**Binary Search Tree**

In a binary tree, every node can have maximum of two children but there is no order of nodes based on their values. In binary tree, the elements are arranged as they arrive to the tree, from top to bottom and left to right.

A binary tree has the following time complexities...

**1. Search Operation - O(n)**

**2. Insertion Operation - O(1)**

**3. Deletion Operation - O(n)**

To enhance the performance of binary tree, we use special type of binary tree known as **Binary**

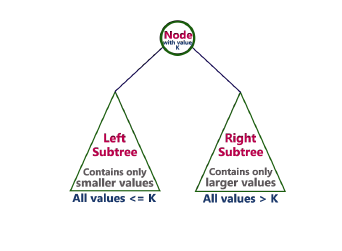
**Search Tree**. Binary search tree mainly focus on the search operation in binary tree. Binary search

tree can be defined as follows...

**Binary Search Tree is a binary tree in which every node contains only smaller values in its**

**left subtree and only larger values in its right subtree.**

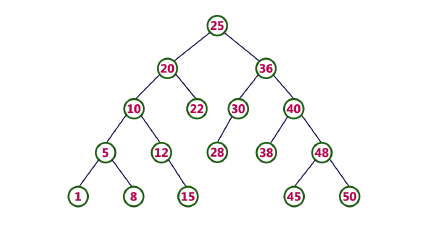
In a binary search tree, all the nodes in left subtree of any node contains smaller values and all the nodes in right subtree of that contains larger values as shown in following figure...



**Example**

The following tree is a Binary Search Tree. In this tree, left subtree of every node contains nodes

with smaller values and right subtree of every node contains larger values.



**Every Binary Search Tree is a binary tree but all the Binary Trees need not to be binary search**

**trees.**

**Operations on a Binary Search Tree**

The following operations are performed on a binary earch tree...

**1. Search**

**2. Insertion**

**3. Deletion**

**Search Operation in BST**

In a binary search tree, the search operation is performed with **O(log n)** time complexity. The

search operation is performed as follows...

**Step 1:** Read the search element from the user

**Step 2:** Compare, the search element with the value of root node in the tree.

**Step 3:** If both are matching, then display "Given node found!!!" and terminate the function

**Step 4:** If both are not matching, then check whether search element is smaller or larger than

that node value.

**Step 5:** If search element is smaller, then continue the search process in left subtree.

**Step 6:** If search element is larger, then continue the search process in right subtree.

**Step 7:** Repeat the same until we found exact element or we completed with a leaf node

**Step 8:** If we reach to the node with search value, then display "Element is found" and

terminate the function.

**Step 9:** If we reach to a leaf node and it is also not matching, then display "Element not

found" and terminate the function.

**Insertion Operation in BST**

In a binary search tree, the insertion operation is performed with **O(log n)** time complexity. In

binary search tree, new node is always inserted as a leaf node.The insertion operation is performed as follows...

**Step 1:** Create a newNode with given value and set its **left** and **right** to **NULL**.

**Step 2:** Check whether tree is Empty.

**Step 3:** If the tree is **Empty**, then set set **root** to **newNode**.

**Step 4:** If the tree is **Not Empty**, then check whether value of newNode

is **smaller** or **larger** than the node (here it is root node).

**Step 5:** If newNode is **smaller** than **or equal** to the node, then move to its **left** child. If

newNode is **larger**than the node, then move to its **right** child.

**Step 6:** Repeat the above step until we reach to a **leaf**node (e.i., reach to NULL).

**Step 7:** After reaching a leaf node, then isert the newNode as **left child** if newNode is **smaller**

**or equal** to that leaf else insert it as **right child**.

**Deletion Operation in BST**

In a binary search tree, the deletion operation is performed with **O(log n)** time complexity. Deleting a node from Binary search tree has follwing three cases...

**Case 1: Deleting a Leaf node (A node with no children)**

**Case 2: Deleting a node with one child**

**Case 3: Deleting a node with two children**

**Case 1: Deleting a leaf node**

We use the following steps to delete a leaf node from BST...

**Step 1: Find** the node to be deleted using **search operation**

**Step 2:** Delete the node using **free** function (If it is a leaf) and terminate the function.

**Case 2: Deleting a node with one child**

We use the following steps to delete a node with one child from BST...

**Step 1: Find** the node to be deleted using **search operation**

**Step 2:** If it has only one child, then create a link between its parent and child nodes.

**Step 3:** Delete the node using **free** function and terminate the function.

**Case 3: Deleting a node with two children**

We use the following steps to delete a node with two children from BST...

**Step 1: Find** the node to be deleted using **search operation**

**Step 2:** If it has two children, then find the **largest** node in its **left subtree** (OR)

the **smallest** node in its **right subtree**.

