Esercizio su alberi binari

```
class Node  {
   public Comparable data;
   public Node left;
   public Node right;
   /** Inserts a new node as a descendant of this node.
   @param newNode the node to insert */
   public void addNode(Node newNode)
   {
        int comp = newNode.data.compareTo(data);
        if (comp < 0)
        {
        if (left == null) left = newNode;
        else left.addNode(newNode);
        }
        else if (comp > 0)
               { if (right == null) right = newNode;
                 else right.addNode(newNode);
               }
   }
```

Esercizio su alberi binari

```
/** This class implements a binary search tree whose nodes hold
   objects that implement the Comparable interface. */

public class BinarySearchTree
{
   private Node root;

   /**Constructs an empty tree. */
   public BinarySearchTree()
   {   root = null;  }

   /** Inserts a new node into the tree.
   @param obj the object to insert  */
   public void add(Comparable obj)
   {
        Node newNode = new Node();
        newNode.data = obj;
        newNode.left = null;
        newNode.right = null;
        if (root == null) root = newNode;
        else root.addNode(newNode);
}
```

Esercizio su alberi binari 2

```
/** Tries to find an object in the tree.
@param obj the object to find
@return true if the object is contained in the tree */

public boolean find(Comparable obj)
{
   Node current = root;
   while (current != null)
        {
        int d = current.data.compareTo(obj);
        if (d == 0) return true;
        else if (d > 0) current = current.left;
                else current = current.right;
        }
   return false;
}
```

Esercizio su alberi binari 3.1

```
/**  Tries to remove an object from the tree.
Does nothing if the object is not contained in the tree.
@param obj the object to remove */

   public void remove(Comparable obj)
   {
   // Find node to be removed
        Node toBeRemoved = root;
        Node parent = null;
        boolean found = false;
        while (!found && toBeRemoved != null)
                { int d = toBeRemoved.data.compareTo(obj);
                if (d == 0) found = true;
                else
                { parent = toBeRemoved;
                if (d > 0) toBeRemoved = toBeRemoved.left;
                else toBeRemoved = toBeRemoved.right;
                }
                }
   if (!found) return;
```

Esercizio su alberi binari 3.2

```
// toBeRemoved contains obj
// If one of the children is empty, use the other one

if (toBeRemoved.left == null ||
                        toBeRemoved.right == null)
   {
        Node newChild;
        if (toBeRemoved.left == null)
               newChild = toBeRemoved.right;
        else   newChild = toBeRemoved.left;
        if (parent == null)         // Found in root
               root = newChild;
        else   if (parent.left == toBeRemoved)
               parent.left = newChild;
               else
               parent.right = newChild;
        return;   //exit method remove
   }
// Neither subtree is empty
```

Esercizio su alberi binari 3.3

```
// Find smallest element of the right subtree

   Node smallestParent = toBeRemoved;
   Node smallest = toBeRemoved.right;
   while (smallest.left != null)
   {
        smallestParent = smallest;
        smallest = smallest.left;
   }

// smallest contains smallest child in right subtree
// Move its contents and unlink child

   toBeRemoved.data = smallest.data;
   if (smallestParent == toBeRemoved)
        smallestParent.right = smallest.right;
   else
        smallestParent.left = smallest.right;
}  //end of remove
```