Jadavpur University
Department of Electronics and Telecommunication Engineering,
Faculty of Engineering & Technology

DSA LAB REPORT
2nd Year First Semester 2020

Name : RAHUL SAHA
Roll: 001910701009

Group 1

IMPLEMENTATION OF BINARY TREES AND BINARY SEARCH TREES

Date Of Submission: 13/05/2021

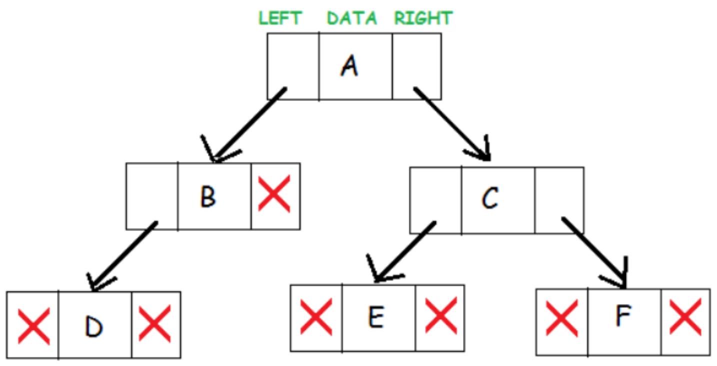
Introduction

Binary Tree

A binary tree is a hierarchical data structure in which each node has at most two children generally referred as left child and right child.
Each node contains three components:

1. Pointer to left subtree
2. Pointer to right subtree
3. Data element

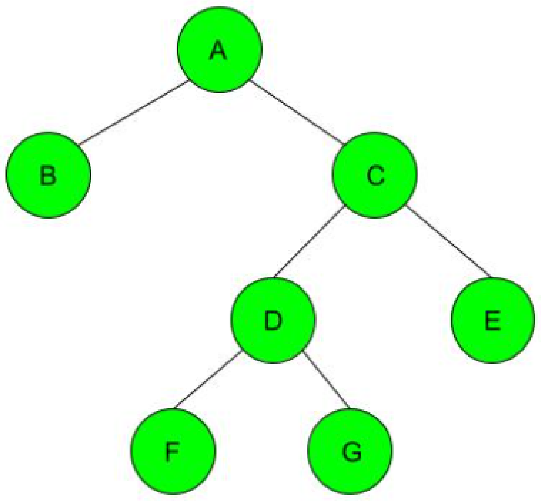
Here is a representation of a binary tree:



Strictly Binary Tree

If every non leaf node in a binary tree has nonempty left and right subtrees, the tree is called a strictly binary tree. A strictly binary tree with n leaves always contains 2n -1 nodes.

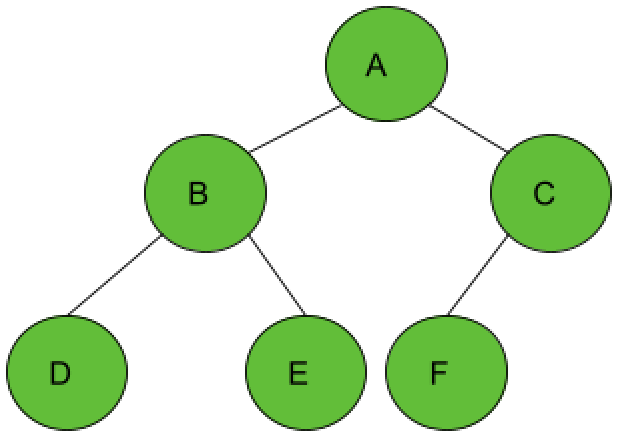
Example:



Complete Binary Tree

A complete binary tree is a binary tree in which all the levels are completely filled except possibly the lowest one, which is filled from the left.

Example:



Source Code:

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<stdbool.h>

//structure defining the tree
typedef struct node {
  int item;
  struct node *left, *right;
}Tree;

Tree *root=NULL;

//creates new node
Tree *create_node(int num)
{
        Tree *new_node;
        new_node=(Tree *)malloc(sizeof(Tree));
        new_node->item = num;
        new_node->left = new_node->right = NULL;
        return new_node;
};

//inserts as left child
void insert_left(Tree *new_root,int val)
{
          new_root->left=create_node(val);
}

//inserts as right child
void insert_right(Tree *new_root,int val){
          new_root->right=create_node(val);
}

//search for the root node given as input from the user
bool search_node(Tree *temp,int user_root, char a, int num){
  if(temp == NULL)
   return 0;
  if(temp->item == user_root  &&(temp->left==NULL || temp->right==NULL))
    {
     if(a == 'l' || a == 'L')//for left child
       {
         if(temp->left == NULL)
          {
           insert_left(temp, num);
           return 1;
          }
         else
           return 0;
       }
     else
       {
         if(temp->right == NULL)//for right child
          {
            insert_right(temp, num);
            return 1;
          }
        else
          return 0;
       }

    }
  else
   return search_node(temp->left, user_root, a, num) || search_node(temp->right,
user_root, a, num);
}
```

```
//creates the tree
void create_tree(){
	char a;
	printf("\n\nEnter the node in format specified:");
	int num,user_root;
	scanf("%d %d %c",&num,&user_root,&a);
	bool found = 0;
	if(a=='l'||a=='L'){
		found = search_node(root,user_root,a,num);

	}
	else if(a=='r'||a=='R'){
		found =search_node(root,user_root,a,num);
	}
  if(!found)
   printf("The parent node doesn't exist\n");
}

//checkes if the tree is stricty binary
bool check_strict_binary(Tree *root) {
	if (root == NULL)// Check if tree is empty
		return true;

	if (root->left == NULL && root->right == NULL)// Check if children are present
		return true;

	if ((root->left) && (root->right))
		return (check_strict_binary(root->left) &&
check_strict_binary(root->right));

	return false;
}

//checkes if the tree is complete binary
bool check_complete_binary(Tree *root, int index, int numberNodes) {
	if (root == NULL)// Check if the tree is complete
		return true;

	if (index >= numberNodes)
		return false;

	return (check_complete_binary(root->left, 2 * index + 1, numberNodes) &&
check_complete_binary(root->right, 2 * index + 2, numberNodes));
}
```

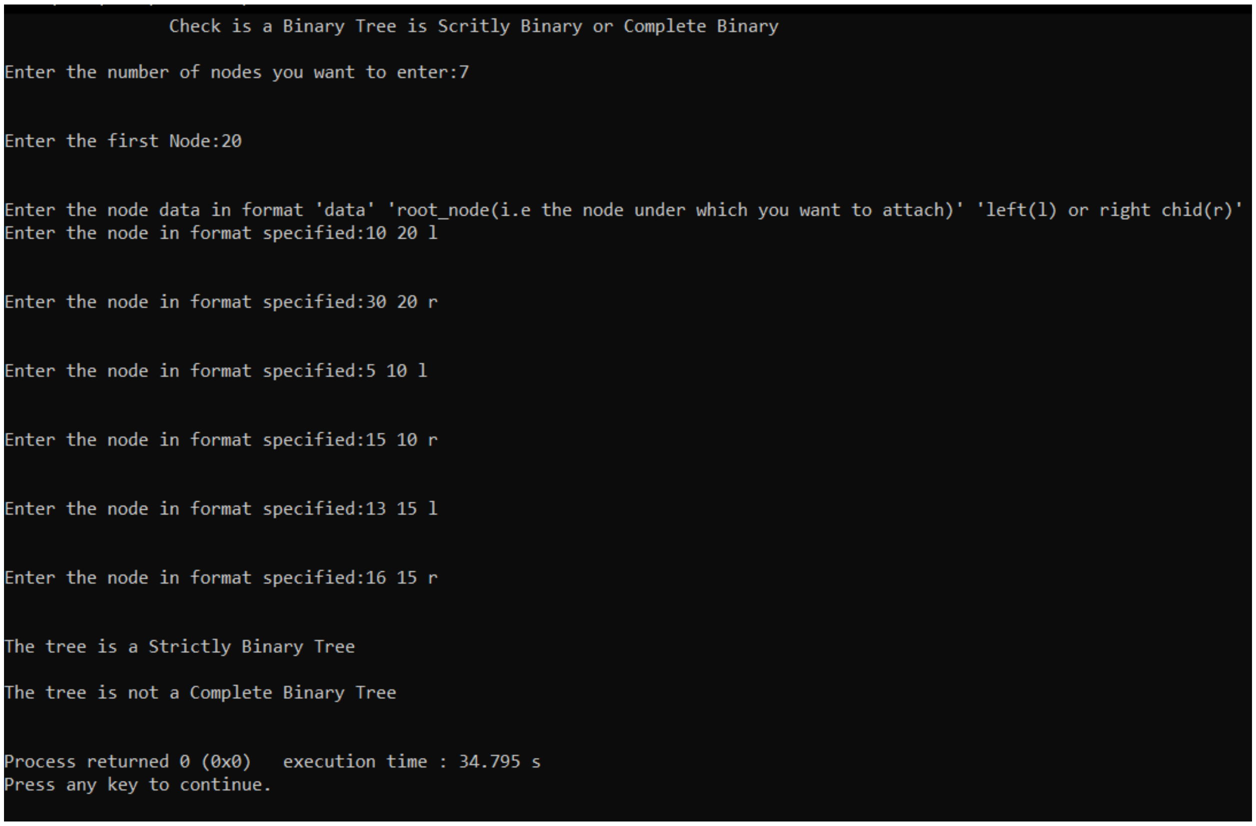
```c
//driver code
int main(){
	system("cls");
	printf("\t\tCheck is a Binary Tree is Scritly Binary or Complete Binary\n\n");
	int val,num_nodes,index=0;
	printf("Enter the number of nodes you want to enter:");
	scanf("%d",&num_nodes);
	printf("\n\nEnter the first Node:");
	scanf("%d",&val);
	root=create_node(val);
	printf("\n\nEnter the node data in format 'data' 'root_node(i.e the node under
which you want to attach)' 'left(l) or right chid(r)'");
	for(int i=1;i<num_nodes;i++){
		create_tree();
	}
	if(check_strict_binary(root)){
        printf("\n\nThe tree is a Strictly Binary Tree");
    }
    else{
        printf("\n\nThe tree is not a Strictly Binary Tree");
    }
    if(check_complete_binary(root,index,num_nodes)){
        printf("\n\nThe tree is a Complete Binary Tree");
    }
    else{
        printf("\n\nThe tree is not a Complete Binary Tree\n\n");
    }
    return 0;
}
```

OUTPUT CONSOLE

As soon as the root node is entered, all the other nodes will be entered in the following format.

data, parent node, left child(l) ***or*** *right child(r)*

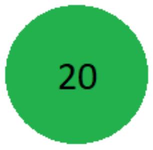
I. To check if the tree is strictly binary or not



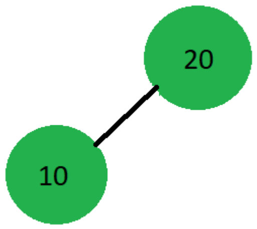
```
                Check is a Binary Tree is Scritly Binary or Complete Binary

Enter the number of nodes you want to enter:7


Enter the first Node:20


Enter the node data in format 'data' 'root_node(i.e the node under which you want to attach)' 'left(l) or right chid(r)'
Enter the node in format specified:10 20 l


