General directions: Algorithms may be described at "high level" without actual code. You may use any lower bound, algorithm or data structure from the text or in class, and their correctness and analysis, but be careful. For example, if you construct a graph with $n^2$ nodes and $n^3$ edges and then run Dijkstra's algorithm on the resulting graph, the total run-time is $O(n^3 \log n)$, because Dijkstra's algorithm is $O(E \log V)$. If you want to use the correctness of Dijkstra's algorithm on the graph, you must check that edge weights are positive, since otherwise the proof that Dijkstra's algorithm gives shortest paths does not go through. If a problem has multiple size parameters, you should express the run-time as a function of all relevant parameters; e.g., saying maximum bi-partite matching takes time $O(|V||E|)$ is more accurate than to say it takes time $O(V^3)$, although both are correct.

For the first four problems, you must prove correctness and give a time analysis. For some problems, correctness may be trivial; for others the analysis may be trivial. If it is trivial, you do not need to go into detail, but you should at least give a one or two line explanation. (Conversely, if you only give a one or two line explanation, I will view this as implicitly claiming that it is trivial.) Each problem is worth 20 points. Grading may be based on any of the following that are relevant for the problem: the efficiency of your algorithm; the correctness and proof thereof; and the time analysis. The number of points depending on each part is given after the problem, as well as a ballpark estimate of the time analysis for my solutions of the problem.

**Grade maximization** Problem 20 on pp. 329-330. Note that this is essentially the same as the homework grade maximization problem on the previous assignment, except that we drop the assumption that the improvement in your grade is monotone. (The total grade is monotone, but not necessarily the improvement). 14 points correct polynomial-time algorithm, 6 points time analysis and efficiency. My best time is $O(nH^2)$.

**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: there are no two bonded pairs $i_1, j_1$ and $i_2, j_2$ with $i_1 < i_2 < j_1 < j_2$. 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 protein sequence that maximizes the total bond strength. (15 points
correct, polytime algorithm with proof; 5 pts efficiency. I know an \( O(n^3) \) algorithm.)

**Monotone matching** Let \( G = (L, R, E) \) be a bipartite graph with \(|L| = |R| = n\), and where the nodes in \( L \) are ordered as \( u_1...u_n \) and the nodes in \( R \) as \( v_1...v_n \).

Then a matching \( M \) is *monotone* if whenever \( u_i \) is matched to \( v_j \), and \( u_{i'} \) is matched to \( v_{j'} \), if \( i < i' \) then \( j < j' \). The problem is: given such a bipartite graph \( G \), find the maximum size of a monotone matching in \( G \). (4 points correct poly time algorithm, 10 points correctness proof, 6 points efficiency and time analysis. My algorithm is \( O(n^2) \).)

**Job allocation** You are given a set of \( n \) jobs and a set of \( n \) machines. For each \( 1 \leq i \leq n \) and \( 1 \leq j \leq n \), you are given a quantity \( T_{i,j} \) so that if you assign job \( i \) to machine \( j \), it will take \( T_{i,j} \) time. You need to assign each job \( j \) to a machine \( M[j] \), so that every machine gets exactly one job assigned to it. Your goal is to minimize the time all jobs are finished, \( \max_j T_{j, M[j]} \).

(10 points correct algorithm and correctness proof, 10 points efficiency. A hint is to consider the decision version: Given \( T \) is there a way to assign jobs to machine so that all are finished within \( T \) time?).

**Number puzzle** You are trying to solve the following puzzle. You are given the sums for each row and column of an \( n \times n \) matrix of integers in the range 1...\( M \), and wish to reconstruct a matrix that is consistent. In other words, your input is \( M, r_1,...r_n, c_1,...c_n \). Your output should be a matrix \( a_{i,j} \) of integers between 1 and \( M \) so that \( \forall j \sum_i a_{i,j} = r_j \) and \( \forall i \sum_j a_{i,j} = c_i \); if no such matrix exists, you should output, "Impossible". Give an efficient algorithm for this problem. (15 points correctness for a poly-time algorithm; 5 points efficiency. My best algorithm takes \( O(n^3) \)).

**Dynamic Programming vs. Memoization** Implement the bottom-up dynamic programming algorithm for the length of the sequence alignment algorithm (section 6.6) (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 cross-over point?