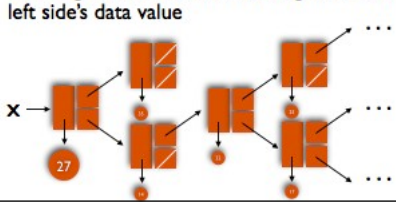
Binary Trees

Definitions

- a binary tree is either the empty binary tree (NIL) or a node with a data value containing an object and left and right values containing binary trees
- an ordered binary tree is a binary tree whose left and right values are ordered binary trees, and whose data value is comparable, less than or equal to its right side's data value, and greater than its left side's data value

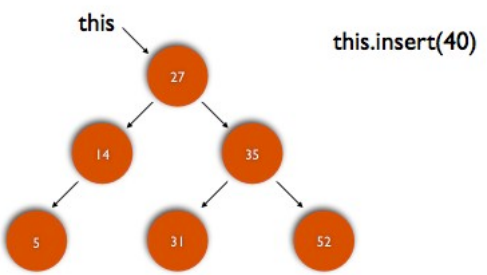


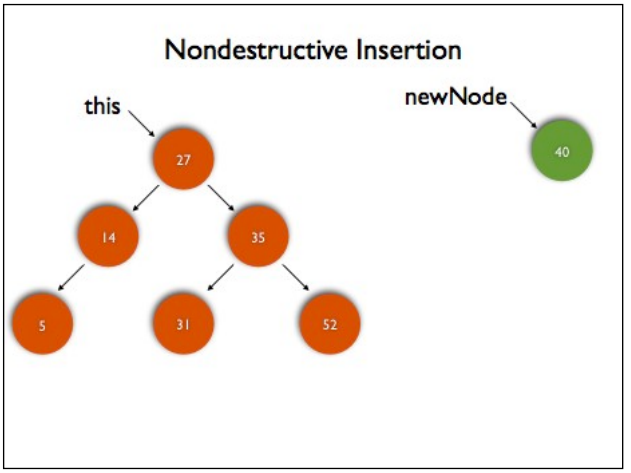

```
pubic class BinaryTree {
    private Comparable data;
    private BinaryTree left;
    private BinaryTree right;
    public final static BinaryTree NIL =
           new BinaryTree(null, null, null);
.
.
.
}
```

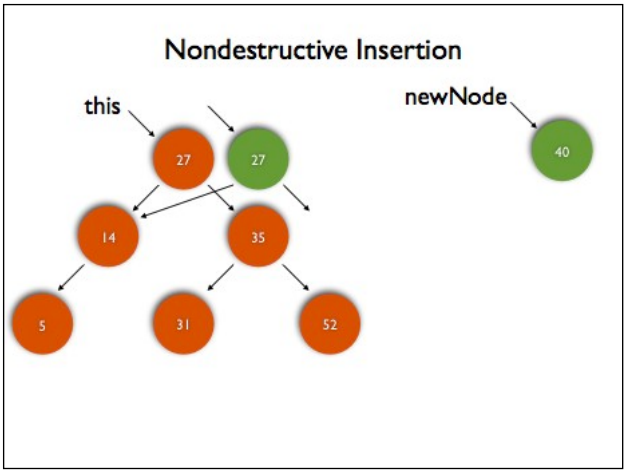
```
pubic class BinaryTree {
.
.
.
    public BinaryTree (Comparable data,
                       BinaryTree left,
                       BinaryTree right) {
        this.data = data;
        this.left = left;
        this.right = right;
    }

    public BinaryTree (Comparable data) {
        this(data, BinaryTree.NIL, BinaryTree.NIL);
    }
.
.
.
}
```

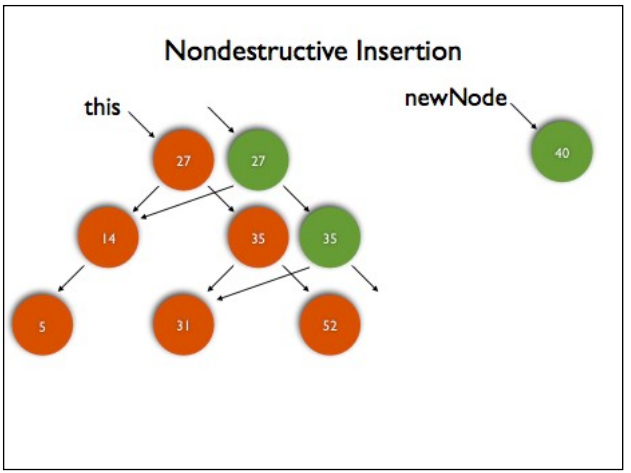
```
pubic class BinaryTree {
.
.
.
    public Comparable data () { return this.data; }
    public BinaryTree left () { return this.left; }
    public BinaryTree right () { return this.right; }

    public void setData (Comparable d) { this.data = d; }
    public void setLeft (BinaryTree t) { this.left = t; }
    public void setRight (BinaryTree t) { this.right = t; }

    public boolean isEmpty () {
        return this == BinaryTree.NIL;
    }
.
.
.
}
```