Enter the node in format specified:30 20 r


Enter the node in format specified:5 10 l


Enter the node in format specified:15 10 r


Enter the node in format specified:13 15 l


Enter the node in format specified:16 15 r


The tree is a Strictly Binary Tree

The tree is not a Complete Binary Tree


Process returned 0 (0x0)   execution time : 34.795 s
Press any key to continue.
```

Steps of creating this binary tree

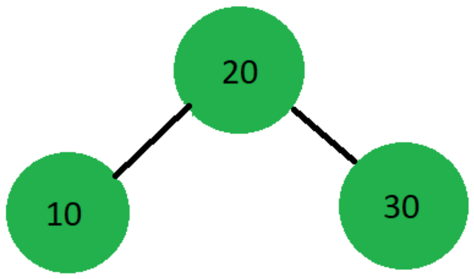
Step 1:



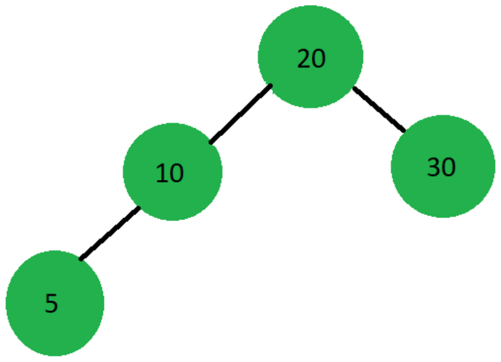
Step 2:



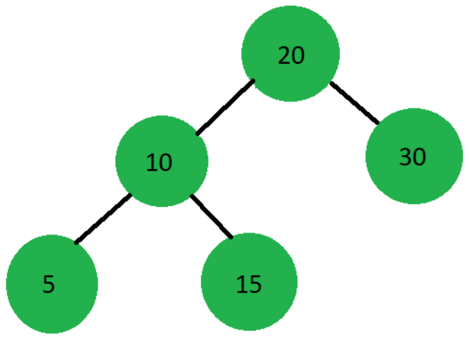
Step 3:



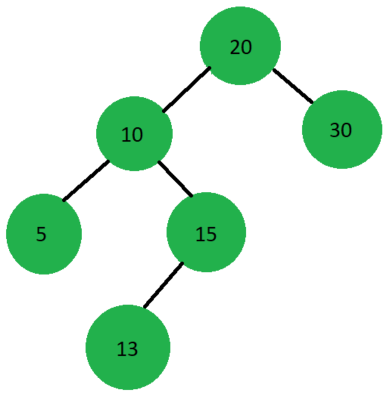
Step 4:



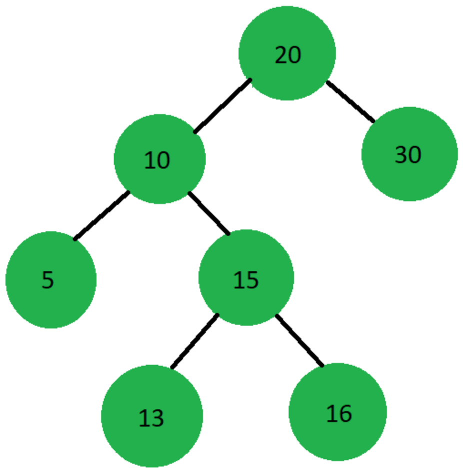
Step 5:



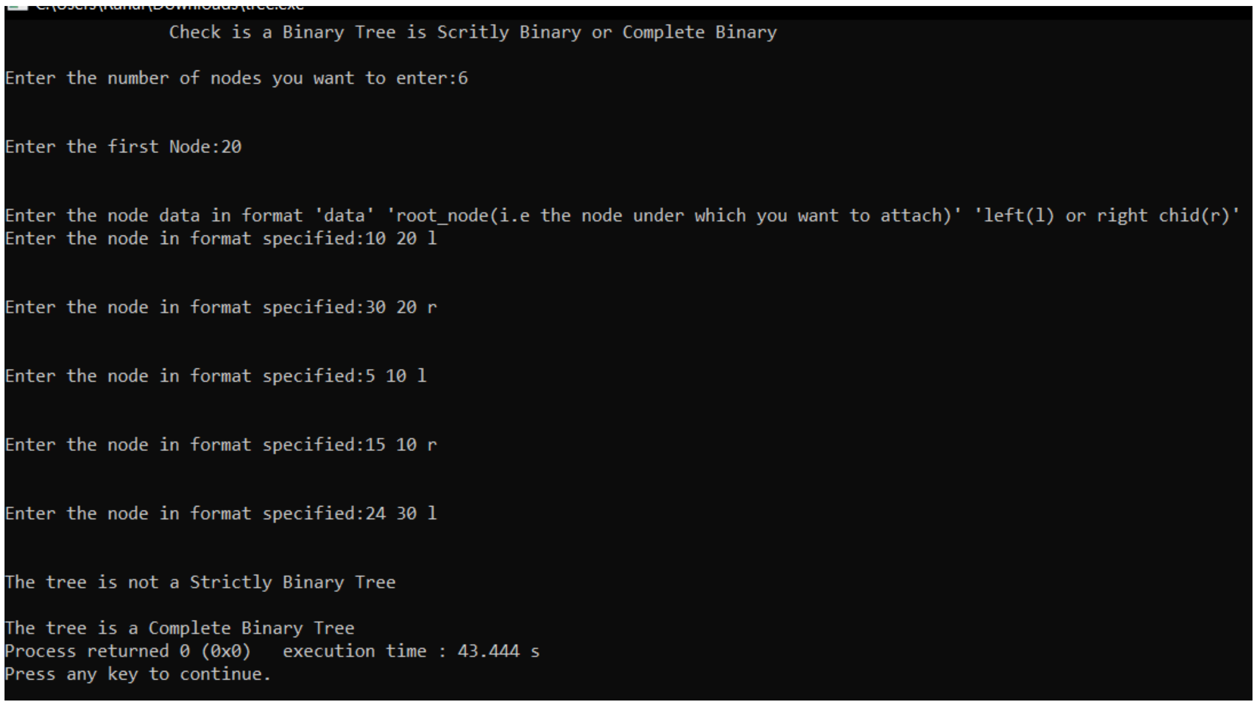
Step 6:



Step 7:



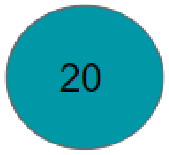
II. Checking if the tree is a complete binary tree



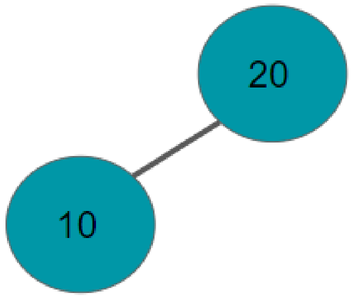
```
                Check is a Binary Tree is Scritly Binary or Complete Binary

Enter the number of nodes you want to enter:6


Enter the first Node:20


Enter the node data in format 'data' 'root_node(i.e the node under which you want to attach)' 'left(l) or right chid(r)'
Enter the node in format specified:10 20 l


Enter the node in format specified:30 20 r


Enter the node in format specified:5 10 l


Enter the node in format specified:15 10 r


Enter the node in format specified:24 30 l


The tree is not a Strictly Binary Tree

The tree is a Complete Binary Tree
Process returned 0 (0x0)   execution time : 43.444 s
Press any key to continue.
```

Steps of creating this binary tree

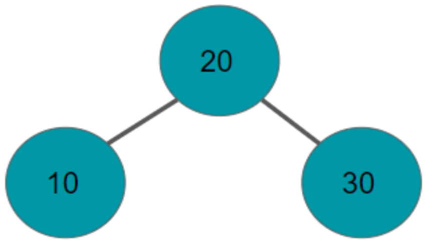
Step1:



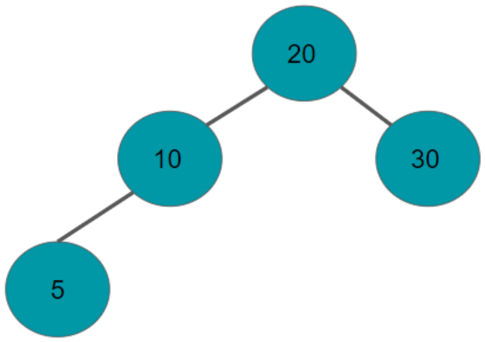
Step 2:



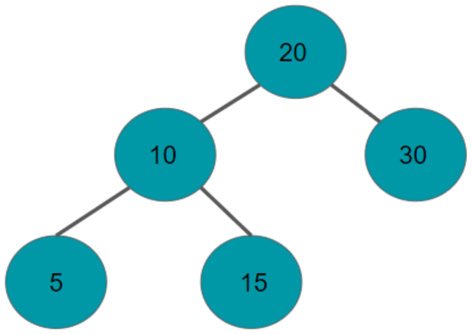
Step 3:



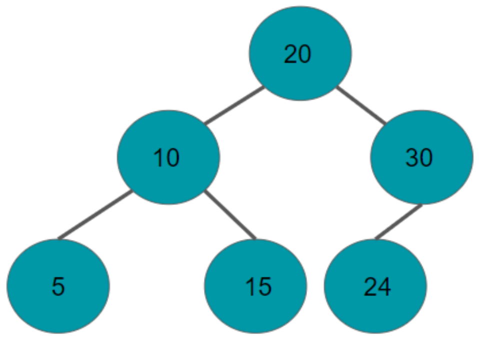
Step 4:



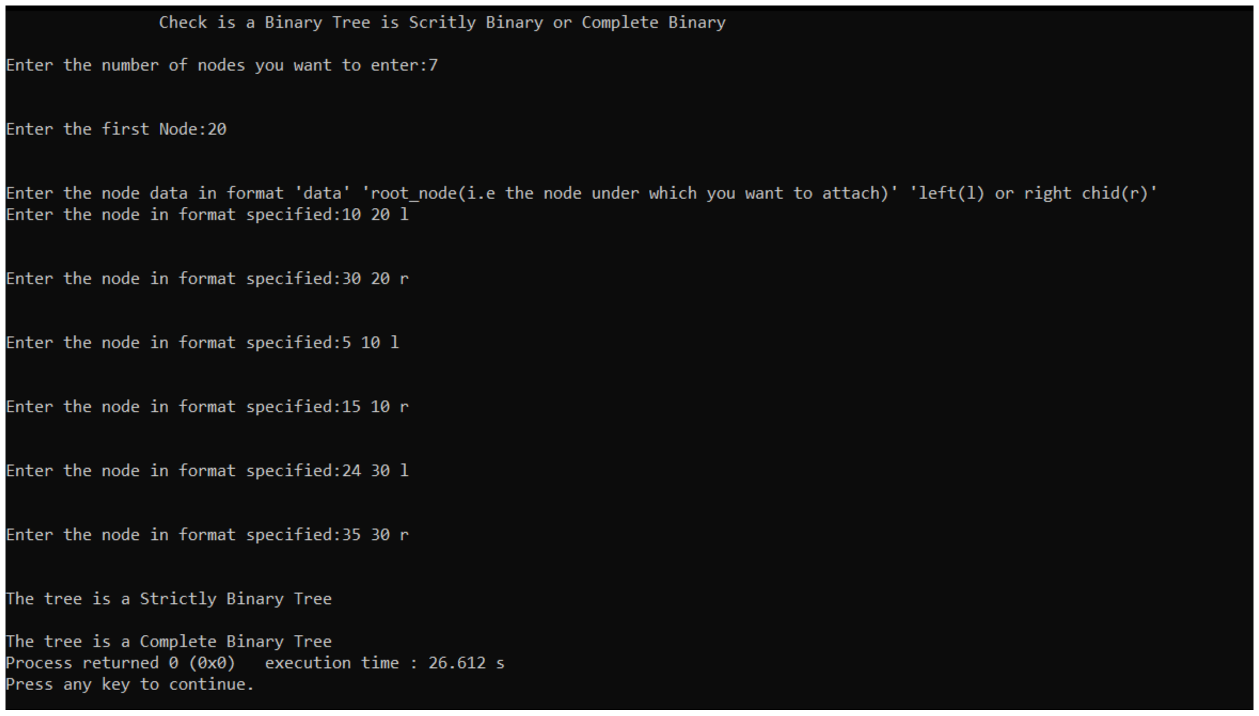
Step 5:



Step 6:



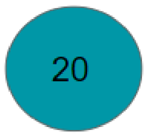
III. Checking whether a binary tree is both strictly binary tree and complete binary tree



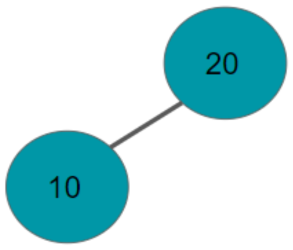
```
                Check is a Binary Tree is Scritly Binary or Complete Binary

Enter the number of nodes you want to enter:7


Enter the first Node:20


Enter the node data in format 'data' 'root_node(i.e the node under which you want to attach)' 'left(l) or right chid(r)'
Enter the node in format specified:10 20 l


