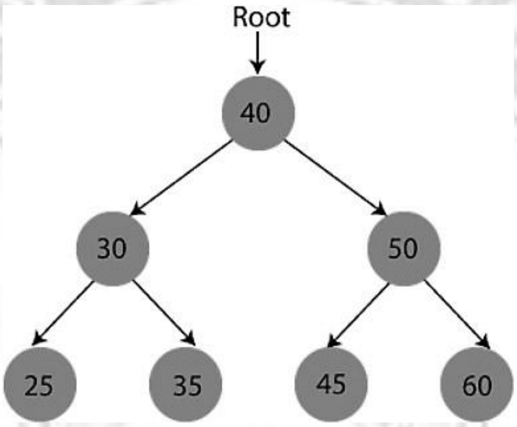
BINARY SEARCH TREE

- In a Binary search tree, the value of left node must be smaller than the parent node, and the value of right node must be greater than the parent node. This rule is applied recursively to the left and right subtrees of the root.
- Example for Binary Search Tree



- In the above figure, we can observe that the root node is 40, and all the nodes of the left subtree are smaller than the root node, and all the nodes of the right subtree are greater than the root node.
- Similarly, we can see the left child of root node is greater than its left child and smaller than its right child. So, it also satisfies the property of binary search tree. Therefore, we can say that the tree in the above image is a binary search tree.

Advantages of Binary search tree

- Searching an element in the Binary search tree is easy as we always have a hintthat which subtree has the desired element.
- As compared to array and linked lists, insertion and deletion operations are fasterin BST.

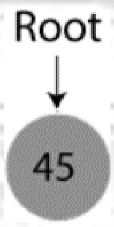
Example of creating a binary search tree

- Now, let's see the creation of binary search tree using an example. Suppose thedata elements are : 45, 15, 79, 90, 10, 55, 12, 20, 50
  - First, we have to insert 45 into the tree as the root of the tree.
  - Then, read the next element; if it is smaller than the root node, insert it as the root of the left subtree, and move to the next element.

- Otherwise, if the element is larger than the root node, then insert it as the root of the right subtree.

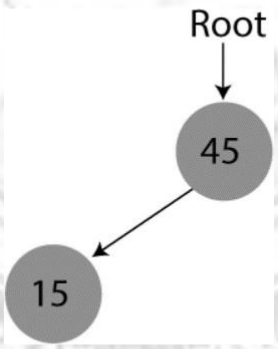
➢ Now, let's see the process of creating the Binary search tree using the given dataelement. The process of creating the BST is shown below

➢ Step 1 - Insert 45.



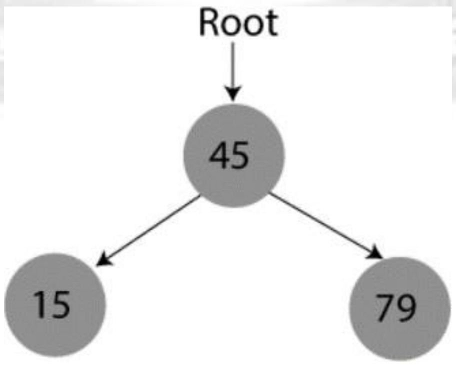
➢ Step 2 - Insert 15.

- As 15 is smaller than 45, so insert it as the root node of the left subtree.



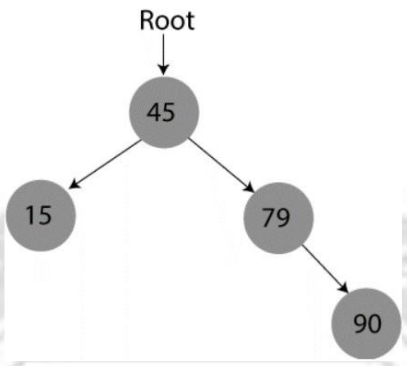
➢ Step 3 - Insert 79.

- As 79 is greater than 45, so insert it as the root node of the right subtree.

