CS 6363 PhD Practice Qualifying Examination

This exam comprises four groups of problems. Each student should try to solve *one* of the four problems from each group during a single 2.5 hour period. Each algorithm should be accompanied by its running time analysis and proof of correctness.

The real QE exam is open book but no calculators allowed.

GROUP #1. (Divide-and-conquer, sorting, searching, etc.)

1. Recall, given an array of distinct numbers $X[1 .. m]$, an element of the array has rank k if it would appear in the kth position after sorting the array. There is an $O(m)$ time procedure SELECT$(X[1 .. m], k)$ which returns the element of rank k in X.

   Suppose we are given an array $A[1 .. n]$ of distinct numbers and an array $R[1 .. r]$ of distinct ranks where $r \leq n$. We wish to compute the element of rank $R[i]$ in A for all $i \in \{1, 2, \ldots, r\}$. We can do so in $O(nr)$ time by running the SELECT procedure once for each value in R.

   Describe an algorithm that returns the element of each rank in only $O(n \log r)$ time total.

2. Let $A[1 .. n]$ be an array of distinct numbers where $n \geq 2$. An element $A[i]$ is a *local maximum* if it is larger than its neighboring entries in A. In other words, if $1 < i < n$, then $A[i-1] < A[i]$ and $A[i] > A[i+1]$. Element $A[1]$ is a local maximum if $A[1] > A[2]$, and element $A[n]$ is a local maximum if $A[n] > A[n-1]$. We can easily find a local maximum in $O(n)$ time by checking each entry of A.

   Describe an $O(\log n)$ time algorithm to find a local maximum.

GROUP #2. (Dynamic programming)

1. Recall the rod cutting problem. We are given a non-negative integer n and an array $P[1 .. n]$ of prices. We wish to cut the rod into a number of pieces whose lengths sum to n in order to sell the pieces. We are paid $P[i]$ for selling a piece of length i. Our goal is to find the maximum total selling price for our pieces of rod.

   Consider a variant of the problem where we can cut the rod into at most k pieces where $k \leq n$ is given as part of the input. Describe an efficient algorithm for this version of the problem.

2. Recall, a subsequence X of a sequence Y is a subset of the elements of Y appearing in order. The elements *do not* need to be consecutive. Equivalently, X is a subsequence of Y if X is obtained by deleting some number of elements from Y. We call Y a *supersequence* of X if X is a subsequence of Y.

Describe an efficient algorithm that, given two arrays $A[1 \,..\, m]$ and $B[1 \,..n]$ of English characters, returns the length of the *shortest* array C that is a *supersequence* of both A and B.

GROUP #3. (NP-Completeness)

(For this group you may assume NP-hardness of SAT, 3SAT, MaxIndSet (and its decision version), MaxClique (and its decision version)), MinVertexCover (and its decision version), 3Color, HamiltonianCycle (in directed and undirected graphs), HamiltonianPath (in directed and undirected graphs), and SubsetSum.

1. The SetCover problem takes as input a set of n *elements* X and a collection of m *subsets* $\{Y_1, Y_2, \ldots, Y_m\}$ of X (i.e., $Y_i \subseteq X$ for all i). The goal is to compute a minimum size sub-collection $\{Y_{i_1}, Y_{i_2}, \ldots, Y_{i_r}\}$ of the subsets such that every element of X is covered. In other words, $\bigcup_{j=1}^{r} Y_{i_j} = X$.

   Prove that SetCover is ***NP-complete***.

2. The three complete graph partition problem (3-CGP) problem takes as input an undirected graph $G = (V, E)$ and asks if it is possible to partition the vertices into at most three subsets so that the induced subgraph of each subset is a complete graph. In other words, for any pair of distinct vertices u and v within a subset, G should contain the edge uv.

   Prove that 3-CGP is ***NP-complete***.

GROUP #4. (Graph algorithms)

1. Suppose you are given a connected undirected graph $G = (V, E)$ along distinct edge weights $w : E \to \mathbb{R}$. Are you also *given* the minimum spanning tree T of G.

   Now, suppose exactly one edge $e \in E$ has its weight decreased to $w'(e) < w(e)$. Describe an $O(E)$ time algorithm to find the new minimum spanning tree of G (the new minimum spanning tree may be equal to T). You may assume the edge weights are distinct, even after decreasing the weight on e.

2. A *cycle cover* of a given directed graph $G = (V, E)$ is a set of vertex-disjoint cycles that cover every vertex in G. Describe an efficient algorithm that either computes a cycle cover for a given graph or correctly reports that no cycle cover exists.

   [Hint: Every vertex in a cycle has one outgoing edge.]