1. **Protein Bonding** Let $\Sigma$ be a finite set of amino acids, and let $w = w_1 \ldots w_n$ be a sequence of acids from $\Sigma$. For $\sigma, \sigma' \in \Sigma$, let $b(\sigma, \sigma')$ be the strength of a bond between the two types of acids, a non-negative real number. A *bonding* of the sequence is a partial matching between positions in the word so that matched pairs can be connected with lines drawn below the word without lines crossing. Equivalently, it should satisfy:

1. Each position $i$ is bonded with at most one other position $j \neq i$.
2. There are no two bonded pairs $i_1, j_1$ and $i_2, j_2$ with $i_1 \leq i_2 \leq j_1 \leq j_2$.

The **total bond strength** is the sum over all bonded positions $i, j$ of the bond strength $b(w_i, w_j)$.

Give as efficient as possible algorithm to find the bonding of a proteing sequence that maximizes the total bond strength. (8 points correct, polyn-time algorithm; 2 pts efficiency. I know an $O(n^3)$ algorithm.)

2. **Viterbi** Problem 15-5 from text (p. 367)

3. **Addition chains** An *addition-chain* for $n$ is a sequence of numbers starting with 1 and ending with $n$ so that any element of the sequence other than 1 is the sum of two (not necessarily distinct) earlier elements. For example, two addition-chains for 13 are $1, 2 = 1 + 1, 4 = 2 + 2, 8 = 4 + 4, 9 = 8 + 1, 13 = 9 + 4$ and $1, 2 = 1 + 1, 3 = 1 + 2, 5 = 3 + 2, 8 = 5 + 3, 13 = 8 + 5$. The **cost** of an addition-chain is the total number of additions, or, equivalently, the number of elements -1. So both addition chains above have cost 5. Describe and give a worst-case analysis of a back-tracking algorithm to find the minimum cost addition chain for input $n$.

4. **Distributed data structure** You have a data structure $S$ that is too large to store on one machine, so you plan to store the $n$ different elements on $k$ identical machines. The elements are ordered, and, to make location easier, each machine will store a consecutive sequence of elements. Some elements $e_i$ have pointers to other elements $e_j$. You want to minimize the number of pointers that go between different machines. At most $t$ elements can fit on any one machine, where $tk \geq n$.

   a. (2 points) Formalize the above problem specification as a combinatorial optimization problem, e.g., give the Instance, Solution Format, Constraints, and Objective.
   
   b. (8 points) Give an efficient algorithm for the problem.
5. **Dynamic Programming vs. Memoization** Implement the bottom-up dynamic programming algorithm for the length of the longest common subsequence (page 353), and implement a top-down, memoized version of the same recurrence. For a range of sizes $n$ (say, $n = 2^4$ to $n = 2^{15}$ or as high as time permits), compare the performance of these two algorithms on inputs consisting of two random strings of length $n$ over an alphabet of size 4. (You can present times with a log-log chart. Use all times, not just cycles.) Which approach is better and why? Is there a crossover point?

When describing an algorithm, don't write out an entire pseudocode; just describe it at a high level. Be sure to specify completely all data structures used in the algorithm. Include time analysis for all algorithms. Some problems will refer to or be easier if you know material from the book not covered in class. Read the preceding sections.