



EXPERIMENT - 9

Data Structures

Aim

Create a Binary Tree using linked list (Display using Graphics) and perform the following operations: Tree traversals (Preorder, Postorder, Inorder) using the concept of recursion.

Syeda Reeha Quasar

14114902719

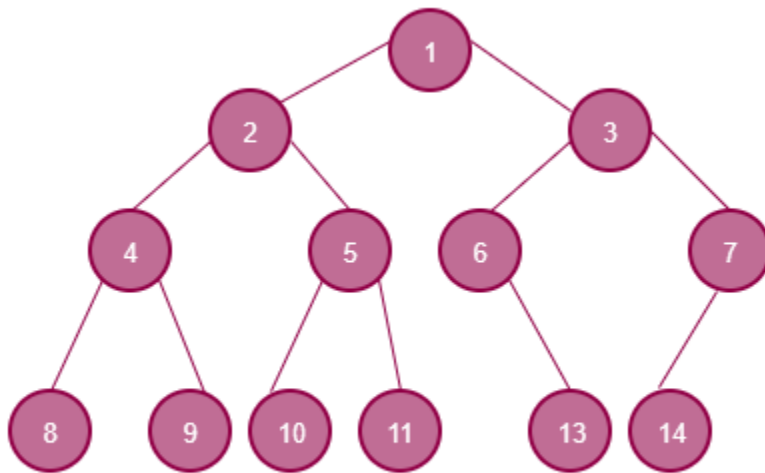
3C7

EXPERIMENT – 9

AIM: Create a Binary Tree using linked list (Display using Graphics) and perform the following operations: Tree traversals (Preorder, Postorder, Inorder) using the concept of recursion

THEORY

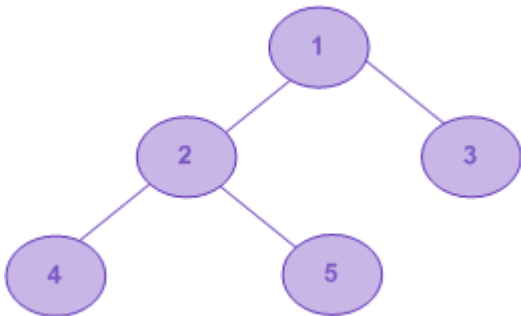
A tree whose elements have at most 2 children is called a binary tree. Since each element in a binary tree can have only 2 children, we typically name them the left and right child.



A Binary Tree node contains following parts.

1. Data
2. Pointer to left child
3. Pointer to right child

Unlike linear data structures (Array, Linked List, Queues, Stacks, etc) which have only one logical way to traverse them, trees can be traversed in different ways. Following are the generally used ways for traversing trees.



Example Tree

Depth First Traversals:

(a) Inorder (Left, Root, Right) : 4 2 5 1 3

(b) Preorder (Root, Left, Right) : 1 2 4 5 3

(c) Postorder (Left, Right, Root) : 4 5 2 3 1

Breadth First or Level Order Traversal : 1 2 3 4 5

Please see [this](#) post for Breadth First Traversal.

Inorder Traversal:

Algorithm Inorder(tree)

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.

Preorder Traversal:

Algorithm Preorder(tree)

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 prefix expression on of an expression tree. Please see http://en.wikipedia.org/wiki/Polish_notation to know why prefix expressions are useful.

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

Postorder Traversal:

Algorithm Postorder(tree)

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. Please see [the question for deletion of tree](#) for details.

Postorder traversal is also useful to get the postfix expression of an expression tree. Please see http://en.wikipedia.org/wiki/Reverse_Polish_notation to for the usage of postfix expression.

