# Check dimension of array required

# Base case

# code for recurrence relation

# Q. Fibonacci series

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))

# Q. You are given money present in n adjacent houses, There is robber who wants to rob the houses. But he can not rob from 2 adjacent houses. Find max loot of robber.

*#Loot HOUSE*

def lootBU(n,arr):

    dp=[0]\*(n)

    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))

# Q. Given a number x, you can do 3 different operations on x: # 1. Subtract 1 from it # 2. If it is divisible by 2, divide by 2 # 3. If it is divisible by 3, divide by 3 # Find the minimum number of steps that it takes to get to 1 using only the above operations.

*#minimum steps to 1*

*#Top down*

def minStepsToOneTD(n,dp):

    if n==1: return 0

    if n==2 or n==3: return 1

    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))

*# Given a number x, you can do 3 different operations on x:*

*# 1. Subtract 1 from it*

*# 2. If it is divisible by 2, divide by 2*

*# 3. If it is divisible by 3, divide by 3*

*# Find the minimum number of steps that it takes to get to 1 using only the above operations.*

*#Bottom Up*

def min\_steps\_to\_one(x):

    dp = [1]\*(x+1)

    dp[1]=0

    dp[2],dp[3]=1,1

    for i in range(4,x+1):

        dp[i]=1+min(dp[i-1], dp[i//2] if i%2==0 else x, dp[i//3] if i%3==0 else x)

    return dp[x]

print(min\_steps\_to\_one(10))

# [Q. Question](https://cses.fi/problemset/task/1634/)

*#min coin, top down*

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)

*#bottom up approach*

def min\_coins(target,arr):

    dp = [float('inf')]\*(target+1)

    dp[0]=0

    for i in range(1,target+1):

        min\_val = float('inf')

        for j in arr:

            if i>=j:

                min\_val = min(min\_val, 1+dp[i-j])

        dp[i] = min\_val

    return dp[target]

target =int(input())

arr = list(map(int,input().split()))

print(min\_coins(target,arr))

Q. Find the length of longest subsequence from array which is strictly increasing.

[Question](https://cses.fi/problemset/task/1145)

*#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))

Q. [Question](https://atcoder.jp/contests/dp/tasks/dp_k)

*#K-stones*

N,K = map(int,input().split())

a=list(map(int,input().split()))

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

for i in range(1,K+1):

    for j in range(N):

        if i-a[j]>=0 and dp[i-a[j]]==0:

            dp[i]=1

if dp[K]==1:

    print('First')

else:

    print('Second')

# Q. Find the length of longest common subsequence of 2 string.

*#Longest common subsequence*

def longest\_common\_subsequence(s1,s2):

    n,m = len(s1),len(s2)

    dp = [[0 for i in range(m+1)] for j in range(n+1)]

    for i in range(n):

        for j in range(m):

            if s1[i] == s2[j]:

                dp[i+1][j+1] = dp[i][j]+1

            else:

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

    return dp[n][m]

print(longest\_common\_subsequence("axby","abcy"))

## Q. [k-Ordered LCS](https://www.hackerearth.com/problem/algorithm/mancunian-and-k-ordered-lcs-e6a4b8c6/)