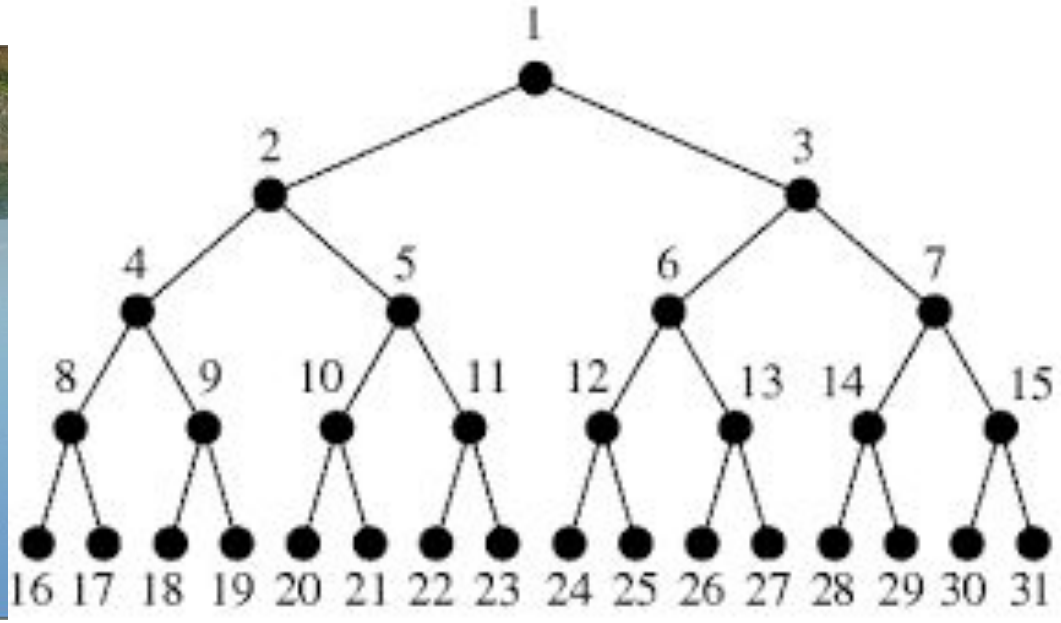
Trees

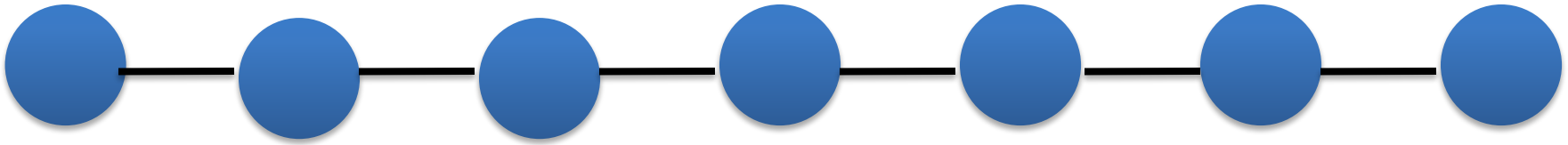
Hierarchical (or) non-linear data structure

Trees

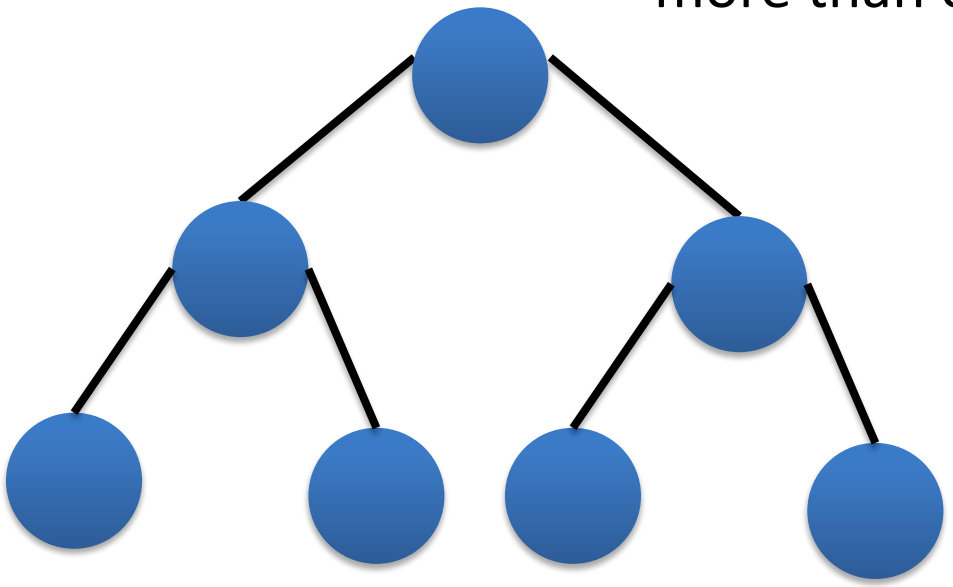


Tree is a collection of nodes connected by edges.

Linear vs Non linear

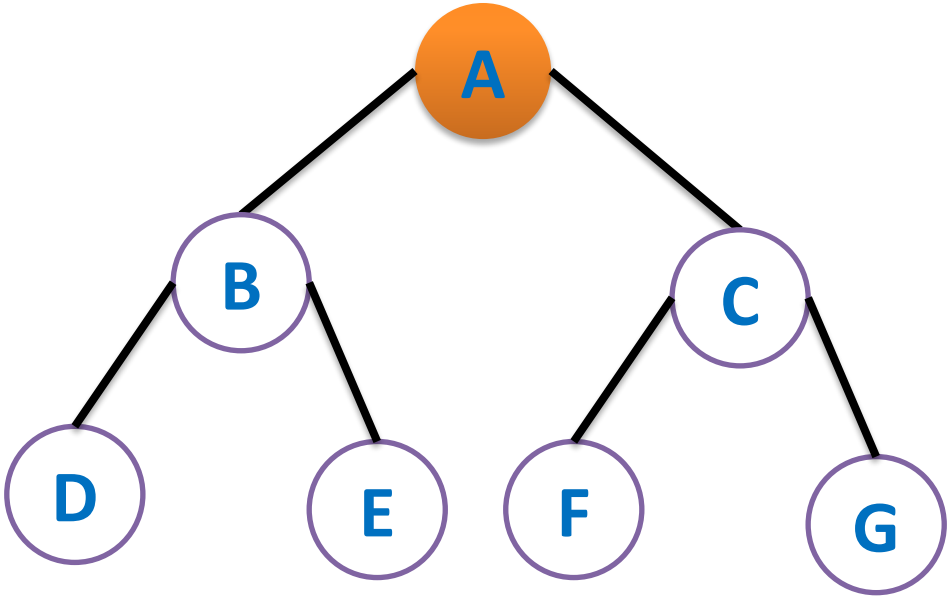


In non-linear data structure, a data item is connected to more than one data items.



Terminologies used in Trees

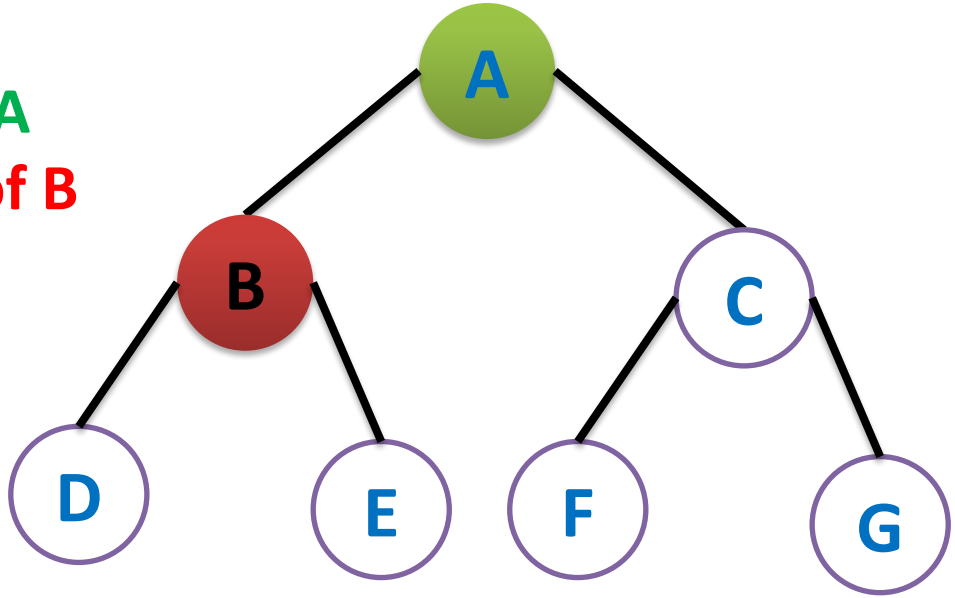
- **Root**
- **Child**
- **Parent**
- **Siblings**
- **Descendant**
- **Ancestor**
- **Leaf**
- **Degree**
- **Edge**
- **Height of node**
- **Height of tree**
- **Depth**



The top node in a tree.

Terminologies used in Trees

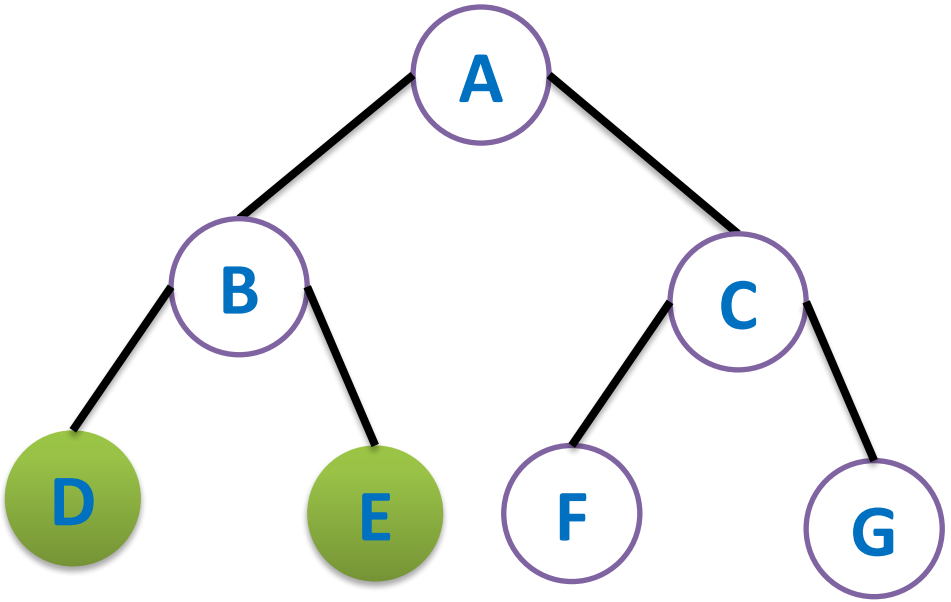
- Root
- Child B is a child of A
- Parent A is a parent of B
- Siblings
- Descendant
- Ancestor
- Leaf
- Degree
- Edge
- Height of node
- Height of tree
- Depth



A node directly connected to another node when moving away from the Root.

Terminologies used in Trees

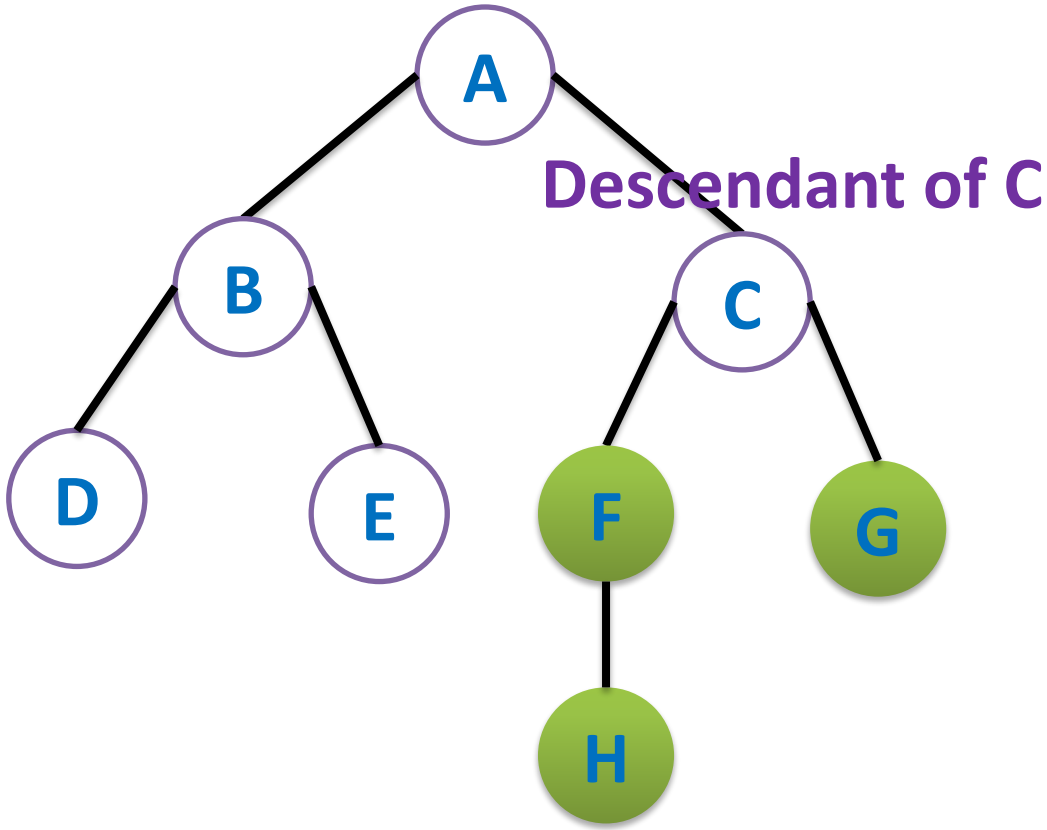
- Root
- Child
- Parent
- Siblings
- Descendant
- Ancestor
- Leaf
- Degree
- Edge
- Height of node
- Height of tree
- Depth



A group of nodes with the same parent.

Terminologies used in Trees

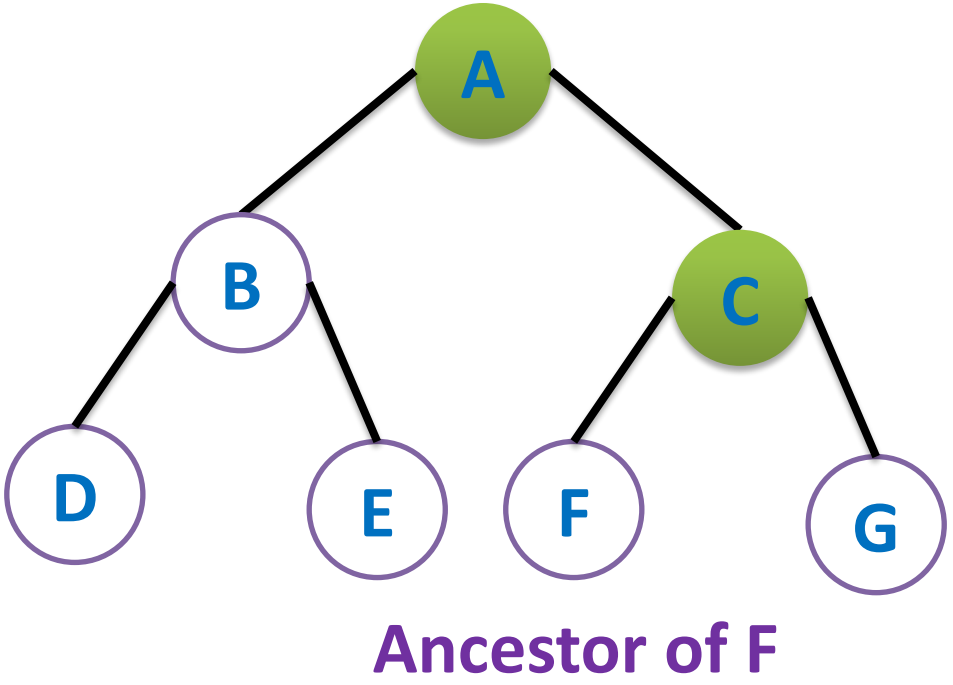
- **Root**
- **Child**
- **Parent**
- **Siblings**
- **Descendant**
- **Ancestor**
- **Leaf**
- **Degree**
- **Edge**
- **Height of node**
- **Height of tree**
- **Depth**



A node reachable by repeated proceeding from parent to child.

Terminologies used in Trees

- **Root**
- **Child**
- **Parent**
- **Siblings**
- **Descendant**
- **Ancestor**
- **Leaf**
- **Degree**
- **Edge**
- **Height of node**
- **Height of tree**
- **Depth**

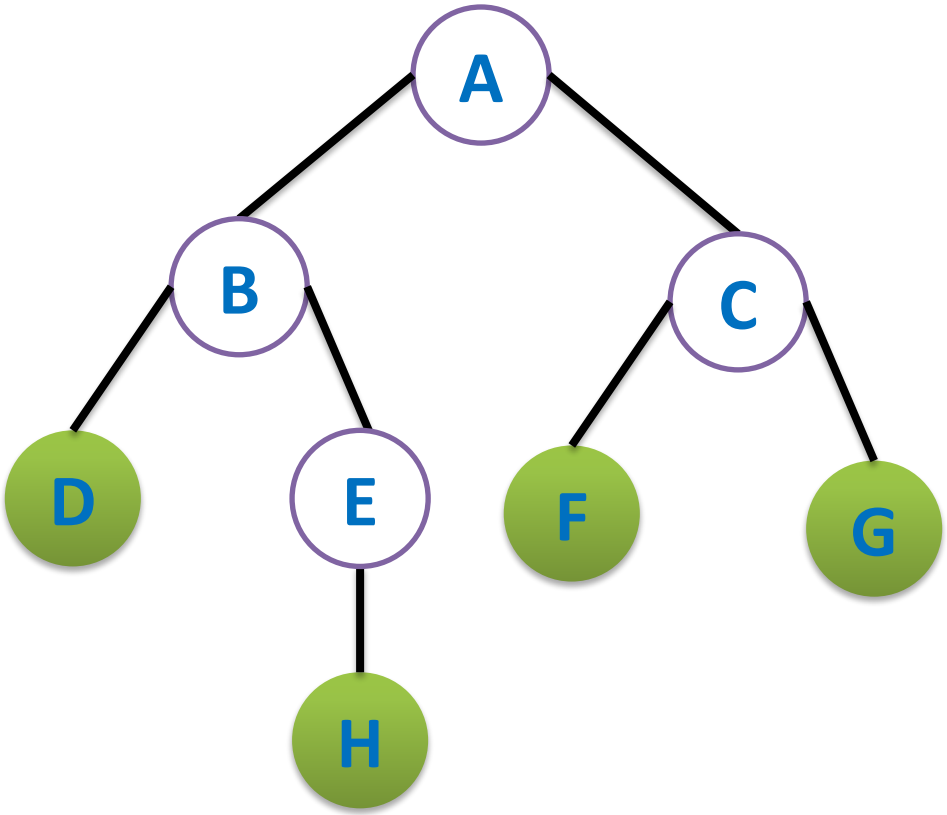


Ancestor of F

A node reachable by repeated proceeding from child to parent.

Terminologies used in Trees

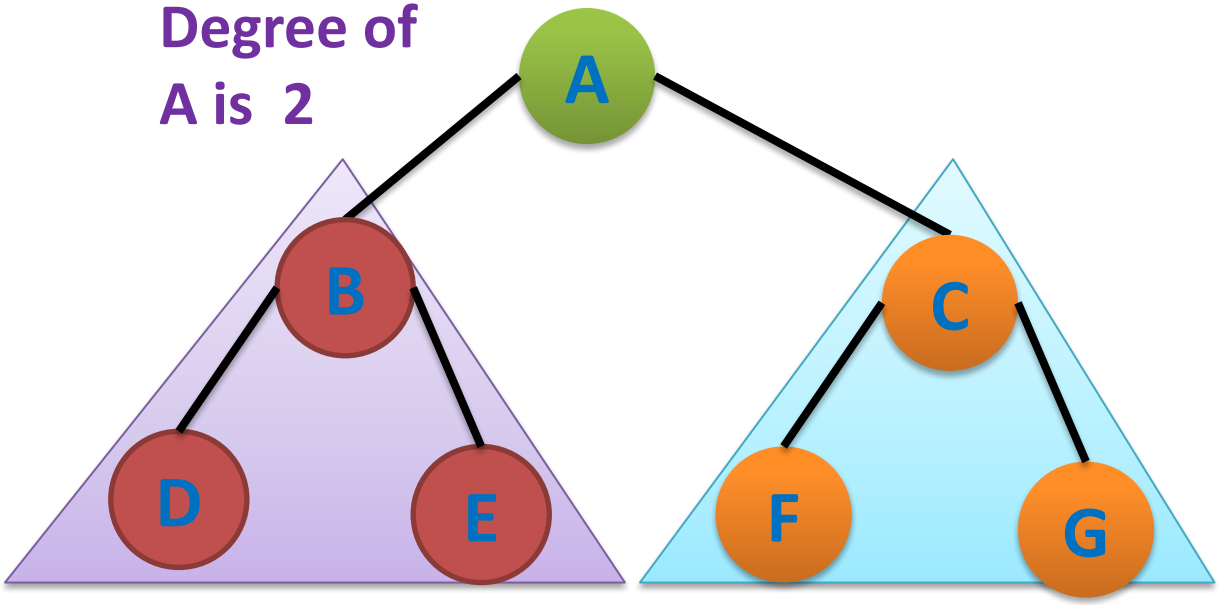
- **Root**
- **Child**
- **Parent**
- **Siblings**
- **Descendant**
- **Ancestor**
- **Leaf**
- **Degree**
- **Edge**
- **Height of node**
- **Height of tree**
- **Depth**



A node with no children.

