CS:3330 Algorithms
Midterm Exam (100 points)

Closed books and notes (except one sheet of notes)
None of the digital devices is allowed.
10/10/2019

1. (40 points) We insert the following numbers in the given order into an empty binary search tree. (**a)** If the tree is an AVL tree, for each rotation in an insertion, please display the tree before and after rotation. If no rotation happens during an insertion, simply draw the tree after the insertion. Please repeat the above task when the tree is (**b)** a Red-Black tree (use single circle for black nodes and double circle for red nodes) and (**c**) a splay tree:

   1, 5, 2, 3, 4.

2. (30 points) Given a binary search tree t (which is a tree node served as the root of the tree) and an integer x served as key, the predecessor of x in t is the maximal node among all nodes in t whose key is less than x. Please write a non-recursive algorithm (in pseudo-code) *predecessor*(t, x), which takes t and x and returns the predecessor of x in t; if no such node exists, returns null. The available methods include only *key*(n), *isNull*(n), *leftChild*(n), *rightChild*(n), *minimum*(n), and *maximum*(n), where n is a tree node and their meanings are given in the class.

Answer:

```
treeNode predecessor(treeNode t, int x)
    treeNode pred = null
    while true
        if isNull(t) return pred  // x doesn't appear in t
        if (key(t) < x)
            pred = t;              // pred remembers the last right turn
            t = rightChild(t)
        else if (key(t) > x)
            t = leftChild(t)
        else // key(t) = x
            if (isNull(leftChild(t))) return pred
            return maximum(leftChild(t))
```

The worst complexity of predecessor is O(h), where h is the height of t.

3. (30 points) A path in a binary tree t (which is a tree node served as the root of the tree) is a sequence of distinct nodes $x_0, x_1, x_2, \ldots, x_k$, such that x_{i-1} is either parent of x_i or a child of x_i in t, where $0 < i \leq k$, and the length of this path is k. Please design an efficient

algorithm, called diameter(t), which computes the length of the longest paths in the tree t. The available methods include only *isNull*(n), *leftChild*(n), and *rightChild*(n), where n is a tree node.

Answer:

We use a global variable to remember the longest path length found so far, and modify the algorithm for computing the height of a tree to compute the longest path containing the current node. When height(t) is complete, every node has been visited and maxPath contains the longest length of such paths passing each node.

```
int maxPath = 0;

int height(treeNode t)
    // Pre: t is not null
    // Post: If the length of the path from the deepest left side to the deepest right side through t is
    //         longer than the current maxPath, record the length in maxPath;
    //         return the height of the subtree rooted by t.
    if (isNull(leftChild(t)) left = 0 else left = 1 + height(leftChild(t))
    // left is the distance from t to its deepest left side node
    if (isNull(rightChild(t)) right = 0 else right = 1 + heght(rightChild(t))
    // right is the distance from t to its deepest right side node
    // left + right is the length of the longest path containing t in the subtree rooted by t
    if (maxPath < left + right) maxPath = left + right
    return max(left, right)  // height of the subtree rooted by t

int diameter(treeNode t)
   if (isNull(t)) return 0   //
   int h = height(t)
   return maxPath
```

The height algorithm visits each node once and does a constant number of steps at each node, so its complexity is O(n), where n is the number of nodes in the tree t. Hence, the complexity of diameter is also O(n).