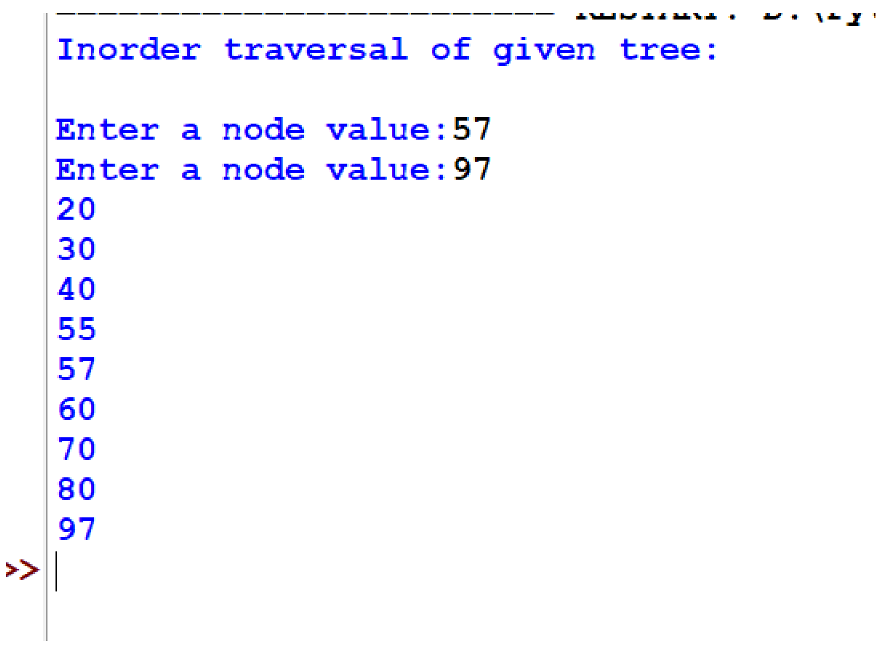
```python
# Python program for implementation of MergeSort
def mergeSort(arr):
    if len(arr) > 1:

        # Finding the mid of the array
        mid = len(arr)//2

        # Dividing the array elements
        L = arr[:mid]

        # into 2 halves
        R = arr[mid:]

        # Sorting the first half
        mergeSort(L)

        # Sorting the second half
        mergeSort(R)

        i = j = k = 0

        # Copy data to temp arrays L[] and R[]
        while i < len(L) and j < len(R):
            if L[i] < R[j]:
                arr[k] = L[i]
                i += 1
            else:
                arr[k] = R[j]
                j += 1
            k += 1

        # Checking if any element was left
        while i < len(L):
            arr[k] = L[i]
            i += 1
            k += 1

        while j < len(R):
            arr[k] = R[j]
            j += 1
            k += 1

# Code to print the list


def printList(arr):
    for i in range(len(arr)):
        print(arr[i], end=" ")
    print()


# Driver Code
if __name__ == '__main__':
    arr = [12, 11, 13, 5, 6, 7]
    print("Given array is", end="\n")
    printList(arr)
    mergeSort(arr)
    print("Sorted array is: ", end="\n")
    printList(arr)

# This code is contributed by Mayank Khanna
```

--

```
Given array is
12 11 13 5 6 7
Sorted array is:
5 6 7 11 12 13
```

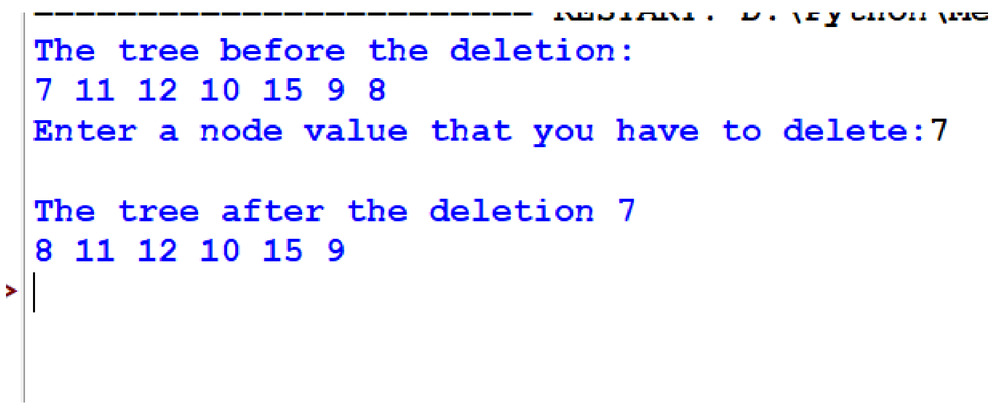
```python
# Python program to demonstrate
# insert operation in binary search tree

# A utility class that represents
# an individual node in a BST


class Node:
    def __init__(self, key):
        self.left = None
        self.right = None
        self.val = key

# A utility function to insert
# a new node with the given key


def insert(root, key):
    if root is None:
        return Node(key)
    else:
        if root.val == key:
            return root
        elif root.val < key:
            root.right = insert(root.right, key)
        else:
            root.left = insert(root.left, key)
    return root

# A utility function to do inorder tree traversal

print("Inorder traversal of given tree:\n")
def inorder(root):

    if root:
        inorder(root.left)
        print(root.val)
        inorder(root.right)


# Driver program to test the above functions
# Let us create the following BST
#    50
#  /     \
# 30     70
#  / \ / \
# 20 40 60 80

r = Node(80)
r = insert(r, 60)
r = insert(r, 40)
r = insert(r, 20)
r = insert(r, 30)
r = insert(r, 70)
r = insert(r, 55)
a=int(input("Enter a node value:"))
r=insert(r,a)
a=int(input("Enter a node value:"))