Nondestructive Insertion

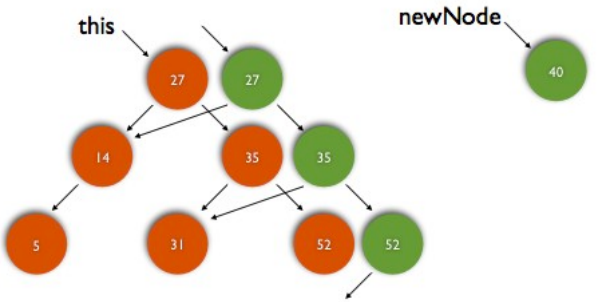



Nondestructive Insertion
this
newNode
27
14
35
40
5
31
52

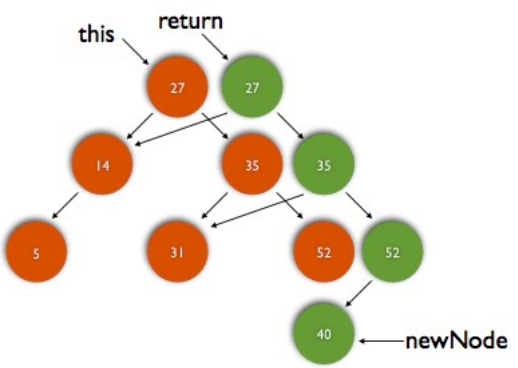

Nondestructive Insertion
this
newNode
27
27
14
35
40
5
31
52


Nondestructive Insertion
this
newNode
27
27
14
35
35
40
5
31
52

Nondestructive Insertion



Nondestructive Insertion



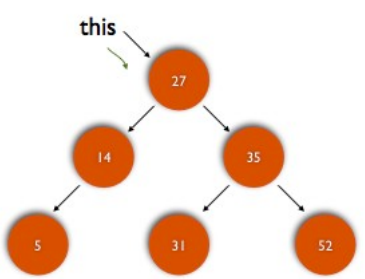

```
pubic class BinaryTree {
.
.
.
    public BinaryTree insert (Comparable d) {
        return this.insert(new BinaryTree(d));
    }

    public BinaryTree insert (BinaryTree newNode) {
        return this.isEmpty()) ? newNode :
               newNode.data().compareTo(this.data()) < 0 ?
            new BinaryTree(this.data(),
                           this.left().insert(newNode),
                           this.right()) :
            new BinaryTree(this.data(),
                           this.left(),
                           this.right().insert(newNode));
    }
.
.
.
}
```

```
pubic class BinaryTree {
.
.
.
    public BinaryTree insertX (Comparable d) {
        return this.insertX(new BinaryTree(d));
    }

    public BinaryTree insertX (BinaryTree newNode) {
        if (this.isEmpty()) return newNode;
        if (newNode.data().compareTo(this.data()) < 0)
            this.setLeft(this.left().insertX(newNode));
        else
            this.setRight(this.right().insertX(newNode));
        return this;
    }
.
.
.
}
```

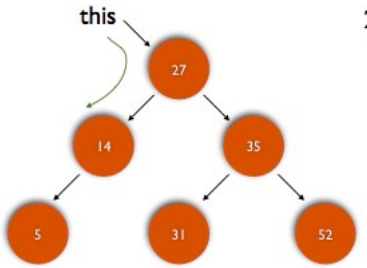
Ordered Binary Tree Traversal

- nodes in a binary tree can be visited in three different orders
- preorder traversal visits nodes as they are "passed on the left"
- inorder traversal visits nodes as they are "passed underneath"
- postorder traversal visits nodes as they are "passed on the right"

