Given two strings ‘X’ and ‘Y’, find the length of the longest common substring. For example, if the given strings are “GeeksforGeeks” and “GeeksQuiz”, the output should be 5 as longest common substring is “Geeks”

Let m and n be the lengths of first and second strings respectively.

A **simple solution** is to one by one consider all substrings of first string and for every substring check if it is a substring in second string. Keep track of the maximum length substring. There will be O(m^2) substrings and we can find whether a string is subsring on another string in O(n) time (See this). So overall time complexity of this method would be O(n * m^2)

Dynamic Programming can be used to find the longest common substring in O(m*n) time. The idea is to find length of the longest common suffix for all substrings of both strings and store these lengths in a table.

```
The longest common suffix has following optimal substructure property
   LCSuff(X, Y, m, n) = LCSuff(X, Y, m-1, n-1) + 1 if X[m-1] = Y[n-1]
                        0  Otherwise (if X[m-1] != Y[n-1])

The maximum length Longest Common Suffix is the longest common substring.
   LCSubStr(X, Y, m, n)  = Max(LCSuff(X, Y, i, j)) where 1 <= i <= m
                                                     and 1 <= j <= n
```

Following is C++ implementation of the above solution.

```
/* Dynamic Programming solution to find length of the longest common substring */
#include<iostream>
#include<string.h>
using namespace std;

// A utility function to find maximum of two integers
int max(int a, int b)
{   return (a > b)? a : b; }

/* Returns length of longest common substring of X[0..m-1] and Y[0..n-1] */
int LCSubStr(char *X, char *Y, int m, int n)
{
    // Create a table to store lengths of longest common suffixes of
    // substrings.   Notethat LCSuff[i][j] contains length of longest
    // common suffix of X[0..i-1] and Y[0..j-1]. The first row and
    // first column entries have no logical meaning, they are used only
    // for simplicity of program
    int LCSuff[m+1][n+1];
    int result = 0;  // To store length of the longest common substring

    /* Following steps build LCSuff[m+1][n+1] in bottom up fashion. */
    for (int i=0; i<=m; i++)
    {
        for (int j=0; j<=n; j++)
        {
            if (i == 0 || j == 0)
                LCSuff[i][j] = 0;

            else if (X[i-1] == Y[j-1])
            {
                LCSuff[i][j] = LCSuff[i-1][j-1] + 1;
                result = max(result, LCSuff[i][j]);
            }
            else LCSuff[i][j] = 0;
        }
    }
    return result;
}

/* Driver program to test above function */
```

```
int main()
{
    char X[] = "OldSite:GeeksforGeeks.org";
    char Y[] = "NewSite:GeeksQuiz.com";

    int m = strlen(X);
    int n = strlen(Y);

    cout << "Length of Longest Common Substring is " << LCSubStr(X, Y, m, n);
    return 0;
}
```

Output:

```
Length of Longest Common Substring is 10
```

Time Complexity: O(m*n)
Auxiliary Space: O(m*n)

References: http://en.wikipedia.org/wiki/Longest_common_substring_problem

The longest substring can also be solved in O(n+m) time using Suffix Tree. We will be covering Suffix Tree based solution in a separate post.

Exercise: The above solution prints only length of the longest common substring. Extend the solution to print the substring also.