Generic Binary Search Trees

From Textbook

```
public class BinarySearchTree<AnyType extends
Comparable<? super AnyType>>
{
stuff
}
```

```
private static class BinaryNode<AnyType> {

   BinaryNode( AnyType theElement ) {
      this( theElement, null, null );
   }

   BinaryNode( AnyType theElement, BinaryNode<AnyType> lt,
                        BinaryNode<AnyType> rt ) {
   element = theElement; left = lt; right = rt;
   }
```

```
    AnyType element;
    BinaryNode<AnyType> left;
    BinaryNode<AnyType> right;

}
```

```
private BinaryNode<AnyType> root;
public BinarySearchTree( ) { root = null; }
```

```
public boolean contains( AnyType x ) {
   return contains( x, root );
}
```

```
private boolean contains( AnyType x, BinaryNode<AnyType> t
) {

if( t == null ) return false;
int compareResult = x.compareTo( t.element );
if( compareResult < 0 )
        return contains( x, t.left );
else if( compareResult > 0 )
        return contains( x, t.right );
else
        return true;
}
```

```
public void insert( AnyType x )
{
    root = insert( x, root );
}


public void remove( AnyType x )
{
    root = remove( x, root );
}
```

```
private BinaryNode<AnyType> insert( AnyType x,
BinaryNode<AnyType> t ) {
        if( t == null )
            return new BinaryNode<AnyType>( x, null, null );
        int compareResult = x.compareTo( t.element );
        if( compareResult < 0 )
            t.left = insert( x, t.left );
        else if( compareResult > 0 )
            t.right = insert( x, t.right );
        else
            ;  // Duplicate; do nothing
        return t;
}
```

```
private BinaryNode<AnyType> findMin(
                    BinaryNode<AnyType> t )
  {
      if( t == null )
          return null;
      else if( t.left == null )
          return t;
      return findMin( t.left );
  }
```

```
private BinaryNode<AnyType> remove( AnyType x,
BinaryNode<AnyType> t )
    {
        if( t == null )
            return t;   // Item not found; do nothing
        int compareResult = x.compareTo( t.element );
```

```
    if( compareResult < 0 )
        t.left = remove( x, t.left );
    else if( compareResult > 0 )
        t.right = remove( x, t.right );
    else if( t.left != null && t.right != null ) // Two children
    {
        t.element = findMin( t.right ).element;
        t.right = remove( t.element, t.right );
    }
    else
        t = ( t.left != null ) ? t.left : t.right;
    return t;
}
```

```
public static void main( String [ ] args )
{
    BinarySearchTree<Integer> t = new
                        BinarySearchTree<Integer>( );
    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( i );
    for( int i = 1; i < NUMS; i+= 2 )
        t.remove( i );
    for( int i = 2; i < NUMS; i+=2 )
         if( !t.contains( i ) )
             System.out.println( "Find error1!" );
    for( int i = 1; i < NUMS; i+=2 )
    {
        if( t.contains( i ) )
            System.out.println( "Find error2!" );
    }
}
```