In [ ]:
```
# LCS: Longest common sub sequence
# LIS: Longest increasing sub sequence
```

In [ ]:
```
a = 1,3,5,2,9

s = 1,9,5

s = 1,5,9

"abcd"
"acd"

SubStr: Continuous
Sub Sequence:
Subset:
```

Longest common sub sequence

https://leetcode.com/problems/longest-common-subsequence/ (https://leetcode.com/problems/longest-common-subsequence/)

```
    public int rec(String t1, String t2, int i , int , j ){
            if(i == t1.length() || j == t2.length()) return 0;

            if(t1.charAt(i) == t2.charAt(j)) return 1+rec(t1,t2,i+1,j+1);

            return Math.max(rec(t1,t2,i+1,j),rec(t1,t2,i,j+1));
        }



    int lcs(String text1, String text2, int n, int m) {
            if (n == text1.length() || m == text2.length()return 0;

            if (text1.charAt(n - 1) == text2.charAt(m - 1))
                return 1 + lcs(text1, text2, n - 1, m - 1);

            return Math.max(lcs(text1, text2, n - 1, m), lcs(text1, text
    2, n, m - 1));
         }
```

```
class Solution {
public:
    int longestCommonSubsequence(string text1, string text2) {

        return longestCommonSubsequenceUtil(text1,text2,text1.size()-
1,text2.size()-1);
    }
    int longestCommonSubsequenceUtil(string text1, string text2,int
i,int j)
    {
        if(i<0||j<0) return 0;

        if(text1[i]==text2[j])
         return 1+ longestCommonSubsequenceUtil(text1,text2,i-1,j-1);

        return max(longestCommonSubsequenceUtil(text1,text2,i-1,j),lo
ngestCommonSubsequenceUtil(text1,text2,i,j-1));
    }
};
```

TC: O(2^n)

In [ ]:

Memoization

```
int lcsDP(String text1, String text2, int n, int m, int[][] dp) {

    if (n == text1.length() || m == text2.length())
        return 0;

    if (dp[n][m] != -1)
        return dp[n][m];

    int lcs = 0;
    if (text1.charAt(n - 1) == text2.charAt(m - 1)) {
        lcs = 1 + lcsDP(text1, text2, n - 1, m - 1, dp);
    } else {
        lcs = Math.max(lcsDP(text1, text2, n - 1, m, dp), lcsDP(text
1, text2, n, m - 1, dp));
    }

    return dp[n][m] = lcs;
}
```

TC: O(n*m)
SC: O(n*m)

```
n   1 2 3 4   5   10     100
n   1 2 3 4   5   10     100
n^2 1 4 9 16  25  100    10000
2^n 2 4 8 16  32  1024   126765060022822940149670320537 6
```

```
class Solution {
public:
    int longestCommonSubsequence(string text1, string text2) {
        vector<vector<int>> dp(text1.size(),vector<int>(text2.size(),
-1));
        return longestCommonSubsequenceUtil(text1,text2,text1.size()-
1,text2.size()-1,dp);
    }

    int longestCommonSubsequenceUtil(string text1, string text2,int
i,int j,vector<vector<int>>& dp)
    {
        if(i<0||j<0) return 0;

        if(dp[i][j]!=-1) return dp[i][j];

        if(text1[i]==text2[j])
         return dp[i][j] = 1+ longestCommonSubsequenceUtil(text1,text
2,i-1,j-1,dp);

        return dp[i][j] = max(longestCommonSubsequenceUtil(text1,text
2,i-1,j,dp),longestCommonSubsequenceUtil(text1,text2,i,j-1,dp));
    }
};
```

In [ ]:

Tabulation

```
int longestCommonSubsequence(string text1, string text2) {
        int m = text1.length();
        int n = text2.length();
        vector<vector<int>> dp(m+1, vector<int>(n+1));

        for(int i=0;i<=n;i++){
            dp[0][i]=0;
        }
        for(int i=0;i<=m;i++){
            dp[i][0]=0;
        }

        for (int i=1; i<=m; i++) {
            for (int j=1; j<=n; j++) {
                if (text1[i-1] == text2[j-1]) {
                    dp[i][j] = 1 + dp[i-1][j-1];
                }
                else {
                    dp[i][j] = max(dp[i][j-1], dp[i-1][j]);
                }
            }
        }

        return dp[m][n];
    }
```

TC: O(m*n)
SC: O(m*n)

```
class Solution {
public:
    int longestCommonSubsequence(string text1, string text2) {
        vector<vector<int>> dp(text1.size()+1,vector<int>(text2.size
()+1,0));

        for(int i=1;i<=text1.size();i++)
        {
            for(int j=1;j<=text2.size();j++)
            {
                if(text1[i-1]==text2[j-1])
                    dp[i][j] = 1+ dp[i-1][j-1];
                else
                    dp[i][j] = max(dp[i-1][j],dp[i][j-1]);
            }
        }


        return dp[text1.size()][text2.size()];
    }
};
```

In [ ]:

In [5]:
```
2**100
```

Out[5]: 1267650600228229401496703205376

In [ ]:

In [ ]:

LIS: Longest increasing sub sequence

https://leetcode.com/problems/longest-increasing-subsequence/ (https://leetcode.com/problems/longest-increasing-subsequence/)

Brute Force

```C++
class Solution {
public:
    int lengthOfLIS(vector<int>& nums) {

        return lengthOfLISUtil(nums,0,-1);
    }
```

```
    int lengthOfLISUtil(vector<int>&nums,int curr,int pre)
    {
         if(curr>=nums.size()) return 0;
        int in=0;

        if dp[pre+1][curr+1] is set:
            return
        if(pre==-1|| nums[curr]>nums[pre])
        {
            in=1+lengthOfLISUtil(nums,curr+1,curr);
        }
        int ex=lengthOfLISUtil(nums,curr+1,pre);
        return dp[pre+1][curr+1] = max(in,ex);
    }
};
```
```
TC: O(2^n)
SC: O(N)
```

In [ ]:

Memoization

```
class Solution {
     public int lengthOfLIS(int[] a) {
        int n = a.length;
        int[][] dp=new int[n][n+1];
        for(int[] row: dp) Arrays.fill(row,-1);
        return help(0,-1,a,dp);
    }

    public static int help(int i, int prev, int[] a,int[][] dp){
        if(i == a.length) return 0;

        if(dp[i][prev+1] != -1) return dp[i][prev+1];

        int notTake = help(i+1,prev,a,dp);
        int take = -1;
        if(prev == -1 || a[i] > a[prev]){
            take = 1 + help(i+1,i,a,dp);
        }
        return dp[i][prev+1] = Math.max(take,notTake);
    }
}
```

```
class Solution {
public:
    int lengthOfLIS(vector<int>& nums) {
        vector<vector<int>> dp(nums.size()+1,vector<int>(nums.size()+
1,-1));
        return lengthOfLISUtil(nums,0,-1,dp);
    }

    int lengthOfLISUtil(vector<int>&nums,int curr,int pre,vector<vect
or<int>> &dp)
    {
        if(curr>=nums.size()) return 0;
        int in=0;

        if(  dp[curr][pre+1]!=-1) return   dp[curr][pre+1];

        if(pre==-1|| nums[curr]>nums[pre])
        {
            in=1+lengthOfLISUtil(nums,curr+1,curr,dp);
        }

        int ex=lengthOfLISUtil(nums,curr+1,pre,dp);

        dp[curr][pre+1] = max(in,ex);

        return max(in,ex);
    }
};
```

In [ ]:
```
data          10,9,2,5,3,7,101,18
LIS till Now   1 1 1 1 1 1  1   1
```

https://www.geeksforgeeks.org/dynamic-programming-building-bridges/