Introduction to Algorithms (CELEN086)

Problem Sheet 7

Topics: Binary tree; Recursive algorithms on binary tree

(Note: For the exercises, assume all values stores in the tree are different.)

1. Draw the following binary trees, and find their sizes and heights.

   i. node(node(leaf,3,leaf),21,node(node(leaf,8,leaf),9,leaf))

   ii. node(leaf,19,(node(node(leaf,45,leaf),32,node(node(leaf,17,leaf),12,leaf))))

2. Consider a binary tree T:

   node(node(leaf,40,node(node(leaf,15,leaf),30,leaf)),10,node(node(leaf,20,leaf),5,leaf))

   i. Draw this binary tree

   ii. Find the depth and height of node with value 30.

   iii. For the node with value 40, find the values stored in its children nodes.

   iv. Find all the values stored in the leaf nodes.

3. Write an algorithm called **isSingle(T)** to determine if the input binary tree has only one node.

4. Write a recursive algorithm called **sum(T)** that takes a binary tree and returns the sum of all the node values.

5. Write a recursive algorithm called **min(T)** that returns the minimum value stored in a non-empty binary tree.

6. Write a recursive algorithm called **add(x,T)** that takes a number x and a non-empty binary tree T, and returns a (new) binary tree with x added to all numbers stored in the (old) tree.

7. Write a recursive algorithm called **height(T)** that compute the height of a non-empty binary tree.

   Note: the height of a single-element tree is 0.

8. Write a recursive algorithm called **depth(x,T)** that computes depth of the node with value x in the binary tree T. If x is not a value stored in the tree, your algorithm should return -1.

9. Write a recursive algorithm called leaf**nodeNum(T)** that counts the total numbers of leaf nodes in a binary tree.

10. [Optional] You have learned two numerical methods in your math class, namely bisection method and iteration method. They are efficient schemes in finding the root of $f(x) = 0$. Moreover, we can implement them in programing with recursive structures.

    Given a function $f(x)$, design a recursive algorithm called **bisection(a, b, error)** that returns the approximate solution to $f(x) = 0$ on the interval (a, b) under the given tolerating error.

    If the existence of root is not guaranteed by the IVT (intermediate value theorem), it should return a message 'Choose a different interval'; if the root exists on (a, b), it should perform the bisection process and return an approximate root value c, such that $|f(c)| < tol$.

    You can call the function f(x) directly as a sub-algorithm, and call the function abs(x) to compute $|x|$.

    (Note: normally the tolerating error are designed as small numbers like 0.01, 0.0001, similar to "correct the root to 2 d.p. or 4 d.p.)