Enter the node in format specified:30 20 r


Enter the node in format specified:5 10 l


Enter the node in format specified:15 10 r


Enter the node in format specified:24 30 l


Enter the node in format specified:35 30 r


The tree is a Strictly Binary Tree

The tree is a Complete Binary Tree
Process returned 0 (0x0)   execution time : 26.612 s
Press any key to continue.
```

Steps of creating this binary tree

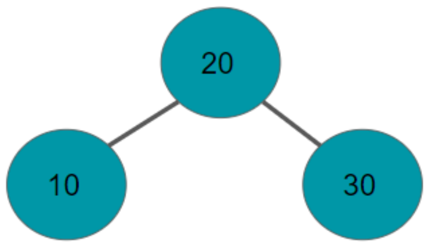
Step 1:



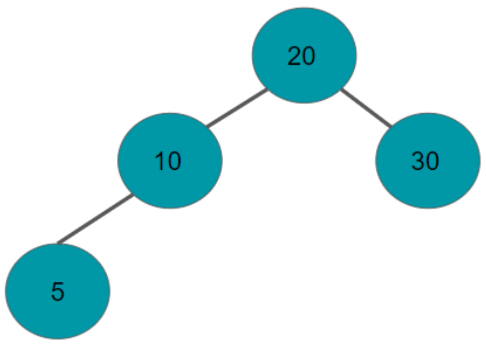
Step 2:



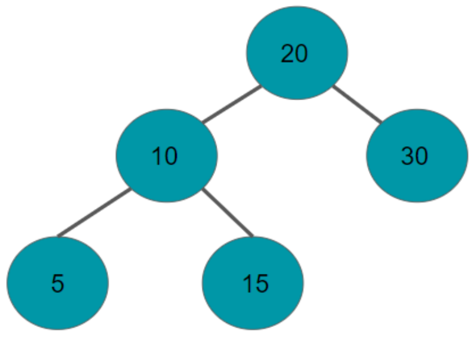
Step 3:



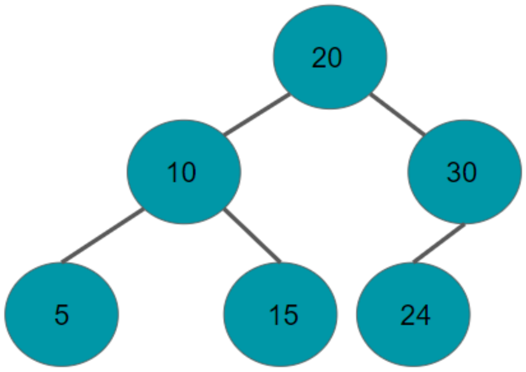
Step 4:



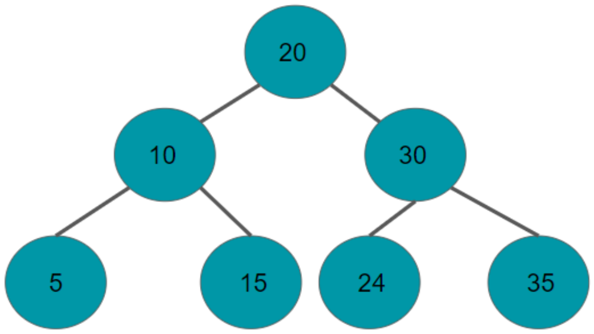
Step 5:



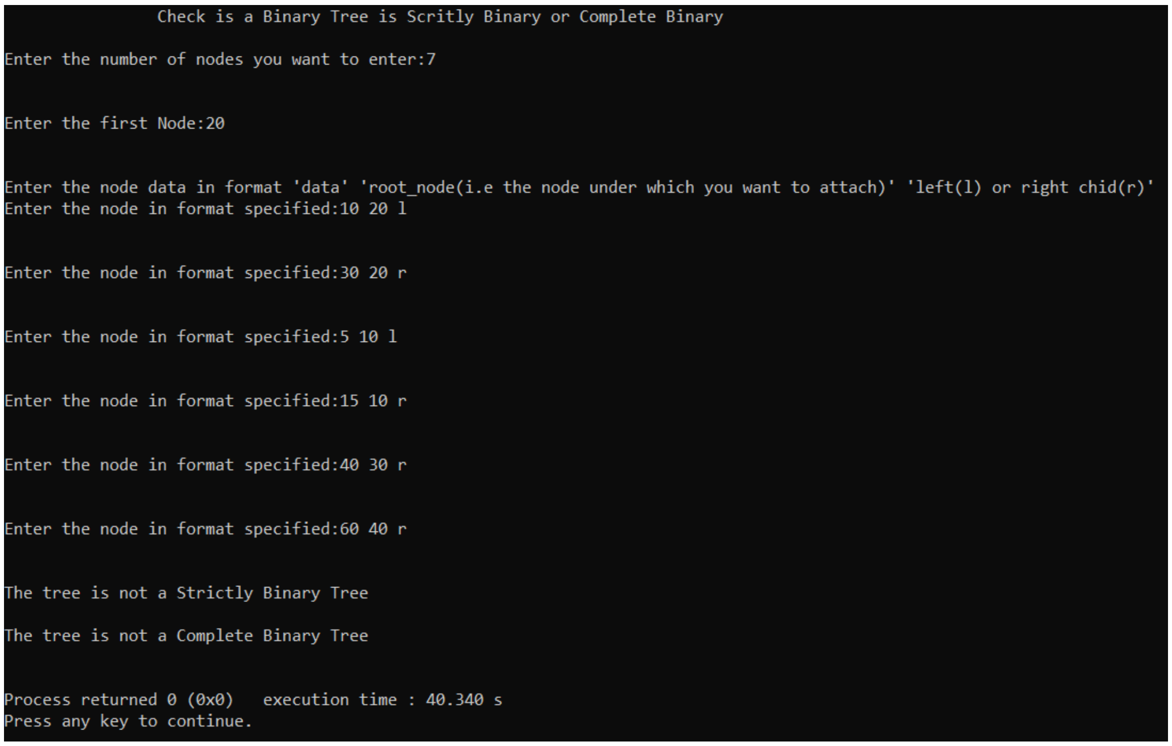
Step 6:



Step 7:



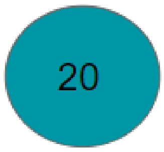
IV. Checking whether the binary tree is neither a complete binary tree nor a strictly binary tree.



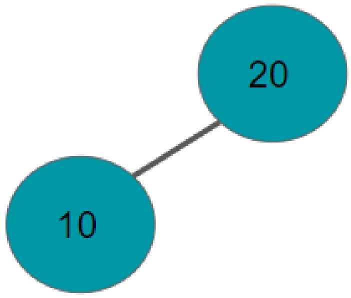
```
                Check is a Binary Tree is Scritly Binary or Complete Binary

Enter the number of nodes you want to enter:7


Enter the first Node:20


Enter the node data in format 'data' 'root_node(i.e the node under which you want to attach)' 'left(l) or right chid(r)'
Enter the node in format specified:10 20 l


Enter the node in format specified:30 20 r


Enter the node in format specified:5 10 l


Enter the node in format specified:15 10 r


Enter the node in format specified:40 30 r


Enter the node in format specified:60 40 r


The tree is not a Strictly Binary Tree

The tree is not a Complete Binary Tree


Process returned 0 (0x0)   execution time : 40.340 s
Press any key to continue.
```

Steps for creating this binary tree

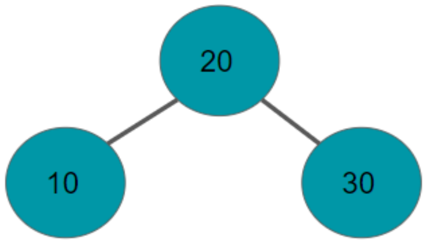
Step 1:



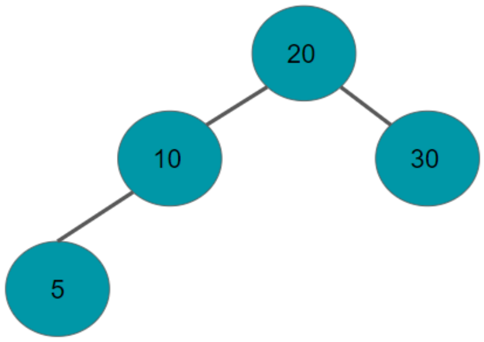
Step 2:



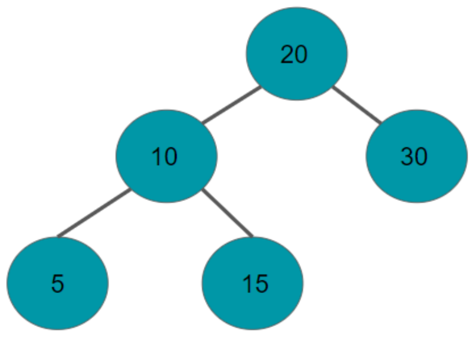
Step 3:



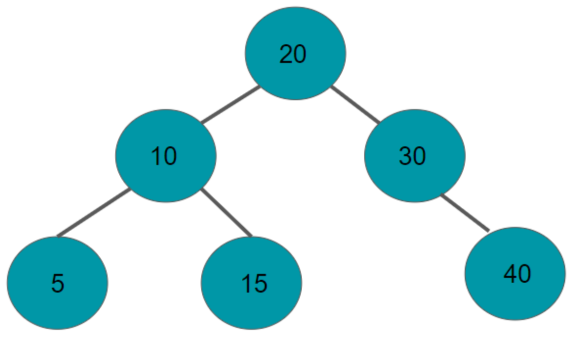
Step 4:



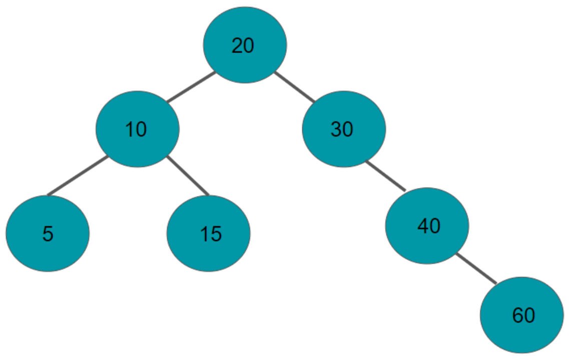
Step 5:



Step 6:



Step 7:

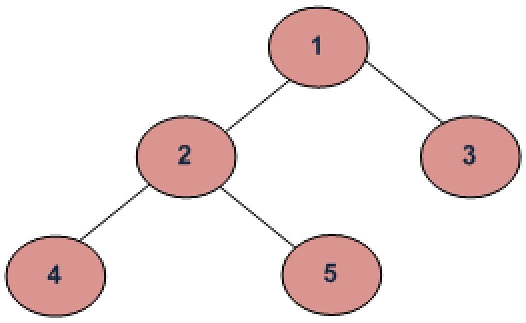


Tree Traversal

Traversing a tree means visiting every node in the tree. Linear data structures like arrays, stacks, queues, and linked lists have only one way to read the data. But a hierarchical data structure like a tree can be traversed in different ways.

There are basically three traversal techniques for a binary tree. We will be showing the algorithms for each traversal.

Inorder Traversal:



```
Algorithm for Inorder Traversal:

1. Traverse the left subtree, i.e., call Inorder(left-subtree)
2. Visit the root.
3. Traverse the right subtree, i.e., call Inorder (right-  subtree)
```

Uses of Inorder

In case of binary search trees (BST), Inorder traversal gives nodes in non-decreasing order. To get nodes of BST in non-increasing order, a variation of Inorder traversal where Inorder traversal is reversed can be used.

Example: Inorder traversal for the above-given figure is 4 2 5 1 3.

Pre Order Traversal:

```
Algorithm for Preorder Traversal:

   1. Visit the root.
   2. Traverse the left subtree, i.e., call Preorder(left-subtree)
   3. Traverse the right subtree, i.e., call Preorder(right-subtree)
```

Uses of Preorder

Preorder traversal is used to create a copy of the tree. Preorder traversal is also used to get a prefix expression on of an expression tree.

Example: Preorder traversal for the above given figure is 1 2 4 5 3.

Post Order Traversal

```
Algorithm for Postorder Traversal:

   1. Traverse the left subtree, i.e., call Postorder(left-subtree)
   2. Traverse the right subtree, i.e., call Postorder(right-subtree)
   3. Visit the root.
```

Uses of Postorder

Postorder traversal is used to delete the tree. Postorder traversal is also useful to get the postfix expression of an expression tree.

Example: Postorder traversal for the above given figure is 4 5 2 3 1.

Source code

```
#include <stdio.h>
#include<conio.h>
#include<stdlib.h>
#define max 100

//structure defining the tree
typedef struct node {
  int item;
  struct node *left, *right;
}Tree;

Tree *root=NULL;
//creates new node
Tree *create_node(int num)
{
     Tree *new_node;
     new_node=(Tree *)malloc(sizeof(Tree));
     new_node->item = num;
     new_node->left = new_node->right = NULL;
     return new_node;
};

//inserts as left child
void insert_left(Tree *new_root,int val)
{
     new_root->left=create_node(val);
}

//inserts as right child
void insert_right(Tree *new_root,int val){
     new_root->right=create_node(val);
}

//search for the root node given as input from the user
bool search_node(Tree *temp,int user_root, char a, int num){
  if(temp == NULL)
   return 0;
  if(temp->item == user_root  &&(temp->left==NULL ||
temp->right==NULL))
    {
     if(a == 'l' || a == 'L')//for left child
       {
         if(temp->left == NULL)
          {
           insert_left(temp, num);
           return 1;
          }
         else
           return 0;
       }
     else
       {
         if(temp->right == NULL)//for right child
          {
            insert_right(temp, num);
            return 1;
          }
        else
          return 0;
       }