Terminologies used in Trees

- Root
- Child
- Parent
- Siblings
- Descendant
- Ancestor
- Leaf
- Degree
- Edge
- Height of node
- Height of tree
- Depth

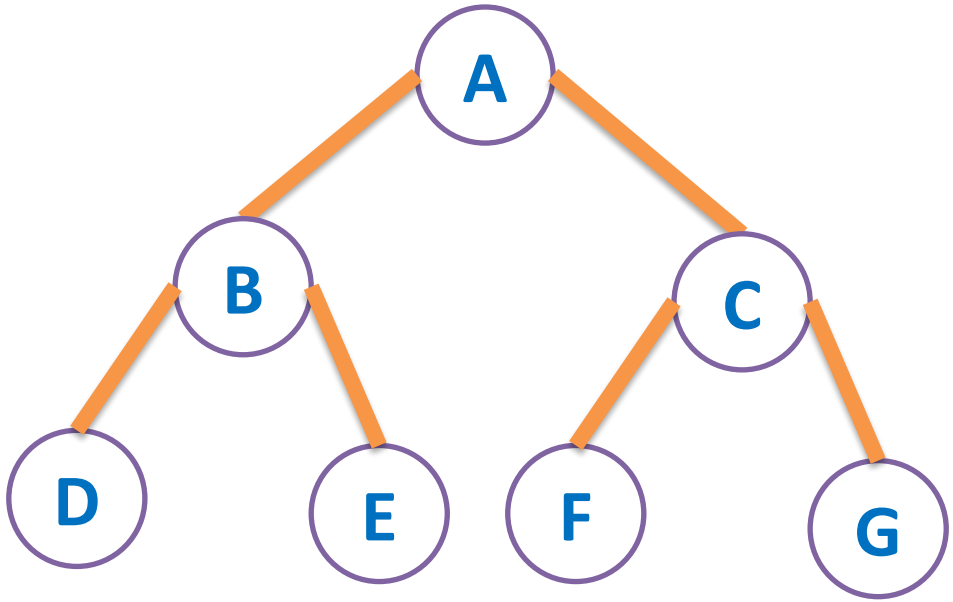


The number of sub trees of a node.

The **degree** of a **tree** is the
maximum degree of any of its nodes.

Terminologies used in Trees

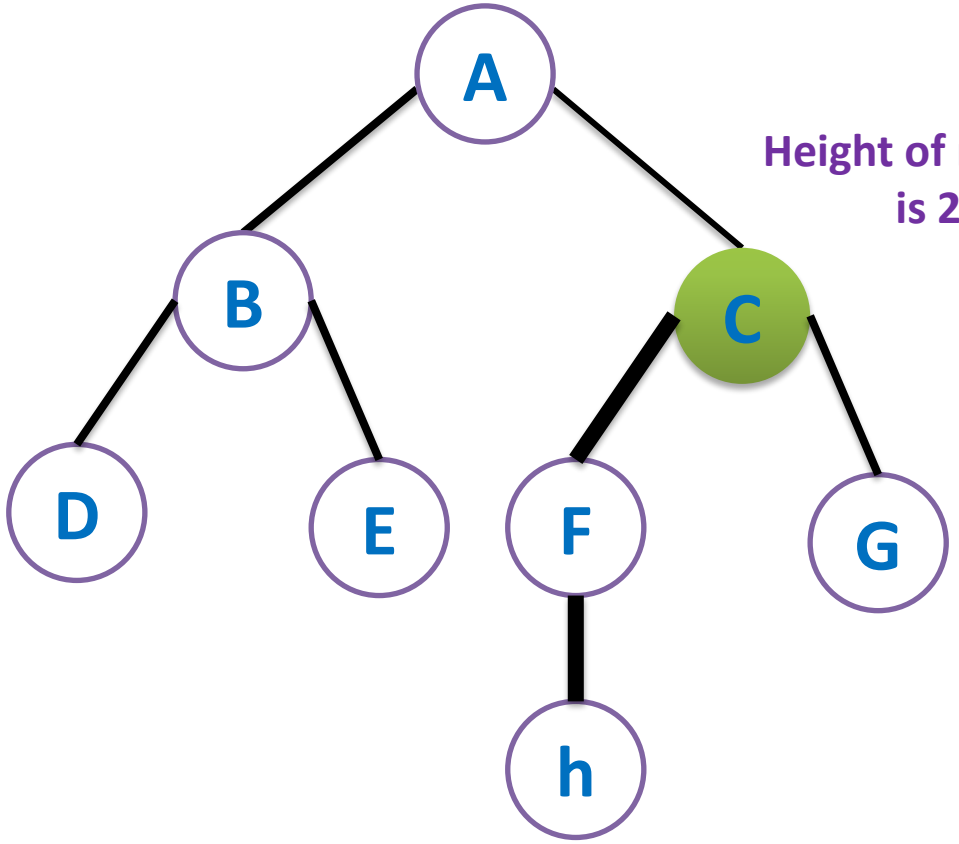
- **Root**
- **Child**
- **Parent**
- **Siblings**
- **Descendant**
- **Ancestor**
- **Leaf**
- **Degree**
- **Edge**
- **Height of node**
- **Height of tree**
- **Depth**



The connection between one node and another.

Terminologies used in Trees

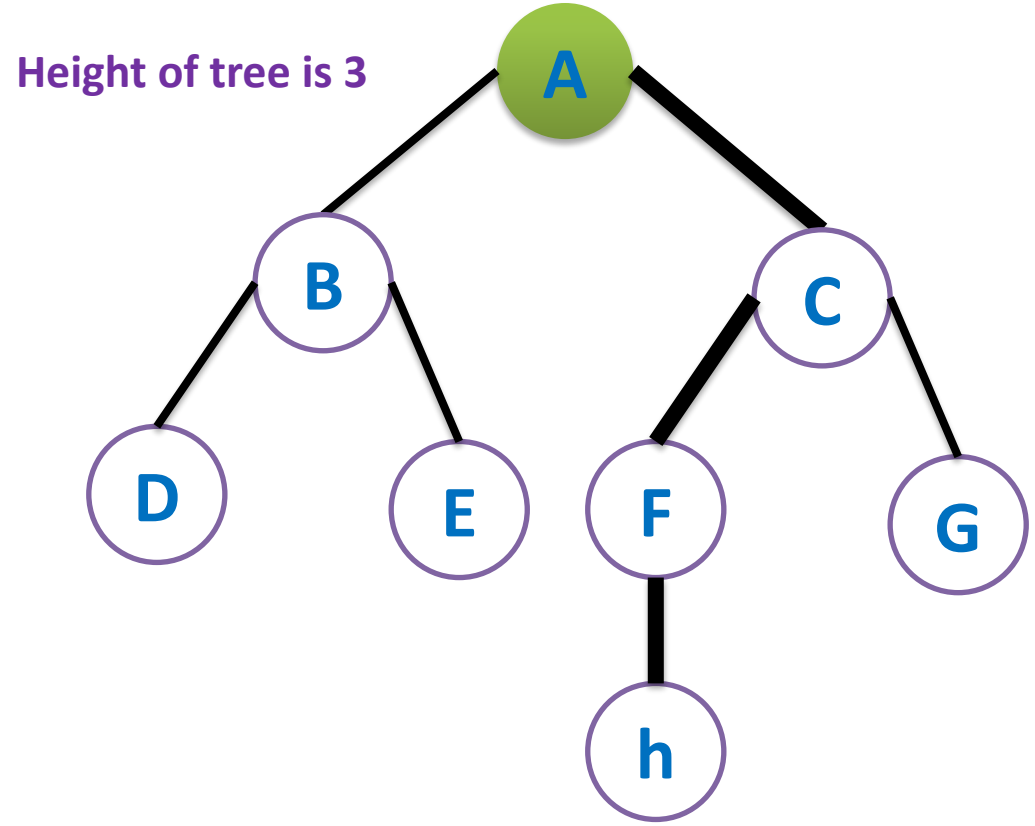
- **Root**
- **Child**
- **Parent**
- **Siblings**
- **Descendant**
- **Ancestor**
- **Leaf**
- **Degree**
- **Edge**
- **Height of node**
- **Height of tree**
- **Depth**



number of edges on the longest path between that node and a leaf.

Terminologies used in Trees

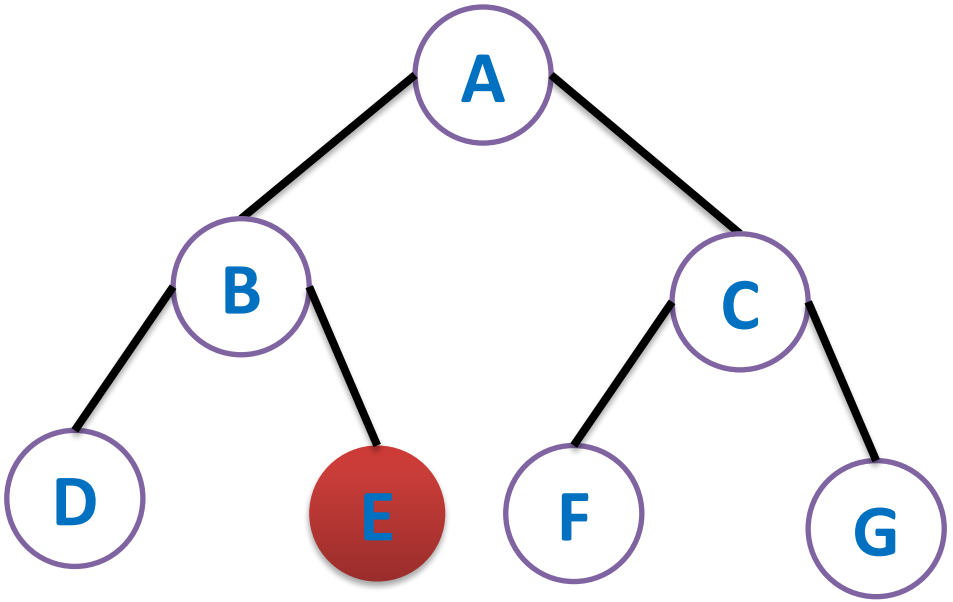
- **Root**
- **Child**
- **Parent**
- **Siblings**
- **Descendant**
- **Ancestor**
- **Leaf**
- **Degree**
- **Edge**
- **Height of node**
- **Height of tree**
- **Depth**



The height of a tree is the height of its root node.

Terminologies used in Trees

- **Root**
- **Child**
- **Parent**
- **Siblings**
- **Descendant**
- **Ancestor**
- **Leaf**
- **Degree**
- **Edge**
- **Height of node**
- **Height of tree**
- **Depth**

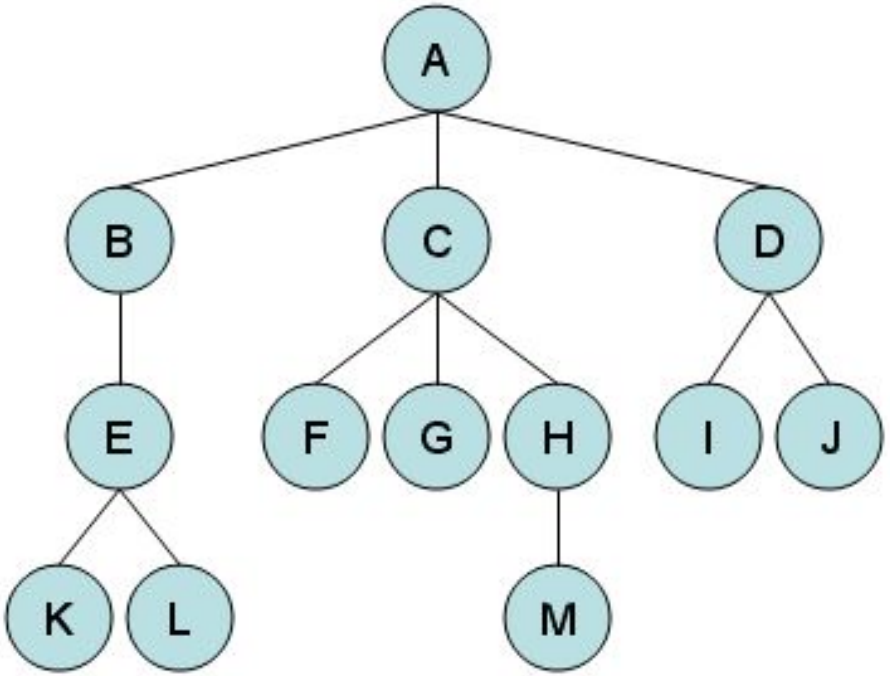


depth of a node E is 2

The depth of a node is the number of edges from the node to the tree's root node.

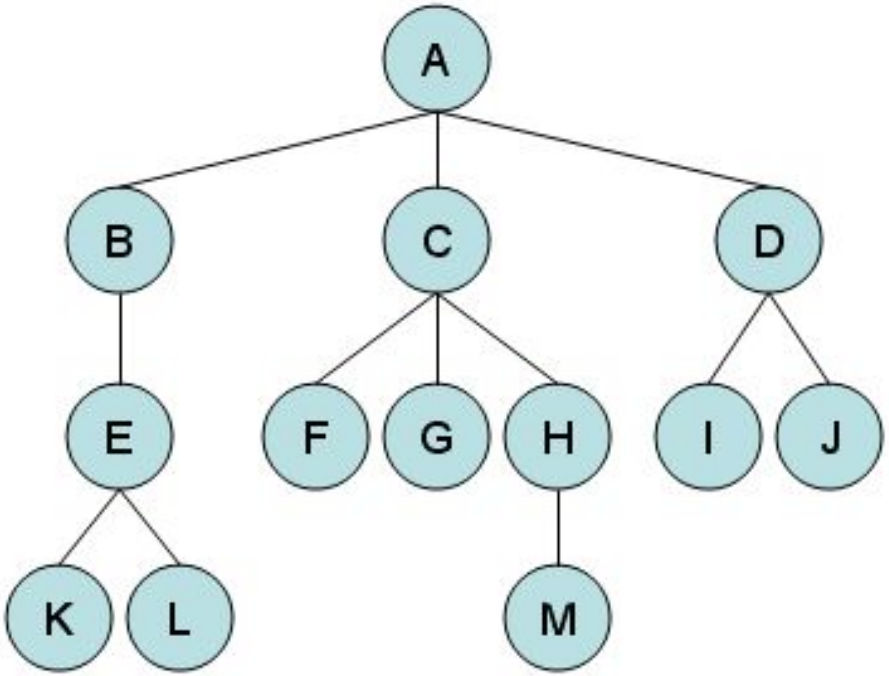
Find the following..

- **Root**
- **Leaf**
- **Degree**
- **Height of tree**



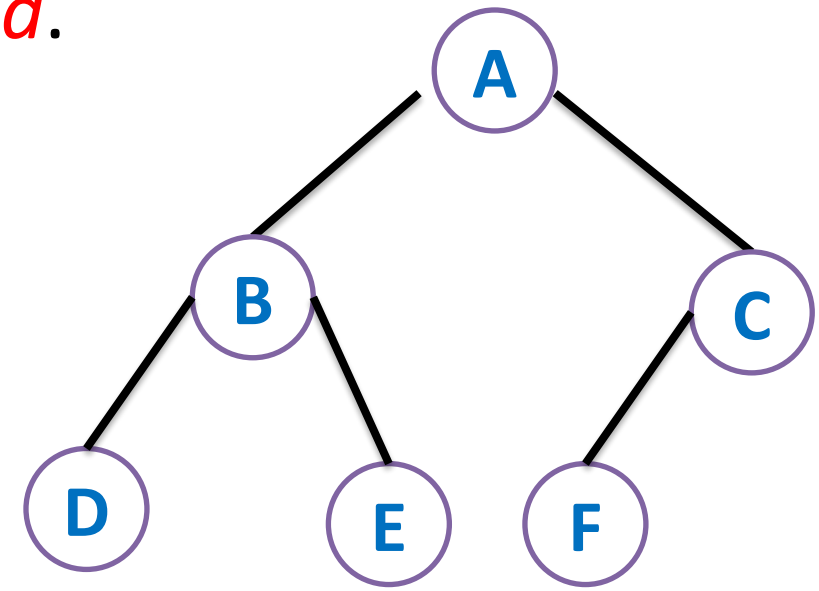
Answer

- **Root = A**
- **Leaf = K, L, F, G, M, I, J**
- **Degree = 3**
- **Height of tree= 3**



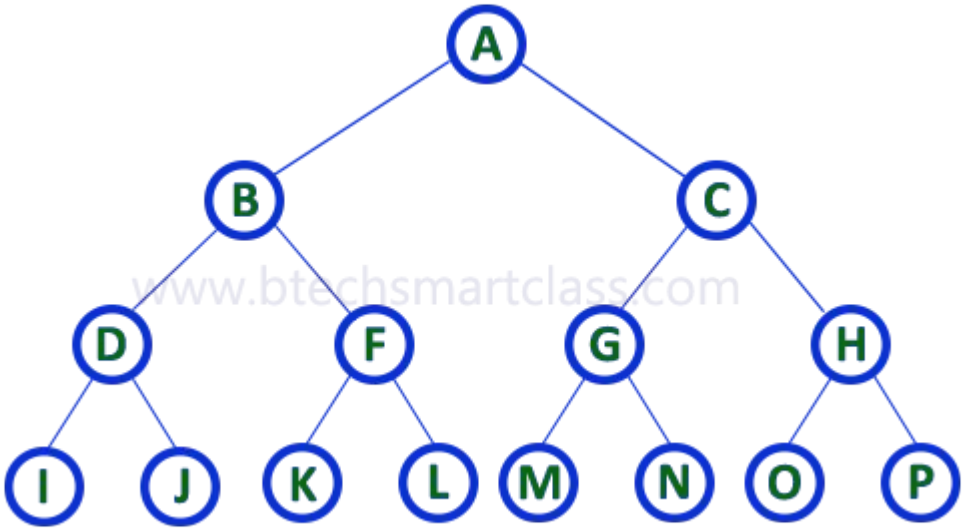
Binary tree

Binary tree is a tree data structure in which each node has **at most two children**, which are referred to as the *left child* and the *right child*.



