

Section Solution

Problem 1: Binary Tree Recursion

a.)

```
void IdentifyEndpoints(node *arbitraryNode, node *& front, node *& back)
{
    front = back = arbitraryNode;
    if (front == NULL) return;
    while (front->left != NULL) { front = front->left; }
    while (back->right != NULL) { back = back->right; }
}
```

b.)

```
node *FlattenTree(node *root)
{
    if (root == NULL) return NULL;
    node *leftList = FlattenTree(root->left);
    node *rightList = FlattenTree(root->right);
    node *leftFront, *leftBack, *rightFront, *rightBack;
    IdentifyEndpoints(leftList, leftFront, leftBack);
    IdentifyEndpoints(rightList, rightFront, rightBack);
    root->left = leftBack;
    if (leftBack != NULL) leftBack->right = root; else leftFront = root;
    root->right = rightFront;
    if (rightFront != NULL) rightFront->left = root;
    return leftFront;
}
```

Problem 2: Tree Rotations

a)

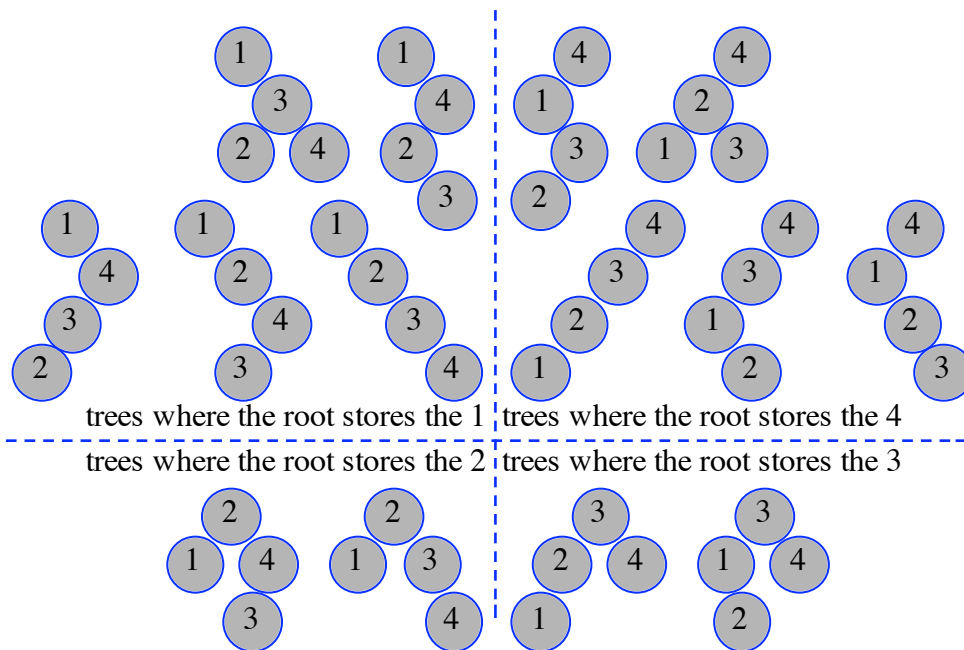
```
void RotateLeft(node **parentp)
{
    node *parent = *parentp;
    node *rightChild = parent->right;
    parent->right = rightChild->left;
    rightChild->left = parent;
    *parentp = rightChild;
}
```

b)

```
void PullToRoot(node **rootp, int value)
{
    node *root = *rootp;
    if ((root == NULL) ||
        (root->value == value)) return; // no rotating to be done
    if (root->value < value) { // value would be in right subtree
        PullToRoot(&(root->right), value);
        LeftRotate(rootp);
    } else {
        PullToRoot(&(root->left), value);
        RightRotate(rootp);
    }
}
```

Problem 3: Binary Trees and Recursion

a.)



b.)

Top Left: 2 3 1 4
 Top Right: 1 3 2 4
 Bottom Left: 1 2 3 4
 Bottom Middle: 2 1 3 4
 Bottom Right: 3 2 1 4

c.)

```
int NumBinarySearchTrees(int numValues)
{
    if (numValues == 0) return 1;
    int count = 0;
    for (int n = 1; n <= numValues; n++)
        count += NumBinarySearchTrees(n - 1) *
                 NumBinarySearchTrees(numValues - n);

    return count;
}
```

Problem 4: Binary Tree Synthesis

a.)

```
treeNode *ListToBinaryTree(listNode *head)
{
    if (head == NULL) return NULL;
    treeNode *root = new treeNode;
    root->value = head->value;
    root->left = ListToBinaryTree(head->next);
    root->right = ListToBinaryTree(head->next);
    return root;
}
```

b.)

```
treeNode *ListToBinaryTree(listNode *head)
{
```

```
treeNode *root;
Queue<treeNode **> children;
children.enqueue(&root);

for (listNode* curr = head; curr != NULL; curr = curr->next) {
    int numChildren = children.size();    // take a snapshot of the size
    for (int i = 0; i < numChildren; i++) {
        treeNode **nodep = children.dequeue();
        *nodep = new treeNode;
        (*nodep)->value = curr->value;
        children.enqueue(&((*nodep)->left));
        children.enqueue(&((*nodep)->right));
    }
}

// everything in Queue points to what needs to be NULLED out
while (!children.isEmpty()) {
    treeNode **nodep = children.dequeue();
    *nodep = NULL;
}

return root;
}
```