**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}.

Ex: **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}

***Method 1***

***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.

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.

**Complexity Analysis:**

* **Time Complexity:** The time complexity of this recursive approach is exponential as there is a case of overlapping sub problems as explained in the recursive tree diagram above.
* **Auxiliary Space:** O(1). No external space used for storing values apart from the internal stack space.

***Method 2***

Dynamic Programming:  
We can see that there are many sub problems in the above recursive solution which are solved again and again. So this problem has Overlapping Substructure property and precomputation of same sub problems can be avoided by either using Memorization or Tabulation.