KATHMANDU UNIVERSITY

DHULIKHEL KAVRE

Subject: COMP 314: Algorithms and Complexity
Lab no: 2

Submitted By:

Name: Ayush Kumar Shah
Roll no: 44
Group: CE 3rd year 2nd sem
Level: UNG

Submitted To:

Dr. Bal Krishna Bal

Date of Submission:
27/05/2018

Lab 2 – COMP 314 – Algorithms and Complexity
Objectives:

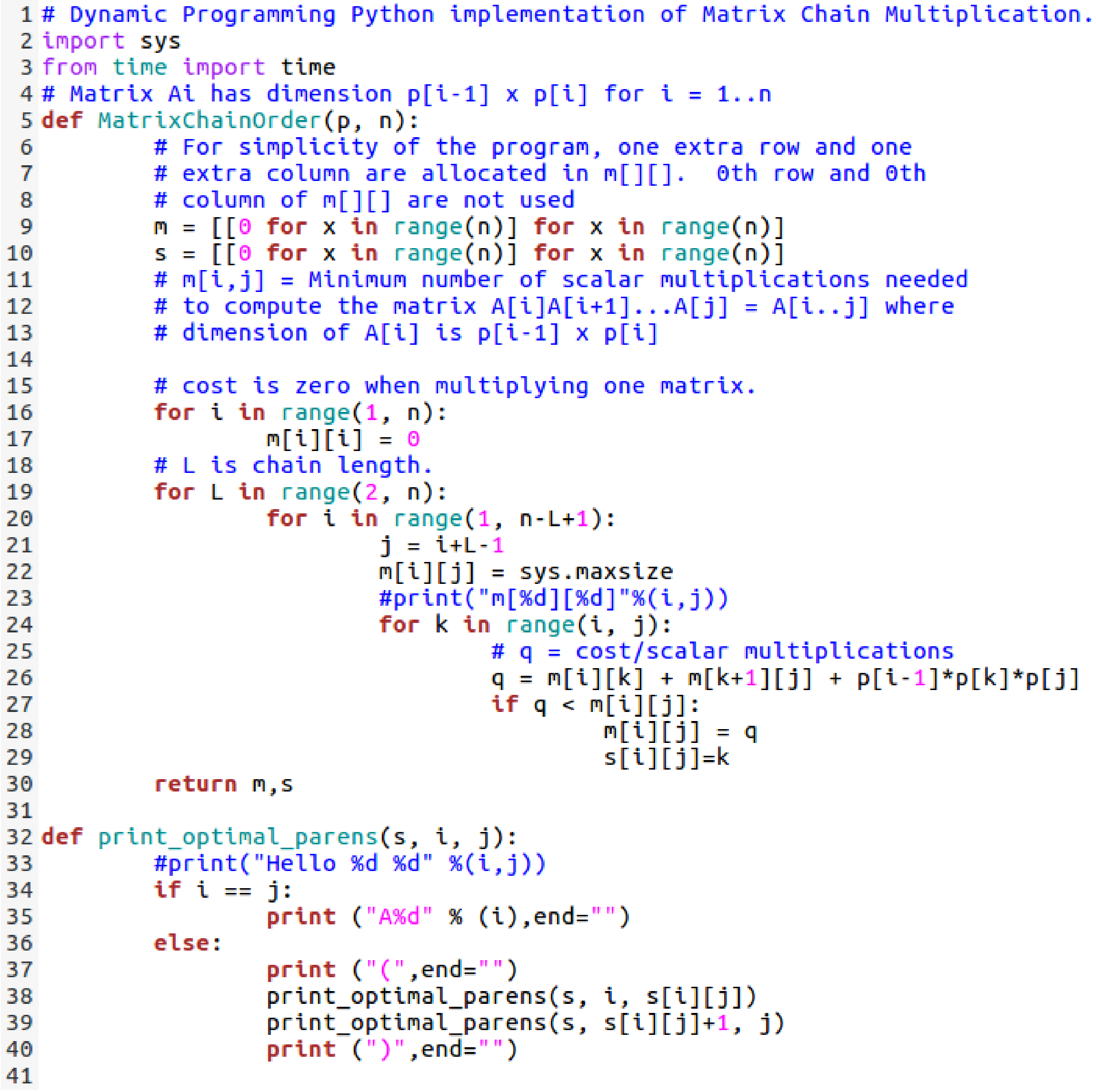
Understanding Dynamic Programming through the coding of the problems - “Matrix Chain Multiplication” and “Longest Common Subsequence”.

