14-11-2023

Trees

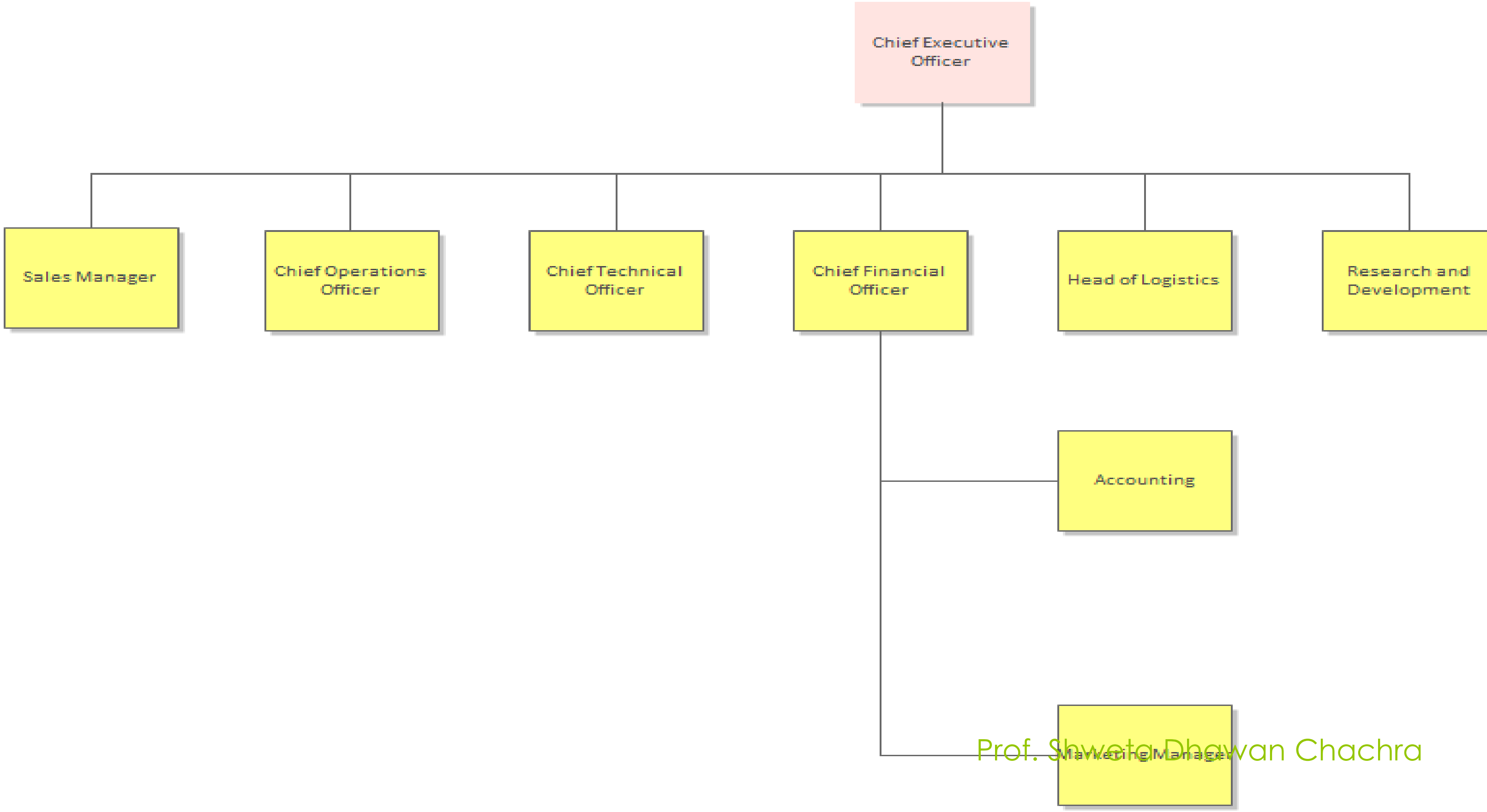
3.1

Tree

- **Non-linear data structure**
- Used to represent **hierarchical relationship among several data items**

Tree

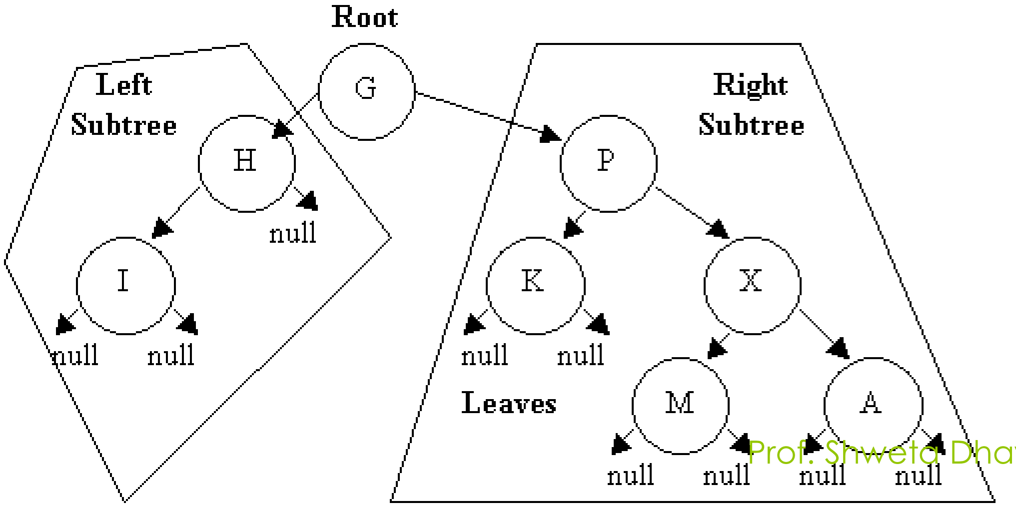
- Represents Hierarchy
- For eg- The organization structure of an Corporation



Tree

- Characteristics:

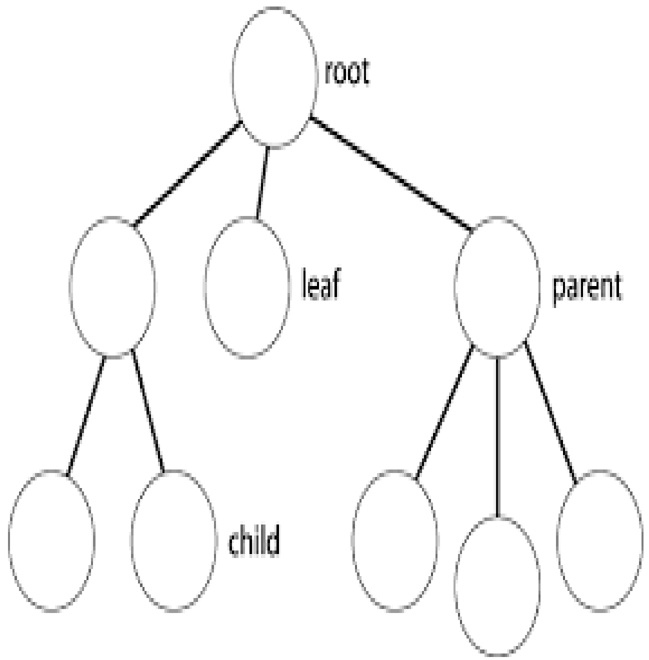
1) There is a special data item called **root of the tree**
2) Remaining data items are partitioned into number of **mutually exclusive subsets** , each of which is itself a tree, called subtrees



Tree

Basic Terminology-

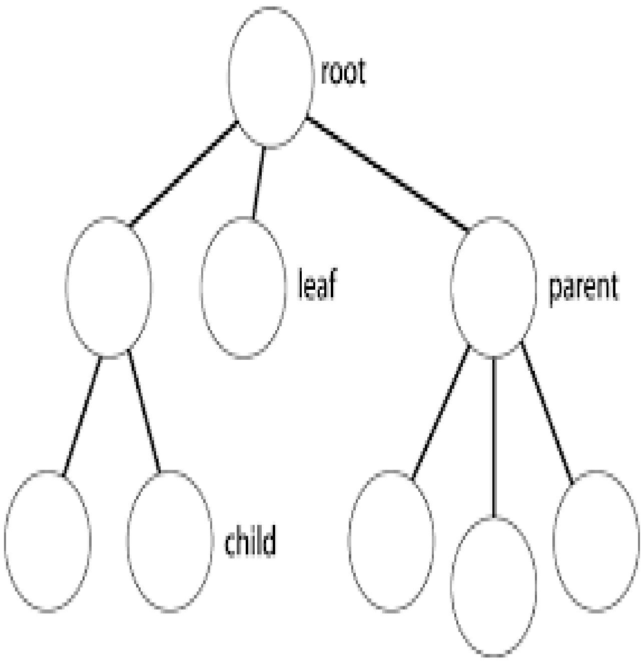
- Root-
  - Special Node in a tree structure.
  - Entire tree is referenced through it.
  - First in the hierarchal arrangement



Tree

Basic Terminology-

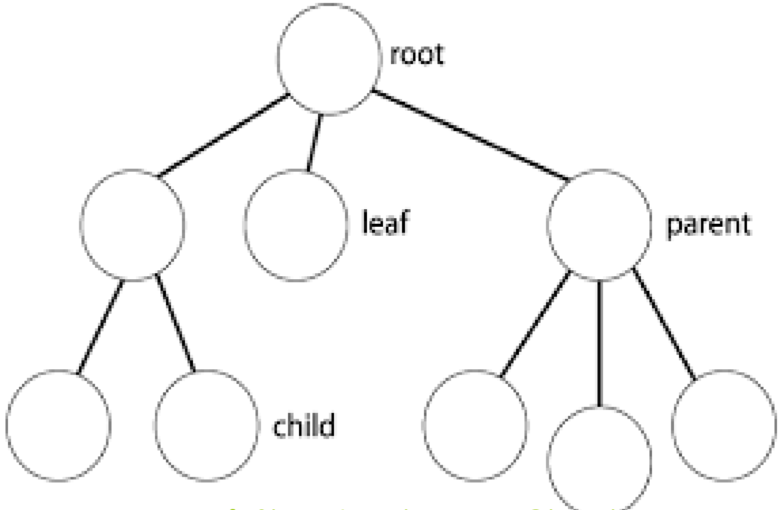
- Node-
  - Each data item in a tree.
  - Specifies the data and links to other data items



Tree

Basic Terminology-

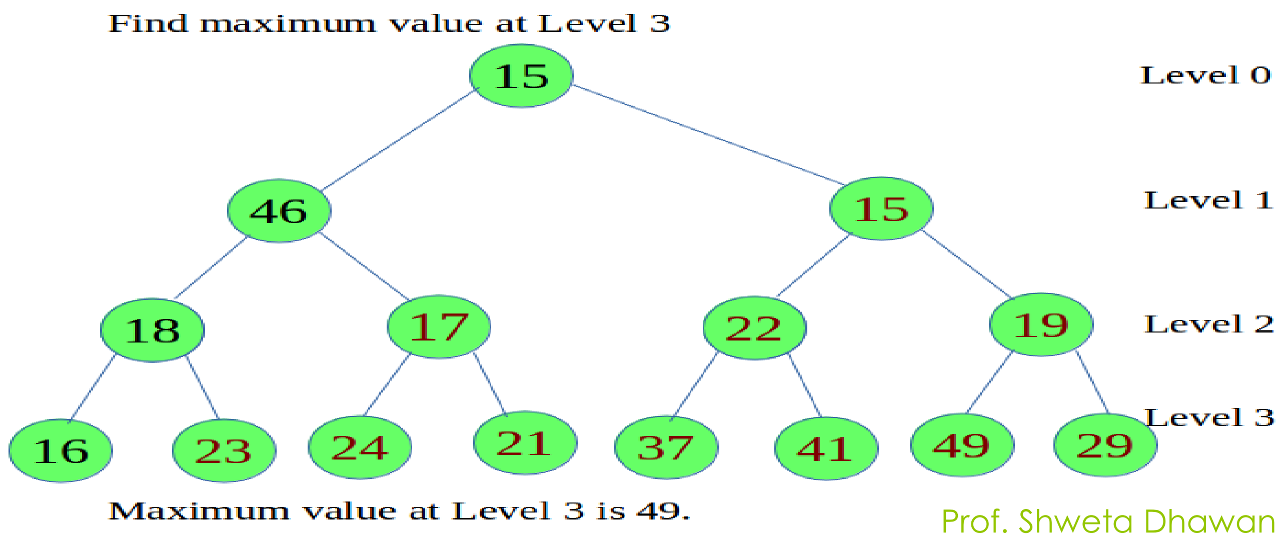
- Parent
  - Immediate predecessor of a node
- Child
  - Immediate successor of a node
- Siblings-
  - Nodes with the same parent



Tree

Basic Terminology-

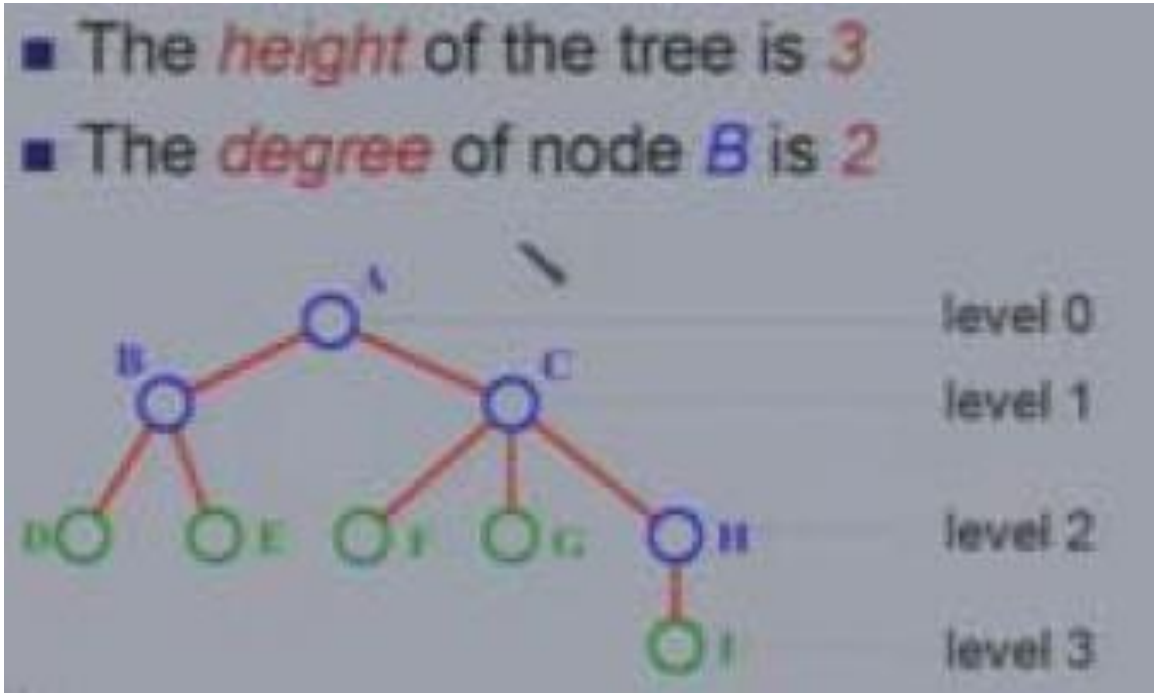
- Depth/Level-
  - Root Node is always at Level 0
  - Its immediate successor are Level 1,
  - Their Successor at level 2 and so on
  - If a node is at Level n then its children will be at Level n+1



Tree

Basic Terminology-

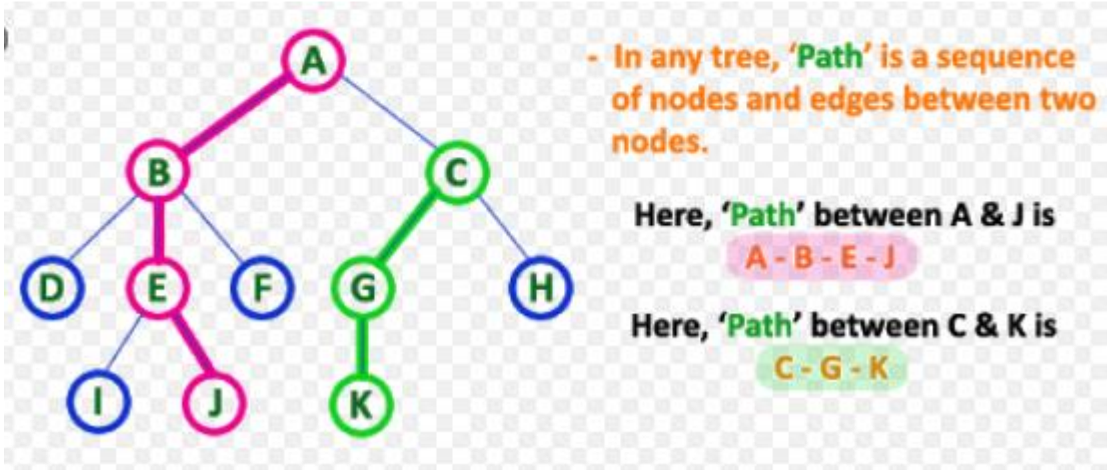
- Height -
  - Maximum level of any leaf in the tree.