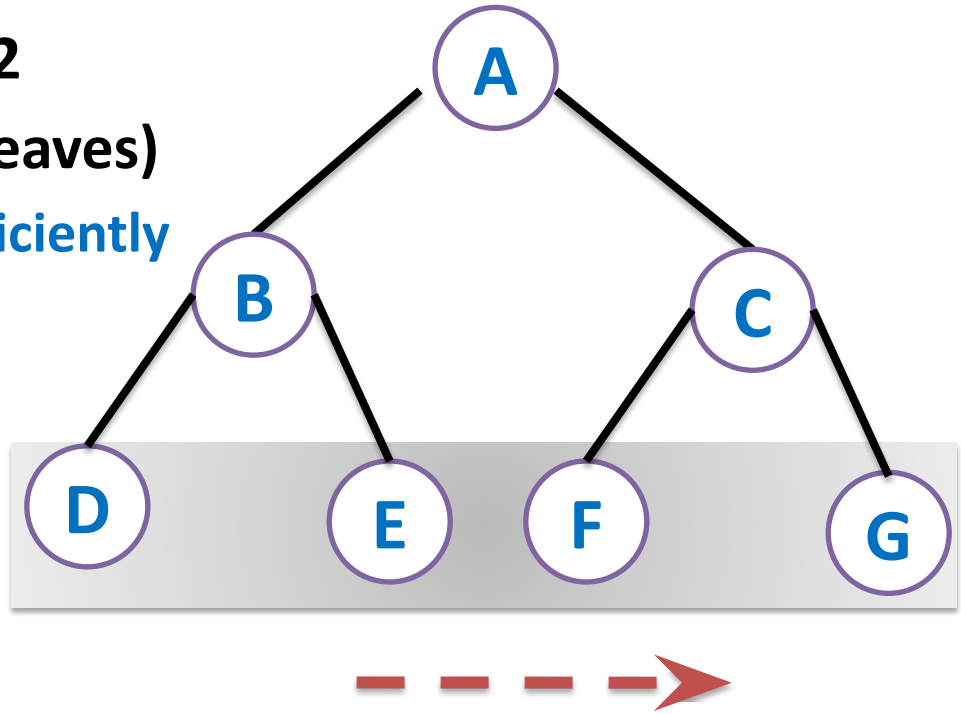
Perfect or Full binary tree

A **perfect** binary tree is a binary tree in which all interior nodes have two children *and* all leaves have the same ***depth*** **or same** ***level*****.**



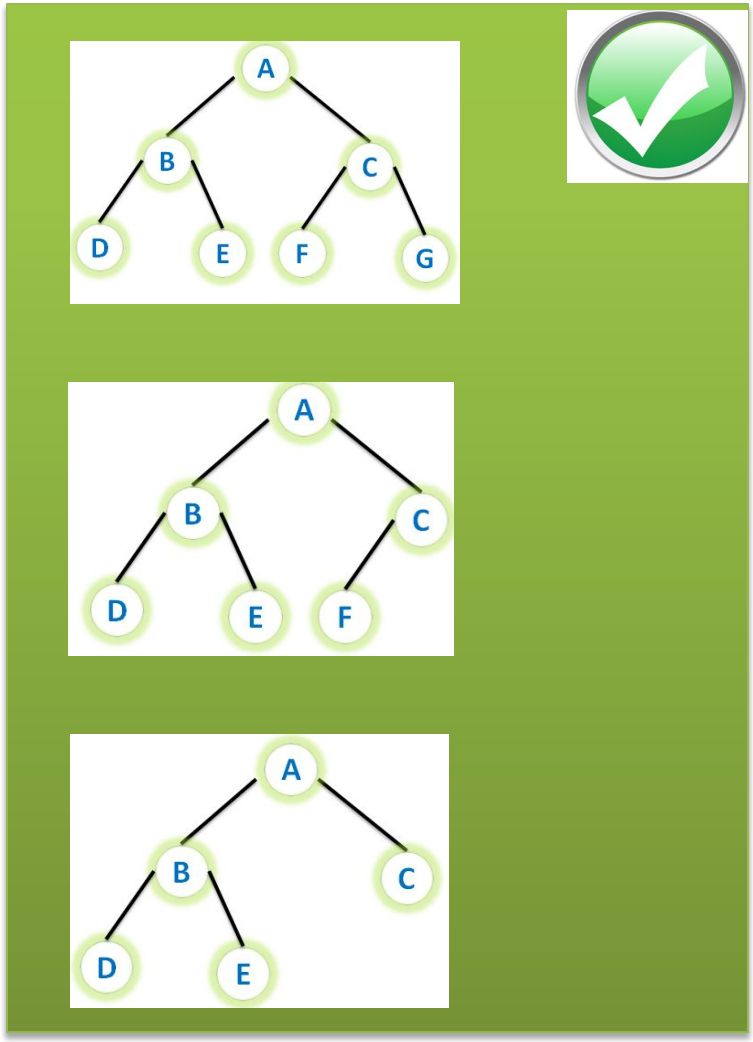
Full Binary Tree

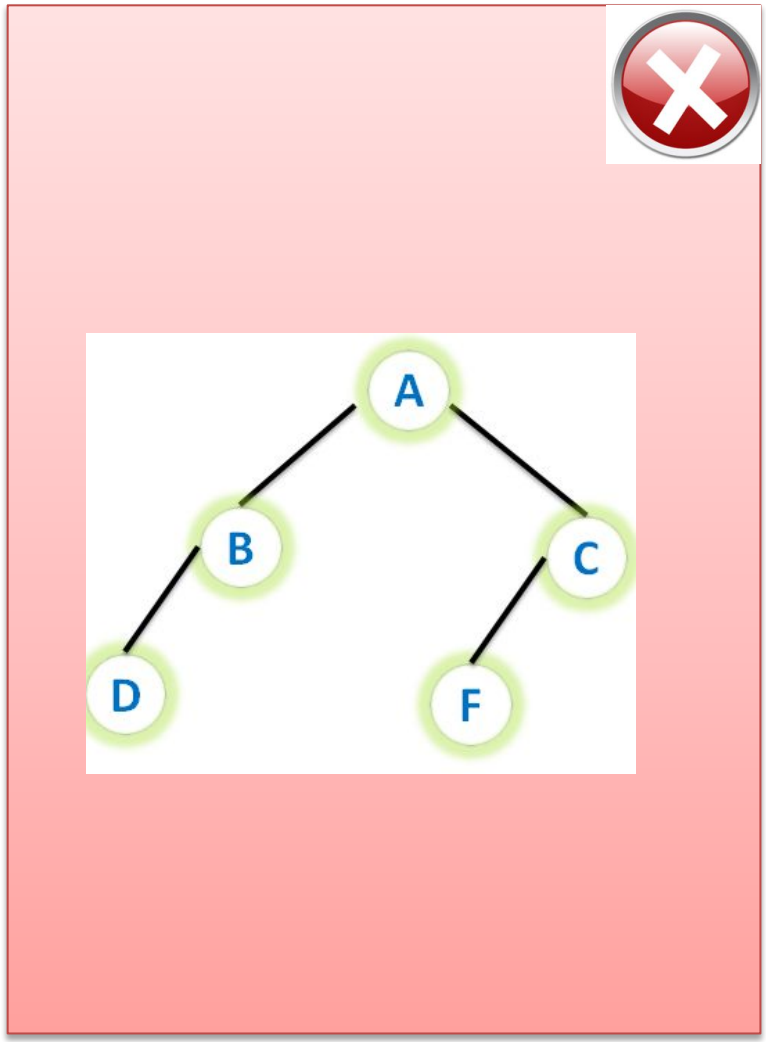
- **In a tree of height h**
  - All the leaves are at level **h**
  - Number of leaves 1 to $\mathbf{2^h}$
  - Number of internal nodes = $\mathbf{2^h - 1}$
- **In a full binary tree of n nodes**
  - Number of leaves is **(n+1) / 2**
  - Height = $\mathbf{\log_2}$ **(number of leaves)**
- **A complete binary tree can be efficiently represented using an array.**



Complete Binary Tree

Completely filled, with the possible exception of the bottom level, which is filled from left to right.



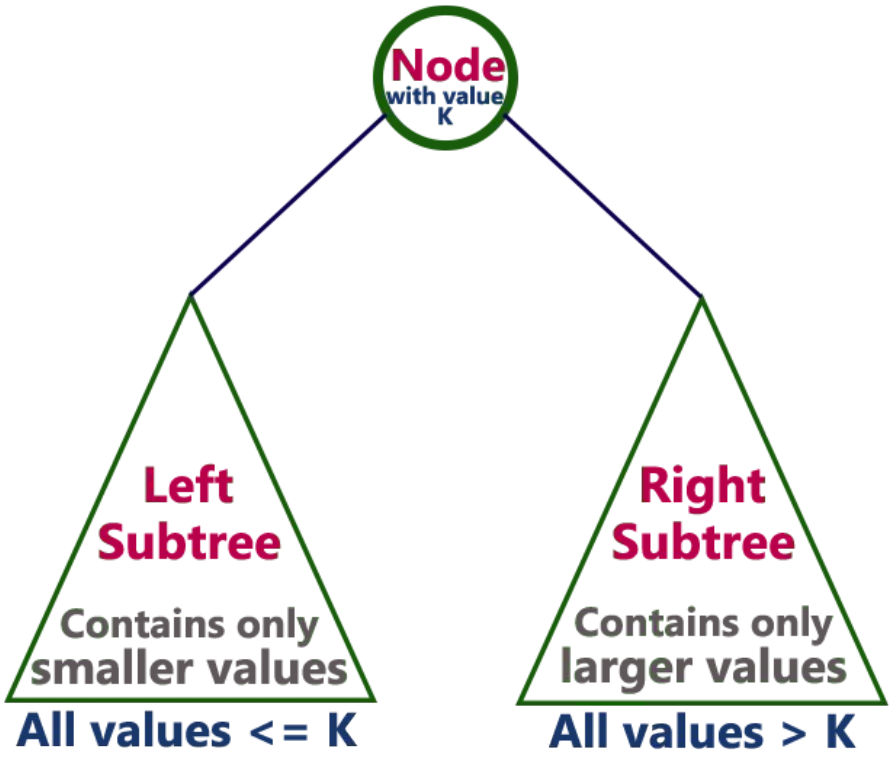


A complete binary tree can be efficiently represented using an array.

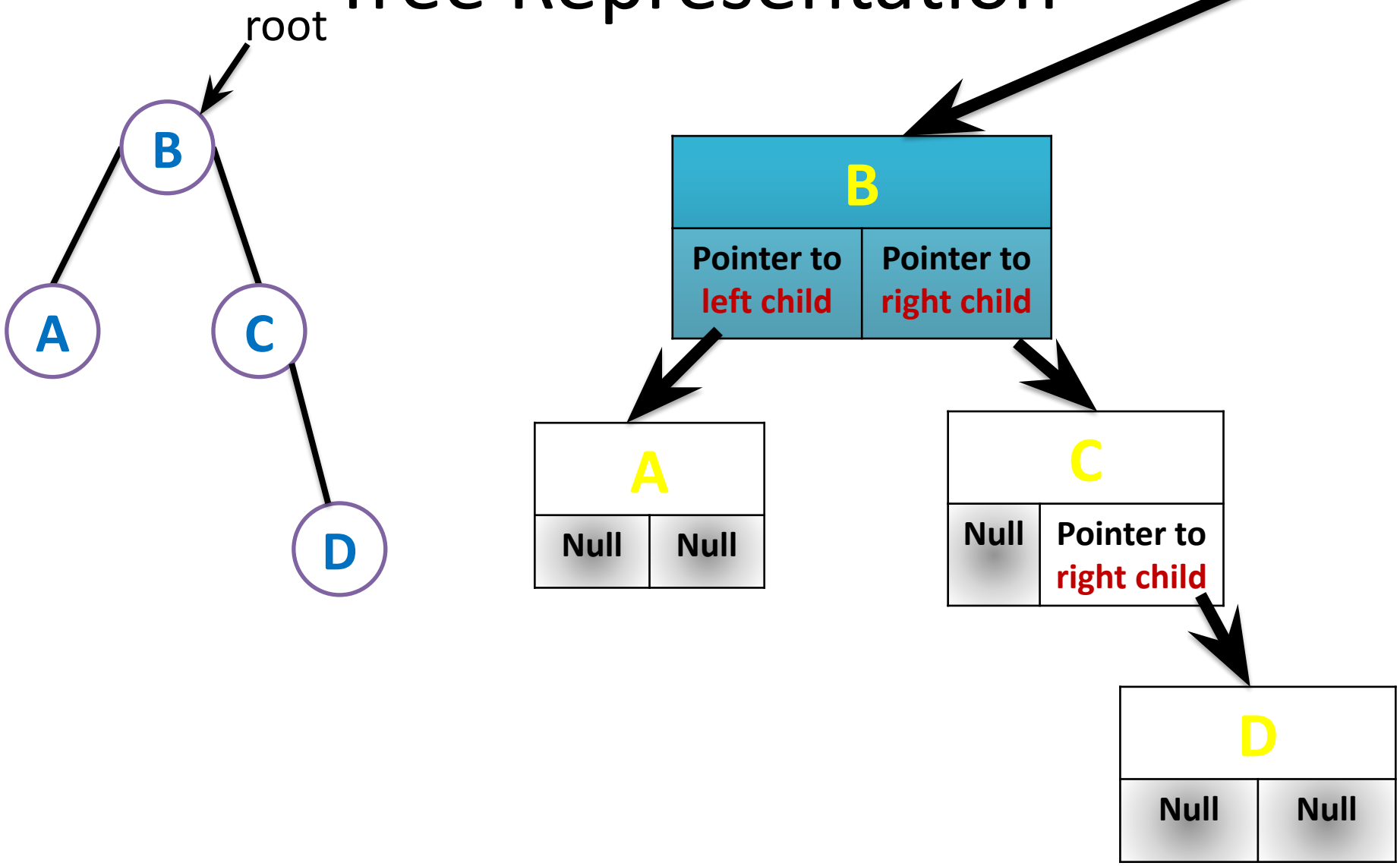
Binary search trees (BST)

ordered or **sorted binary trees**,

- Binary Search Tree is a binary tree in which every node contains only **smaller values in its left** subtree and only **larger values in its right** subtree**.**

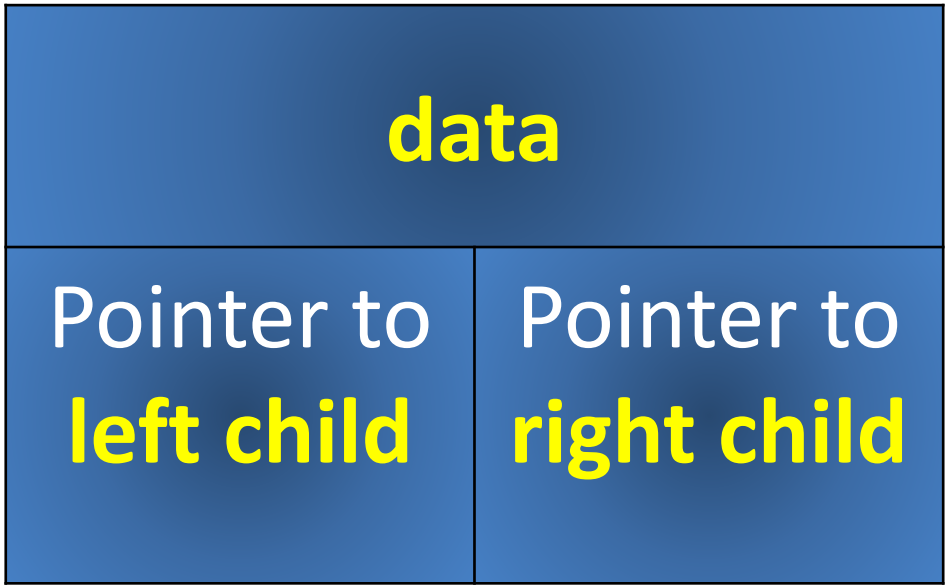


Tree Representation



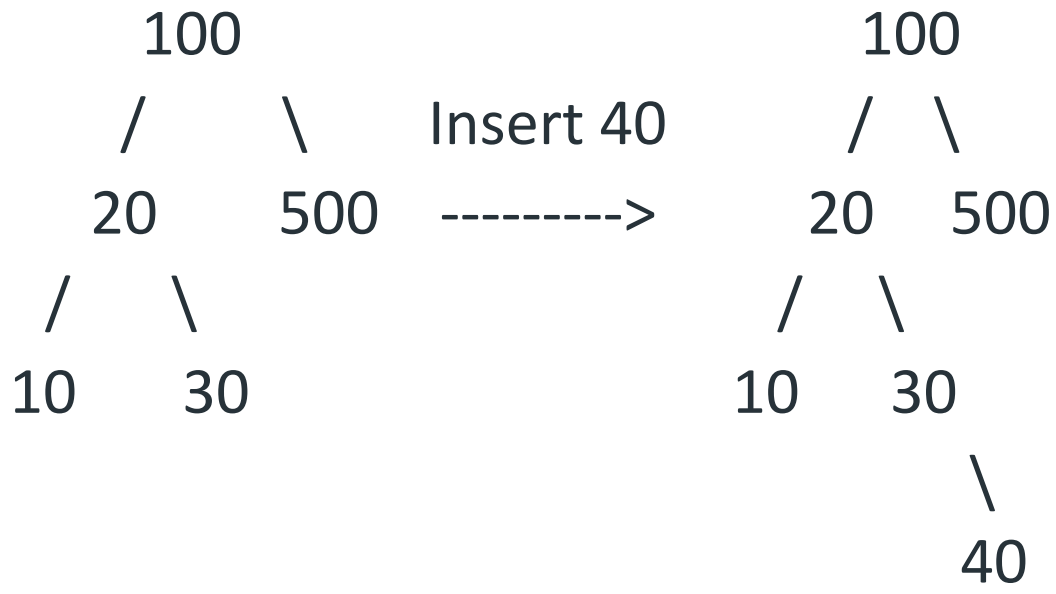
Node Structure

```
struct   Treenode{
    int    data;
    Treenode   *left ;
    Treenode   *right ;
}
```



BST Insertion

- A new key is always inserted at the leaf.
- Start searching a key from the root until a leaf node is hit; The new node is added as a child of the leaf node.



BST Insertion

```
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
```

Tree Traversal

- Process of **checking** and/or **updating** each node in the tree data structure, **exactly once**.
- Such **traversals** are classified by the order in which the nodes are visited.
  - **Inorder**
  - **Preorder**
  - **Postorder**

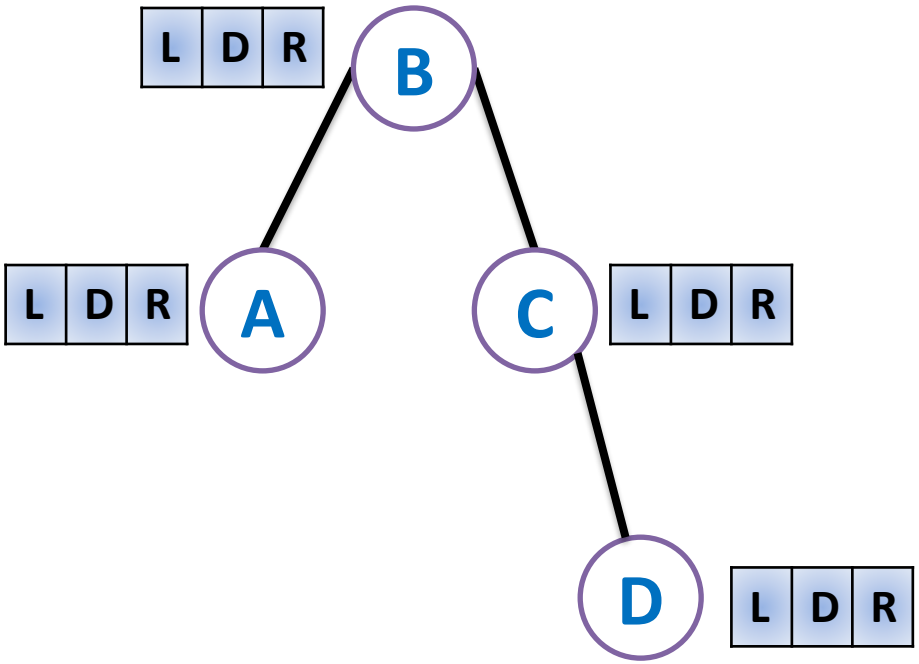
Inorder traversal

Algorithm Inorder(tree)

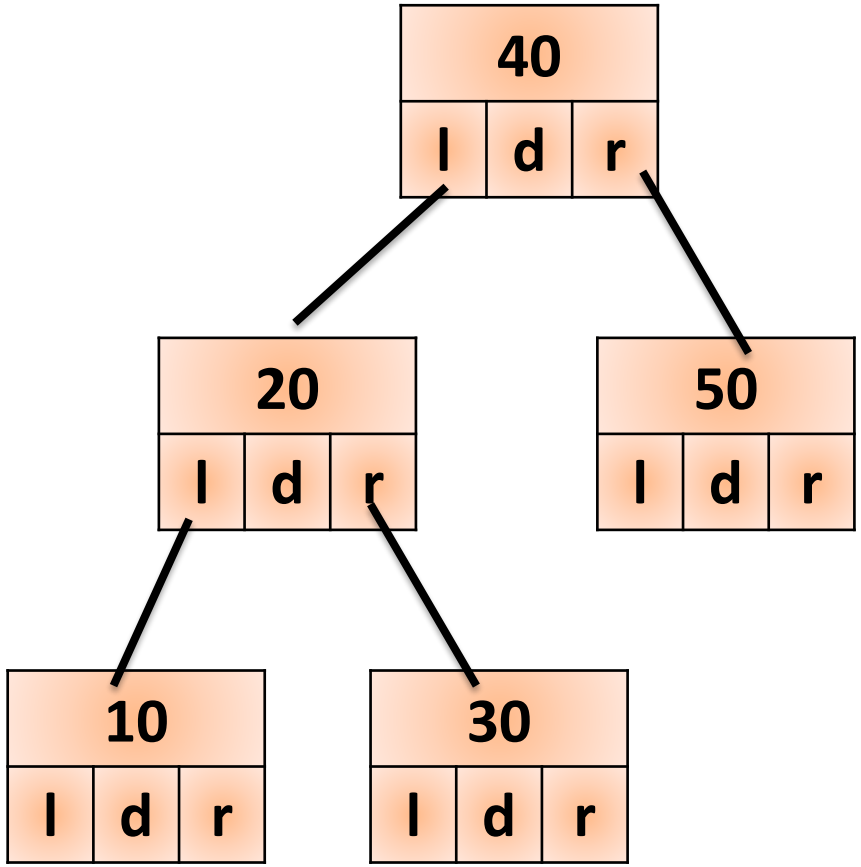
Traverse the left subtree, i.e., call Inorder(left-subtree)

Print Data.