Matrix Chain Multiplication Problem:

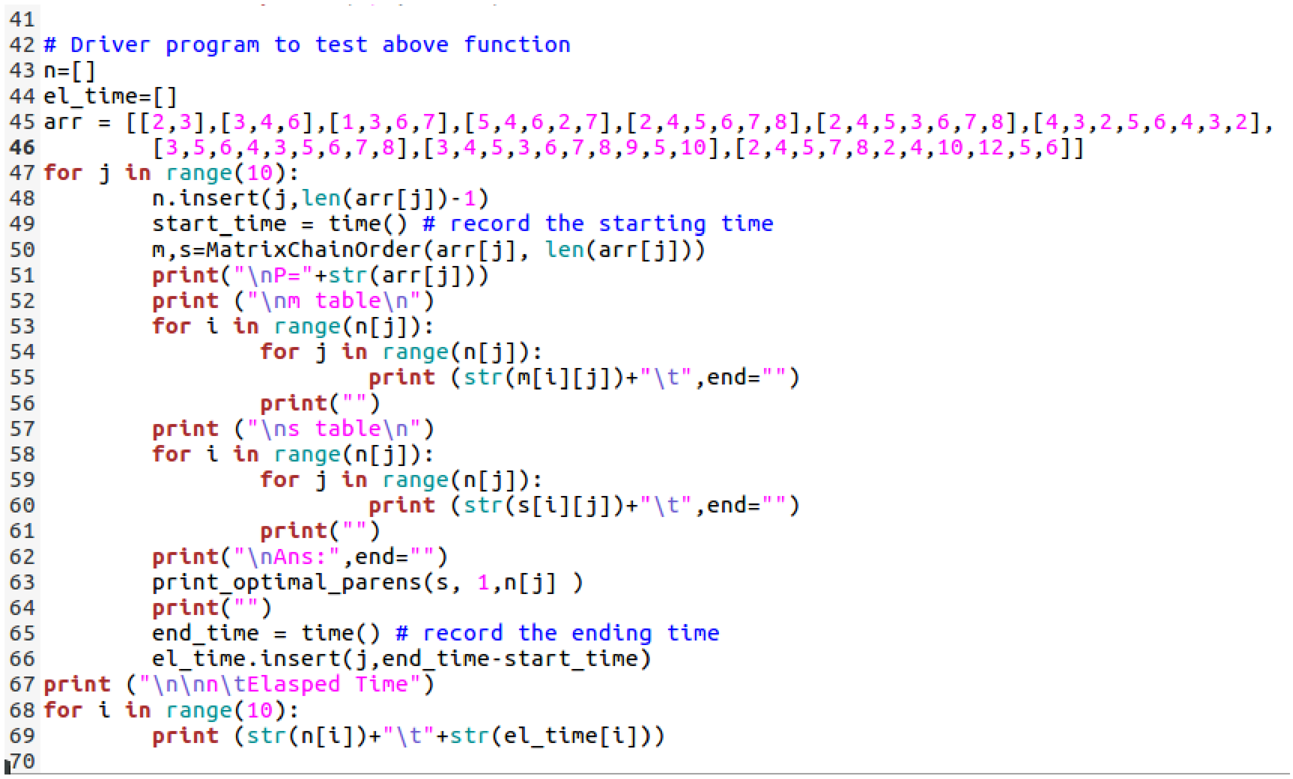
For the matrix chain multiplication problem, you would need to develop a Python code that would not only give you the minimum cost for the multiplication of a sequence of matrices but also suggest the optimum split values. Build the ‘m’ and ‘s’ tables. Based on the split value for the minimum cost of multiplying the given sequence of matrices, also output the optimal parenthesization string of matrices.

You may reuse the code available at http://www.geeksforgeeks.org/dynamic-programming-set-8-matrix-chain-multiplication/
Take different sizes of matrices and record the time required to get the solutions and plot a graph to see whether your results confirm to O(n^3) time complexity, where is the size of the matrix chain.

Ans: Python code:



```python
# Dynamic Programming Python implementation of Matrix Chain Multiplication.
import sys
from time import time
# Matrix Ai has dimension p[i-1] x p[i] for i = 1..n
def MatrixChainOrder(p, n):
        # For simplicity of the program, one extra row and one
        # extra column are allocated in m[][].  0th row and 0th
        # column of m[][] are not used
        m = [[0 for x in range(n)] for x in range(n)]
        s = [[0 for x in range(n)] for x in range(n)]
        # m[i,j] = Minimum number of scalar multiplications needed
        # to compute the matrix A[i]A[i+1]...A[j] = A[i..j] where
        # dimension of A[i] is p[i-1] x p[i]

        # cost is zero when multiplying one matrix.
        for i in range(1, n):
                m[i][i] = 0
        # L is chain length.
        for L in range(2, n):
                for i in range(1, n-L+1):
                        j = i+L-1
                        m[i][j] = sys.maxsize
                        #print("m[%d][%d]"%(i,j))
                        for k in range(i, j):
                                # q = cost/scalar multiplications
                                q = m[i][k] + m[k+1][j] + p[i-1]*p[k]*p[j]
                                if q < m[i][j]:
                                        m[i][j] = q
                                        s[i][j]=k
        return m,s

def print_optimal_parens(s, i, j):
        #print("Hello %d %d" %(i,j))
        if i == j:
                print ("A%d" % (i),end="")
        else:
                print ("(",end="")
                print_optimal_parens(s, i, s[i][j])
                print_optimal_parens(s, s[i][j]+1, j)
                print (")",end="")

