CSC212

Tutorial 9

Binary Search Trees

Problem 1:

Write the recursive method ***maxKey*** member of the class ***BST*** that returns the maximum key value in the tree. Assume that the tree is not empty.

Method: *public int maxKey()*

Solution:

```
public int maxKey() {
        return maxKey(root);
}
private int maxKey(BSTNode<T> n) {
        if(n.right == null)
                return n.key;
        else
                return maxKey(n.right);
}
```

Problem 2:

Write the recursive method ***sumKeys*** member of the class ***BST*** that returns the sum of all the keys.

Method: *public int sumKeys()*

Solution:

```
public int sumKeys() {
        return sumKeys(root);
}

private int sumKeys(BSTNode<T> n) {
        if(n == null)
                return 0;
        else
                return n.key + sumKeys(n.left) + sumKeys(n.right);
}
```

Problem 3:

Write the method ***range*** member of the class **BST** that returns the range of the binary search tree. The rage is defined as the difference between the maximum key and the minimum key. Assume that the tree is not empty.

Method: *public int range()*

Solution:

```
public int range() {
        BSTNode<T> temp = root;
        int min, max;
        while(temp.left != null)
                temp = temp.left;
        min = temp.key;
        temp = root;
        while(temp.right != null)
                temp = temp.right;
        max = temp.key;
        return max - min;
}
```

Problem 4:

Write an efficient method ***inRange***, member of the class ***BST***, that takes as input a key ***k*** and returns true if the binary search tree contains at least two keys ***k1*** and ***k2*** such that $k1 \leq k \leq k2$, false otherwise. Try to minimize the number of visited nodes.

Method: *public boolean inRange(int k)*

Solution:

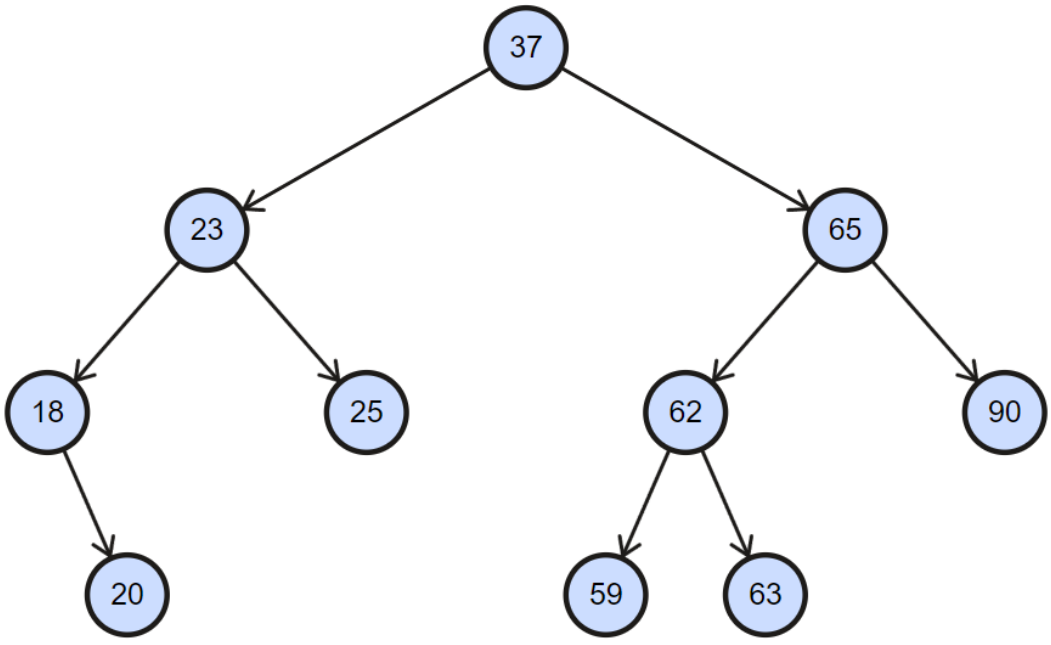
```
// Solution #1
public boolean inRange(int k) {
        if(root == null)
                return false;
        BSTNode<T> temp = root;
        int k1, k2;
        while(temp.left != null)
                temp = temp.left;
        k1 = temp.key;
        temp = root;
        while(temp.right != null)
                temp = temp.right;
        k2 = temp.key;
        return k1 <= k && k <= k2;
}
```

```
// Solution #2 (Faster)
public boolean inRange(int k) {
        if(root == null)
                return false;
        BSTNode<T> temp = root;
        while(temp.left != null) {
                if(temp.key <= k)
                        break;
                temp = temp.left;
        }
        if(temp.key > k)
                return false;
        temp = root;
        while(temp.right != null) {
                if(k <= temp.key)
                        break;
                temp = temp.right;
        }
        if(k > temp.key)
                return false;
        return true;
}
```