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

Source code:

```
//necessary libs

#include <stdio.h>

#include <stdlib.h>

// A binary tree node containing data, left pointer and right pointer
struct bNode {
    int data;
    struct bNode* left;
```

```
    struct bNode* right;
```

```
};
```

```
// newNode creator for tree nodes
```

```
struct bNode* bNewNode(int data) {
```

```
    struct bNode* bNode = (struct bNode*) malloc(sizeof(struct bNode));
```

```
    bNode -> data = data;
```

```
    bNode -> left = NULL;
```

```
    bNode -> right = NULL;
```

```
    return (bNode);
```

```
}
```

```
// function for post order traversal
```

```
void printPostorder(struct bNode* bNode) {
```

```
    if (bNode == NULL) //empty tree
```

```
        return;
```

```
    printPostorder(bNode -> left); // first recur on left subtree
```

```
    printPostorder(bNode -> right); // then recur on right subtree
```

```
    printf("%d ", bNode -> data); //printing each node data
```

```
}
```

```
// function for inorder traversal
```

```
void printInorder(struct bNode* bNode) {
```

```
    if (bNode == NULL) // tree empty
```

```
        return;
```

```
    printf("%d ", bNode -> left); // first recur on left child

    printf("%d ", bNode -> data); // then print the data of node

    printf("%d ", bNode -> right); // now recur on right child
}

// preorder traversal function
void printPreorder(struct bNode* bNode) {
    if (bNode == NULL) // empty tree case
        return;

    printf("%d ", bNode -> data); // first print data of node

    printPreorder(bNode -> left); // then recur on left subtree

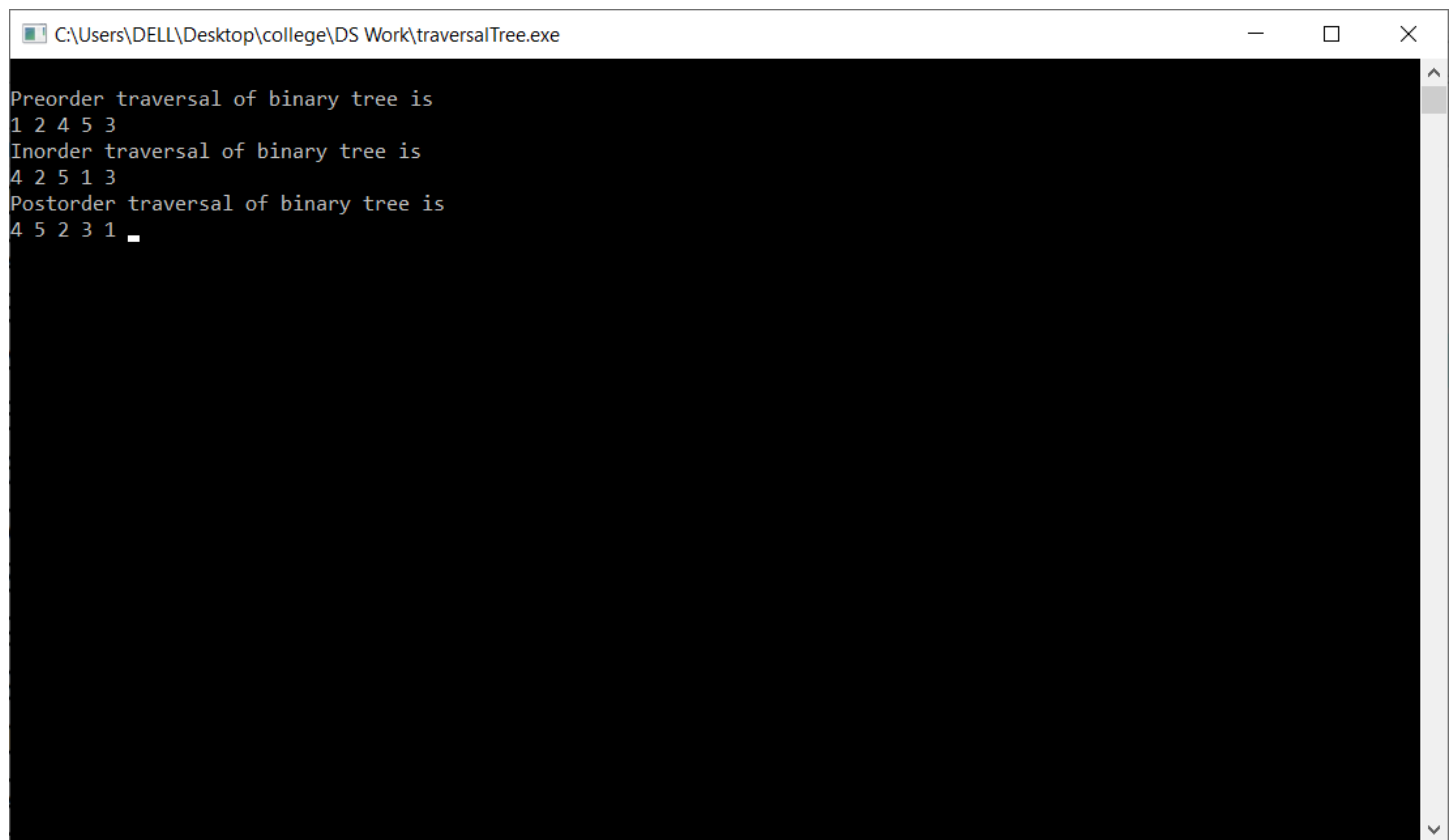
    printPreorder(bNode -> right); // now recur on right subtree
}

int main() {
    struct bNode *root = bNewNode(1);
    root -> left = bNewNode(2);
    root -> right = bNewNode(3);
    root -> left -> left = bNewNode(4);
    root -> left -> right = bNewNode(5);

    printf("\nPreorder traversal of binary tree is \n");
    printPreorder(root);
}
```

```
printf("\nInorder traversal of binary tree is \n");  
  
printInorder(root);  
  
  
printf("\nPostorder traversal of binary tree is \n");  
printPostorder(root);  
  
  
getchar();  
return 0;  
  
}
```