```



```python
# Driver program to test above function
n=[]
el_time=[]
arr = [[2,3],[3,4,6],[1,3,6,7],[5,4,6,2,7],[2,4,5,6,7,8],[2,4,5,3,6,7,8],[4,3,2,5,6,4,3,2],
        [3,5,6,4,3,5,6,7,8],[3,4,5,3,6,7,8,9,5,10],[2,4,5,7,8,2,4,10,12,5,6]]
for j in range(10):
        n.insert(j,len(arr[j])-1)
        start_time = time() # record the starting time
        m,s=MatrixChainOrder(arr[j], len(arr[j]))
        print("\nP="+str(arr[j]))
        print ("\nm table\n")
        for i in range(n[j]):
                for j in range(n[j]):
                        print (str(m[i][j])+"\t",end="")
                print("")
        print ("\ns table\n")
        for i in range(n[j]):
                for j in range(n[j]):
                        print (str(s[i][j])+"\t",end="")
                print("")
        print("\nAns:",end="")
        print_optimal_parens(s, 1,n[j] )
        print("")
        end_time = time() # record the ending time
        el_time.insert(j,end_time-start_time)
print ("\n\nn\tElasped Time")
for i in range(10):
        print (str(n[i])+"\t"+str(el_time[i]))
```

Output:

```
P=[2, 3]

m table

0

s table

0

Ans:A1

P=[3, 4, 6]

m table

0       0
0       0

s table

0       0
0       0

Ans:(A1A2)
```

P=[1, 3, 6, 7]

m table

0	0	0
0	0	18
0	0	0

s table

0	0	0
0	0	1
0	0	0

Ans:((A1A2)A3)

P=[5, 4, 6, 2, 7]

m table

0	0	0	0
0	0	120	88
0	0	0	48
0	0	0	0

s table

0	0	0	0
0	0	1	1
0	0	0	2
0	0	0	0

Ans:((A1(A2A3))A4)

P=[2, 4, 5, 6, 7, 8]

m table

0	0	0	0	0
0	0	40	100	184
0	0	0	120	288
0	0	0	0	210
0	0	0	0	0

s table

0	0	0	0	0
0	0	1	2	3
0	0	0	2	3
0	0	0	0	3
0	0	0	0	0

Ans:((((A1A2)A3)A4)A5)

P=[2, 4, 5, 3, 6, 7, 8]

m table

0	0	0	0	0	0
0	0	40	70	106	190
0	0	0	60	132	270
0	0	0	0	90	231
0	0	0	0	0	126
0	0	0	0	0	0

s table

0	0	0	0	0	0
0	0	1	2	3	4
0	0	0	2	3	3
0	0	0	0	3	3
0	0	0	0	0	4
0	0	0	0	0	0

Ans:(((((A1A2)A3)A4)A5)A6)

P=[4, 3, 2, 5, 6, 4, 3, 2]

m table

0	0	0	0	0	0	0
0	0	24	64	132	164	180
0	0	0	30	96	132	150
0	0	0	0	60	108	132
0	0	0	0	0	120	162
0	0	0	0	0	0	72
0	0	0	0	0	0	0

s table

0	0	0	0	0	0	0
0	0	1	2	2	2	2
0	0	0	2	2	2	2
0	0	0	0	3	4	5
0	0	0	0	0	4	4
0	0	0	0	0	0	5
0	0	0	0	0	0	0

Ans:(A1(A2((((A3A4)A5)A6)A7)))

P=[3, 5, 6, 4, 3, 5, 6, 7, 8]
m table

0	0	0	0	0	0	0	0
0	0	90	162	198	243	333	459
0	0	0	120	162	237	342	483
0	0	0	0	72	162	270	414
0	0	0	0	0	60	162	300
0	0	0	0	0	0	90	216
0	0	0	0	0	0	0	210
0	0	0	0	0	0	0	0

s table

0	0	0	0	0	0	0	0
0	0	1	2	3	4	5	6
0	0	0	2	2	4	4	4
0	0	0	0	3	4	4	4
0	0	0	0	0	4	4	4
0	0	0	0	0	0	5	6
0	0	0	0	0	0	0	6
0	0	0	0	0	0	0	0

