1. [10 points]

Show the process of constructing a bottom up dynamic programming table for the longest common subsequence of the two strings: **1101** and **1010**. Explain how you find the longest common subsequence from the table. Please show how you develop the table for these two sequences rather than just giving an answer.

First, you want to construct a table of size mn where m is the length of one of the sequences and n is the length of the other. Our sequences are 1101 and 1010, which are both length 4, so we make a 4x4 table with 1101 along the top x-axis and 1010 along the left y-axis (or the other way around). Then we fill out both the first row and first column with zeroes, since at these coordinates one of our strings with be length zero and thus there can be no longest common subsequence. Now we can start filling out our table following this logic:

If the characters from the two sequences at that coordinate are the same: set the value to one plus the value at the coordinate to the top left of it, and draw an arrow pointing top left.

Otherwise: set the value to the maximum of the coordinate above it and the coordinate to the left of it, and draw an arrow pointing toward where the maximum came from.

After following these rules in order to set a value and arrow at every coordinate, all we have to do is retrace our steps starting from the bottom right coordinate and following our arrows. To construct our longest comon subsequence, we will add a character to it only if, at that coordinate, the two sequences had the same character, otherwise we will skip this coordinate and simply follow the arrow to the next. The steps we took are indicated by the circled coordinates, and are as follows:

		J	1	1	0	1
		0	1	2	3	4
i	0	0	0	0	0	0
1	1	0	↖1	↖1	←1	↖1
0	2	0	↑1	↑1	↖2	←2
1	3	0	↖1	↖2	↑2	↖3
0	4	0	↑1	↑2	↖3	↑3

1 0 1

1) At coordinate (4, 0), the bottom right, one sequence had character 0 and the other had 1, so they are not the same and we will simply follow the arrow.

2) At coordinate (4, 1), both sequences have character 1 so they are the same and we will **add 1** to our longest common subsequence and follow the arrow.

3) At coordinate (3, 2), both sequences have character 0 so they are the same and we will **add 0** to our longest common subsequence and follow the arrow.

4) At coordinate (2, 1), both sequences have character 1 so they are the same and we will **add 1** to our longest common subsequence and follow the arrow.

5) At coordinate (1, 0), the value is 0 since we are in the first row, so we are done (no more arrows to follow).

Therefore, the longest common subsequence of the sequences 1101 and 1010 is 101.

2. [10 points]

For the dynamic programming formulations of the Rod cutting problem and Fibonacci, and Longest Common Subsequence: (a) how many subproblems are there (this is the size of the table we saved). Include a sketch of the table with these dimensions.; (b) how many previous entries in the table do we look at for a given subproblem we are computing? (the book refers to this as choices per subproblem); (c) What is the total run time? Explain your answers.

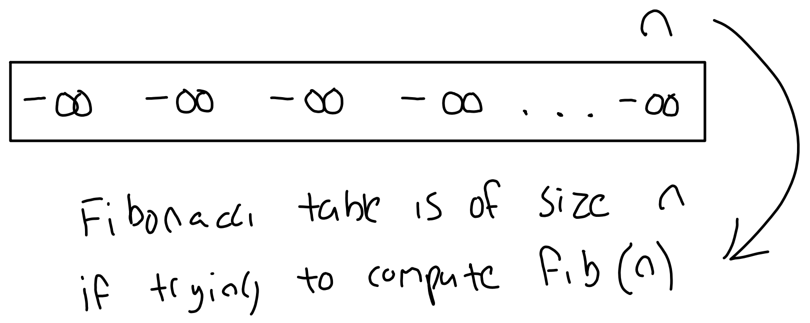
For the **rod cutting problem** we have n subproblems as indicated by the table of size n below, where n is the maximum revenue of a rod of size n. For each subproblem, we also have n choices, which makes our total run time Θ(n^2), and this is clear if you look at the pseudocode for this problem which has a for loop from 1 to n and another for loop from 1 to j inside it which ends up iterating n times when j = n.

	0	1	2	3		n
r_i	$-\infty$	$-\infty$	$-\infty$	$-\infty$	. . .	$-\infty$

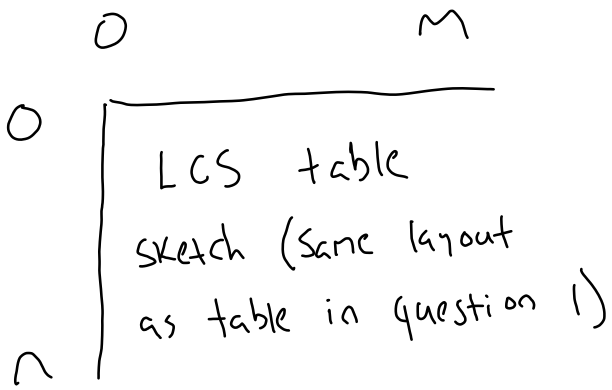
In the rod-cutting table, n is the maximum revenue possible from cutting a rod of size n.

from cutting a rod of size n.