    }
  else
```

```
   return search_node(temp->left, user_root, a, num) ||
search_node(temp->right, user_root, a, num);
}

//creates the tree
void create_tree(){
      char a;
      printf("\n\nEnter the node in format specified:");
      int num,user_root;
      scanf("%d %d %c",&num,&user_root,&a);
      bool found = 0;
      if(a=='l'||a=='L'){
            found = search_node(root,user_root,a,num);

      }
      else if(a=='r'||a=='R'){
            found =search_node(root,user_root,a,num);
      }
  if(!found)
   printf("The parent node doesn't exist\n");
}

//Funtion for Inorder traversal
void inorder_traversal(Tree* root) {
    if (root == NULL)
        return;
    inorder_traversal(root->left);
    printf("%d ", root->item);
    inorder_traversal(root->right);
}

//Funtion For Preorder Traversal
void preorder_traversal(Tree* root) {
    if (root == NULL)
        return;
    printf("%d ", root->item);
    preorder_traversal(root->left);
    preorder_traversal(root->right);
}

//Function for Postorder Traversal
void postorder_traversal(Tree* root) {
    if (root == NULL)
        return;
    postorder_traversal(root->left);
    postorder_traversal(root->right);
    printf("%d ", root->item);
}

// Driver code
int main()
{
    system("cls");
    printf("\t\tInorder,Preorder,Postorder traversal\n\n");
    int val,num_nodes,index=0;
    printf("Enter the number of nodes you want to enter:");
    scanf("%d",&num_nodes);
    printf("\n\nEnter the first Node:");
    scanf("%d",&val);
    root=create_node(val);
    printf("\n\nEnter the node data in format 'data' 'parent_node(i.e
the node under which you want to attach)' 'left(l) or right
chid(r)'");
    for(int i=1;i<num_nodes;i++){
      create_tree();
    }

    printf("\n\nInorder traversal \n");
    inorder_traversal(root);

    printf("\n\nPreorder traversal \n");
    preorder_traversal(root);

    printf("\n\nPostorder traversal \n");
    postorder_traversal(root);
}
```

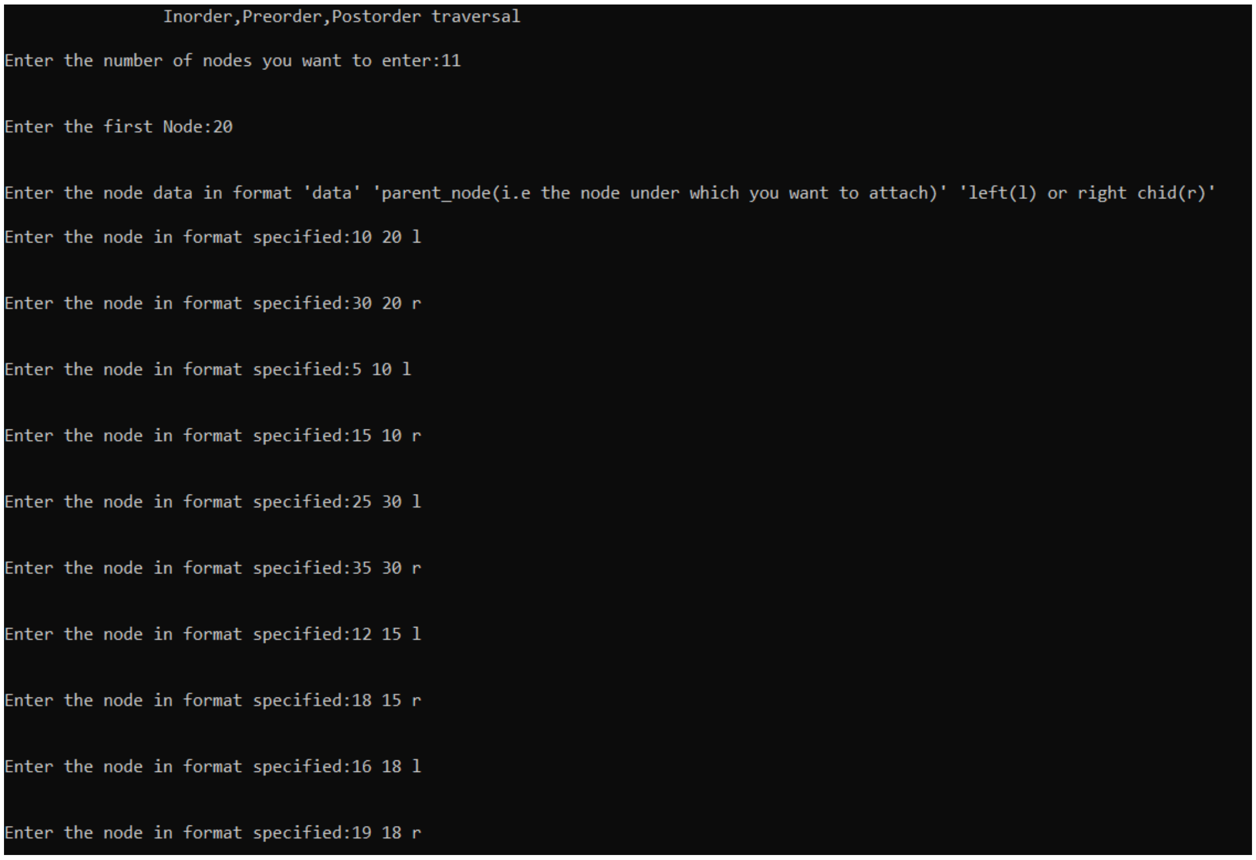
Output Console

In the output console,

1. Specify the number of nodes
2. Write down the root node
3. Write the non-root nodes in the following format

```
<root> <parent_node> <left (l) / right (r) >
```

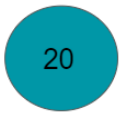
Run - I



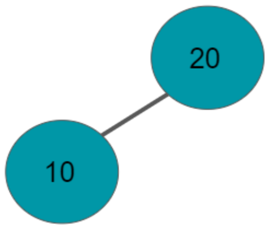
```
                Inorder,Preorder,Postorder traversal

Enter the number of nodes you want to enter:11


Enter the first Node:20


Enter the node data in format 'data' 'parent_node(i.e the node under which you want to attach)' 'left(l) or right chid(r)'

Enter the node in format specified:10 20 l


Enter the node in format specified:30 20 r


Enter the node in format specified:5 10 l


Enter the node in format specified:15 10 r


Enter the node in format specified:25 30 l


Enter the node in format specified:35 30 r


Enter the node in format specified:12 15 l


Enter the node in format specified:18 15 r


Enter the node in format specified:16 18 l


Enter the node in format specified:19 18 r
```

Steps for creating this binary tree:

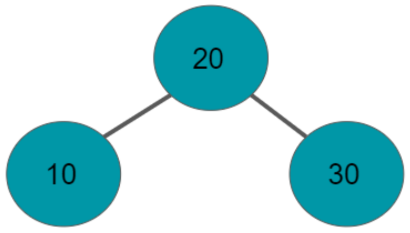
Step 1:



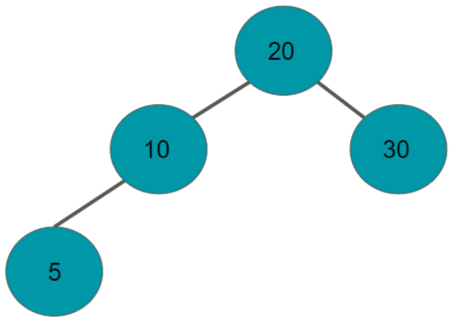
Step 2:



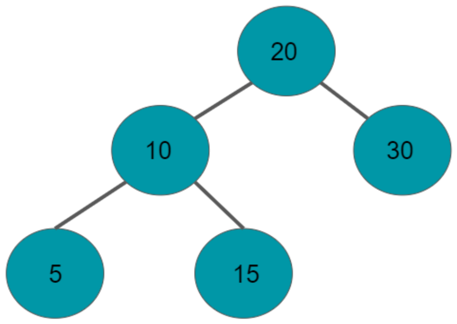
Step 3:



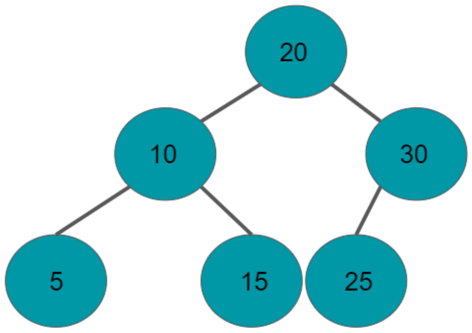
Step 4:



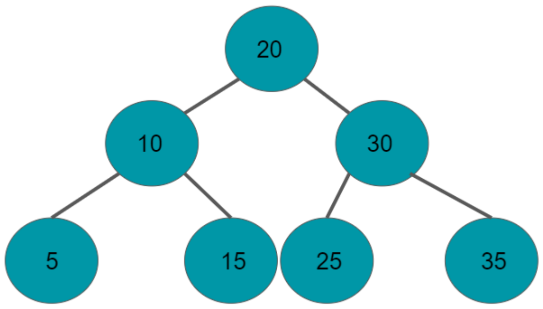
Step 5:



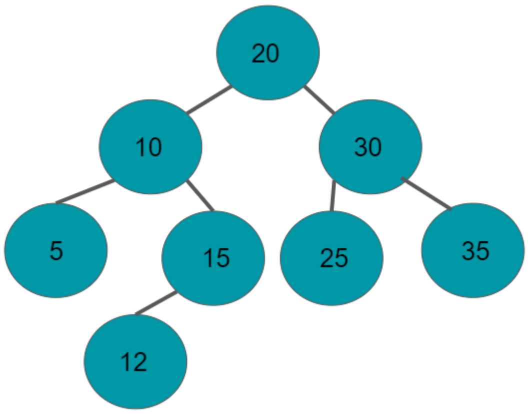
Step 6:



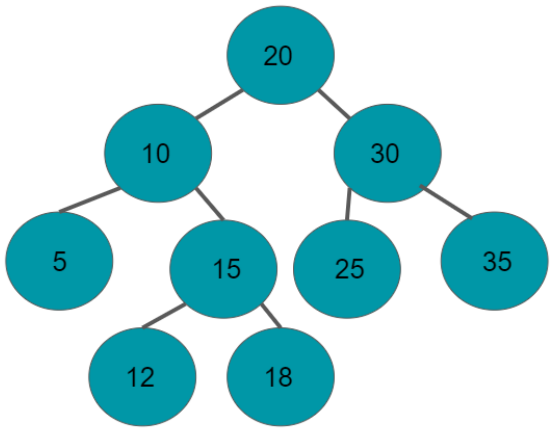
Step 7:



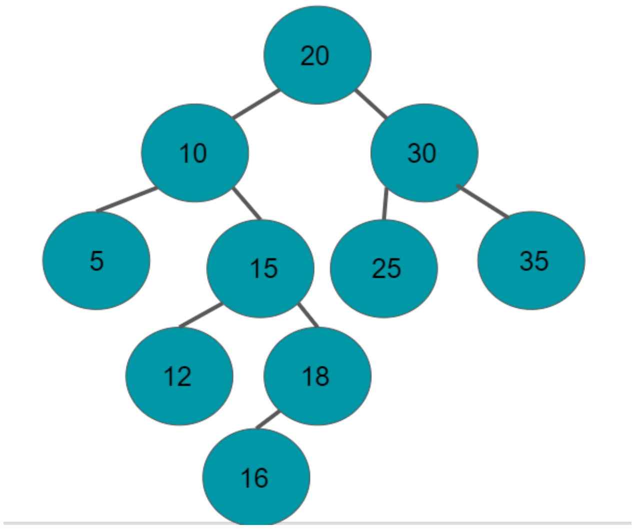
Step 8:



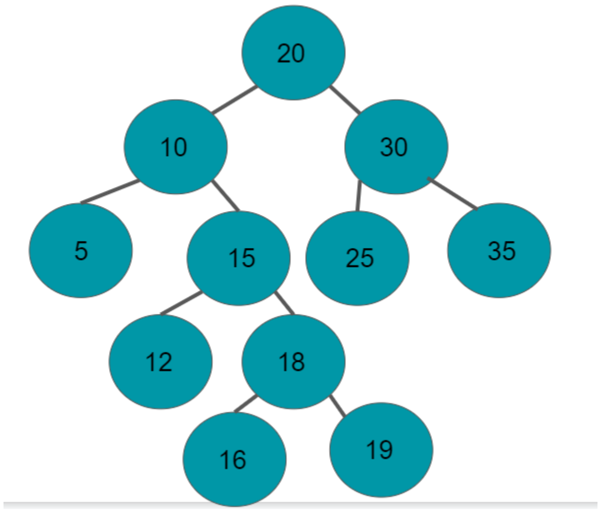
Step 9:



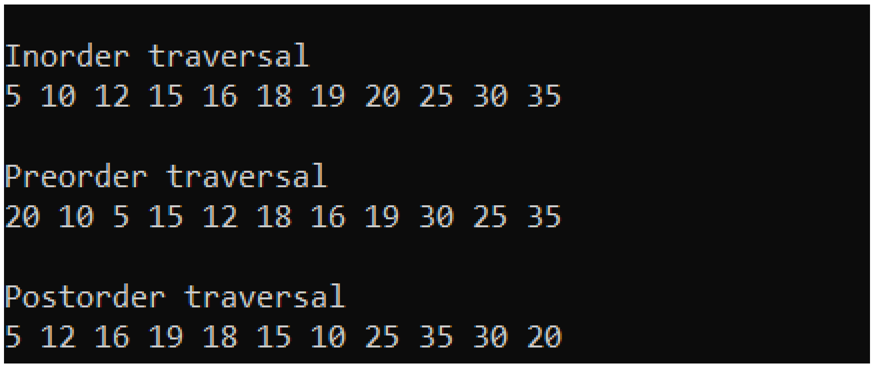
Step 10:



Step 11:

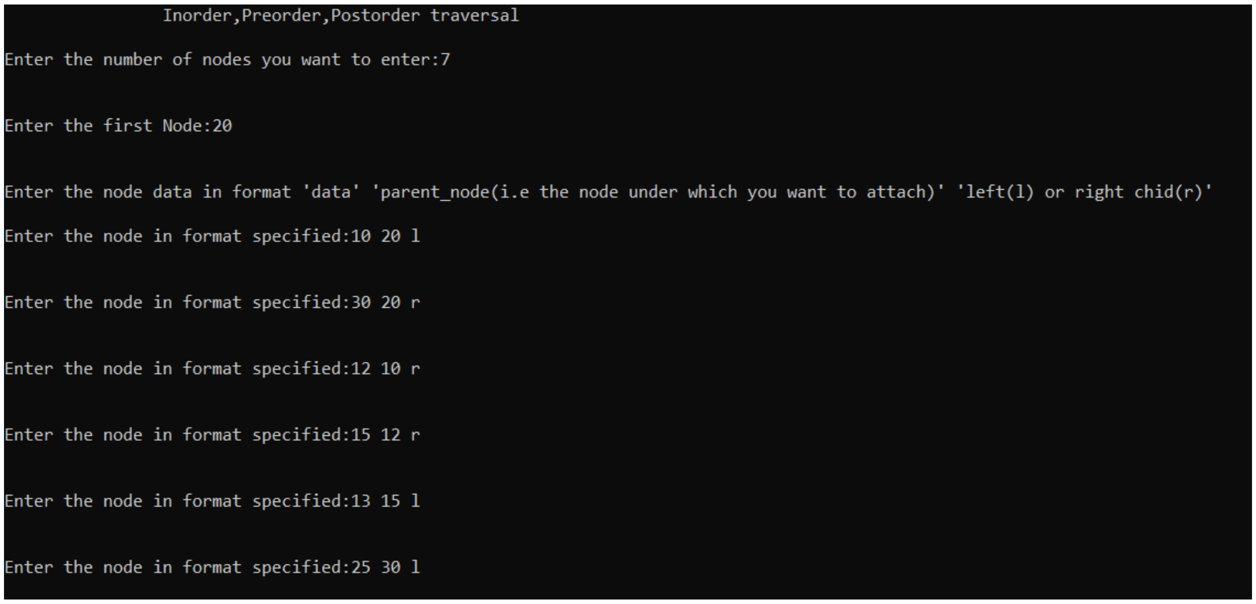


RESULTS



