University of Illinois at Urbana-Champaign
Department of Computer Science

Second Examination

CS 225 Data Structures and Software Principles
Sample Exam 2
75 minutes permitted

Print your name, netID, and lab section day/time neatly in the space provided below; print your name at the upper right corner of every page.

Name:
NetID:
Lab Section (Day/Time):

- This is a **closed book** and **closed notes** exam. In addition, you are not allowed to use any electronic aides of any kind.
- Do all 5 problems in this booklet. Read each question very carefully.
- You should have 7 sheets total (the cover sheet, plus numbered pages 1-12). The last sheet is scratch paper; you may detach it while taking the exam, but must turn it in with the exam when you leave.
- Unless otherwise stated in a problem, assume the best possible design of a particular implementation is being used.
- Unless the problem specifically says otherwise, (1) assume the code compiles, and thus any compiler error is an exam typo (though hopefully there are not any typos), and (2) assume you are NOT allowed to write any helper methods to help solve the problem, nor are you allowed to use additional arrays, lists, or other collection data structures unless we have said you can.

Problem	Points	Score	Grader
1	12		
2	30		
3	18		
4	15		
5	15		
Total	90		

1. **[Short Answer – 12 points (4 points each)].**

   (a) What was the "problem" with using path compression and union-by-height together? That is, what difficulty does using the two techniques together present? Please be specific. (The word "problem" is in quotes because we said it turned out that this "problem" didn't actually affect things too badly, even if it seems like it would.)

   (b) If you have a complete tree of 17 nodes, how many nodes are on the deepest level?

   (c) Insert the integers `1` through `6`, in that order, into an AVL tree. Draw the resulting tree. How many rotation operations, total, did you perform? Count a "double rotation" operation as one rotation operation.

2. **[Algorithms – 30 points (6 points each)].**

   (a) Explain why an in-order traversal on a binary search tree should produce the values of the tree in lexigraphical order (i.e. numerical, alphabetical, or whatever the order is that is appropriate for those values).

   (b) In an AVL tree, why does storing the height of a subtree, in the root node of that subtree, improve the efficiency of the AVL rebalancing work (versus not storing the height at all)?

(c) After performing a combine operation during B-Tree removal, why is it that we need to check the parent for underflow? i.e. justify that such a combine operation could have caused the parent to underflow.

(d) Explain why we can implement a complete tree using an array – that is, explain why we don't lose information when we get rid of the pointers, i.e. explain why it is that, given an array, we can always produce the corresponding complete tree.

(e) Explain the "repair case" of the Red-Black Tree removal algorithm (the "repair case" was case 2b, where the node we labelled "x" had a black sibling and that black sibling had a red child in the child position further from "x'). That is, explain what we do in this case and justify that it fixes the problems we have without causing new ones.

3. **[Analysis – 18 points (9 points each)].**

   (a) If you want to remove some value from a min-heap – not necessarily the minimum value, just some random value from the heap – one way you could go about this would be to decrease the priority of the value so that it rises to the top of the heap – i.e. decrease the priority of the value so that it is the minimum value in the heap – and then perform a `DeleteMin` operation. Assuming you already know where the value you want to remove is located in the min-heap, what would be the order of growth of the running time of the above removal procedure? Express your answer in big-$\mathcal{O}$ notation and justify your answer.

(b) Explain why it is that the rebalancing work performed by the AVL tree insert or remove is at most $\mathcal{O}(\lg n)$ on a tree of height $\mathcal{O}(\lg n)$. Your answer should be detailed enough to convince us you know what you are talking about. You don't need to *justify* the steps of the algorithm here – simply indicate what those steps are and their running times – and indicate that those running times add up to what we claim they add up to.

4. **[List to tree – 15 points].**

   You have the following two standard node classes (which are publicly accessible and not encapsulated in another class):

```
class ListNode {
  public:
      int element;
      ListNode* next;
};

class TreeNode {
  public:
      int element;
      TreeNode* left;
      TreeNode* right;
};
```

   Write a function `LevelOrderToTree`. The function should take as parameter a pointer to a `ListNode`, which is the first element of a list that represents the level-order traversal of a *perfect* binary tree. This function should reproduce the binary tree from the level-order listing received as a parameter. That is, `LevelOrderToTree` should return a `TreeNode` pointer which will be the root of a perfect binary tree such that, if a level-order traversal is run on it, it will yield the same listing as the one received as parameter. If the parameter `ListNode` pointer is NULL, the the returned `TreeNode` pointer should also be NULL.

   You have one `Queue` available to you to use as a local variable, if you wish.

```
TreeNode* LevelOrderToTree(ListNode* head) {
   // your code goes here
```

(List to tree, continued)

5. **[Counting Leaves – 15 points].**

   You have the following node class available to you, which is publicly accessible and not encapsulated in another class:

```
class TreeNode {
  public:
      int element;
      TreeNode* left;
      TreeNode* right;
};
```

   Write a function `CountLeaves` that takes as a parameter, a pointer to a `TreeNode`, and returns the number of leaves in the tree whose root is that `TreeNode`. (Hint: Use recursion)

```
int CountLeaves(TreeNode* ptr) {
   // your code goes here
```

(Counting Leaves, continued)

.

(scratch paper)