val
                    split[i][j]=k

            mat[i][j]=min
    return mat[1][l-1]
```

Printing order of matrices for multiplication using split matrix

```
def order_mat(split,i,j):
    if(i==j):
        print(chr(64+split[i][j]),end="")
    else:
        print('(',end="")
        order_mat(split,i,split[i][j])
        order_mat(split,split[i][j]+1,j)
        print(')',end="")
```

Code:

```
def min_matrix():
    global mat, split
    l=len(mat)
    for i in range(1,l):
        split[i][i] = i
    for d in range(1,l-1):
        for i in range(1,l-d):
            j=i+d
            min=999999999
            for k in range(i,j):
                val=mat[i][k]+mat[k+1][j]+arr_d[i-1]*arr_d[k]*arr_d[j]
                if(val<min):
                    min=val
                    split[i][j]=k

            mat[i][j]=min
    return mat[1][l-1]
```

5	4	3	2

0	0	0	0	
–	–	60	64	
–	–	–	24	
–	–	–	–	

4	0	0	0
0	1	1	
0		2	
0			3

```
def order_mat(split,i,j):
    if(i==j):
        print(chr(64+split[i][j]),end="")
    else:
        print('(',end="")
        order_mat(split,i,split[i][j])
        order_mat(split,split[i][j]+1,j)
        print(')',end="")

n=int(input())
arr_d=list(map(int,input().split()))
print('Given matrices ',end="")
for i in range(len(arr_d)-1):
    print('  ',arr_d[i],'x',arr_d[i+1],end=" ")
mat=[[0 for j in range(n)] for i in range(n)]
split=[[0 for j in range(n)] for i in range(n)]
print('\n_____________________\nMinimum number of operations(multiplications) required',min_matrix())
print('order of matrices chosen is ',end=" ")
order_mat(split,1,n-1)
```

Input and Output:

```
PS E:\books and pdfs\sem4 pdfs\DAA lab\week11> python .\multiply.py
6
4 10 3 12 20 7
Given matrices     4 x 10     10 x 3     3 x 12     12 x 20     20 x 7

_____________________
Minimum number of operations(multiplications) required 1344
order of matrices chosen is  ((AB)((CD)E))
PS E:\books and pdfs\sem4 pdfs\DAA lab\week11> python .\multiply.py
5
5 4 6 2 7
Given matrices     5 x 4     4 x 6     6 x 2     2 x 7

_____________________
Minimum number of operations(multiplications) required 158
order of matrices chosen is  ((A(BC))D)
PS E:\books and pdfs\sem4 pdfs\DAA lab\week11> python .\multiply.py
4
1 2 3 4
Given matrices     1 x 2     2 x 3     3 x 4

_____________________
Minimum number of operations(multiplications) required 18
order of matrices chosen is  ((AB)C)
PS E:\books and pdfs\sem4 pdfs\DAA lab\week11>
```

Run time complexity of Matrix Chain Multiplication:

The time complexity of matrix chain multiplication using dynamic programming is O(n^3), where n is the number of matrices.
This is because we traverse 3 nested loops.

Space complexity:

The space complexity of matrix chain multiplication using dynamic programming is O(n^2), as we used a 2-D list

Observation & Learning:

I have observed and learned that
i) Matrix chain multiplication uses the tabularization method(Bottom-up approach).
ii) The idea is to use memorization.

Conclusion:

I have successfully written and executed the Matrix chain multiplication algorithm in the python programming language.