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 $O(|V| \log |E|)$ 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 10 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.

**Maximum bandwidth path** Let $G$ be a directed graph with positive edgeweights representing the bandwidth of channels between nodes in a network. If $p$ is a path from $s$ to $t$ in $G$, the bandwidth of $p$, denoted $bw(p)$ is the minimum weight edge in $p$. The Maximum Bandwidth problem is given $G$ and two nodes $s$ and $t$ in $G$, find a path from $s$ to $t$ in $G$ of maximum possible bandwidth. Give an efficient algorithm for this problem. (3 pts. correct polytime, 7 pts. efficiency; efficient algorithms will be better than quadratic time.)

**Data structures for all pairs shortest paths** You need to solve single source shortest path repeatedly on the same underlying graph $G$, but with different edge weights. For each, give an answer for both the case when edge weights are positive and where edge weights can be negative but there are no negative cycles. Describe what algorithms and data structures you would use for optimal efficiency if:
a. **2d grid** $G$ is the $n^{1/2} \times n^{1/2}$ grid, whose nodes are pairs $(i, j)$ with $1 \leq i, j \leq n^{1/2}$ and where $(i, j)$ is connected to $(i+1, j), (i-1, j), (i, j+1)$ and $(i, j-1)$ for $1 < i, j < n^{1/2}$.

b. **hypercube** $G$ is the $d = \log n$ dimensional hypercube, where nodes are $S \subseteq \{1, \ldots, d\}$, and two nodes $S, T$ are connected if and only if $S \subseteq T$ and $|S| = |T| - 1$ or vice versa.

c. **complete bipartite graph** $G$ is the complete bipartite graph with $n/2$ nodes on each side. (All points for efficiency, 4, 3, 3 pts per problem.)

**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 $1, \ldots, M$, and wish to reconstruct a matrix that is consistent. In other words, your input is $M, r_1, \ldots, r_m, c_1, \ldots, c_n$. 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. (8 points correct, poly-time algorithm; 2 points efficiency. My best algorithm takes $O(n^3)$.)

**Sub-trees** A similar problem to the tree isomorphism problem from the first homework is to decide whether one tree is a subtree of another. Let $T_1$ and $T_2$ be rooted trees and let $r_1$ and $r_2$ be their roots. Let $p_1(x), p_2(y)$ be the parent of nodes $x \in T_1, y \in T_2$ in the relevant tree. We say that $T_1$ is a sub-tree of $T_2$ if and only if we can find a 1-1 map $m$ (not necessarily onto) such that $m(r_1) = r_2$ and $m(p_1(x)) = p_2(m(x))$ for every node $x \in T_1$. Give a poly-time algorithm to determine whether $T_1$ is a sub-tree of $T_2$ if $T_1$ and $T_2$ are rooted trees. Don’t assume they are binary trees, or have small degrees. (7 pts. correct poly-time, 3 pts. efficiency. I think all algorithms will be at least cubic time, but maybe you’ll surprise me.)

**Implementation problem: Popular Websites** A web-company wants a data structure that will display webpages by popularity, displaying the top k. The input will be a sequence of IP addresses. Intermittent, the data structure will need to display the top $k$ most frequently visited sites. The data structure should update its list after every new site hit. So the data structure needs to store a list of webpages, ordered by $hit(site)$, the number of hits on the site. It needs to update this list each time a new hit is made, i.e., Update(site) adds site to the list with 1 hit, if it isn’t already there, or increments $hit(site)$ if it is. Top(k) needs to report the top $k$ most popular sites. Describe and implement at least two data structures for this problem. Compare their performances on test data generated as follows: Discuss any conclusions or issues that arose.
Test distribution: A sequence of 1,000,000 random web addresses, each with probability 1/4 of being of the form: random 3 letter word.cs.edu probability 1/4 of the form: free.random 3 letter word.com and probability 1/2 of the form: random 2 letter word.random 2 letter word.com, org, edu. All words are lower case and have only standard letters. After every 1000 sites, perform $Top(k)$ for $k = 2^i$, $i$ uniformly chosen from 0 to 10.