Ans:(((((((A1A2)A3)A4)A5)A6)A7)A8)

P=[3, 4, 5, 3, 6, 7, 8, 9, 5, 10]

m table

0	0	0	0	0	0	0	0	0
0	0	60	96	150	276	444	660	786
0	0	0	60	132	270	450	678	765
0	0	0	0	90	231	414	645	720
0	0	0	0	0	126	294	510	645
0	0	0	0	0	0	336	768	850
0	0	0	0	0	0	0	504	640
0	0	0	0	0	0	0	0	360
0	0	0	0	0	0	0	0	0

s table

0	0	0	0	0	0	0	0	0
0	0	1	1	3	4	5	6	3
0	0	0	2	3	3	3	3	3
0	0	0	0	3	3	3	3	3
0	0	0	0	0	4	5	6	7
0	0	0	0	0	0	5	6	5
0	0	0	0	0	0	0	6	6
0	0	0	0	0	0	0	0	7
0	0	0	0	0	0	0	0	0

Ans:(((A1(A2A3))((((A4A5)A6)A7)A8))A9)

P=[2, 4, 5, 7, 8, 2, 4, 10, 12, 5, 6]

m table

0	0	0	0	0	0	0	0	0	0
0	0	40	110	222	238	254	334	574	694
0	0	0	140	364	222	254	382	638	702
0	0	0	0	280	182	222	362	622	672
0	0	0	0	0	112	168	332	600	622
0	0	0	0	0	0	64	240	512	520
0	0	0	0	0	0	0	80	320	440
0	0	0	0	0	0	0	0	480	720
0	0	0	0	0	0	0	0	0	600
0	0	0	0	0	0	0	0	0	0