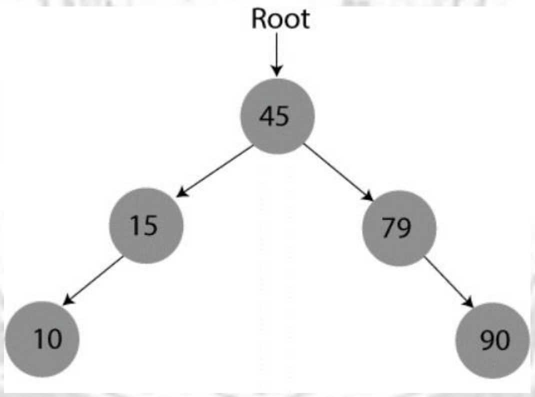


➢ Step 4 - Insert 90.

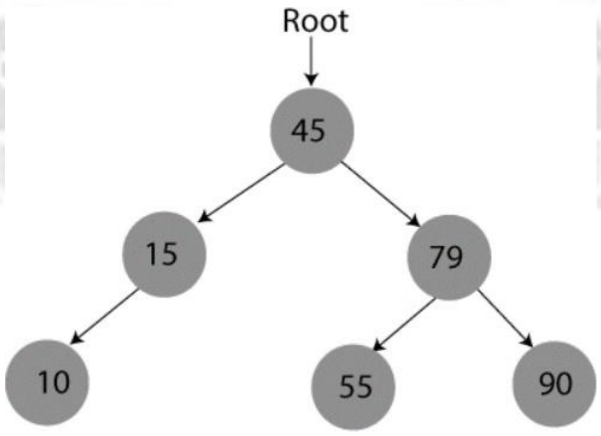
- 90 is greater than 45 and 79, so it will be inserted as the right subtree of 79.



- Step 5 - Insert 10
  - 10 is smaller than 45 and 15, so it will be inserted as a left subtree of 15.

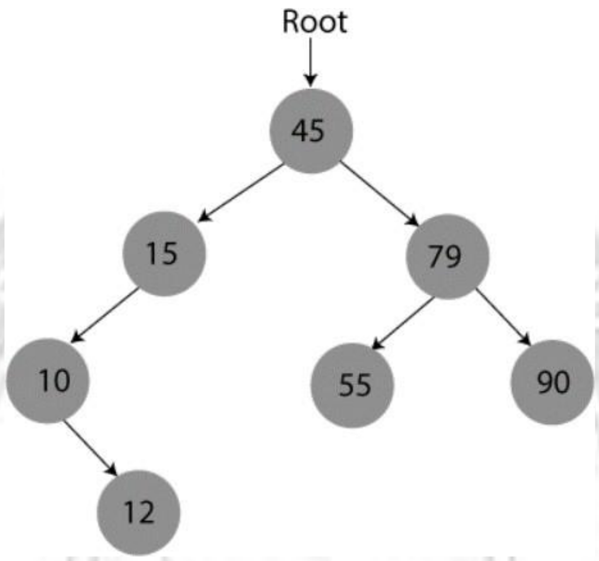


- Step 6 - Insert 55
  - 55 is larger than 45 and smaller than 79, so it will be inserted as the left subtree of 79.

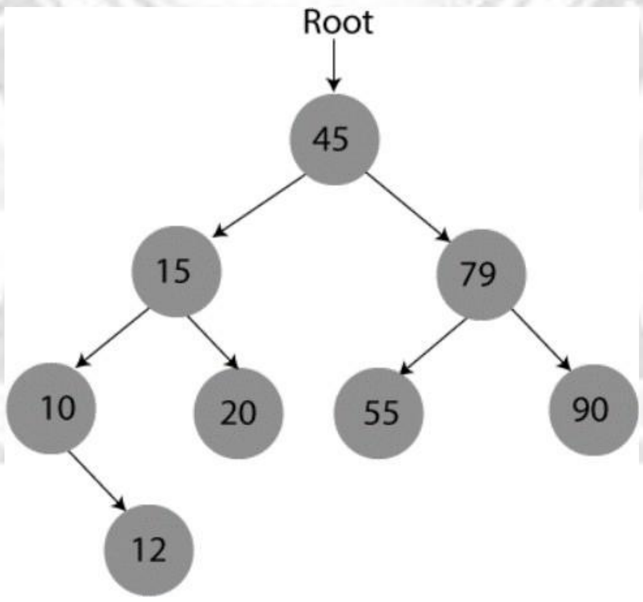


- Step 7 - Insert 12
  - 12 is smaller than 45 and 15 but greater than 10, so it will be inserted as the

