

**CS 14: Data Structures and Algorithms**

**Monday, November 17, 2003**

**Quiz 6, Form: A**

Name: \_\_\_\_\_

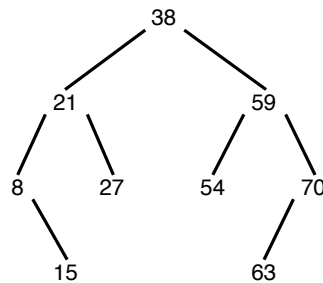
Login: \_\_\_\_\_

ID Number: \_\_\_\_\_

Signature: \_\_\_\_\_

**This quiz is worth 16 points and lasts 12 minutes. Good luck!**

1. [3 points] Given the Binary Search Tree below, write the sequence of its keys resulting from a pre-order traversal of the tree.

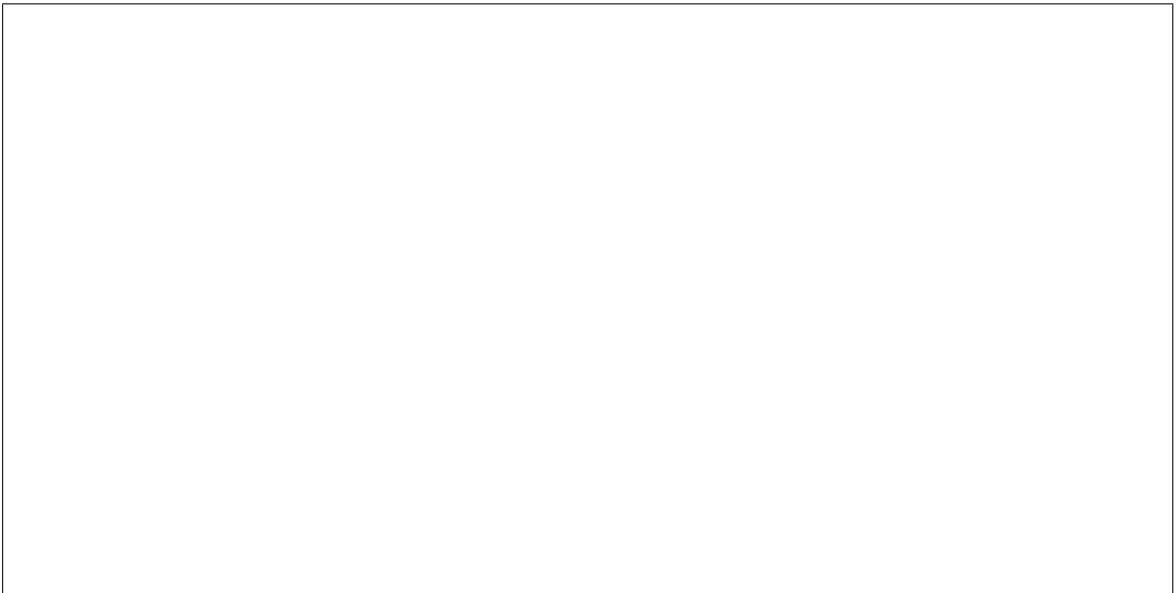


2. [3 points] Write the C++ code for the recursive implementation of the function `void post-PrintKeys(Node * cur)`, which prints (using `cout`) the keys of a Binary Tree, one per line, using post-order traversal. You may assume that the `Binary Tree` class is a friend of the `Node` class (ie, you don't need to call any member functions of the `Node` class).

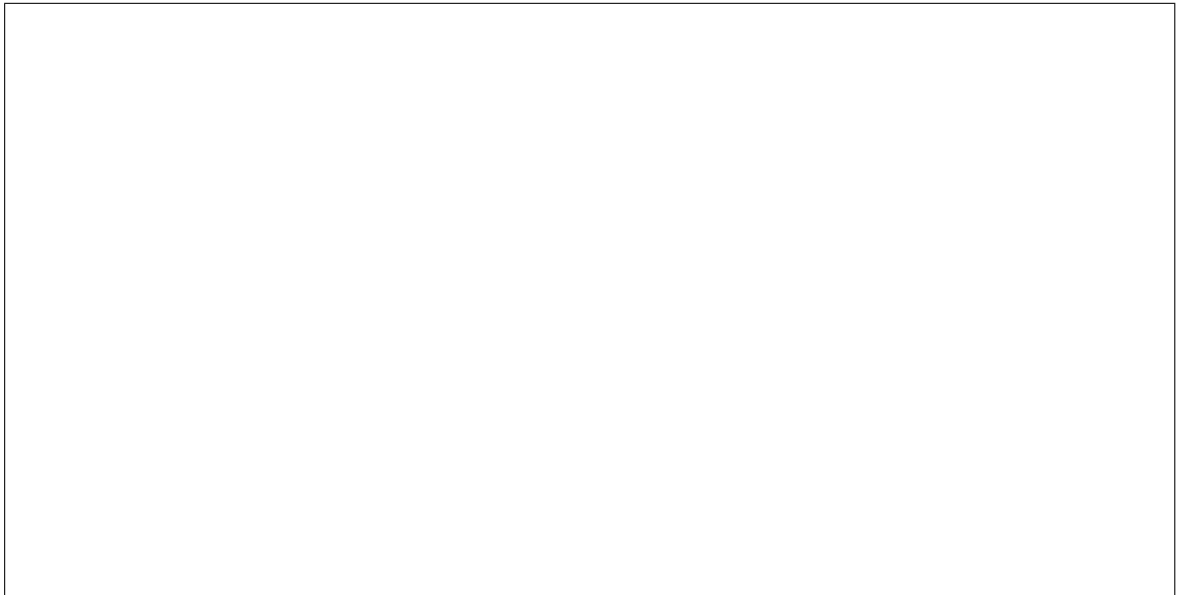
3. [3 points] Draw the Binary Search Tree that results from the insertion, one at a time, of nodes with keys 27, 59, 21, 38, 54, 63, 8, 70, 15, in that order. Draw **only the final result**.



