ICS 311, Spring 2016, Problem Set 04, Topics 7 & 8

Due by midnight Tuesday 2/16

#1. Peer Credit Assignment

1 Point Extra Credit for replying

Please list the names of the other members of your peer group for class last week (2/8 and 2/10) and the number of points you think they deserve for their participation in group work on the two days combined.

- You have a total of 6 points to allocate across all of your peers.
- You can distribute the points equally, give them all to one person, or do something in between.
- You need not allocate all the points available to you.
- You cannot allocate any points to yourself! Points allocated to yourself will not be recorded.

#2. Master Method Practice (6 pts)

Use the Master Method to give tight Θ bounds for the following recurrence relations. Show $a$, $b$, and $f(n)$. Then explain why it fits one of the cases, choosing ε where applicable. Write and simplify the final Θ result.

(a) $T(n) = 2T(n/4) + \sqrt{n}$

(b) $T(n) = 3T(n/9) + n$

(c) $T(n) = 7T(n/3) + n$

#3. Substitution (7 pts)

Use substitution as directed below to solve

$T(n) = 7T(n/3) + n$

It is strongly recommended that you read page 85-86 "Subtleties" before trying this!
(a) First, use the result from the Master Method in 2c as your "guess" and inductive assumption. (When removing Θ, just use one constant c rather than c₁ and c₂.) Take the proof up to where it fails and say where and why it fails. Remember that the steps are:
   1. Write the definition of T(n)
   2. Substitute the inductive hypothesis into this guess on the right hand side.
   3. Simplify the right hand side algebraically to try to derive the exact form of your guess.

(b) Redo the proof, but subtracting \( d_n \) from the guess to construct a new guess. Don’t forget to use the new guess when you do substitution. This time it should succeed: once you get close to your guess you will solve for \( d \) to determine the exact guess.

#4. Binary Search Tree Proof (7 pts)
The procedure for deleting a node in a binary search tree relies on a fact that was given without proof in the notes:

**Lemma:** If a node X in a binary search tree has two children, then its successor S has no left child and its predecessor P has no right child.

In this exercise you will prove this lemma. Although the proof can be generalized to duplicate keys, for simplicity assume no duplicate keys. The proofs are symmetric.

(Hints: Rule out where the successor cannot be to narrow down to where it must be. Draw Pictures!!!)

(a) Prove by contradiction that the successor S cannot be an ancestor or cousin of X, so S must be in a subtree rooted at X.

(b) Identify and prove the subtree of X that successor S must be in.

(c) Show by contradiction that successor S cannot have a left child.

(d) Indicate how this proof would be changed for predecessor.
#5. Deletion in Binary Search Trees (3 pts)

Consider Tree-Delete (page 298, and also copied below).

```
TREE-DELETE(T, z)
1 if z.left == NIL
2 TRANSPLANT(T, z, z.right)
3 elseif z.right == NIL
4 TRANSPLANT(T, z, z.left)
5 else y = TREE-MINIMUM(z.right) // successor
6 if y.p != z
7 TRANSPLANT(T, y, y.right)
8 y.right = z.right
9 y.right.p = y
10 TRANSPLANT(T, z, y)
11 y.left = z.left
12 y.left.p = y
```

(a) How does this code rely on the lemma you just proved?

(b) When node z has two children, we arbitrarily decide to replace it with its successor. We could just as well replace it with its predecessor. (Some have argued that if we choose randomly between the two options we will get more balanced trees.) **Rewrite** Tree-Delete to use the predecessor rather than the successor. Modify this code just as you need to and underline or boldface the changed portions.

#6. Constructing Balanced Binary Search Trees (7 pts)

Suppose you have some data keys sorted in an array and you want to construct a balanced binary search tree from them. Assume a tree node representation TreeNode that includes instance variables key, left, and right.

(a) Write pseudocode (or Java if you wish) for an algorithm that constructs the tree and returns the root node. (We won't worry about making the enclosing BinaryTree class instance.) You will need to use methods for making a new TreeNode, and for setting its left and right children.
Hints: First, identify the array location of the key that would have to be the root of the balanced BST. Now think about how BinarySearch works on the array. Which item does it access first in any given subarray it is called with? Using a similar strategy a simple recursive algorithm is possible.

(b) What is the $\Theta$ cost to construct the tree? Justify your answer.

(c) Compare the expected runtime of BinarySearch on the array to the expected runtime of BST TreeSearch in the tree you just constructed. Have we saved time?