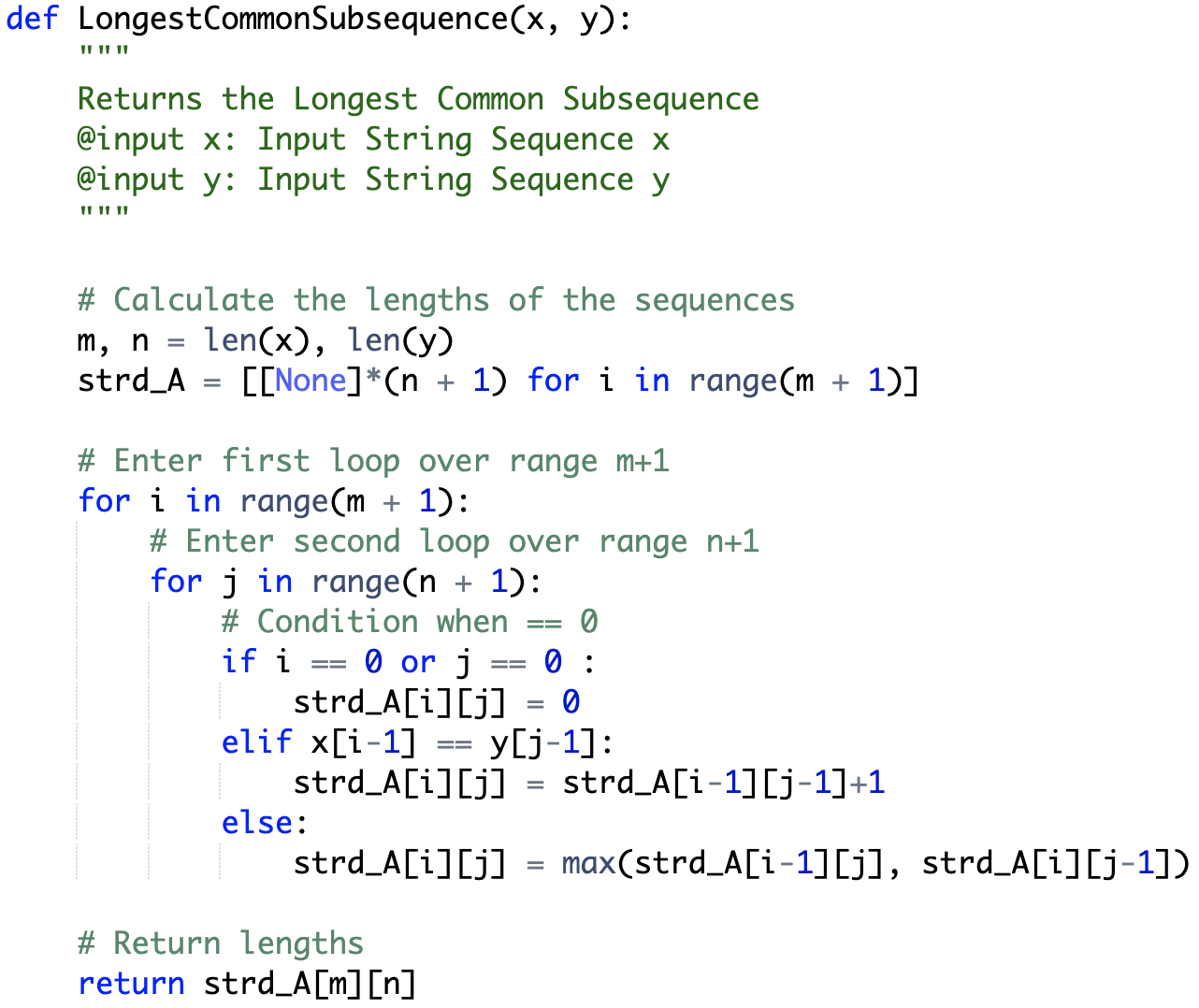
**Problem 1. Longest common subsequence problem (10 pts)**

Given two strings x1...xn, y1...ym wish to find the length of their longest common subsequence, that is, the largest k for which there exist indices i1<...<ik and j1<=<jk such that xi1...xik... = yj1 = yjk. Show how to do this in time O(nm).

* The longest common subsequence is defined as the longest subsequence found across two or more sequences, regardless of position within the individual sequences
* One standard method of tackling such an item within the confines of dynamic programming is as follows:
  + We can keep a record of the longest common suffixes by storing them
  + As we store these, we maintain that the longest common suffix for and
  + We can begin by iterating over the sequence in the form of a loop over the length of m, building them in a bottom-up approach
  + We then enter a second loop over the sequence over the length of n
  + When the condition of i and j is met, we get the longest common suffix
* Since we iterate over m and n respectively, the algorithm gives us a time complexity of O(nm)



* Using this method, if we use the input values of “AGTCCGCGAA”, and “GAA”, we will see that GAA is the longest common value with a length of 3.

