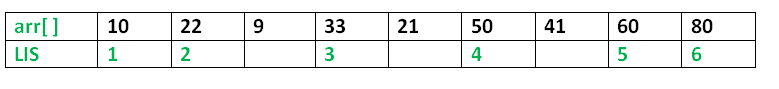
# 1) Longest Increasing Subsequence: - Let us discuss Longest Increasing Subsequence (LIS) problem as an example problem that can be solved using Dynamic Programming.

# The Longest Increasing Subsequence (LIS) problem is to find the length of the longest subsequence of a given sequence such that all elements of the subsequence are sorted in increasing order. For example, the length of LIS for {10, 22, 9, 33, 21, 50, 41, 60, 80} is 6 and LIS is {10, 22, 33, 50, 60, 80}.



# More Examples:-

# Input : arr[] = {3, 10, 2, 1, 20}

# Output : Length of LIS = 3

# The longest increasing subsequence is 3, 10, 20

# Input : arr[] = {3, 2}

# Output : Length of LIS = 1

# The longest increasing subsequences are {3} and {2}

# Input : arr[] = {50, 3, 10, 7, 40, 80}

# Output : Length of LIS = 4

# The longest increasing subsequence is {3, 7, 40, 80}

**2) Longest Common Subsequence :-** *LCS Problem Statement:* Given two sequences, find the length of longest subsequence present in both of them. A subsequence is a sequence that appears in the same relative order, but not necessarily contiguous. For example, “abc”, “abg”, “bdf”, “aeg”, ‘”acefg”, .. etc are subsequences of “abcdefg”. So a string of length n has 2^n different possible subsequences.

It is a classic computer science problem, the basis of [diff](http://en.wikipedia.org/wiki/Diff)(a file comparison program that outputs the differences between two files), and has applications in bioinformatics.

**Examples:-** LCS for input Sequences “ABCDGH” and “AEDFHR” is “ADH” of length 3.  
LCS for input Sequences “AGGTAB” and “GXTXAYB” is “GTAB” of length 4.

The naive solution for this problem is to generate all subsequences of both given sequences and find the longest matching subsequence. This solution is exponential in term of time complexity. Let us see how this problem possesses both important properties of a Dynamic Programming (DP) Problem.

**3) Count number of ways to cover a distance:-** Given a distance ‘dist, count total number of ways to cover the distance with 1, 2 and 3 steps.

Input: n = 3 Output: 4

Below are the four ways :-

1 step + 1 step + 1 step

1 step + 2 step

2 step + 1 step

3 step

Input: n = 4

Output: 7

**4) Find the longest path in a matrix with given constraints: -** Given a n\*n matrix where all numbers are distinct, find the maximum length path (starting from any cell) such that all cells along the path are in increasing order with a difference of 1.

We can move in 4 directions from a given cell (i, j), i.e., we can move to (i+1, j) or (i, j+1) or (i-1, j) or (i, j-1) with the condition that the adjacent cells have a difference of 1. Example:

Input: mat[][] = {{1, 2, 9}

{5, 3, 8}

{4, 6, 7}}

Output: 4 The longest path is 6-7-8-9.

**5) Count number of ways to reach destination in a Maze:-** Given a maze with obstacles, count number of paths to reach rightmost-bottommost cell from topmost-leftmost cell. A cell in given maze has value -1 if it is a blockage or dead end, else 0. From a given cell, we are allowed to move to cells (i+1, j) and (i, j+1) only.

Input: maze[R][C] = {{0, 0, 0, 0},

{0, -1, 0, 0},

{-1, 0, 0, 0},

{0, 0, 0, 0}};

Output: 4

There are four possible paths as shown in below diagram.

# blockage

**6) Edit Distance:-** Given two strings str1 and str2 and below operations that can performed on str1. Find minimum number of edits (operations) required to convert ‘str1’ into ‘str2’ .

# Insert

# Remove

# Replace

# All of the above operations are of equal cost.

# Input: str1 = "geek", str2 = "gesek"

# Output: 1

# We can convert str1 into str2 by inserting a 's'.

# Input: str1 = "cat", str2 = "cut"

# Output: 1

# We can convert str1 into str2 by replacing 'a' with 'u'.

# Input: str1 = "sunday", str2 = "saturday"

# Output: 3

# Last three and first characters are same. We basically need to convert "un" to "atur". This can be done using below three operations.

