Problem 3-1. The Longest Common Subsequence Problem [35 points]

Suppose we are given two strings, and we want to compare how “similar” these two strings are, by comparing how much common subsequence they share. This problem, called the longest-common-subsequence problem, comes up in biological applications, such as when we need to compare the DNA of two (or more) different organisms.

Given two sequences $X$ and $Y$, we say that a sequence $Z$ is a common subsequence of $X$ and $Y$ if $Z$ is a subsequence of both $X$ and $Y$. For instance, if $X = \langle B, C, G, A, E, O, F, E \rangle$ and $Y = \langle G, B, K, A, E, C, F, N \rangle$, the sequence $\langle B, A, F \rangle$ is a common subsequence of both $X$ and $Y$, and has length 3. It is not the longest common subsequence, however, because the subsequence $\langle B, A, E, F \rangle$ is also a common subsequence and has length 4. The sequence $\langle B, A, E, F \rangle$ is an longest common subsequence (LCS) of $X$ and $Y$, because there is no common subsequence of length greater than 4. Thus, the longest-common-subsequence (LCS) problem is defined as follows.

More formally, the longest-common-subsequence (LCS) problem is defined as follows.

**Definition 1 (The Longest Common Subsequence Problem)** Given two sequences $X = \langle x_1, x_2, \ldots, x_n \rangle$ and $Y = \langle y_1, y_2, \ldots, y_m \rangle$, find a maximum-length common subsequence of $X$ and $Y$.

In this problem, you will design dynamic programming solutions to the LCS problem using different amount of space.

(a) **Computing the LCS** [10 points]

Write down the pseudocode for computing LCS using a doubly nested loops. This exercise will help you figure out the right recurrence to use to compute LCS.

(b) **Solving LCS using quadratic amount of space** [10 points]

Now write a divide and conquer solution to this problem and parallelize it. You should provide the full pseudocode and analyze the work and span. Make sure that your pseudocode allows you to calculate the actual LCS and not just the length — that is, you should maintain backward pointers so that you can backtrack to give the actual LCS.
(c) Solving LCS using linear amount of space [15 points]
Now implement a divide and conquer solution, but using only a linear amount of
space; here you need not return the actual LCS, just the length of the LCS. You should
think about the dependencies of filling the table to solve LCS, and figure out how you
can keep the information you need using linear amount of space. You should analyze
the work and span of your linear space solution as well.

Problem 3-2. Text Parsing [20 points]
Suppose you are given a sequence of characters \( S = \langle c_0, c_1, c_2, \ldots, c_{n-1} \rangle \) that has no spaces or
punctuation. You want to determine if you can parse the string into a sequence of words, given
a dictionary of valid words. For example, “summerisalmosthere” can be broken into “summer is
almost here”. If the sequence has \( n \) characters, then are \( n - 1 \) places in which to put a space or not
and so there are \( 2^{n-1} \) ways to insert spaces. You are to develop a dynamic programming algorithm
to break up the sequence into words. You may assume that words are bounded by length \( k \) and that
you can determine if a word \( w \) is in a dictionary in \( O(|w|) \) work and span.

(a) [5 points] A greedy algorithm would find a word at the beginning of the sequence,
then repeat on the rest of the sequence. Give an example where using a greedy method
would not find a valid segmentation of the sequence when one exists.

(b) [15 points] Give a dynamic programming solution to determine if a sequence of char-
acters can be broken into words based on a dictionary \( D \). Be sure to include all the
base cases. Assume a word is itself a sequence of characters, and that for a word \( w \),
\( D[w] \) returns \text{true} if \( w \) is in the dictionary and \text{false} otherwise. Also assume that
the maximum word length is \( k \).

Problem 3-3. Probability Practice [25 points]
This problem aims at helping you brush up on probability, gearing up for randomized algorithms
that we’ll start covering. Let \( k \) and \( r \) be positive integers. For \( i = 1, 2, \ldots, k \), let \( X_i \) be integer
drawn uniformly at random from the interval \( [0, r] \). We’ll assume that the \( X_i \)’s are independent.

(a) [8 points] What is the expected sum of the values (in terms of \( k \) and \( r \))? That is,
compute \( E \left[ \sum_{i=1}^{k} X_i \right] \).

(b) [8 points] Calculate \( E \left[ \max(X_1, X_2) \right] \) in terms of \( r \).

(c) [9 points] If \( k \leq r \), what is the probability that they are all distinct?

Problem 3-4. Simple in Series, Painful in Parallel [20 points] (2 parts)
You are given an array of non-negative integers \( A[1..n] \) and another integer \( X \). For each index \( i \),
you want to find index \( j \), such that \( \sum_{k=i}^{j} A[k] \leq X \), but \( \sum_{k=i}^{j+1} A[k] > X \). That is, you want to
compute an array $C$ such that $C[i] = j$ is the largest $j$ such that the sum of all the elements from $A[i]$ to $A[j]$ is at most $X$.

(a) [10 points] Provide a sequential algorithm for this problem with work $\Theta(n)$.

(b) [10 points] Provide a parallel algorithm with work $\Theta(n \lg n)$ and span $\Theta(\lg n)$.

(c) Optional Give a work-efficient parallel algorithm to solve this problem.