l=len(a)

for i in range(l,0,-1):

print(a[i])

class Node:

def \_init\_(self,data):

self.left=None

self.right=None

self.data=data

class bst: #To create a tree

def \_init\_(self):

self.root=None

def insert(self,data,root):

if root is None:

return Node(data)

if data<root.data:

root.left=self.insert(data,root.left)

elif data>root.data:

root.right=self.insert(data,root.right)

return root

def inorder\_traversal(self,root):

if root:

self.inorder\_traversal(root.left)

print(root.data,end=" ")

self.inorder\_traversal(root.right)

def preorder\_traversal(self,root):

if root:

print(root.data,end=" ")

self.preorder\_traversal(root.left)

self.preorder\_traversal(root.right)

def postorder\_traversal(self,root):

if root:

self.postorder\_traversal(root.left)

self.postorder\_traversal(root.right)

print(root.data,end=" ")

bst\_tree=bst()

root=None

root=bst\_tree.insert(20,root)

root=bst\_tree.insert(10,root)

root=bst\_tree.insert(900,root)

bst\_tree.inorder\_traversal(root)

bst\_tree.preorder\_traversal(root)

bst\_tree.postorder\_traversal(root)

class Node:

def \_init\_(self,data):

self.left=None

self.right=None

self.data=data

class bst: #To create a tree

def \_init\_(self):

self.root=None

def insert(self,data,root):

if root is None:

return Node(data)

if data<root.data:

root.left=self.insert(data,root.left)

elif data>root.data:

root.right=self.insert(data,root.right)

return root

def inorder\_traversal(self,root):

if root:

self.inorder\_traversal(root.left)

print(root.data,end=" ")

self.inorder\_traversal(root.right)

def preorder\_traversal(self,root):

if root:

print(root.data,end=" ")

self.preorder\_traversal(root.left)

self.preorder\_traversal(root.right)

def postorder\_traversal(self,root):

if root:

self.postorder\_traversal(root.left)

self.postorder\_traversal(root.right)

print(root.data,end=" ")

def search(self,root,key):

if root.data==key:

return True

if root is None:

return False

def search(self,root,key):

if(key<root.data):

return self.search(root.left,key)

elif(key>root.data):

return self.search(root.right,key)

else:

return True

bst\_tree=bst()

root=None

root=bst\_tree.insert(20,root)

root=bst\_tree.insert(10,root)

root=bst\_tree.insert(900,root)

bst\_tree.inorder\_traversal(root)

print("\n inorder\_traversal")

bst\_tree.preorder\_traversal(root)

print("\n preorder\_traversal")

bst\_tree.postorder\_traversal(root)

print("\n postorder\_traversal")

print("\n Search\_in\_bst")

print(bst\_tree.search(root,10))

class Node:

def \_init\_(self,data):

self.left=None

self.right=None

self.data=data

class bst: #To create a tree

def \_init\_(self):

self.root=None

def insert(self,data,root):

if root is None:

return Node(data)

if data<root.data:

root.left=self.insert(data,root.left)

elif data>root.data:

root.right=self.insert(data,root.right)

return root

def inorder\_traversal(self,root):

if root:

self.inorder\_traversal(root.left)

print(root.data,end=" ")

self.inorder\_traversal(root.right)

def preorder\_traversal(self,root):

if root:

print(root.data,end=" ")

self.preorder\_traversal(root.left)

self.preorder\_traversal(root.right)

def postorder\_traversal(self,root):

if root:

self.postorder\_traversal(root.left)

self.postorder\_traversal(root.right)

print(root.data,end=" ")

def search(self,root,key):

if root.data==key:

return True

if root is None:

return False

def search(self,root,key):

if(key<root.data):

return self.search(root.left,key)

elif(key>root.data):

return self.search(root.right,key)

else:

return True

def Height(self,root):

if root is None:

return 0

else:

return max(self.Height(root.left),self.Height(root.right))+1

def sum\_of\_roots(self,root):

if root is None:

return 0

return root.data + self.sum\_of\_roots(root.left)+self. sum\_of\_roots(root.right)

def count\_root(self,root):

if root is None:

return 0

return 1+self.count\_root(root.left)+self.count\_root(root.right)

bst\_tree=bst()

root=None

root=bst\_tree.insert(20,root)

root=bst\_tree.insert(10,root)

root=bst\_tree.insert(900,root)

bst\_tree.inorder\_traversal(root)

print("\n inorder\_traversal")

bst\_tree.preorder\_traversal(root)

print("\n preorder\_traversal")

bst\_tree.postorder\_traversal(root)

print("\n postorder\_traversal")

print("\n Search\_in\_bst")

print(bst\_tree.search(root,10))

print("\n Height of BST")

print(bst\_tree.Height(root))

print("\n sum of root")

print(bst\_tree.sum\_of\_roots(root))

print("\n count\_root")

print(bst\_tree.count\_root(root))