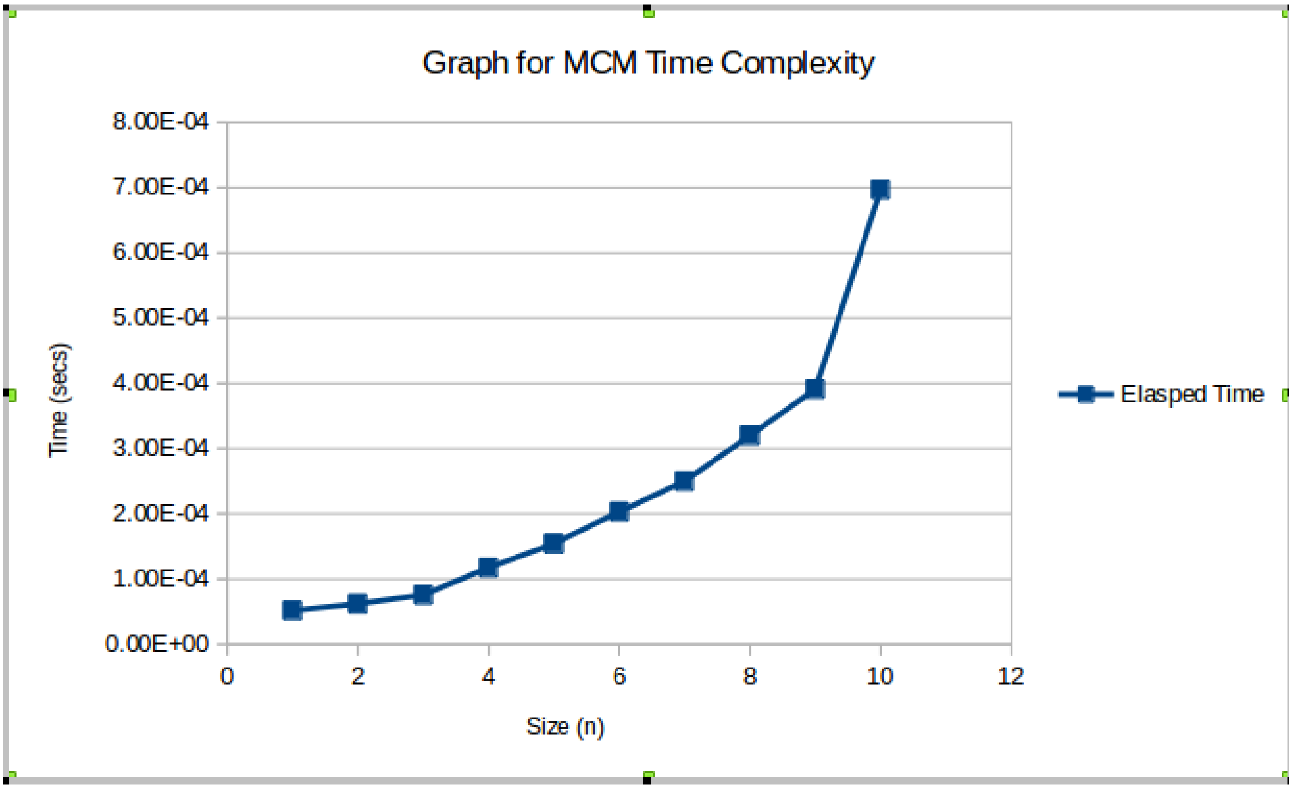
s table

0	0	0	0	0	0	0	0	0	0
0	0	1	2	3	1	5	6	7	8
0	0	0	2	3	2	5	5	5	5
0	0	0	0	3	3	5	5	5	5
0	0	0	0	0	4	5	5	5	5
0	0	0	0	0	0	5	5	5	5
0	0	0	0	0	0	0	6	7	8
0	0	0	0	0	0	0	0	7	8
0	0	0	0	0	0	0	0	0	8
0	0	0	0	0	0	0	0	0	0

Ans:((((((A1(A2(A3(A4A5))))A6)A7)A8)A9)A10)

n	Elasped Time
1	5.173683166503906e-05
2	6.222724914550781e-05
3	7.581710815429688e-05
4	0.00011754035949707031
5	0.00015425682067871094
6	0.0002033710479736328
7	0.00025010108947753906
8	0.00032019615173339844
9	0.00039124488830566406
10	0.0006971359252929688

Graph:



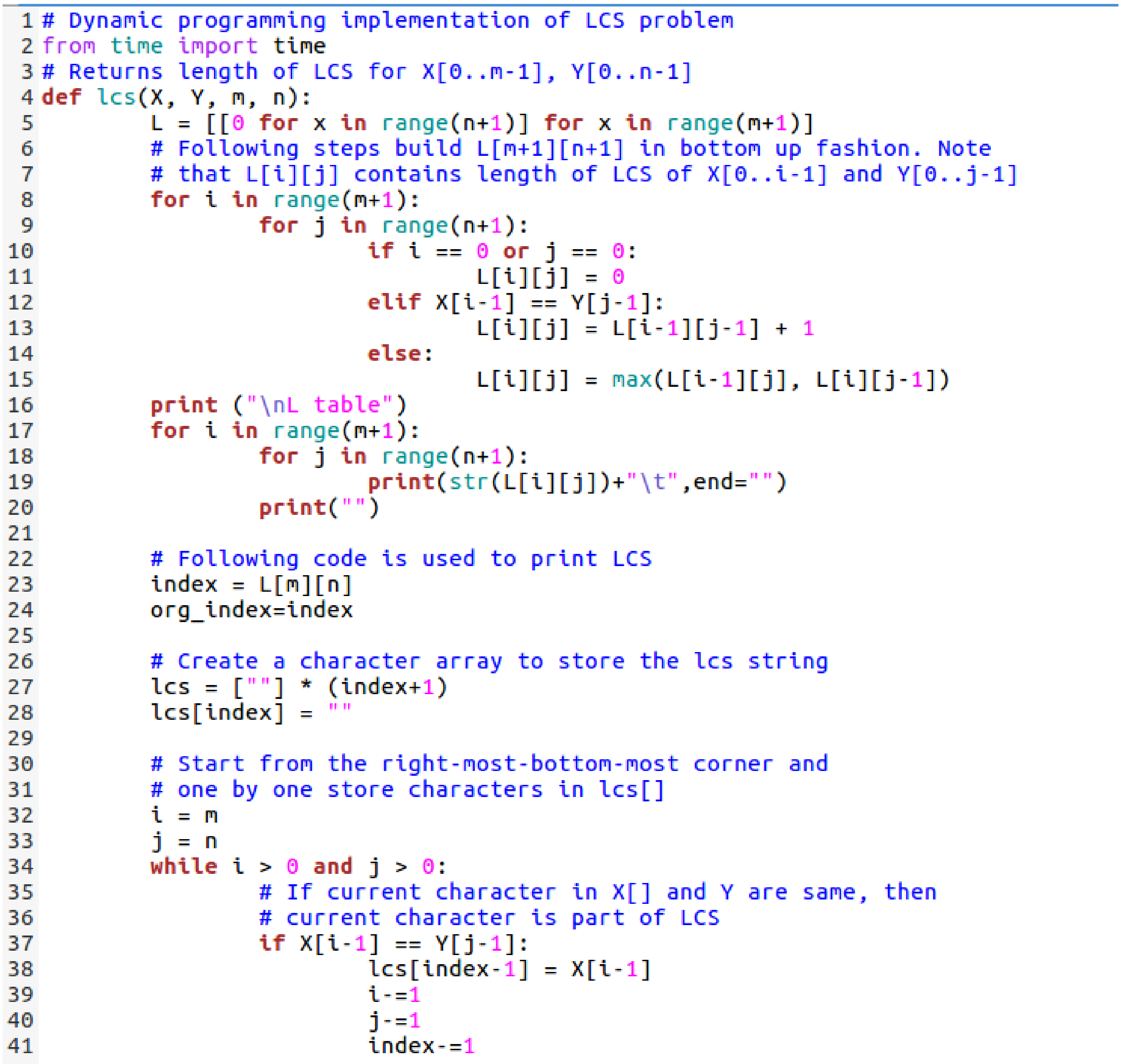
Hence, the graph shows that the time complexity of Matrix Chain Multiplication is O(n^3)

