Recursion: Sublist Code

```
class ListReferenceBased implements ListInterface {
  private Node head;

  void insert (int index, Object item) {
  ...
  }

}
```

Sublist Code #1

```
void insert (int index, Object item) {
  if (index==1)
      head = new Node(item, head);
  else {
      ListReferenceBased temp
            = new ListReferencedBased(head.getNext());
      temp.insert(index-1, item);
      head.setNext(temp.head);
  }
}
```

Sublist Code #2

```
void insert (int index, Object item) {
  if (index==1)
      head = new Node(item, head);
  else {
      Node temp = head;
      head = head.getNext();
      insert(index-1, item);
      temp.setNext(head);
      head = temp;
  }
}
```

Sublist Code #3

```
void insert (int index, Object item) {
   head = insertRec(head, index, item);
}

private Node insertRec(Node head, int index, Object item) {
   if (index==1)
        head = new Node(item, head);
   else
        head.setNext(insertRec(head.getNext(), index-1, item));
   return head;
}
```

Linked Binary Trees

```
public class LinkedBinaryTree implements BinaryTreeInterface {
   private TreeNode root = null;

   public LinkedBinaryTree (TreeNode ref) {
        root = ref;
   }
   public boolean isEmpty() {
        return (root == null);
   }

   ...

}
```

```
public void attachLeftSubTree (LinkedBinaryTree leftTree)
             throws TreeException {
   if (isEmpty() || root.getLeft() != null)
       throw new TreeException("Empty tree. Bad.");
   root.setLeft(leftTree.root);
   leftTree.makeEmpty();
}
public BinaryTreeInterface detachRightSubtree()
             throws TreeException {
   if (isEmpty())
       throw new TreeException("Empty tree. Stop it.");
   BinaryTreeInterface tree = new
             LinkedBinaryTree(root.getRight();
   root.setRight(null);
   return tree;
}
```

```
public boolean contains (Object item) {
   if (isEmpty())
        return false;
   if (root.getItem().equals(item))
        return true;
   BinaryTreeInterface left = detachLeftSubtree();
   BinaryTreeInterface right = detachRightSubtree();
   boolean result = left.contains(item)||right.contains(item);
   attachLeftSubtree(left);
   attachRightSubtree(right);
   return result;
}
```

```
public Object findRightMost() {
   if (isEmpty())
        return null;
   BinaryTreeInterface right = detachRightSubtree();
   Object result = null;
   if (right.isEmpty())
        result = getRootItem();
   else
        result = right.findRightMost();
   attachRight(right);
   return result;
}
```