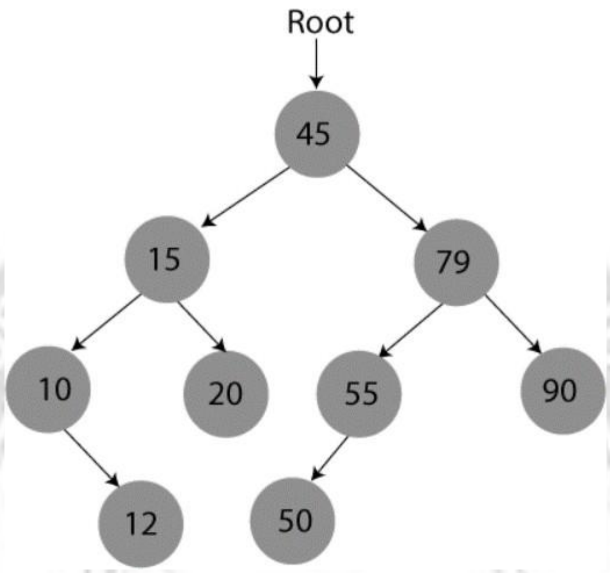
right subtree of 10.



- Step 8 - Insert 20
  - 20 is smaller than 45 but greater than 15, so it will be inserted as the right subtree of 15.



- Step 9 - Insert 50.
  - 50 is greater than 45 but smaller than 79 and 55. So, it will be inserted as a left subtree of 55.



- Now, the creation of binary search tree is completed.

Operations performed on a Binary Search Tree

- We can perform insert, delete and search operations on the binary search tree.

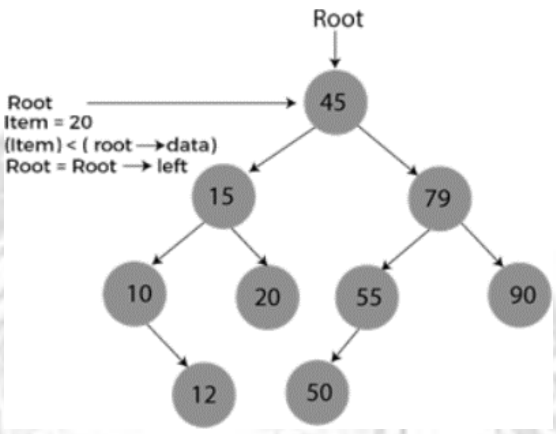
Searching in Binary search tree

- Searching means to find or locate a specific element or node in a data structure. In Binary search tree, searching a node is easy because elements in BST are stored in a specific order.

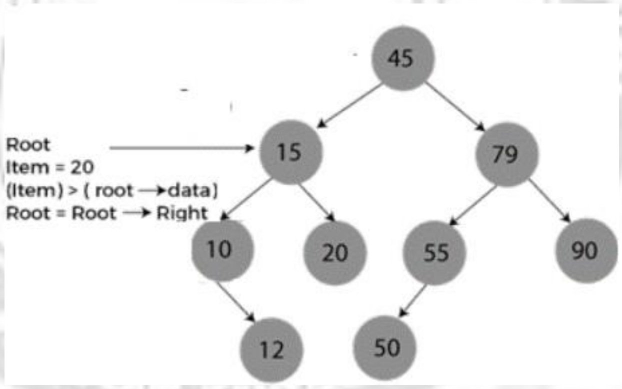
Steps involved in Searching in a Binary Search Tree

- First, compare the element to be searched with the root element of the tree.
- If root is matched with the target element, then return the node's location.
- If it is not matched, then check whether the item is less than the root element, if it is smaller than the root element, then move to the left subtree.
- If it is larger than the root element, then move to the right subtree.
- Repeat the above procedure recursively until the match is found.
- If the element is not found or not present in the tree, then return NULL.
    - Now, let's understand the searching in binary tree using an example. We are taking the binary search tree formed above. Suppose we have to find node 20 from the below tree.

