Answer all questions. Give informal (at least) proofs for all answers. Grading will be on completeness and logical correctness, and if applicable, efficiency, as well as correctness. Out of 80 points.

Greedy algorithms
Below, a greedy strategy is given for a problem. Then a lemma that helps prove the strategy optimal is stated, and a proof with some missing phrases is given. Fill in the missing phrases to complete the proof (20 points). Then give an efficient algorithm that implements the greedy strategy (with time analysis) (20 points).

Ripping Sequence
A ripping sequence in an undirected graph is an ordering of the nodes as \(v_1 \ldots v_n\).

The forward degree of a node \(v_i\), is the number of neighbors that come later in the order, i.e., \(|N(v_i) \cap \{v_{i+1} \ldots v_n\}|\). The problem is to find a ripping sequence of minimal maximum forward degree.

The greedy strategy is to find a node of lowest degree, place that node first, and delete the node from the graph. Repeat, placing the new low degree node second, etc.

Theorem: The greedy algorithm produces an ordering of minimal maximum forward degree.

Proof: Let \(GO\) be the ordering produced by the greedy algorithm and let \(OO\) be any optimal solution, i.e., any ordering with the smallest possible \(d\). Let \(d\) be the maximum forward degree of \(GO\), and \(d'\) that of \(OO\). We need to show \(d\leq d'\). Let \(u\) be a vertex that has the maximum forward degree in \(GO\), i.e., where the forward degree of \(u\) is \(d\). Let \(G'\) be the graph when we delete all the nodes that occur before \(x\) in \(GO\). Then, in \(G'\), \(x\) has degree \(d'\), and this is the degree of any node in \(G'\). Thus, every node in \(G'\) has at least \(d\) neighbors that are also in \(G'\).

Let \(y\) be the first node from \(G'\) to occur in \(OO\). From the above, \(y\) has \(d\) neighbors. Since \(y\) is the first node in \(G'\), all of these neighbors are later in the order in \(OO\). Thus, the forward degree of \(y\) in \(OO\) is at least \(d\), so the forward degree for \(OO\), \(d'\), is at least \(d\). Since \(OO\) had a minimal maximum forward degree \(d'\) and \(G'\)'s maximum forward degree \(d\) is at most this, \(GO\) also has \(d\) as its maximum forward degree. QED.

Divide-and-Conquer
Below, we give a divide-and-conquer algorithm in pseudo-code. Write down a recurrence for its time (20 points). Then solve the recurrence to get an explicit formula up to order for the time of the algorithm. (20 points)

Binary Tree Iso Revisited
Some graduate students in 202 pointed out that the Tree Isomorphism algorithm I gave on the homework can be improved, changing its run-time to something smaller. Their improvement follows.

Consider the following recursive algorithm, which makes the following assumptions. \(x, y\) are the roots of two binary trees, \(T_x\) and \(T_y\). \(Left(z)\) is a pointer to the left child of node \(z\) in either tree, and \(Right(z)\) points to the right child. If the node doesn’t have a left or right child, the pointer returns “NIL”. Each node \(z\) also has a field \(Size(z)\) which returns the number of nodes in the sub-tree rooted at \(z\). \(Size(NIL)\) is defined to be 0. \(Size(x)\) has already been computed before this part of the algorithm is run, and so is just an \(O(1)\) time lookup.

The algorithm \(SameTree(x,y)\) returns a boolean answer that says whether or not the trees rooted at \(x\) and \(y\) are isomorphic, i.e., the same if you ignore the difference between left and right pointers.

1. Program: \(SameTree(x,y: Nodes): Boolean;\)
2. IF \(Size(x) \neq Size(y)\) THEN return \(False;\) halt.
3. IF \(x = NIL\) THEN return \(True;\) halt.
4. IF \(SameTree(Left(x), Left(y))\) THEN Return \(SameTree(Right(x), Right(y))\)
5. ELSE Return \((SameTree(Right(x), Left(y))\ AND SameTree(Left(x), Right(y)))\)