Exercise 1: Sub-Sequences  

(a) Consider a sequence $A = a_1, a_2, \ldots, a_m$ of characters. A sub-sequence $S$ of $A$ is a sequence that can be obtained by deleting some characters in $A$. Hence, for example, $a, r, e, w, a, a$ is a sub-sequence of $b, a, a, r, e, z, w, a, b, a$. Given two sequences $A$ of length $m$ and $B$ of length $n$, describe an $O(m \cdot n)$ time dynamic programming algorithm to compute the length of a longest common sub-sequence of $A$ and $B$! To describe the algorithm, specify the sub-problems you need to solve (and store in a table), the initialization, as well as the recursive rule to compute the sub-problems.

(b) Apply your algorithm to the inputs $A = a, e, a, a, g, s, s, t$ and $B = b, a, e, b, s, a, g, t$ by setting up and filling out the table you need. Explain how you get the length of the longest common sub-sequence from your table. Also show how to compute a longest common sub-sequence of $A$ and $B$.

(c) For two character sequences $A$ and $B$, a super sequence $C$ is a character sequence such that both $A$ and $B$ are sub-sequences of $C$. Give an efficient algorithm for finding the shortest common super-sequence of two strings $A$ and $B$.

**Hint:** Try to find a connection between the longest common sub-sequence of $A$ and $B$ and the shortest common super-sequence of $A$ and $B$.

Exercise 2: Sharing an Inheritance  

Assume that Alice and Bob are two siblings who inherit some goods from their deceased parents. In order to share the inherited goods in a fair way, Alice and Bob decide about a value $v_i$ for each inherited item $i$ and they would like to share the goods so that both of them get items of the same total value.

Assume that there are items $1, \ldots, n$ and that each item $i$ has an integer value $v_i > 0$. Give a dynamic programming algorithm that determines whether the $n$ items can be partitioned into two parts of equal total value. If the items can be partitioned like this, the algorithm should also allow to compute such a partition. What is the time complexity of your algorithm? What assumptions do you need to get an algorithm that runs in time polynomial in the total number of items $n$?
Exercise 3: Binomial Queue [9 Points]

We have seen in the lecture that when using a Binomial heap to implement Dijkstra’s algorithm for a graph $G$ with $n$ nodes and $m$ edges, we can upper bound the total running time by $O(m \log n)$. The goal of this exercise is to show that this bound is tight for large enough $m$.

(a) Give a graph $G = (V,E)$ with positive edge weights and $m = \binom{n}{2} = \Theta(n^2)$ edges on which Dijkstra’s algorithm implemented with a Binomial heap requires $\Theta(n^2 \log n)$ time.

Hint: You can assume that the initially, the nodes are inserted into the heap in the worst possible order.

(b) Try to generalize your construction from Question (a) such that for a large enough value of $m$, you get a graph with $m$ edges on which Dijkstra’s algorithm implemented with a Binomial heap requires $\Theta(m \log n)$ time. How small can you make $m$ such that your construction still works?

Exercise 4: Fibonacci Heap [8 Points]

Fibonacci heaps are only efficient in an amortized sense. The time to execute a single, individual operation can be large. Show that in the worst case, both the delete-min and the decrease-key operations can require time $\Omega(n)$ (for any heap size $n$).

Hint: Describe an execution in which there is a delete-min operation that requires linear time and describe an execution in which there is a decrease-key operation that requires linear time.