**Step 3: Swap** both **deleting node** and node which found in above step.

**Step 4:** Then, check whether deleting node came to **case 1** or **case 2** else goto steps 2

**Step 5:** If it comes to **case 1**, then delete using case 1 logic.

**Step 6:** If it comes to **case 2**, then delete using case 2 logic.

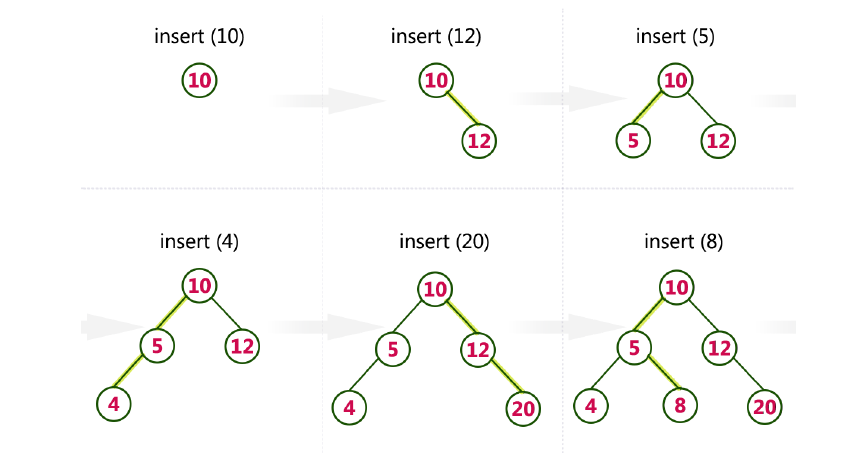
**Step 7:** Repeat the same process until node is deleted from the tree.

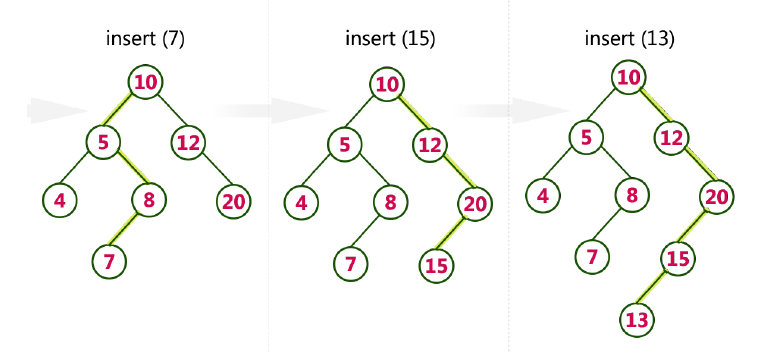
**Example**

Construct a Binary Search Tree by inserting the following sequence of numbers...

**10, 12, 5, 4,20,8,7,15 and 13.**

Above elements are inserted into a Binary Search Tree as follows...





/\*Binary Search Tree – Addition, Deletion with Preorder, inorder and postorder traversal \*/

#include <stdio.h>

#include <stdlib.h>

struct node

{

int data;

struct node \*left,\*right;

};

typedef struct node \*NODE;

NODE getnode()

{

NODE x;

x = (NODE)malloc(sizeof(struct node));

if(x==NULL)

{

printf("No free menory\n");

exit(0);

}

return x;

}

//Function to create a BST by inserting elements.

NODE insert(int item,NODE root)

{

NODE temp,pre,cur;

temp = getnode();

temp->data = item;

temp->left = temp->right = NULL;

if (root==NULL)

return temp;

pre = NULL;

cur = root;

while(cur != NULL)

{

pre = cur;

cur=(item<cur->data)? cur->left:cur->right;

}

if (item < pre->data)

pre->left = temp;

else

pre->right = temp;

return root;

}

//Function to traverse tree in preorder

void preorder(NODE root)

{

if(root!=NULL)

{

printf("%d\t",root->data);

preorder(root->left);

preorder(root->right);

}

}

//Function to traverse tree in inorder

void inorder(NODE root)

{

if(root!=NULL)

{

inorder(root->left);

printf("%d\t",root->data);

inorder(root->right);

}

}

//Function to traverse tree in postorder

void postorder(NODE root)

{

if(root!=NULL)

{

postorder(root->left);

postorder(root->right);

printf("%d\t",root->data);

}

}

//Function to delete a node

NODE delete(int key,NODE root)

{

NODE cur,parent,suc,q;

if (root==NULL)

{

printf("Empty tree");

return root;

}

parent=NULL;

cur=root;

while(cur!=NULL && key!=cur->data)

{

parent=cur;

cur=(key<cur->data)?cur->left:cur->right;

}

if(cur==NULL)

{

printf("Key not found");

return root;

}

if(cur->left==NULL)

{

q=cur->right;

if(cur->right==NULL)

q=cur->left;

else{

suc=cur->right;

while(suc->left!=NULL)

suc=suc->left;

suc->left=cur->left;

q=cur->right;

}

if(parent==NULL)return q;

if(cur==parent->left)

parent->left=q;

else

parent->right=q;

free(cur);

return root;

}

}

int main(int argc, char\*\* argv) {

NODE root=NULL;

int opt,item;

for(;;)

{

printf("Creating a binary tree and traversing the tree\n");

printf("Enter your optin\n");

printf("1: Insert an element to tree\n");

printf("2: Pre order traversal\n");

printf("3: In order traversal\n");

printf("4: Post order traversal\n");

printf("5: Delete an element from a tree\n");

printf("6: Exit\n");

scanf("%d",&opt);

switch(opt)

{

case 1: printf("Enter enter the element to be inserted\n");

scanf("%d",&item);

root=insert(item,root);

break;

case 2: printf("PREORDER TRAVERSAL\n");

preorder(root);

break;

case 3: printf("INORDER TRAVERSAL\n");

inorder(root);

break;

case 4: printf("POSTORDER TRAVERSAL\n");

postorder(root);

break;

case 5: printf("Delete an element from a tree\n");

printf("Enter an element to delete:");

scanf("%d/n",&item);

delete(item,root);

break;

case 6:

default:exit(0);

}

}

}