OUTPUT



```
C:\Users\DELL\Desktop\college\DS Work\traversalTree.exe  
Preorder traversal of binary tree is  
1 2 4 5 3  
Inorder traversal of binary tree is  
4 2 5 1 3  
Postorder traversal of binary tree is  
4 5 2 3 1 _
```

Tree construction using graphics:

```
#include <graphics.h>
```

```
#include <iostream>
```

```
#include <math.h>
```

```
#include <sstream>
```

```
using namespace std;
```

```
void printTree(int x, int y, int* array, int index, int total_elements){
```

```
    // Base Case
```

```
    if (index >= total_elements)
```

```
        return;
```

```
    // Convert int value into string
```

```
    ostringstream str1;
```

```
    str1 << array[index];
```

```
    string str2 = str1.str();
```

```
    char* str = &str2[0u];
```

```
    setcolor(GREEN);
```

```
    // Draw the circle of radius 15 that represent node of Tree
```

```
    circle(x, y, 15);
```

```
    // Print the values of the node in the circle
```

```
    outtextxy(x - 2, y - 3, str);
```

```
    // Set the color of the line from parent to child as green
```

```
    setcolor(GREEN);
```

```
    // Evaluating left and right child
```

```
int left = 2 * index + 1;
```

```
int right = 2 * index + 2;
```

```
// Recursively draw the left subtree and the right subtree
```

```
printTree(x - y / (index + 1), y + 50, array, left, total_elements);
```

```
printTree(x + y / (index + 1), y + 50, array, right, total_elements);
```

```
// Draw the line (or link) when the node is not the leaf node
```

```
if (left < total_elements) {
```

```
    line(x, y, x - y / (index + 1), y + 50);
```

```
}
```

```
if (right < total_elements) {
```

```
    line(x, y, x + y / (index + 1), y + 50);
```

```
}
```

```
return;
```

```
}
```

```
int main()
```

```
{
```

```
    initwindow(900, 900);
```

```
    // Given array arr[]
```