Preorder Traversal

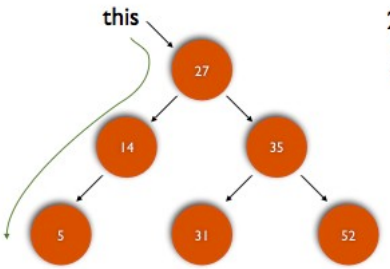


Preorder Traversal



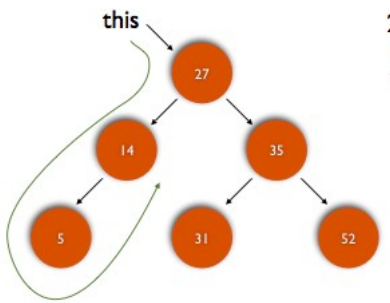
27

Preorder Traversal



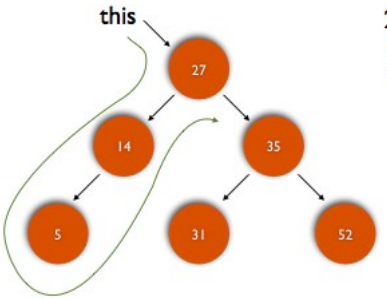
27
14
5

Preorder Traversal



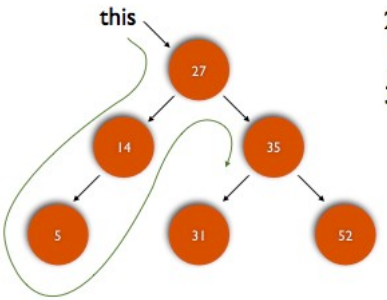
27
14
5

Preorder Traversal



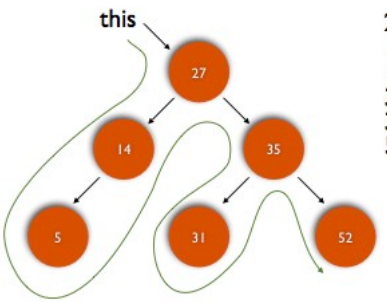
27
14
5

Preorder Traversal



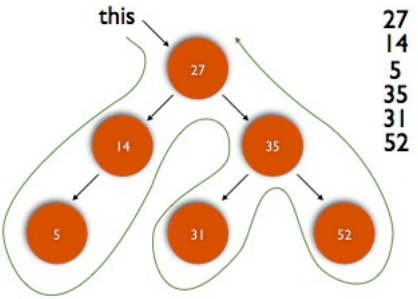
27
14
5
35

Preorder Traversal

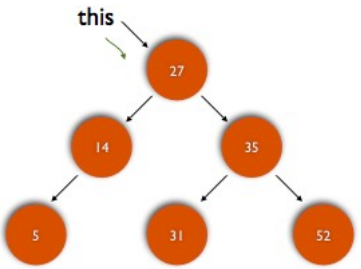


27
14
5
35
31
52

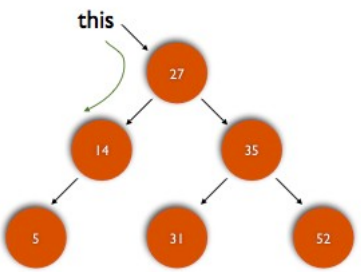
Preorder Traversal



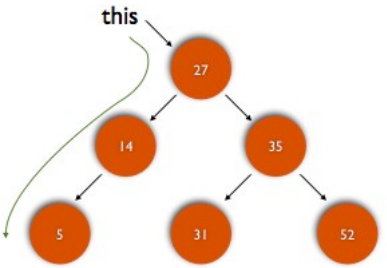
Inorder Traversal



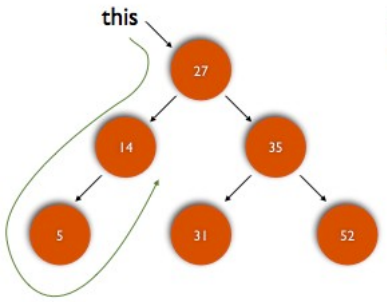
Inorder Traversal



Inorder Traversal

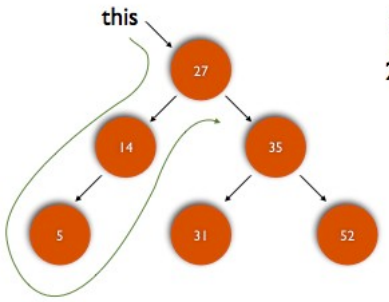


Inorder Traversal



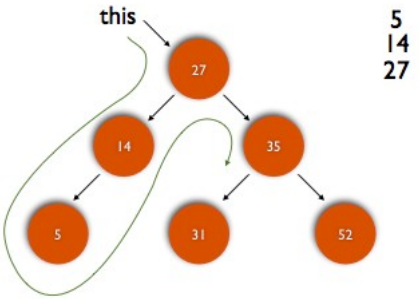
5
14

Inorder Traversal



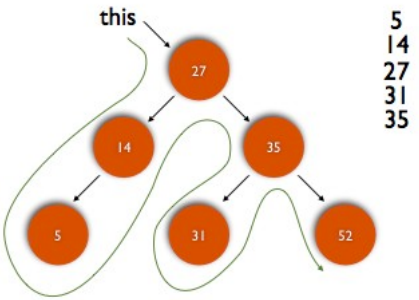
5
14
27

Inorder Traversal



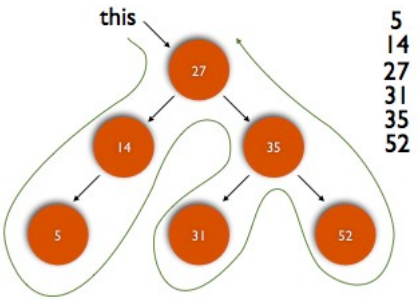
5
14
27

Inorder Traversal



5
14
27
31
35

Inorder Traversal



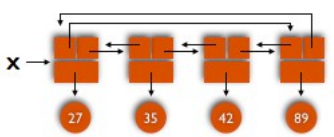
5
14
27
31
35
52

```
pubic class BinaryTree {
.
.
    private String elementsString (String order) {
        if (this.isEmpty()) return "";
        String dataStr = this.data().toString();
        String leftStr = this.left().elementsString(order);
        String rightStr = this.right().elementsString(order);
        return order.equals("PreOrder") ?
                    dataStr + (leftStr != "" ? " " : "") +
                    leftStr + (rightStr != "" ? " " : "") +
                    rightStr :
               order.equals("InOrder") ?
                    leftStr + (leftStr != "" ? " " : "") +
                    dataStr + (rightStr != "" ? " " : "") +
                    rightStr :
               order.equals("PostOrder") ?
                    leftStr + (leftStr != "" ? " " : "") +
                    rightStr + (rightStr != "" ? " " : "") +
                    dataStr :
              "Unknown Ordering: " + order;
    }
.
.
```

```
pubic class BinaryTree {
.
.
.
    public String toString () {
        return this.toString("PreOrder");