```
Inorder traversal
5 10 12 15 16 18 19 20 25 30 35

Preorder traversal
20 10 5 15 12 18 16 19 30 25 35

Postorder traversal
5 12 16 19 18 15 10 25 35 30 20
```

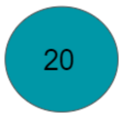
RUN - II



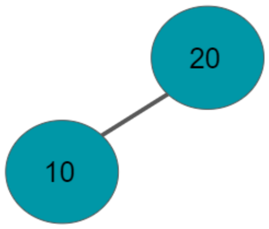
```
                Inorder,Preorder,Postorder traversal

Enter the number of nodes you want to enter:7


Enter the first Node:20


Enter the node data in format 'data' 'parent_node(i.e the node under which you want to attach)' 'left(l) or right chid(r)'

Enter the node in format specified:10 20 l


Enter the node in format specified:30 20 r


Enter the node in format specified:12 10 r


Enter the node in format specified:15 12 r


Enter the node in format specified:13 15 l


Enter the node in format specified:25 30 l
```

Steps for creating this tree:

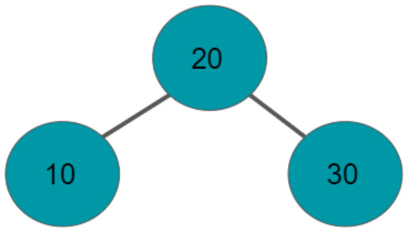
Step 1:



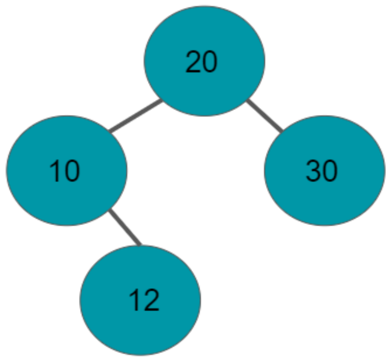
Step 2:



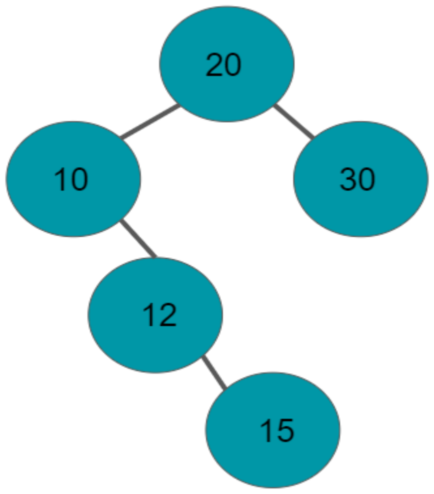
Step 3:



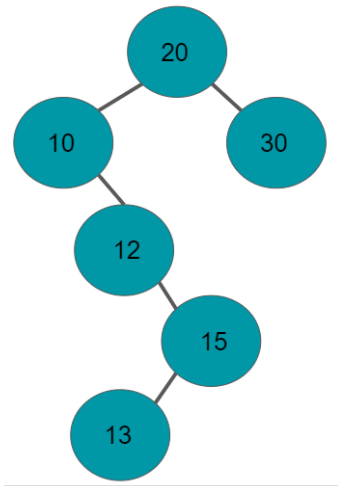
Step 4:



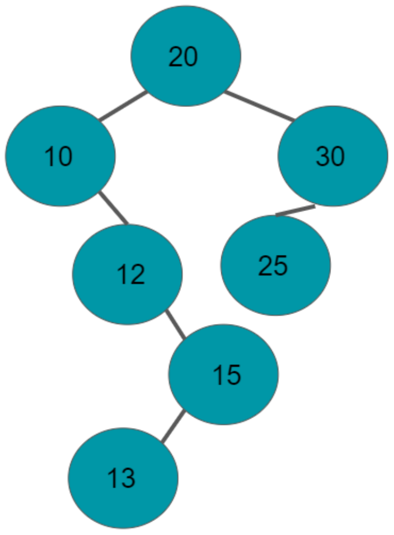
Step 5:



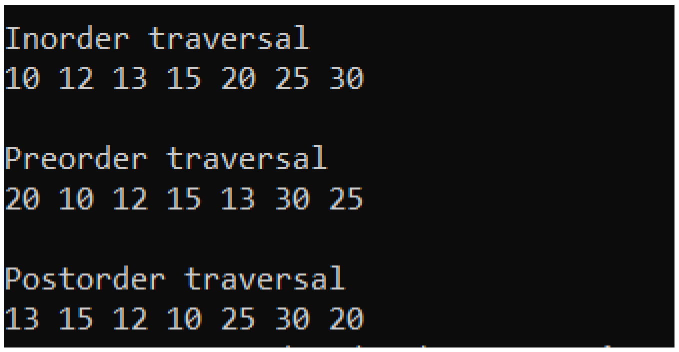
Step 6:



Step 7:



RESULTS



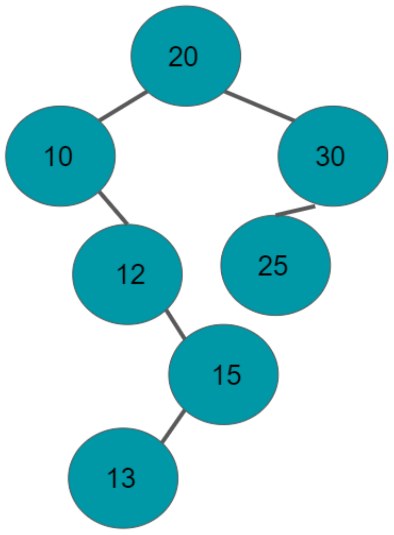
```
Inorder traversal
10 12 13 15 20 25 30

Preorder traversal
20 10 12 15 13 30 25

Postorder traversal
13 15 12 10 25 30 20
```

3. Based on a random set of inputted integers create a binary search tree with the integers as the nodes. Delete at least a) one leaf node, b) a node with one child and c) a node with two children from the tree you have created.

In a binary search tree,

1. All nodes of left subtrees are less than the root node.
2. All nodes of right subtrees are more than the root node.
3. The sub-subtrees, sub-sub-subtrees and so on until leaf nodes obeys the above two properties.

Example



Source Code:

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>

typedef struct node {
  int item;
  struct node *left, *right;
}Tree;

Tree* root = NULL;
//creates new node
Tree *create_node(int num){
        Tree *new_node;
```