Tree

Basic Terminology-

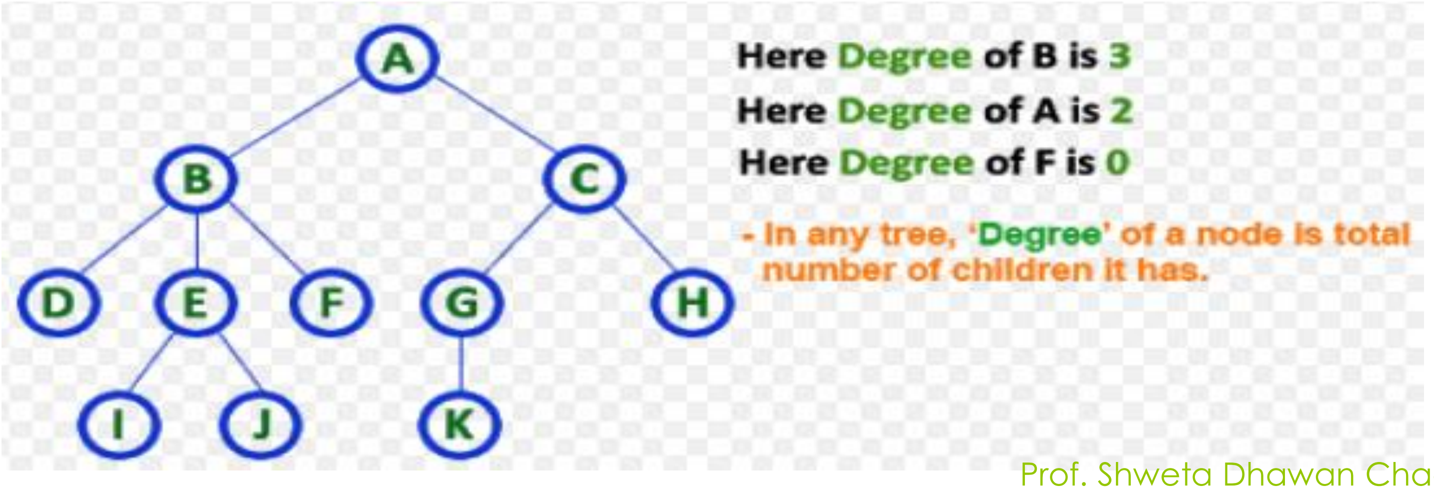
- Edge
  - Line drawn from one node to other
- Path
  - Sequence of consecutive edges from the source node to the destination node



Tree

Basic Terminology-

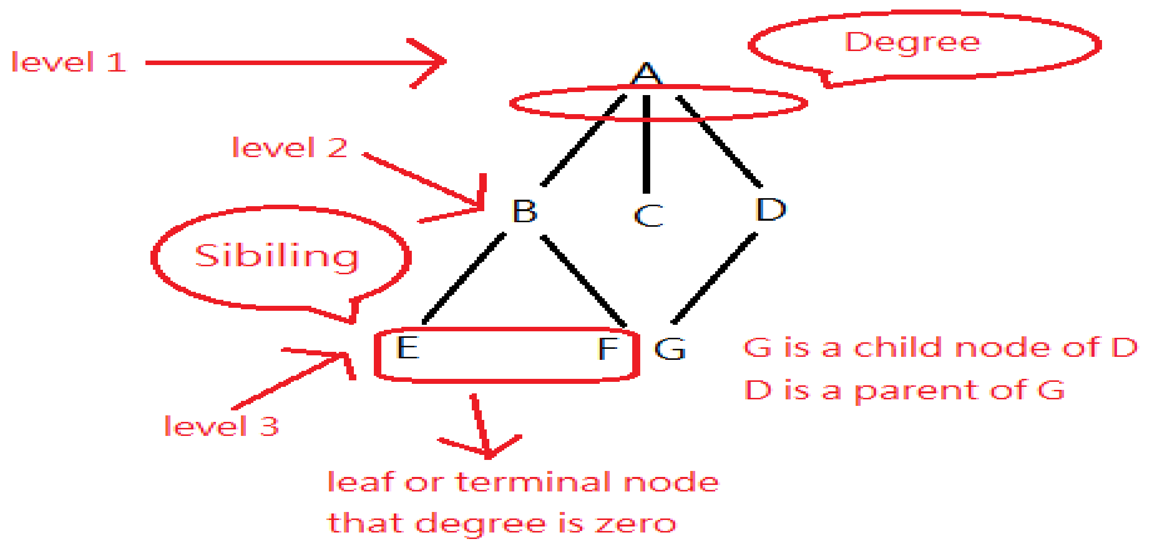
- Degree of a Node
  - Number of subtrees of a node in a given tree
- Degree of a Tree
  - The maximum degree of nodes in a given tree



Tree

Basic Terminology-

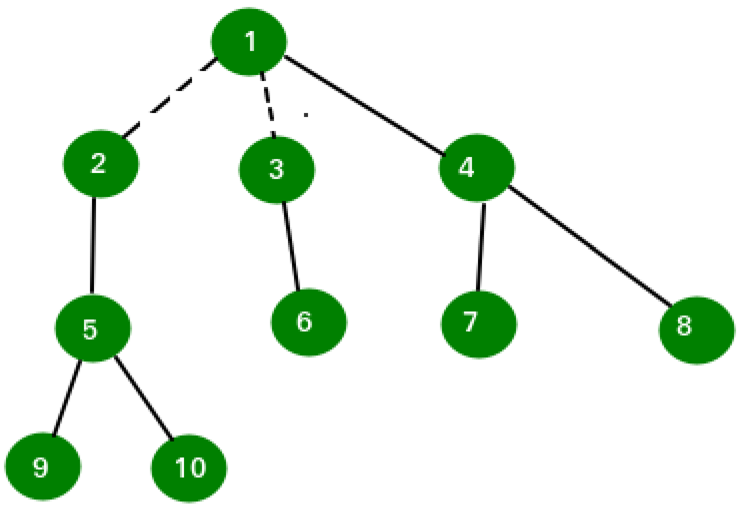
- Teminal Node/Leaf
  - Any node whose degree is 0
- Non-Terminal Node
  - Any node whose degree is Non-Zero



Tree

Basic Terminology-

- Forest
  - Set of Disjoint trees,
  - If you remove the Root node, it becomes forest.



Binary Tree

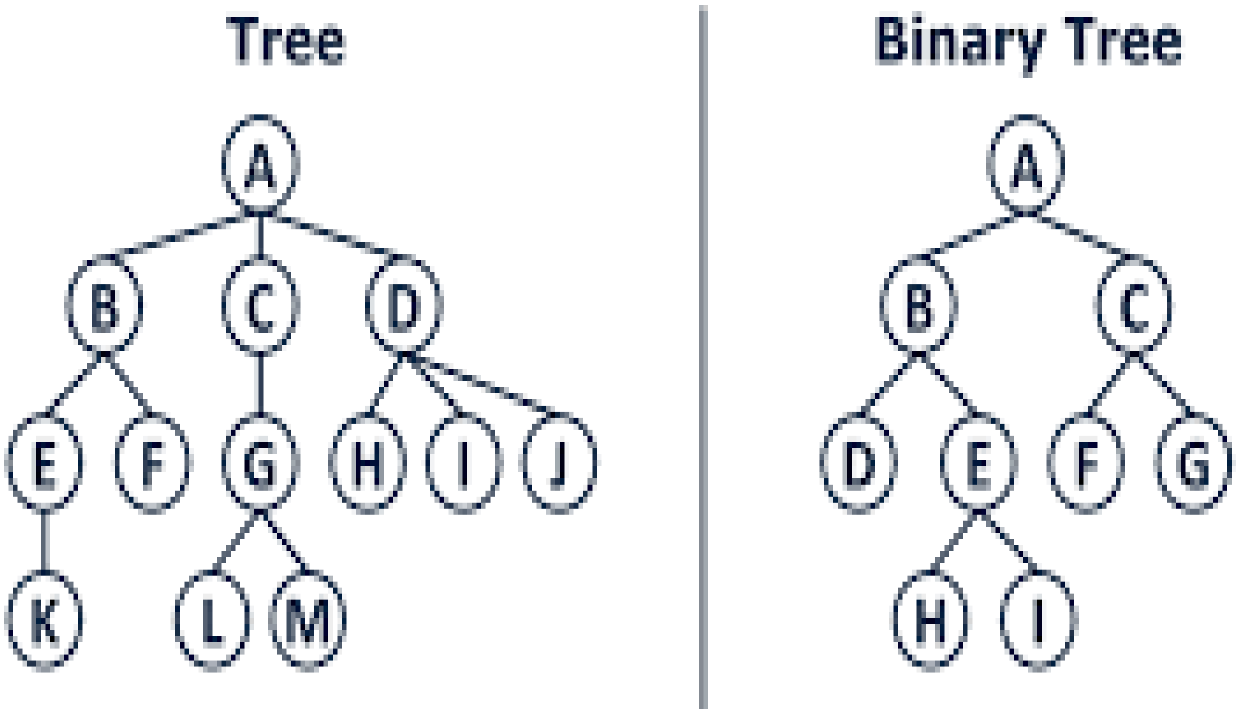
- A binary tree is a finite set of nodes:

1) It is either empty or
2) It consists a node called root with two disjoint binary trees-
   - Left subtree
   - Right subtree

Binary Tree

- **The Maximum degree of any node is 2**
- **Ordered tree with all nodes having at most 2 children**

Binary Tree

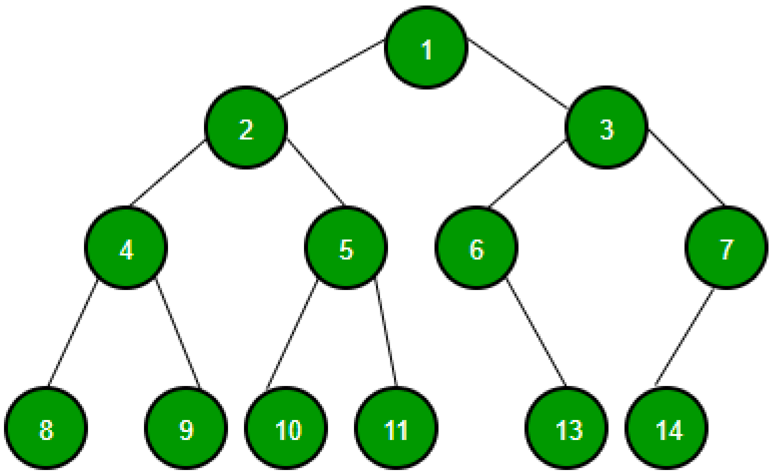


Binary Tree

- Till the discussion so far,
- Linked List had 2 fields in a node-
  - Data field
  - Address field
- Tree has 3 fields in a node-
  - Data field
  - Address of left child
  - Address of right child

Binary Tree

- Graph is used to represent tree
- Every node=Circle
- Line from one node to other=Edge
- The beginning node is called root.



Traversal in Binary Tree

- A tree walk or traversal is a way of visiting all the nodes in the tree in a specified order
- Lets take-

  N=Node

  L=Left Child/Subtree

  R=Right Child/Subtree

Traversal in Binary Tree

In traversal, we have 6 combinations-

- NRL
- NLR
- LRN
- LNR
- RNL
- RLN

But only 3 are standard.

Binary Tree Traversal Methods

- Preorder
- Inorder
- Postorder
- Level Order

Binary Tree Traversal Methods

- A Preorder tree walk processes each node before processing its children
- A Postorder tree walk processes each node after processing its children

Binary Tree Traversal Methods

Preorder

- NLR
- Visit the Root
- Traverse the left subtree of root in Preorder
- Traverse the right subtree of root in Preorder

Inorder

- LNR
- Traverse the left subtree of root in Inorder
- Visit the Root
- Traverse the right subtree of root in Inorder

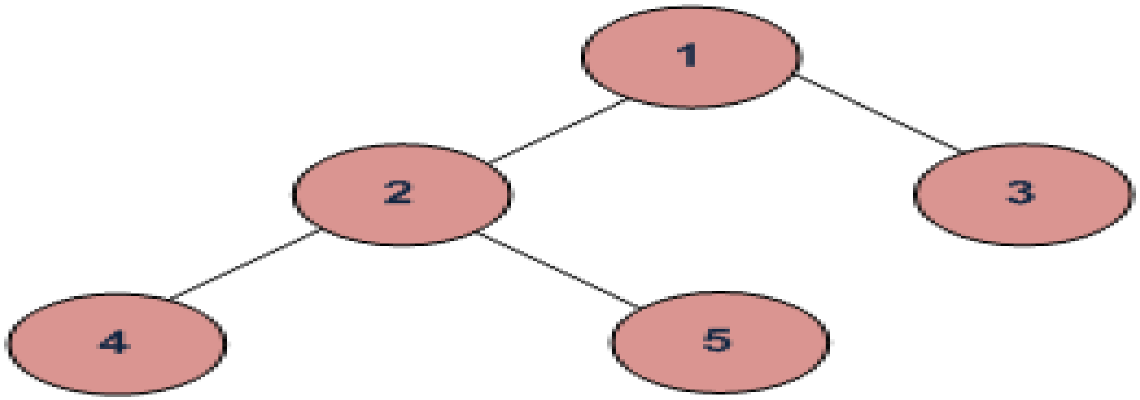
Postorder

- LRN
- Traverse the left subtree of root in Postorder
- Traverse the right subtree of root in Postorder
- Visit the Root

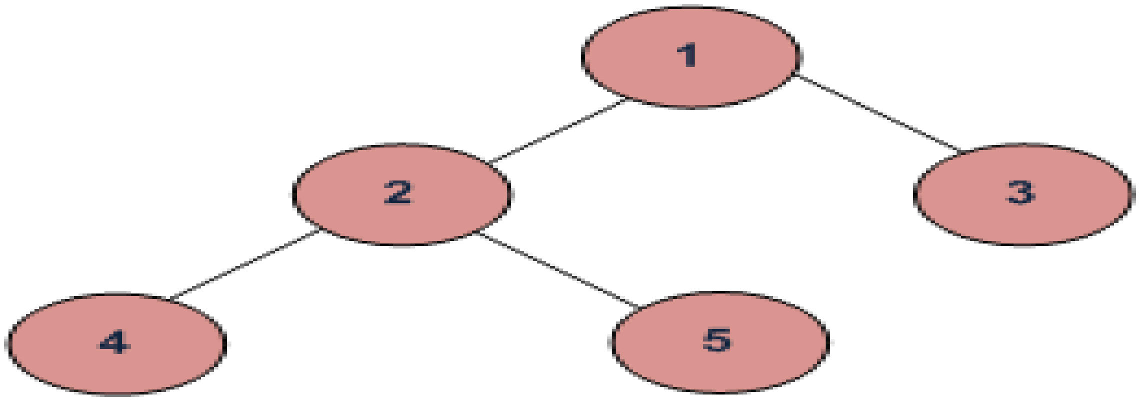
Binary Tree Traversal Methods

- We assume that we are only printing the data in a node when we visit it.

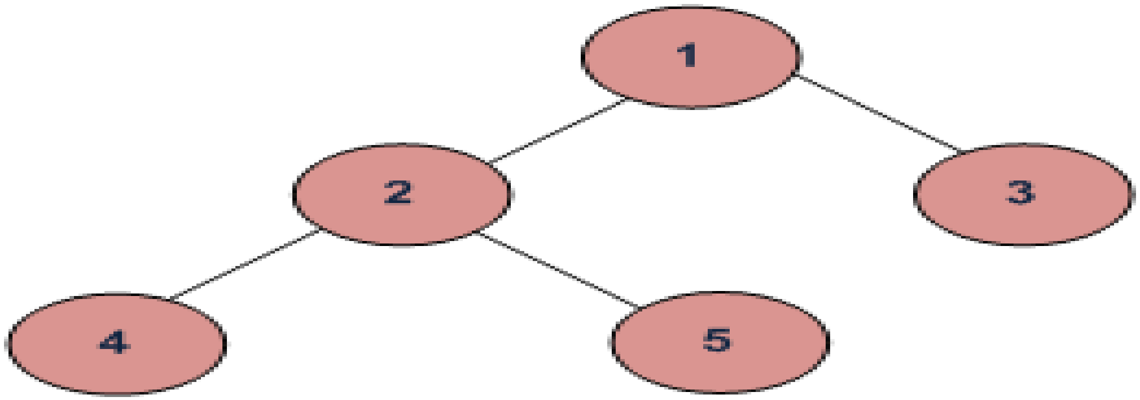
Preorder?



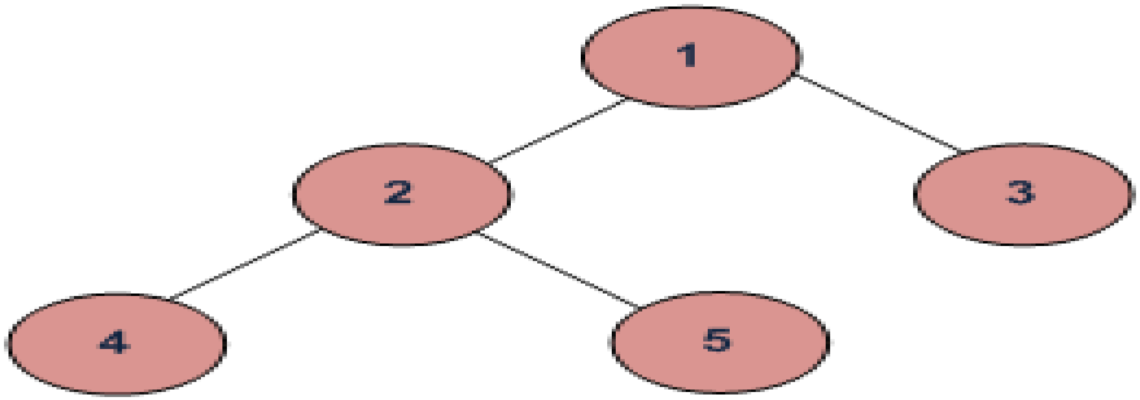
Inorder?



Postorder?

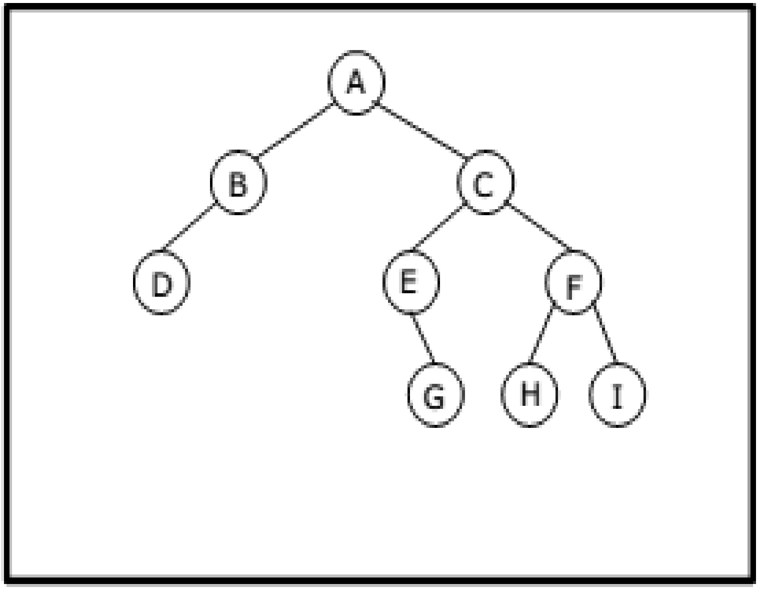


Traversal in Binary Tree



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

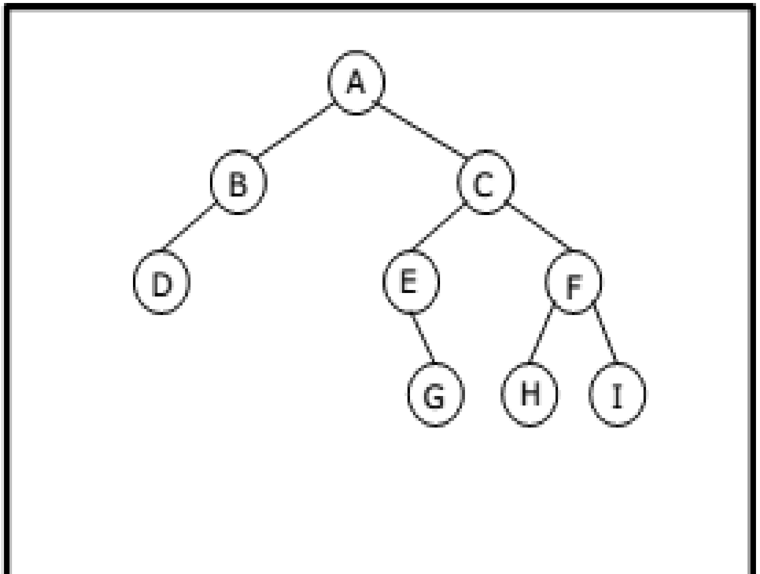
Traversal in Binary Tree



Binary Tree

??

Traversal in Binary Tree

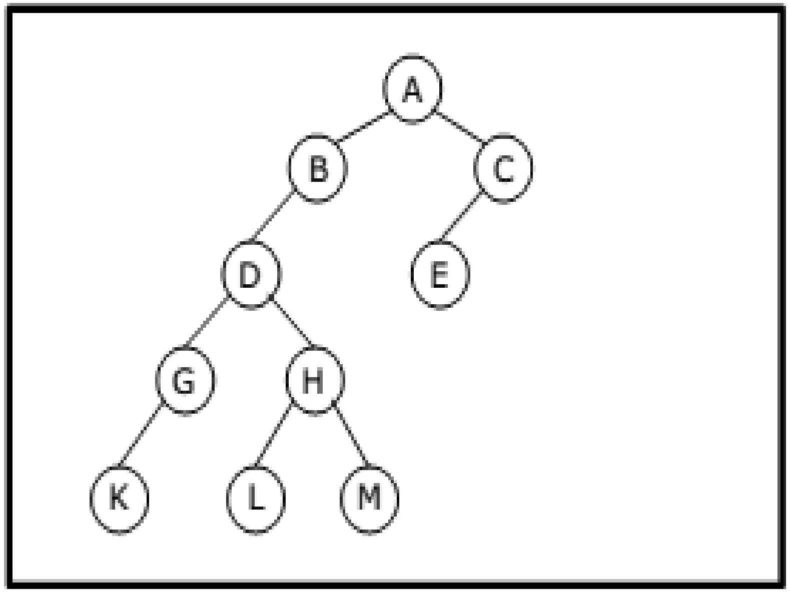


Binary Tree

- Preorder traversal yields:
  A, B, D, C, E, G, F, H, I
- Postorder traversal yields:
  D, B, G, E, H, I, F, C, A
- Inorder traversal yields:
  D, B, A, E, G, C, H, F, I
- Level order traversal yields:
  A, B, C, D, E, F, G, H, I

Pre, Post, Inorder and level order Traversing

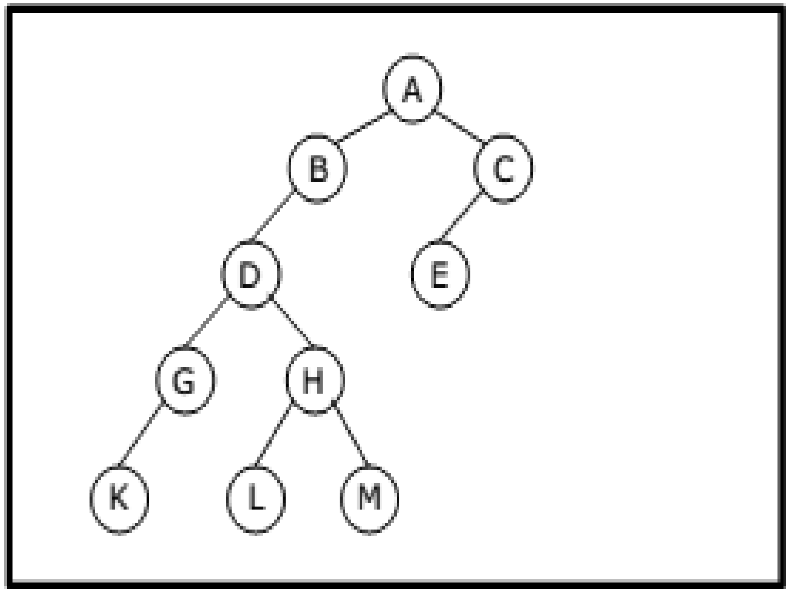
Traversal in Binary Tree



Binary Tree

??

