

Java 1998 AP Computer Science Question AB4

Assume that binary trees are implemented using the following declarations.

```
public class TreeNode
{
    public TreeNode(Object initialValue, TreeNode initLeft, TreeNode initRight)
    {
        // not shown
    }

    public Object getValue() { // not shown }
    public TreeNode getLeft() { // not shown }
    public TreeNode getRight() { // not shown }

    public void setValue(Object theNewValue) { // not shown }
    public void setLeft(TreeNode theNewLeft) { // not shown }
    public void setRight(TreeNode theNewRight) { // not shown }
};
```

You may find the following method useful in writing solutions for this problem.

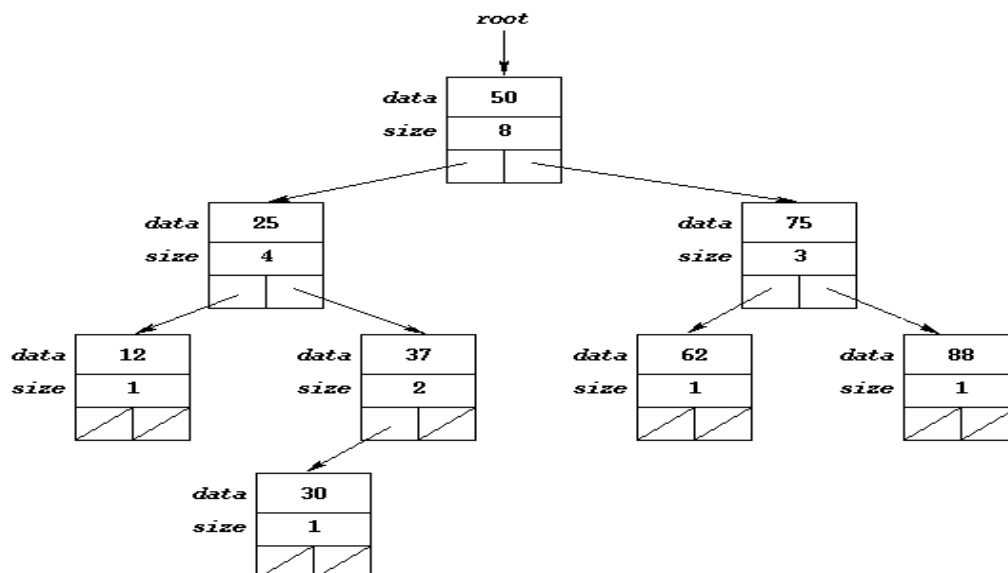
```
// returns number of nodes in t

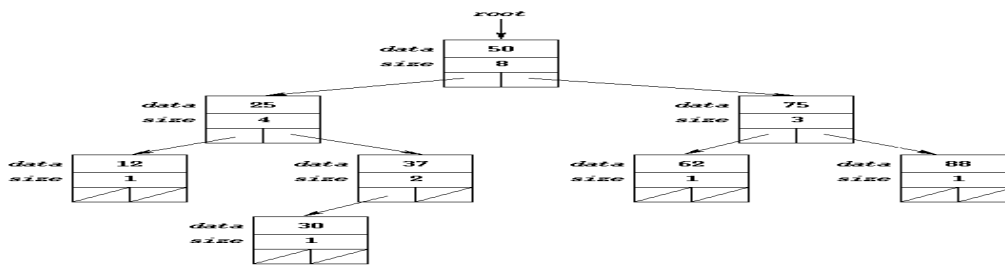
public int nodeCount(TreeNode t)
{
    if (t == null) return 0;
    return 1 + nodeCount(t.getLeft()) + nodeCount(t.getRight());
}
```

Part A

Write method *setSize*, whose header is given below. *setSize* should replace the value stored in each node of the tree with an `Integer` object whose value is the number of nodes in that node's subtree (including the node itself). The number of nodes in a tree is equal to the number of nodes in its left subtree plus the number of nodes in its right subtree plus one.

For example, the following picture shows the result of the call *setSize(root)*. The values labelled *data* could represent the values stored before calling *setSize*, the values labelled *size* are the values stored in each node after *setSize* has completed.





Complete function *setSize* below.

```

public class AB4
{
    // precondition: t != null
    // postcondition: The value of every node in the tree t
    //                has been set to an Integer whose value is the
    //                the size of that node's subtree

    public void setSize(TreeNode t)
    {

    }
}
  
```

[answer](#)

Part B

Assume that tree *t* is a binary search tree ordered by the node's values, and that there are no duplicate values. The *k*th value in a binary search tree is the *k*th smallest value in the tree. For example, the tree shown in part (a) includes the data values 12, 25, 30, 37, 50, 62, 75, 88.

The 1st value is 12.
 The 4th value is 37.
 The 7th value is 75.
 The 8th value is 88.

Write method *findKth*, whose header is given below. *findKth* returns the *k*th value in a binary search tree. Assume that the *size* fields in all nodes of the tree have been correctly initialized. One way to determine the location of the *k*th value is as follows:

Consider the size of the left subtree of the current node
 If *k* is equal to the size of the left subtree + 1, the *k*th value is in the current node.
 If *k* is less than the size of the left subtree + 1, the *k*th value is in the left subtree.
 Otherwise, the *k*th value is in the right subtree.

Complete function *kthValue* below.

```

Object kindKth(TreeNode * t, int k)
// precondition: t is not NULL, the size fields of all nodes in t
//               are correctly initialized.
//               1 <= k <= t->size
// postcondition: returns the kth value in t
  
```

```
{
```

```
}
```

[answer](#)

[Owen L. Astrachan](#)

Last modified: Wed Jul 2 17:14:11 EDT 2003