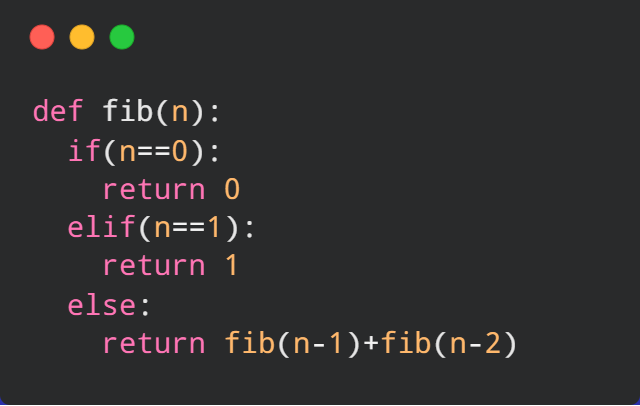
**19I510 Design and Analysis of Algorithms**

**Exercise 7 – Dynamic Programming**

1. Consider the recursive version of Fibonacci given below. Its time complexity is NP. Modify the recursive algorithm so that it takes polynomial time.



**Input Format**The first line of input consists of an integer(N) for which Fibonacci to be computed where N can take values from 0 to 70   
**Output Format**The output will print the nth Fibonacci number

**Sample Input**

**12**

**Sample Output**

144

1. Given a string, print the longest repeating subsequence such that the two subsequences don’t have same string character at same position, i.e., any i’th character in the two subsequences shouldn’t have the same index in the original string.

**Input: str = "aabb"**

**Output: "ab"**

**Input: str = "aab"**

**Output: "a"**

**The two subsequence are 'a'(first) and 'a'**

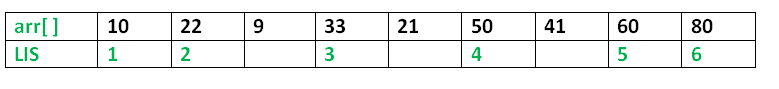
**(second). Note that 'b' cannot be considered**

**as part of subsequence as it would be at same**

**index in both.**

This problem is just the modification of Longest Common Subsequence problem. The idea is to find the LCS(str, str) where str is the input string with the restriction that when both the characters are same, they shouldn’t be on the same index in the two strings.

1. **Implementation of Longest Increasing Subsequence**

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}.  
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}

1. Implementation of 0-1 Knapsack Problem

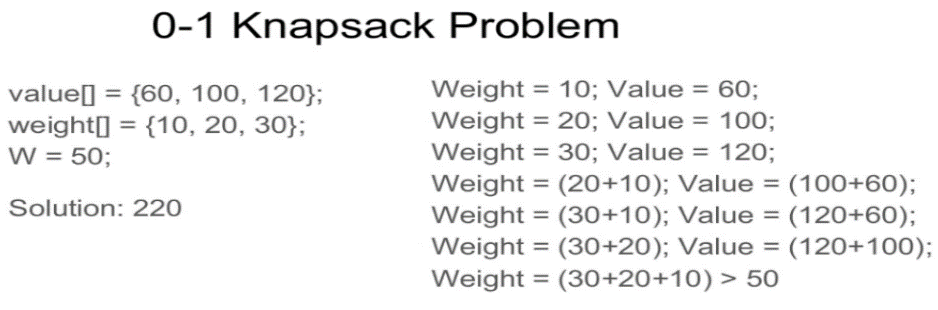
A Naive recursive implementation of 0-1 Knapsack problem – Time complexity –2n

Dynamic approach

/ Returns the maximum value that can be put in a knapsack of capacity W

Time complexity : O(nW) where n is the number of items and W is the capacity of knapsack.

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).



1. Implementation of Longest Common Subsequence

Problem: 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.

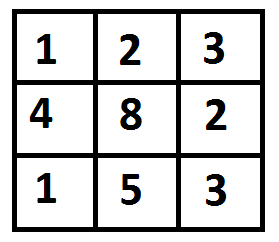
For example, “abc”, “abg”, “bdf”, “aeg”, ‘”acefg”, .. etc are subsequences of “abcdefg”. So a string of length n has 2^n different possible subsequences.

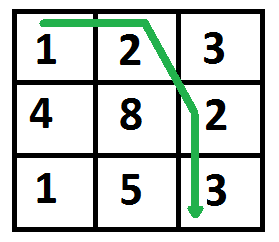
**Examples:**  
LCS for input Sequences “ABCDGH” and “AEDFHR” is “ADH” of length 3.  
LCS for input Sequences “AGGTAB” and “GXTXAYB” is “GTAB” of length 4.

1. Implementation of Min Cost Path

Problem: 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).

Each cell of the matrix represents a cost to traverse through that cell. Total cost of a path to reach (m, n) is sum of all the costs on that path (including both source and destination). You can only traverse down, right and diagonally lower cells from a given cell, i.e., from a given cell (i, j), cells (i+1, j), (i, j+1) and (i+1, j+1) can be traversed. You may assume that all costs are positive integers.

For example, in the following figure, what is the minimum cost path to (2, 2)?  
[](http://www.geeksforgeeks.org/wp-content/uploads/dp.png)

The path with minimum cost is highlighted in the following figure. The path is (0, 0) –> (0, 1) –> (1, 2) –> (2, 2). The cost of the path is 8 (1 + 2 + 2 + 3).  
 [](http://www.geeksforgeeks.org/wp-content/uploads/dp2.png)

**Appoaches**

Naïve solution which is recursive and takes time complexity in exponential.