Q. 32 Given 3 strings of all having length < 100, write a program to find the longest common sub-sequence in all three given sequences.

Algorithm LCS(X,Y,Z):

INPUT: Three strings.
OUTPUT: Length of longest subsequence.

```
set m = length(X)
set n = length(Y)
set o = length(Z)
set c = [o+1,n+1,m+1]
FOR i 0 to m+1 DO:
  c[i,0,0] = 0
FOR i 0 to n+1 DO:
  c[0,i,0] = 0
FOR i 0 to o+1 DO:
  c[0,0,i] = 0
FOR i 0 to m DO:
  FOR i 0 to n DO:
    FOR i 0 to o DO:
      IF (X[i] == Y[j] and Y[j]==Z[k] ) THEN DO:
        c[i,j,k] = c[i-1,j-1,k-1]+1
      ELSE IF c[i-1,j,k] >= c[i,j-1,k] and c[i-1,j,k] >= c[i,j,k-1] :
        c[i,j,k] = c[i-1,j,k]
      ELSE IF c[i,j-1,k] >= c[i-1,j,k] and  c[i,j-1,k] >= c[i,j,k-1]:
        c[i,j,k] = c[i,j-1,k]
      ELSE:
        c[i,j,k] = c[i,j,k-1]

RETURN c[m-1,n-1,o-1]
```

Algorithm Analysis:

T(n) = $O(n^3)$

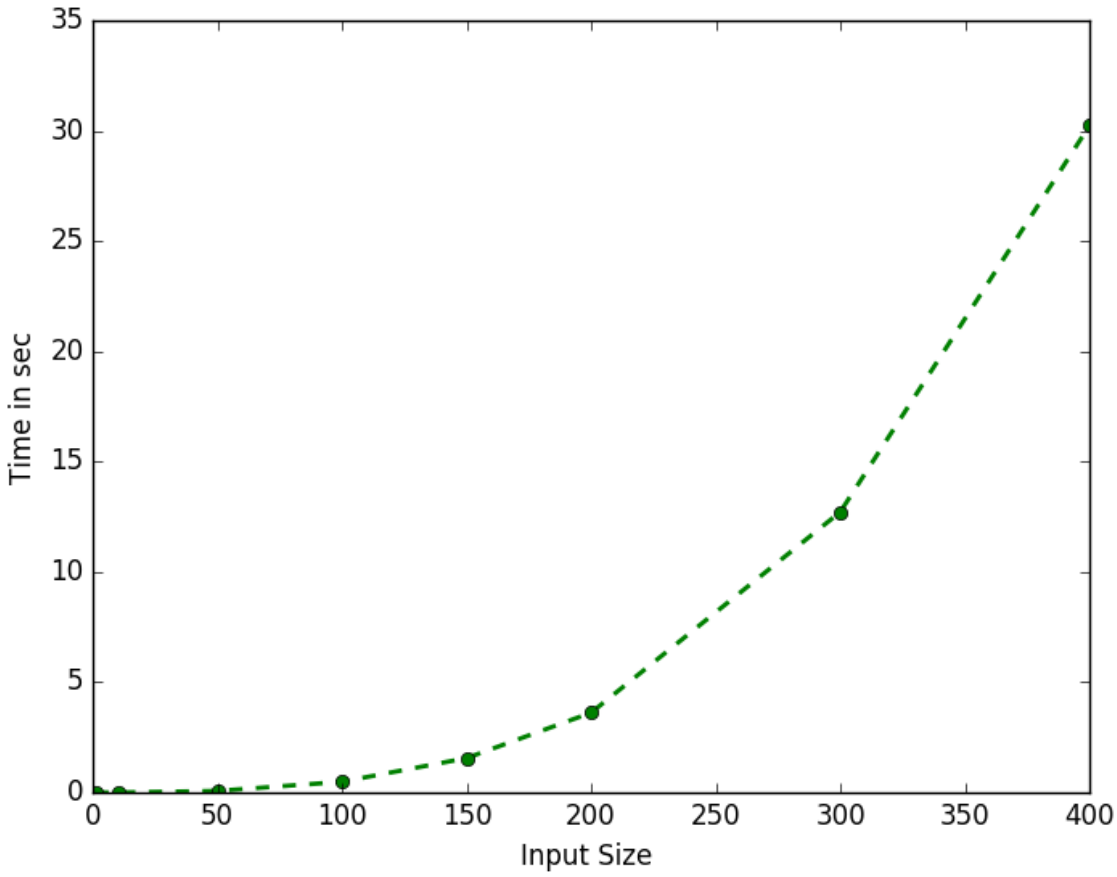
C[i,j,k] =

0	**for i,j,k = 0**
	for i,j,k > 0
c[i-1,j-1,k-1]+1	**for $x_i = Y_j = Z_k$**
max(c[i-1,j,k],c[i,j-1,k],c[i,j,k-1])	**for x_i != Y_j != Z_k**

Input/Output:
Input is n = string length.
At, CPU: Dual Core 3Ghz, Memory: 4GB

Input	Time (in Sec)
1	0.000107049942017
10	0.000690937042236
50	0.0716681480408
100	0.486089944839
150	1.55700111389
200	3.61730313301
300	12.7171878815
400	30.2279579639

Complexity Graph:



Conclusion :
Since this algorithm is to compare 3 strings to find longestsubsequence using dynamic programming it takes $O(n^3)$ time complexity.

Q. 33(A) 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.

Algorithm MCM(p):

INPUT: Dimensions of Matrix.
OUTPUT: Minimum number of multiplication required.

```
set n = len(p)
set M = [[0 for i in range(0,n)] for i in range(0,n)]
set S = [[0 for i in range(0,n)] for i in range(0,n)]
FOR i (1 to n) DO:
  M[i][i] = 0
FOR l in 2 to n DO:
  FOR i 1 to (n-l+1) DO:
    set j = i+l -1
    set M[i][j]= INFINITY
    FOR k i to j:
      set q = M[i][k] + M[k+1][j] + p[i-1]*p[k]*p[j]
      IF q < M[i][j] THEN DO:
        set M[i][j] = q
        set S[i][j] = k
RETURN M[1][n-1],S
```

Algorithm Analysis:

T(n) = O(n³)

$$M[i,j] = \begin{cases} 0 & \text{for } i = j \\ \min(M[i][k] + M[k+1][j] + p[i-1]*p[k]*p[j]) & \text{for } i<j \end{cases}$$

Conclusion :
Matrix chain multiplication algorithm uses dynamic programming to store previous result to optimaly solve problem by reducing number of overlapping problems.

Q. 33(B) Write a program to find the contiguous subsequence of maximum sum (a subsequence of length zero has sum zero). A contiguous subsequence of a list S is a subsequence made up of consecutive elements of S. For instance, if S is 5; 15;-30; 10;-5; 40; 10; then 15;-30; 10 is a contiguous subsequence but 5; 15; 40 is not. For the preceding example, the answer would be 10;-5; 40; 10, with a sum of 55. Give a linear-time algorithm for the following task:

Algorithm CONTIGUOUSSUM(a):

INPUT: List of Integers.
OUTPUT: Maximum contiguous sum.

```
set maxSum = -INFINITY
set maxend = 0
set maxINew = 0
set maxI = 0
set maxJ = 0
FOR i ( 0 to |a| ) DO:
    set maxend = maxend + a[i]
    IF(maxend < 0) THEN DO:
      set maxend = 0
      set maxINew = i+1
    IF(maxSum < maxend) THEN DO:
      set maxI = maxINew
      set maxSum = maxend
      set maxJ = i
FOR i maxI to (maxJ+1) DO:
    PRINT(a[i])
RETURN maxSum
```

Algorithm Analysis:

T(n) = O(n)

Conclusion :
Using certain flag above algorithm stores previous sum and use it to find max sum for contiguous sub sequence.