Longest Common Subsequence Problem:

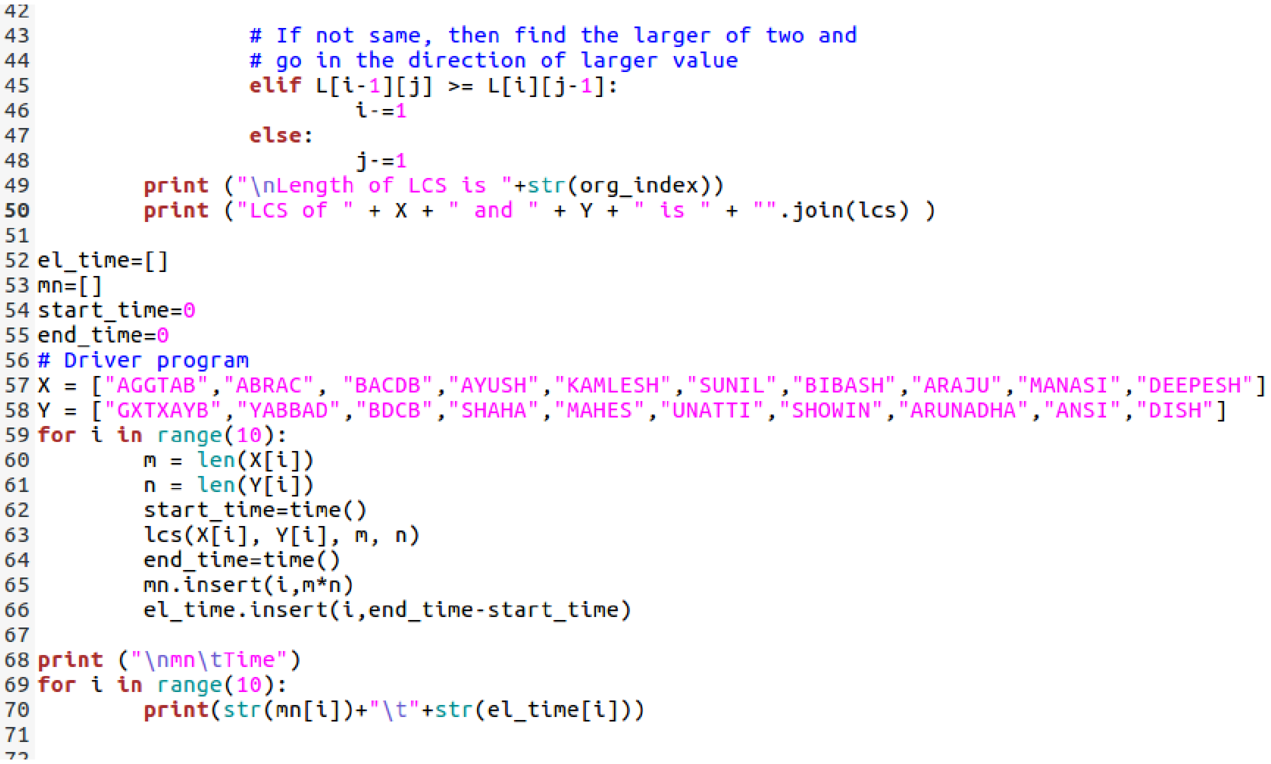
http://www.geeksforgeeks.org/dynamic-programming-set-4-longest-common-subsequence/
Implement the Dynamic Programming implementation of the Longest Common Subsequence Problem.