Traverse the right subtree, i.e., call Inorder(right-subtree)

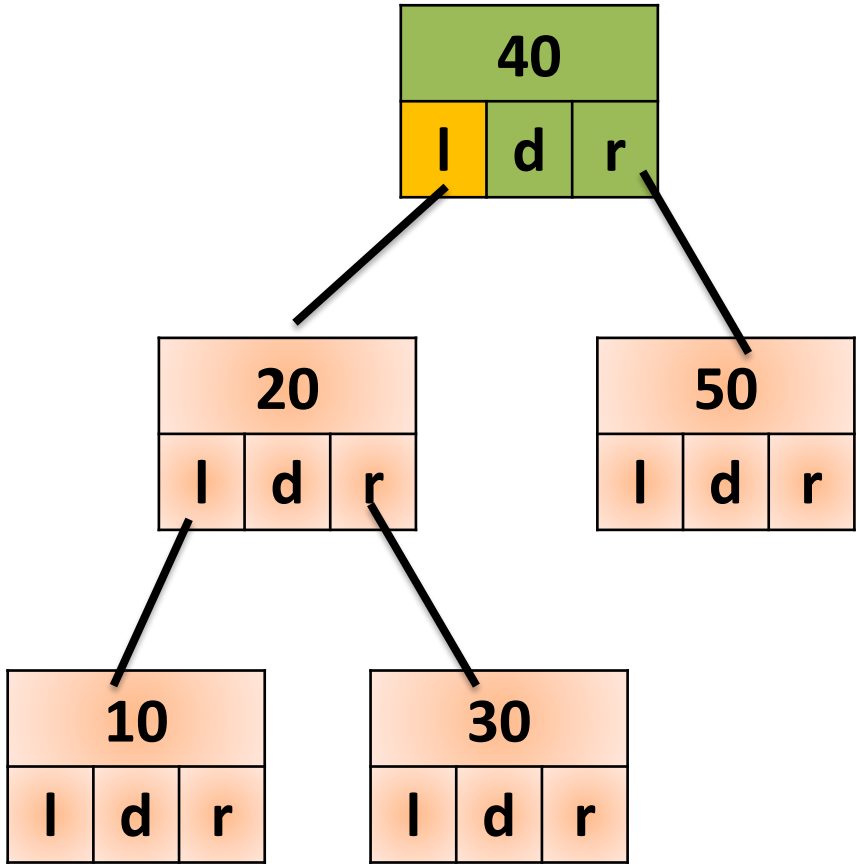


Inorder



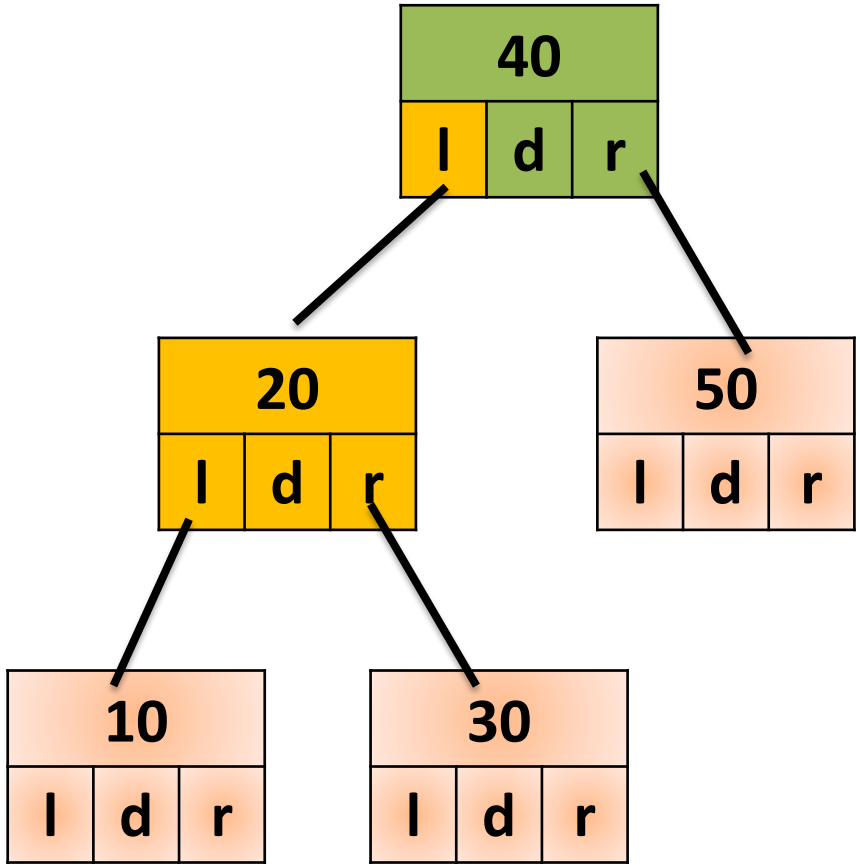
OUTPUT

Inorder



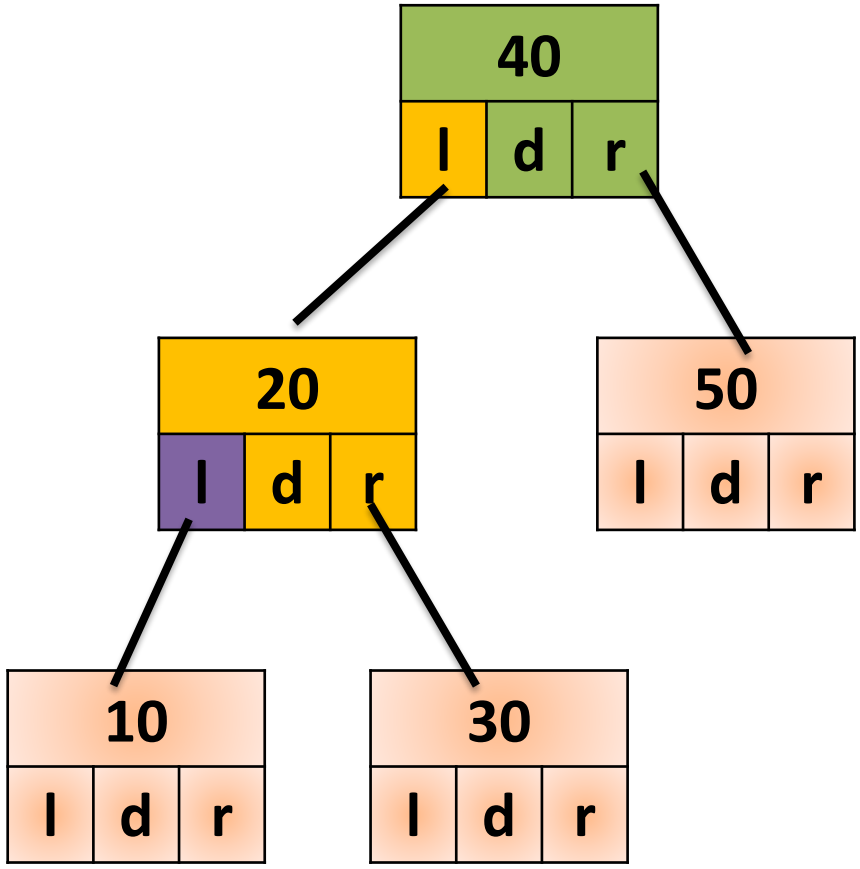
OUTPUT

Inorder



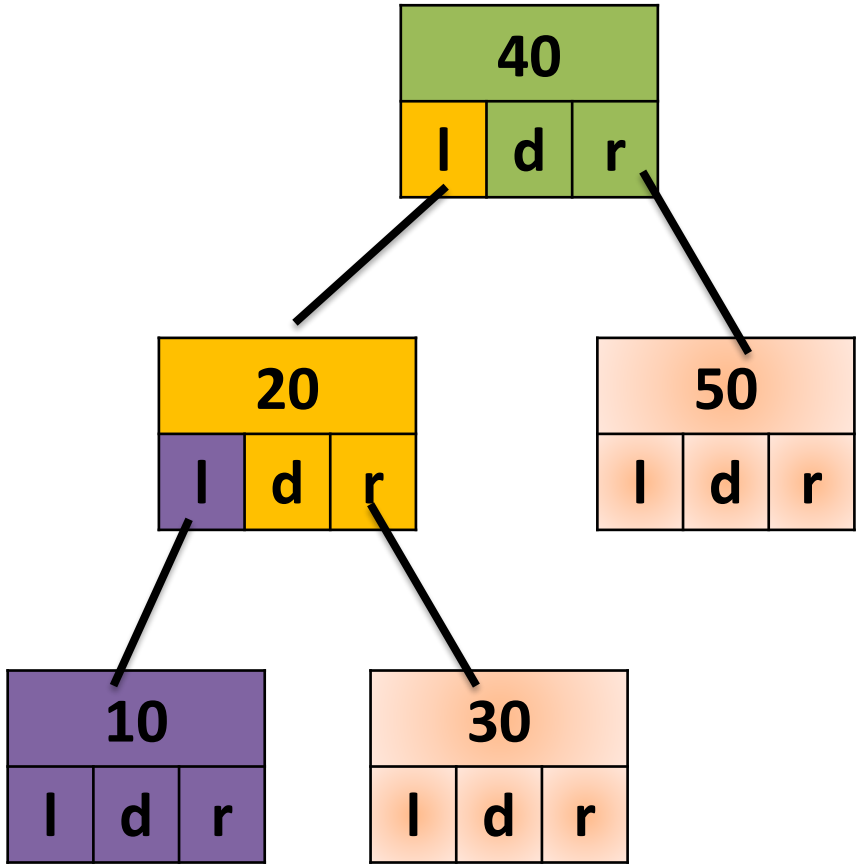
OUTPUT

Inorder



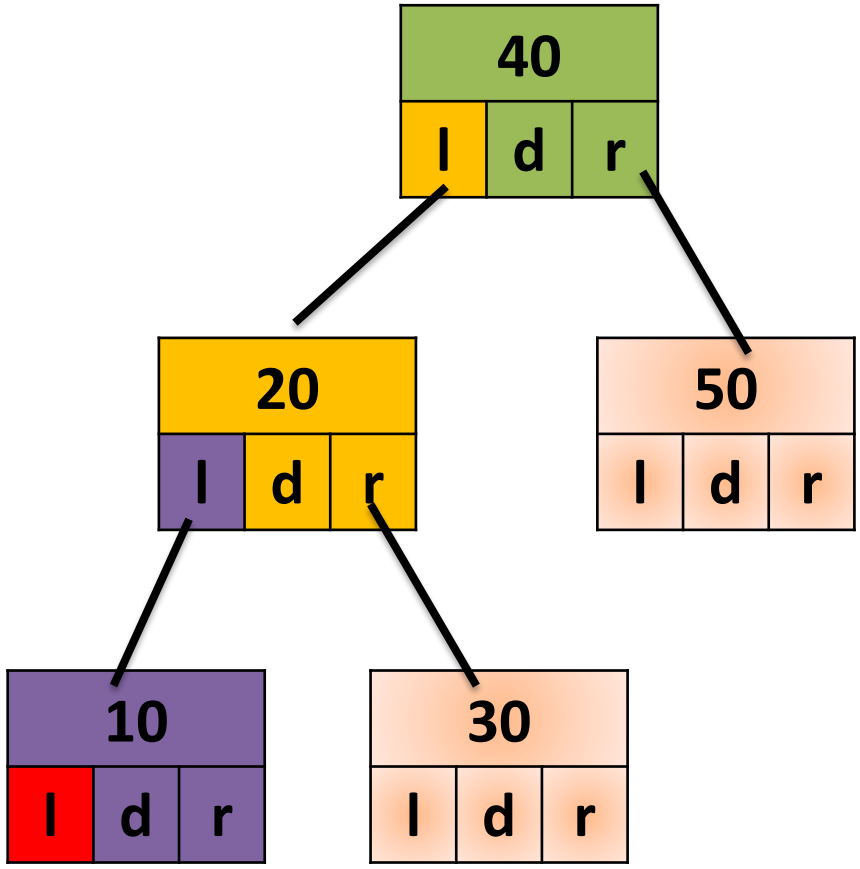
OUTPUT

Inorder



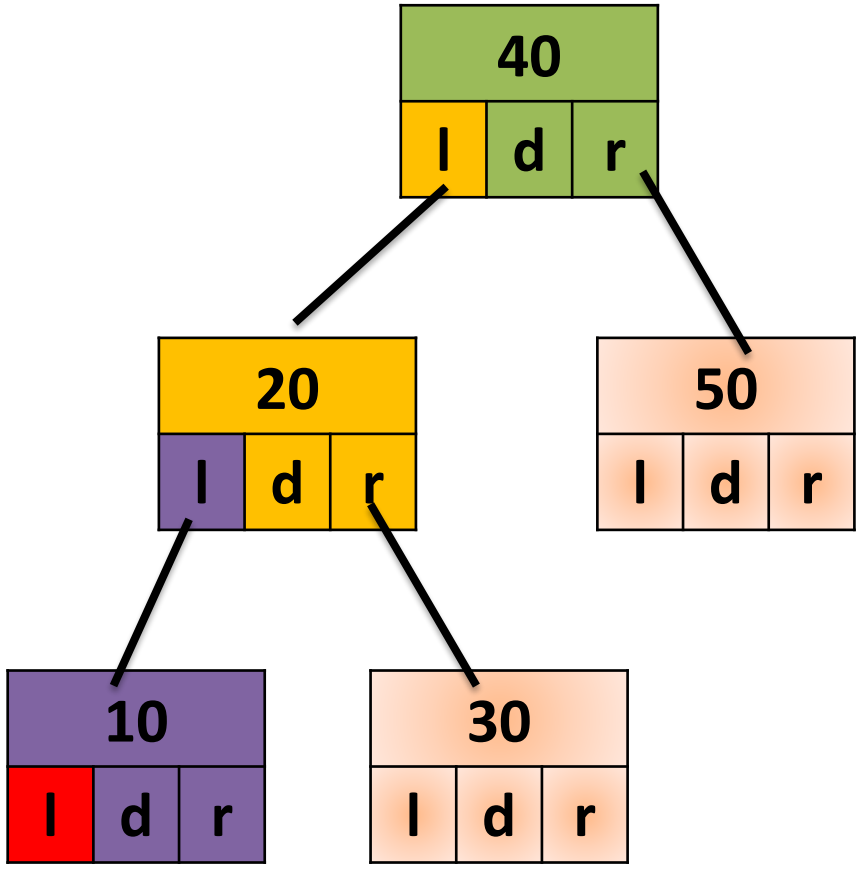
OUTPUT

Inorder



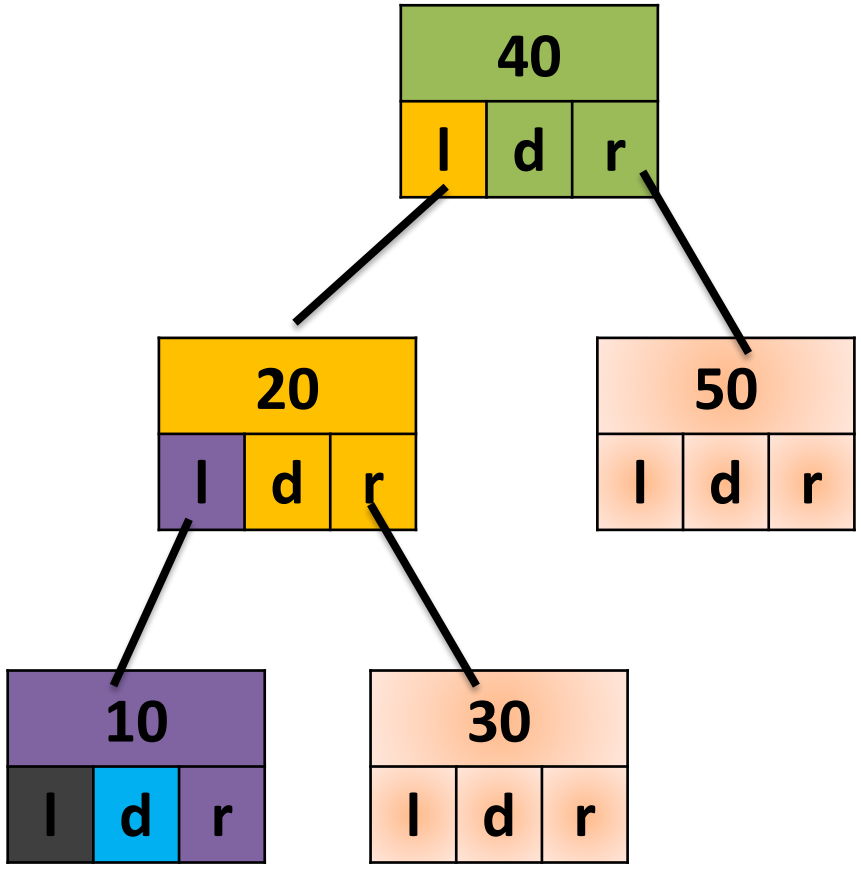
OUTPUT

Inorder



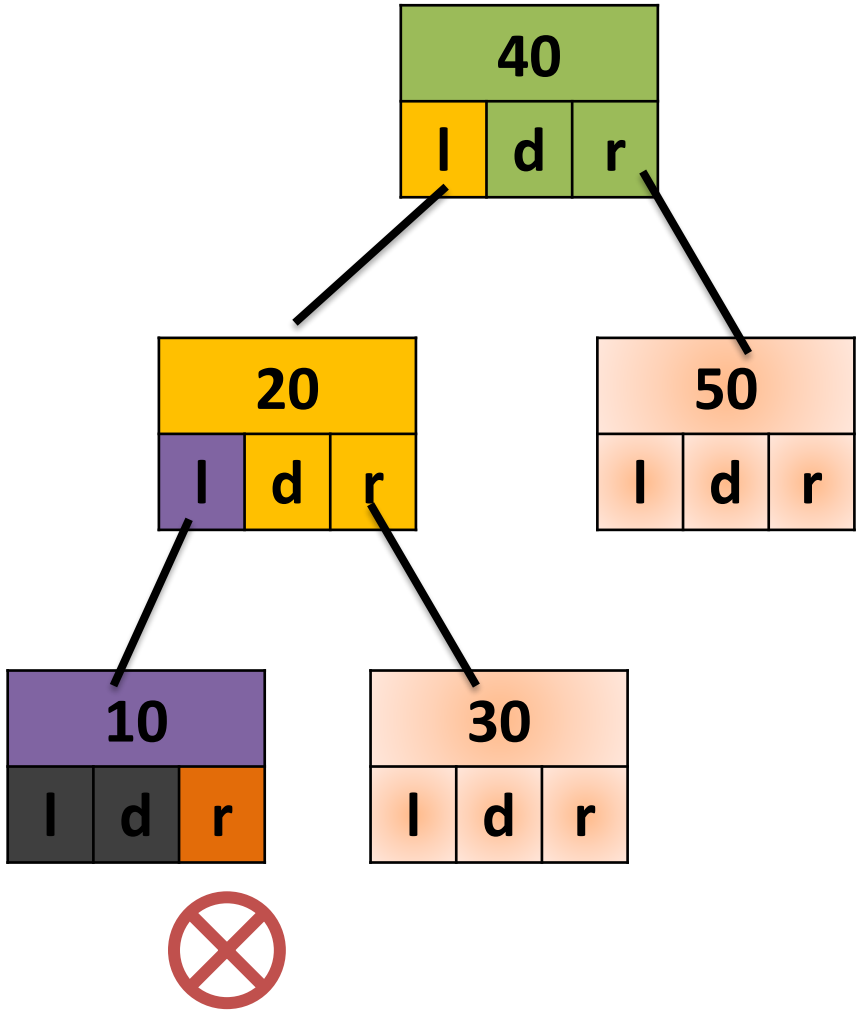
OUTPUT

Inorder



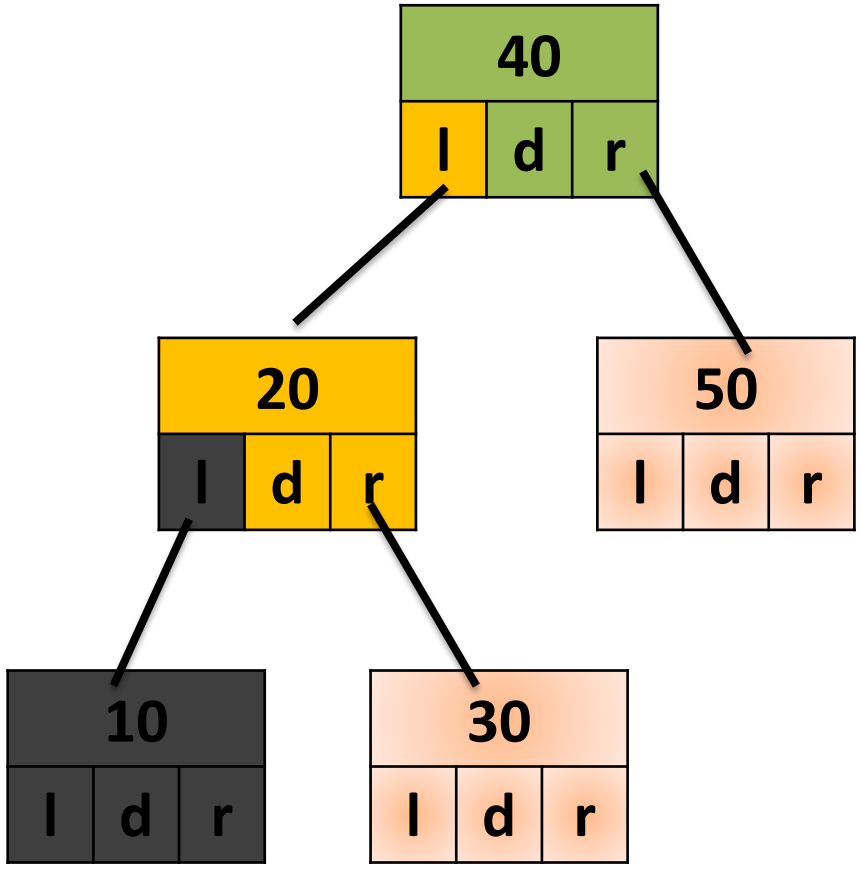
OUTPUT

Inorder



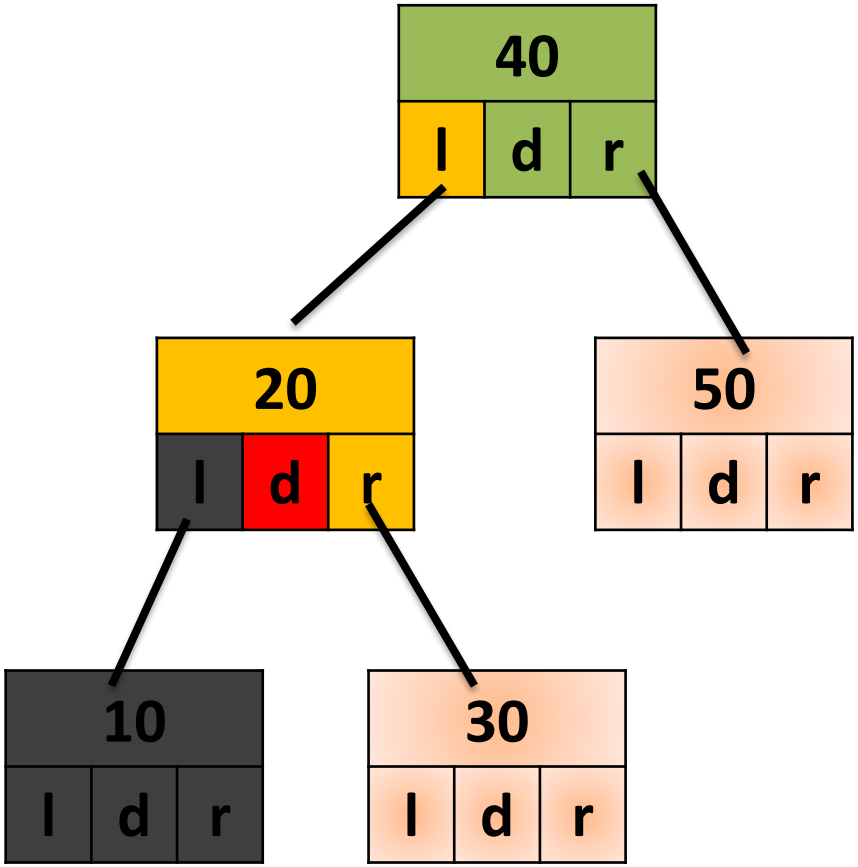
OUTPUT

Inorder

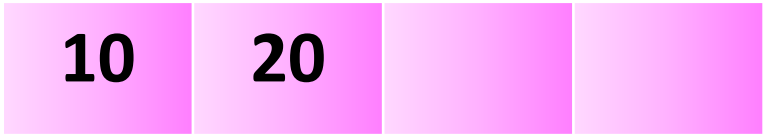


OUTPUT

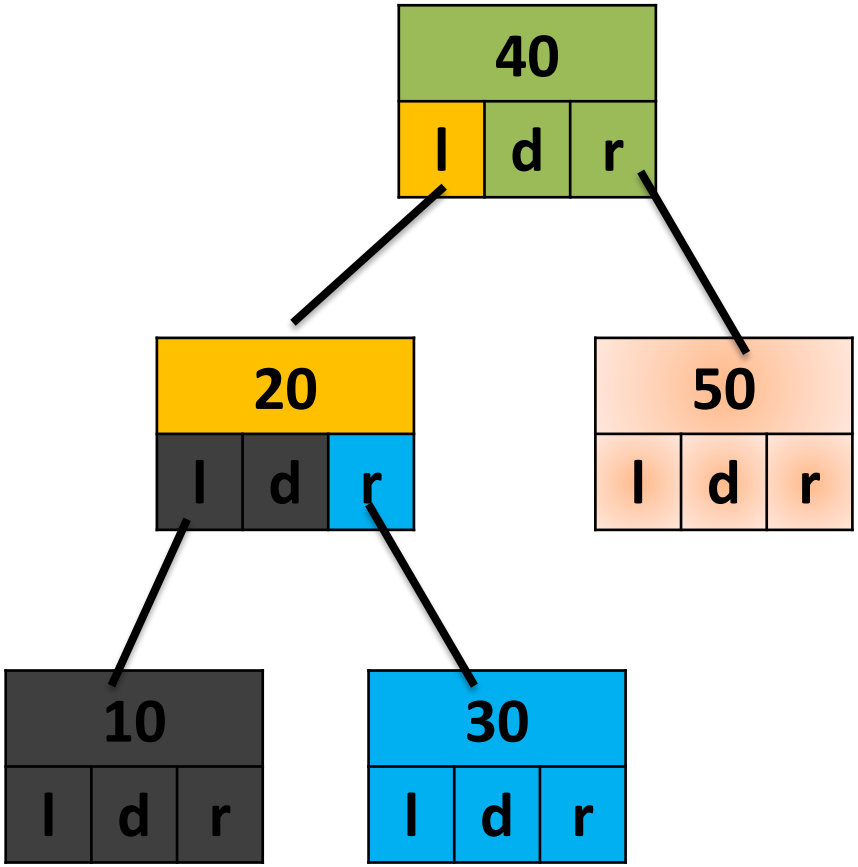
