Problem 1. Longest common subsequence problem (10 pts)

Given two strings $x_1 \ldots x_n$, $y_1 \ldots y_m$ wish to find the length of their longest common subsequence, that is, the largest k for which there exist indices $i_1 < \ldots < i_k$ and $j_1 <= < j_k$ such that $x_{i1} \ldots x_{ik} \ldots = y_{j1} = y_{jk}$. Show how to do this in time *O(nm)*.

```java
class P1 {

  public int longestCommonSubsequence(String text1, String text2) {
    // If text1 doesn't reference the shortest string, swap them.
    if (text2.length() < text1.length()) {
      String temp = text1;
      text1 = text2;
      text2 = temp;
    }
    // The previous and current column starts with all 0's and like
    // before is 1 more than the length of the first word.
    int[] previous = new int[text1.length() + 1];
    int[] current = new int[text1.length() + 1];
    // Iterate through each column, starting from the last one.
    for (int col = text2.length() - 1; col >= 0; col--) {
      for (int row = text1.length() - 1; row >= 0; row--) {
        if (text1.charAt(row) == text2.charAt(col)) {
          current[row] = 1 + previous[row + 1];
        } else {
          current[row] = Math.max(previous[row], current[row + 1]);
        }
      }
      // The current column becomes the previous one, and vice versa.
      int[] temp = previous;
      previous = current;
      current = temp;
    }
    // The original problem's answer is in previous[0]. Return it.
    return previous[0];
  }
}
```

Since we're solving mn subproblems, and each is an O(1) operation to solve, so time complexity O(nm).

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

Find a space complexity of your solution for Problem 1.

We don't need to create a matrix, we only use two 1D arrays at a time; each the length of the shortest input word. One records the values of row of cell we are visiting, another records the row just above the current row, assigning value to the last cell of current row, leaving us with the minimum length out of the two words. Space complexity O(min(m,n)),

Problem 3. Longest increasing subsequence problem (10 pts)

Given a string $x_1 \ldots x_n$ we wish to find the length of its longest increasing subsequence, that is, the largest k for which there exist indices $i_1 < \ldots < i_k$ such that $x_{i_1} < x_{i_2} < \ldots < x_{i_k}$ Show how to do this in time $O(n^2)$.

```java
public int lengthOfLIS(int[] nums) {
  int[] dp = new int[nums.length];
  Arrays.fill(dp, val: 1);
  for (int i = 1; i < nums.length; i++) {
    for (int j = 0; j < i; j++) {
      if (nums[i] > nums[j]) {
        dp[i] = Math.max(dp[i], dp[j] + 1);
      }
    }
  }
  int longest = 0;
  for (int c : dp) {
    longest = Math.max(longest, c);
  }
  return longest;
}
```

We use two nested for loops resulting in 1+2+3+4+...+N=N∗(N+1)/2 operations, resulting in a time Complexity of O(n^2). Space complexity O(N), the only extra space we use relative to input size is the dp array, which is the same length as nums.