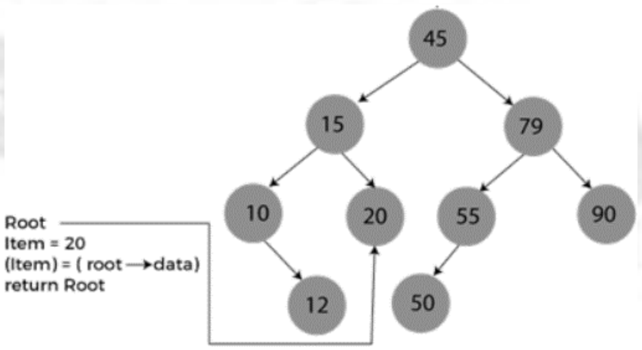
Step1:



Step2:



Step3:



Algorithm to search an element in Binary search tree

Search (root, item)

Step 1 - if (item = root → data) or (root = NULL)

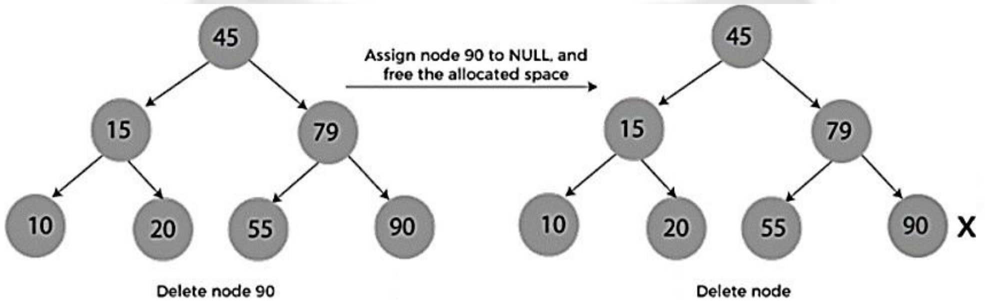
```
        return root
        else if (item < root → data)
        return Search(root → left, item)
        else
        return Search(root → right, item)
        END if
    Step 2 - END
```

Deletion in Binary Search tree

- In a binary search tree, we must delete a node from the tree by keeping in mind that the property of BST is not violated. To delete a node from BST, there are three possible situations occur -
  - The node to be deleted is the leaf node, or,
  - The node to be deleted has only one child, and,
  - The node to be deleted has two children

When the node to be deleted is the leaf node

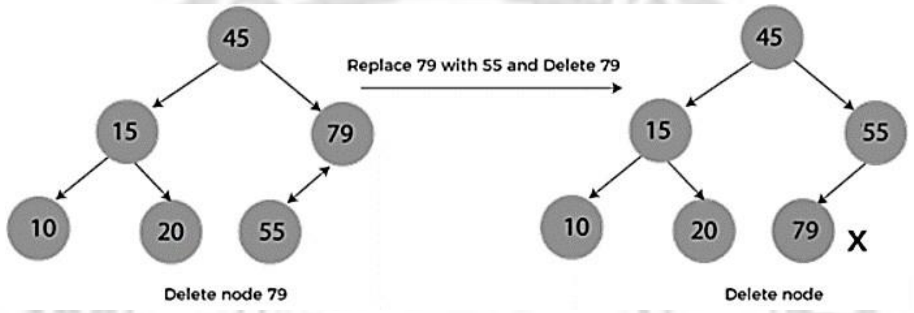
- It is the simplest case to delete a node in BST. Here, we have to replace the leaf node with NULL and simply free the allocated space.
- We can see the process to delete a leaf node from BST in the below image. In below image, suppose we have to delete node 90, as the node to be deleted is a leaf node, so it will be replaced with NULL, and the allocated space will free.