Traversal in Binary Tree

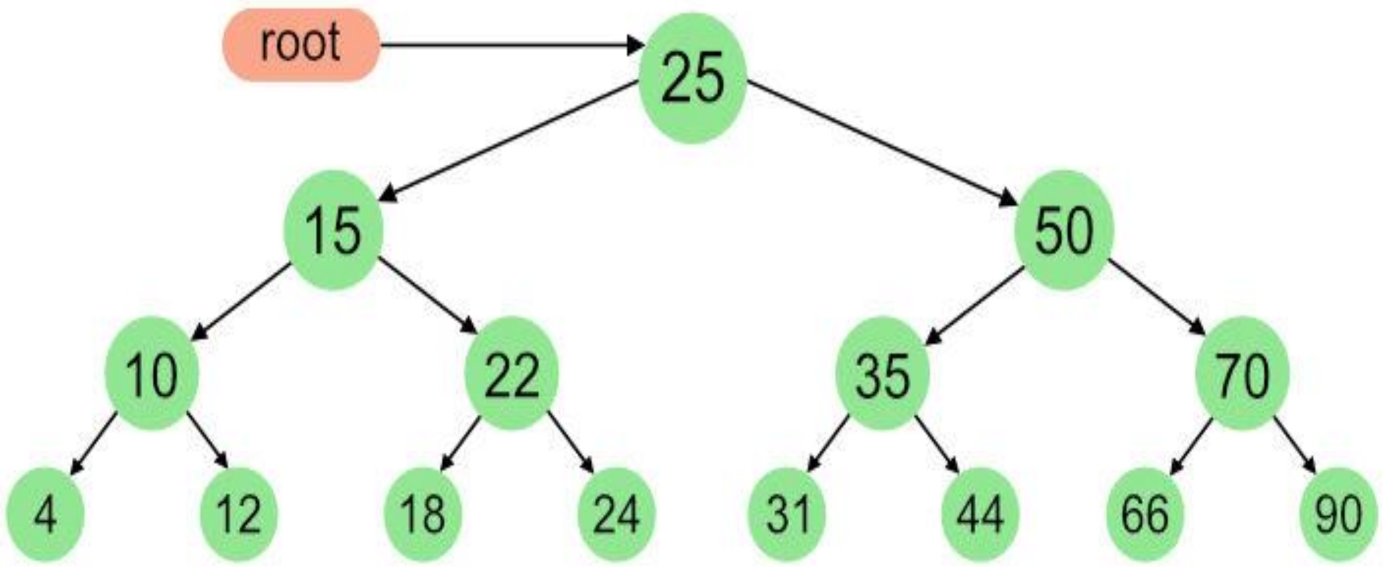


Binary Tree

- Preorder traversal yields:
  A, B, D, G, K, H, L, M, C, E
- Postorder travarsal yields:
  K, G, L, M, H, D, B, E, C, A
- Inorder travarsal yields:
  K, G, D, L, H, M, B, A, E, C

Pre, Post and Inorder Traversing

Traversal in Binary Tree

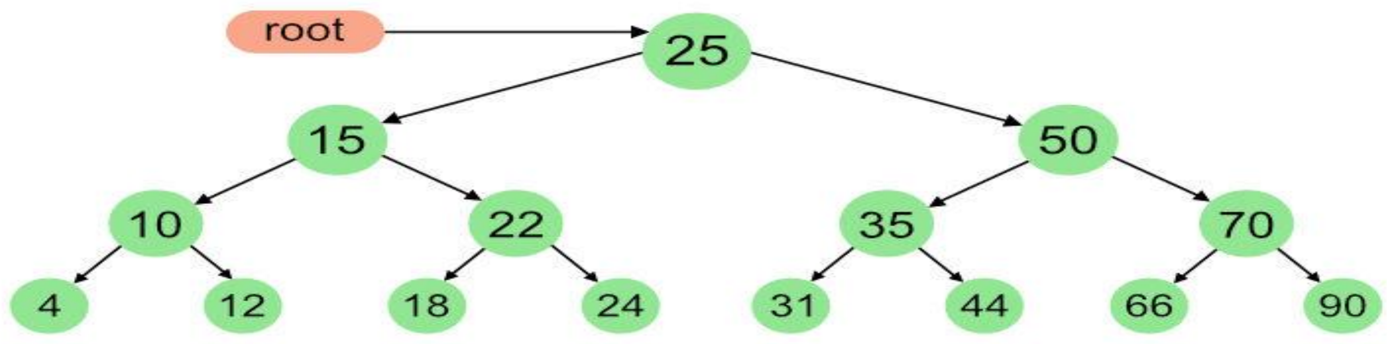


Traversal in Binary Tree

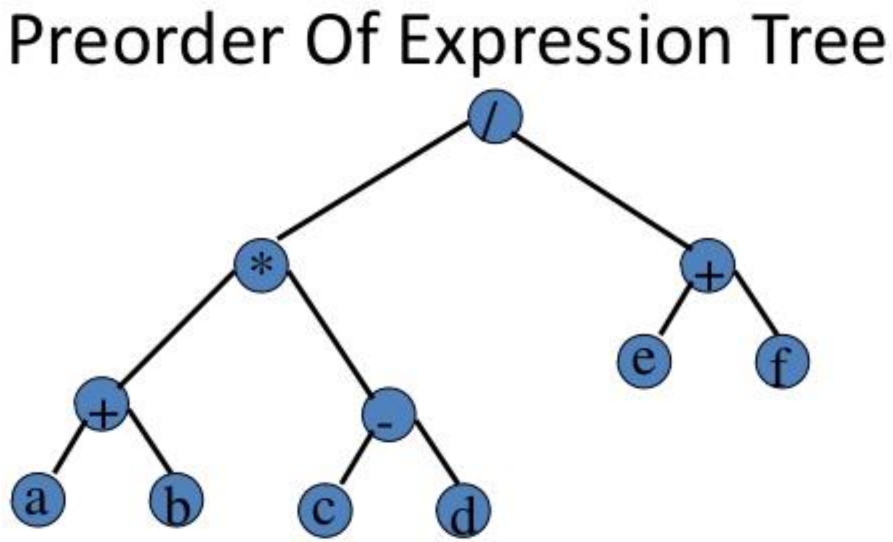
InOrder(root) visits nodes in the following order:
4, 10, 12, 15, 18, 22, 24, 25, 31, 35, 44, 50, 66, 70, 90

A Pre-order traversal visits nodes in the following order:
25, 15, 10, 4, 12, 22, 18, 24, 50, 35, 31, 44, 70, 66, 90

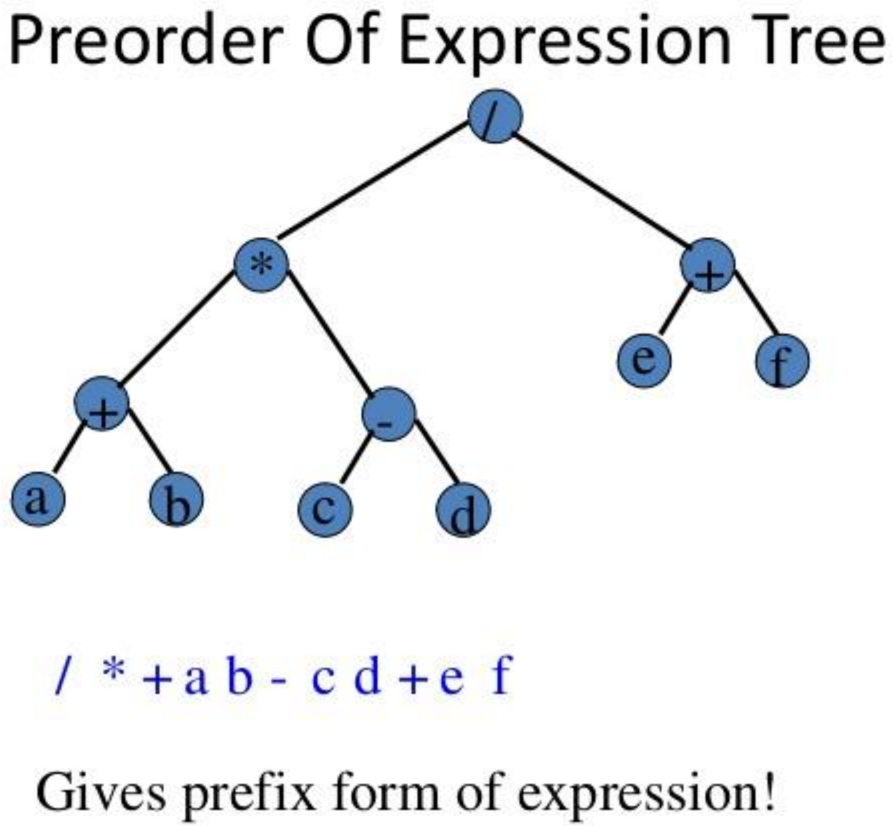
A Post-order traversal visits nodes in the following order:
4, 12, 10, 18, 24, 22, 15, 31, 44, 35, 66, 90, 70, 50, 25



Preorder Traversal in Binary Tree –Prefix Expression



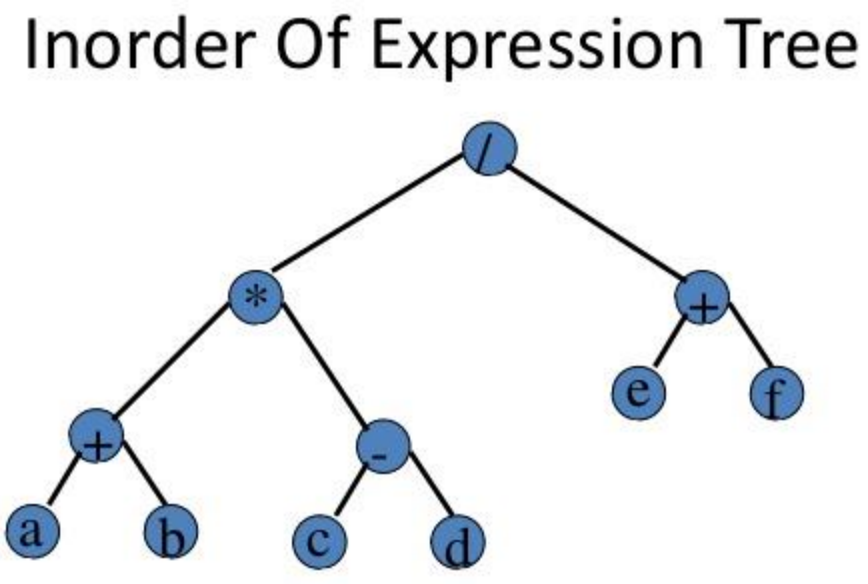
Preorder Traversal in Binary Tree –Prefix Expression



Algebraic Expression representation in tree

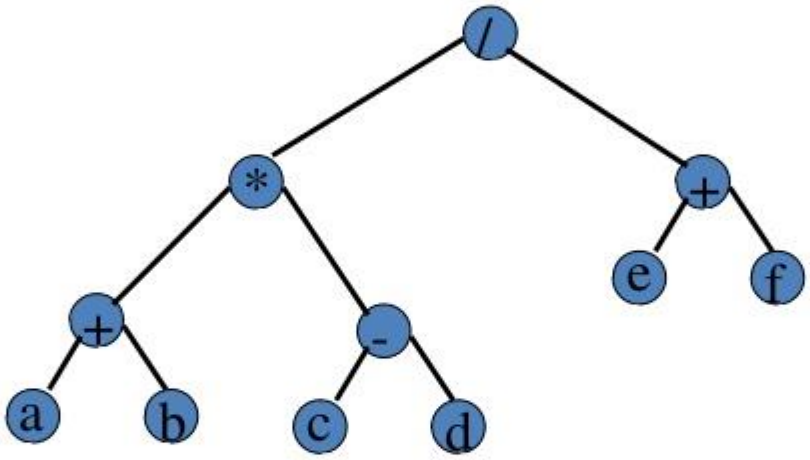
- Any algebraic expression can be represented in binary tree
- **Left, Right Child=Operand of the expression**
- **Parent of the child=Operator**

Inorder Traversal in Binary Tree –Infix Expression



Inorder Traversal in Binary Tree –Infix Expression

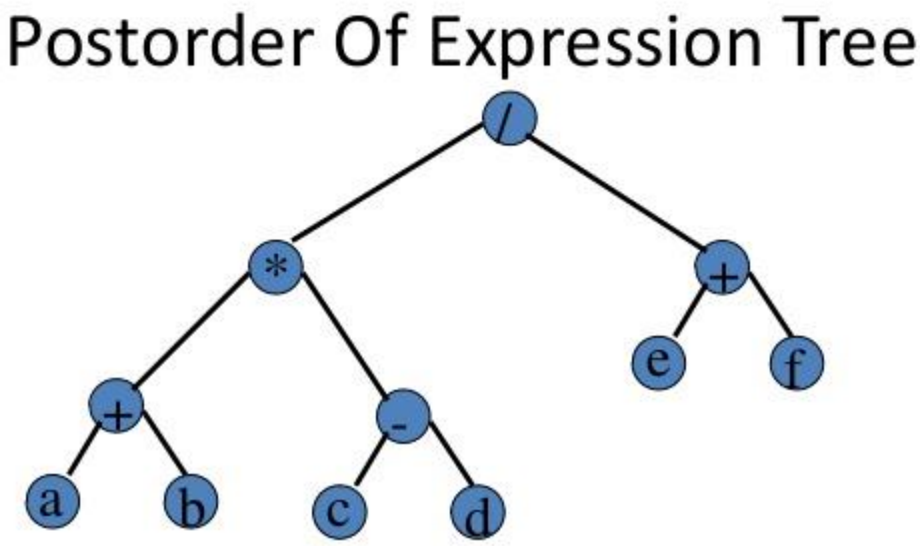
Inorder Of Expression Tree



a + b * c - d / e + f

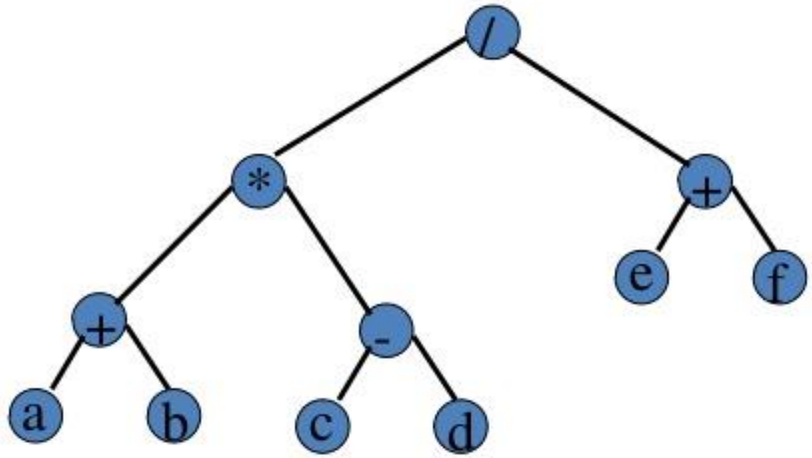
Gives infix form of expression

Postorder Traversal in Binary Tree –Postfix Expression



Postorder Traversal in Binary Tree –Postfix Expression

Postorder Of Expression Tree



a b + c d - * e f + /

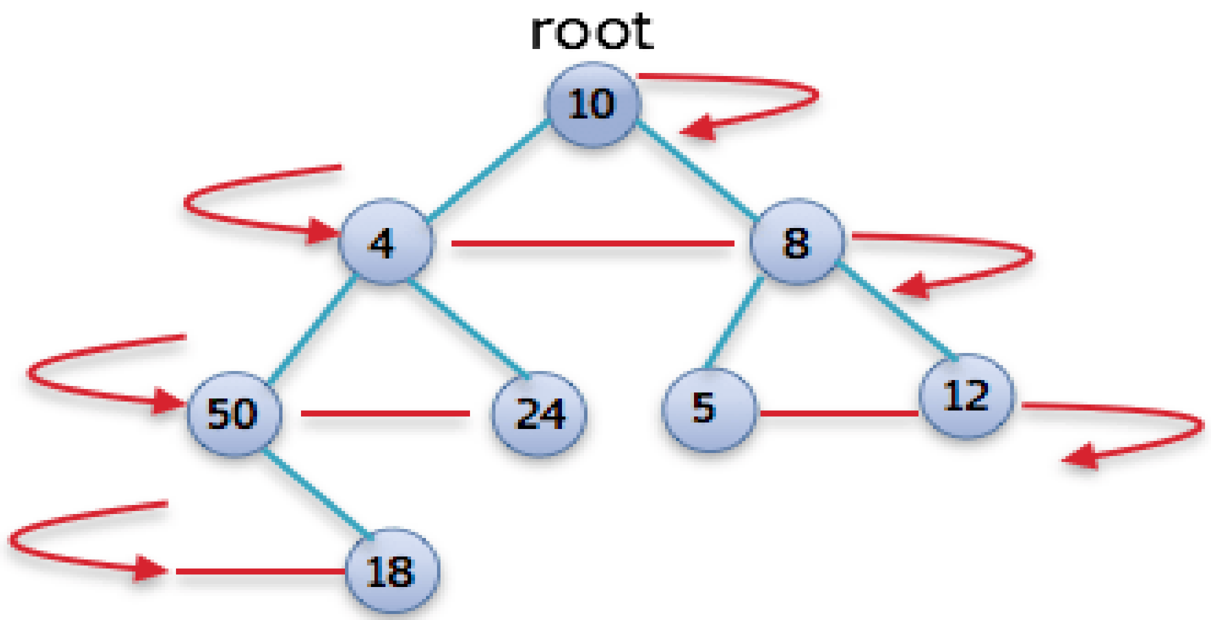
Gives postfix form of expression!

