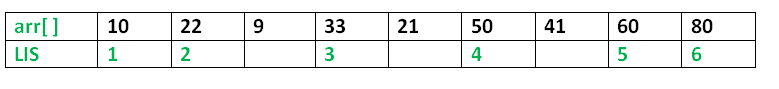
**Longest Increasing Subsequence:**

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}

**Recursion Method for finding LIS:**

***Optimal Substructure:*** Let arr[0..n-1] be the input array and L(i) be the length of the LIS ending at index i such that arr[i] is the last element of the LIS.

Then, L(i) can be recursively written as:

L(i) = 1 + max( L(j) ) where 0 < j < i and arr[j] < arr[i]; or

L(i) = 1, if no such j exists.

To find the LIS for a given array, we need to return max(L(i)) where 0 < i < n.  
Formally, the length of the longest increasing subsequence ending at index i, will be 1 greater than the maximum of lengths of all longest increasing subsequences ending at indices before i, where arr[j] < arr[i] (j < i).  
Thus, we see the LIS problem satisfies the optimal substructure property as the main problem can be solved using solutions to sub problems.

The recursive tree given below will make the approach clearer:

Input : arr[] = {3, 10, 2, 11}

**f(i): Denotes LIS of subarray ending at index 'i'**

(LIS(1)=1)

f(4) {f(4) = 1 + max(f(1), f(2), f(3))}

/ | \

f(1) f(2) f(3) {f(3) = 1, f(2) and f(1) are > f(3)}

| | \

f(1) f(2) f(1) {f(2) = 1 + max(f(1)}

|

f(1) {f(1) = 1}