Binary Search Tree

Ex.1

```
package A7BinarySearchTree;

public class A1BinarySearchTree {
	static class Node{
	   int data;
	   Node left;
	   Node right;
	   Node(int data){
		   this.data=data;
		   this.left=null;
		   this.right=null;
	   }
	}

	//insert time complexity: O(height)
	public static Node insert(Node root, int val) {
	   if(root == null) {
		   root = new Node(val);
		   return root;
	   }
	   if(root.data>val) {
		   root.left=insert(root.left,val);
	   }
	   else {
		   root.right=insert(root.right,val);
	   }

	   return root;
	}


	public static void inorder(Node root) {
	   if(root==null) {
		   return;
	   }
	   inorder(root.left);
	   System.out.print(root.data+" "); //Ans:1 2 3 4 5 7
	   inorder(root.right);
	}

	//Time Complexity: O(H)   H means Height of tree
	public static boolean search(Node root, int target) {
	   if(root==null) { //base condition when target not found in tree
		   return false;
	   }
	   if(target==root.data) {
		   return true;
	   }
	   if(target<root.data) {
		   return search(root.left,target);
	   }
	   else {
		   return search(root.right,target);
	   }
```

```
	}


	public static Node inorderSuccessor(Node root){
	  //MyAns:
//	  if(root.left==null) {
//		return root;
//	  }
//	  return inorderSuccessor(root.left);

	  //Apna Clg Ans
	  while(root.left!=null) {
		root=root.left;
	  }
	  return root;
	}

	public static Node delete(Node root, int val) { //here val is value which we
want to delete
	  if(root.data==val) {
		//case 1 when delete node is leaf node
		 if(root.left==null && root.right==null) {
			return null;
		 }

		//case 2 when delete node has one child
		 if(root.left==null) {
			return root.right;
		 }else if(root.right==null) {
			return root.left;
		 }

		//case 3 when delete node has 2 children

			Node IS = inorderSuccessor(root.right);
			root.data=IS.data;
			root.right = delete(root.right,IS.data);  //here we go into the
right tree and where we got
		       //inorder successor data we delete that and return all right tree
without that inorder successor



	  }
	  if(root.data>val) {
		root.left =  delete(root.left,val);
	  }
	  if(root.data<val) {
		 root.right = delete(root.right,val);
	  }
	  return root;
	}

	public static void main(String[] args) {
		int values[] = {5,1,3,4,2,7};
```

```
        Node root = null;
        for(int i=0;i<values.length;i++) {
             root = insert(root,values[i]);
        }
        System.out.println("Root: "+root.data);//Ans:5
        inorder(root);//Ans: 1 2 3 4 5 7
        System.out.println();
        System.out.println("Is target found: "+search(root,2)); //Ans:true
        delete(root,2);
        inorder(root); //Ans:1 3 4 5 7
        System.out.println();
        delete(root,3);
        inorder(root); //Ans: 1 4 5 7
      }

}
```

Ex.2

```
package A7BinarySearchTree;
import java.util.*;
public class A2PrintInRange {

      static class Node{
            int data;
            Node left;
            Node right;
            Node(int data){
                  this.data=data;
                  this.left=null;
                  this.right=null;
            }
      }

      public static Node insert(Node root,int val) {
            if(root==null) {
                  root = new Node(val);
                  return root;
            }
            if(val<root.data) {
                  root.left=insert(root.left,val);
            }else if(val>root.data) {
                  root.right= insert(root.right,val);
            }
            return root;
      }

      public static void inorder(Node root) {
            if(root==null) {
                  return;
            }
            inorder(root.left);
```

```
		System.out.print(root.data+" ");
		inorder(root.right);
	}

	//Que. Print the value between given range 6 and 10 (including 6 and 10)
	 public static void printValueGivenRange(Node root,int x,int y) {
		if(root==null) {
			return;
		}
		if(root.data>=x && root.data<=y) {
			printValueGivenRange(root.left,x,y);
			System.out.print(root.data+" ");
			printValueGivenRange(root.right,x,y);
		}
		if(root.data<x) {
			printValueGivenRange(root.right,x,y);
		}
		if(root.data>y) {
			printValueGivenRange(root.left,x,y);
		}

	 }

	 //Que. Print all the path from root to leaf
	 public static void printRootToLeaf(Node root,ArrayList<Integer> list) {
		  if(root==null) {
			  return;
		  }

		  list.add(root.data);

		if(root.left==null && root.right==null) { //if leaf node
			System.out.println("Path:"+list);

		}else { //if not leaf node
			printRootToLeaf(root.left,list);
			printRootToLeaf(root.right,list);
		}
		list.remove(list.size()-1);
	 }

	public static void main(String[] args) {

		int values[]= {8,5,3,1,4,6,10,11,14};
		Node root=null;
		for(int i=0;i<values.length;i++) {
			root = insert(root,values[i]);
		}
		System.out.println("Root: "+root.data);
	 inorder(root); //Ans:1 3 4 5 6 8 10 11 14
	 System.out.println();
	 int x=6;
	 int y=10;
	 printValueGivenRange(root,x,y); //Ans: 6 8 10
	 System.out.println();
```

```
        ArrayList<Integer> list = new ArrayList<Integer>();
        printRootToLeaf(root,list);
        /*
         * Ans
         * Path:[8, 5, 3, 1]
           Path:[8, 5, 3, 4]
           Path:[8, 5, 6]
           Path:[8, 10, 11, 14]
         */
    }

}
```