    }

    public String toString (String order) {
        return order + "<" + elementsString(order) + ">";
    }

.
.
.
}
```

Doubly Linked Lists

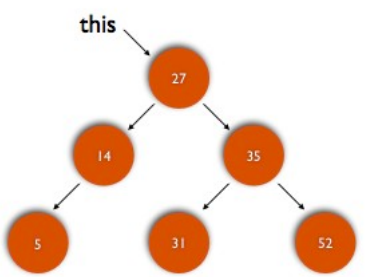
- doubly linked lists are made from nodes with data values and previous (left) and next (right) pointers to adjacent nodes
- thus, doubly linked lists have the same node structure as binary trees



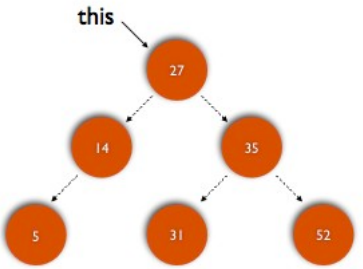
```
pubic class BinaryTree {
.
.
    public BinaryTree insertDL (Comparable d) {
        BinaryTree dl = new BinaryTree(d);
        BinaryTree p;
        if (this.isEmpty()) {
           dl.setLeft(dl); dl.setRight(dl);
           return dl;
        }
        if (d.compareTo(this.data()) < 0) {
           dl.setLeft(this.left); dl.setRight(this);
           this.setLeft(dl); dl.left().setRight(dl);
           return dl;
        }
        for (p = this; p.right() != this &&
                       d.compareTo(p.right().data()) > 0;
             p = p.right()) ;
        dl.setLeft(p); dl.setRight(p.right());
        p.setRight(dl); dl.right().setLeft(dl);
        return this;
    }
.
.
```

```
pubic class BinaryTree {
.
.
.
    public BinaryTree appendDL (BinaryTree that) {
        if (this.isEmpty()) return that;
        if (that.isEmpty()) return this;
        BinaryTree thisEnd = this.left();
        BinaryTree thatEnd = that.left();
        thisEnd.setRight(that);
        that.setLeft(thisEnd);
        thatEnd.setRight(this);
        this.setLeft(thatEnd);
        return this;
    }
.
.
.
}
```

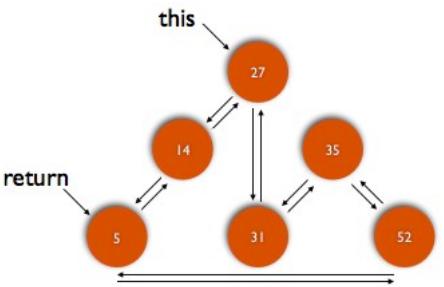
Converting an Ordered Binary Tree to a Doubly Linked List



Converting an Ordered Binary Tree to a Doubly Linked List



Converting an Ordered Binary Tree to a Doubly Linked List




```
pubic class BinaryTree {
.
.
.
    public BinaryTree tree2DL () {
        if (this.isEmpty()) return this;
        BinaryTree a = this.left().tree2DL();
        BinaryTree b = this.right().tree2DL();
        this.setLeft(this);
        this.setRight(this);
        return a.appendDL(this.appendDL(b));
    }
.
.
.
}
```