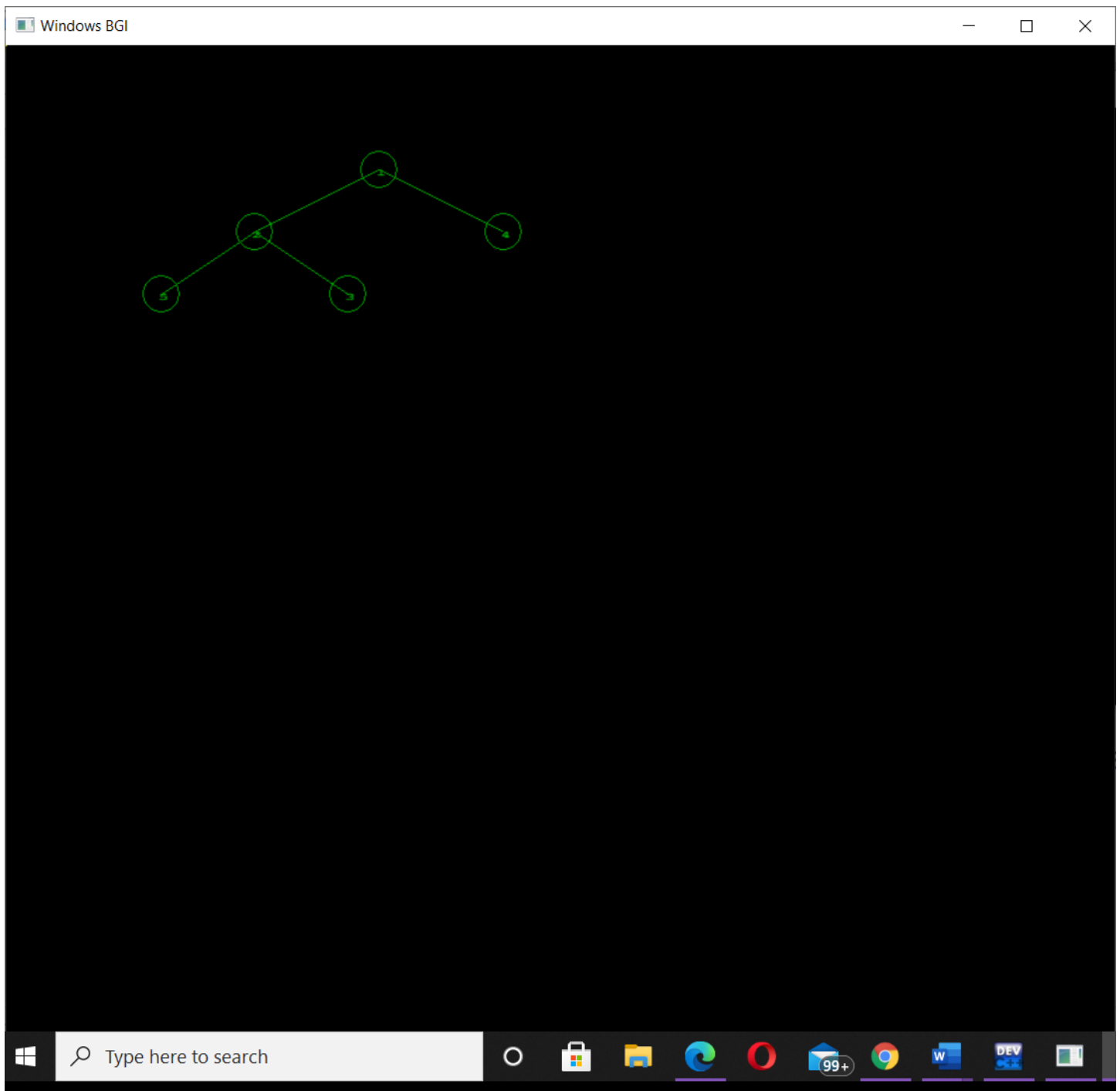
```
    int array[] = {1, 2, 4, 5, 3};
```

```
    printTree(300, 100, array, 0, 5); //printing the tree
```

```
    getch();
```

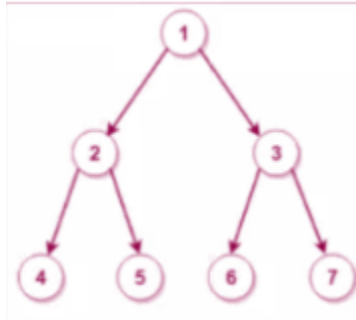


```
closegraph();  
return 0;  
}
```



VIVA VOICE

Q1. Calculate height of a binary tree?