Level order Traversal in Binary Tree

- We traverse the nodes according to their levels.
  - **Traverse Level 0**
  - **Then Traverse all the nodes of Level 1**
  - **Then Traverse all the nodes of Level 2**
  - **And so on…..**

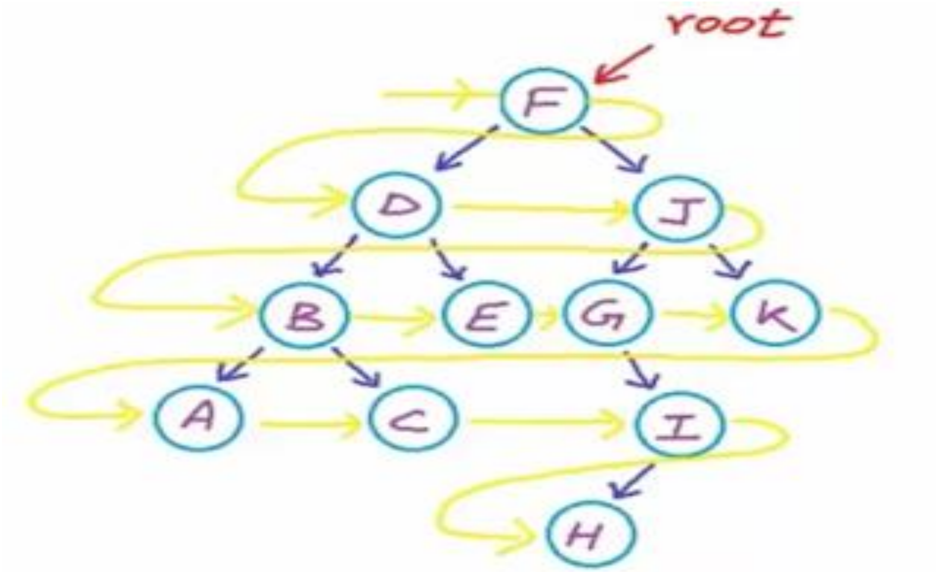
- **Traverse the nodes of a particular level from Left to Right**

Level order Traversal in Binary Tree



Level Order Traversal=10, 4, 8, 50, 24, 5, 12 and 18.

Level order Traversal in Binary Tree



Level Order Traversal=F, D, J, B, E, G, K, A, C I, H

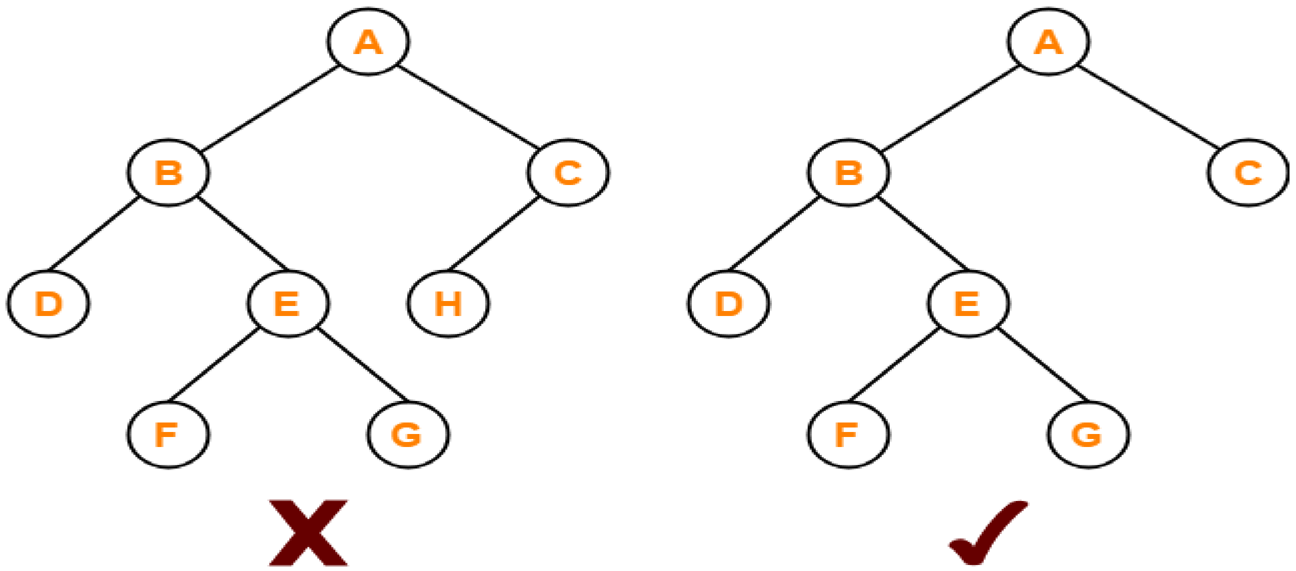
Types of Binary Trees-

Binary trees can be of the following types-

- Rooted Binary Tree/Simple Binary Tree
- Full / Strictly Binary Tree
- Complete / Perfect Binary Tree

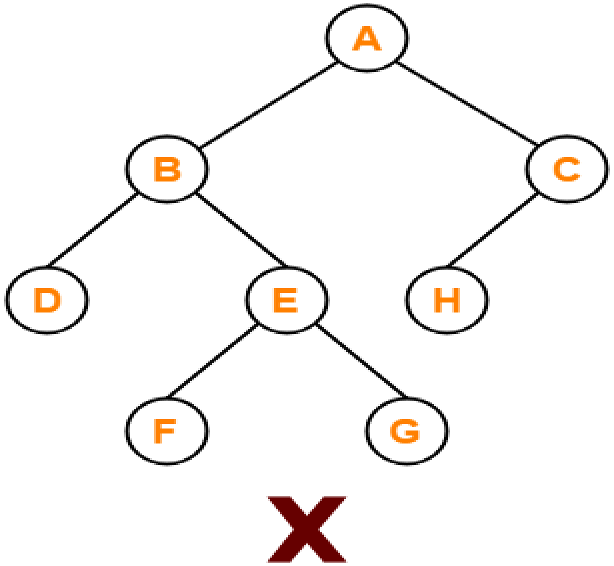
Strictly Binary Tree

- Strict
- If every node is either a Leaf or has 2 children
- **Also called Full binary tree**

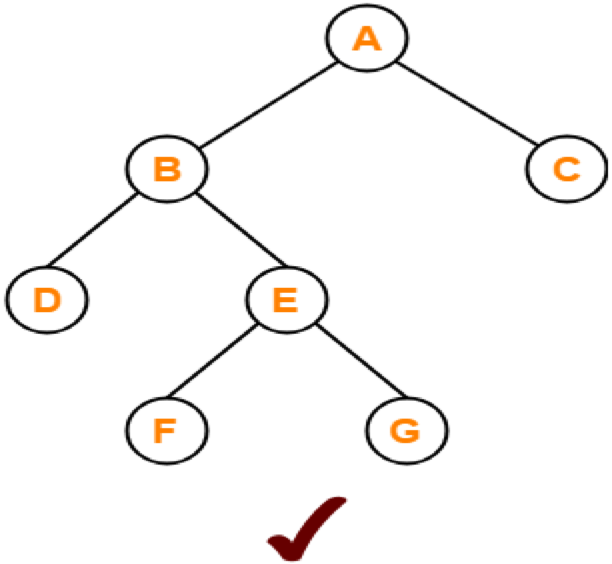


Strictly Binary Tree

- A strictly Binary tree with n leaf nodes always contains 2n-1 total no of Nodes



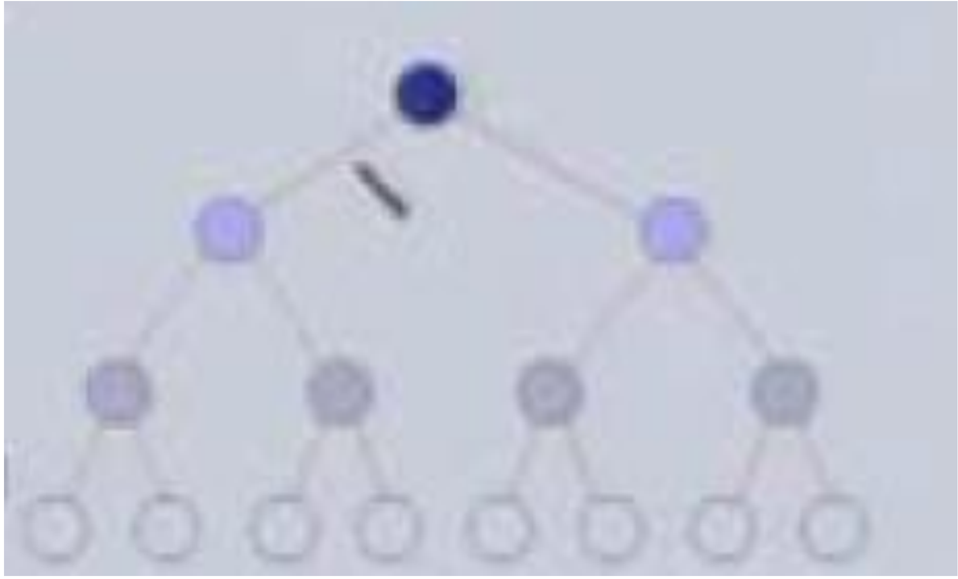
no of leaves=4
2n-1=7



no of leaves=4
2n-1=7

Completely Binary Tree

- In a CBT tree of height h, All whose leaves are at Level h



Completely Binary Tree

No of nodes at Level 0=2^0

No of nodes at Level 1=2^1

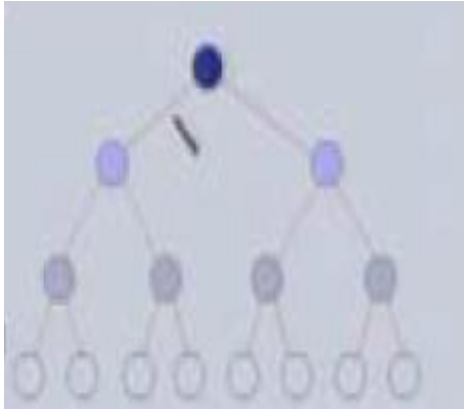
No of nodes at level 2=2^2

|

|

|

No of nodes at level h=2^h



Completely Binary Tree

No of nodes at Level 0=2^0

No of nodes at Level 1=2^1

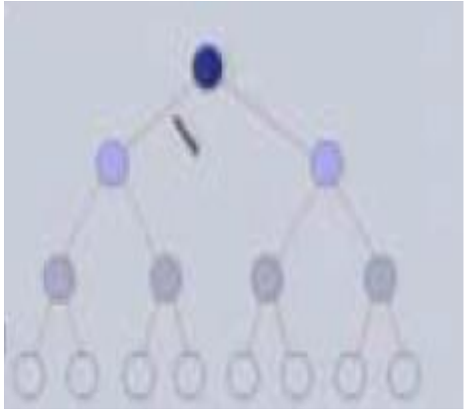
No of nodes at level 2=2^2

|

|

|

No of nodes at level h=2^h



Total Number of nodes in a complete Binary Tree of depth h=$2^0+2^1+2^2+$ ---------2^h

$=2^{h+1}-1$

Using Sum of Geometric Progression Series- $S_n = \dfrac{a(r^n-1)}{r-1}$

Completely Binary Tree

Let the total nodes in tree =n, then

$n=2^{h+1}-1$

$2^{h+1}=n+1$

Using log:

$\log_2(n+1)=h+1$

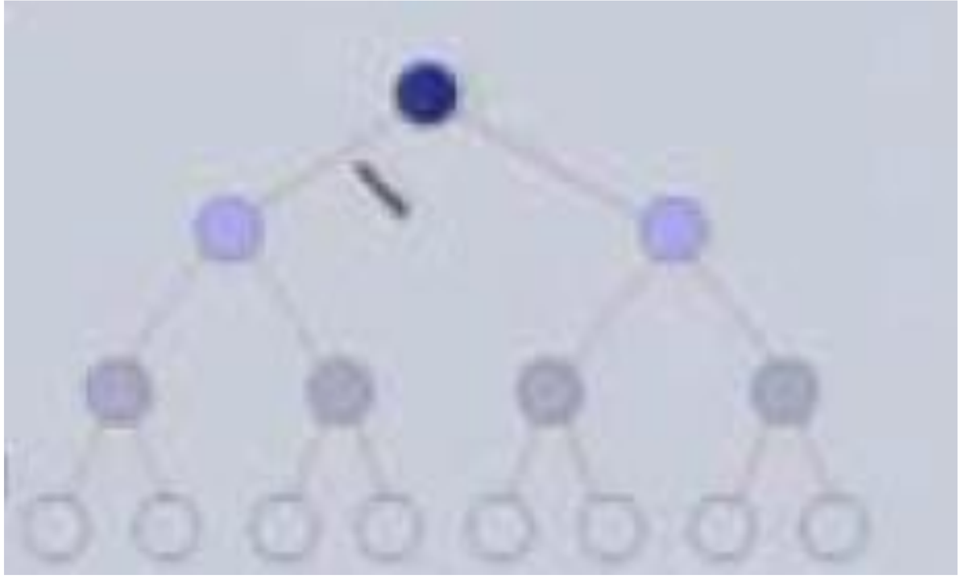
$\log_2(n+1)-1=h$

i.e. Height=h=$\log_2(n+1)-1$

Height & No of Nodes

Completely Binary Tree

In Complete Binary Tree, no of leaves=(n+1)/2
n=total no of nodes



n=15
no of leaves=16/2=8

Completely Binary Tree

Let the total nodes in tree =n, then

$n=2^{h+1}-1$

$2^{h+1}=n+1$

$2^h*2=n+1$

$2^h=(n+1)/2$

In Complete Binary Tree, no of leaves=(n+1)/2

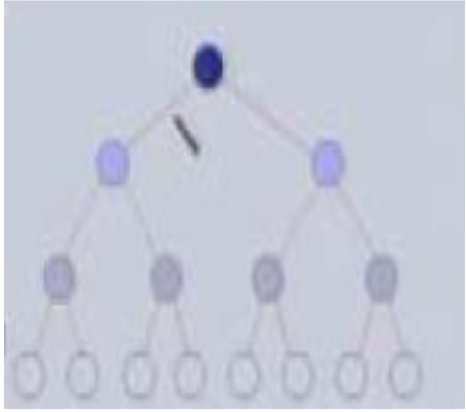
Thus,

$h=\log_2(n+1)/2$

Height=h=$\log_2$(No of leaves)

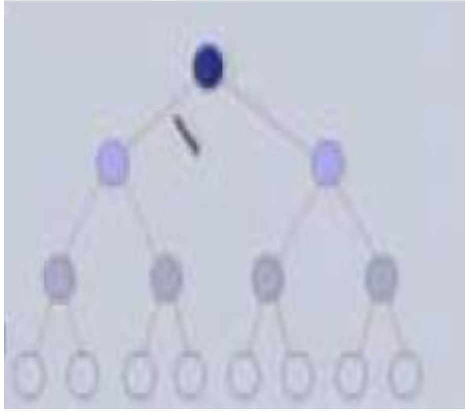
Height & No of Leaves

Completely Binary Tree



- No of internal nodes=no of leaves-1
- No of leaves=8
- No of Internal Nodes=8-1=7

Completely Binary Tree



- **Level I has 2^i nodes**
- So at level h, no of nodes= 2^h
- No of leaves=2^h
- No of internal node=$2^0+2^1+2^2+\ldots\ldots+2^{h-1}=2^h-1$

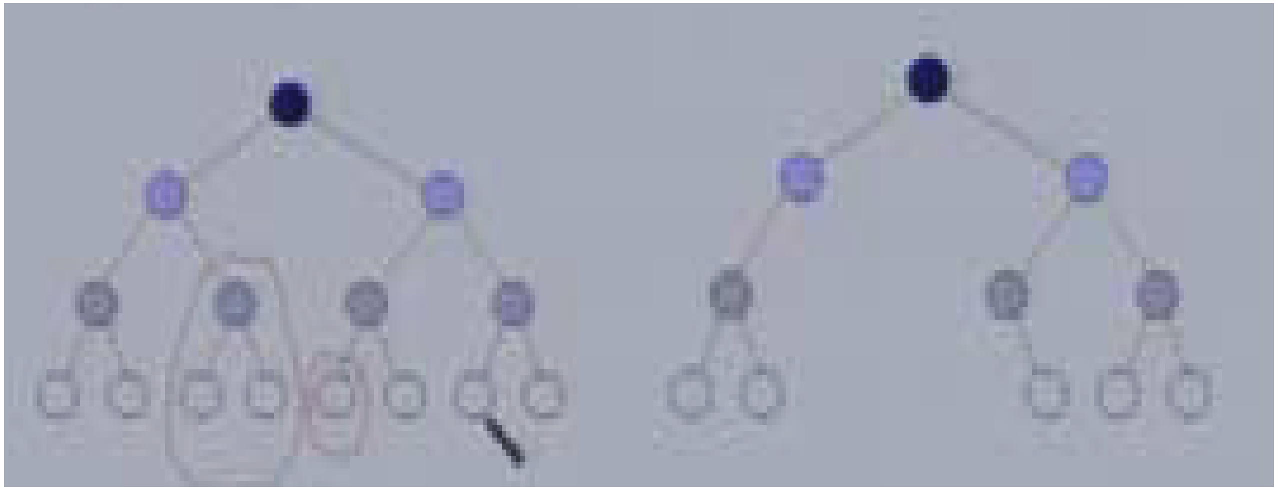
Using Sum of Geometric Progression Series- $S_n = \dfrac{a(r^n - 1)}{r - 1}$

- **No of internal nodes=no of leaves-1**

Complete Binary Tree

- A binary tree can be obtained from an complete binary tree by pruning

- Take a complete binary tree, cut off some branches then you will get a binary tree.



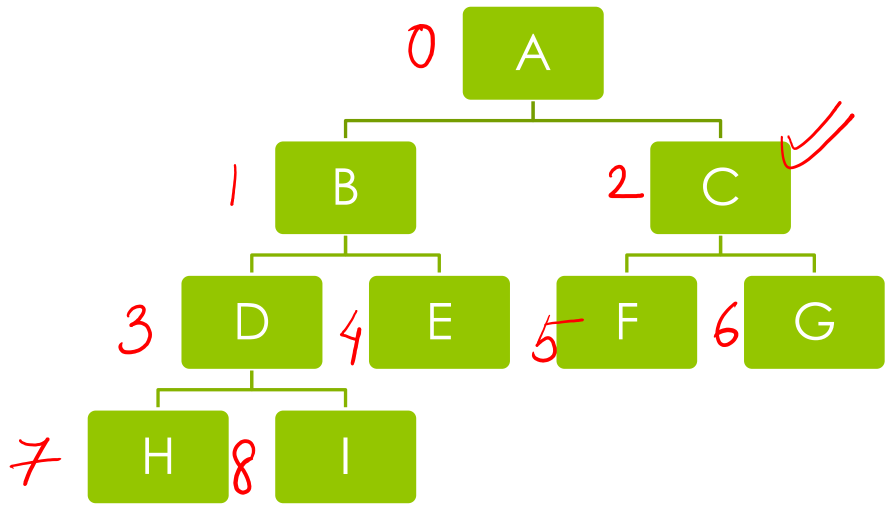
Representation of a Binary Tree

- **Array Representation**
- **Linked List Representation**

Array Representation of a Binary Tree

- Using 1-D Array
- **Nodes are numbered sequentially level by level left to right.**
- **Even empty nodes are numbered**
- When data of the tree is stored in an array then the number appearing against the node will work as indices of the node in the array