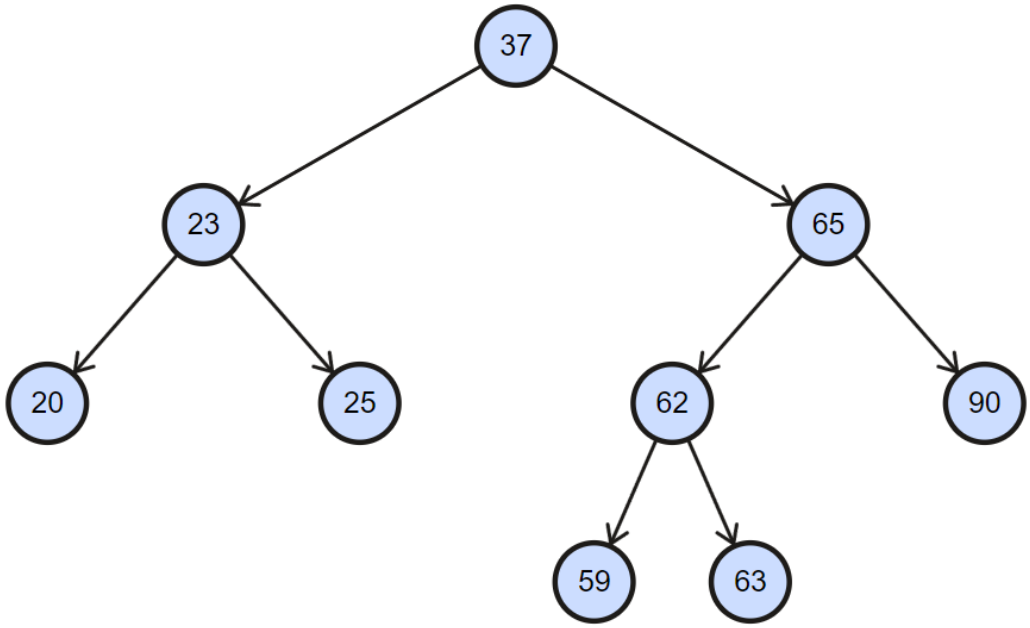
Problem 5:

a) Insert the following keys into an empty binary search tree: 37, 23, 18, 65, 25, 62, 20, 59, 63, 90, 18.
b) Remove the following keys from the final tree in part a: 18, 90, 37.
c) If we wish to print the keys in increasing order, then which traversal method should we use?