Ans.

Height of Binary tree is 2.

The height of the binary tree is the longest path from root node to any leaf node in the tree. Here it can be any of the path all giving same height 2.

Q2. Inorder Tree Traversal (Iterative & Recursive Implementation).

Ans.

Inorder traversal of binary tree is

4 2 5 1 6 3 7

An iterative inorder traversal of a binary tree is done using the stack data structure.

Algorithm

- Initialize an empty stack.
- Push the current node (starting from the root node) onto the stack. Continue pushing nodes to the left of the current node until a NULL value is reached.
- If the current node is NULL and the stack is not empty:
 1. Remove and print the last item from the stack.
 2. Set the *current* node to be the node to the *right* of the *removed* node.
 3. Repeat the second step of the algorithm.
- If the current node is NULL and the stack is empty, then the algorithm has finished.

Algorithm Inorder(tree) Recursive

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)

Q3. Preorder Tree Traversal (Iterative & Recursive Implementation).

Ans.

Preorder traversal of binary tree is

1 2 4 5 3 6 7

ITERATIVE

Algorithm: 1. Push root to first stack.

2. Loop while first stack is not empty

2.1 Pop a node from first stack and push it to second stack

2.2 Push left and right children of the popped node to first stack

3. Print contents of second stack

RECURSIVE

Algorithm: Preorder(tree)

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)

Q4. Postorder Tree Traversal (Iterative & Recursive Implementation).

Ans.

Postorder traversal of binary tree is

4 5 2 6 7 3 1

ITERATIVE

1. Push root to first stack.

2. Loop while first stack is not empty

2.1 Pop a node from first stack and push it to second stack

2.2 Push left and right children of the popped node to first stack

3. Print contents of second stack

RECURSIVE

Algorithm Postorder(tree)

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.

Q5. Level Order Traversal of Binary Tree.

Ans.

Level Order traversal of binary tree is

1 2 3 4 5 6 7

printLevelorder(tree) **ALGORITHM**

- 1) Create an empty queue q
- 2) temp_node = root /*start from root*/
- 3) Loop while temp_node is not NULL
 - a) print temp_node->data.
 - b) Enqueue temp_node's children
(first left then right children) to q
 - c) Dequeue a node from q.

Q6. How many child nodes can there be for any parent node in a binary tree?

Ans

2

Q7. How can you traverse a binary tree?

Ans

Following are the generally used ways for traversing trees.-

Inorder

Postorder

preorder

Q8. Does the last node of post order traversal and first node of preorder traversal of a binary tree always same?

Ans false

Q9. How can you represent a binary tree in memory?

Ans

By – sequential representation(arrays) and by linked list

Q10. What is order of nodes in inorder traversal?

Ans

Left – root node – right

Q11. Can a tree be called graph also?

Ans

A connected graph with no cycles is called a tree.

More Viva Questions

Q1. Which of the following is false about a binary search tree?

- a) The left child is always lesser than its parent
- b) The right child is always greater than its parent
- c) The left and right sub-trees should also be binary search trees
- d) In order sequence gives decreasing order of elements

Ans. Answer: d

Explanation: In order sequence of binary search trees will always give ascending order of elements. Remaining all are true regarding binary search trees.

Q2. What is the speciality about the inorder traversal of a binary search tree?

- a) It traverses in a non increasing order
- b) It traverses in an increasing order
- c) It traverses in a random fashion
- d) It traverses based on priority of the node

Ans. Answer: b

Explanation: As a binary search tree consists of elements lesser than the node to the left and the ones greater than the node to the right, an inorder traversal will give the elements in an increasing order.

