Name: _____

Write all of your responses on these exam pages. If you need extra space please use the backs of the pages.

1. **Short Answer:** (*25 Points*)

   (a) What are the criteria for a Red-Black tree?

   (b) What are the criteria for an AVL tree?

   (c) What are the height bounds for a Red-Black tree?

   (d) What are the height bounds for an AVL tree, assuming all nodes have counts of 1?

   (e) What is the complexity of node insertion into a Red-Black tree?

   (f) What is the complexity of the deletion of a node from a Red-Black tree?

   (g) What is the complexity of insertion into an AVL tree, assuming all nodes have distinct values, counts of 1, and the inserted value is different from the current values stored in the tree?

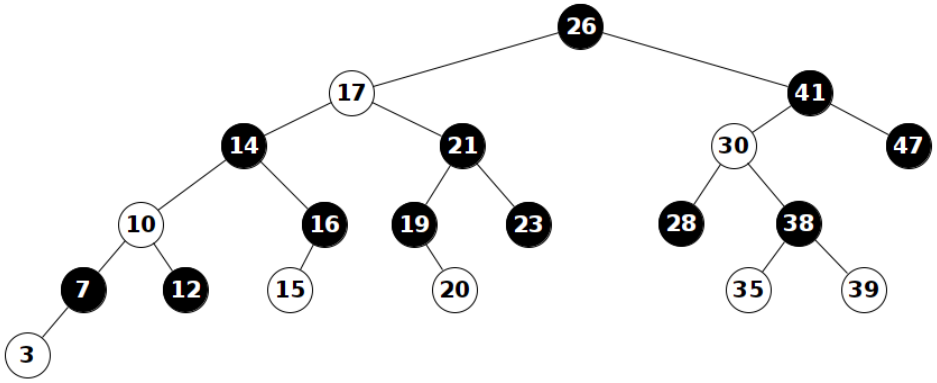
   (h) What is the complexity of deletion of a node from an AVL tree, assuming all nodes have distinct values and all node counts are 1?

   (i) On average, what is the height ratio between a standard BST and an AVL tree, assuming that all node counts are 1?

2. **Red-Black Tree Manipulation:** (*25 Points*)

   Consider the following Red-Black tree. Black nodes have a black background with white numbers and red nodes have a white background with black numbers.

   For each of the following exercises you will start with this tree before the operation is done. Shade all black nodes and use an unshaded circle for the red nodes, or put R or B beside the node value. You may also just display the branch from the root that is effected by the operation and not redraw the unaffected branch. Red-Black tree algorithms are listed at the end of the exam.



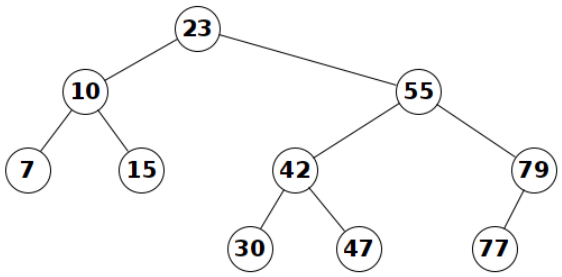
   (a) Insert 36

(b) Delete 17

(c) Delete 41

3. **AVL Tree Manipulation:** (*25 Points*)

   Consider the following AVL tree. For each of the following exercises you will start with this tree before the operation is done. Display the entire tree after the operation is finished. Assume that all node counts are 1.



   (a) Insert 33

   (b) Insert 60

   (c) Delete 55

4. **AVL Tree Code:** (*30 Points*)

   For each of the following write the function for the AVL tree using our implementation. That is, the value is templated and the node is the following private struct.

```
struct TreeNode {
  T value;
  unsigned int count = 1;  // Number of occurrences of the value.
  unsigned int height = 1; //  Height of the subtree below the node.
  TreeNode *left = nullptr;
  TreeNode *right = nullptr;
};
```

   (a) Right Rotate: Does a right rotation at the input node. The function also updates the height values for the nodes that are affected.

```
template <class T> void AVLTree<T>::RightRotation(TreeNode *&nodePtr)
```

(b) Balance: Balances the subtree at just the given node.

```
template <class T> void AVLTree<T>::Balance(TreeNode *&nodePtr)
```