When the node to be deleted has only one child

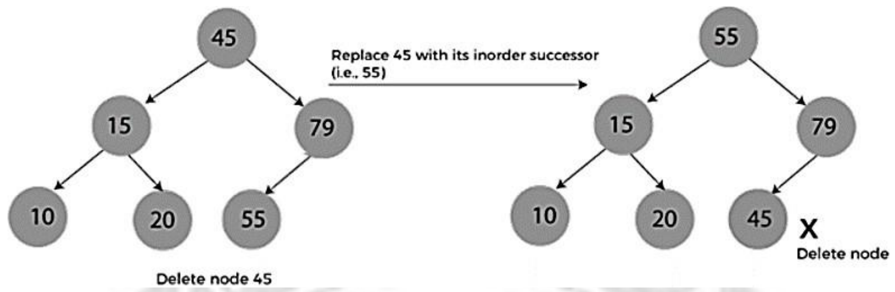
- In this case, we have to replace the target node with its child, and then delete the child node. It means that after replacing the target node with its child node, the child node will now contain the value to be deleted. So, we simply have to replace the child node with NULL and free up the allocated space.

- We can see the process of deleting a node with one child from BST in the below image. In the below image, suppose we have to delete the node 79, as the node to be deleted has only one child, so it will be replaced with its child 55.
- So, the replaced node 79 will now be a leaf node that can be easily deleted.



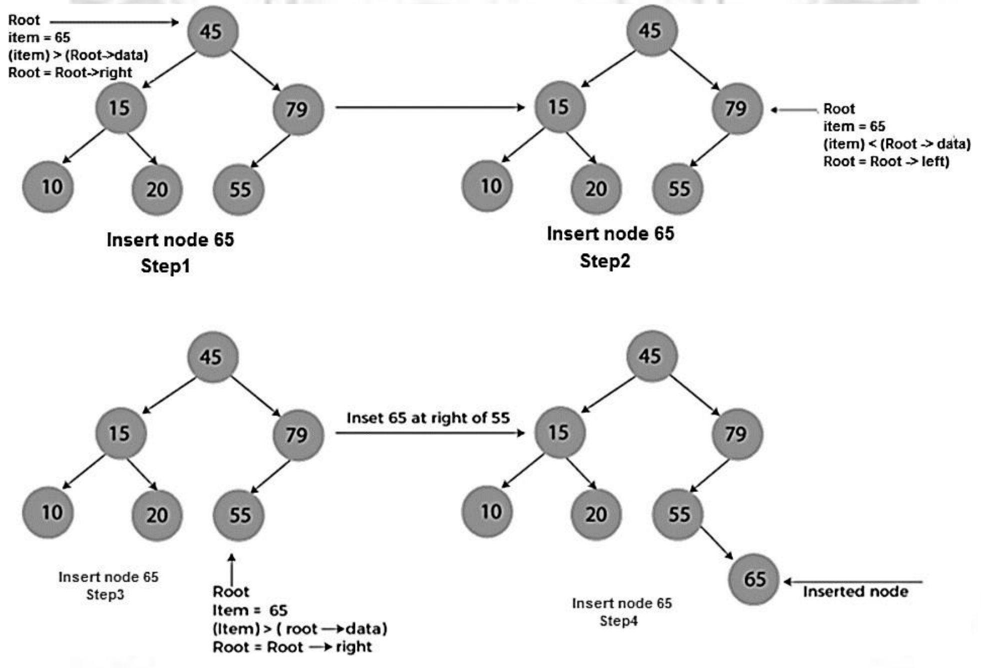
When the node to be deleted has two children

- This case of deleting a node in BST is a bit complex among other two cases. In such a case, the steps to be followed are listed as follows -
  - First, find the inorder successor of the node to be deleted.
  - After that, replace that node with the inorder successor until the target node is placed at the leaf of tree.
  - And at last, replace the node with NULL and free up the allocated space.
- The inorder successor is required when the right child of the node is not empty. We can obtain the inorder successor by finding the minimum element in the right child of the node.
- We can see the process of deleting a node with two children from BST in the below image.
- In the below image, suppose we have to delete node 45 that is the root node, as the node to be deleted has two children, so it will be replaced with its inorder successor. Now, node 45 will be at the leaf of the tree so that it can be deleted easily.



Insertion in Binary Search tree

- A new key in BST is always inserted at the leaf. To insert an element in BST, we have to start searching from the root node; if the node to be inserted is less than the root node, then search for an empty location in the left subtree.
- Else, search for the empty location in the right subtree and insert the data. Insert in BST is similar to searching, as we always have to maintain the rule that the left subtree is smaller than the root, and right subtree is larger than the root.