**Problem 2. Longest subsequence problem with linear space complexity (10 pts)**

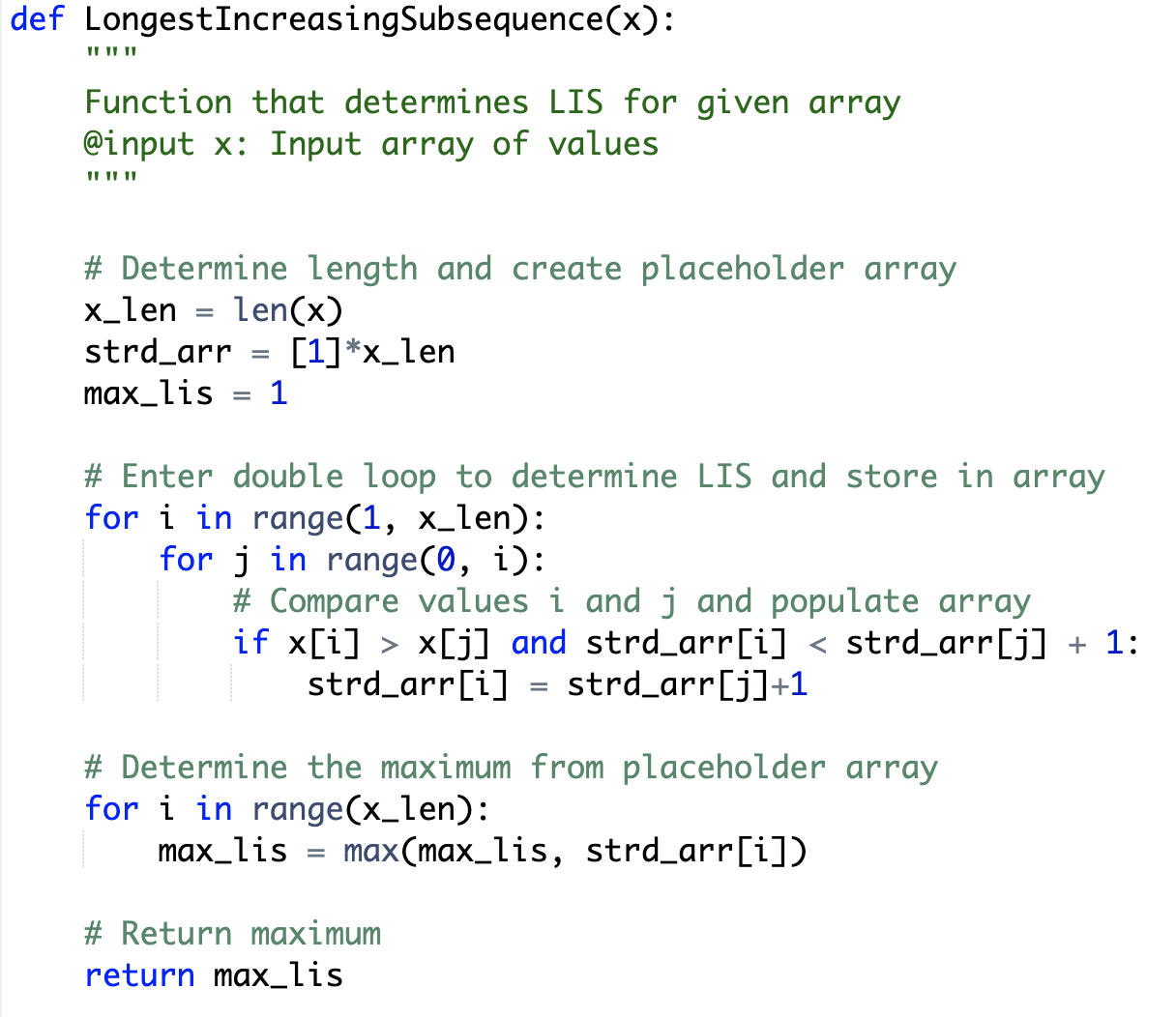
Find a space complexity of your solution for Problem 1.

* In the algorithm above, we use the variable strd\_A to store m and n, we can determine that the space complexity would be O(n\*m), similar to the time complexity.

**Problem 3. Longest increasing subsequence problem (10 pts)**

Given a string x1 ... xn we wish to find the length of its longest increasing subsequence, that is, the largest k for which there exist indices i1 < ... < ik such that xi1 < xi2 < ... < xik Show how to do this in time O(n2).

* We can define the longest increasing subsequence as the length of a sequence by which the values remain to be increasing within a given input list. For example, in the list [1, 2, 4, 3, 7, 9], we get an LIS of length 5 from [1, 2, 4, 7, 9].
* We can use a dynamic programming approach here by storing a list for LIS as part of the process
* Algorithm:
  + We begin by determining the length of the given input array
  + Next, we create a list to store the values
  + We then create a variable to remember the maximum value visited thus far
  + Next, we iterate over the array in the form of loop, and then iterate over the individual element
  + We check to see if the value of the first loop is greater than the second, thereby checking to determine if its incrementing, essentially seeing of
  + If the condition is met, we store that value in our previously created array
  + At the end of both loops, we can check the maximum value that was stored in the process in order to return that length as part of the question.
  + Since two nested loops are involved in the process, we will see a time complexity of O(n2)



* If we add an input value of [1, 2, 3, 2, 5, 6, 1, 7], we will see the LIS have a length of 6 represented by the array of [1,2,3,5,6,7]
* Since we used a nested loop here, we will see the time complexity at O(n2)

Resources:

[1] https://northeastern.instructure.com/courses/117409/pages/module-9

[2] https://en.wikipedia.org/wiki/Longest\_increasing\_subsequence

[3] Introduction to Algorithms, Cormen, Third Edition. (CLRS)

[4] https://en.wikipedia.org/wiki/Longest\_common\_subsequence\_problem