Inorder



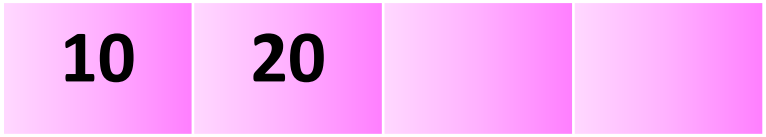
OUTPUT



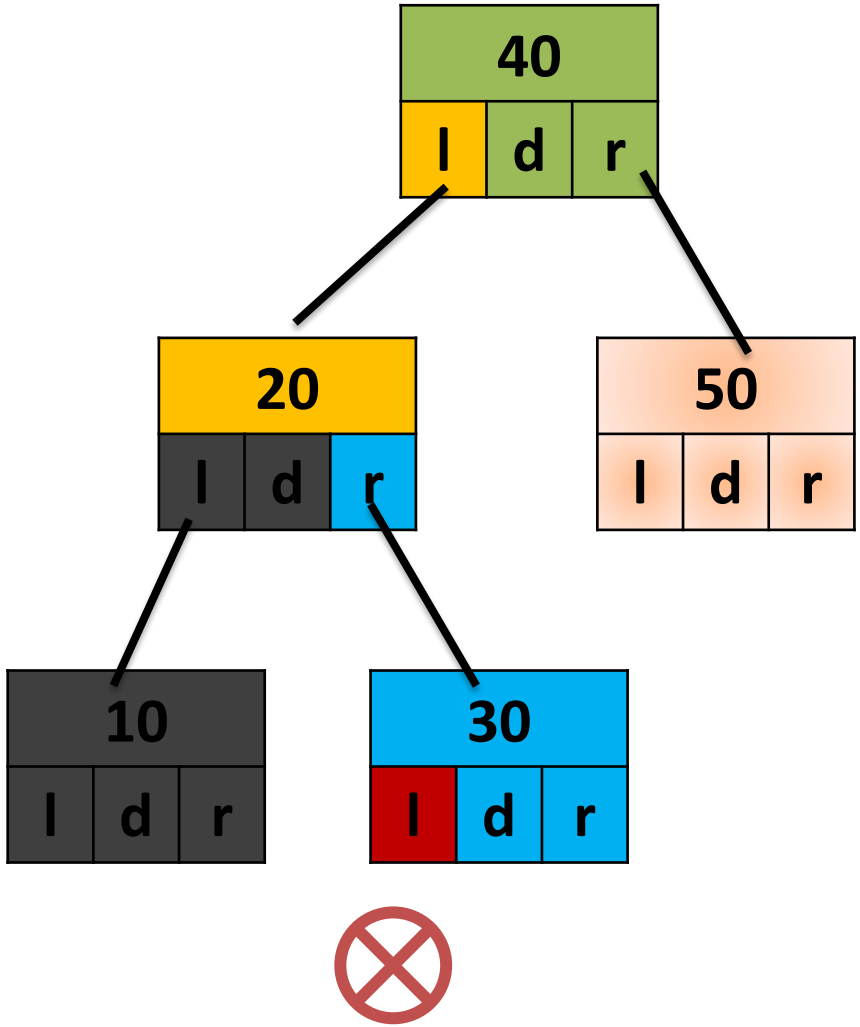
Inorder



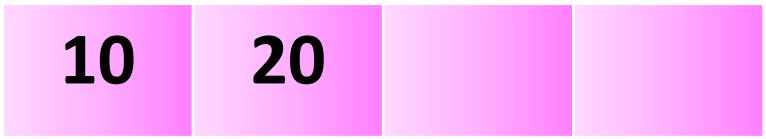
OUTPUT



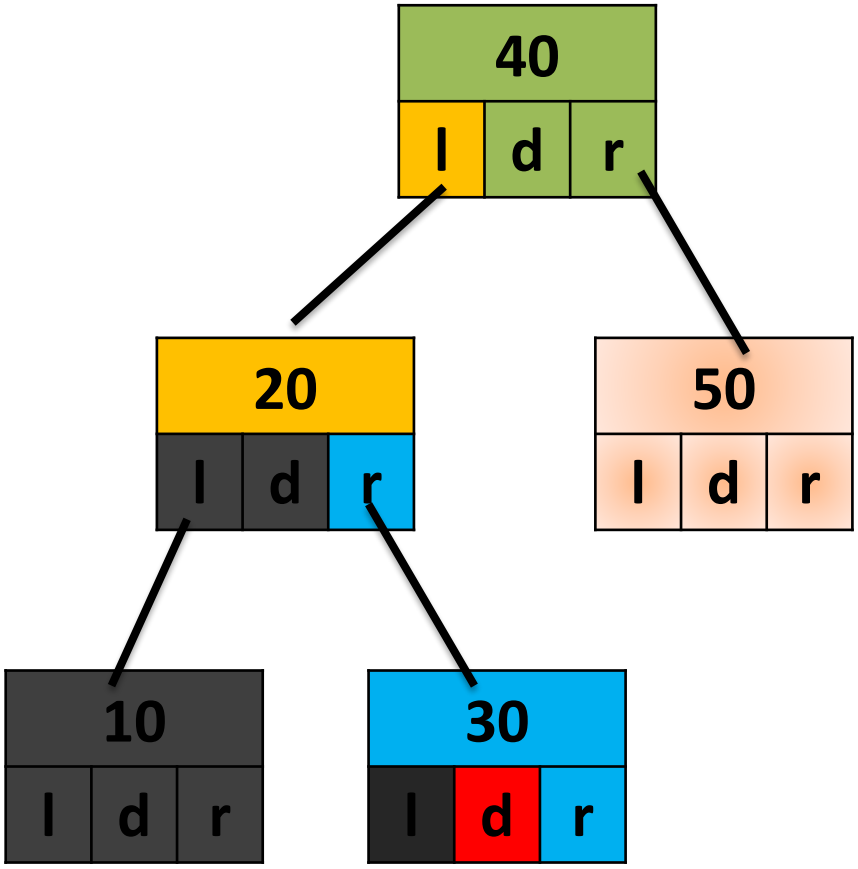
Inorder



OUTPUT



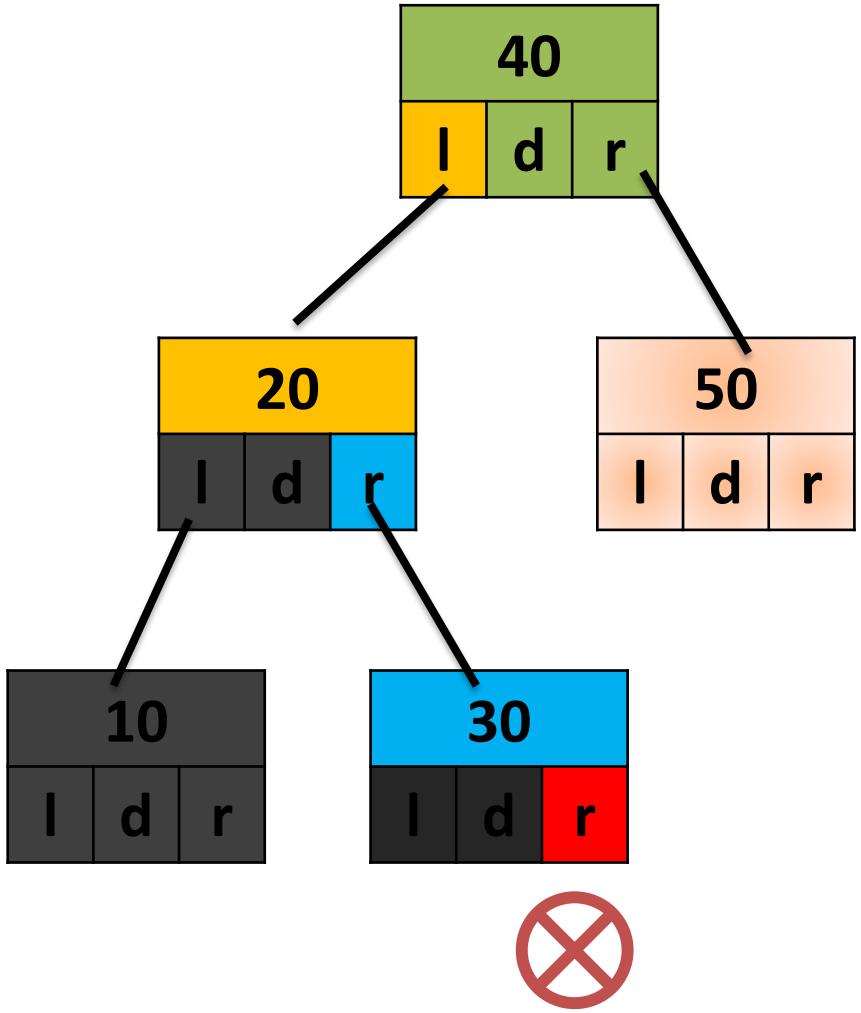
Inorder



OUTPUT

10	20	30	

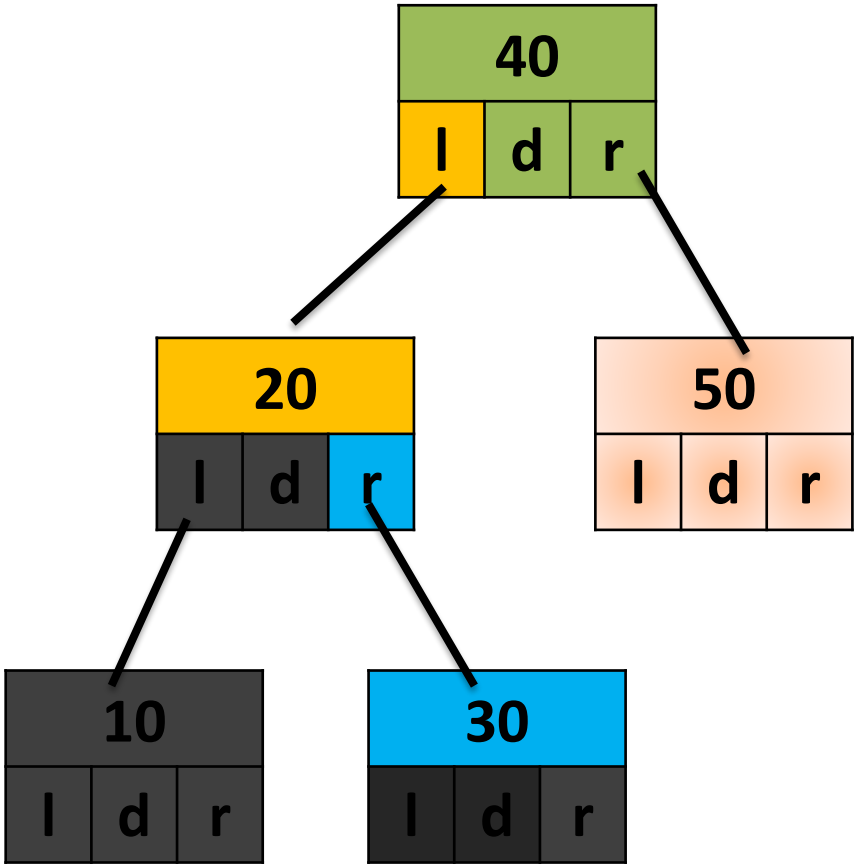
Inorder



OUTPUT

10	20	30	

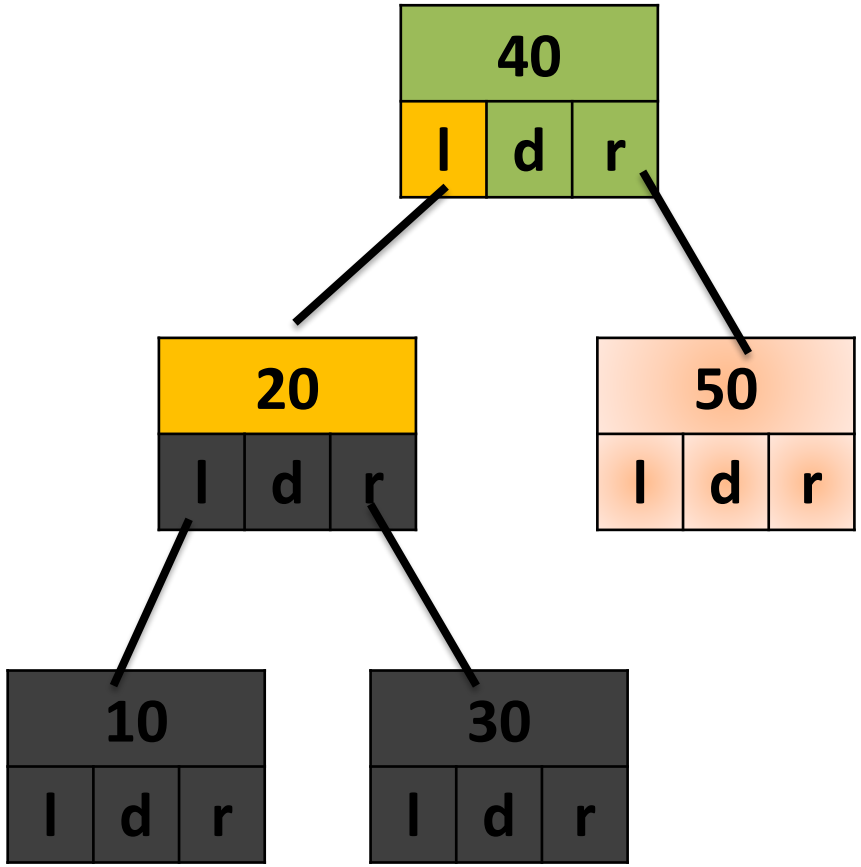
Inorder



OUTPUT

10	20	30	

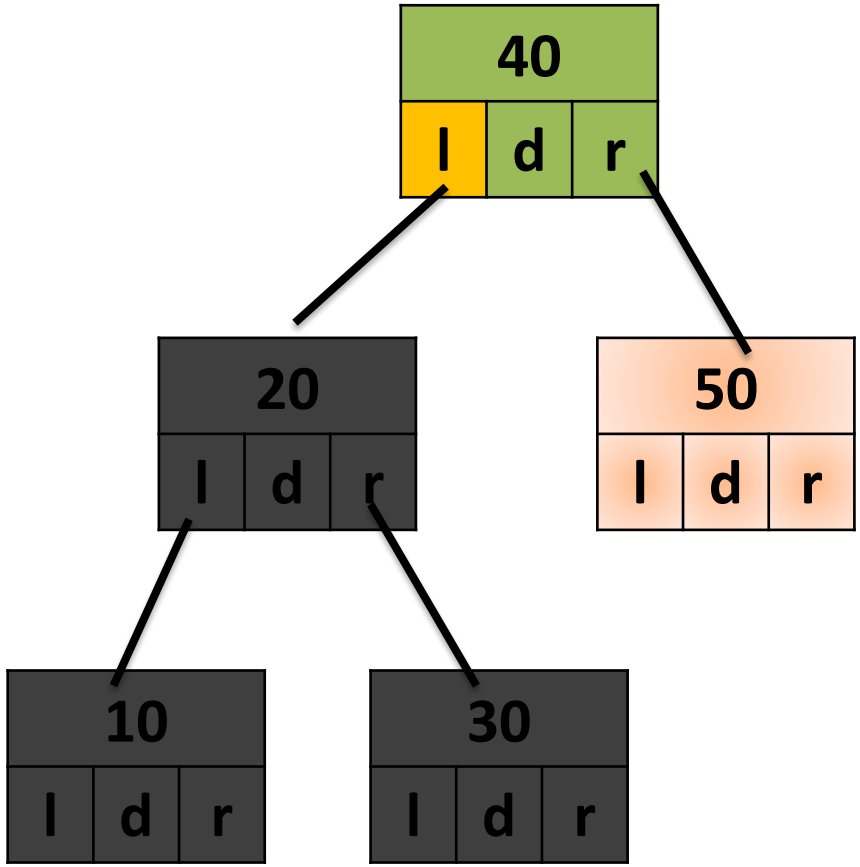
Inorder



OUTPUT

10	20	30	

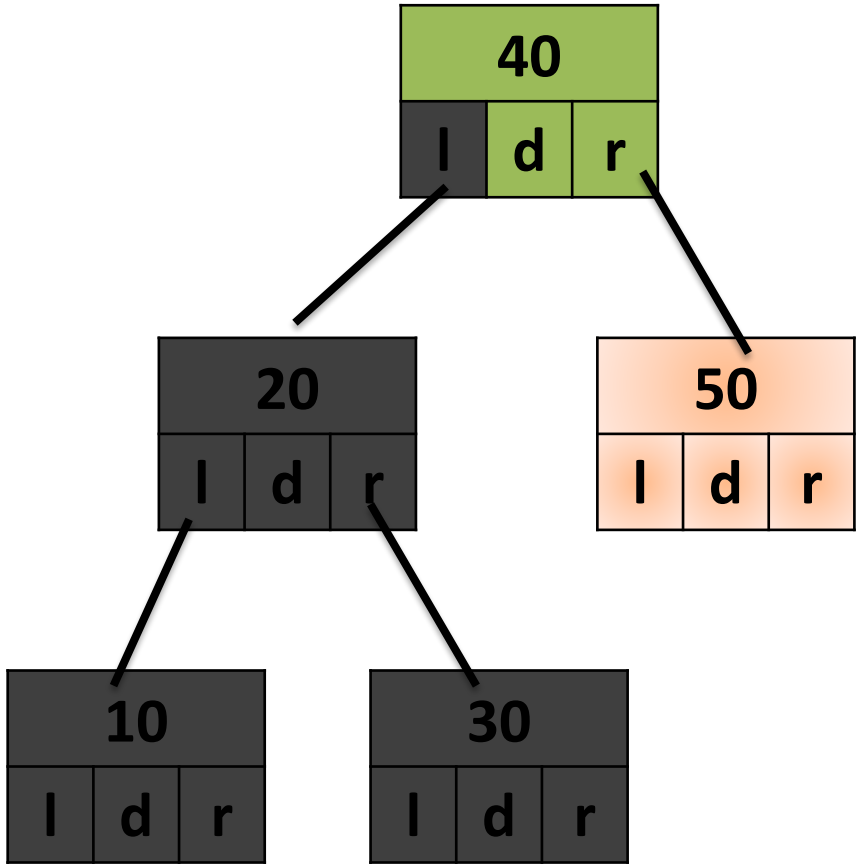
Inorder



OUTPUT

10	20	30	

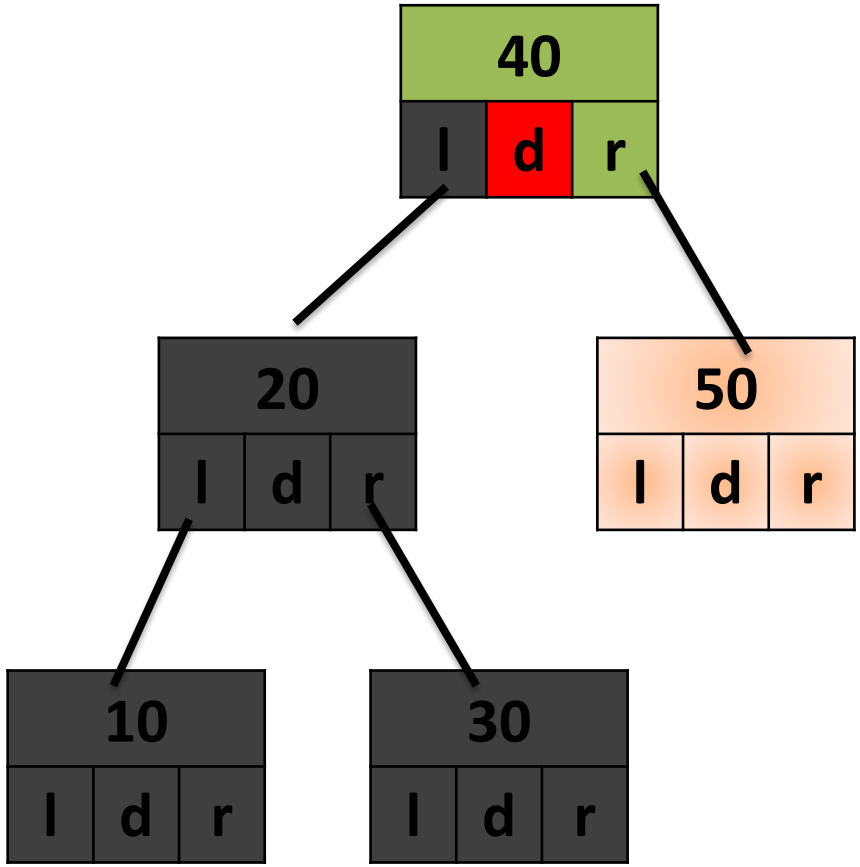
Inorder



OUTPUT

10	20	30	

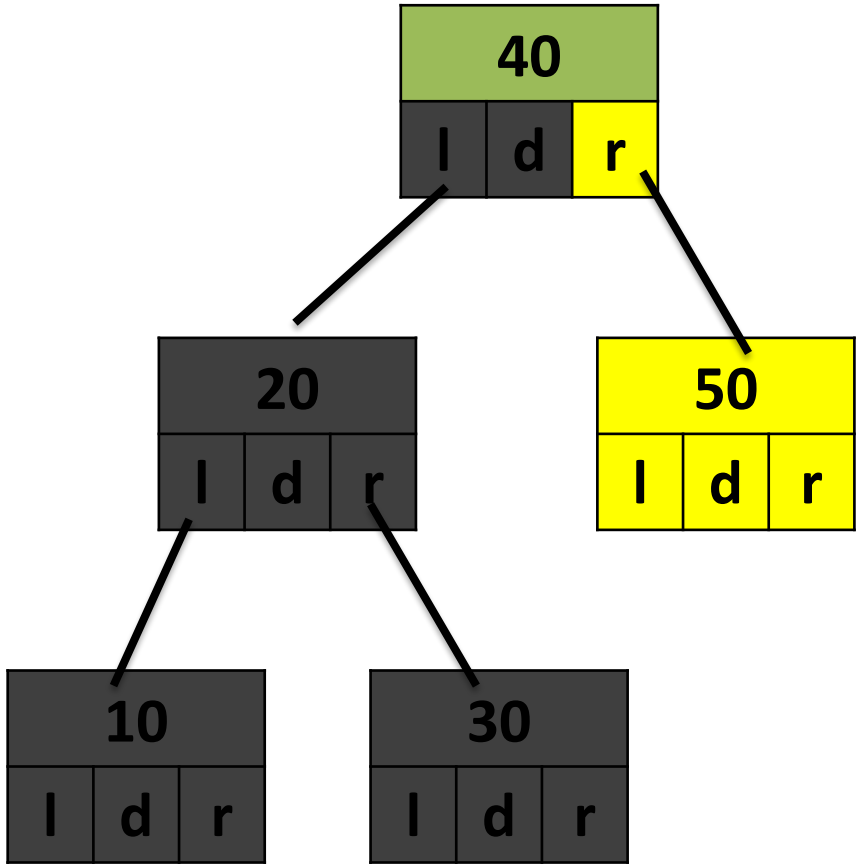
Inorder



OUTPUT