The complexity of the Binary Search tree

- Let's see the time and space complexity of the Binary search tree. We will see the time complexity for insertion, deletion, and searching operations in best case, average case, and worst case.

Implementation of Binary search tree

```
#include  <iostream>
using       namespace
std;struct Node {
   int data;
   Node *left;
   Node *right;
};
Node* create(int item)
{
   Node* node = new
   Node;node->data = item;
   node->left = node->right = NULL;
   return node;
}
/*Inorder traversal of the tree formed*/
void inorder(Node *root)
{
   if (root == NULL)
      return;
   inorder(root->left); //traverse left subtree
   cout<< root->data << " "; //traverse root node
   inorder(root->right); //traverse right subtree
}
Node* findMinimum(Node* cur) /*To find the inorder successor*/
{
   while(cur->left != NULL) {
      cur = cur->left;
   }
   return cur;
```

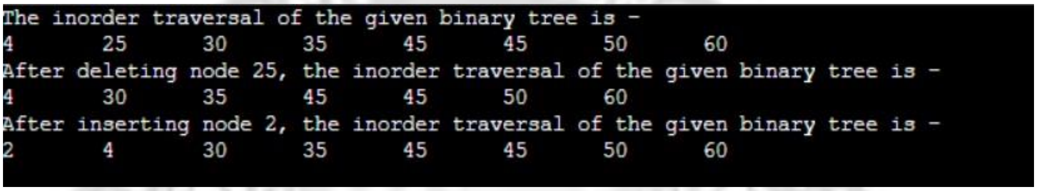
```
}
Node* insertion(Node* root, int item) /*Insert a node*/
{
    if (root == NULL)
        return create(item); /*return new node if tree is empty*/
    if (item < root->data)
        root->left = insertion(root->left, item);
    else
        root->right = insertion(root->right, item);
    return root;
}
void search(Node* &cur, int item, Node* &parent)
{
    while (cur != NULL && cur->data != item)
    {
        parent = cur;
        if (item < cur->data)
            cur = cur->left;
        else
            cur = cur->right;
    }
}
void deletion(Node*& root, int item) /*function to delete a node*/
{
    Node* parent = NULL;
    Node* cur = root;
    search(cur, item, parent); /*find the node to be deleted*/
    if (cur == NULL)

        return;
    if (cur->left == NULL && cur->right == NULL) /*When node has no
children*/
    {
        if (cur != root)
        {
            if (parent->left == cur)
                parent->left = NULL;
            else
                parent->right = NULL;
        }
        else
            root = NULL;
        free(cur);
    }
    else if (cur->left && cur->right)
    {
        Node* succ = findMinimum(cur->right);
        int val = succ->data;
        deletion(root, succ-
        >data);cur->data = val;
    }
    else
    {
        Node* child = (cur->left)? cur->left: cur->right;
        if (cur != root)
        {
            if (cur == parent-
                >left)parent->left =
                child;
            else
                parent->right = child;
        }
        else
            root = child;
        free(cur);
    }
}
int main()
{
  Node* root = NULL;
  root = insertion(root,
  45);root = insertion(root,
  30);root = insertion(root,
  50);root = insertion(root,
  25);root = insertion(root,
  35);root = insertion(root,
  45);root = insertion(root,
  60);root = insertion(root,
  4);
  printf("The inorder traversal of the given binary tree is - \n");
  inorder(root);
  deletion(root, 25);
  printf("\nAfter deleting node 25, the inorder traversal of the given binary tree is
- \n");
  inorder(root);
  insertion(root, 2);
  printf("\nAfter inserting node 2, the inorder traversal of the given binary tree is
- \n");
```

```
    inorder(root);

    return 0;
  }
```

Output



```
The inorder traversal of the given binary tree is -
4       25      30      35      45      45      50      60
After deleting node 25, the inorder traversal of the given binary tree is -
4       30      35      45      45      50      60
After inserting node 2, the inorder traversal of the given binary tree is -
2       4       30      35      45      45      50      60
```