r=insert(r,a)

# Print inoder traversal of the BST
inorder(r)
```



```
Inorder traversal of given tree:

Enter a node value:57
Enter a node value:97
20
30
40
55
57
60
70
80
97
>>
```

```
# Python3 program to illustrate deletion in a Binary Tree

# class to create a node with data, left child and right child.
class Node:
```

```python
    def __init__(self,data):
        self.data = data
        self.left = None
        self.right = None

# Inorder traversal of a binary tree
def inorder(temp):
    if(not temp):
        return
    inorder(temp.left)
    print(temp.data, end = " ")
    inorder(temp.right)

# function to delete the given deepest node (d_node) in binary tree
def deleteDeepest(root,d_node):
    q = []
    q.append(root)
    while(len(q)):
        temp = q.pop(0)
        if temp is d_node:
            temp = None
            return
        if temp.right:
            if temp.right is d_node:
                temp.right = None
                return
            else:
                q.append(temp.right)
        if temp.left:
            if temp.left is d_node:
                temp.left = None
                return
            else:
                q.append(temp.left)

# function to delete element in binary tree
def deletion(root, key):
    if root == None :
        return None
    if root.left == None and root.right == None:
        if root.key == key :
            return None
        else :
            return root
    key_node = None
    q = []
    q.append(root)
    temp = None
    while(len(q)):
        temp = q.pop(0)
        if temp.data == key:
            key_node = temp
        if temp.left:
            q.append(temp.left)
        if temp.right:
            q.append(temp.right)
    if key_node :
        x = temp.data
        deleteDeepest(root,temp)
        key_node.data = x
    return root

# Driver code
if __name__=='__main__':
    root = Node(10)
    root.left = Node(11)
    root.left.left = Node(7)
    root.left.right = Node(12)
    root.right = Node(9)
    root.right.left = Node(15)
    root.right.right = Node(8)
    print("The tree before the deletion:")
    inorder(root)
    key = int(input("\nEnter a node value that you have to delete:"))
    root = deletion(root, key)
    print()
    print("The tree after the deletion",key)
    inorder(root)

# This code is contributed by Monika Anandan
```



```
================================ RESTART: D:\Python\Me
The tree before the deletion:
7 11 12 10 15 9 8
Enter a node value that you have to delete:7

The tree after the deletion 7
8 11 12 10 15 9
>
```