Data Structures - Assignment no. 2

Remarks:

- **Write both your name and your ID number very clearly on the top of the exercise. Write your exercises in pen, or in clearly visible pencil. Please write *very* clearly.**
- **Give correctness and complexity proofs for every algorithm you write.**
- **For every question where you are required to write pseudo-code, also explain your solution in words.**

1. (a) Implement, in pseudo-code, a trenary (base-3) counter: You are given an infinite array such that each cell can only hold the digits 0,1,2, and you want the counter to support the operation *increment()* that increases the value of the counter by 1.

   (b) Suppose you start from a counter initialized to 0 and you perform m *increment*() operations. What is their total cost? What is the amortized time complexity of increment? Prove your answer.

2. The pre-order read of a binary tree is the sequence of elements that are printed by the following procedure:

```
PRE-ORDER(v)
IF (v = null) RETURN
ELSE
    PRINT v.key
    PRE-ORDER(v.left)
    PRE-ORDER(v.right)
    RETURN
```

The post-order read of a binary tree is the sequence of elements that are printed by the following procedure:

```
POST-ORDER(v)
IF (v = null) RETURN
ELSE
    POST-ORDER(v.left)
    POST-ORDER(v.right)
    PRINT v.key
    RETURN
```

The in-order read of a binary tree is the sequence of elements that are printed by the following procedure:

```
IN-ORDER(v)
IF (v = null) RETURN
ELSE
   IN-ORDER(v.left)
   PRINT v.key
   IN-ORDER(v.right)
   RETURN
```

(a) Give a tree with at least 3 nodes (all nodes must have different keys) such that both its in-order read and its pre-order read are the same, or prove that there is no such tree.

(b) Give a tree with at least 3 nodes (all nodes must have different keys) such that both its in-order read and its post-order read are the same, or prove that there is no such tree.

(c) Give a tree with at least 3 nodes (all nodes must have different keys) such that both its pre-order read and its post-order read are the same, or prove that there is no such tree.

3. You are given a binary search tree, where each node contains a pointer LEFT to its left child, a pointer RIGHT to its right child, a pointer PARENT to its parent, and a key KEY. The keys are integers that can be positive, negative or zero. Describe a procedure that performs the following operation, and then write pseudo-code for it: The operation is given a pointer to the root of the tree, and needs to output the number of nodes who have the property that the sum of the keys in the node's sub-tree is positive. The procedure should take $O(n)$ time, where n is the number of nodes in the tree.
Hint: Recursion may help.

4. Define the vertex-depth of a tree to be the distance between its root and the furthest leaf, measured in vertices, not in edges. For example, the depth of a tree which contains a single vertex is 1. The depth of an empty tree is 0.

   A binary search tree is called a *valid AVL tree* if for each node v the following condition holds: Let v_1, v_2 be v's children. Then we require that the difference between the depth of the subtree whose root is v_1 and the depth of the subtree whose root is v_2 is -1, 0, or $+1$. For example, Figure **??** depicts a valid AVL tree (the keys are not listed).

   (a) Prove that a valid AVL tree of depth d always has at least F_d vertices, where F_d is the d'th Fibonacci number. ($F_0 = 0, F_1 = 1, F_n = F_{n-1} + F_{n-2}$). (Hint: use induction on d).

   (b) Let T be a valid AVL tree with n vertices. Prove that the depth of T is $O(\log n)$. You may use item (a).

5. Find the order of growth for the following recursively given functions (you may use the Master Method).

   (a) $T(n) = 4T(n/2) + n$
   (b) $T(n) = 4T(n/2) + n^2$
   (c) $T(n) = 4T(n/2) + n^3$
   (d) Explain why $T(n) = 2T(n/2) + n \log^2 n$ cannot be solved using the Mater Method.

6. Find the order of growth for the following recursively given functions. Explain your answer.

   (a) $T(n) = T(n - a) + T(a) + n$ where $a \geq 1$ is constant.
   (b) $T(n) = T(cn) + T((1 - c)n) + n$ where $0 < c < 1$ is a constant.
   (c) $T(n) = 2T(n/2) + n \log^4 n$ (Hint: The approach to solving this is similar to the approach to solving $T(n) = 2T(n/2) + n$).