10	20	30	40	

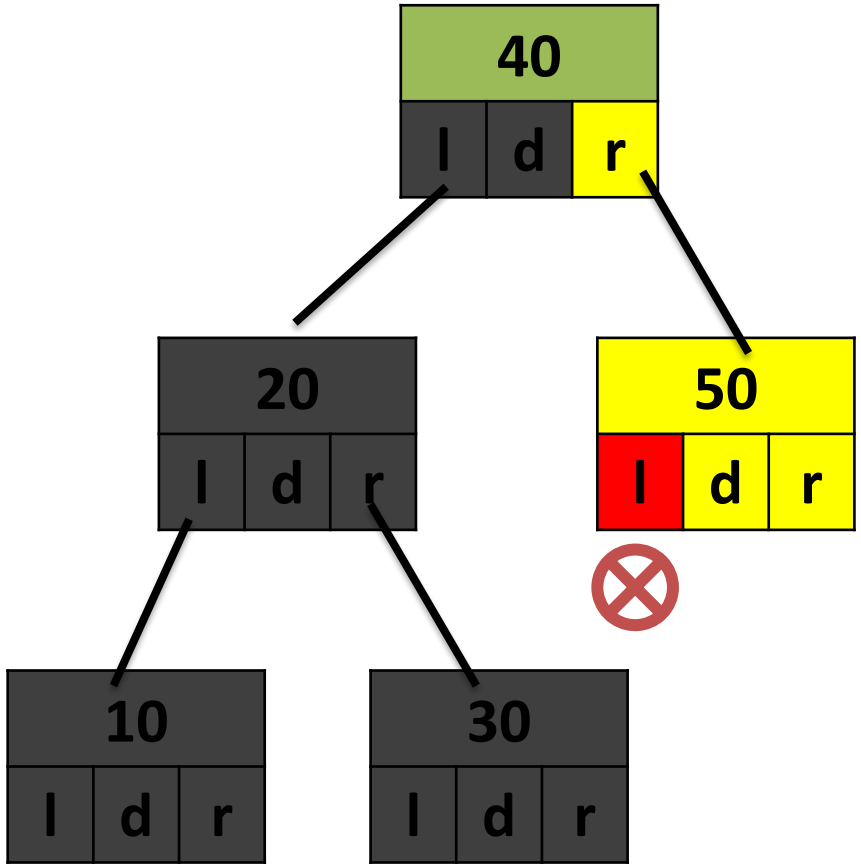
Inorder



OUTPUT

10	20	30	40	

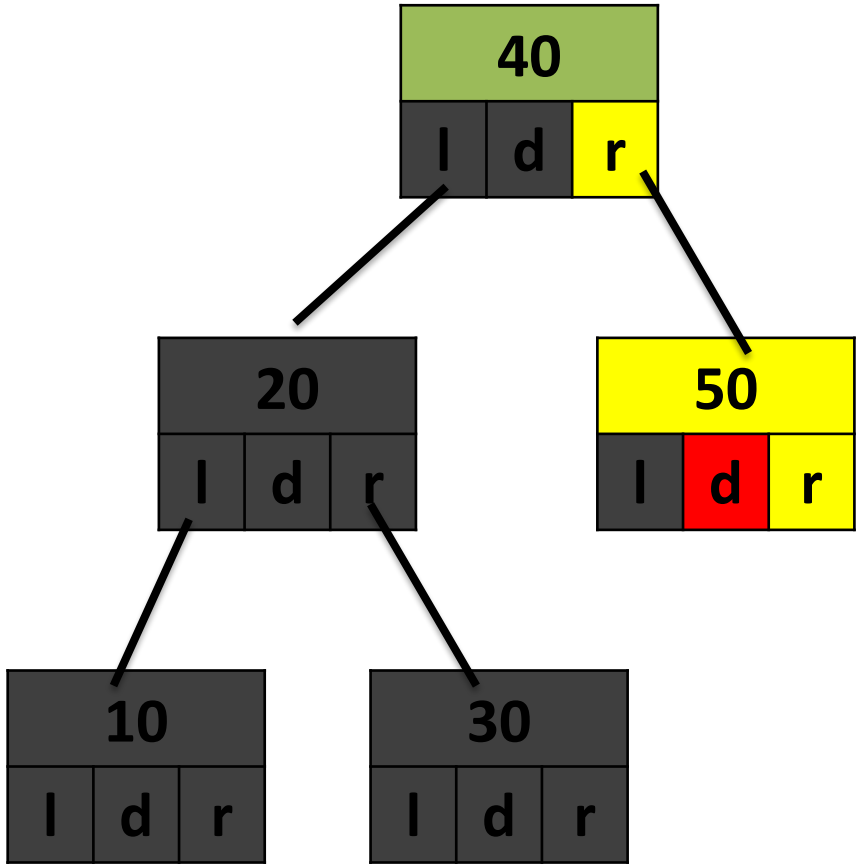
Inorder



OUTPUT

10	20	30	40	

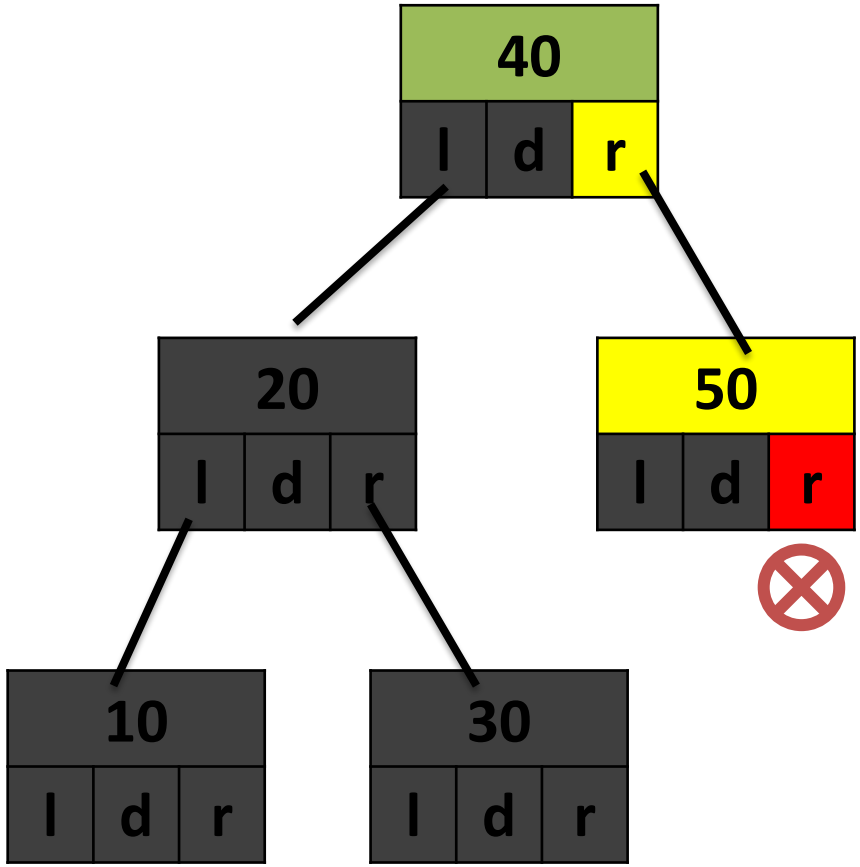
Inorder



OUTPUT

10	20	30	40	50

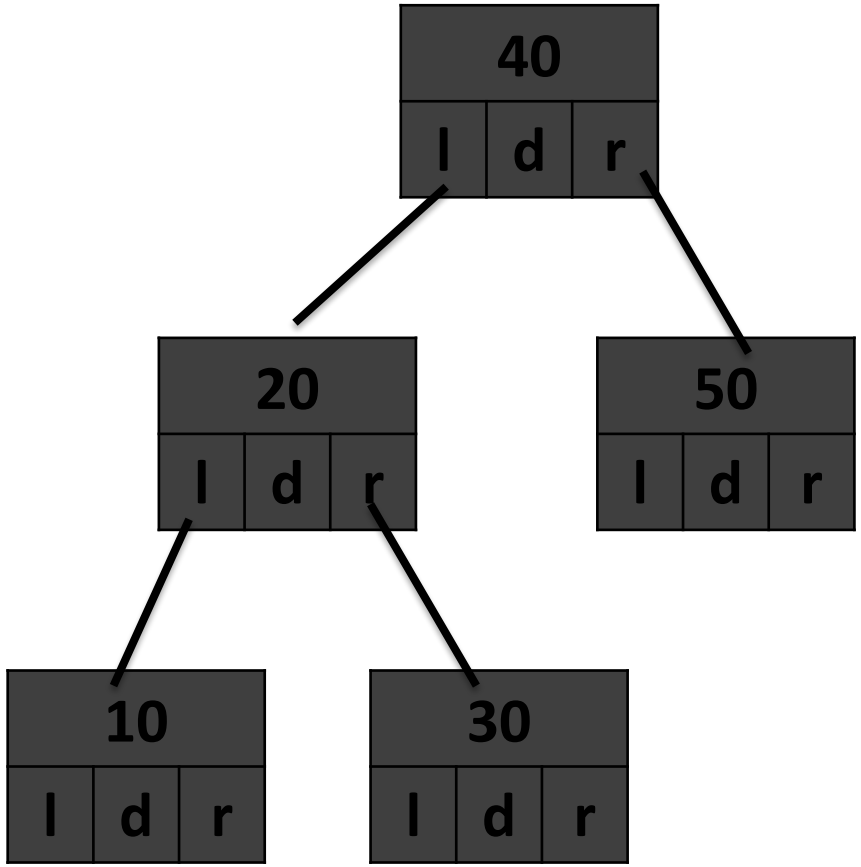
Inorder



OUTPUT

10	20	30	40	50

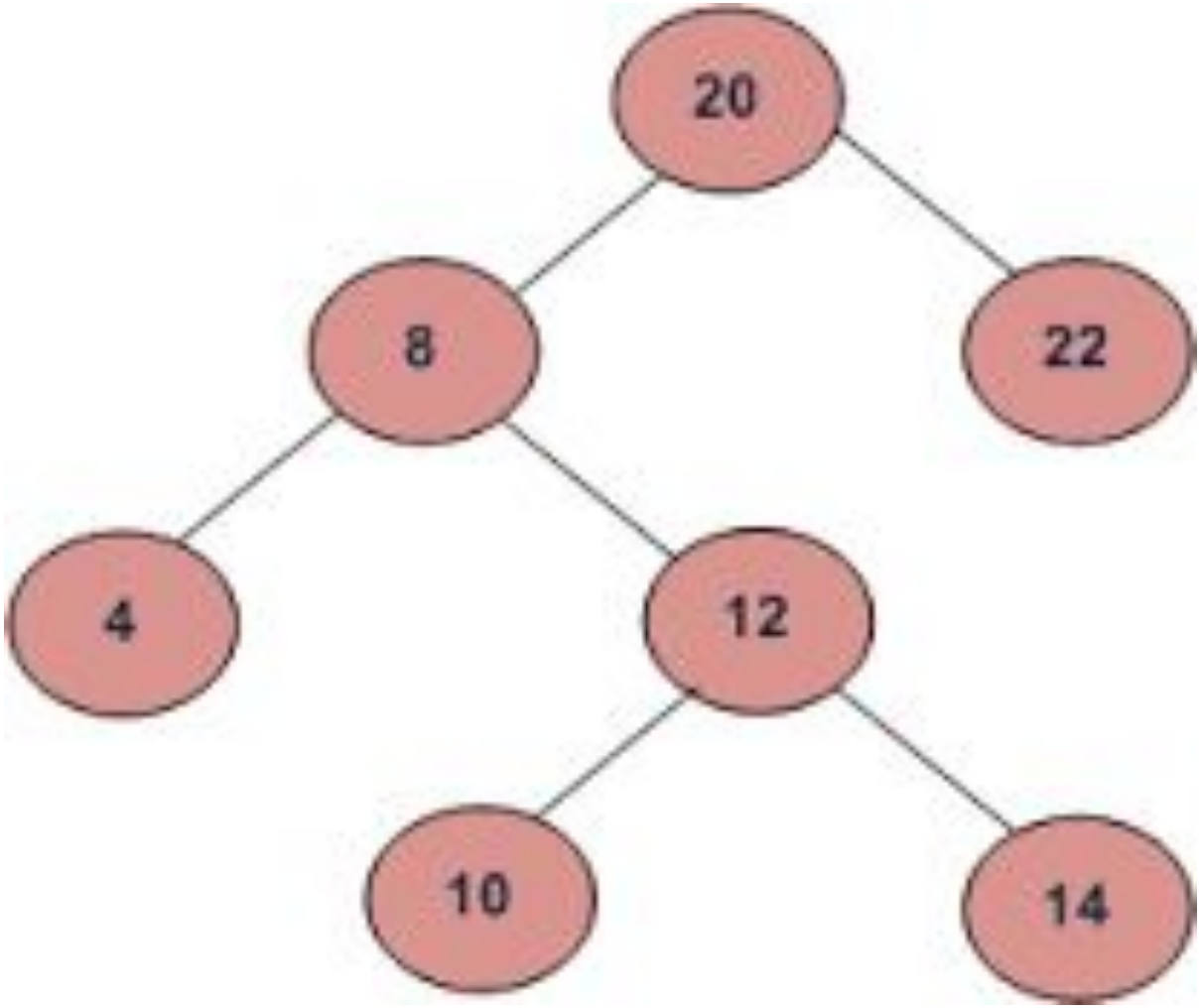
Inorder



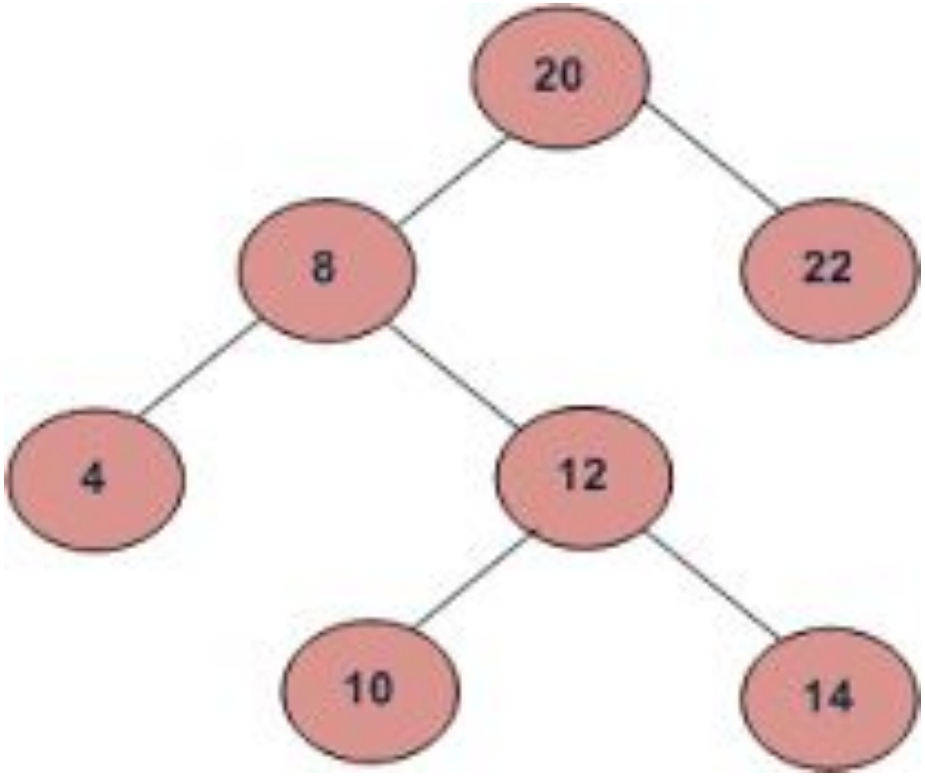
OUTPUT

10	20	30	40	50

Find inorder traversal sequence of this tree.



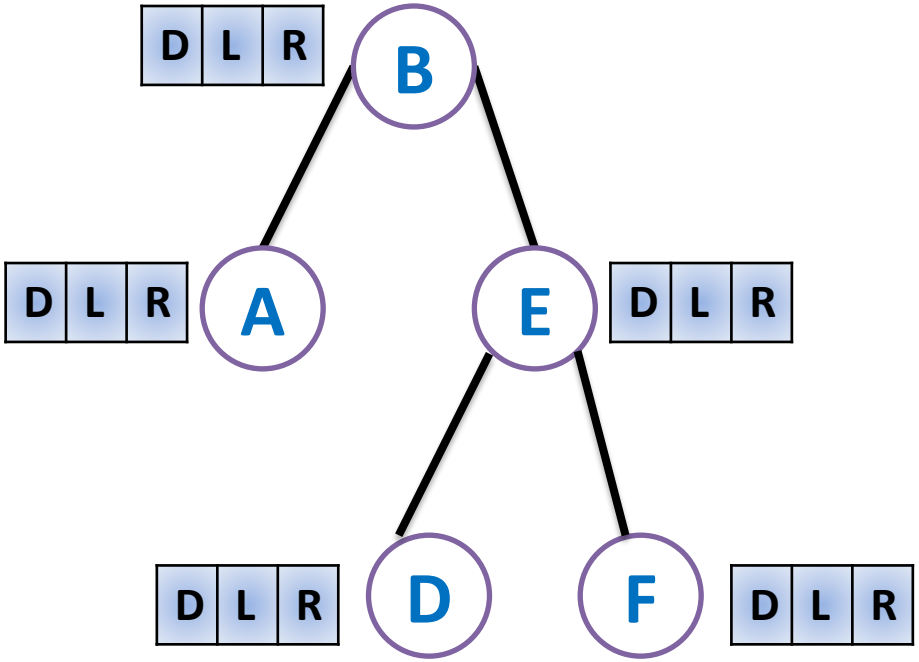
Find inorder traversal sequence of this tree.



Answer : 4, 8,10, 12,14,20,22

Preorder traversal

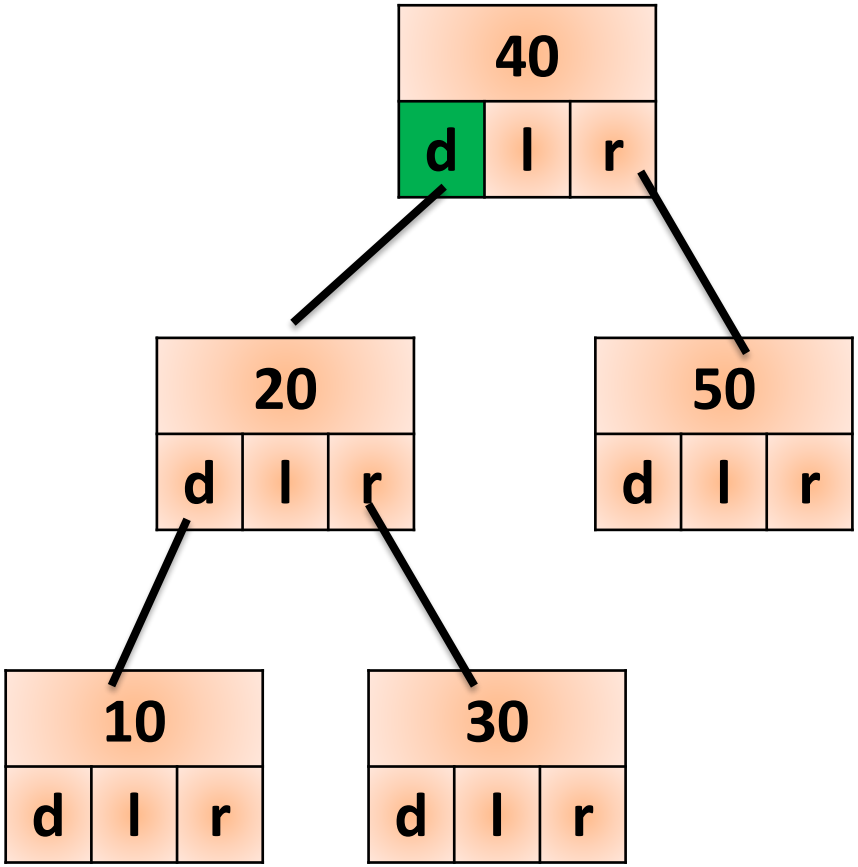
1. Print data.
2. Traverse the left subtree
3. Traverse the right subtree