Provide 10 random string pairs and calculate the length of the LCS as well as print the LCS string. Record the times spent for calculating the lengths of the LCS as well as the LCS string and plot a graph to see whether it conforms to O(mn) time complexities where “m” and “n” respectively refer to the lengths of the first and the second strings.

Ans: Python Code:



```python
# Dynamic programming implementation of LCS problem
from time import time
# Returns length of LCS for X[0..m-1], Y[0..n-1]
def lcs(X, Y, m, n):
        L = [[0 for x in range(n+1)] for x in range(m+1)]
        # Following steps build L[m+1][n+1] in bottom up fashion. Note
        # that L[i][j] contains length of LCS of X[0..i-1] and Y[0..j-1]
        for i in range(m+1):
                for j in range(n+1):
                        if i == 0 or j == 0:
                                L[i][j] = 0
                        elif X[i-1] == Y[j-1]:
                                L[i][j] = L[i-1][j-1] + 1
                        else:
                                L[i][j] = max(L[i-1][j], L[i][j-1])
        print ("\nL table")
        for i in range(m+1):
                for j in range(n+1):
                        print(str(L[i][j])+"\t",end="")
                print("")

        # Following code is used to print LCS
        index = L[m][n]
        org_index=index

        # Create a character array to store the lcs string
        lcs = [""] * (index+1)
        lcs[index] = ""

        # Start from the right-most-bottom-most corner and
        # one by one store characters in lcs[]
        i = m
        j = n
        while i > 0 and j > 0:
                # If current character in X[] and Y are same, then
                # current character is part of LCS
                if X[i-1] == Y[j-1]:
                        lcs[index-1] = X[i-1]
                        i-=1
                        j-=1
                        index-=1
```



```python

                # If not same, then find the larger of two and
                # go in the direction of larger value
                elif L[i-1][j] >= L[i][j-1]:
                        i-=1
                else:
                        j-=1
        print ("\nLength of LCS is "+str(org_index))
        print ("LCS of " + X + " and " + Y + " is " + "".join(lcs) )

el_time=[]
mn=[]
start_time=0
end_time=0
# Driver program
X = ["AGGTAB","ABRAC", "BACDB","AYUSH","KAMLESH","SUNIL","BIBASH","ARAJU","MANASI","DEEPESH"]
Y = ["GXTXAYB","YABBAD","BDCB","SHAHA","MAHES","UNATTI","SHOWIN","ARUNADHA","ANSI","DISH"]
for i in range(10):
        m = len(X[i])
        n = len(Y[i])
        start_time=time()
        lcs(X[i], Y[i], m, n)
        end_time=time()
        mn.insert(i,m*n)
        el_time.insert(i,end_time-start_time)

print ("\nmn\tTime")
for i in range(10):
        print(str(mn[i])+"\t"+str(el_time[i]))

