Give precise arguments. If using dynamic programming, you must clearly state what each entry of the table denotes, and in which order to compute the entries. Also give an explanation of the recurrence used.

A subsequence of a string $x = x_1, x_2, \ldots, x_n$ is a subset of characters $x_{i_1}, x_{i_2}, \ldots, x_{i_k}$ such that $i_1 < i_2 < \ldots < i_k$. Such a subsequence is said to be palindromic if it is the same whether read left to right or right to left. For instance, the string $A, C, G, T, G, T, C, A, A, C, T, C, G$ has many palindromic subsequences, including $A, C, G, C, A$ and $A, A, A, A$ (on the other hand, the subsequence $A, C, T$ is not palindromic). Devise an algorithm that takes a string $x[1 \ldots n]$ and returns the length of the longest palindromic subsequence in it. Its running time should be $O(n^2)$.

**Solution** We build a 2-dimensional table $T$, where the entry $T[i, j]$ stores the longest palindrome in the string $x_i, \ldots, x_j$. Now we write a recursive definition of $T[i, j]$. If symbols $x_i$ and $x_j$ are same, we can assume that they are part of the longest palindrome. So, $T[i, j]$ will be equal to $T[i + 1, j - 1]$. If the two symbols are different, then they both cannot be part of any palindrome. So, $T[i, j]$ will be maximum of $T[i + 1, j]$ and $T[i, j - 1]$. Thus, we get

$$T[i, j] = \begin{cases} 
T[i + 1, j - 1] & \text{if } x_i = x_j \\
\max(T[i + 1, j], T[i, j - 1]) & \text{otherwise}
\end{cases}$$

The base case happens when $i = j$, in which case the answer is 1. The for loops are:

```
for i = 1 to n
    T[i, i] = 1;
for i=n-1 downto 1
    for j = i+1 to n
        T[i, j] = as mentioned above
```

The algorithm returns $T[1, n]$. 

---

**Quiz3A (COL 351)**

Name Entry No.
Quiz3B (COL 351)

Name Entry No.

Give precise arguments. If using dynamic programming, you must clearly state what each entry of the table denotes, and in which order to compute the entries. Also give an explanation of the recurrence used.

Given two strings \( x = x_1, x_2, \ldots, x_n \) and \( y = y_1, y_2, \ldots, y_m \), a string \( z \) of length \( m + n \) is said to be an interleaving of \( x \) and \( y \) if it contains all the characters of \( x \) and \( y \), and the order of these characters in the individual strings is preserved. For example if \( x = AAB \) and \( y = AAC \), then each of the strings AAABAC, AAAACB, AAAABC, AAACAB are interleavings of \( x \) and \( y \), whereas AABACA and ABAAAC are not interleavings of \( x \) and \( y \). Devise an algorithm that takes strings \( x[1 \ldots n], y[1 \ldots m], z[1 \ldots (m+n)] \) and returns true if \( z \) is an interleaving of \( x \) and \( y \); otherwise it returns false. The running time of the algorithm should be \( O(mn) \).

**Solution** We build a 2-dimensional table \( T \), where the entry \( T[i, j] \) stores true if \( z[(i+j-1), \ldots, (m+n)] \) can be written as interleaving of \( x[i, \ldots, n] \) and \( y[j, \ldots, m] \). Now we write a recursive definition of \( T[i, j] \).

If \( z[i+j-1] \) is equal to \( x[i] \) (or \( y[j] \)), then it must match with \( x[i] \) (or \( y[j] \)). Therefore, \( T[i, j] \) is true if one of the following cases happen – (i) \( z[i+j-1] = x[i] \) and \( T[i+1, j] \) is true, or (ii) \( z[i+j-1] = y[j] \) and \( T[i, j+1] \) is true. If neither of these two cases happen, i.e., \( z[i+j-1] \neq x[i], y[j] \), then \( T[i, j] \) must be false. Thus, we get (in boolean terms)

\[
T[i, j] = ((z[i+j-1] = x[i]) \text{ AND } T[i+1, j])) \text{ OR } ((z[i+j-1] = y[j]) \text{ AND } T[i, j+1]))
\]

For base case, notice that it is better to define \( T[i, m + 1] \) as true if \( x[i], \ldots, x[n] \) matches with \( z[i + m], \ldots, z[m + n] \), false otherwise. Similarly for \( T[n + 1, j] \).

Base case as above.

\[
\text{for } i = n \text{ downto 1} \\
\quad \text{for } j = m \text{ downto 1} \\
\quad \quad T[i, j] = \text{as mentioned above}
\]

The algorithm returns \( T[1, 1] \).