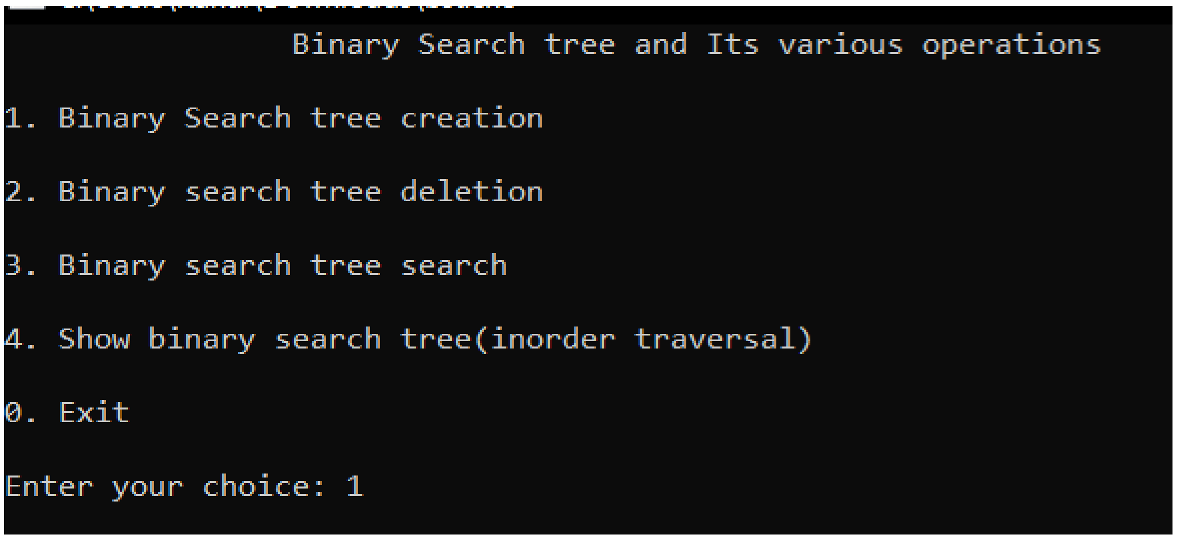
```
          new_node=(Tree *)malloc(sizeof(Tree));
          new_node->item = num;
          new_node->left = new_node->right = NULL;
          return new_node;
};

//Funtion for Inorder traversal
void inorder_traversal(Tree* root) {
    if (root == NULL)
        return;
    inorder_traversal(root->left);
    printf("%d ", root->item);
    inorder_traversal(root->right);
}

Tree* insert_node(Tree* root, int val){
    if (root == NULL) //If the tree is empty, return a new,single node
        return create_node(val);
    if (val < root->item)
        root->left = insert_node(root->left, val);
    else
        root->right = insert_node(root->right, val);
    return root;//return the (unchanged) root pointer

}

//Function to find the inorder successor
struct node *inorder_successor(Tree *node) {
  Tree*current = node;
  while (current && current->left != NULL)//Finds the leftmost leaf
    current = current->left;
  return current;
}

Tree* delete_node(Tree *root,int val){
    if(root==NULL){
        printf("\n\nValue not found");
        return root;
    }
    if(val < root->item)
        root->left=delete_node(root->left,val);
    else if(val > root->item)
        root->right=delete_node(root->right,val);
    else{
        if(root->left==NULL && root->right==NULL){//Node with no elements
```

```
            free(root);
            return NULL;
        }
        else if(root->left == NULL) {//Node with one element
            Tree* temp = root->right;
            free(root);
            return temp;
        }
        else if (root->right == NULL) {//Node with one element
            Tree* temp = root->left;
            free(root);
            return temp;
        }
        //Node with more than two children
        Tree* temp = inorder_successor(root->right);
        root->item = temp->item;//Placing the inorder successor in position of the node to
be deleted
        root->right = delete_node(root->right, temp->item);//Deleting the inorder
successor
    }
  return root;
}

void search(Tree *root, int val, Tree *parent){
    if(root==NULL){
        printf("\n\nValue not found");
        return;
    }
    if(val==root->item){
        if(parent==NULL)
            printf("\n\nThe node with value %d is root node",val);
        else if(val<parent->item)
            printf("\n\nThe node node with value %d is the left child of the parent node
%d",val,parent->item);
        else
            printf("\n\nThe node node with value %d is the right child of the parent node
%d",val,parent->item);
        return;
    }
    else if(val<root->item)
        return search(root->left,val,root);
    return(search(root->right,val,root));
}

void create_tree(){
    system("cls");
    int val;
    char ch;
    printf("\t\tBinary search tree creation");
    do{
        printf("\n\nEnter the node value:");
        scanf("%d",&val);
        root=insert_node(root,val);
        printf("\nThe inorder traversal of the tree is------>  ");
        inorder_traversal(root);
        printf("\nDo you want to enter more nodes(press 'Y' for yes and 'N' for no):");
        fflush(stdin);
        scanf("%c",&ch);
    }while(ch=='Y'||ch=='y');
}
int main(){
   int ch;
    do{
        system("cls");
        int val;
        Tree* node=NULL;
        printf("\t\tBinary Search tree and Its various operations\n\n");
        printf("1. Binary Search tree creation\n\n");
        printf("2. Binary search tree deletion\n\n");
        printf("3. Binary search tree search\n\n");
        printf("4. Show binary search tree(inorder traversal)\n\n");
        printf("0. Exit\n\n");
        printf("Enter your choice: ");
        scanf("%d",&ch);
        switch (ch)
        {
        case 1:
            create_tree();
            break;
        case 2: system("cls");
            printf("\t\tBinary Search Tree Deletion operation\n\n");
            printf("Enter the value of the Node to be deleted: ");
            scanf("%d",&val);
            printf("\n\nThe Binary Search Tree before Deletion---->");
            inorder_traversal(root);
            delete_node(root,val);
            printf("\n\nThe Binary Search Tree after Deletion---->");
            inorder_traversal(root);
            getch();
            break;
```

```
      case 3:  system("cls");
         printf("\t\tBinary Search Tree Searching operation\n\n");
         printf("Enter the value of the Node to be Searched: ");
         scanf("%d",&val);
          search(root,val,NULL);
          getch();
          break;
      case 4:
          system("cls");
          printf("\t\tDisplaying the Binary search tree\n\n");
          inorder_traversal(root);
          getch();
          break;
      case 0: exit(0);
          break;
      default: printf("Enter valid choice");
      }

   }while(1);
}
```

OUTPUT CONSOLE:

Menu template

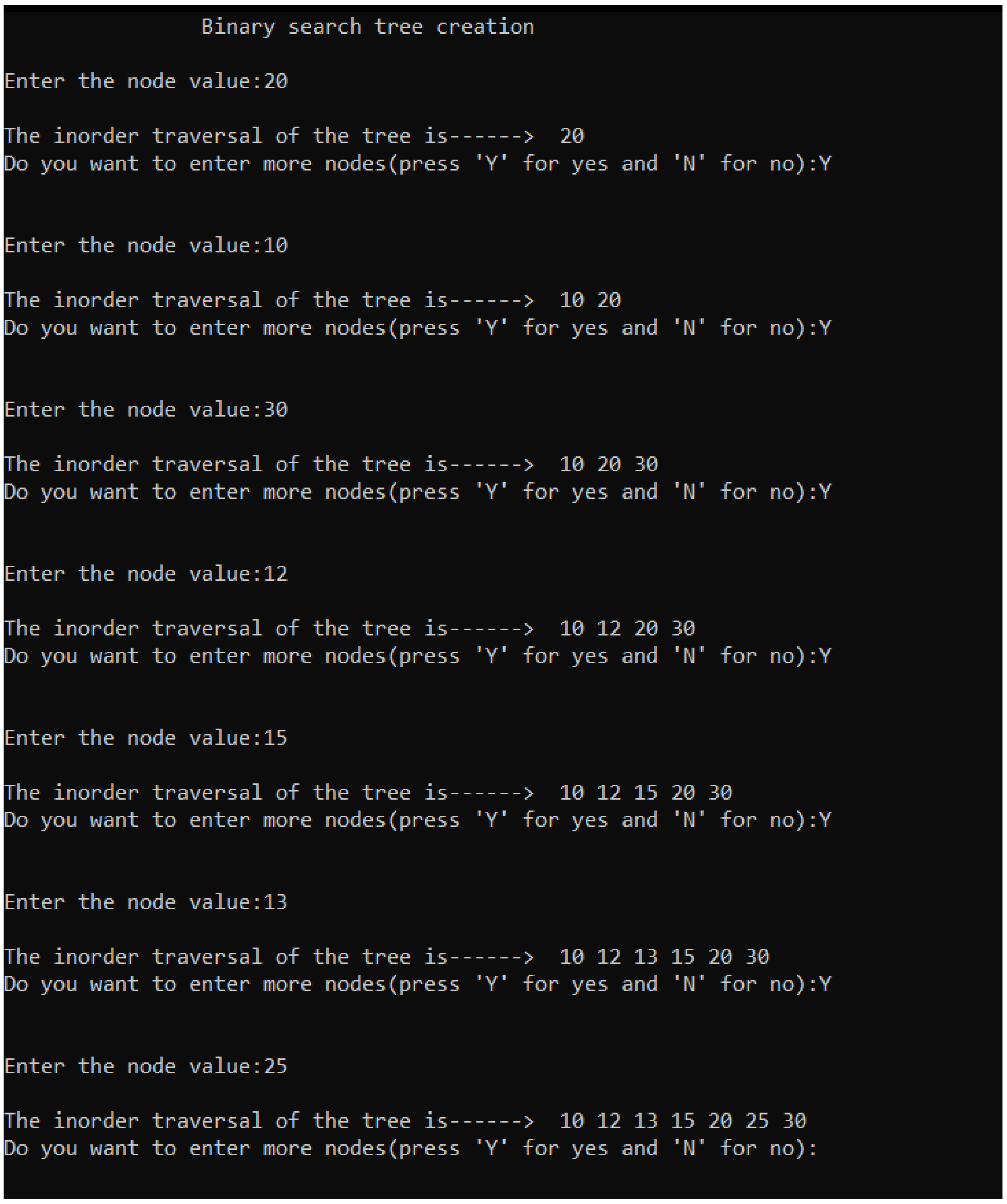


```
                Binary Search tree and Its various operations

1. Binary Search tree creation

2. Binary search tree deletion

3. Binary search tree search

4. Show binary search tree(inorder traversal)

0. Exit

Enter your choice: 1
```

<u>Inserting elements to a binary search tree</u>

```
Algorithm:
1. Start from the root.
2. Compare the inserting element with root, if less than root, then
recurse for left, else recurse for right.
3. After reaching the end, just insert that node at left(if less than
current) else right.
```

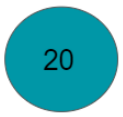
```
Procedure:
1. We need to enter the elements we want in our binary search tree.
2. After an element is inserted in the binary search tree a prompt will be
generated if we want to enter more elements.
3. Press Y to indicate yes and enter more elements or press N to indicate
no and stop entering element.
4. Inorder traversal for each step is shown.
```



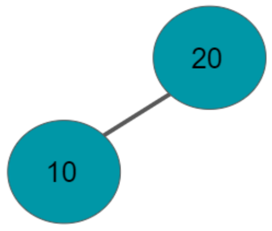
```
                Binary search tree creation

Enter the node value:20

The inorder traversal of the tree is------>  20
Do you want to enter more nodes(press 'Y' for yes and 'N' for no):Y


Enter the node value:10

The inorder traversal of the tree is------>  10 20
Do you want to enter more nodes(press 'Y' for yes and 'N' for no):Y


Enter the node value:30

The inorder traversal of the tree is------>  10 20 30
Do you want to enter more nodes(press 'Y' for yes and 'N' for no):Y


Enter the node value:12

The inorder traversal of the tree is------>  10 12 20 30
Do you want to enter more nodes(press 'Y' for yes and 'N' for no):Y


Enter the node value:15

The inorder traversal of the tree is------>  10 12 15 20 30
Do you want to enter more nodes(press 'Y' for yes and 'N' for no):Y


Enter the node value:13

The inorder traversal of the tree is------>  10 12 13 15 20 30
Do you want to enter more nodes(press 'Y' for yes and 'N' for no):Y


Enter the node value:25

The inorder traversal of the tree is------>  10 12 13 15 20 25 30
Do you want to enter more nodes(press 'Y' for yes and 'N' for no):
```

Steps on how this tree has been created

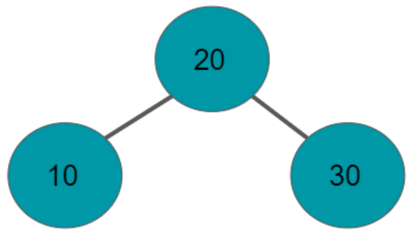
Step 1: Element 20 has been inserted



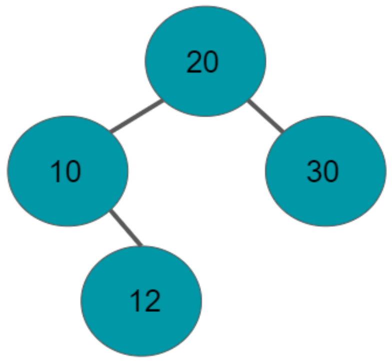
Step 2: Element 10 is inserted into the binary search tree. Since 10 is less than 20 it will be added as a left child to 20.



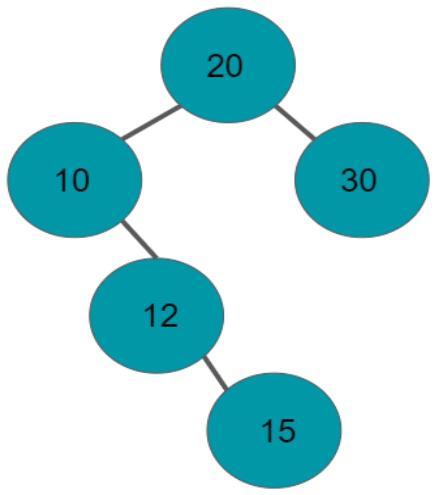
Step 3: Element 30 is inserted into the binary search tree. Since 30 is more than 20 it will be added as a right child to 20.



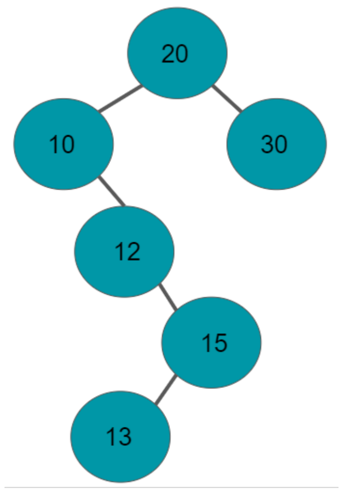
Step 4: Element 12 is inserted into the binary search tree. Since 12 is more than 10 but less than 20 it will be added as a right child to 10.



