P1 -Write a program for insertion and deletion in a Binary Search Tree ?Any pattern you can see while in the inorder traversal of BST?

BST.java file:

```
class BSTNode {
    int val;
    BSTNode left;
    BSTNode right;

    BSTNode(int val) {
        this.val = val;
        this.left = null;
        this.right = null;
    }
}

public class BST {
    public BSTNode root;

    public void insert(int val) {
        this.root = this.insert(val, this.root);
    }

    public BSTNode insert(int val, BSTNode node) {
        if (node == null)
            return new BSTNode(val);

        int cmp = val - node.val;
        if (cmp > 0)
            node.right = insert(val, node.right);
        else
            node.left = insert(val, node.left);

        return node;
    }

    public void delete(int key) {
        this.root = deleteNode(this.root, key);
    }

    public BSTNode deleteNode(BSTNode root, int key) {
        if (root == null)
            return root;
```

```
    int cmp = key - root.val;

    if (cmp > 0)
        root.right = deleteNode(root.right, key);
    else if (cmp < 0)
        root.left = deleteNode(root.left, key);
    else {
        if (root.left == null)
            return root.right;
        if (root.right == null)
            return root.left;

        BSTNode temp = root;
        root = min(temp.right);
        root.right = deleteMin(temp.right);
        root.left = temp.left;
    }

    return root;
}

public BSTNode deleteMin(BSTNode root) {
    if (root.left == null)
        return root.right;

    root.left = deleteMin(root.left);

    return root;
}

public BSTNode min(BSTNode root) {
    if (root.left == null)
        return root;

    return min(root.left);
}

public void inorder(BSTNode node) {
    if (node == null)
        return;
    inorder(node.left);
    System.out.println(node.val);
    inorder(node.right);
```

```
    }

    public void inorder() {
        inorder(this.root);
    }

    public static void main(String[] args) {
        BST bst = new BST();

        bst.insert(3);
        bst.insert(33);
        bst.insert(7);
        bst.insert(9);
        bst.insert(4);

        bst.inorder();

        bst.delete(33);
        bst.inorder();
    }

}
```

Output:

The inorder traversal always yields an ascending order of elements. Also, the traversal this depth-first traversal and left-to-right traversal.

P4- Write a program for level Order traversal in a Binary Tree and also explain your code complexities(Time/Space)?

```
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.Queue;

class BSTNode {
    int val;
    BSTNode left;
    BSTNode right;

    BSTNode(int val) {
        this.val = val;
        this.left = null;
        this.right = null;
    }
}

public class BST {
    public BSTNode root;

    public void insert(int val) {
        this.root = this.insert(val, this.root);
    }

    public BSTNode insert(int val, BSTNode node) {
        if (node == null)
            return new BSTNode(val);

        int cmp = val - node.val;
        if (cmp > 0)
            node.right = insert(val, node.right);
        else
            node.left = insert(val, node.left);

        return node;
    }

    public ArrayList<ArrayList<Integer>> levelOrder(BSTNode root) {
        if(root==null)
        {
            ArrayList<ArrayList<Integer>> res=new ArrayList<>();
            return res;
```

```
    }
    ArrayList<ArrayList<Integer>> res=new ArrayList<>();
    Queue<BSTNode> ob =new LinkedList<>();
    ob.add(root);
    ob.add(null);
    ArrayList<Integer> subRes=new ArrayList<>();
    while(!ob.isEmpty())
    {
        BSTNode front=ob.poll();

        if(front==null)
        {
            if(!ob.isEmpty())
            {
                res.add(subRes);
                subRes=new ArrayList<>();
                ob.add(front);
            }
            else
            {
                res.add(subRes);
                return res;
            }
        }
        else{
            subRes.add(front.val);

            if(front.left!=null) ob.add(front.left);
            if(front.right!=null) ob.add(front.right);
        }

    }
    return res;
}

public ArrayList<ArrayList<Integer>> levelOrder() {
    return levelOrder(this.root);
}


public static void main(String[] args) {
    BST bst = new BST();

    bst.insert(3);
```

```
        bst.insert(33);
        bst.insert(7);
        bst.insert(9);
        bst.insert(4);

        ArrayList<ArrayList<Integer>> result = bst.levelOrder();

        for (int i = 0; i < result.size(); i++) {
            for (int j = 0; j < result.get(i).size(); j++) {
                System.out.print(result.get(i).get(j) + " ");
            }
            System.out.println();
        }
    }

}
```

Time Complexity:

We can see that we are traversing the whole tree by visiting each node exactly once. Hence, the time complexity will be O(n) where n = number of nodes in tree

Space Complexity:

Since, we are storing the nodes at every level, the space complexity will be equal to the number of nodes in the tree and hence it will also be O(n) where n = number of nodes in tree