```
class BST

Node root;

// find and return node in the tree with the key value using recursion
Node find(int key)
     return find(root,key)

// recursive part of find
Node find(Node root, int key)

      check root is null/None or key is present at root and return root
      check key is greater than root's key
             return find(root's right subtree, key)
      check key is smaller than root's key
          return find(root's left subtree, key)


// find smallest node   recursively (left most leaf node)
Node findMin()
      return findMin(root)

//  recursive part of findMin
Node findMin(Node root)

      check root's left subtree is null/None and return root
      otherwise return findMin(root's left subtree)

// find max node  recursively (right most leaf node)
Node findMax()
      return findMax(root)

// recursive pat of findMax
Node findMax(Node root)

      check root's right subtree is null/None and return root
      otherwise return findMin(root's right subtree)


// insert node in tree using recursion
// it needs to find the correct place to keep binary search tree
void insert(int data)

      root = insertInTree(root, data)

// recursive part of insert
Node insertInTree(Node root, int key)

      if sub-tree is empty return new Node instance
```

```
      if key is less than root’s key call recursively insertinTree for left
subtree
            root.left = insertInTree(root’s left subtree, key)
      else if key is greater than root’s key call recursively insertinTree
for left subtree

            root.right = insertInTree(root’s right subtree, key)

      return root
// delete node in tree using recursion
void delete(int data)

      root = deleteInTree(root, data)

// recursive part of delete
Node deleteInTree(Node root, int key)

      If sub-tree tree is empty return root

     /* recurse down the tree to find the node to delete*/

     if key is less than root’s key
            root.left = deleteInTree(root’s left subtree, key)
     else if key greater root’s key
            root.right = deleteInTree(root’s right subtree, key)

     else  // found node to delete

            // node with only one child or no child
            if root’s left is null/None return root’s right (no child)
            else if root’s right is null/None return root’s left (one child)

            //node with two children: Get the inorder successor
            // smallest in the right subtree
            root’s key = minValue(root.right)

            // Delete the inorder successor
            root.right = deleteInTree(root.right, root’s key)

      return root

// get smallest value of the tree (left most leaf node)
int minValue(Node root)
    {
        int minv = root’s key
        loop while root.left != null/None
            minv = root.left()’s key
            root = root.left
        return minv
```

```
// traverse node’s in in-order and print key values
void traverseInOrder()
      traverseInOrder(root)

// recursive part of inorder
void traverseInOrder(Node root)

      if root != null/None
            traverseInOrder(root.left)
            visit(root)
            traverseInOrder(root.right)


void visit(Node n)
      print n’s key

Node getRoot()
      return root
```