Array Representation of a Binary Tree



for C,

$p = 2$

$2p+1 = 5$ = left child

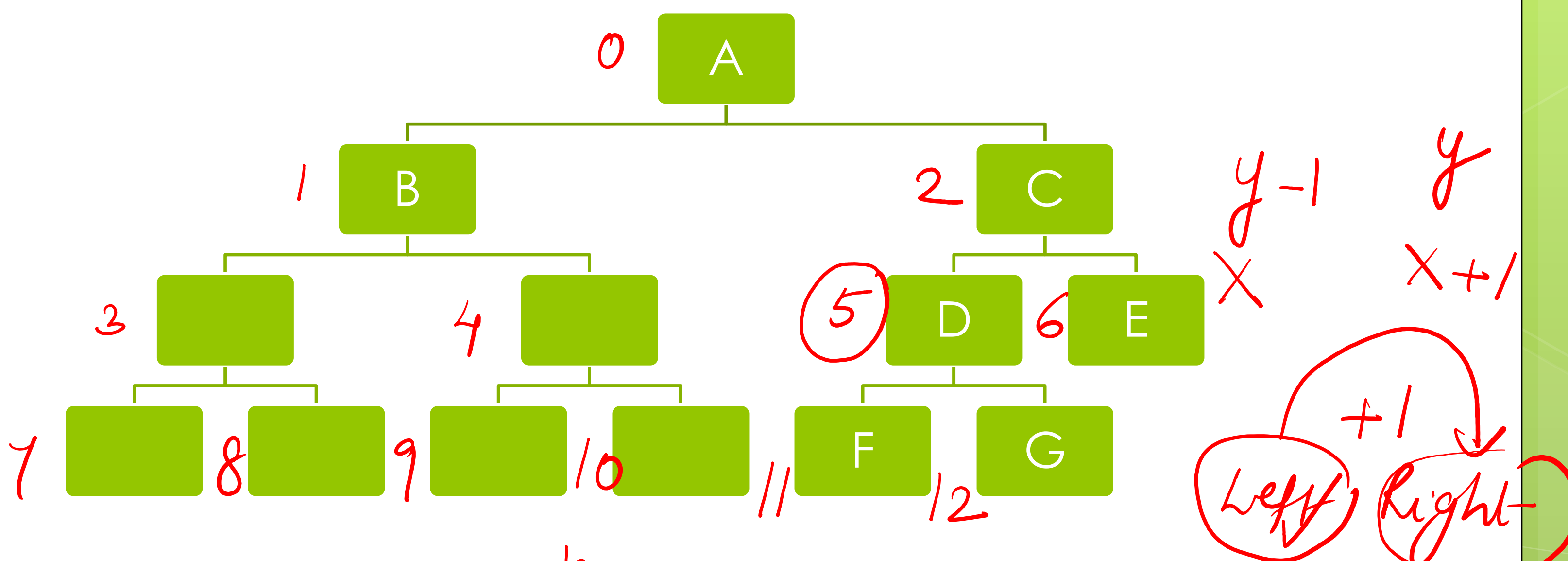
$2p+2 = 6$ = Right child

index

p

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
A	B	C	D	E	F	G	H	I

Array Representation of a Binary Tree

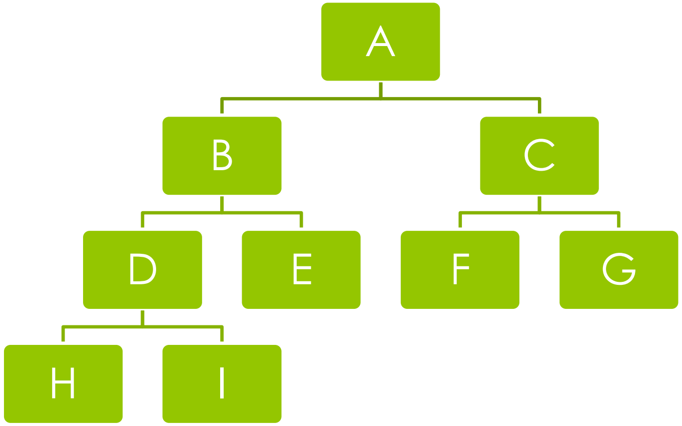


p

2p+1 2p+2

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]
A	B	C			D	E					F	G

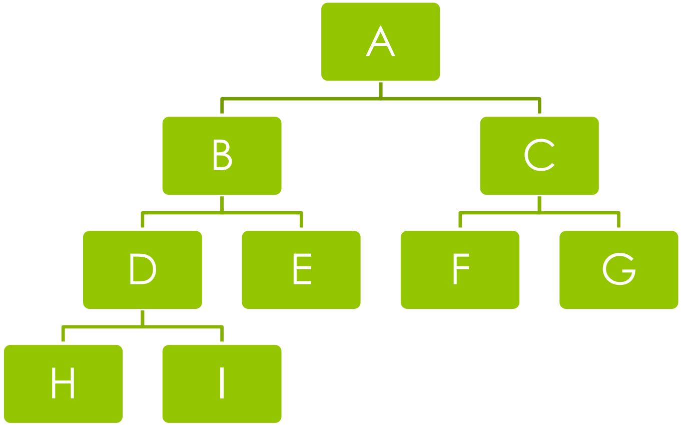
Array Representation of a Binary Tree



[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
A	B	C	D	E	F	G	H	I

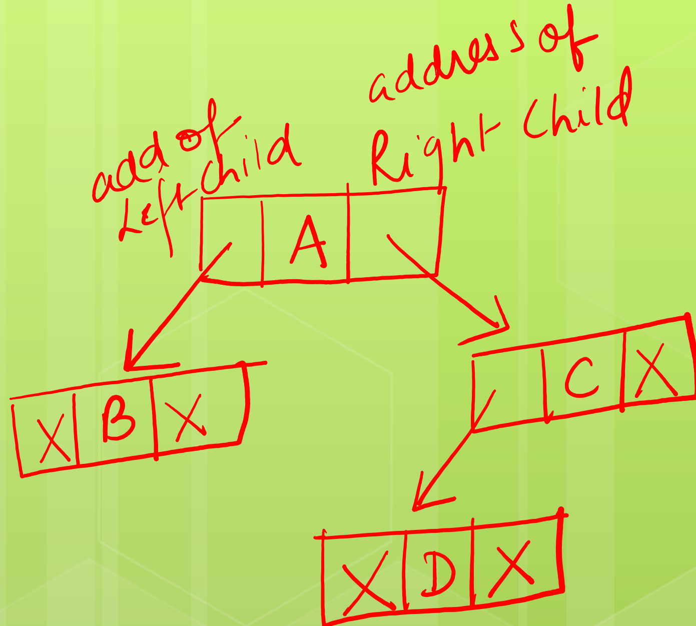
- Node in position p
  - Is the implicit father of nodes 2p+1 and 2p+2
  - **Left child=2p+1**
  - **Right child=2p+2**

Array Representation of a Binary Tree



[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
A	B	C	D	E	F	G	H	I

- Given a Left child at position p then
  - **Right brother=p+1**
- Given a Right child at position p then
  - **Left brother=p-1**



1) data
2) Ad of LC
3) Ad of RC

14-11-2023

Linked Representation of Binary Trees

Recursive functions for Traversal in Binary Tree

Structure for Binary Tree Node

```
struct node
{
        int num;
        struct node *left;
        struct node *right;
};
```

A B

Preorder Traversal in Binary Search Tree

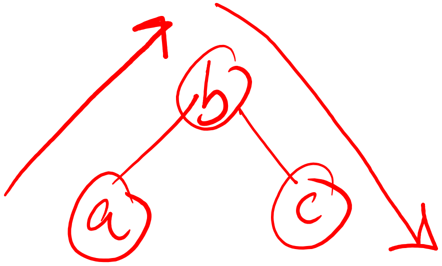

```
void preorder(struct node *tree)
{
    if(tree!=NULL)
    {
        printf("%d\n",tree->num);      N
        preorder(tree->left);          L
        preorder(tree->right);         R
    }
}
```

✓ root is not empty

preorder(root);
if(root!=NULL){
yes
print A
print root→data
preorder(root→left);
if(root→left!=NULL)
yes
print B
preorder(p→left);
if(p→left!=NULL)
false
preorder(p→right)
if(p→right)!=NULL
false

preorder(root→right);

Inorder Traversal in Binary Search Tree




```
void inorder(struct node *tree)
{
        if(tree!=NULL)
          {
                inorder(tree->left);   L
                printf("%d\n",tree->num); N
                inorder(tree->right);  R
          }
}
```

Postorder Traversal in Binary Search Tree

```
void postorder(struct node *tree)        LRN
{
        if(tree!=NULL)
         {
                postorder(tree->left);   L
                postorder(tree->right);  R
                printf("%d\n",tree->num);  N
         }

}
```

14-11-2023

Binary Tree Construction

Binary Tree Construction

- Can you construct Binary Tree , Given two traversals

Binary Tree Construction

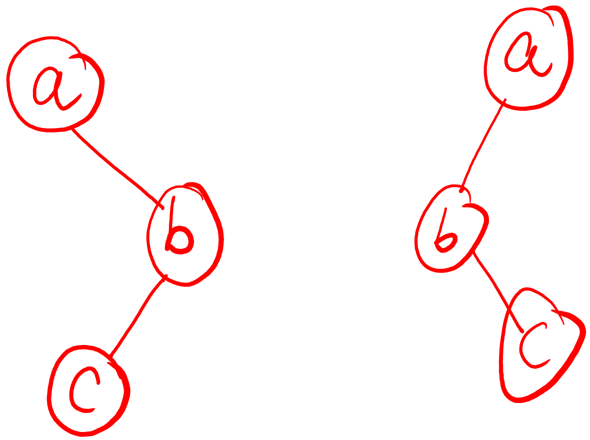
- Can you construct Binary Tree , Given two traversals
- Yes

Binary Tree Construction

- **Depends on which 2 sequences are given**

- **Can be constructed , the following combination is given-**
  - **Preorder + Inorder**

    or
  - **Postorder + Inorder**



Binary Tree Construction

Usage-

- **Preorder/Postorder= To find out the root**
- **Inorder=To find the left and right subtree/child**

14-11-2023

Binary Tree Construction=In+PostOrder

Binary Tree Construction=In+PostOrder

Postorder=LRN

Postorder=HDIEBJFKLGCA

Inorder=HDBIEAFJCKGL

Binary Tree Construction=In+PostOrder

Postorder=LRN

Postorder=HDIEBJFKLGCA

Inorder=HDBIEAFJCKGL

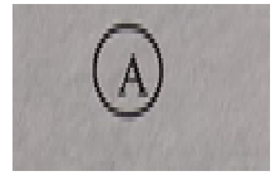
Binary Tree Construction=In+PostOrder

Postorder=LRN

Postorder=HDIEBJFKLGCA

Inorder=HDBIEAFJCKGL

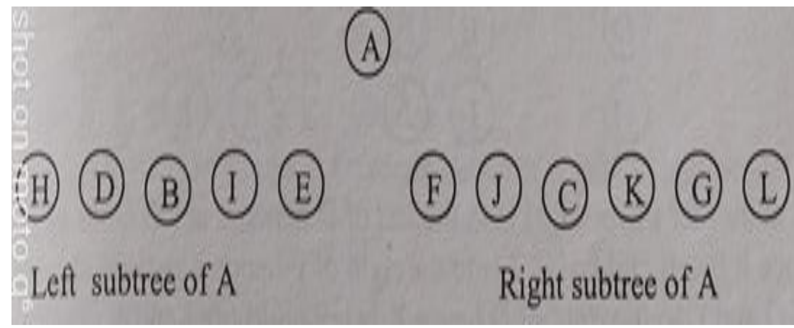
1) In Postorder traversal, The Last Node is the Root. So, **A is the Root Node**



2) From Inorder traversal,

Inorder=HDBIEAFJCKGL

Left Subtree **Right Subtree**



- Left subtree=Nodes on Left of the Node=HDBIE
- Right subtree=Nodes on the Right of the node=FJCKGL

Binary Tree Construction=In+PostOrder

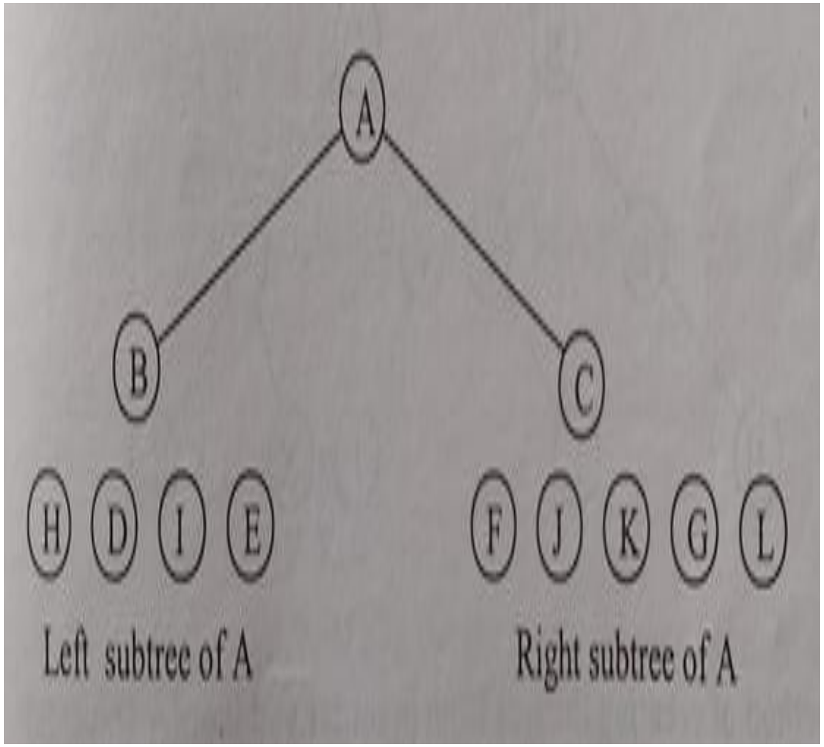
Postorder=HDIEBJFKLGCA
Inorder=HDBIEAFJCKGL

1) In Postorder traversal,

- **HDIEB JFKLGC A**

Left Subtree **Right Subtree**

1) Now Right child of A will be the node which comes just before node A.
2) C is the right child
3) Left child of A will be the first node before nodes of right subtree in postorder traversal.
4) B is the left child



Binary Tree Construction=In+PostOrder

Postorder=HDIEBJFKLGCA

Inorder=HDBIEAFJCKGL

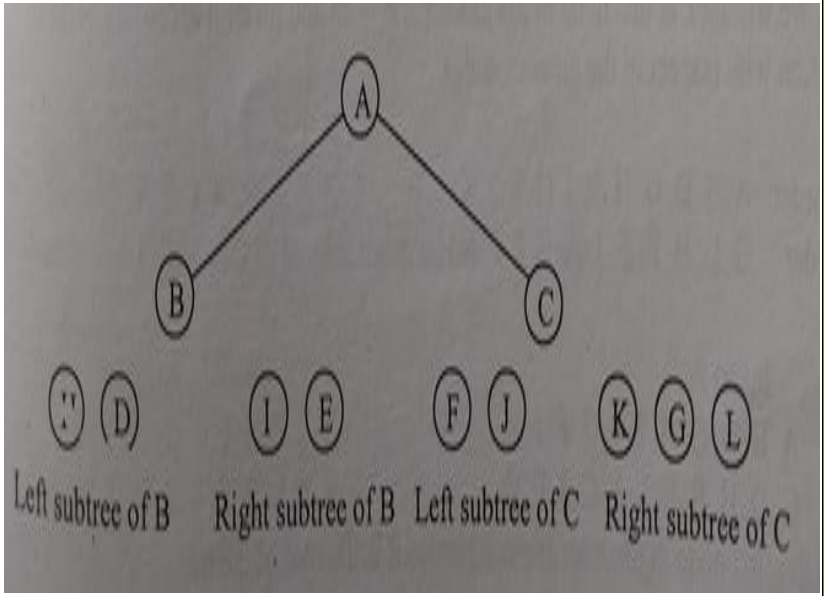
1) Now Look at C in InOrder Traversal , **Inorder=HDBIEAFJCKGL**

Left Subtree **Right Subtree**

2) **So for C**
3) **Left subtree=FJ**
4) **Right subtree=KGL**
5) Now Look at B in InOrder Traversal , **Inorder=HDBIEAFJCKGL**

Left Subtree **Right Subtree**

6) **So for B**
7) **Left subtree=HD**
8) **Right subtree=IE**



Binary Tree Construction=In+PostOrder

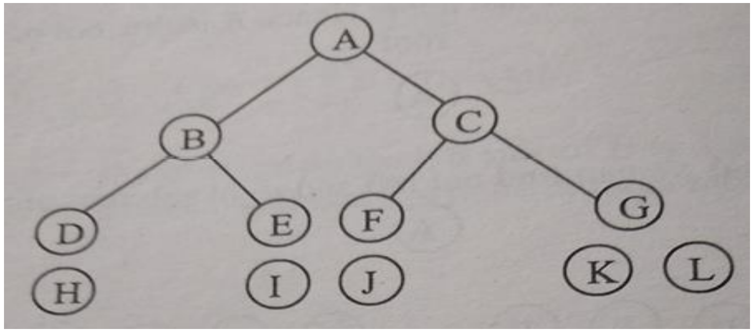
Postorder=HDIEBJFKLGCA

Inorder=HDBIEAFJCKGL

1) Now Look at node just before B in PostOrder Traversal ,
2) **Postorder=HDIEBJFKLGCA**

Left Subtree **Right Subtree**

3) **E is right child**
4) Left child of A will be the first node before nodes of right subtree in postorder traversal.
5) **D is the left child**



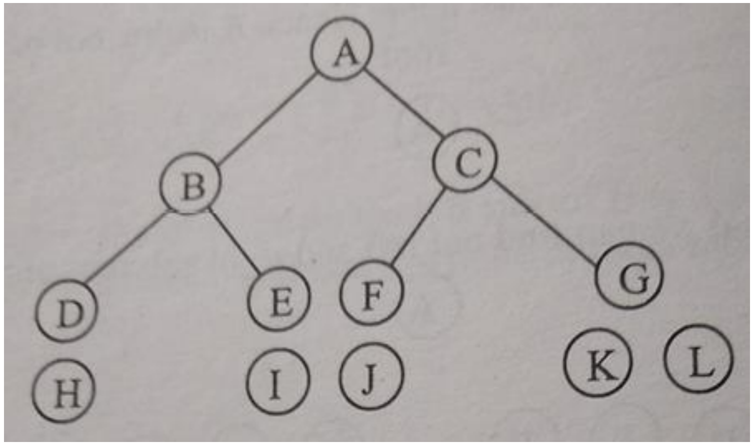
Binary Tree Construction=In+PostOrder

Postorder=HDIEBJFKLGCA

Inorder=HDBIEAFJCKGL

1) Now Look at node just before C in PostOrder Traversal ,
2) **Postorder=HDIEBJFKLGCA**

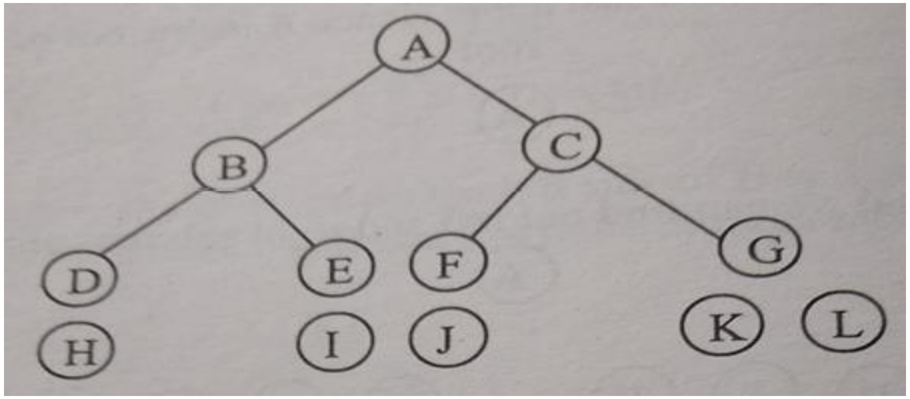
Left Subtree **Right Subtree**



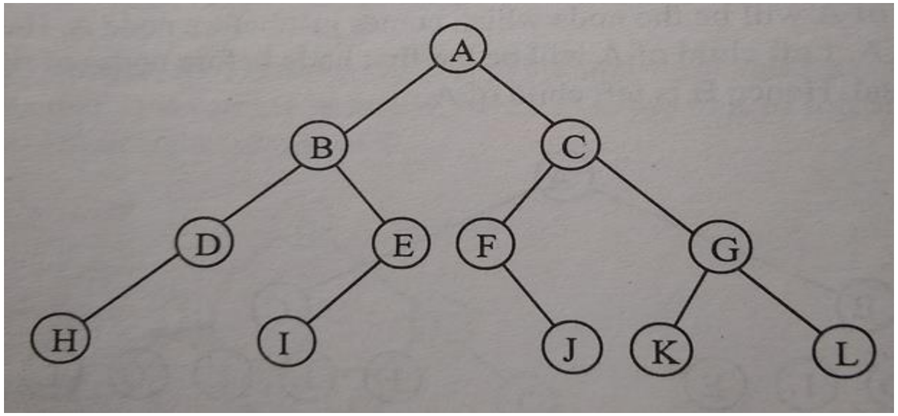
3) **G is right child**
4) Left child of C will be the first node before nodes of right subtree in postorder traversal.
5) **F is the left child**

Binary Tree Construction=In+PostOrder

Postorder=LRN
Postorder=HDIEBJFKLGCA
Inorder=HDBIEAFJCKGL



1) From Inorder traversal,
   - H is to the left of D so
   - **Left child of D =H**
   - I is to the Left of E
   - **Left child of E =I**



