Design and Analysis of Algorithms
Homework 2

Clearly number your solution to each problem. Staple your solutions and bring them to class on the due date. Express your algorithms in pseudo-code when directed. Always provide justification for your answer when asked to give the running time of an algorithm. Be brief and concise, and draw pictures where appropriate.

1. Draw a single binary tree T such that each of the following properties holds:
   - each internal node of T stores a single character
   - a *preorder* traversal of T yields SHELDON, and
   - a *inorder* traversal of T yields LEDHOSN.

2. (a) Give an $O(n)$-time algorithm for computing the depth of each node of a tree T, where n is the number of nodes of T. Assume the existence of methods setDepth(v,d) and getDepth(v) that run in $O(1)$-time.

   (b) Design algorithms for performing the following operations on a binary tree T of size n, and analyze their worst-case running time. Your algorithms should avoid performing traversals of the entire tree.
   - preorderNext(v): return the node visited after node v in a preorder traversal of T
   - inorderNext(v): return the node visited after node v in an inorder traversal of T

3. Let T be a binary tree with n nodes. It is realized with an implementation of the Binary Tree ADT that has $O(1)$ running time for all methods except *positions*() and *elements*(), which have $O(n)$ running time. Give an $O(n)$ time algorithm that uses the methods of the Binary Tree interface to visit the nodes of T by increasing the values of the level numbering function p given in Section 2.3.4. This traversal is known as the **level order traversal**. Assume the existence of an $O(1)$ time visit(v) method (it should get called once on each vertex of T during the execution of your algorithm)

4. (a) Illustrate the execution of the selection-sort algorithm on the following input sequence: (21, 14, 32, 10, 44, 8, 2, 11, 20, 26)

   (b) Illustrate the execution of the insertion-sort algorithm on the following input sequence: (21, 14, 32, 10, 44, 8, 2, 11, 20, 26)

5. Let S be a sequence containing pairs (k, e) where e is an element and k is its key. There is a simple algorithm called count-sort that will construct a new sorted sequence from S provided that all the keys in S are different from each other. For each key k, count-sort scans S to count how many keys are less than k. If c is the count for k then (k, e) should have rank c in the sorted sequence.

   (a) Give the pseudocode for count-sort.

   (b) Determine the number of comparisons made by count-sort. What is its running time?

   (c) As written, count-sort only works if all of the keys have different values. Explain how to modify count-sort to work if multiple keys have the same value.