Fibonacci using DP:

*#recursive code*

def fib(n):

    if(n==0 or n==1):

        return n

    return fib(n-1)+fib(n-2)

print(fib(6))

#Size of dp should be minimum of number of unique states, so here unique states are: 0, 1, 2, 3, 4, 5, 6

*#Top down approach*

def fibTD(n,dp):

    if(n==0 or n==1):

        return n

    if dp[n]!=-1:    *#dp[n] value already computed*

        return dp[n]

    dp[n]= fibTD(n-1,dp)+fibTD(n-2,dp)

    return dp[n]

n=6

dp=[-1]\*(n+1)

print(dp)

print(fibTD(n,dp))

print(dp)

*#Bottom Up approach*

def fibBU(n):

    dp=[None]\*(n+1)

    dp[0],dp[1] = 0,1

    for i in range(2,n+1):

        dp[i] = dp[i-1]+dp[i-2]

    return dp[n]

print(fibBU(6))

*#Loot HOUSE*

def lootBU(n,arr):

    dp=[0]\*(n+1)

    dp[0],dp[1] = arr[0],max(arr[0],arr[1])

    for i in range(2,n):

        dp[i] = max(arr[i]+dp[i-2] ,dp[i-1])

    print(dp)

    return dp[n-1]

arr = [6,2,3,9]

print(lootBU(len(arr),arr))

*#minimum steps to 1*

*#Top down*

def minStepsToOneTD(n,dp):

    if n==1: return 0

    if n==2 or n==3: return 1

    if n<1: return float('inf')

    if dp[n]!=0: return dp[n]

    div\_by\_3, div\_by\_2, less\_by\_1 = float('inf'),float('inf'),float('inf')

    if(n%3==0):

        div\_by\_3 = 1+minStepsToOneTD(n//3,dp)

    if(n%2==0):

        div\_by\_2 = 1+minStepsToOneTD(n//2,dp)

    less\_by\_1 = 1+minStepsToOneTD(n-1,dp)

    dp[n]=min(div\_by\_3, div\_by\_2, less\_by\_1)

    return dp[n]

n=7

dp=[0]\*(n+1)

print(minStepsToOneTD(n,dp))

*#min coin*

def min\_coins(n,arr,dp):

    if n==0: return 0

    if dp[n]!=0: return dp[n]

    min\_value = float('inf')

    for i in arr:

        if n-i>=0:

            min\_value = min(min\_value,1+min\_coins(n-i,arr,dp))

    dp[n]=min\_value

    return dp[n]

n=11

arr=[1,5,7]

dp=[0]\*(n+1)

print(min\_coins(n,arr,dp),dp)

*#longest subsequence*

def longest\_subsequence(seq):

    n=len(seq)

    dp=[1]\*n

    for i in range(n):

        for j in range(i):

            if seq[i]>seq[j]:

                dp[i]=max(dp[i],1+dp[j])

    print(dp)

    return dp[n-1]

seq = [3,1,2,5,4,6,5,6]

print(longest\_subsequence(seq))