

Binary Trees

```
public void printPreOrder( )
{
    System.out.println( element );    // Node
    if( left != null ) left.printPreOrder();    // Left
    if( right != null ) right.printPreOrder();    // Right
}
public void printPostOrder( )
{
    if( left != null ) left.printPostOrder();    // Left
    if( right != null ) right.printPostOrder();    // Right
    System.out.println( element );    // Node
}
public void printInOrder( )
{
    if( left != null ) left.printInOrder();    // Left
    System.out.println( element );    // Node
    if( right != null ) right.printInOrder();    // Right
}
```

Binary Trees

```
public void merge( Object rootItem, BinaryTree t1, BinaryTree t2 )
{
    if( t1.root == t2.root && t1.root != null ) {
        System.err.println( "leftTree==rightTree; merge aborted" );
        return;
    }
    root = new BinaryNode( rootItem, t1.root, t2.root );
    if( this != t1 ) t1.root = null;
    if( this != t2 ) t2.root = null;
}
```

```
public BinaryNode duplicate( )
{
    BinaryNode root = new BinaryNode( element, null, null );
    if( left != null ) root.left = left.duplicate( );
    if( right != null ) root.right = right.duplicate( );
    return root;           // Return resulting tree
}
```

Binary Search Trees

```
// BinarySearchTree class
//
// void insert( x )    --> Insert x
// void remove( x )   --> Remove x
// void removeMin( )  --> Remove minimum item
// Comparable find( x ) --> Return item that matches x
// Comparable findMin( ) --> Return smallest item
// Comparable findMax( ) --> Return largest item
// boolean isEmpty( ) --> Return true if empty; else false
// void makeEmpty( )  --> Remove all items
public class BinarySearchTree
{
    private Comparable elementAt( BinaryNode t ) { return t == null ? null : t.element; }
    protected BinaryNode insert( Comparable x, BinaryNode t )
    protected BinaryNode remove( Comparable x, BinaryNode t )
    protected BinaryNode removeMin( BinaryNode t )
    protected BinaryNode findMin( BinaryNode t )
    private BinaryNode findMax( BinaryNode t )
    private BinaryNode find( Comparable x, BinaryNode t )

    protected BinaryNode root;
}
```

Binary Search Trees

```
public static void main( String [ ] args ) {
    BinarySearchTree t = new BinarySearchTree( );
    final int NUMS = 4000;
    final int GAP = 37;
    System.out.println( "Checking... (no more output means success)" );
    for( int i = GAP; i != 0; i = ( i + GAP ) % NUMS )
        t.insert( new Integer( i ) );
    for( int i = 1; i < NUMS; i+= 2 )
        t.remove( new Integer( i ) );
    if( ((Integer)(t.findMin( ))).intValue( ) != 2 ||
        ((Integer)(t.findMax( ))).intValue( ) != NUMS - 2 )
        System.out.println( "FindMin or FindMax error!" );
    for( int i = 2; i < NUMS; i+=2 )
        if( ((Integer)(t.find( new Integer( i ) ))).intValue( ) != i )
            System.out.println( "Find error1!" );
    for( int i = 1; i < NUMS; i+=2 )
    {
        if( t.find( new Integer( i ) ) != null )
            System.out.println( "Find error2!" );
    }
}
```

Binary Search Trees

```
protected BinaryNode insert( Comparable x, BinaryNode t ) {
    if( t == null )
        t = new BinaryNode( x );
    else if( x.compareTo( t.element ) < 0 )
        t.left = insert( x, t.left );
    else if( x.compareTo( t.element ) > 0 )
        t.right = insert( x, t.right );
    else throw new DuplicateItemException( x.toString() ); // Duplicate
    return t;
}

protected BinaryNode remove( Comparable x, BinaryNode t ) {
    if( t == null ) throw new ItemNotFoundException( x.toString() );
    if( x.compareTo( t.element ) < 0 ) t.left = remove( x, t.left );
    else if( x.compareTo( t.element ) > 0 ) t.right = remove( x, t.right );
    else if( t.left != null && t.right != null ) {
        t.element = findMin( t.right ).element;
        t.right = removeMin( t.right );
    }
    else
        t = ( t.left != null ) ? t.left : t.right;
    return t;
}
```

Binary Search Trees

```
protected BinaryNode removeMin( BinaryNode t )
{
    if( t == null )
        throw new ItemNotFoundException( );
    else if( t.left != null )
    {
        t.left = removeMin( t.left );
        return t;
    }
    else
        return t.right;
}
```

```
protected BinaryNode findMin( BinaryNode t )
{
    if( t != null )
        while( t.left != null )
            t = t.left;
    return t;
}
```

```
private BinaryNode
    find( Comparable x, BinaryNode t )
{
    while( t != null )
    {
        if( x.compareTo( t.element ) < 0 )
            t = t.left;
        else if( x.compareTo( t.element ) > 0 )
            t = t.right;
        else
            return t;    // Match
    }
    return null;    // Not found
}
```