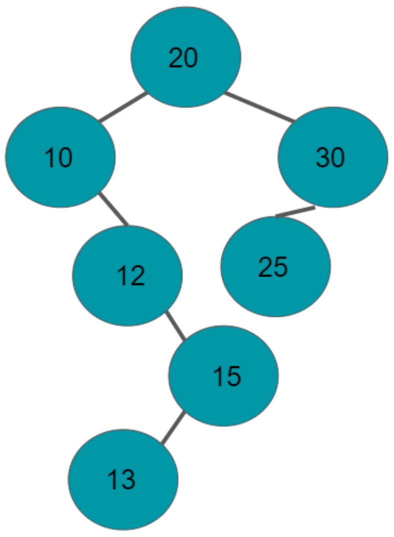
Step 5: Element 15 is inserted into the binary search tree. Since 15 is more than 12 and 10 but less than 20 it will be added as a right child to 12.



Step 6: Element 13 is inserted into the binary search tree. Since 13 is more than 12 and 10 but less than 15 it will be added as a left child to 15.

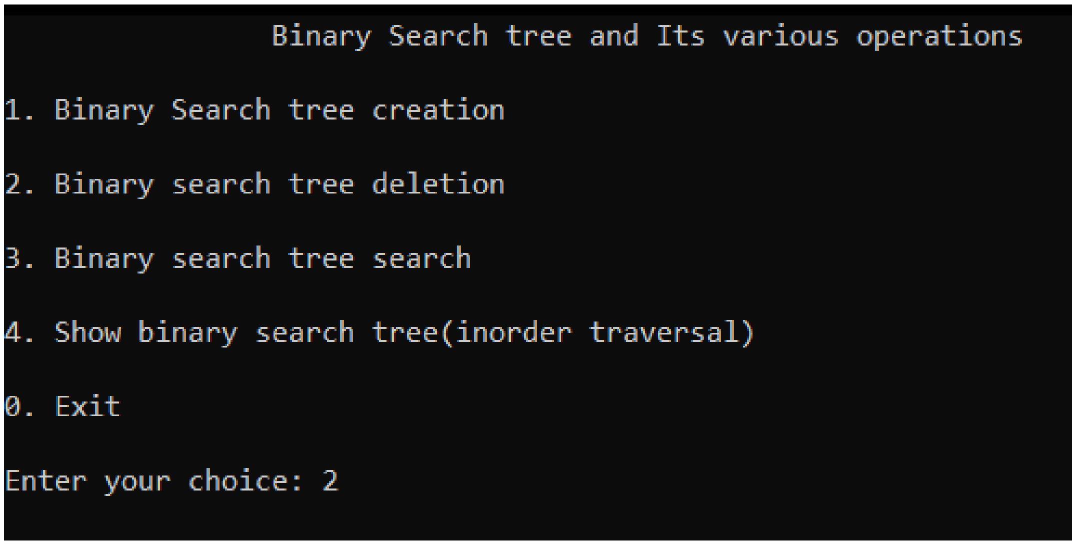


Step 7: Element 25 is inserted into the binary search tree. Since 25 is more than 20 but less than 30 it will be added as a left child to 30.



Deleting elements from the binary search tree

To delete a node from the binary search tree we will select the option 2 from the menu



```
                Binary Search tree and Its various operations

1. Binary Search tree creation

2. Binary search tree deletion

3. Binary search tree search

4. Show binary search tree(inorder traversal)

0. Exit

Enter your choice: 2
```

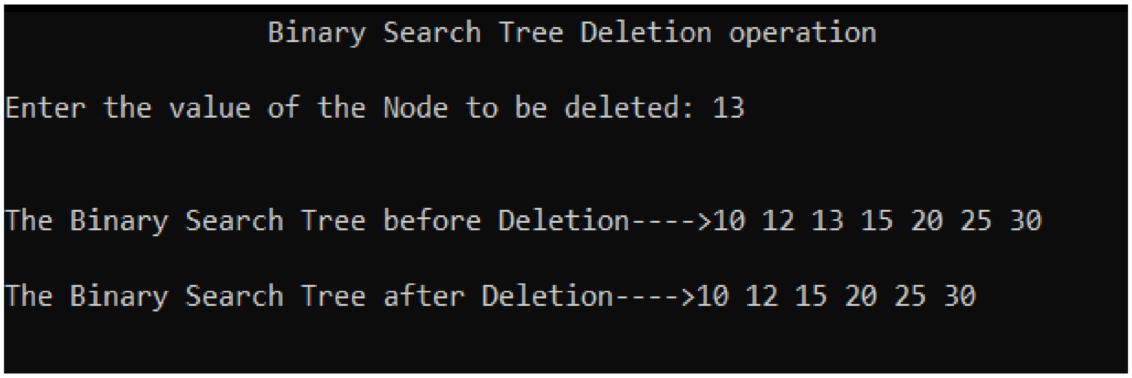
We will demonstrate tree deletion operations performed in the binary search tree

```
1. Delete a leaf node
2. Delete a node with only one child.
3. Delete a node with two child.
```

Task 1: Delete a leaf node

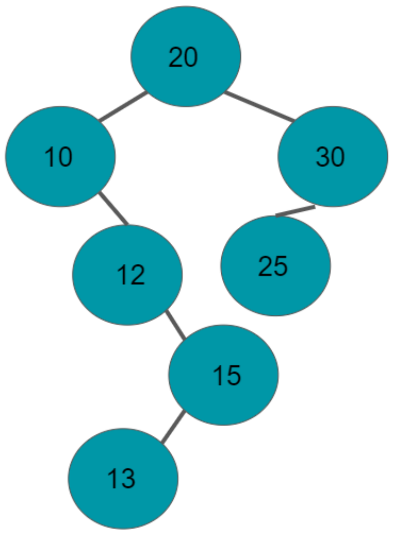
Steps to follow:

```
Algorithm:
1.If the node to be deleted is the leaf node. In such a case, simply
delete the node from the tree.
```

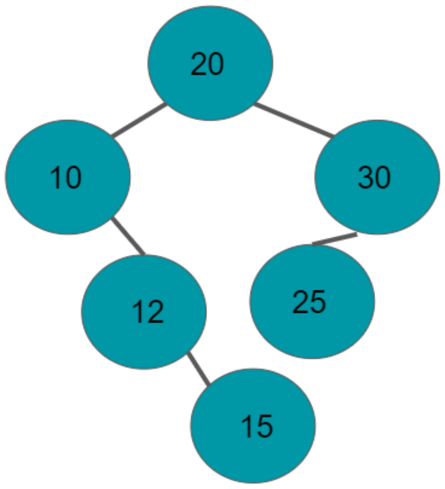


```
                Binary Search Tree Deletion operation

Enter the value of the Node to be deleted: 13


The Binary Search Tree before Deletion---->10 12 13 15 20 25 30

The Binary Search Tree after Deletion---->10 12 15 20 25 30
```

Binary Search Tree before deletion:



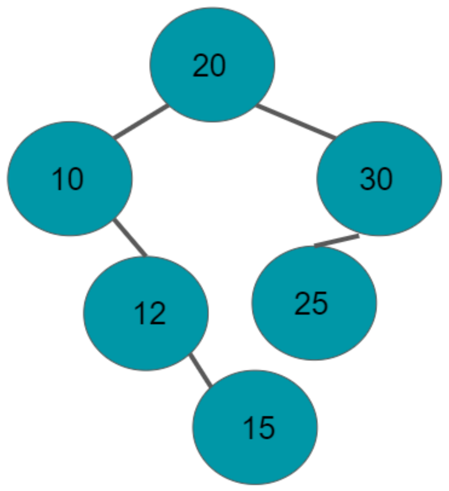
Binary Search Tree after deletion of the leaf node ‘13’



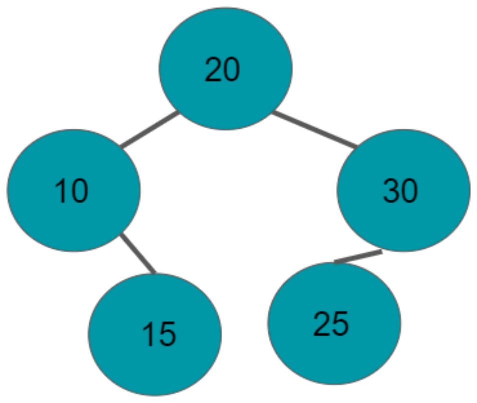
Task 2: Delete a node with only one child

```
Algorithm:
1.Replace that node with its child node.
2.Remove the child node from its original position.
```

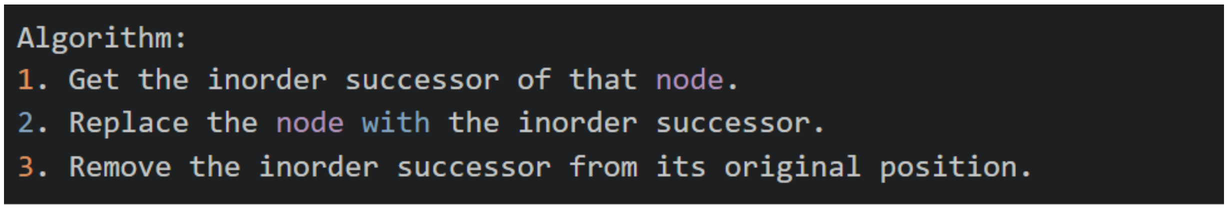
Binary Search Tree before deletion



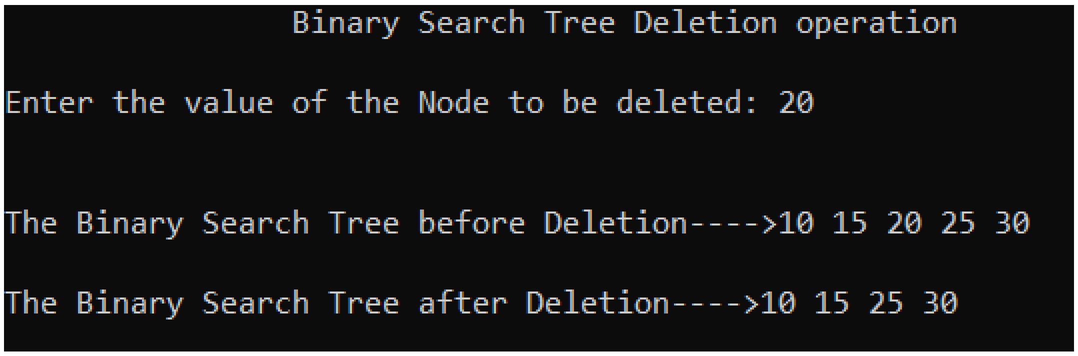
Binary Search Tree after deletion:



Task 3: Delete a node with two children

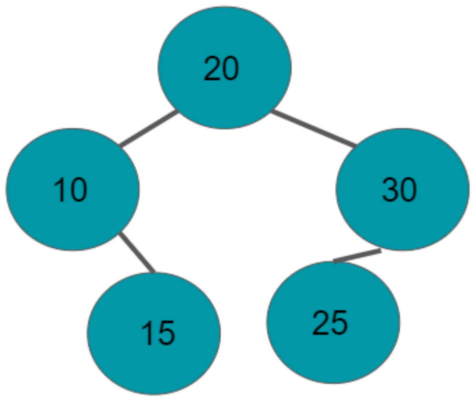


```
Algorithm:
1. Get the inorder successor of that node.
2. Replace the node with the inorder successor.
3. Remove the inorder successor from its original position.
```

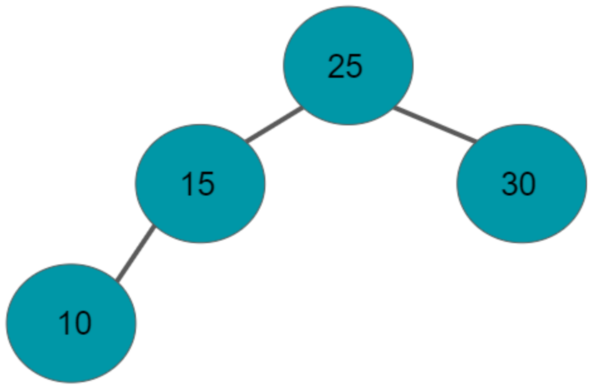


```
                Binary Search Tree Deletion operation

Enter the value of the Node to be deleted: 20


The Binary Search Tree before Deletion---->10 15 20 25 30

The Binary Search Tree after Deletion---->10 15 25 30
```

