CSE 101 Sample Final

Topics: Order, Recurrence Relations, Analyzing Loops, Invariants and induction, Divide-and-Conquer, Back-tracking, Dynammic Programming, Greedy Algorithms and Correctness Proofs, 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 balanced binary tree T, where a tree is balanced if, for every node, its left sub-tree has the same size as its right sub-tree. 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}

Give a recurrence relation for the time the algorithm takes 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;
3. IF $n = 1$ return $w[1]$;
4. IF $w[1] = w[n]$
5. THEN return $w[1] \circ MaxPal[w[2]...w[n-1]] \circ w[n]$
6. 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.

Part 1: 5 points Show the recursion tree of the above algorithm on the word $EDGED$.

Part 2: 5 points Give a bound on the worst-case number of recursive calls the algorithm could make on a word with n symbols.

Part 3: 10 points Give a polynomial-time dynammic programming version of the recurrence.

Part 4: 5 points Give a time analysis of the dynammic programming algorithm.

Part 5: 5 points Show the array that your algorithm produces on the word $EDGED$.

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.

Part 1 : 5 points The following is an incorrect greedy strategy for this problem. Give a counter-example that shows that this strategy fails to produce optimal schedules.

First Start Date: Schedule the assignment in order of start days, breaking ties in favor of the first one due. For each assignment, if there is an unscheduled day between its start and end date, schedule it for the first such day; otherwise, omit it from the schedule.

Part 2: 5 points The following is a correct greedy strategy for this problem. Show the steps of the algorithm on the counter-example you gave above. (If you couldn't find a counter-example, use the example below.)

Just-in-time: 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.

If you couldn't find an example in Part 1, use this example for part 2:

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 3: 10 points The following is a proof of a modify-the-solution lemma for the Just-in-time strategy, with some phrases missing. Fill in the missing spaces in the proof, which have Roman numerals indexing them.

Lemma: Let $Assign$ be a set of (unscheduled) assignments, and let d be the first unscheduled day. Let $A = (start_A, end_A)$ be an assignment in $Assign$ so that $\quad I \quad \leq d$ and of all such assignments, $\quad II \quad$ is minimal. Let S_2 be any schedule of assignments in $Assign$ to days d or later, so that S_2 does not schedule $\quad III \quad$ on $\quad IV \quad$. Then there is a schedule of assingnments in $Assign$ to days d or later, S_1 so that $S_1 \quad V \quad$ and the $\quad VI \quad$ is at least that for S_2.

Proof: One of four cases must hold:

1. S_2 schedules some other assignment A' on d, and schedules A on some other day, d'.
2. S_2 schedules some other assignment A' on d, and does not $\quad VII \quad$ at all.

3. S_2 schedules no assignment on d, and schedules $VIII$
4. S_2 IX and does not X at all.

In the first case, let S_1 be the schedule that schedules all assignments except A and A' as S_2 does, but XI and XII. S_1 schedules the $XIII$ assignments S_2 does, so in particular the number of assignments scheduled is at least as large. To see that S_1 is a legal schedule, observe that $start_{A'} \leq d$, since otherwise it couldn't be scheduled on date d in S_2. Then $end_{A'} \leq$ XIV $\leq d'$ since A was the unscheduled assignment that started before d and was the first to end. Also, $d \leq d'$, since d is the first unscheduled date. Thus, $start_{A'} \leq$ XV $\leq d' \leq$ XVI, so scheduling A' on d' is legal.

In the second case, let S_1 be the schedule that schedules all assignments except A and A' as S_2 does, but schedules A on d, and $XVII$ at all. Since S_1 schedules one assignment S_2 does not, and does not schedule one assignment S_2 does, it schedules the same number of assignments. Since by assumption, A can be scheduled on d, S_1 is a legal schedule.

In the third case, let S_1 be the schedule that schedules all assignments except $XVIII$, which it schedules on XIX. Then S_1 schedules the same set of assignments as S_2, so it schedules at least as many.

In the last case, let S_1 be the schedule that XX. Then S_1 schedules XXI assignment than S_2.

Thus, in all cases, there is a legal schedule S_1 that $XXII$, and $XXIII$, as required.

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