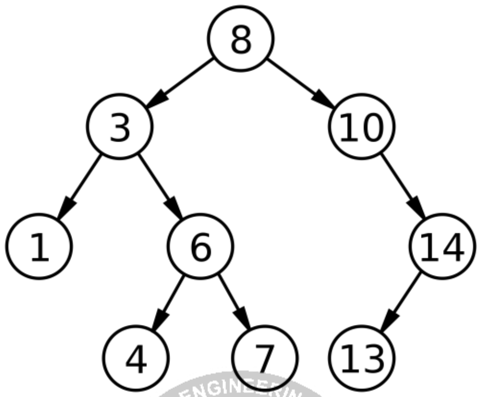
BINARY SEARCH TREE OR SEARCH TREE ADT

Definition: -

When we place constraints on how data elements can be stored in the tree, the items must be stored in such a way that the key values in left subtree of the root less than the key value of the root, and then the key values of all the node in the right subtree of the root are greater than the key values of the root. When this relationship holds in the entire node in the tree then the tree is called as a binary search tree.

The property that makes a binary tree into a binary search tree. That is every node X in the tree, the values of all the keys in its left subtree are smaller than the key value in X, and the values of all the keys in its right subtree are larger than the key value in X.

Example:

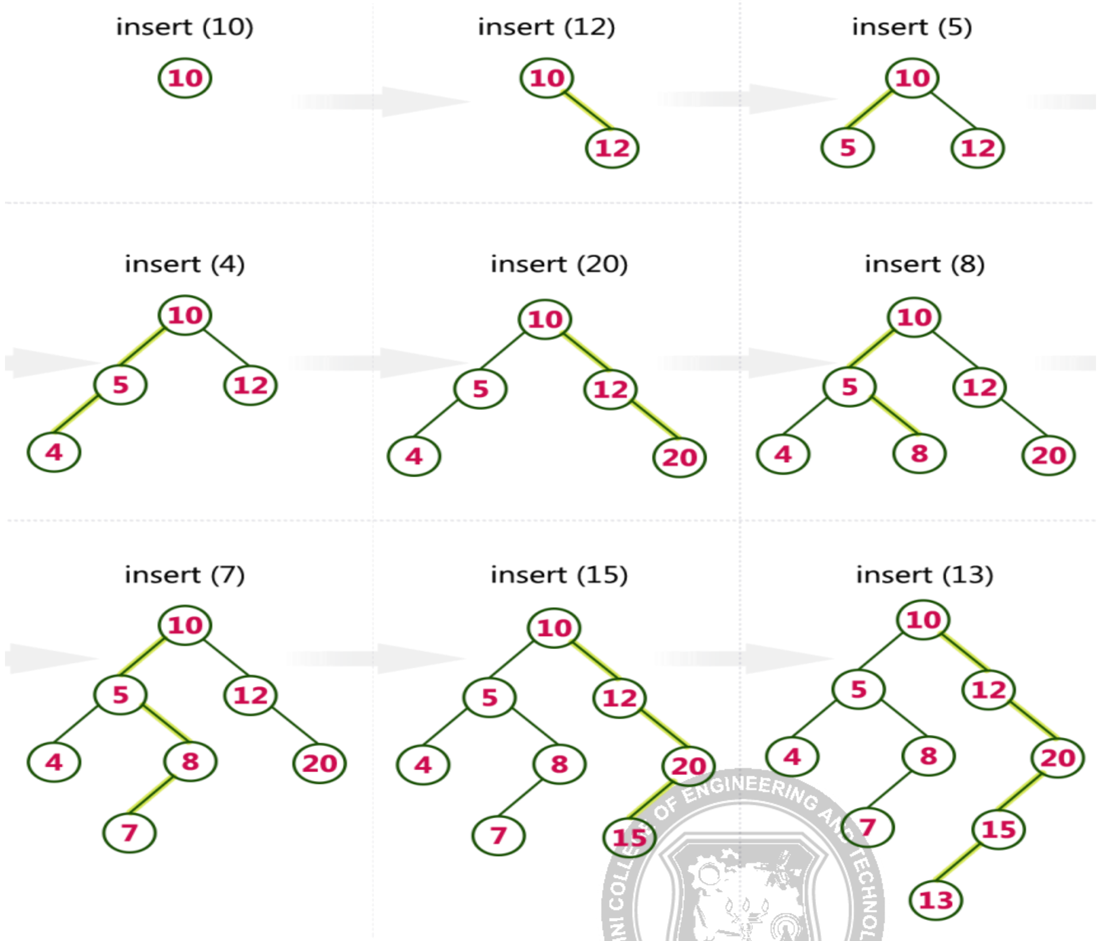


Operations of BST.

1. Insertion
2. Deletion
3. Find
4. Find min
5. Find max
6. Retrieve

Notes: when you're constructing the binary tree the given elements are read from first.