4. [3 points] Starting with the tree from question 1, draw the Binary Search Tree that results from the deletion of the root node, using either one of the two standard methods discussed in lecture. Draw **only the final result** and for **only one of the two methods**.



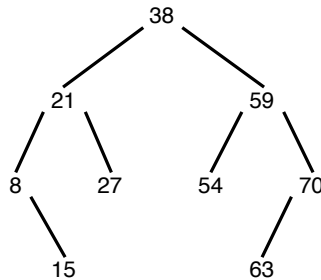
5. [4 points] The Binary Search Tree from question 1 is an AVL tree. The insertion of a node with key 67 violates the AVL balancing condition. Drawing **only the final result**, perform the appropriate rotation (single or double) to re-balance the tree. **Also**, write down what kind of rotation you performed.



# Answer Key for Exam A

This quiz is worth **16 points** and lasts **12 minutes**. Good luck!

1. [3 points] Given the Binary Search Tree below, write the sequence of its keys resulting from a pre-order traversal of the tree.



**Answer:** Recall that in a pre-order traversal, we first process the current node, then its left-child (if any), then its right-child (if any). Thus, the correct result here is: 38, 21, 8, 15, 27, 59, 54, 70, 63.

2. [3 points] Write the C++ code for the recursive implementation of the function `void postPrintKeys(Node * cur)`, which prints (using `cout`) the keys of a Binary Tree, one per line, using post-order traversal. You may assume that the Binary Tree class is a friend of the Node class so that it has direct access to the private member variables of the Node class (ie, you don't need to call any member functions of the Node class).

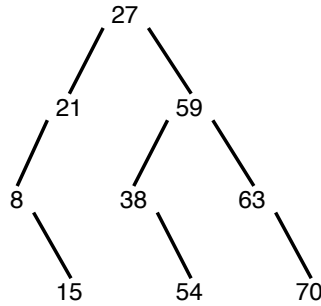
**Answer:**

```
void postPrintKeys(Node * cur)
{
    if (cur == NULL) return;

    postPrintKeys(cur -> left);
    postPrintKeys(cur -> right);
    std::cout << (cur -> key) << std::endl;
}
```

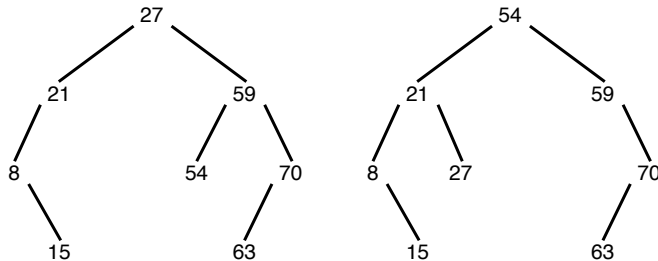
3. [3 points] Draw the Binary Search Tree that results from the insertion, one at a time, of nodes with keys 27, 59, 21, 38, 54, 63, 8, 70, 15, in that order. Draw **only the final result**.

**Answer:**



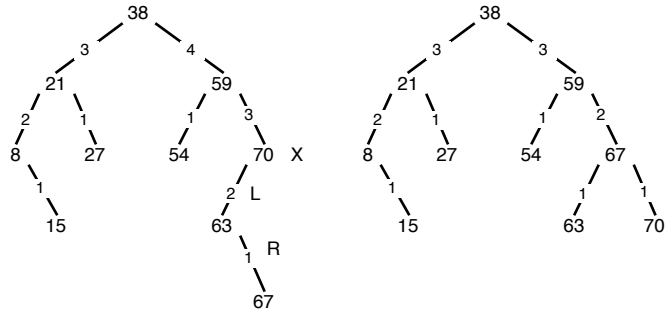
4. [3 points] Starting with the tree from question 1, draw the Binary Search Tree that results from the deletion of the root node, using either one of the two standard methods discussed in lecture. Draw **only the final result** and for **only one of the two methods**.

**Answer:** Recall that deletion of a node which has **both** of its children (as is the case with the root node of the tree given) is accomplished by replacing the node to be deleted with either the node with the **maximum** key from the **left** subtree of the node to be deleted, or the node with the **minimum** key from the **right** subtree of the node to be deleted. Thus, the two standard results here are:

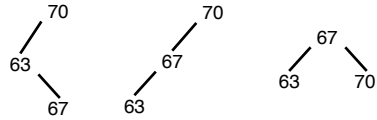


5. [4 points] The Binary Search Tree from question 1 is an AVL tree. The insertion of a node with key 67 violates the AVL balancing condition. Drawing **only the final result**, perform the appropriate rotation (single or double) to re-balance the tree. **Also**, write down what kind of rotation you performed.

**Answer:** In the picture below, the tree on the left is the tree resulting from the insertion, while the tree on the right is the final result after re-balancing. Note the node marked X, which is the first node from the point of insertion up along the path of insertion which violates the balancing condition.



Since the insertion happened in the **R**ight subtree of the **L**eft child of **X**, this is a **LR**-insertion, which requires a **double** rotation to re-balance the tree. The affected part of the tree is shown on the left below.



The middle picture is the result of the first rotation (a single rotation between **X**'s child and **X**'s grandchild), while the second picture is the result of the second rotation (a single rotation between **X** and its new child).