**Longest Increasing Subsequence**

Problem

You are given an array a[] of size n. Find the length of longest increasing subsequences.

Subarray: Continuous block of elements

Subsequences: Part of the array in order. It may or may not be continuous.

Every subarray is a subsequence but every subsequence is not a subarray.

Example



*LIS can be {1,4,5} or {1,2,3}*

*Therefore length of LIS =* ***3***

We define

LIS(i) : Length of longest increasing subsequence ending at ith element.

Therefore, LIS(i) depends on LIS(k), where 0 <= k< i as

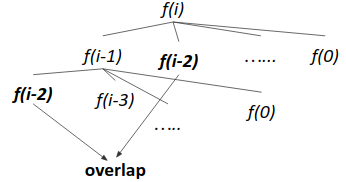
*If (a[i] > a[k])*

*then LIS(i) = max( LIS(i), 1+LIS(k) )*

Since we can write recurrence relation, hence it has Optimal Substructure Property.

Checking whether it has overlapping subproblem property also..

Making recursion tree



Since f(i-2) repeats, it follows overlapping subproblem property.

Since it follows both optimal substructure property and overlapping subproblem property , hence we can apply **dynamic programming** here.

Approach (Tabulation)

1. Make a dp array and initialize all the dp[i] by 1 {since single element is also an LIS}.
2. For every i from left to right, iterate from j=0 to j=i-1 simultaneously checking

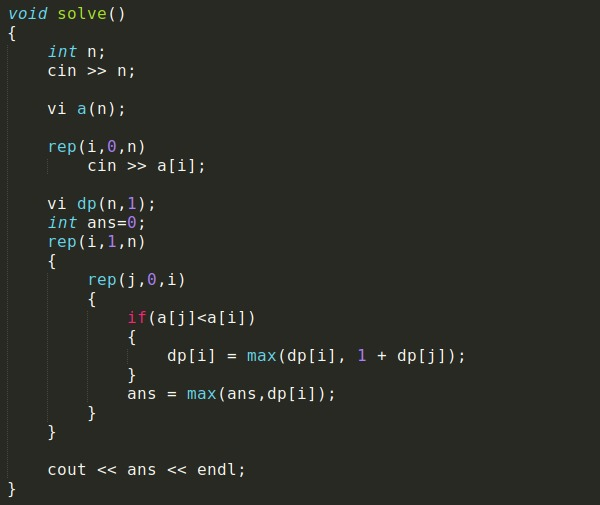
*if(a[i] > a[j])*

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

1. After loop ends, output dp[n-1]

Time Complexity: O(n2)

Code (Iterative)



Code (Recursive)

