

```
class Node // Node for binary search tree
{
    int value;
    Node left;
    Node right;

    Node (int x)
    {
        value = x;
    }

    void print() // print the information about the treenode
    {
        System.out.print(this);
        System.out.print(": value: " + value);
        System.out.print(" left: " + left);
        System.out.print(" right: " + right);
        System.out.println();
    }
}
```

```
class Tree // binary search tree
{
    Node root = null;

    Node find(int x)
        // find value x in the tree, return node containing it
        // if not present, return null
    {
        Node p = root;
        while (p != null && p.value != x)
            if (x < p.value)
                p = p.left;
            else
                p = p.right;
        return p;
    }
}
```

```

Node findinsert(int x)
    // insert a new element if not present
    // return pointer to it
{
    Node p = root, parent = null;
    while (p != null && p.value != x)
    {
        parent = p;
        if (x < p.value)
            p = p.left;
        else
            p = p.right;
    }
    if (p != null) // item was found, end
        return p;

    p = new Node(x); // to be inserted
    if (root == null) // have to change the root explicitly
        root = p;
    else
        if (x < parent.value)
            parent.left = p;
        else
            parent.right = p;
    return p;
}

```

```
int min() // Tree minimum, tree must be nonempty
{
    Node p = root;
    while (p.left != null)
        p = p.left;
    return p.value;
}
```

```
int max() // Tree maximum, tree must be nonempty
{
    Node p = root;
    while ( p.right != null)
        p = p.right;
    return p.value;
}
```

```
void print() // print tree in order
{
    recprint(root);
    System.out.println();
}

void recprint(Node p)
    // recursively print the tree values in sorted order
{
    if (p == null)
        return;
    recprint (p.left);
    System.out.print(p.value + " ");
    recprint (p.right);
}

void structureprint()
    // print the internal structure of the tree
{
    internalprint(root, 0);
}
```

```
void internalprint(Node p, int depth) // recursion depth
{
    if (p == null)
        return;
    for (int k=0; k < depth; k++)
        System.out.print("  ");
    p.print();
    internalprint(p.left, depth+1);
    internalprint(p.right, depth+1);
}
```

```
public static void main (String[] args)
{
    Tree T = new Tree();
    for (int i=0; i < args.length; i++)
        T.findinsert(Integer.parseInt(args[i]));
    T.print();
    T.structureprint();
    System.out.println("Minimum: "+ T.min());
    System.out.println("Maximum: "+ T.max());
}
```

```
}
```

```
/* output:
```

```
$ java Tree 5 7 8 2 1 9 4 6 3
```

```
1 2 3 4 5 6 7 8 9
```

```
Node@8269298: value: 5 left: Node@8269418 right: Node@82692d8
```

```
Node@8269418: value: 2 left: Node@82692b8 right: Node@82695b8
```

```
Node@82692b8: value: 1 left: null right: null
```

```
Node@82695b8: value: 4 left: Node@82695f8 right: null
```

```
Node@82695f8: value: 3 left: null right: null
```

```
Node@82692d8: value: 7 left: Node@82695d8 right: Node@82693f8
```

```
Node@82695d8: value: 6 left: null right: null
```

```
Node@82693f8: value: 8 left: null right: Node@8269598
```

```
Node@8269598: value: 9 left: null right: null
```

```
Minimum: 1
```

```
Maximum: 9
```

```
*/
```