Example:



```
{
        int data;
        struct treenode *left;
        struct treenode *right;
};
```

Routine for perform find.

```
struct treenode * find(struct treenode *T,int x)
{
        if(T==NULL)
                return NULL;
        else if(x<T->data)
                return find(T->left,x);
        else if(x>T->data)
                return find(T->right,x);
        else
                return T;
}
```

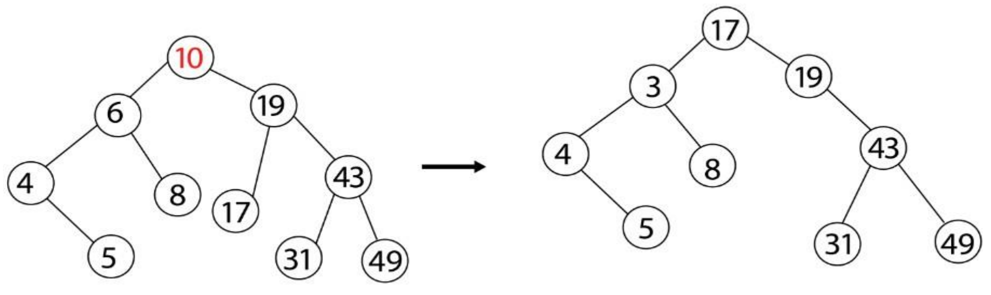
Routine for perform insertion.

```
struct treenode* insert(struct treenode *t,int x)
{
        if(t==NULL)

                t=(struct treenode *)malloc(sizeof(struct treenode));
                t->data=x;
                t->left=t->right=NULL;
                return t;
        }
else if(x<t->data)
t->left=insert(t->left,x);else if(x>t->data)
t->right=insert(t->right,x);
return t;
}
```

Deletion — I/P = **10**

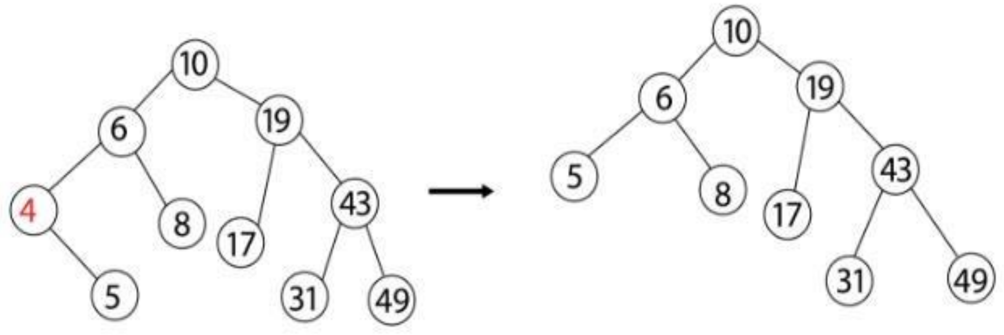
Node deleted to have 2 child.



Before Deletion After Deletion

Deletion — I/P = **4**

Node deleted to have 1 child.



Before Deletion After Deletion

```
   return NULL;  /* There is no element in the tree */
  else if(t->left==NULL) /* Go to the left sub tree to find the min element */
   return t;
  else
   return findmin(t->left);
}

struct treenode* deletion(struct treenode *t,int x)
{
        struct treenode *temp;
  if(t==NULL)
              printf("Element not found\n");
 else if(x < t->data )
            t->left = deletion (t->left,x);
else if(x > t->data )
            t->right = deletion (t->right,x);
else if(T->right && T->left)
 {
            /* Here we will replace with minimum element in the right sub tree */
             temp = findmin(t->right);
             t->data = temp->data ;
            /* As we replaced it with some other node, we have to delete that node */
             t->right = deletion (t->right,t->data);
}
else
  {
          /* If there is only one or zero children then we can directly remove it from the tree
        and connect itsparent to its child */
temp = T;
if(t->left==NULL)
t = t->right;
else if(T->right ==
NULL)t = t->left;
free(temp); /* temp is longer required */
        }
         return T;
 }

 void inorder(struct treenode *t)
 {
if(t!=NULL)
{
inorder(t->left);
printf("%d \t",t
>data);
inorder(t->right);
 }
 }
```

Applications of Tree

1. Manipulate hierarchical data.
2. Make information easy to search 3. Manipulate sorted lists of data.
4. As a workflow for compositing digital images for visual effects.
5. Router algorithms

DIFFERENCE BETWEEN BINARY AND BINARY SEARCH TREES:

BINARY TREE	BINARY SEARCH TREE
It is a tree with only two children	It is also a tree with only two children.
It has no restrictions regarding its children	In this the left child is lesser than the parent and the right child is greater than the parent