# Replace 'n' with 'r', insert t, insert a

7) Count all possible paths from top left to bottom right f mXn matrix:-

Happy Number :-

8) Longest Palindrome :-

9) Maximum Sum increasing subsequence:-

10) Count number of binary without consecutives

11) count the number of tree preorder

12) Optimal Tree search

13) Min Jump to reach End

14) Subset sum

15) Cutting Rob.

16) N -pots each with some number of god coins are arranged in a line you are playing a game against another player you tak

17) Given set of jobs with start and interval and profit, how to maximize profit such that jobs in subset do not overlap.

18) Given a total and coins of certain denomination with infinite supply, what is the minimum number of coins it takes to form this total.

19) Given a total an coins of certain denominations find number of ways total can be formed from coins assuming infinity of coins.

20) Longest Bitonic Subsequence

21) Longest common Substring

22) Maximum size square sub-matrix with all 1s.

23) Given two string how many minimum edits(update, delete or add) is needed to convert one string to another

24)Given a cost matrix cost[][] and a position(m,n) in cost[][], write a function that returns cost of minimum cost path to reach(m,n) from (0,0).

25) Total number of possible Binary search trees with n keys.

**) Find minimum number of coins that make a given value :-** Given a value V, if we want to make change for V cents, and we have infinite supply of each of C = { C1, C2, .. , Cm} valued coins, what is the minimum number of coins to make the change?

**Examples:-** Input: coins[] = {25, 10, 5}, V = 30

Output: Minimum 2 coins required

We can use one coin of 25 cents and one of 5 cents

Input: coins[] = {9, 6, 5, 1}, V = 11

Output: Minimum 2 coins required

We can use one coin of 6 cents and 1 coin of 5 cents

This problem is a variation of the problem discussed [Coin Change Problem](http://www.geeksforgeeks.org/dynamic-programming-set-7-coin-change/). Here instead of finding total number of possible solutions, we need to find the solution with minimum number of coins.

The minimum number of coins for a value V can be computed using below recursive formula.

If V == 0, then 0 coins required.

If V > 0

minCoin(coins[0..m-1], V) = min {1 + minCoins(V-coin[i])}

where i varies from 0 to m-1

and coin[i] <= V

**Coin Change: -** Given a value N, if we want to make change for N cents, and we have infinite supply of each of S = { S1, S2, .. , Sm} valued coins, how many ways can we make the change? The order of coins doesn’t matter.

For example, for N = 4 and S = {1,2,3}, there are four solutions: {1,1,1,1},{1,1,2},{2,2},{1,3}. So output should be 4. For N = 10 and S = {2, 5, 3, 6}, there are five solutions: {2,2,2,2,2}, {2,2,3,3}, {2,2,6}, {2,3,5} and {5,5}. So the output should be 5.

# Subset Sum Problem:- Given a set of non-negative integers, and a value sum, determine if there is a subset of the given set with sum equal to given sum.

# Examples: set[] = {3, 34, 4, 12, 5, 2}, sum = 9

# Output: True //There is a subset (4, 5) with sum 9.

# Let isSubSetSum(int set[], int n, int sum) be the function to find whether there is a subset of set[] with sum equal to *sum*. n is the number of elements in set[].

# The isSubsetSum problem can be divided into two subproblems …a) Include the last element, recur for n = n-1, sum = sum – set[n-1] …b) Exclude the last element, recur for n = n-1. If any of the above the above subproblems return true, then return true.

# Following is the recursive formula for isSubsetSum() problem.

# isSubsetSum(set, n, sum) = isSubsetSum(set, n-1, sum) ||

# isSubsetSum(set, n-1, sum-set[n-1])

**0-1 Knapsack Problem :-** Given weights and values of n items, put these items in a knapsack of capacity W to get the maximum total value in the knapsack. In other words, given two integer arrays val[0..n-1] and wt[0..n-1] which represent values and weights associated with n items respectively. Also given an integer W which represents knapsack capacity, find out the maximum value subset of val[] such that sum of the weights of this subset is smaller than or equal to W. You cannot break an item, either pick the complete item, or don’t pick it (0-1 property).

**Count ways to reach the nth stairs.**