```

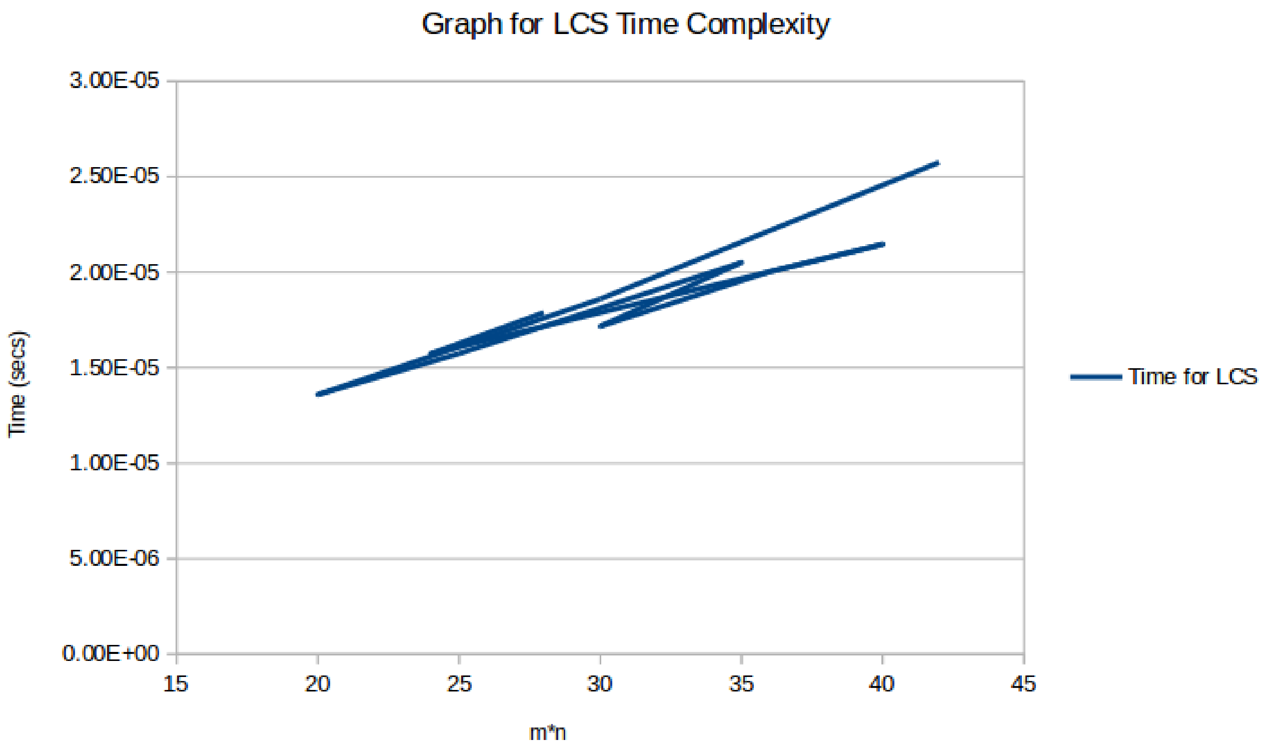
Output:

```
L table
0       0       0       0       0       0       0       0
0       0       0       0       0       1       1       1
0       1       1       1       1       1       1       1
0       1       1       1       1       1       1       1
0       1       1       2       2       2       2       2
0       1       1       2       2       3       3       3
0       1       1       2       2       3       3       4

Length of LCS is 4
LCS of AGGTAB and GXTXAYB is GTAB

L table
0       0       0       0       0       0       0
0       0       1       1       1       1       1
0       0       1       2       2       2       2
0       0       1       2       2       2       2
0       0       1       2       2       3       3
0       0       1       2       2       3       3

Length of LCS is 3
LCS of ABRAC and YABBAD is ABA

L table
0       0       0       0       0
0       1       1       1       1
0       1       1       1       1
0       1       1       2       2
0       1       2       2       2
0       1       2       2       3

Length of LCS is 3
LCS of BACDB and BDCB is BCB

L table
0       0       0       0       0       0
0       0       0       1       1       1
0       0       0       1       1       1
0       0       0       1       1       1
0       1       1       1       1       1
0       1       2       2       2       2

Length of LCS is 2
LCS of AYUSH and SHAHA is AH
```

```
L table
0       0       0       0       0       0
0       0       0       0       0       0
0       0       1       1       1       1
0       1       1       1       1       1
0       1       1       1       1       1
0       1       1       1       2       2
0       1       1       1       2       3
0       1       1       2       2       3

Length of LCS is 3
LCS of KAMLESH and MAHES is AES

L table
0       0       0       0       0       0       0
0       0       0       0       0       0       0
0       1       1       1       1       1       1
0       1       2       2       2       2       2
0       1       2       2       2       2       3
0       1       2       2       2       2       3

Length of LCS is 3
LCS of SUNIL and UNATTI is UNI

L table
0       0       0       0       0       0       0
0       0       0       0       0       0       0
0       0       0       0       0       1       1
0       0       0       0       0       1       1
0       0       0       0       0       1       1
0       1       1       1       1       1       1
0       1       2       2       2       2       2

Length of LCS is 2
LCS of BIBASH and SHOWIN is SH

L table
0       0       0       0       0       0       0       0       0
0       1       1       1       1       1       1       1       1
0       1       2       2       2       2       2       2       2
0       1       2       2       2       3       3       3       3
0       1       2       2       2       3       3       3       3
0       1       2       3       3       3       3       3       3

Length of LCS is 3
LCS of ARAJU and ARUNADHA is ARA
```

```
L table
0       0       0       0       0
0       0       0       0       0
0       1       1       1       1
0       1       2       2       2
0       1       2       2       2
0       1       2       3       3
0       1       2       3       4

Length of LCS is 4
LCS of MANASI and ANSI is ANSI

L table
0       0       0       0       0
0       1       1       1       1
0       1       1       1       1
0       1       1       1       1
0       1       1       1       1
0       1       1       1       1
0       1       1       2       2
0       1       1       2       3

Length of LCS is 3
LCS of DEEPESH and DISH is DSH

mn              Time
42              2.574920654296875e-05
30              1.8596649169921875e-05
20              1.3589859008789062e-05
25              1.5735626220703125e-05
35              2.0503997802734375e-05
30              1.71661376953125e-05
36              2.002716064453125e-05
40              2.1457672119140625e-05
24              1.5735626220703125e-05
28              1.7881393432617188e-05
```

Graph:



Hence, the graph shows that the time complexity of Longest Common Subsequence is O(mn)