Q. 34. A subsequence is palindromic if it is the same whether read left to right or right to left. For instance, the sequence A;C; G; T; G; T;C; A; A; A; A; T;C;G has many palindromic subsequences, including A;C; G;C;A and A; A; A;A (on the other hand, the subsequence A;C; T is not palindromic). Devise an algorithm that takes a sequence x[1 : : : n] and returns the (length of the) longest palindromic subsequence. Its running time should be O(n2).

Algorithm PELINDROMSUBSEQ(s):

INPUT: string.
OUTPUT: longest pelindrom subsequence length.

```
set n = len(s)
set L = [[0 for i in range(0,n)] for i in range(0,n)]
FOR i (0 to n) DO:
    L[i][i] = 1
FOR i 2 to (n+1) DO:
  FOR j (0 to n-i+1) DO:
      set k = i + j -1
      IF s[k] == s[j] and i==2 THEN DO:
        set L[j][k] = 2
      ELSE IF s[k]==s[j] THEN DO:
        set L[j][k] = L[j+1][k-1] + 2
      ELSE:
        set L[j][k] = max(L[j][k-1],L[j+1][k])
RETURN L[0][n-1]
```

Algorithm Analysis:

T(n) = O(n)
L[i,j] = 2 for i=2 and $s_i = s_j$
= L[j+1,k-1] + 2 for $s_i = s_j$
= max(L[j,k-1],L[j+1,k]) otherwise.

Conclusion :
Initialy for string of length 2 if both character are same L=2 and if length is greater than 2 and last character is same L = L + 2 and so on, using dynamic programming we store this result and if character is not same we find max of L from last stored value in array, thus DP reduces multiple calls in recursion since we already store the value we need not to do calculation multiple times.

Q. 35. A list of n positive integers a1; a2; : : : ; an; and a positive integer t is given. Write a program to find subset of the ai's add up to t? (You can use each ai at most once).

Algorithm SUBSETSUM(a,sum,n):

INPUT: List of positive number.
OUTPUT: True or false wether it is possible to generate t from given numbers.

```
set result = [[False for i in range(0,n+1)] for i in range(0,sum+1)]
FOR i in range(0,n+1):
    result[0][i] = True
FOR i in range(1,sum+1):
    result[i][0] = False
FOR i in range(1,sum+1):
    FOR j in range(1,n+1):
      set result[i][j] = result[i][j-1]
      IF i >= a[j-1] THEN DO:
        set result[i][j]=result[i][j] or result[i-a[j-1]][j-1];
RETURN result[sum][n];
```

Algorithm Analysis:

T(n) = O(sum*n)
L[i,j] = 2 for i=2 and $s_i = s_j$
= L[j+1,k-1] + 2 for $s_i = s_j$
= max(L[j,k-1],L[j+1,k]) otherwise.

Conclusion :
Using a 2D matrix for storing boolean table we can solve this problem O(sum*n) creating a table of sum vs n elements and for every combination we store boolean of wether sum is possible by subset combination and hence stored value can be used further. And as it can be seen that the complexity of above problem is reduced to O(sum*n) times.

Q. 38. Write a program that calculates the highest sum of numbers passed on a route that starts at the top and ends somewhere on the base.

7
3 8
8 1 0
2 7 4 4
4 5 2 6 5
(Figure 1)
For the above figure shows a number triangle and its output is 30(7,3,8,7,5). Each step can go either diagonally down to the left or diagonally down to the right.

Algorithm SUMTRIANGLE(a,n):

INPUT: List of positive number in form of triangle.
OUTPUT: Maximum sum possible starting from top till bottom.

```
FOR i ( (n-2) to -1) DO:
    FOR j  0 to (|a[i]|) DO:
      set a[i][j] = a[i][j] + max(a[i+1][j],a[i+1][j+1])

RETURN a[0][0]
```

Algorithm Analysis:

T(n) = O(n^2)
a[i,j] = a[i,j] + max(a[i+1, j],a[i+1, j+1])

Conclusion :
Instead of going top to bottom we parse the tree from bottom to top and selecting maximum available node value and then adding it to store maximum sum value from top to bottom. Here we use array to store sum and updating its value to found Max sum.