```
/*
Create an inordered threaded binary tree and perform inorder and preorder traversals. Analyze ti
me and space complexity of the algorithm.
*/

#include <iostream>
using namespace std;

class Node
{
    int data;
    Node *rchild, *lchild;
    bool rbit, lbit;

public:
    Node()
    {
        data = 0;
        lchild = NULL;
        rchild = NULL;
        rbit = lbit = false;
    }
    Node(int key)
    {
        data = key;
        lchild = NULL;
        rchild = NULL;
        rbit = lbit = false;
    }
    friend class TBT;
```

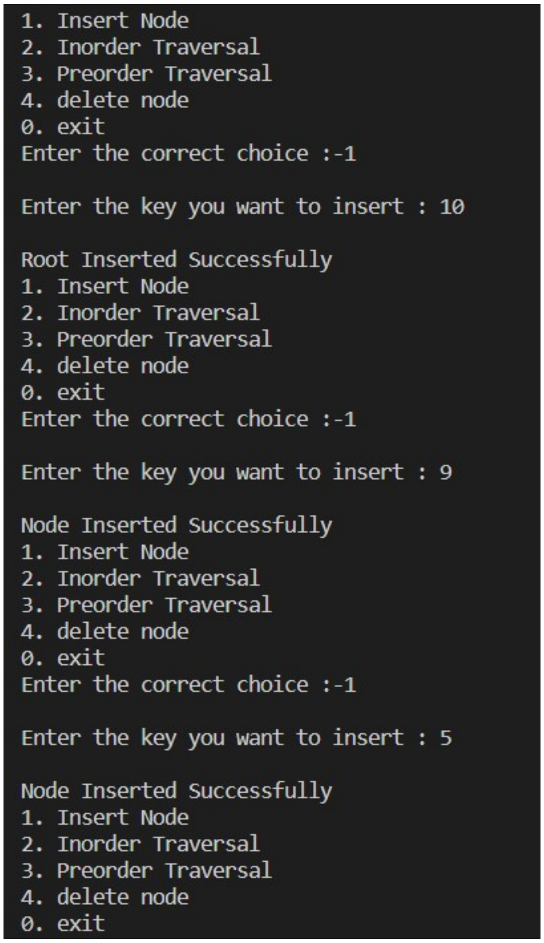
```
};

class TBT
{
   Node *header, *root;

public:
   TBT()
   {
      header = NULL;
      root = NULL;
   }
   void insertInTBT(int);
   void inorderTraversal();
   void preorderTraversal();

};

void TBT::insertInTBT(int key)
{
   if (root == NULL)
   {
      header = new Node(-99);
      header->rchild = header;
      root = new Node(key);
      root->lchild = header;
      root->rchild = header;
      header->lchild = root;
      cout << "\nRoot Inserted Successfully";
      return;
```

```
}
Node *ptr, *temp;
ptr = root;
temp = new Node(key);
while (true)
{
    if (ptr->data > key)
    {
        if (ptr->lbit)
        {
            ptr = ptr->lchild;
        }
        else
        {
            temp->lchild = ptr->lchild;
            temp->rchild = ptr;
            ptr->lbit = true;
            ptr->lchild = temp;
            cout << "\nNode Inserted Successfully";
            return;
        }
    }
    else{
    if (ptr->rbit)
    {
        ptr = ptr->rchild;
    }
    else
    {
        temp->lchild = ptr;
```

