Recursive Characterization

- Based on the above recurrence the algorithm for computing LCS is

```
m = length[X]; n = length[Y];
for (i = 1; i ≤ m; i++){
    c[i,0] = 0; c[0,i] = 0;
}
for (i = 1; i ≤ m; i++)
    for (j = 1; i ≤ m; i++)
        if (x_i == y_j) {
            c[i,j] = c[i-1,j-1] + 1;
            b[i,j] =↖;
        } else if (c[i-1,j] ≥ c[i,j-1]) {
            c[i,j] = c[i-1,j];
            b[i,j] =↑;
        } else {
            c[i,j] = c[i,j-1];
            b[i,j] =←;
        }
return c and b;
```


Example

i \ j		0	1	2	3	4	5	6
		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B	0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	C	0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
4	B	0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3
5	D	0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
6	A	0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7	B	0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

Constructing LCS

- Using table b we can find the index sequence for LCS.
- An arrow "$\nwarrow$" in $b[i, j]$ indicates that $x_i = y_j$ is an element of LCS.
- So starting from $b[n, n]$ we can traverse upto slot in b array where first "$\nwarrow$" is found.
- From there hop to slot in b where next "$\nwarrow$" is found.
- Continue till index i becomes 0.

Constructing LCS

```
printLCS(b, X, i, j) {
    if ((i = 0) || (j == 0))
        return 0;
    if (b[i, j] == "↖") {
        printLCS(b, X, i − 1, j − 1);
        print x_i;
    } else if (b[i, j] == "↑")
            printLCS(b, X, i − 1, j);
        else
            printLCS(b, X, i, j − 1);
}
```

Constructing LCS

- Note that we do not require to store b array.
- We can effectively hop from a matching slot to next by using index matching information stored in c.
- The entry $c[i, j]$ actually depends on three entries $c[i-1, j-1]$, $c[i-1, j]$, and $c[i, j-1]$.
- Clearly, there may be a match if $c[i, j] = c[i-1, j-1] + 1$.
- But in that case, $c[i, j-1]$ or $c[i-1, j]$ both cannot be less than $c[i, j]$.

Constructing LCS

- Time to build c and b arrays take O(mn) time.
- Time to print LCS is O($m + n$), since at least one index i or j decremented each time.
- Space saving by eliminating b array is $\Theta(mn)$.
- But the asymptotic space requirement is still $\Theta(mn)$, as we need to store c.