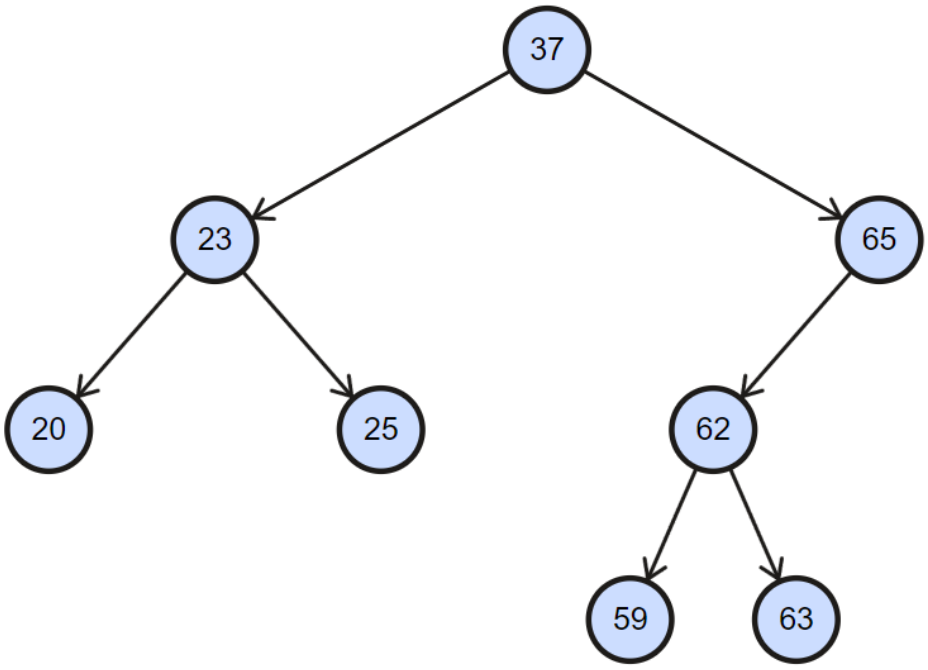
a)



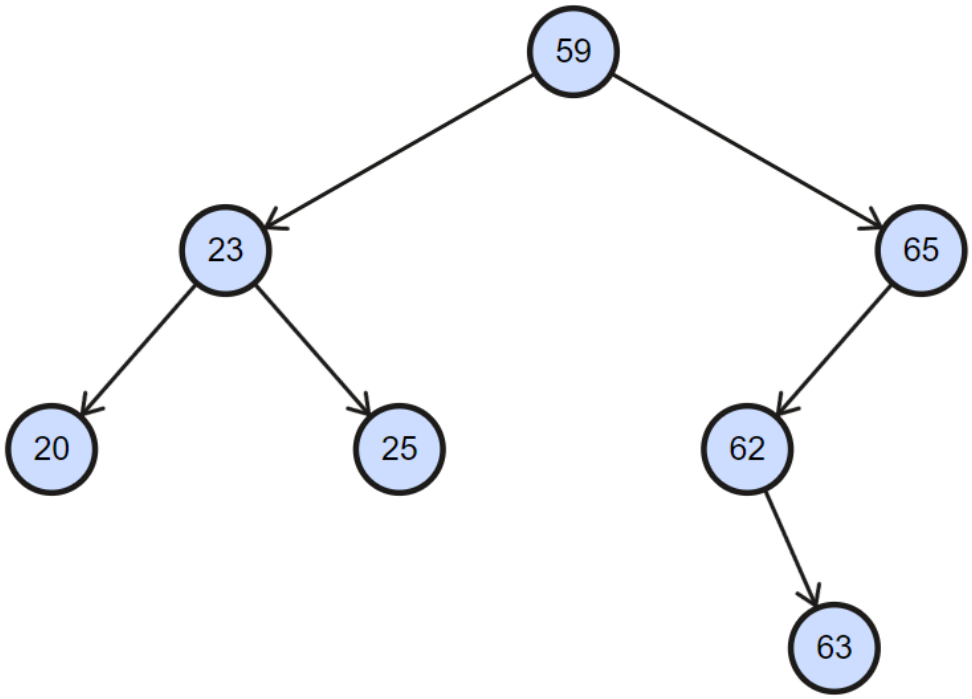
b) removing 18 (case 2: having one child)



removing 90 (case 1: having no children)



removing 37 (case 3: having two children; left-most node in the right sub-tree)



c) Inorder (left, root, right)