OUTPUT

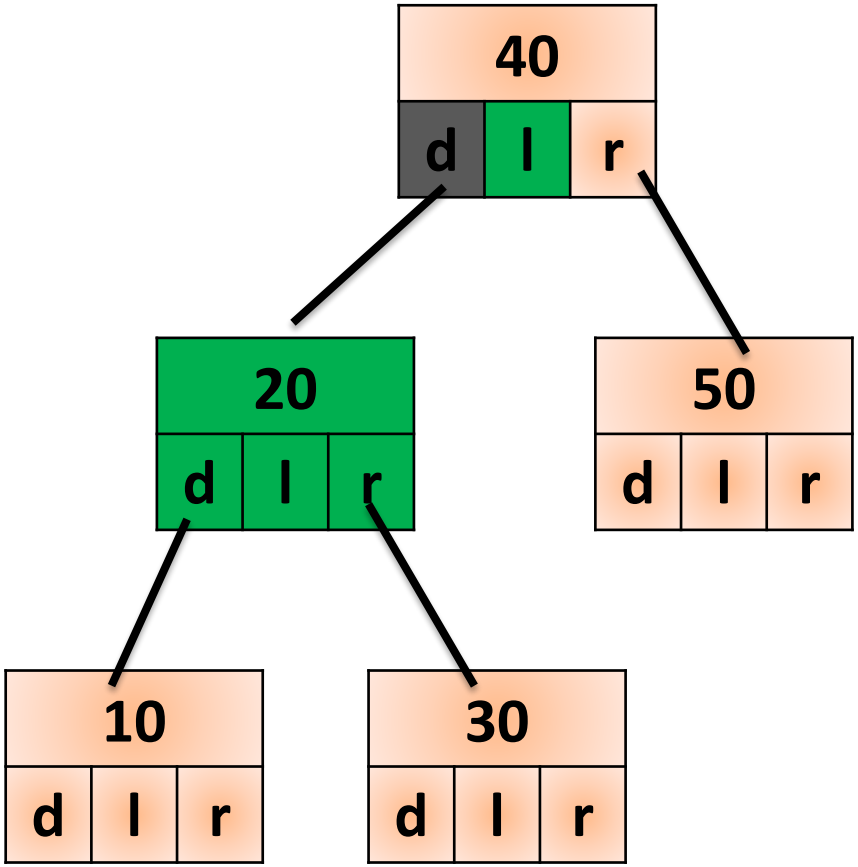
B	A	E	D	F

preorder



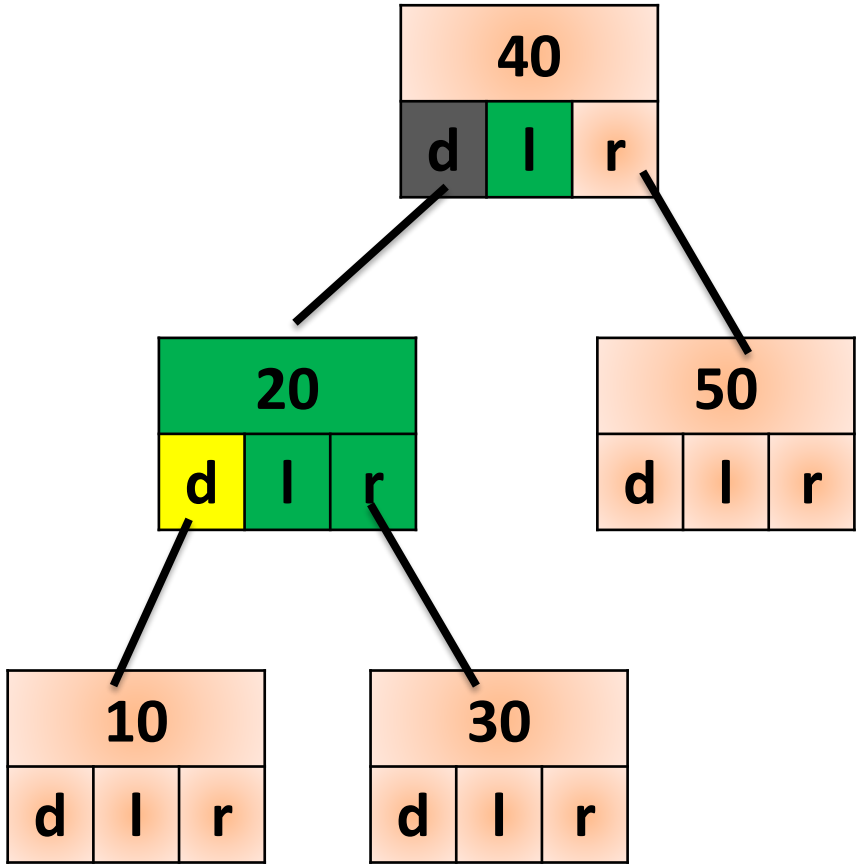
OUTPUT

preorder



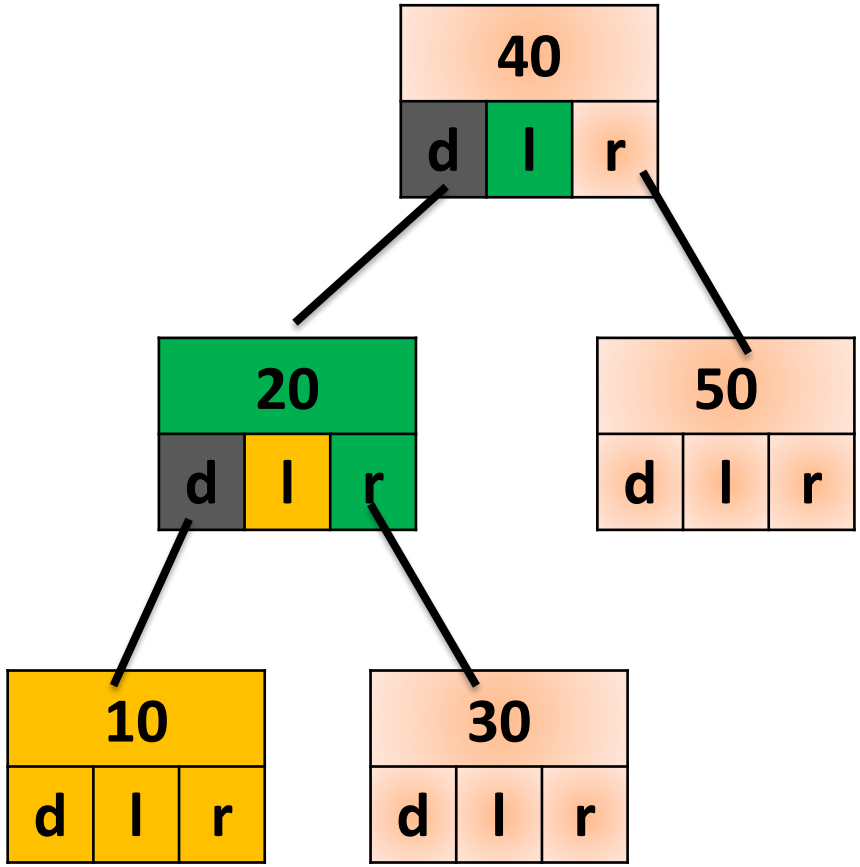
OUTPUT

preorder



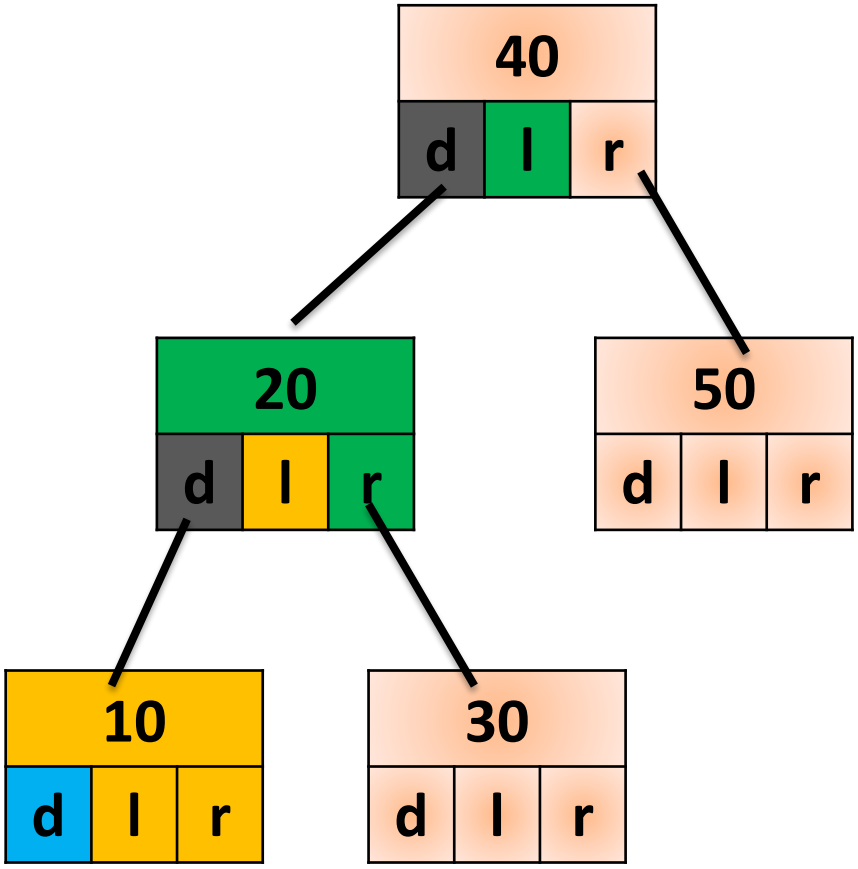
OUTPUT

preorder



OUTPUT

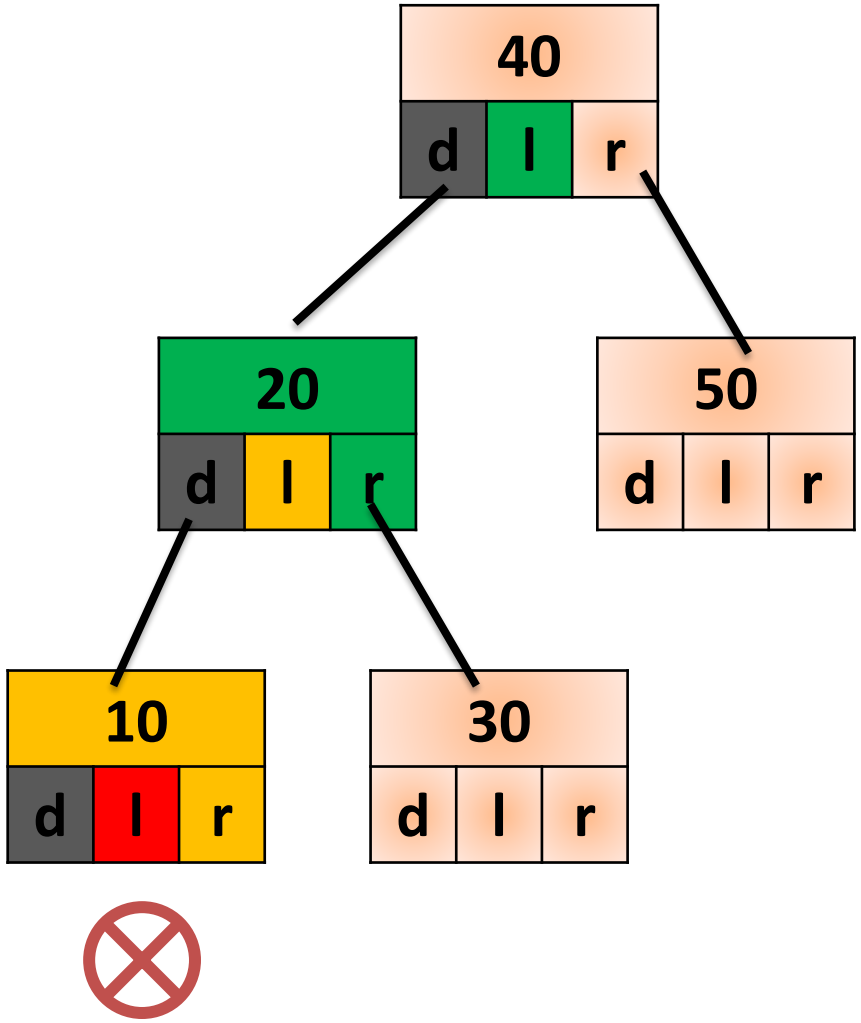
preorder



OUTPUT

40	20	10		

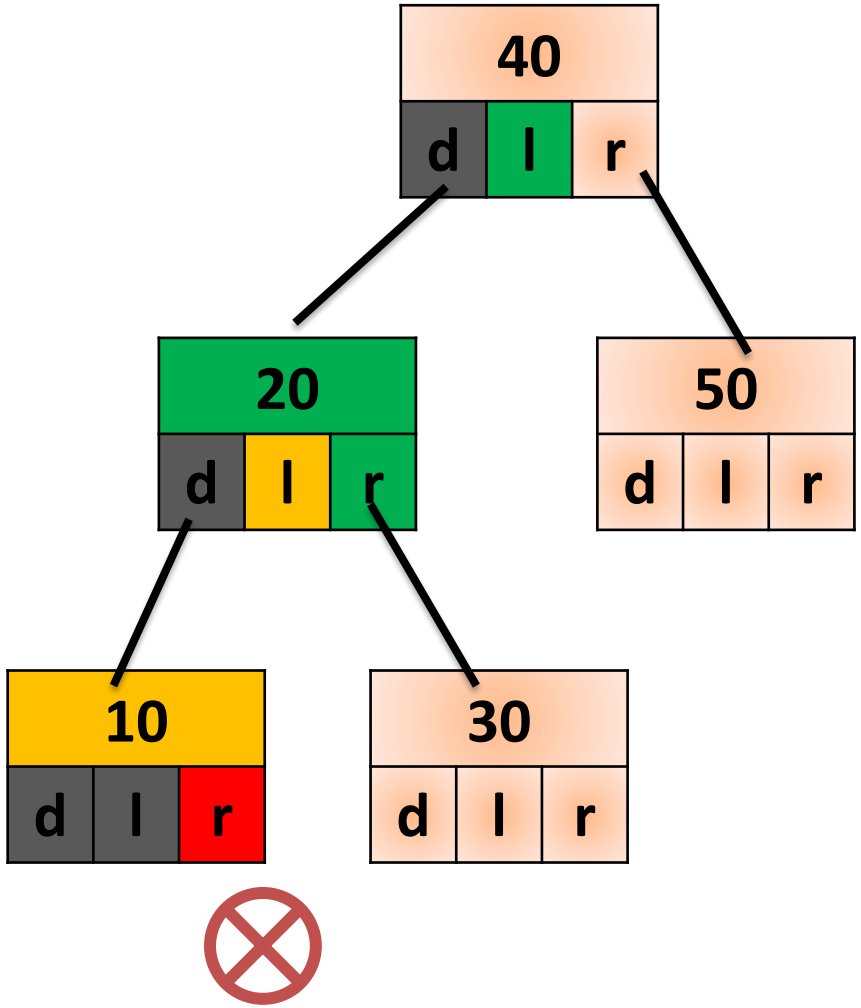
preorder



OUTPUT

40	20	10		

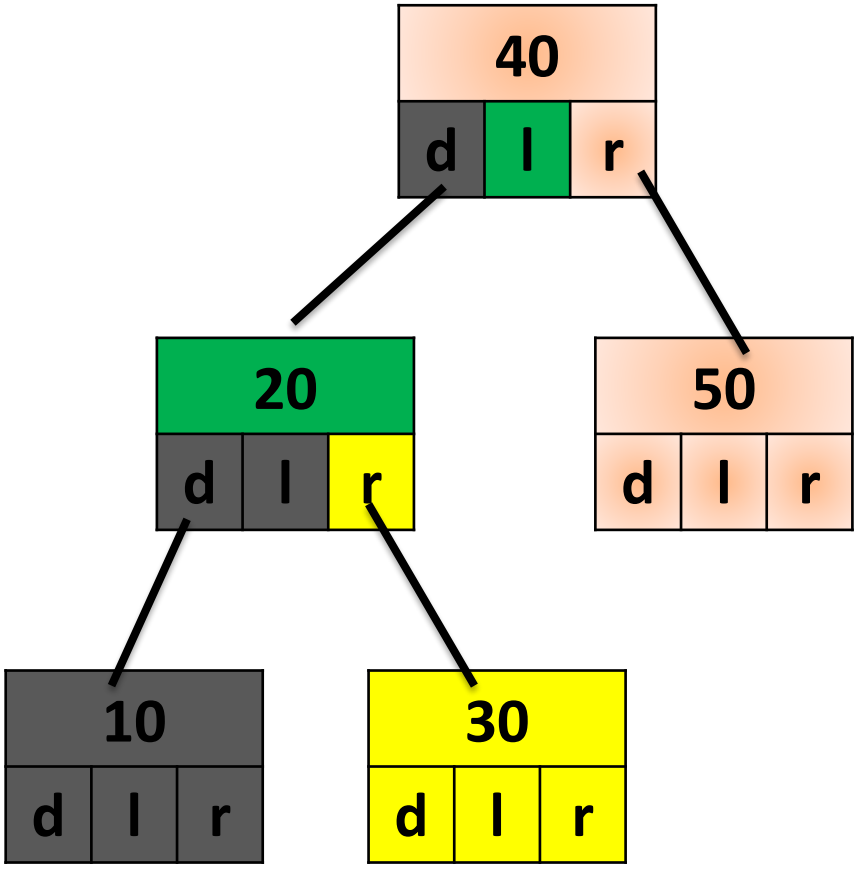
preorder



