Solutions - Topic 8, Recurrence Relations

Useful facts: a binary tree of height 0 consists of exactly one node, the root node. We can also consider “null” to be an empty tree of 0 nodes and undefined height.

1. Write a recursive procedure that counts and returns the number of nodes in a binary tree.

```
countNodes (TreeNode root)
    if (root == null) return 0
    return countNodes(root.left) +
           countNodes(root.right) + 1
```

2. Prove this lemma, following the steps below.

Lemma 1: The number of leaves in a complete binary tree of height h is 2^h.

a. Base case: Show that Lemma 1 is true for h=0.

> When h=0, the formula predicts $2^0 = 1$, which is the number of leaves in a tree of a single node.

b. Induction: Assume that Lemma 1 is true for any complete binary tree of height k-1. Use this to show that the number of leaves in a complete binary tree of height k is 2^k.

> A complete binary tree of height k consists of a tree of a complete binary tree of height k-1, which by hypothesis has 2^{k-1} leaf nodes, plus two more children for each of these leaves, giving $2*2^{k-1} = 2^k$ leaf nodes. QED.

3. Now prove this lemma, following the steps below:

Lemma 2: The number of nodes in a complete binary tree of height h is 2^{h+1}-1.

a. Base case: Show that Lemma 2 is true for h=0.

When h=0, the formula predicts $2^{0+1}-1 = 2^1-1 = 1$ node, so it is correct for h=0 (see "useful facts" above).

b. Induction: Assume that Lemma 2 is true for any complete binary tree of height k-1. Use this to show that the number of nodes in a complete binary tree of height k is $2^{k+1}-1$.

As before, the larger tree consists of a complete binary tree of height k-1, which has $2^{(k-1)+1}-1 = 2^k-1$ nodes by hypothesis, followed by a layer of leaves. By Lemma 1 we know that there are 2^k leaves in the last layer of the larger tree, so we have $2^k-1 + 2^k$ total nodes. Simplifying, $2^k-1 + 2^k = 2(2^k) - 1 = 2^{k+1}-1$. QED.

4. Now consider the runtime complexity of the countNodes procedure you wrote.

a. What is the runtime complexity of countNodes as a function of n, the number of nodes in the tree, given an arbitrary binary tree, and why?

$\Theta(n)$, since it visits each node of the tree exactly once, and there are of course exactly n nodes in the tree (regardless of whether it is a complete binary tree etc.).

b. What is the runtime complexity of countNodes expressed as a function of h, the height of the tree, given an arbitrary binary tree, and why?

$O(2^h)$, since this bounds the maximum number of nodes we can have in a tree of height h, each node must be processed, and each node takes constant time.

We use O because there could be fewer nodes if the tree is not complete.

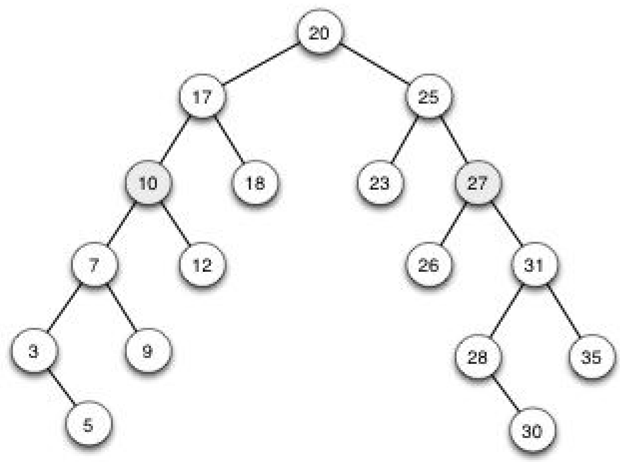
We omit the "-1" because it is a small constant difference.

We also omit the "+1" in the exponent as $2^{h+1} = 2*2^h = O(2^h)$, and 2 is a constant factor (can absorb this in c).

(Notice we ignore the constant "+1" in the exponent of 2, but we cannot ignore the constant difference between n^2 and $n^{1.99999}$. Why?)

Extra Credit (0.4 points each lettered part)

5. Trace the deletion of the nodes with keys 10 and 27 from this Binary Search Tree, indicating for each case what “if/elseif” block is executed. *(You will need to follow the code carefully to get this right: “eyeballing” it may lead to a legal tree that would not result from the code. You do not need to redraw the tree.)*




```
TREE-DELETE(T, z)
 1  if z.left == NIL
 2      TRANSPLANT(T, z, z.right)          // z has no left child
 3  elseif z.right == NIL
 4      TRANSPLANT(T, z, z.left)           // z has just a left child
 5  else y = TREE-MINIMUM(z.right)         // y is z's successor
 6      if y.p ≠ z                         // y lies within z's right subtree
 7          TRANSPLANT(T, y, y.right)      but is not the root of this subtree.
 8          y.right = z.right
 9          y.right.p = y
10      TRANSPLANT(T, z, y)                // Replace z by y.
11      y.left = z.left
12      y.left.p = y
```

(a) Lines executed in deletion of z = node 10:

Tests at 1, 3, 5, and 6, and block 10-12.

(b) Node 12 is now a child of:

node 17

(c) Lines executed in deletion of z = node 27:

Tests at 1, 3, 5 and 6, and all of 7-12.

(d) Node 26 is now a child of:
node 28.

(e) Node 30 is now a child of:
node 31