150015-Semester II - 5781

Data Structures I

**Homework Assignment #6**

**Question 1**

Given: A binary tree (not necesserialy a BST) with the following transversals:

Preorder: 10, 20, 8, 2, 6, 3, 1, 13, 9, 5, 4, 12, 17

Inorder: 2, 8, 3, 6, 20, 10, 4, 5, 12, 9,13, 1, 17

What is the postorder transversal of the tree?

A bit of help for you: Draw the tree and then transverse it.

**Question 2**

Define a function that runs in O(n) time that prints all the values of a rooted tree (not necesseiraly a binary tree) with n nodes. The tree is implemented using a data structure in which each node has 3 pointers: parent, left-child and right-sibling.

**Question 3**

1. Build a BST using the following values from left to right:

29, 21, 17, 40, 8,19, 20, 10, 5, 27, 34, 37.

1. Delete from the above tree the following values, after each delete draw the tree.

20, 29, 8, 21.

**Question 4**

Given: A BST that contains values all different from each other.

*x*  is a leaf and *y* is its parent in the tree.

Prove that one of these conditions exsist:

Either the value of *y* is the greatest among all the values smaller than *x*,

Or

The value of *y* is the smallest among all the values greater than *x*.

**Question 5**

Given: The following function *what* t accepts the roots of two binary trees, the first with n nodes and the the second with m nodes.

what(root1, root2)

{

If (root1 == NULL or root2 == NULL)

return true

if (key(root1) > key(root2))

return false

return what(root1, left(root2)) and

what(root1, right(root2)) and

what(left(root1), root2) and

what(right(root1), root2)

}

1. Explain concisely what the function does. (not how!)
2. Is the run-time complexity of the function Explain.
3. Given that the trees are BST, write a more efficient function that performs the same as the original function but its run time will be *O*(n+m).

**Question 6**

A threaded tree is a BST in which every node of the tree there are two additional attributes(fields), *IsLeft* and *IsRight*, which are logical variables and indicate respectively if the node has a left or right son.

If for a given node IsLeft = false, that is the node does not have a left son, we will store in *left* of the node a pointer to its predecessor.

If for a given node IsRight = false, that is the node does not have a right son, we will store in *right* of the node a pointer to its suceecessor.

1. Write the following functions for a threaded tree: insert, delete, search.
2. What are the run-time complexities of the functions that you wrote? Explain.

**Question 7**

Remember, a binary tree can have 0,1 or 2 non empty sons.

Write a recursive function, nTrees, that accepts an integer n which is greater than or equal to 0, and returns the number of possibille trees containing n nodes. For example, for a tree with exactly 3 nodes there are 5 different trees: