# Yash Sarang.
# Roll No: 47, Class : D6AD.

# Data Structures. Experiment-13.

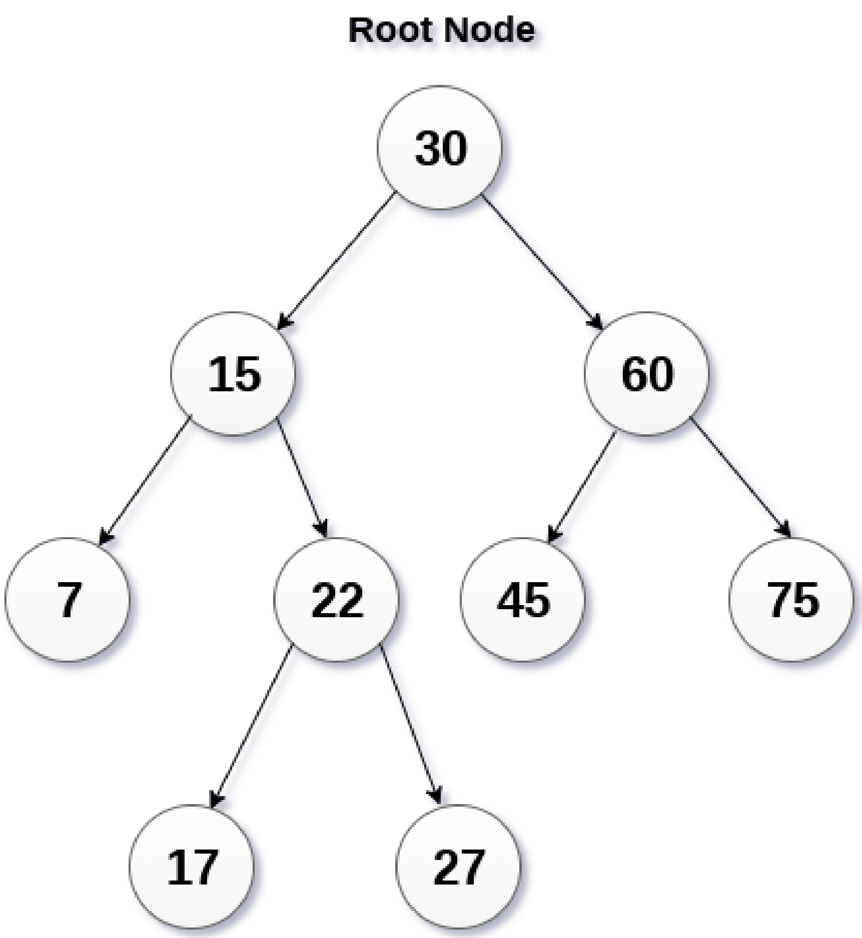
---

**AIM**: To write a program to implement binary search trees.

---

## THEORY:

A binary Search tree can be defined as a class of binary trees, in which the nodes are arranged in a specific order. This is also called an ordered binary tree. In a binary search tree, the value of all the nodes in the left sub-tree is less than the value of the root. Similarly, the value of all the nodes in the right sub-tree is greater than or equal to the value of the root.




Binary Search Tree

OPERATIONS ON BST:

Searching: Finding the location of some specific element in a binary search tree.

Insertion: Adding a new element to the binary search tree at the appropriate location so that the property of BST does not violate.

Deletion: Deleting some specific node from a binary search tree. However, there can be various cases of deletion depending upon the number of children, the node has.

---

ALGORITHM:

**1. Insert:**

**Step 1** START

**Step 2** Store the key to be inserted (x)

**Step 3** Check element present in the tree if not goto step 4 else step 5

**Step 4** Make inserted key Root Node

**Step 5** Compare x with root node if smaller goto step 6 else goto step 7 or no root node find goto step 9.

**Step 6** Element reaches the left subtree repeat Step 5

**Step 7** Element reaches the right subtree repeat Step 5

**Step 8** Insert the key

**Step 9** STOP

**2.Deletion:**

```
Step 1: 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]

Step 2: END

**3. Search:**

Step1: START

Step2: store the element to be searched

Step3: check if the element to be searched is greater or lower than the root value

Step4: If the element value is less than the root value then search in the left subtree, if greater than the root value then search in the right subtree.

Step5: continue this process for the further nodes till you get the element.

Step6: END

PROGRAM:

1. search:

```
#include<stdio.h>
#include<malloc.h>

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

struct node* createNode(int data){
    struct node *n; // creating a node pointer
    n = (struct node *) malloc(sizeof(struct node)); // Allocating memory
    n->data = data; // Setting the data
    n->left = NULL; // Setting the left and right children to NULL
    n->right = NULL; // Setting the left and right children to NULL
    return n; // Finally returning the created node
}

void preOrder(struct  node* root){
    if(root!=NULL){
        printf("%d ", root->data);
        preOrder(root->left);
        preOrder(root->right);
    }
}

void postOrder(struct  node* root){
    if(root!=NULL){
        postOrder(root->left);
        postOrder(root->right);
        printf("%d ", root->data);
    }
}

void inOrder(struct  node* root){
    if(root!=NULL){
        inOrder(root->left);
```

```c
        printf("%d ", 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 * search(struct node* root, int key){
    if(root==NULL){
        return NULL;
    }
    if(key==root->data){
        return root;
    }
    else if(key<root->data){
        return search(root->left, key);
    }
    else{
        return search(root->right, key);
    }
}
```

```c
int main(){

    // Constructing the root node
    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;

    struct node* n = search(p, 3);
    if(n!=NULL){
    printf("Found: %d", n->data);
    }
    else{
        printf("Element not found");
    }
    return 0;
}
```

---

OUTPUT:

```
...Program finished with exit code 0
Press ENTER to exit console.
```

---

2. Insertion:

Program:

```
#include<stdio.h>
#include<malloc.h>

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){
        printf("%d ", root->data);
        preOrder(root->left);
        preOrder(root->right);
    }
}

void postOrder(struct  node* root){
    if(root!=NULL){
        postOrder(root->left);
        postOrder(root->right);
        printf("%d ", root->data);
    }
}

void inOrder(struct  node* root){
    if(root!=NULL){
        inOrder(root->left);
```

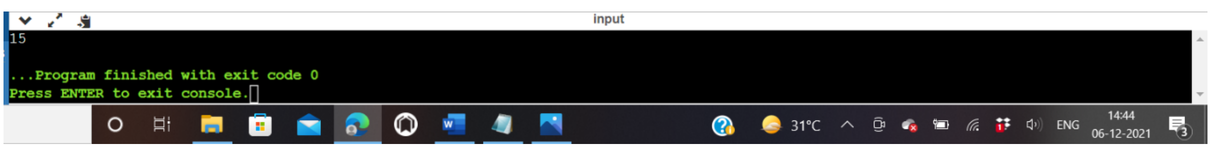
```c
int isBST(struct  node* root){
    static struct node *prev = NULL;
    if(root!=NULL){
        if(!isBST(root->left)){
            return 0;
        }
        if(prev!=NULL && root->data <= prev->data){
            return 0;
        }
        prev = root;
        return isBST(root->right);
   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

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

}

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;
```

```c
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;

    insert(p, 15);
    printf("%d", p->right->right->data);
    return 0;
}
```

---

OUTPUT:




---

3. Deletion

Program:

```c
NODE* del(NODE *node, int data)
{
NODE *temp;

if(node == NULL)
{
printf("\nElement not found");
}

else if(data < node->data)
{
node->left = del(node->left, data);
}

else if(data > node->data)
{
node->right = del(node->right, data);
}
```

```c
else
{ /* Now We can delete this node and replace with
    if(node->right && node->left)
{ /* Here we will replace with minimum element in
    temp = findMin(node->right);
    node -> data = temp->data;
/* As we replaced it with some other node, we have
    node -> right = del(node->right,temp->data);
}

else
{
/* If there is only one or zero children then we 

temp = node;

if(node->left == NULL)
    node = node->right;

else if(node->right == NULL)
    node = node->left;

free(temp); /* temp is longer required */
}
}
return node;
}
```

## OUTPUT:

```
Enter the number of nodes : 6

Input the nodes of the binary search tree : 7 5 13 4 11 19

Inorder traversal of the BST : 4  5  7  11  13  19

Enter the node to be deleted : 7

Inorder traversal after deletion : 4  5  11  13  19

Process returned 0 (0x0)   execution time : 12.532 s
Press any key to continue.
```

**Conclusion :** We have successfully implemented a Binary Search Tree.