2) From Inorder traversal,
   - J is to the right of F so
   - **Right child of F =J**
   - K is to the Left of G
   - L is to the right of G
   - **Left child of G =K**
   - **Right child of G =L**

14-11-2023

Binary Tree Construction=In+PreOrder

Binary Tree Construction=In+PreOrder

inorder = g d h b e i a f j c

preorder = a b d g h e i c f j

Binary Tree Construction=In+PreOrder

inorder = g d h b e i a f j c

preorder = a b d g h e i c f j

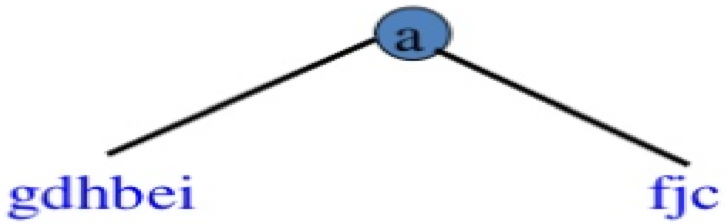
Binary Tree Construction=In+PreOrder

inorder = g d h b e i a f j c

preorder = a b d g h e i c f j

Preorder=NLR

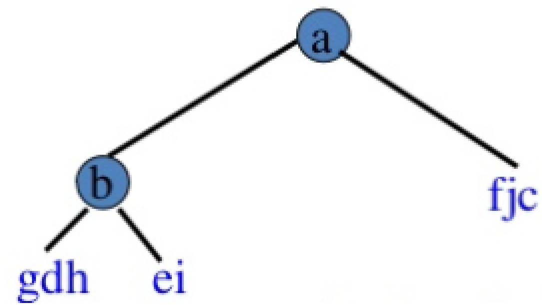
1) In Preorder traversal, The First Node is the Root.
2) From Inorder traversal-Find the left subtree and right subtree
   - Nodes on the left of the root in Inorder =Left Subtree
   - Nodes on the right of the root in Inorder =Right Subtr



Binary Tree Construction=In+PreOrder

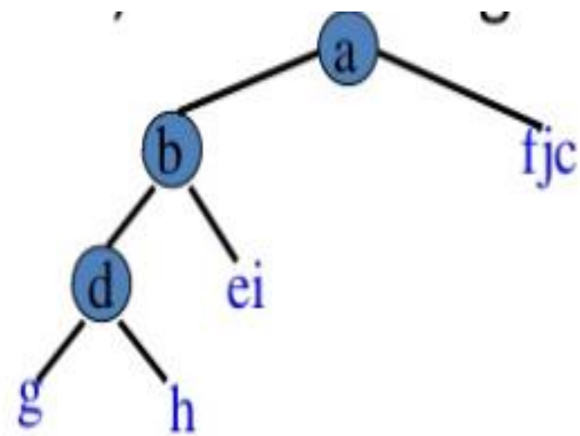
Preorder=bdgheicfj
Inorder=gdh b ei a fjc

1) b is the next root
2) Left subtree=gdh
3) Right Subtree=ei



Preorder=dgheicfj
Inorder=gdh b ei a fjc

1) d is the next root
2) g is on left of d
3) Left subtree=g
4) h is on right of d
5) Right Subtree=h

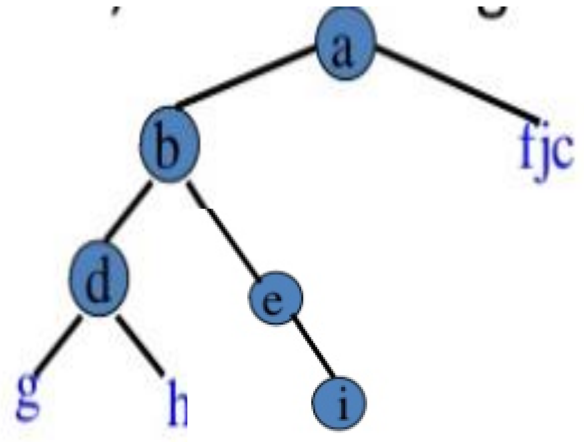


inorder = g d h b e i a f j c
preorder = a b d g h e i c f j

Binary Tree Construction=In+PreOrder

Preorder=eicfj
Inorder=gdh b ei a fjc

1) e is the next root
2) Left subtree=NULL
3) Right Subtree=i



Preorder=cfj
Inorder=gdh b ei a fjc

1) c is the next root
2) Left subtree=fj
3) Right Subtree=NULL

inorder = g d h b e i a f j c
preorder = a b d g h e i c f j

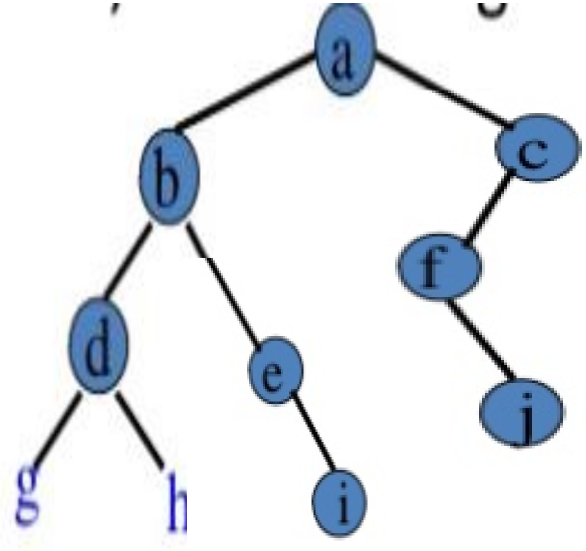
Binary Tree Construction=In+PreOrder

Preorder=fj

Inorder=gdh b ei a fjc

1) f is the next root
2) Left subtree=NULL
3) Right Subtree=j

Done



inorder = g d h b e i a f j c

preorder = a b d g h e i c f j

Can Binary Tree be constructed using Preorder + Postorder Traversals?

Can Binary Tree be constructed using Preorder + Postorder Traversals?No

Why?? Any Guesses?

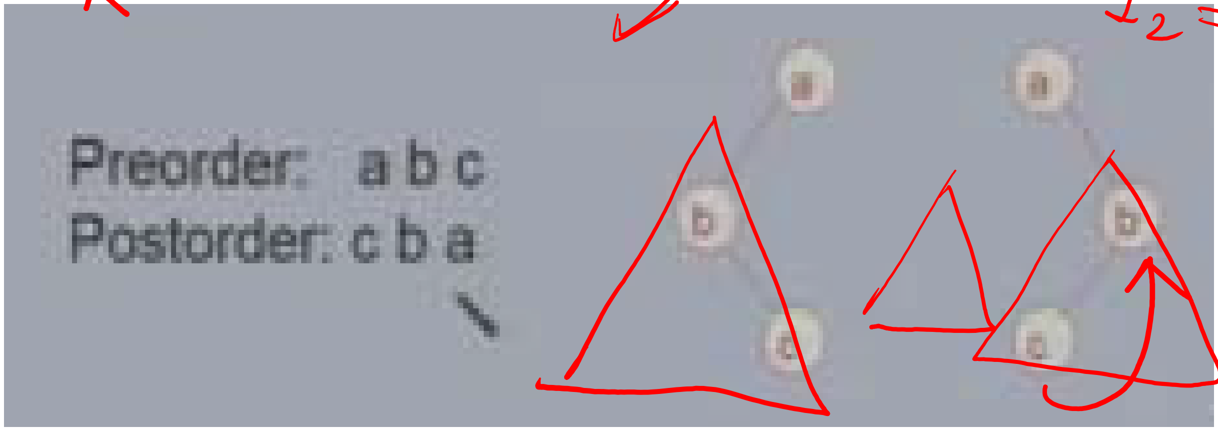
Binary Tree Construction

- Given the Preorder and Postorder traversal of binary tree, we cannot uniquely identify the tree.

- This is because there can be two trees, with the same preorder and postorder traversals

Inorder = LNR

I_2 = acb



- bca

I_1 =

Binary Tree Construction

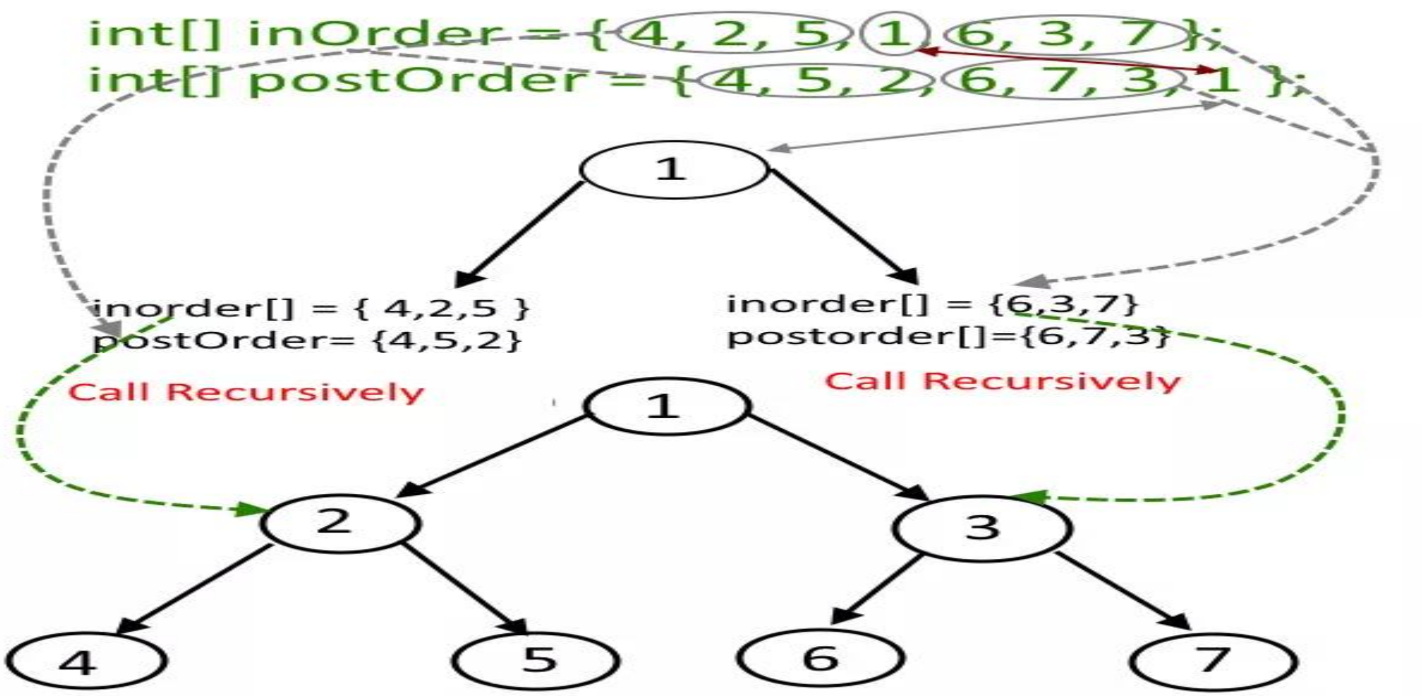
Construct the Binary Tree corresponding to the following traversals

- inOrder = { 4, 2, 5, 1, 6, 3, 7 }
- postOrder = { 4, 5, 2, 6, 7, 3, 1 }

Binary Tree Construction

Construct the Binary Tree corresponding to the following traversals

- inOrder = { 4, 2, 5, 1, 6, 3, 7 }
- postOrder = { 4, 5, 2, 6, 7, 3, 1 }



Binary Tree Construction

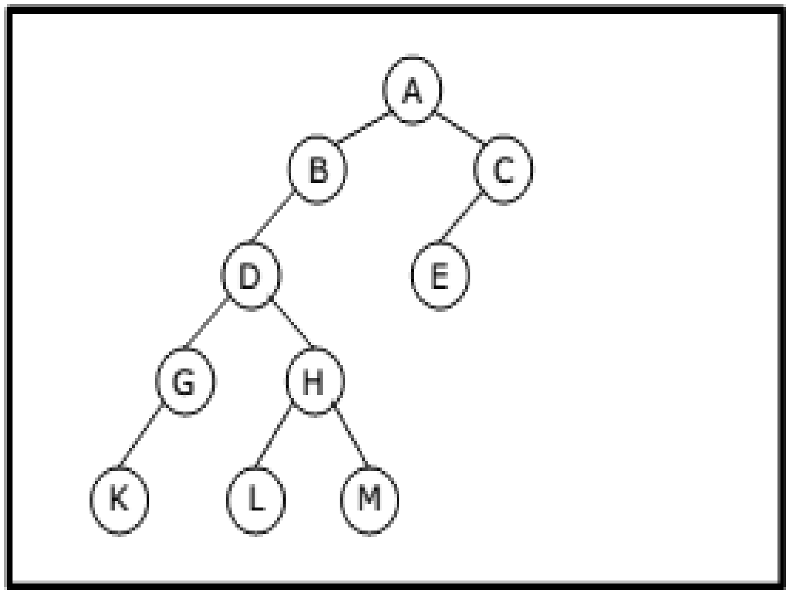
Construct the Binary Tree corresponding to the following traversals

- Preorder traversal yields:
  A, B, D, G, K, H, L, M, C, E
- Postorder travarsal yields:
  K, G, L, M, H, D, B, E, C, A
- Inorder travarsal yields:
  K, G, D, L, H, M, B, A, E, C

Pre, Post and Inorder Traversing

Binary Tree Construction

Construct the Binary Tree corresponding to the following traversals



Binary Tree

- Preorder traversal yields:
  A, B, D, G, K, H, L, M, C, E
- Postorder travarsal yields:
  K, G, L, M, H, D, B, E, C, A
- Inorder travarsal yields:
  K, G, D, L, H, M, B, A, E, C

Pre, Post and Inorder Traversing

Binary Search Tree

Binary Search Tree

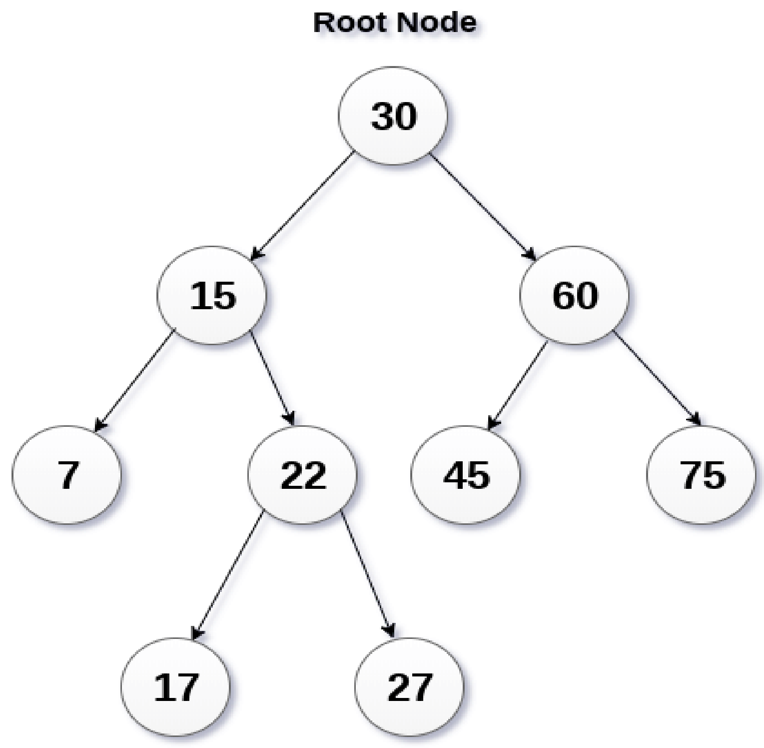
- A Binary tree which is either empty or satisfies the following rules-

1) **The value of the key in the left child or left subtree is less than the value of the root.**

2) **The value of the key in the right child or right subtree is more than the value of the root**

3) **All the subtrees of the left and right children observe the two rules.**

4) **There must be no duplicate nodes**



Binary Search Tree

Binary Search Tree

- **Why? Such Ordering?**

Binary Search Tree

- **The above properties of Binary Search Tree provide an ordering among keys so that the operations like**
  - **search,**
  - **minimum and**
  - **maximum**
- **can be done fast.**
- **If there is no ordering, then we may have to compare every key to search a given key.**

Create the binary search tree using data elements.

Create the binary search tree using data elements.

43, 10, 79, 90, 12, 54, 11, 9, 50

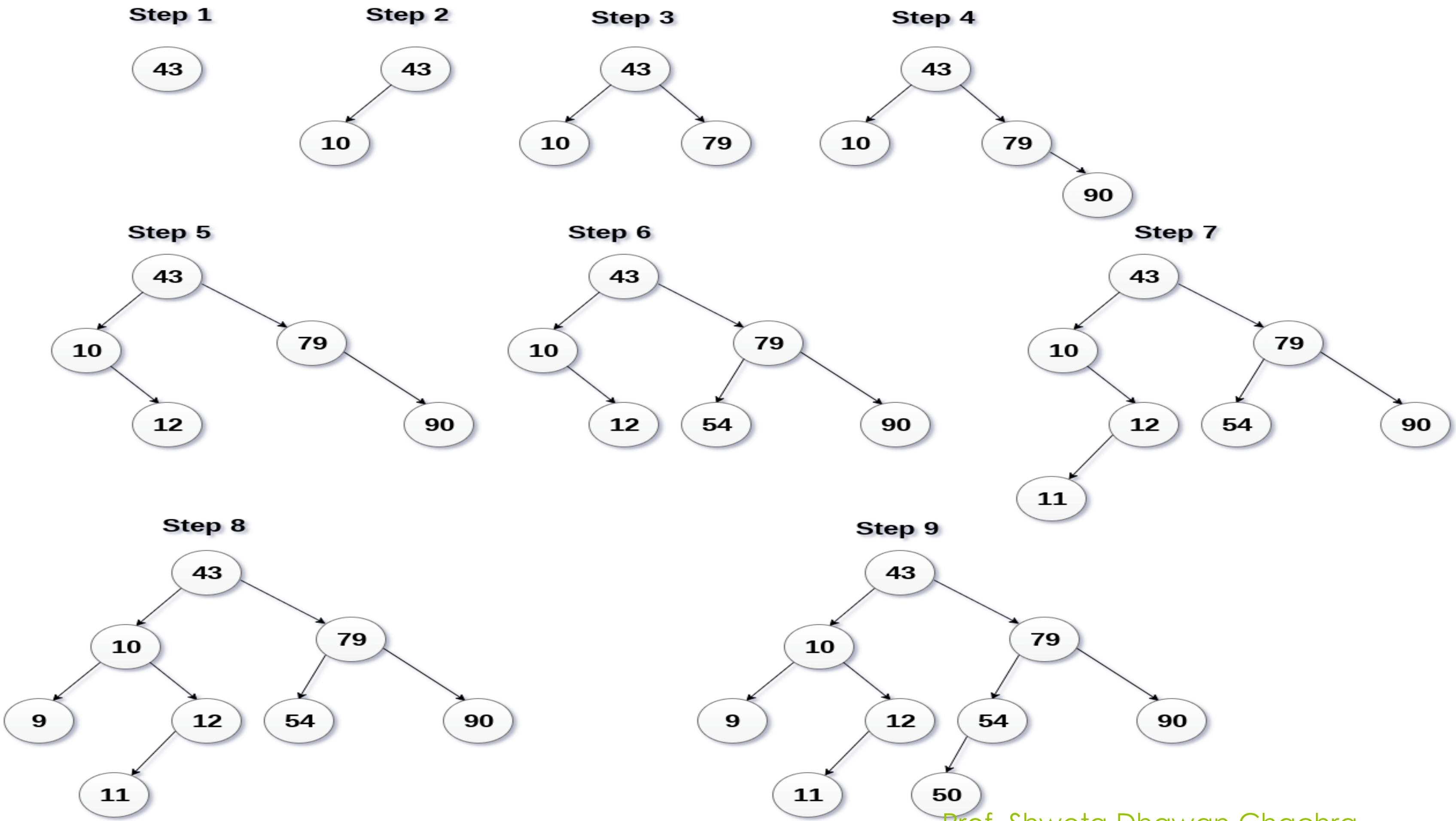
- Insert 43 into the tree as the root of the tree.
- Read the next element, if it is lesser than the root node element, insert it as the root of the left sub-tree.
- Otherwise, insert it as the root of the right of the right sub-tree.
- Continue……

Create the binary search tree using data elements.