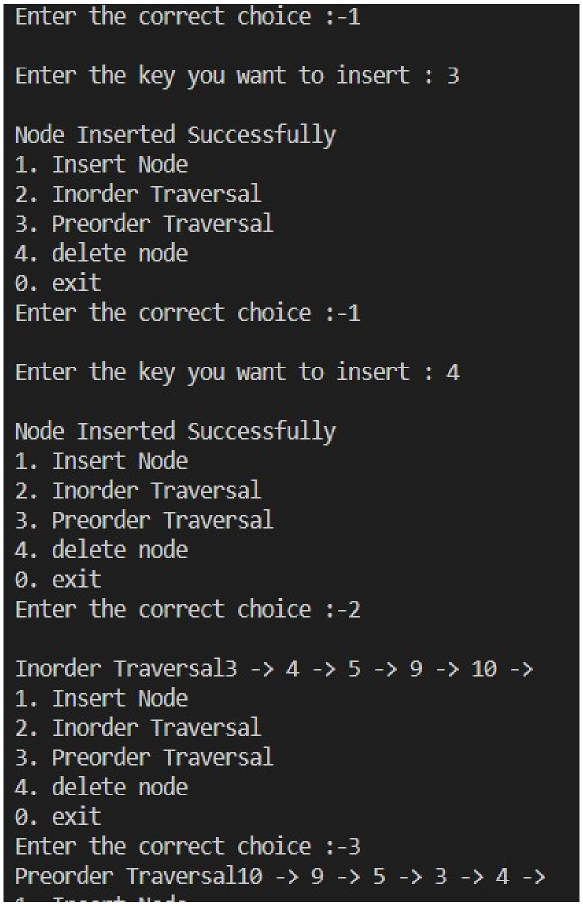
```
            temp->rchild = ptr->rchild;
            ptr->rchild = temp;
            ptr->rbit = true;
            cout << "\nNode Inserted Successfully";
            return;
        }
        }
    }
}

void TBT::inorderTraversal()
{
    Node *temp = root;
    while (temp->lbit)
    {
        temp = temp->lchild;
    }
    while (temp != header)
    {
        cout << temp->data << " -> ";
        if (temp->rbit)
        {
            temp = temp->rchild;
            while (temp->lbit)
            {
                temp = temp->lchild;
            }
        }
        else
        {
            temp = temp->rchild;
        }
    }
}

void TBT::preorderTraversal()
{
    Node *temp = root;
    while (temp != header)
    {
        while(temp->lbit){
            cout<<temp->data<<" -> ";
            temp = temp->lchild;
        }
        cout<<temp->data<<" -> ";
        while(!temp->rbit){
            temp = temp->rchild;
            if(temp == header){
                return;
            }
        }
        temp = temp->rchild;
    }
}

// void TBT::search(int key, Node ** parent, Node **loc){
//  if(root == NULL){
//      loc = NULL;
//      parent = NULL;
//      return;
//  }
//  parent = NULL;
//  loc =NULL;
//  Node *ptr;
//  ptr = root;
//  while(ptr!= NULL){
//      if(key == ptr->data){
//          loc = ptr;
//          return;
//      }
//      else if(key< ptr->data){
//          parent = ptr;
//          ptr=ptr->lchild;
//      }
//      else{
//          parent = ptr;
//          ptr = ptr->rchild;
//      }
//  }
//  if(loc == NULL){
//      cout<<"Not found";
//  }

// }

// void TBT::deleteNodeTBT(Node *ptr, Node *temp)
// {
//     if (temp->lbit && temp->rbit)
//     {
//         Node *cs = temp->rchild;
//        while (cs->lbit != 0)
//        {
//            ptr = cs;
//            cs = cs->lchild;
//        }
//        temp->data = cs->data;
//        temp = cs;
//        delete temp;
//        return;
//    }
//    if(temp->lbit==0 && (temp->rbit == 0)){
//        if(ptr->lbit){
//            ptr->lchild = temp->lchild;
//            ptr->lbit = 0;
//        }
//        ptr->rchild = temp->rchild;
//        ptr->rbit = 0;
//    delete(temp);
//    return;
//    }
//    if(temp->lbit && temp->rbit == 0){
//        temp = temp->lchild;
//        if(ptr->lchild == temp){
//            ptr->lchild = temp;
//        }
//        else{
//            ptr->rchild = temp;
//        }
//        while(temp->rbit){
//            temp = temp->rchild;
//        }
//    }
// }

int main()
{
    TBT t;
    int ch;
    do{
        cout<<"\n1. Insert Node\n2. Inorder Traversal\n3. Preorder Traversal\n0. exit \nEnter the correct choice :-";
        cin>>ch;
        int k;
        switch (ch)
        {
        case 0:
            cout<<"\nEnding the program";
            break;
        case 1:
            cout<<"\nEnter the key you want to insert : ";
            cin>>k;
            t.insertInTBT(k);
            break;
        case 2:
            cout<<"\nInorder Traversal  : ";
            t.inorderTraversal();
            break;
        case 3:
            cout<<"\nPreorder Traversal  : ";
            t.preorderTraversal();
```

```
          break;
        default:
          cout<<"Wrong choice";
          break;
        }
    }while(ch!=0);
    return 0;
}
```

OUTPUT:-



```
1. Insert Node
2. Inorder Traversal
3. Preorder Traversal
4. delete node
0. exit
Enter the correct choice :-1

Enter the key you want to insert : 10

Root Inserted Successfully
1. Insert Node
2. Inorder Traversal
3. Preorder Traversal
4. delete node
0. exit
Enter the correct choice :-1

Enter the key you want to insert : 9

Node Inserted Successfully
1. Insert Node
2. Inorder Traversal
3. Preorder Traversal
4. delete node
0. exit
Enter the correct choice :-1

Enter the key you want to insert : 5

Node Inserted Successfully
1. Insert Node
2. Inorder Traversal
3. Preorder Traversal
4. delete node
0. exit
```



```
Enter the correct choice :-1

Enter the key you want to insert : 3

Node Inserted Successfully
1. Insert Node
2. Inorder Traversal
3. Preorder Traversal
4. delete node
0. exit
Enter the correct choice :-1

Enter the key you want to insert : 4

Node Inserted Successfully
1. Insert Node
2. Inorder Traversal
3. Preorder Traversal
4. delete node
0. exit
Enter the correct choice :-2

Inorder Traversal3 -> 4 -> 5 -> 9 -> 10 ->
1. Insert Node
2. Inorder Traversal
3. Preorder Traversal
4. delete node
0. exit
Enter the correct choice :-3
Preorder Traversal10 -> 9 -> 5 -> 3 -> 4 ->
```