Fibonacci has a table of size n (shown below) where n is the number we would like to compute the fibonacci of, and this means it has n subproblems. However, at each subproblem we are only using 2 previous subproblems to compute our answer since in our algorithm we are using fib(n-1) and fib(n-2) to compute fib(n) recursively. This gives us a run time of Θ(n), since 2 is a constant.


n
$-\infty$ $-\infty$ $-\infty$ $-\infty$ . . . $-\infty$
Fibonacci table is of size n
if trying to compute Fib(n)

Longest common subsequence has mn subproblems, where m is the length of one sequence and n is the length of the other (table below). At each subproblem we have up to 3 choices of previous subproblems since, depending on if the character is the same in both sequences at that given subproblem, we will either add one to the previous subproblem to the top left of it or we will choose the maximum of the subproblems to the left and above it. The diagram to the right of the table shows the three different choices of previous subproblems.



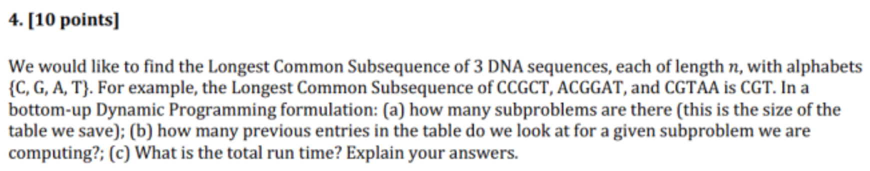
$$c[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0\,, \\ c[i-1,j-1]+1 & \text{if } i,j > 0 \text{ and } x_i = y_j\,, \\ \max(c[i,j-1], c[i-1,j]) & \text{if } i,j > 0 \text{ and } x_i \neq y_j\,. \end{cases}$$

3. [10 points]

Consider a modification of the rod-cutting problem in which, in addition to a price p_i for each rod, each cut incurs a fixed cost of c. The revenue associated with a solution is now the sum of the prices of the pieces minus the costs of making the cuts. Give a dynamic-programming algorithm to solve this modified problem. Explain your answer.

```
memoized_cut_rod_aux(p, n, r, c)
    if r[n] >= 0
        return r[n]
    if n == 0
        q = 0
    else q = -inf
        for i = 1 to n
            q = max(q, p[i] + memoized_cut_rod_aux(p, n - i, r, c) - c)
    r[n] = q
    return q
```

In our memoized rod cutting auxiliary function, we will pass a new parameter c to represent the fixed cost of making a cut. Then, in our call to max we will subtract c from p[i], the price, plus our recursive call to memoized_cut_rod_aux. We never modify the value of c, instead passing it on in our recursive call so that each time the revenue is calculated we ensure that the fixed cost c is subtracted from it.



4. [10 points]

We would like to find the Longest Common Subsequence of 3 DNA sequences, each of length n, with alphabets {C, G, A, T}. For example, the Longest Common Subsequence of CCGCT, ACGGAT, and CGTAA is CGT. In a bottom-up Dynamic Programming formulation: (a) how many subproblems are there (this is the size of the table we save); (b) how many previous entries in the table do we look at for a given subproblem we are computing?; (c) What is the total run time? Explain your answers.

We would have a three dimensional table of size n^3, since we now have 3 sequences all of length n. Therefore, we would have n^3 subproblems. For any given subproblem, we would have 4 choices of previous subproblems. This is because the case for all three sequences having the same character would still be one subproblem, but instead of c[j-1, i-1] + 1 it would simply be c[j-1, i-1, k-1] + 1 since we now have 3 coordinates. So for this case, the number of choices of previous subproblems doesn't change compared to LCS of 2 sequences.

However, if it is the case that the three sequences don't have the same character at a given subproblem, we would have to take the maximum of three different subproblems instead of two. Visually, we will be taking the maximum value of the subproblems to the left, above, and behind (negative z direction) our current subproblem. In our pseudocode it would look like this: max(c[i, j-1, k], c[i-1, j, k], c[i, j, k-1]). You can see that we have three subproblems in this call to max, and also a single subproblem in the case that all three sequences have the same character, so we have a total of 4 choices of previous subproblems.

The total run time would be Θ(n^3), since instead of two strings of sizes m and n we have three strings all of size n (n*n*n = n^3).