```
class Node {
    int key;
    Node left, right;
    public Node (int item)
    {
        key = item;
        left = right = null;

    }
} //BST using recursion
class BST {
    // A utility function to insert a new node
    // with the given key
    static Node insert (Node root, int key)
    {
        //If the tree is Empty, return a new node
        if (root == null)
            return new Node (key);




        //otherwise, recur down the tree
        if (key < root.key)
            root left = insert (root.left, key ):
        else
            root.right = insert (root.right, key);
```

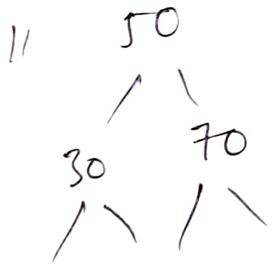
```
//Return the (unchanged) pointer

}
//A utility function to do inorder tree
//traversal
static void inorder(Node root)
{
    if (root != null){
        inorder (root.left);
        S.o.p(root.key + " ");
        inorder(root.right);

    }
}
//Driver method
Run | Debug
p svm(String[] args)
{
    Node root = null;
    //creating the following BST
    //          50
    //         /  \
    //       30    70
    //      /  \   / \
    //    20   40 60  80
```

```
root = insert(root, key: 50);
root = insert(root, key: 30);
 "   "    "   ("    "   20);
 "   "    "   ("    "   40);
 "   "    "   ("    "   70);
 "   "    "   ("    "   60);
 "   "    "   ("    "   80);
 "   "    "   ("    "

//print inorder traversal of the BST
  inorder(root);

 }
}
```

```
class Node {
    int key;
    Node left, right


class BSTiteration{
    // Function to insert a new node with
    // the given key
    static Node insert(Node root, int x){
        Node temp = new Node(x);
        //If tree is empty
```

```
if (root == null){
        return temp;

}
//Find the node who is going to have
//the new node temp as its child
Node parent=null;
Node curr=root;
while (curr != null){
        parent=curr;
        if(




}
//if x is smaller, make it left
//child, else right child
if (parent.key > x){
        parent.left = temp;
}else {
        parent.right = temp;
}
return root;
}
```

```
//A utility function to do inorder tree traversal
static void inorder(Node root)
{
    if (root != null){
            inorder.(root.left);
            S.o.p(root.key+" ");
            inorder(root.right);
    }
}

//Driver method
Run|Debug
p.s.v.m(String[] args)
{
        Node root=null;
```



```
class BST_del_rec {
    //This function deletes a given key x from the
    // given BST and returns the modified root of
    // the BST (if it is modified)
    static Node delNode(Node root, int x) {
                                                (1,0 , 15)
        // Base case
        if (root==null){                        10 ==
            return root;                        10 > 15
                                                10 < 15
        }
        //if key to be searched is in a subtree
        if(root.key>x) {
            root.left = delNode(root.left, x);
        } else if (root.key<x) {
            root.right=delNode(root.right, x);
        }else{
            // if root matches with the given key
            // cases when root has 0 children or
            // only right child.
            if (root.left==null) {
                return root.right;
        }
        // when root has only left child
        if(root.right==null) {
```

```
        return root.left;
    }
    //when both children are present
    Node succ = getSuccessor(root);
    root.key = succ.key;
    root.right = delNode(root.right, succ.key);
    }
    return root;
}
//Note that it is not a generic inorder successo
//function. It mainly works when the right chi
//is not Empty, which is the case we need in
// BST
~~//when root has only left child~~
~~if (root.right == null) {~~
~~    return root.left;~~
~~}~~
~~//when both children are present~~
~~Node~~
static Node getSuccessor(Node curr) {
    curr = curr.right;
    while (curr != null && curr.left != null) {
        curr = curr.left;
    }
    return curr;
```

```
}
//utility function to do inorder traversal
static void inorder(Node root){
    if (root != null){
        inorder (root.left);
        S.o.p(root.key + " " )


}
psvm (String[] args){
Node root = new Node(item:10);
root.left = new Node(item:5);
root.right = new Node(item:15);
root.right.left = new Node(item:12);
root.right.right = new Node(item:10);
                                   17
int x = 15;
root = delNode (root, x);
inorder (root);
```

Step-1: Node root = new Node(item:10)

1000 $\boxed{10}$

root -> 10 (address)

root = del Node (root, x)

S.2: static Node del Node (Node root, int x)

3000 $\boxed{10}$. 15

root·key
3000·key = 15

~~15 > 10~~
10 > 15 ✗
10 < 15 ✓

getSuccessor fun call
↑
3000

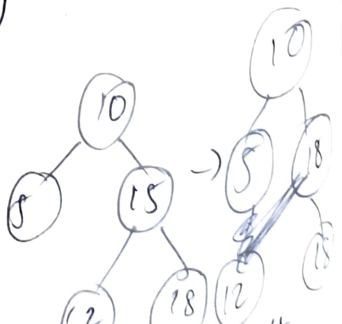
5000 ~~3000~~ 3000 · right
curr = curr.right
5000 · 4000
while (5000 curr != null && curr.left)
∝

4000
4000 Cu~~rr~~ ← ~~Curr.left~~

}
5000
return

5000
5000 . key = 18
root·key = Succ.key

3000·key → 18



5000
5000. key = 18
root·right = delNode(root·right, succ.key);