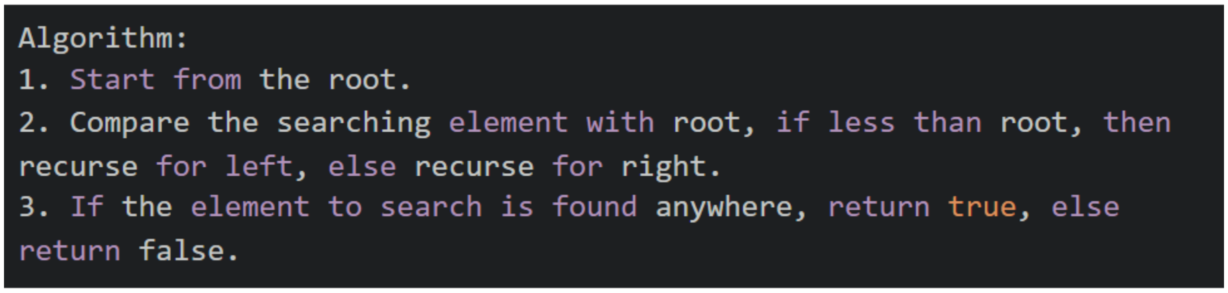
Binary search tree before deletion:



Binary Search Tree after deletion

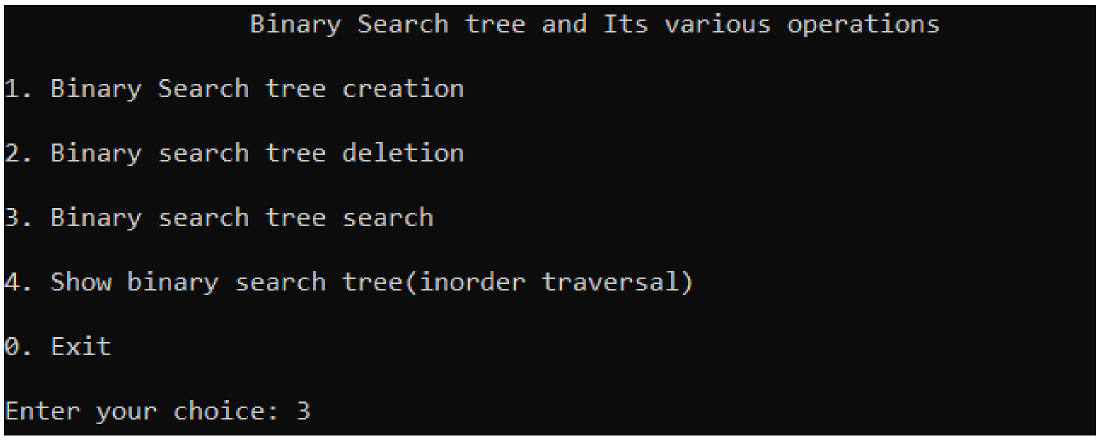


Searching a Node:



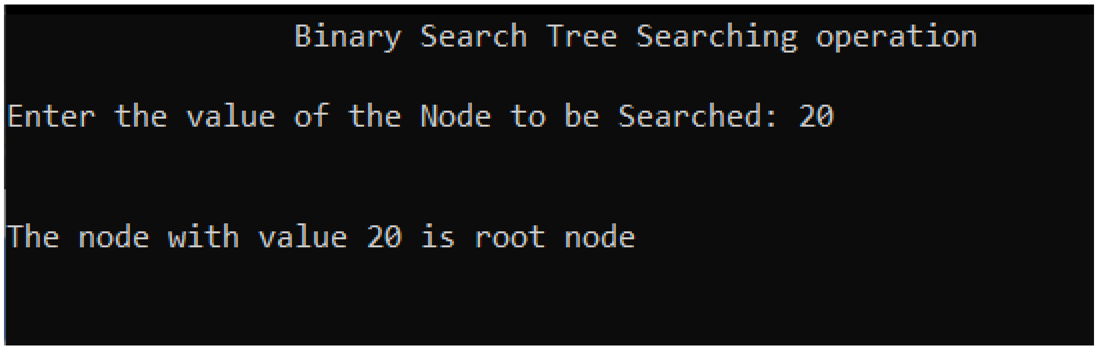
```
Algorithm:
1. Start from the root.
2. Compare the searching element with root, if less than root, then
recurse for left, else recurse for right.
3. If the element to search is found anywhere, return true, else
return false.
```

To search a node from the binary search tree we will select the option 3 from the menu

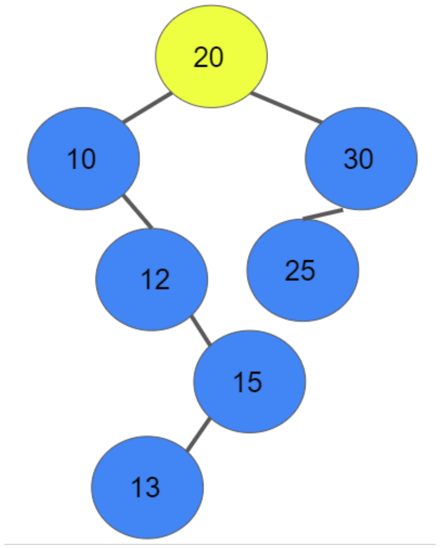


```
                Binary Search tree and Its various operations

1. Binary Search tree creation

2. Binary search tree deletion

3. Binary search tree search

4. Show binary search tree(inorder traversal)

0. Exit

Enter your choice: 3
```

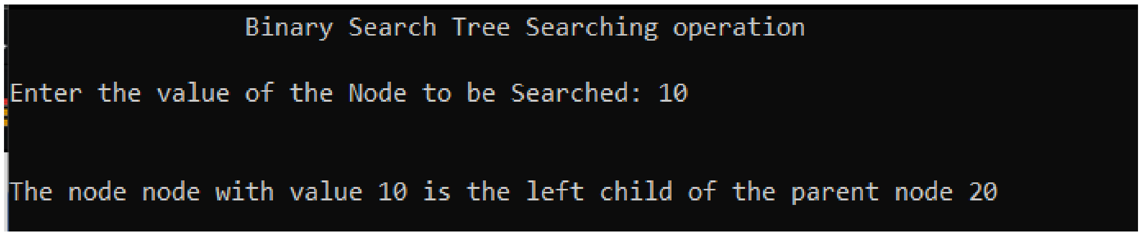
1. Searching of root node → 20



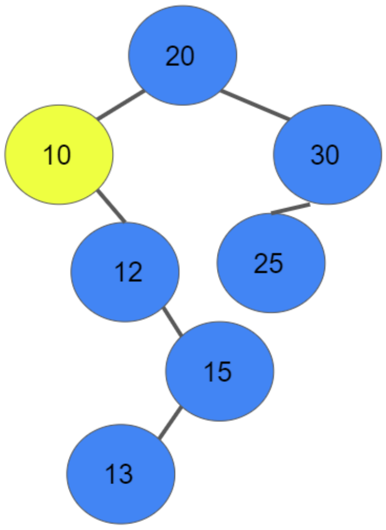
```
                   Binary Search Tree Searching operation

Enter the value of the Node to be Searched: 20


The node with value 20 is root node
```



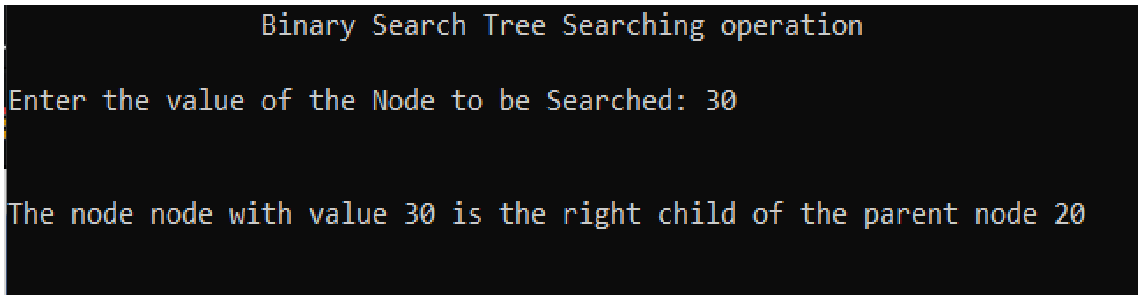
2. Searching for a left child root. In our case it is node with element 10



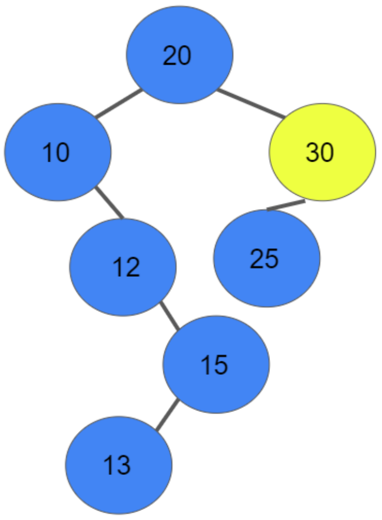
```
                Binary Search Tree Searching operation

Enter the value of the Node to be Searched: 10


The node node with value 10 is the left child of the parent node 20
```



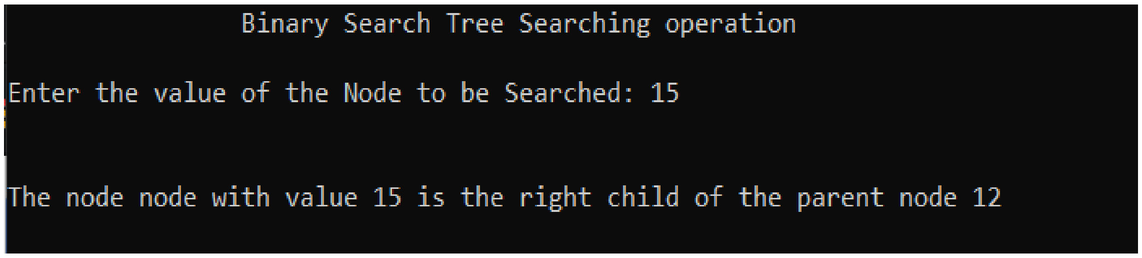
3. Searching for a right child root. In our case it is node with element 30.



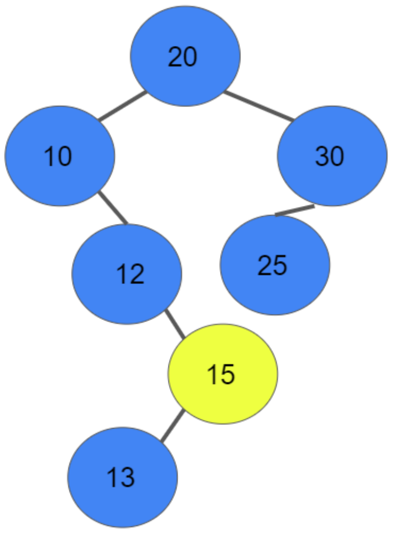
```
                Binary Search Tree Searching operation

Enter the value of the Node to be Searched: 30


The node node with value 30 is the right child of the parent node 20
```



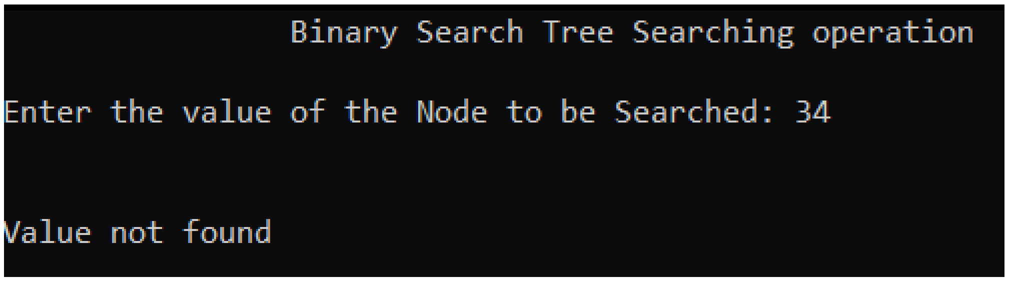
4. Searching for the node 15.



```
                Binary Search Tree Searching operation

Enter the value of the Node to be Searched: 15


The node node with value 15 is the right child of the parent node 12
```



5. Searching for the node value which is not present in the binary search tree.



```
                Binary Search Tree Searching operation

Enter the value of the Node to be Searched: 34


Value not found
```

CONCLUSION

In this laboratory, we have successfully been introduced with binary trees and binary search trees and kept a very important point in our mind that trees are hierarchical data structures. We understood various categories of a binary tree including strictly binary tree and complete binary tree.

The most primary rule has been incorporated while constructing the binary trees that nodes are inserted from left to right at the same level.

We have studied in detail about the various traversal techniques in a binary tree, such as inorder, pre-order and post-order traversals.

We performed several functions with binary search trees like constructing, deleting, searching and the most significant of all this is searching. We observed that the order of searching is in O(h), where h is the height of the binary search tree.

There are different categories of binary trees and binary search trees keeping the time and space complexity, efficiency and applications in concern. Some of them are AVL Trees, Red-Black Trees, Threaded Binary Trees, K-D Trees, etc.

Some of the applications of Binary Trees are:

1. In large stream of datas
2. Indexing
3. Computer Networking
4. Operating Systems