# Experiment – 3.2

**Student Name: Rohit Kumar Mahato** **UID: 21BCS7480**
**Branch: CSE** **Section/Group: 717-A**
**Semester: 3rd** **Date of Performance: 03/11/2022**
**Subject Name: Data structure** **SubjectCode:21CSH-211**

1. **AIM:-** Write a program to illustrate the implementation of different Operations on a binary search tree.
2. **ALGORITHM:**

For Inorder Tree

1. Start
2. Create a node and declear the variables.
3. Check the condition
   if(root!=NULL)
4. Traverse the left subtree, i.e., call Inorder(left->subtree)
5. Visit the root. cout<<root->data<<" ";
6. Traverse the right subtree, i.e., call Inorder(right->subtree)
7. Print the tree.
8. End.

For Preorder Tree

1.Start
2.Create a node and declear the variables.
3.Check the condition
   if(root!=NULL)
4.Visit the root. cout<<root->data<<" ";
5.Traverse the left subtree, i.e., call Inorder(left->subtree)
6.Traverse the right subtree, i.e., call Inorder(right->subtree)
7.Print the tree.
8.End.

For Postorder Tree

1.Start
2.Create a node and declear the variables.
3.Check the condition
   if(root!=NULL)
4.Traverse the left subtree, i.e., call Inorder(left->subtree)
5.Traverse the right subtree, i.e., call Inorder(right->subtree)
6.Visit the root. cout<<root->data<<" ";
7.Print the tree.
8.End.

**Inserting of a node in a tree**

1. Start
2. Create a new BST node and assign values to it.
3. insert(node, key)

i) If root == NULL,

return the new node to the calling function.

ii) if root=>data < key

call the insert function with root=>right and assign the return value in root=>right.

root->right = insert(root=>right,key)

iii) if root=>data > key

call the insert function with root->left and assign the return value in root=>left.

root=>left = insert(root=>left,key)

4. Finally, return the original root pointer to the calling function.
5. End

**Deletion of a node in a tree**

1. Start
2. if tree = null

write "item not found in the tree" else if item < tree -> data

delete(tree->left, item)

else if item > tree -> data

delete(tree -> right, item)

else if tree -> left and tree -> right

set temp = findlargestnode(tree -> left)

set tree -> data = temp -> data

delete(tree -> left, temp -> data)

else

set temp = tree

if tree -> left = null and tree -> right = null

set tree = null

else if tree -> left != null

set tree = tree -> left

else

set tree = tree -> right

[end of if]

free temp

[end of if]

3: End

### 3. Program Code: -

```
#include<iostream>
using namespace std;

struct node{
    int data;
    struct node* left;
    struct node* right;
};

struct node* createNode(int data){
    struct node *n;
    n = (struct node *) malloc(sizeof(struct node));
    n->data = data;
    n->left = NULL;
    n->right = NULL;
    return n;
}

void preOrder(struct  node* root){
    if(root!=NULL){
        cout<<root->data<<" ";
        preOrder(root->left);
        preOrder(root->right);
    }
}

void postOrder(struct  node* root){
    if(root!=NULL){
        postOrder(root->left);
        postOrder(root->right);
        cout<<root->data<<" ";
    }
}

void inOrder(struct  node* root){
    if(root!=NULL){
        inOrder(root->left);
        cout<<root->data<<" ";
        inOrder(root->right);
    }
}

int isBST(struct  node* root){
    static struct node *prev = NULL;
    if(root!=NULL){
        if(!isBST(root->left)){
            return 0;
        }
        if(prev!=NULL && root->data <= prev->data){
            return 0;
        }
        prev = root;
        return isBST(root->right);
```

```
    }
    else{
        return 1;
    }
}

struct node * searchIter(struct node* root, int key){
    while(root!=NULL){
        if(key == root->data){
            return root;
        }
        else if(key<root->data){
            root = root->left;
        }
        else{
            root = root->right;
        }
    }
    return NULL;
}

void insert(struct node *root, int key){
   struct node *prev = NULL;
   while(root!=NULL){
       prev = root;
       if(key==root->data){
           cout<<"Cannot insert" <<key<<" already in BST";
           return;
       }
       else if(key<root->data){
           root = root->left;
       }
       else{
           root = root->right;
       }
   }
   struct node* begin = createNode(key);
   if(key<prev->data){
       prev->left = begin;
   }
   else{
       prev->right = begin;
   }

}

struct node *inOrderPredecessor(struct node* root){
    root = root->left;
    while (root->right!=NULL)
    {
        root = root->right;
    }
    return root;
}

struct node *deleteNode(struct node *root, int value){

    struct node* iPre;
    if (root == NULL){
        return NULL;
```

```
    }
    if (root->left==NULL&&root->right==NULL){
        free(root);
        return NULL;
    }

    if (value < root->data){
        root-> left = deleteNode(root->left,value);
    }
    else if (value > root->data){
        root-> right = deleteNode(root->right,value);
    }
    else{
        iPre = inOrderPredecessor(root);
        root->data = iPre->data;
        root->left = deleteNode(root->left, iPre->data);
    }
    return root;
}

int main(){

    struct node *p = createNode(5);
    struct node *p1 = createNode(3);
    struct node *p2 = createNode(6);
    struct node *p3 = createNode(1);
    struct node *p4 = createNode(4);

    p->left = p1;
    p->right = p2;
    p1->left = p3;
    p1->right = p4;

    inOrder(p);
    cout<<"in-Order tree"<<endl;
    preOrder(p);
    cout<<"Pre-Order tree"<<endl;
    postOrder(p);
    cout<<"Post-Order tree"<<endl;
    insert(p, 16);
    cout<< p->right->right->data<<" "<<endl;
    inOrder(p);
    cout<<endl;
    deleteNode(p, 3);
    inOrder(p);

    return 0;
}
```

**Output:-**


**Learning Outcomes (What I have learnt)**

1. Learnt about trees and its types
2. Also learnt the various operations on trees