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

   (a) What are the criteria for a Red-Black tree?
   Solution:
   i. Every node is either red or black.
   ii. The root is black.
   iii. Every leaf (NIL) is black.
   iv. If a node is red, then both its children are black.
   v. For each node, all simple paths from the node to descendant leaves contain the same number of black nodes.

   (b) What are the criteria for an AVL tree?
   Solution: An AVL tree (originally called an admissible tree) is one in which the height of the left and right subtrees of every node differ by at most one.

   (c) What are the height bounds for a Red-Black tree?
   Solution: $\lg(n+1) \leq h \leq 2\lg(n+1)$

   (d) What are the height bounds for an AVL tree, assuming all nodes have counts of 1?
   Solution: $\lg(n+1) \leq h \leq 1.44\lg(n+2) - 0.328$

   (e) What is the complexity of node insertion into a Red-Black tree?
   Solution: $O(\lg(n))$

   (f) What is the complexity of the deletion of a node from a Red-Black tree?
   Solution: $O(\lg(n))$

   (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?
   Solution: $O(\lg(n))$

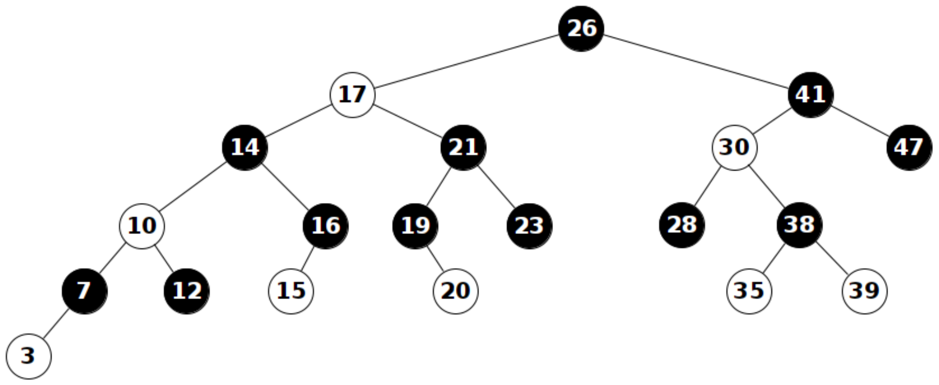
   (h) What is the complexity of deletion of a node from an AVL tree, assuming all nodes have distinct values and all node counts are 1?
   Solution: $O(\lg(n))$

   (i) On average, what is the height ratio between a standard BST and an AVL tree, assuming that all node counts are 1?
   Solution: Approximately 2 to 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

   Solution:

```
                    47 (B)
            41 (R)
                    39 (B)
        38 (B)
                        36 (R)
                    35 (B)
            30 (R)
                    28 (B)
    26 (B)
```

   (b) Delete 17

   Solution:

```
    26 (B)
                    23 (B)
            21 (B)
                    20 (B)
        19 (R)
                    16 (B)
                        15 (R)
            14 (B)
                        12 (B)
                    10 (R)
                        7 (B)
                            3 (R)
```

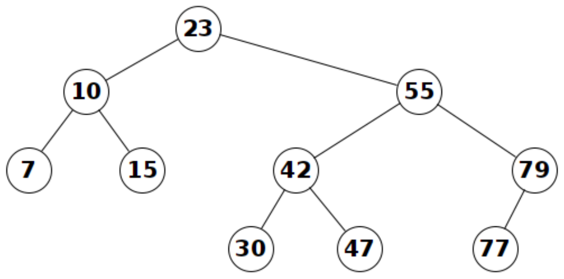
   (c) Delete 41

   Solution:

```
                47 (B)
                    39 (R)
            38 (R)
                35 (B)
        30 (B)
            28 (B)
    26 (B)
```

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

   Solution:

```
                79
                    77
            55
                47
        42
                    33
                30
            23
                    15
                10
                    7
```

   (b) Insert 60

   Solution:

```
                    79
                77
                    60
            55
                    47
                42
                    30
        23
                15
            10
                7
```

   (c) Delete 55

   Solution:

```
                79
                    77
            47
                42
                    30
        23
                15
            10
                7
```

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)
```

   Solution:

```
template <class T> void AVLTree<T>::RightRotation(TreeNode *&nodePtr) {
  TreeNode *L = nodePtr->left;
  TreeNode *temp = L->right;
  L->right = nodePtr;
  nodePtr->left = temp;
  nodePtr->height =
      max(getHeight(nodePtr->left), getHeight(nodePtr->right)) + 1;
  L->height = max(getHeight(L->left), getHeight(L->right)) + 1;
  nodePtr = L;
}
```

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

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

   Solution:

```
template <class T> void AVLTree<T>::Balance(TreeNode *&nodePtr) {
  if (!nodePtr)
    return;

  int balanceFactor = getBalanceFactor(nodePtr);
  if (balanceFactor > 1) {
    if (getBalanceFactor(nodePtr->left) >= 0)
      RightRotation(nodePtr);
    else {
      LeftRotation(nodePtr->left);
      RightRotation(nodePtr);
    }
  }
  if (balanceFactor < -1) {
    if (getBalanceFactor(nodePtr->right) <= 0) {
      LeftRotation(nodePtr);
    } else {
      RightRotation(nodePtr->right);
      LeftRotation(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)
```

Solution:

```
template <class T>
void AVLTree<T>::insert(TreeNode *&nodePtr, TreeNode *&newNode) {
  if (nodePtr == nullptr)
    nodePtr = newNode;
  else if (newNode->value < nodePtr->value)
    insert(nodePtr->left, newNode);
  else if (newNode->value > nodePtr->value)
    insert(nodePtr->right, newNode);
  else {
    nodePtr->count++;
    delete newNode;
  }
  nodePtr->height =
      1 + max(getHeight(nodePtr->left), getHeight(nodePtr->right));
  Balance(nodePtr);
}
```