43, 10, 79, 90, 12, 54, 11, 9, 50

Create the binary search tree using data elements.

43, 10, 79, 90, 12, 54, 11, 9, 50



Create the binary search tree using data elements.

- https://austingwalters.com/wp-content/uploads/2014/10/binary-tree-1-creation.gif

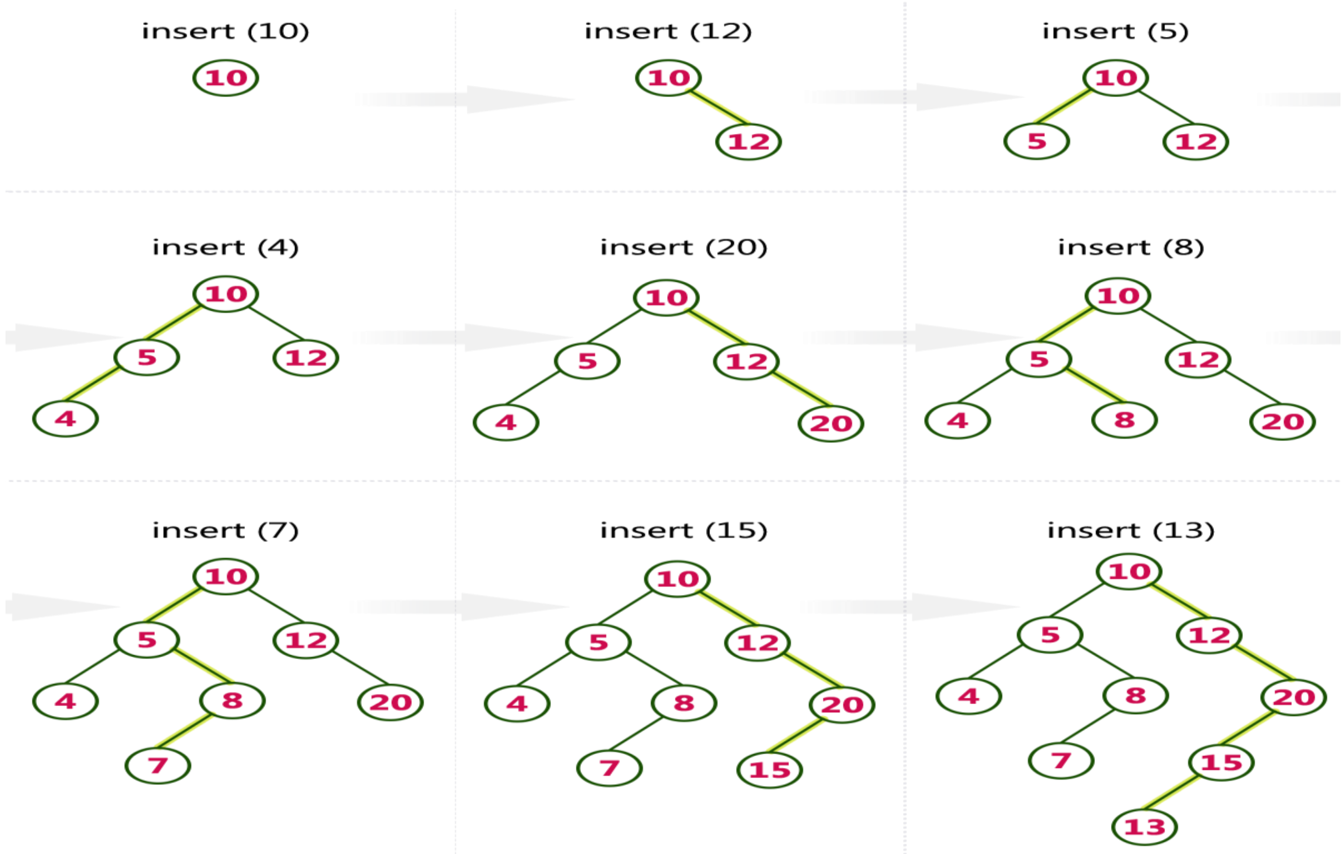
Create the binary search tree using data elements.

10,12,5,4,20,8,7,15 and 13

??

Create the binary search tree using data elements.

10,12,5,4,20,8,7,15 and 13



Searching and Insertion Operation in Binary Search Tree

- Before Insertion of any element , we first search the exact place for inserting

Searching and Insertion Operation in Binary Search Tree

- The steps are-

1) **Compare the data with the root node**

- **If the data<data of node compare with data of left child**
- **If the data>data of node compare with data of right child**

1) **At last , we will reach the exact place where we will insert the data.**

Q)Suppose the numbers 7, 5, 1, 8, 3, 6, 0, 9, 4, 2 are inserted in that order into an initially empty binary search tree. The binary search tree uses the usual ordering on natural numbers. What is the in-order traversal sequence of the resultant tree?

(A) 7 5 1 0 3 2 4 6 8 9
(B) 0 2 4 3 1 6 5 9 8 7
(C) 0 1 2 3 4 5 6 7 8 9
(D) 9 8 6 4 2 3 0 1 5 7

Q)Suppose the numbers 7, 5, 1, 8, 3, 6, 0, 9, 4, 2 are inserted in that order into an initially empty binary search tree. The binary search tree uses the usual ordering on natural numbers. What is the in-order traversal sequence of the resultant tree?

(A) 7 5 1 0 3 2 4 6 8 9
(B) 0 2 4 3 1 6 5 9 8 7
(C) 0 1 2 3 4 5 6 7 8 9
(D) 9 8 6 4 2 3 0 1 5 7

Answer: (C)

Explanation: In-order traversal of a BST gives elements in increasing order. So answer c is correct without any doubt.

Traversal in Binary Search Tree

- Traversal in Binary Search Tree is same as Traversal in Binary tree
  - Preorder
  - Postorder
  - Inorder

Traversal in Binary Search Tree

Some Interesting Facts:

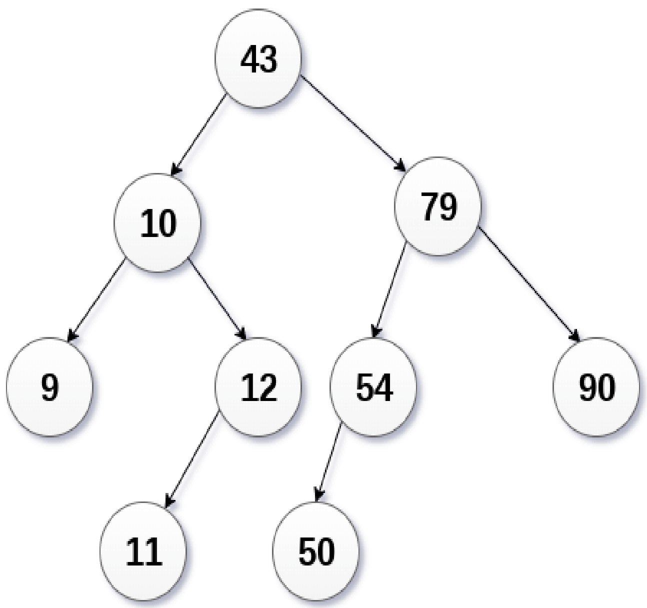
- **Inorder traversal of BST always produces sorted output.**
- We can construct a BST with only Preorder or Postorder or Level Order traversal.
- **Note that we can always get inorder traversal by sorting the only given traversal.**

Traversal in Binary Search Tree

Lets check

Inorder Traversal-
9,10,11,12,43,50,54,79,90

Yes, Sorted!!!!



Delete Operation in Binary Search Tree

- Complex than insertion and searching
- **There are 3 cases-**

Case 1- Node is a leaf node
Case 2- Node has exactly one child node
Case 3- Node has exactly two child nodes

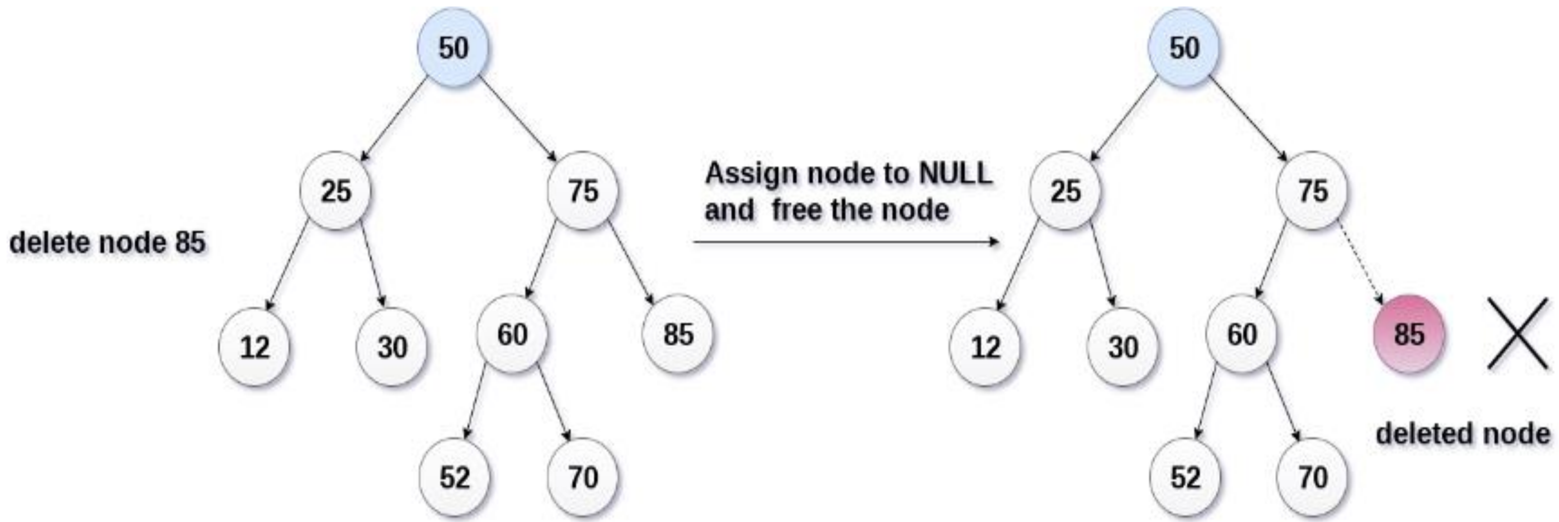
Delete Operation in Binary Search Tree

Case 1- Node is a leaf node

- If a node is a leaf node, it has no children
- Simply delete it
  - by giving NULL value to its Parent's
    - right pointer or
    - left pointer

Delete Operation in Binary Search Tree

Case 1- Node is a leaf node



Delete Operation in Binary Search Tree

Case 2- Node has exactly one child node

- Copy the child to the node and delete the child

```
            50                            50
         /      \        delete(30)      /   \
        30       70      --------->    40     70
          \     /  \                         /  \
          40   60   80                      60   80
```

Delete Operation in Binary Search Tree

Case 3- Node has exactly two child nodes

1) Find the inorder successor of the item
2) Copy contents of the inorder successor to the node
3) Delete the inorder successor.

- Note that inorder predecessor can also be used.

Delete Operation in Binary Search Tree

- Find the inorder successor of the item
- Use the Inorder traversal

Delete Operation in Binary Search Tree

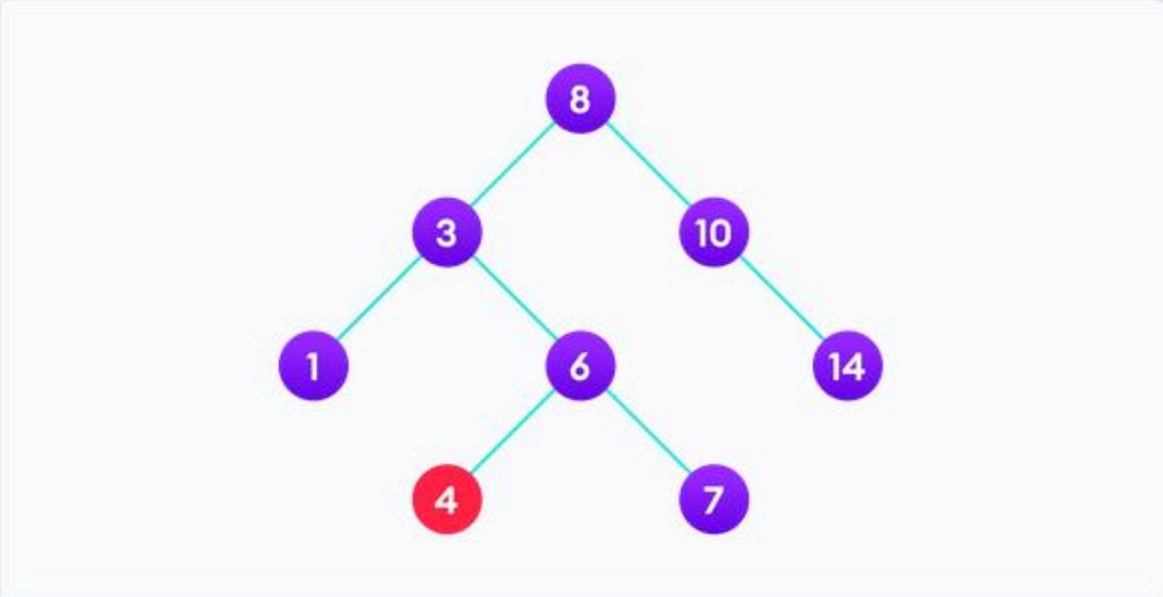
Case 3- Node has exactly two child nodes

```
      50                            60
   /     \         delete(50)      /   \
  40      70       --------->    40     70
         /  \                             \
        60   80                            80
```

