CS 161 **Pre-Lecture Exercise for Lecture 13**
Fall 2025 **Date:** November 13, 2025

Pre-lecture exercises will not be collected for credit. However, you will get more out of each lecture if you do them, and they will be referenced during lecture. We recommend **writing out** your answers to pre-lecture exercises before class. Pre-lecture exercises usually should not take you more than 30 minutes.

Suppose we are given two strings, for example:

$$x = ABCDEFGH$$
$$y = ABXFDEGCT$$

Consider the following definitions:

- A *subsequence* of a string x is a string made up of some of the letters of x, in the same order that they appear x (they need not be contiguous). For example, $ABDH$ and CDF are both subsequences of the string x above.
- A *common subsequence* between two strings x and y is a subsequence that both x and y have in common. For example, ABF and $ABDEG$ are both common subsequences of the strings x and y above.
- A *longest common subsequence* (LCS) is a common subsequence that is the longest. In this example, it is $ABDEG$, and it has length 5.

Suppose we are given x and y, and we want to design a dynamic programming algorithm to find the *length* of an LCS. So in the example above, we should return "5." Recall from last class that the first step of coming up with a DP algorithm is to identify our sub-problems. We want these sub-problems to have the following properties:

1. An optimal solution to the big problem can be built from optimal solutions to the sub-problems.
2. The sub-problems overlap a lot. That is, you can use the same sub-problem again and again when building up a solution to the big problem.

Exercise: Think of some potential sub-problems to use for designing this DP algorithm. (You don't need to actually design the DP algorithm, we'll do that in class; right now we just want to come up with some candidate sub-problems to get us started).

Note: This should feel a bit like coming up with sub-problems in a recursive algorithm; the difference is that unlike in, say, MergeSort, where each sub-problem involves a different set of elements and is only used once to build our final solution, here we expect our sub-problems to be used many times over.