CSE 101 Sample Final
Midterm: Nov 15
Topics: Order, Recurrence Relations, Analyzing Programs, Divide-and-Conquer, Back-tracking, Dynammic Programming, Greedy Algorithms and Correctness Proofs, Data Structures (Heap, Binary Search Tree, B-Tree, Lists), Using Data Structures in Algorithms
Time: 3 Hours
Each problem =10 points. Some problems have multiple parts -do all parts.

Order Notation For each of the following answer "True" or "False" and give a brief explanation (1 or 2 lines or sentences.) (4 points each)

1. $n^2 + n \in \Theta(n^2)$.
2. $2^{\lceil \log_2 n \rceil} \in Theta(n)$.
3. $n \log n \in O(n^2)$
4. $n \log n \in \Omega(n^2)$
5. If $f(n) \in O(g(n))$ then $f(n) + g(n) \in O(g(n))$.

Divide and Conquer: 20 pts Consider the following recursive algorithm: We are given a binary tree T. In addition, at each node x except the root, we are given a positive real value $d(x)$ called the distance between x and its parent, We want to find the distances between every two nodes x, y in the tree and store it in an array $D[x, y]$.

Let L_x represent the left sub-tree rooted at x and R_x the right sub-tree. The algorithm to do so is as follows:

1. Distances(root){compute all distances between pairs of nodes in the sub-tree rooted at *root*. }
2. If $root = NIL$ then halt.
3. ELSE do:
4. begin;{else}
5. Distances(lc(root));
6. Distances(lc(root));
7. For each $I \in L_{root}$, $D[I, root] := D[I, lc(root)] + d(lc(root))$.
8. For each $J \in R_{root}$, $D[J, root] := D[J, rc(root)] + d(rc(root))$.
9. For each $I \in L_{root}$ do:
10. For each $J \in R_{root}$ do:
11. $D[I, J] = D[I, root] + D[J, root]$.
12. end;{else}

Consider the case when the tree is almost perfectly balanced, i.e, for every x, $|L(x)|-1 \leq |R(x)| \leq |L(x)|+1$. Give a recurrence relation for the time the algorithm takes in this case, and solve it to get the order of the time.

Back-tracking and Dynamic Programming A sub-sequence of a word is a word that can be obtained by deleting some letters in the word. A palindrome is a word that is the same backwards as forwards. Let $w \circ u$ represent the concatenation of two words w and u. The following recursive algorithm finds the longest palindrome that is a subsequence of the word $w[1]w[2]...w[n]$:

1. MaxPal[w[1]...w[n]];
2. If $n = 0$ return NIL ELSE:
3. If $n = 1$ return $w[1]$ ELSE :
4. If $w[1] = w[n]$ return $w[1] \circ MaxPal[w[2]...w[n-1]] \circ w[n]$ ELSE: return $Longer(MaxPal[w[2],..w[n]], MaxPal[w[1],..w[n-1]])$

Here *Longer* is a routine that compares two words and returns the longer word, breaking ties arbitrarily.

1. Show the recursion tree of the above algorithm on the word $EDGED$. (5 points)
2. Give a bound on the worst-case number of recursive calls the algorithm could make on a word with n symbols. (5 points)
3. Give a polynomial-time dynammic programming version of the recurrence. (10 points)
4. Give a time analysis of the dynammic programming algorithm. (5 points)
5. Show the array that your algorithm produces on the word $EDGED$. (5 points)

Greedy Algorithms and use of data structures in algorithms Consider the following problem. You are trying to schedule your course work for the next n day quarter. You have a list of m asssignments for all your courses, where for each assignment a_i, you have a day $start_i$ when the assignment will be given to you, and a day end_i, the last day you could do it before the due date. (For example, if the assignment were given out the morning of the 13th and due early the next morning, then both $start_i$ and end_i would equal 13.) You know that each assignment will take one uninterrupted day of work.

You wish to perform as many of the assignments before their due dates as possible, scheduling each assignment you will complete for a different day. To be completed, a_i must be scheduled at a day d_i with $start_i \leq d_i \leq end_i$. No two assignments can be scheduled on the same day.

Parts 1 and 2: 10 points for the proof, 5 points for counter-example Below are two greedy strategies for this problem. One produces optimal solutions, the other does not. Decide which one produces optimal solutions. Give a correctness proof for this strategy, and a counter-example for the other.

Candidate Strategy one: Schedule the assignment in order of start days, except for those moved beyond their due dates by earlier assignments. Break ties by performing the first assignment due, first. Peform each assignment on the first possible free day, given the order.

Candidate Strategy two: Schedule assignments from the first day of the quarter to the last. Each day, look at the set of assignments that are available, but not already scheduled. Schedule the one with the first due date.

Part 3: 5 points Use the strategy you chose to show how to compute the optimal order for the following example:

A1: start day 1, due day 6; A2: start day 1, due day 3; A3: start day 2, due day 6; A4: start day 3, due day 4; A5: start day 3, due day 4; A6: start day 4, due day 6;

Part 4: 10 points For the strategy you choose, describe an efficient algorithm that carries out the strategy. Your description should mention which data structures you use, and any pre-processing steps. Give a time analysis.