Delete the following nodes from the BST

- 4
- 6

In sequence

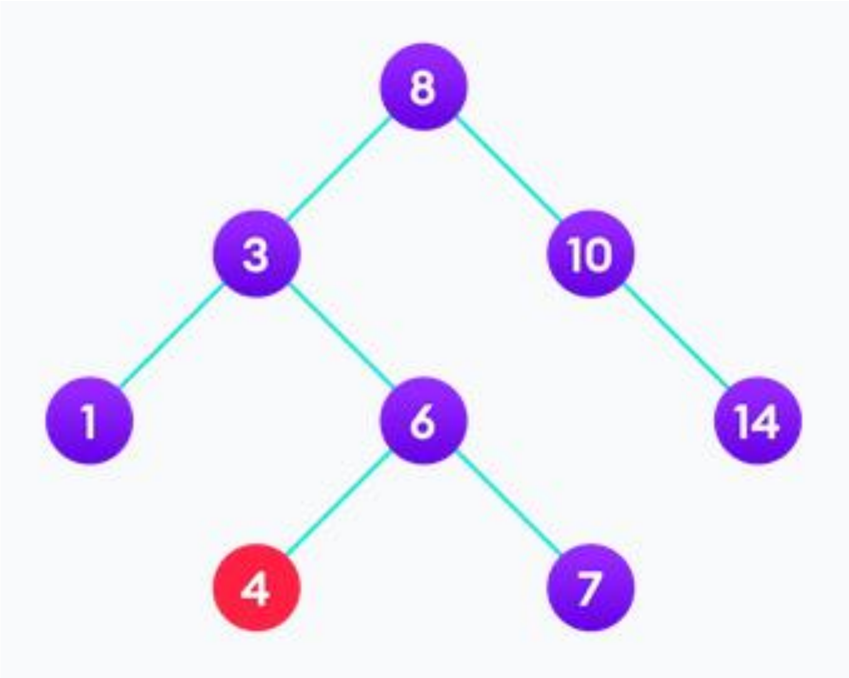


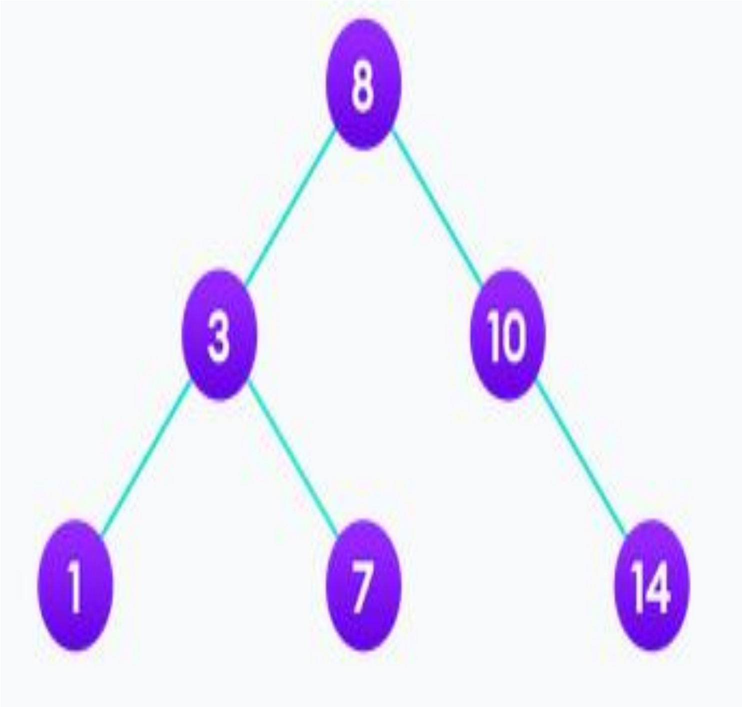
4 is to be deleted

Delete the following nodes from the BST

- 4
- 6

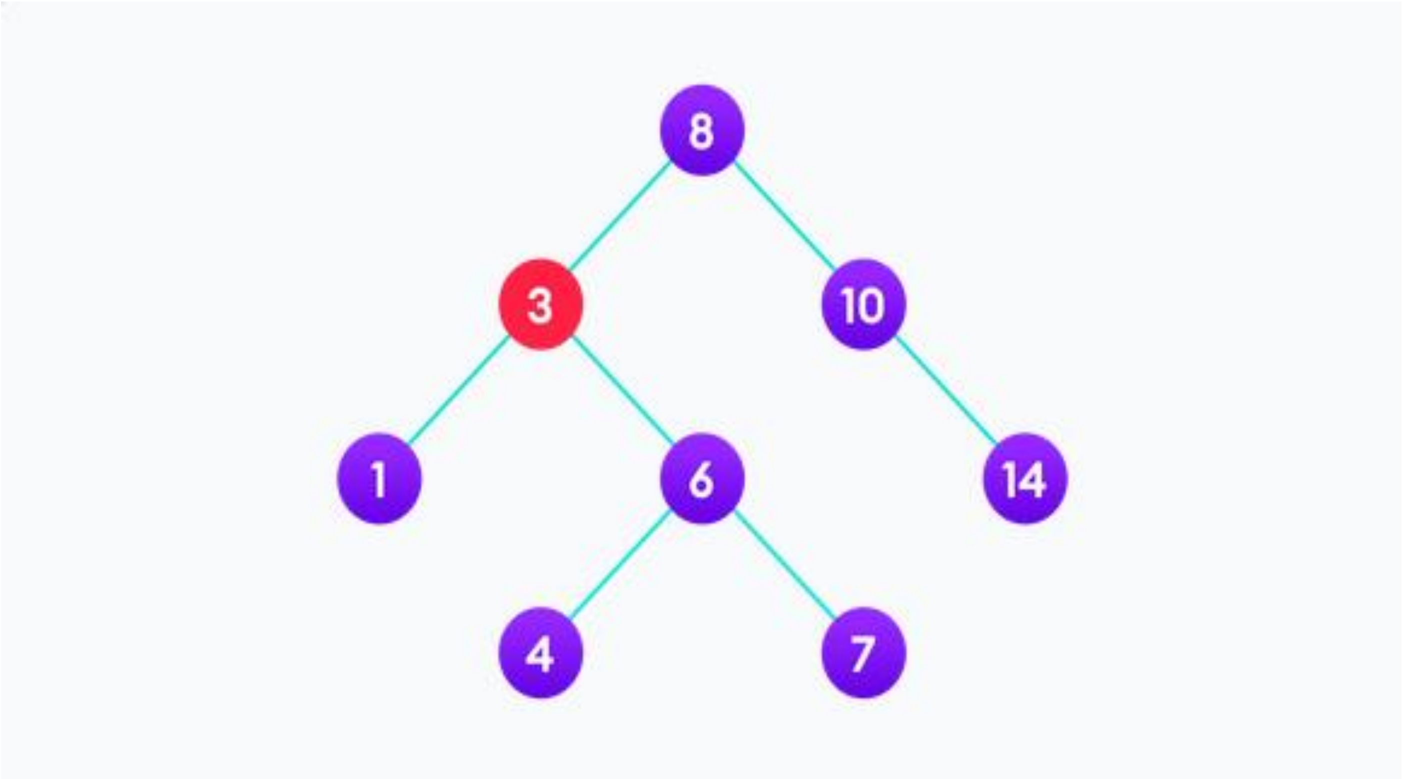
In sequence





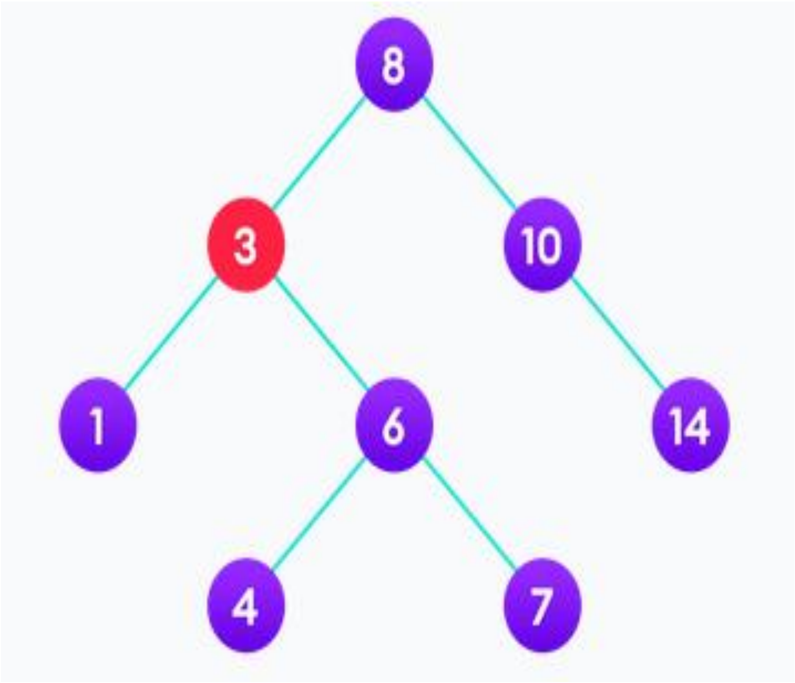
Delete the following nodes from the BST

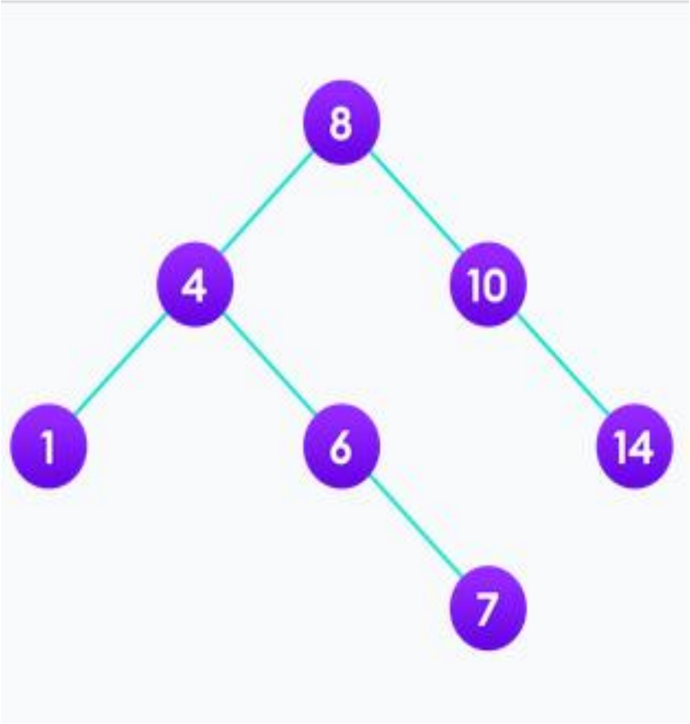
- 3



Delete the following nodes from the BST

- 3

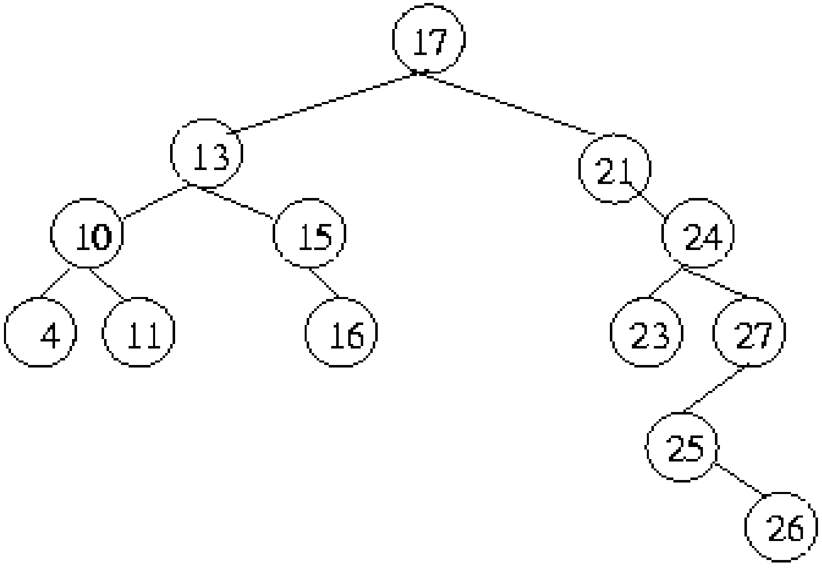




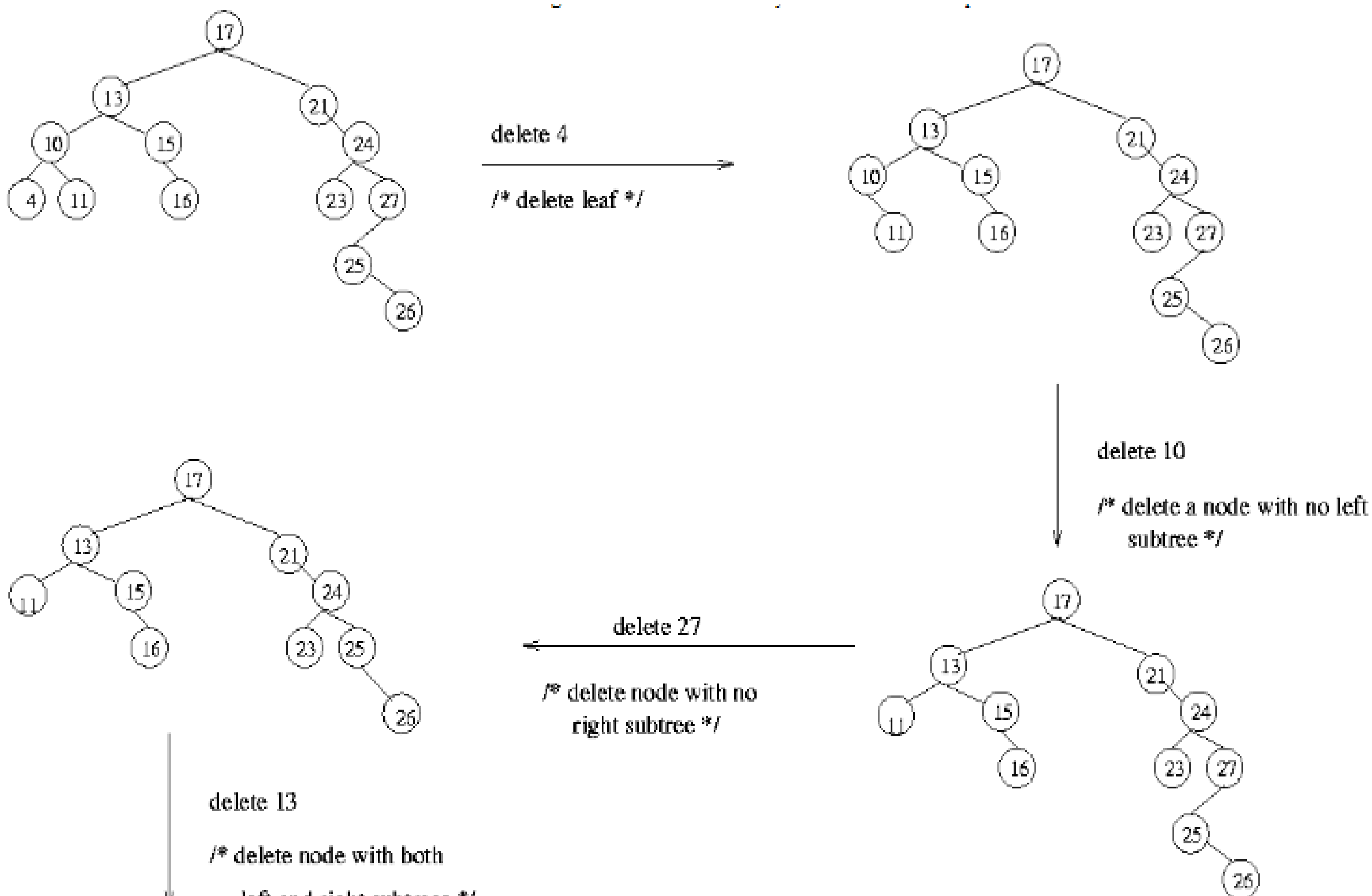
Delete Operation in Binary Search Tree

Consider the given BST and perform the following operations in sequential manner:

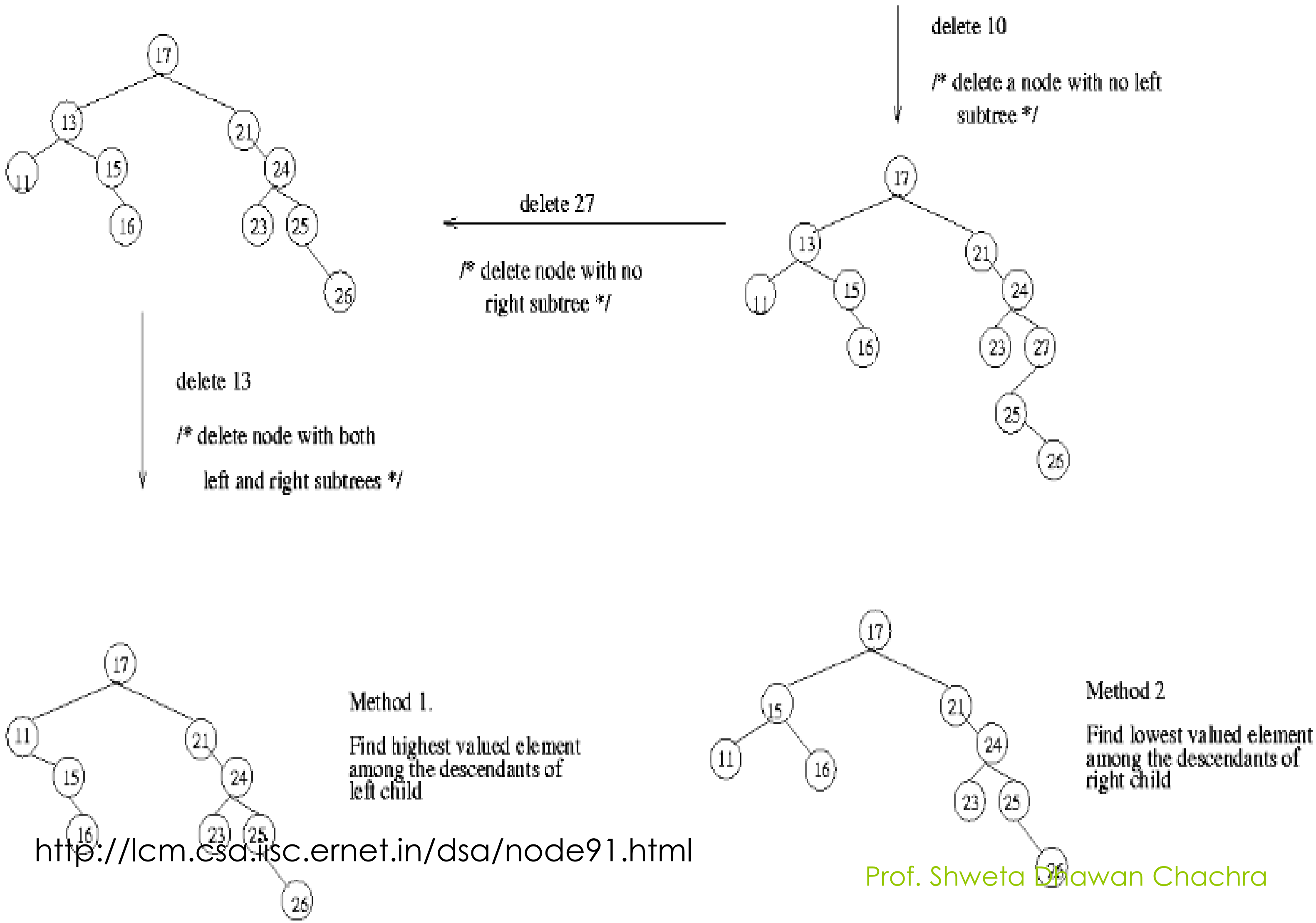
1) **Delete 4**
2) **Delete 10**
3) **Delete 27**
4) **Delete 13**



Delete Operation in Binary Search Tree



Delete Operation in Binary Search Tree



http://lcm.csa.iisc.ernet.in/dsa/node91.html

Q)The preorder traversal sequence of a binary search tree is 30, 20, 10, 15, 25, 23, 39, 35, 42. Which one of the following is the postorder traversal sequence of the same tree?

(A) 10, 20, 15, 23, 25, 35, 42, 39, 30
(B) 15, 10, 25, 23, 20, 42, 35, 39, 30
(C) 15, 20, 10, 23, 25, 42, 35, 39, 30
(D) 15, 10, 23, 25, 20, 35, 42, 39, 30

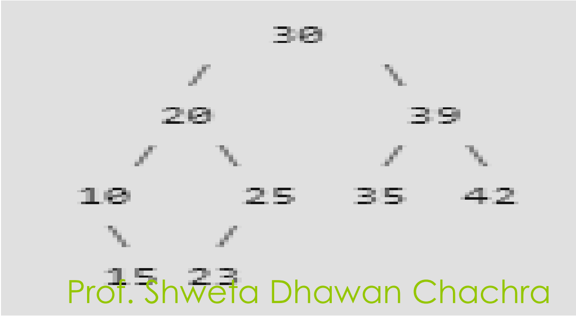
Q)The preorder traversal sequence of a binary search tree is 30, 20, 10, 15, 25, 23, 39, 35, 42. Which one of the following is the postorder traversal sequence of the same tree?

(A) 10, 20, 15, 23, 25, 35, 42, 39, 30
(B) 15, 10, 25, 23, 20, 42, 35, 39, 30
(C) 15, 20, 10, 23, 25, 42, 35, 39, 30
(D) 15, 10, 23, 25, 20, 35, 42, 39, 30

Answer: (D)
Explanation:

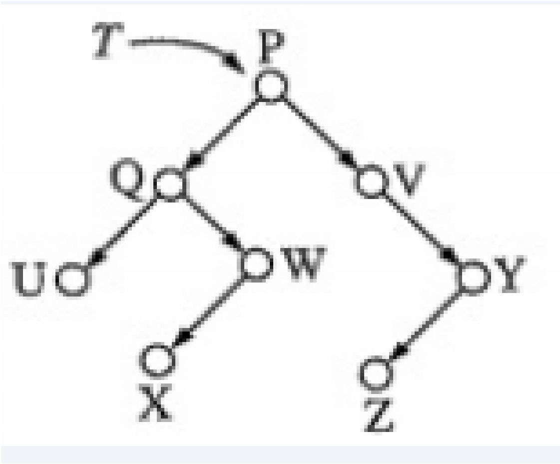
Sort the preorder to get the Inorder Traversal, then construct the tree

The following is the constructed tree



ISRO | ISRO CS 2014 | Question 41

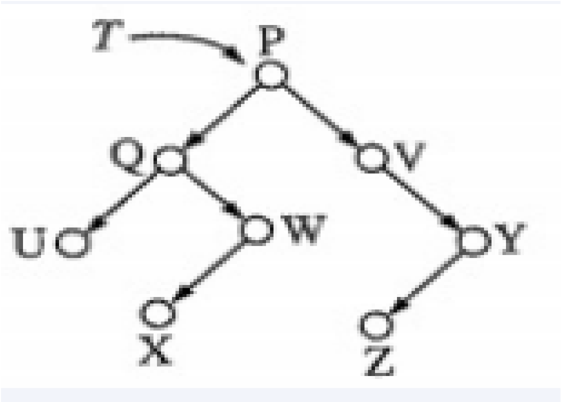
Consider the following binary search tree T given below:
Which node contains the fourth smallest element in T?



(A) Q
(B) V
(C) W
(D) X

ISRO | ISRO CS 2014 | Question 41

Consider the following binary search tree T given below:
Which node contains the fourth smallest element in T?



(A) Q
(B) V
(C) W
(D) X

Answer: (C)

Inorder Traversal=UQXWPVZY

Is in sorted order , so ans=W

ISRO | ISRO CS 2009 | Question 26

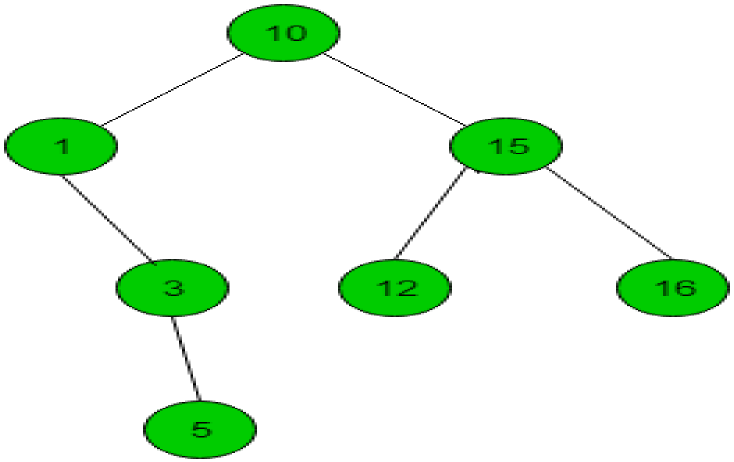
The following numbers are inserted into an empty binary search tree in the given order: 10, 1, 3, 5, 15, 12, 16. What is the height of the binary search tree (the height is the maximum distance of a leaf node from the root)?

(A) 2
(B) 3
(C) 4
(D) 6

ISRO | ISRO CS 2009 | Question 26

The following numbers are inserted into an empty binary search tree in the given order: 10, 1, 3, 5, 15, 12, 16. What is the height of the binary search tree (the height is the maximum distance of a leaf node from the root)?

(A) 2
(B) 3
(C) 4
(D) 6



Answer: (B)

Explanation:

So, height of the tree is 3, option (B) is correct.

Binary search Tree Implementation

```
struct node
{
        int num;
        struct node *left;
        struct node *right;
};
```

Insertion in Binary Search Tree

```
struct node *insert(struct node *tree,int digit)
{
        if(tree==NULL)
         {
                tree=(struct node *)malloc(sizeof(struct node));
                tree->left=tree->right=NULL;
                tree->num=digit;
         }
   else
         if(digit<tree->num)
                tree->left=insert(tree->left,digit);
   else
         if(digit>tree->num)
                tree->right=insert(tree->right,digit);
   else if(digit==tree->num)
          {
            printf("Duplicate node:program exited");
            exit(0);
          }
          return(tree);
}
```

Search in Binary Search Tree

```
void search(struct node *tree,int digit)
{
        if(tree==NULL)
          printf("The number does not exits\n");
  else
        if(digit==tree->num)
        printf("Number=%d\n" ,digit);
  else
        if(digit<tree->num)
          search(tree->left,digit);
  else
         search(tree->right,digit);
}
```