(c) Insert: Inserts newNode into the AVL tree if the value is not already in the tree. If the value is in the tree the count on the node is increased and the new node is released from memory. You may assume that it is called by the following non-recursive insert function.

```
template <class T> void AVLTree<T>::insert(T item) {
  TreeNode *newNode = new TreeNode;
  newNode->value = item;
  insert(root, newNode);
}

template <class T>
void AVLTree<T>::insert(TreeNode *&nodePtr, TreeNode *&newNode)
```

```
LEFT-ROTATE(T, x)
 1  y = x.right
 2  x.right = y.left
 3  if y.left ≠ T.nil
 4       y.left.p = x
 5  y.p = x.p
 6  if x.p == T.nil
 7       T.root = y
 8  elseif x == x.p.left
 9       x.p.left = y
10  else x.p.right = y
11  y.left = x
12  x.p = y
```

```
RB-INSERT(T, z)
 1  x = T.root
 2  y = T.nil
 3  while x ≠ T.nil
 4       y = x
 5       if z.key < x.key
 6            x = x.left
 7       else x = x.right
 8  z.p = y
 9  if y == T.nil
10       T.root = z
11  elseif z.key < y.key
12       y.left = z
13  else y.right = z
14  z.left = T.nil
15  z.right = T.nil
16  z.color = RED
17  RB-INSERT-FIXUP(T, z)
```

```
RB-INSERT-FIXUP(T, z)
 1  while z.p.color == RED
 2       if z.p == z.p.p.left
 3            y = z.p.p.right
 4            if y.color == RED
 5                 z.p.color = BLACK
 6                 y.color = BLACK
 7                 z.p.p.color = RED
 8                 z = z.p.p
 9            else
10                 if z == z.p.right
11                      z = z.p
12                      LEFT-ROTATE(T, z)
13                 z.p.color = BLACK
14                 z.p.p.color = RED
15                 RIGHT-ROTATE(T, z.p.p)
16       else
17            y = z.p.p.left
18            if y.color == RED
19                 z.p.color = BLACK
20                 y.color = BLACK
21                 z.p.p.color = RED
22                 z = z.p.p
23            else
24                 if z == z.p.left
25                      z = z.p
26                      RIGHT-ROTATE(T, z)
27                 z.p.color = BLACK
28                 z.p.p.color = RED
29                 LEFT-ROTATE(T, z.p.p)
30  T.root.color = BLACK
```

```
RB-DELETE(T, z)
 1  y = z
 2  y-original-color = y.color
 3  if z.left == T.nil
 4       x = z.right
 5       RB-TRANSPLANT(T, z, z.right)
 6  elseif z.right == T.nil
 7       x = z.left
 8       RB-TRANSPLANT(T, z, z.left)
 9  else y = TREE-MINIMUM(z.right)
10       y-original-color = y.color
11       x = y.right
12       if y ≠ z.right
13            RB-TRANSPLANT(T, y, y.right)
14            y.right = z.right
15            y.right.p = y
16       else x.p = y
17       RB-TRANSPLANT(T, z, y)
18       y.left = z.left
19       y.left.p = y
20       y.color = z.color
21  if y-original-color == BLACK
22       RB-DELETE-FIXUP(T, x)
```

```
RB-TRANSPLANT(T, u, v)
1  if u.p == T.nil
2       T.root = v
3  elseif u == u.p.left
4       u.p.left = v
5  else u.p.right = v
6  v.p = u.p
```

```
RB-DELETE-FIXUP(T, x)
 1  while x ≠ T.root and x.color == BLACK
 2      if x == x.p.left
 3          w = x.p.right
 4          if w.color == RED
 5              w.color = BLACK
 6              x.p.color = RED
 7              LEFT-ROTATE(T, x.p)
 8              w = x.p.right
 9          if w.left.color == BLACK and w.right.color == BLACK
10              w.color = RED
11              x = x.p
12          else
13              if w.right.color == BLACK
14                  w.left.color = BLACK
15                  w.color = RED
16                  RIGHT-ROTATE(T, w)
17                  w = x.p.right
18              w.color = x.p.color
19              x.p.color = BLACK
20              w.right.color = BLACK
21              LEFT-ROTATE(T, x.p)
22              x = T.root
23      else
24          w = x.p.left
25          if w.color == RED
26              w.color = BLACK
27              x.p.color = RED
28              RIGHT-ROTATE(T, x.p)
29              w = x.p.left
30          if w.right.color == BLACK and w.left.color == BLACK
31              w.color = RED
32              x = x.p
33          else
34              if w.left.color == BLACK
35                  w.right.color = BLACK
36                  w.color = RED
37                  LEFT-ROTATE(T, w)
38                  w = x.p.left
39              w.color = x.p.color
40              x.p.color = BLACK
41              w.left.color = BLACK
42              RIGHT-ROTATE(T, x.p)
43              x = T.root
44  x.color = BLACK
```