CSE 101 Final Exam
Time: 3 Hours

Some problems have multiple parts; do all parts. EXPLAIN ALL ANSWERS, with at least a few lines or sentences of precise English.

Order Notation
For each of the following answer “True” or “False” and give a brief explanation (1 or 2 lines or sentences.) Each is worth 4 points, 2 points for the correct answer and 2 points for the explanation or proof.

1. $1000n + 12500 \in \mathcal{O}(n \log n)$
2. $n^2 + (n - 1)^2 + (n - 2)^2 + ... 1^2 \in \mathcal{O}(n^2)$.
3. $2^{3n} \in \mathcal{O}(2^n)$.
4. If $n \geq 16$, an $\mathcal{O}(n \log n)$ time algorithm is always at least four times faster than an $\mathcal{O}(n^2)$ time algorithm.
5. If $f$ and $g$ are any positive, non-decreasing functions, then $(f(n) + g(n))^2 \in \Theta((f(n))^2 + (g(n))^2)$ (Prove or give counter-example.)

Divide and Conquer
The maximum weight sub-tree problem is as follows.
You are given a balanced binary tree $T$ of size $n$, where each node $i \in T$ has a (not necessarily positive) weight $w(i)$ for each node $i \in T$. (Every node in $T$ has pointers to its left-child, right-child, and parent, and you are given a pointer to the root of the tree. A NIL field for the children means the node is a leaf, and for the parent, means the node is the root. You are given a pointer to the root $r$ of $T$.) A rooted sub-tree of $T$ is a connected sub-graph of $T$ containing the root $r$. (So a sub-tree is not necessarily the entire sub-tree rooted at a node. However, it cannot contain the children of a node without containing the node.) You wish to find the maximum possible value of the sum of weights of nodes in a rooted sub-tree $S$ of $T$, $\sum_{i \in S} w(i)$.
Here is a recursive algorithm that solves this problem, given a pointer to the root of $T$:
MaxWtSubtree $[r]$

1. IF $r = NIL$ return 0.
2. $A \leftarrow \max(O, \text{MaxWtSubtree}[r.leftchild])$
3. $B \leftarrow \max(O, \text{MaxWtSubtree}[r.rightchild])$

1
a Give a recurrence and a time analysis for this algorithm in the case when \( T \) is a complete binary tree of height \( h \) and size \( n = 2^h - 1 \) (10 pts.)

b Prove that the same worst-case bound holds if \( T \) is any tree of size \( n \). (10 pts.)

Monotone matchings All the remaining questions concern variations of the following problem.

Let \( G \) be a bipartite graph, with \( L = \{u_1, \ldots, u_l\} \) the set of nodes on the left, \( R = \{v_1, \ldots, v_r\} \) the set of nodes on the right, \( E \) the set of edges, each with one endpoint in \( L \) and the other in \( R \), and \( m = |E| \) the number of edges.

A matching in \( G \) is a set of edges \( M \subseteq E \) so that no two edges in \( M \) share an endpoint (neither the one in \( L \) nor the one in \( R \)). A matching \( M \) is monotone if for every two edges \((u_{i_1}, v_{j_1})\) and \((u_{i_2}, v_{j_2})\) in \( M \), if \( i_1 < i_2 \) then \( j_1 < j_2 \). That is, one could draw all the edges in the matching without crossing, if the nodes are put in order on the two sides.

The problem is, given a bipartite graph \( G \), find the largest monotone matching in \( G \).

Assume \( l \leq r \). Then a monotone matching \( M \) is perfect if it has size \( l \), i.e., \(|M| = l\).

Greedy Algorithms and data structures Part 1 : 10 points Below is a greedy strategy for the largest monotone matching problem. Give a counter-example where it fails to produce the optimal solution. (Hint: Since below you will show that the algorithm works when the maximum monotone matching is perfect, your example shouldn’t have a perfect monotone matching.)

Candidate Strategy A : For each \( i = 1 \) to \( l \), if \( u_i \) has at least one undeleted neighbor \( v_j \), match it to the unmatched neighbor with smallest value of \( j \). Then delete \( u_i \) and \( v_1, \ldots, v_j \), and repeat.

Part 2: 5 pts Illustrate the above strategy on the following graph with a perfect matching: \( L = \{u_1, u_2, u_3\} \), \( R = \{v_1, v_2, v_3, v_4, v_5\} \), and \( E = \{(u_1, v_2), (u_1, v_3), (u_1, v_4), (u_2, v_1), (u_2, v_2), (u_2, v_3), (u_2, v_5), (u_3, v_1), (u_3, v_5)\} \)

Part 3: 10 points Prove that, if \( G \) has a perfect matching, then Candidate Strategy A finds one.

(Hint: Use one of the following two methods. Let strategy A match \( u_i \) with \( v_{j_i} \) (unless it can’t be matched). Let \( OPT \) be a perfect matching that matches each \( u_i \) with \( v_{k_i} \). (Note: all left nodes will be matched by \( OPT \), since it is perfect.)

Transformation method: Prove by induction on \( T \) that there is a left-perfect matching \( OPT_T \) that matches each \( u_i \) with \( v_{j_i} \) for \( 1 \leq i \leq T \).
OR
Greedy-stays ahead: Prove by induction on \( T \) that \( v_{j_T} \) exists and that \( j_T \leq k_T \).

**Part 4: 10 points** Describe an efficient algorithm that carries out the strategy. Your description should specify which data structures you use, and any pre-processing steps. Assume the graph is given in adjacency list format. Give a time analysis, in terms of \( l, r \) and \( m \).

**Back-tracking and Dynamic Programming** The following recursive algorithm for the maximal monotone matching problem finds the maximum matching whether or not it is perfect. It branches on whether a node on the left is matched or unmatched. By the analysis of greedy algorithm A above, we can see that when a node is matched, it should always be matched to its smallest neighbor. The backtracking algorithm just returns the size of the maximum monotone matching, not the actual matching.

\[
\text{BTMMM}(G = (L = \{u_1, .., u_l\}, R = \{v_1, .., v_r\}, E): \text{bipartite graph})
\]

1. IF \(|L| = 0\) return 0.
2. \(\text{Unmatched} \leftarrow \text{BTMMM}(G - \{u_1\})\).
3. IF \(|N(u_1)| = 0\) return \(\text{Unmatched}\).
4. Let \( J \) be the first neighbor of \( u_1 \), i.e., the smallest value so that \( v_J \in N(u_1) \).
5. \(\text{Matched} \leftarrow 1 + \text{BTMMM}(G - \{u_1, v_1, .., v_J\})\)
6. Return \(\text{Max}(\text{Matched}, \text{Unmatched})\)

**Part 1: 5 points** Illustrate the above algorithm on your counter-example graph for the greedy strategy, (as a tree of recursive calls and answers.)

**Part 2: 5 points** Give an upper bound on the number of recursive calls the above algorithm makes, in the worst-case. (Be sure to explain your answer.)

**Part 3: 10 points** Give a dynamic programming version of the recurrence.

**Part 4: 5 points** Give a time analysis of this dynamic programming algorithm.

**Part 5: 5 points** Show the array or matrix that your dynamic programming algorithm produces on the example graph from Part 2 of the greedy algorithm.