Q3.What does the following piece of code do?

```
public void func(Tree root)
{
    func(root.left());
    func(root.right());
    System.out.println(root.data());
}
```

- a) Preorder traversal
- b) Inorder traversal
- c) Postorder traversal
- d) Level order traversal

Ans. Answer: c

Explanation: In a postorder traversal, first the left child is visited, then the right child and finally the parent.

Q4. What is the worst case time complexity for search, insert and delete operations in a general Binary Search Tree?

- A** $O(n)$ for all
- B** $O(\log n)$ for all
- C** $O(\log n)$ for search and insert, and $O(n)$ for delete
- D** $O(\log n)$ for search, and $O(n)$ for insert and delete

Ans. A). $O(n)$ for all

In skewed Binary Search Tree (BST), all three operations can take $O(n)$. See the following example BST and operations.

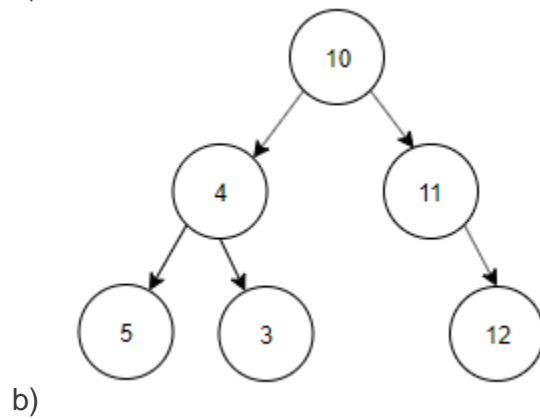
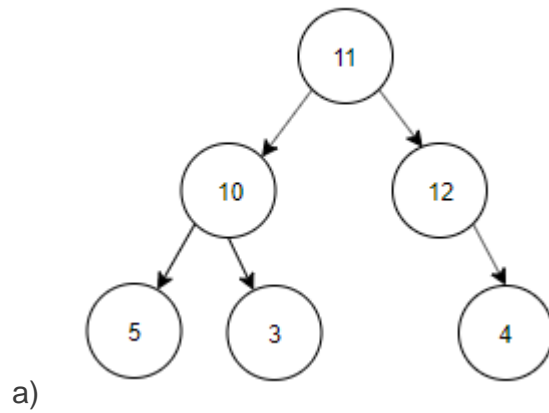
```
    10
   /
  20
 /
30
/
40
```

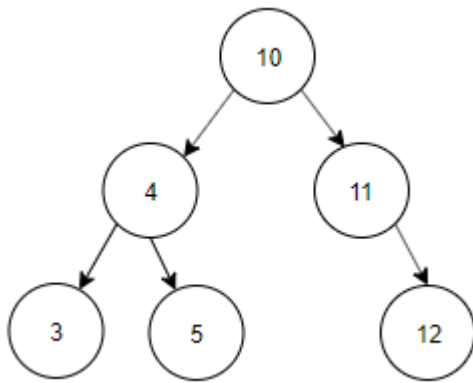
Search 40.

Delete 40

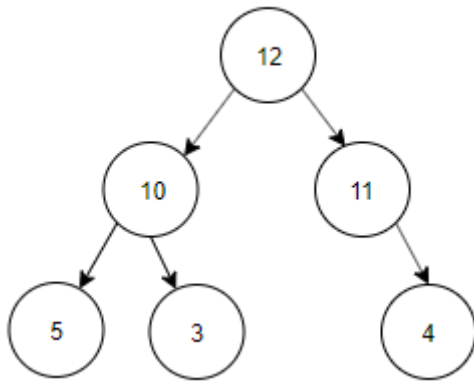
Insert 50.

- Q5. Construct a binary search tree with the below information.**
The preorder traversal of a binary search tree 10, 4, 3, 5, 11, 12.





c)



d)

Ans. Answer: c

Explanation: Preorder Traversal is 10, 4, 3, 5, 11, 12. Inorder Traversal of Binary search tree is equal to ascending order of the nodes of the Tree. Inorder Traversal is 3, 4, 5, 10, 11, 12. The tree constructed using Preorder and Inorder traversal is