OUTPUT

40	20	10		

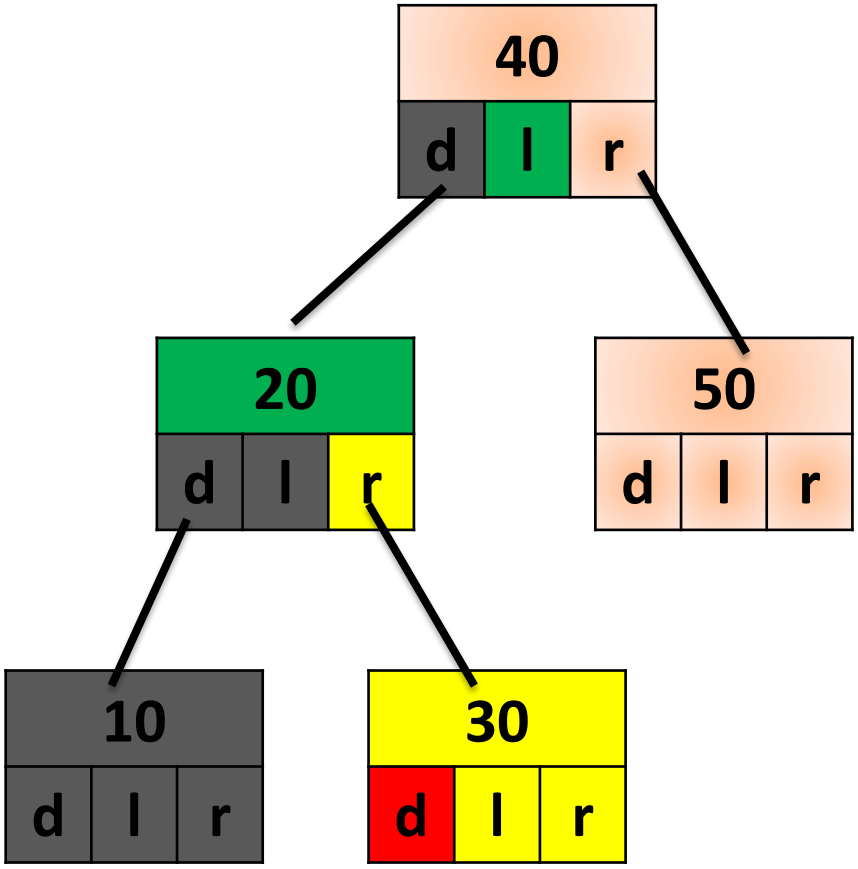
preorder



OUTPUT

40	20	10		

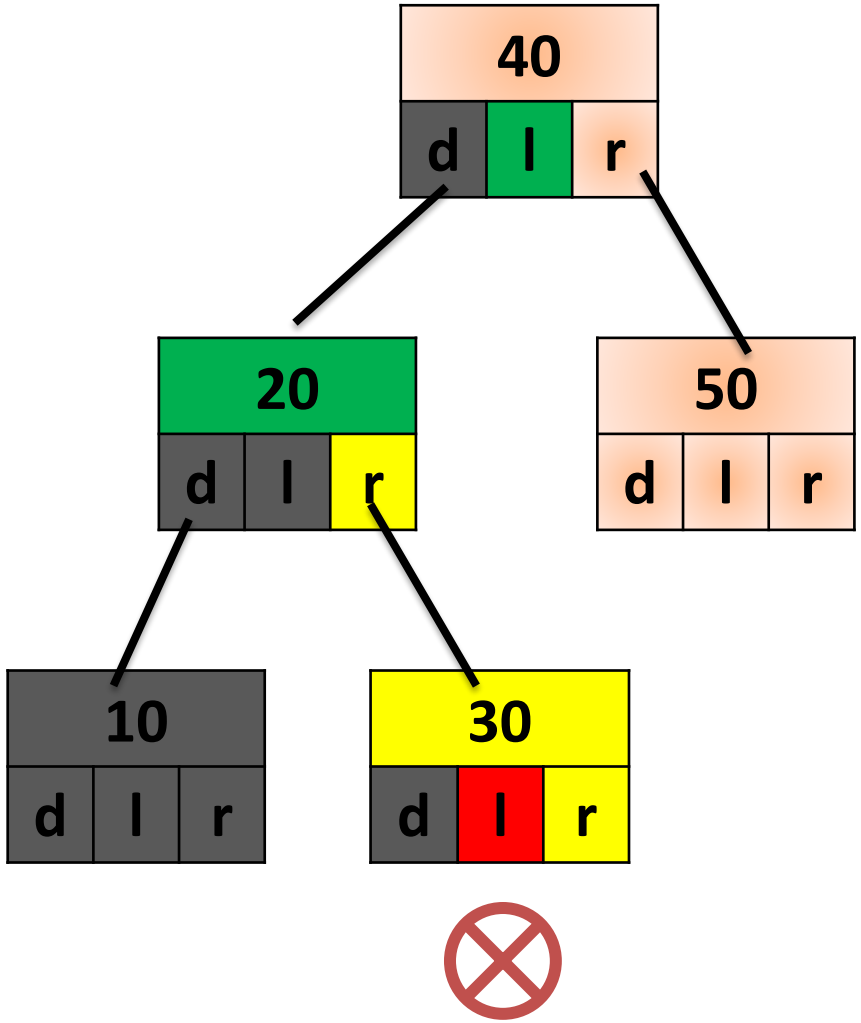
preorder



OUTPUT

40	20	10	30	

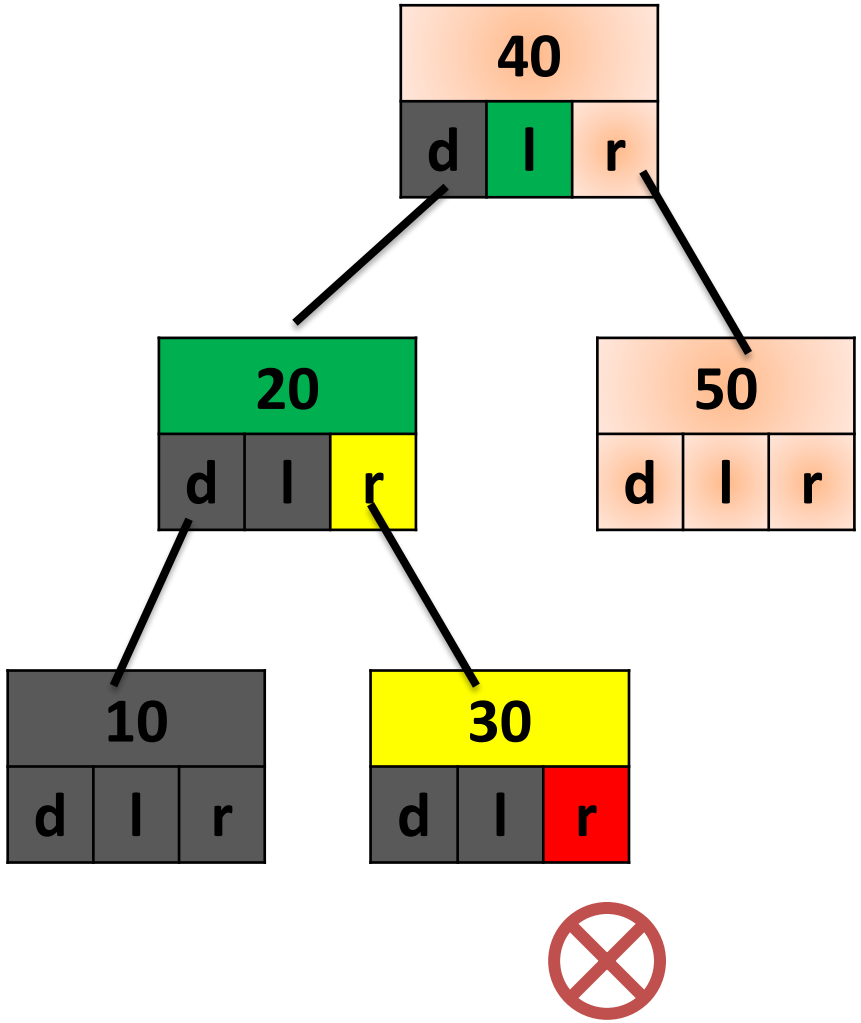
preorder



OUTPUT

40	20	10	30	

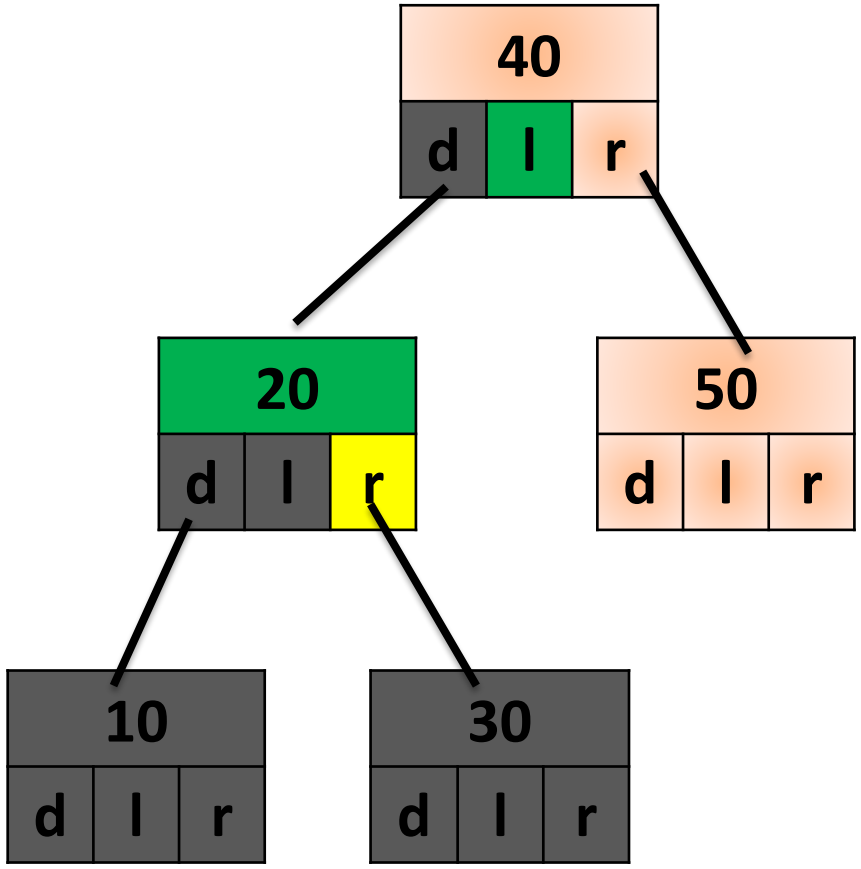
preorder



OUTPUT

40	20	10	30	

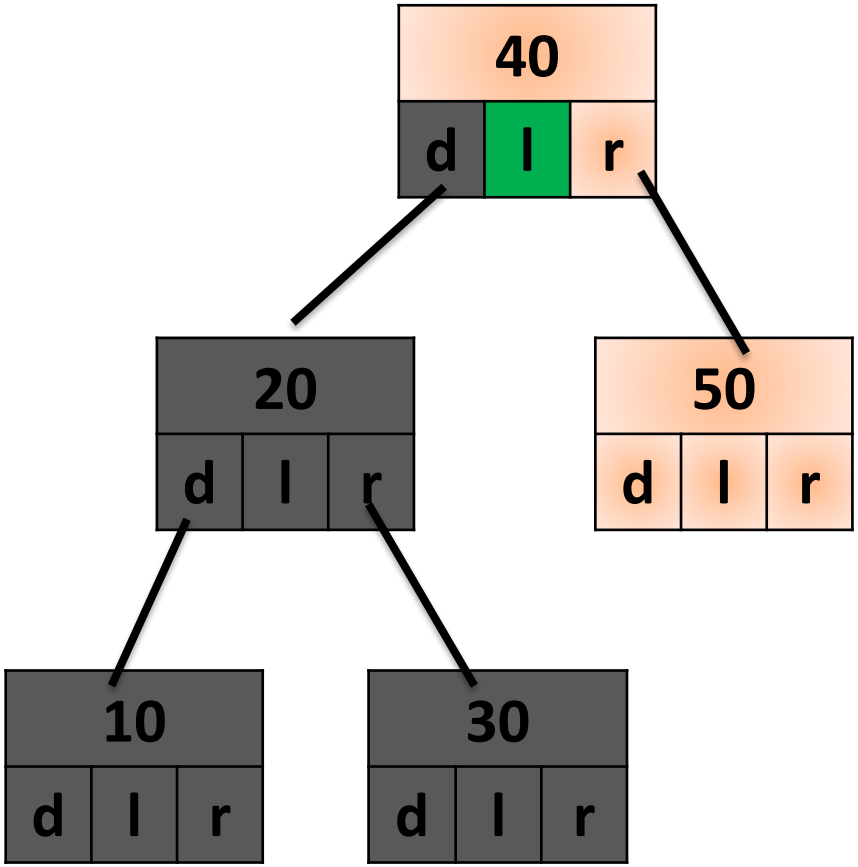
preorder



OUTPUT

40	20	10	30	

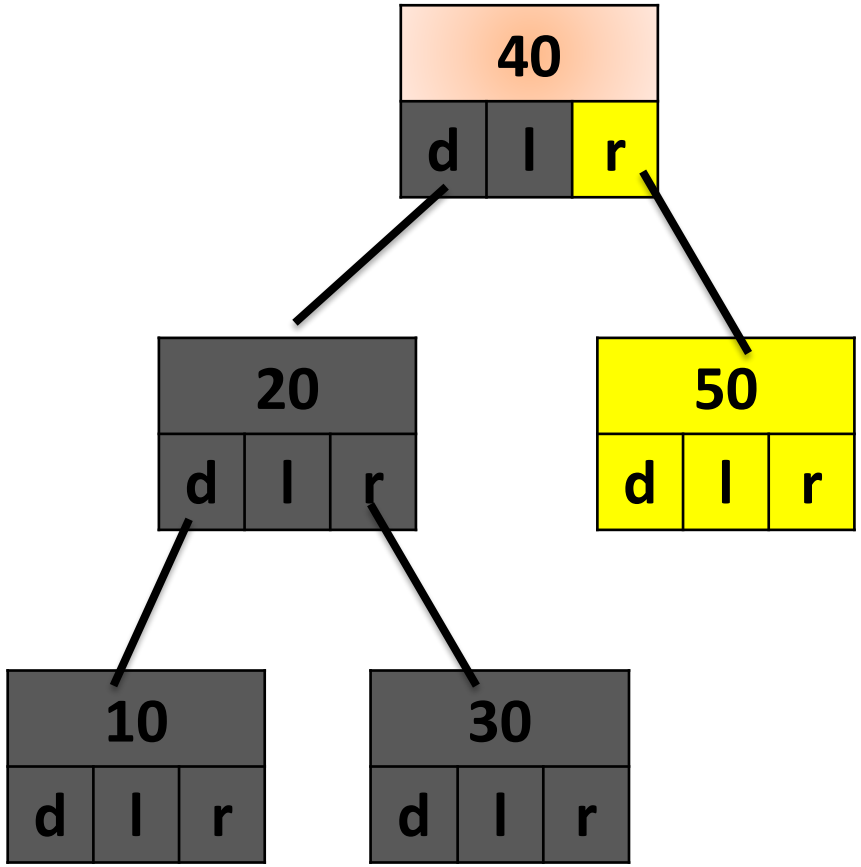
preorder



OUTPUT

40	20	10	30	

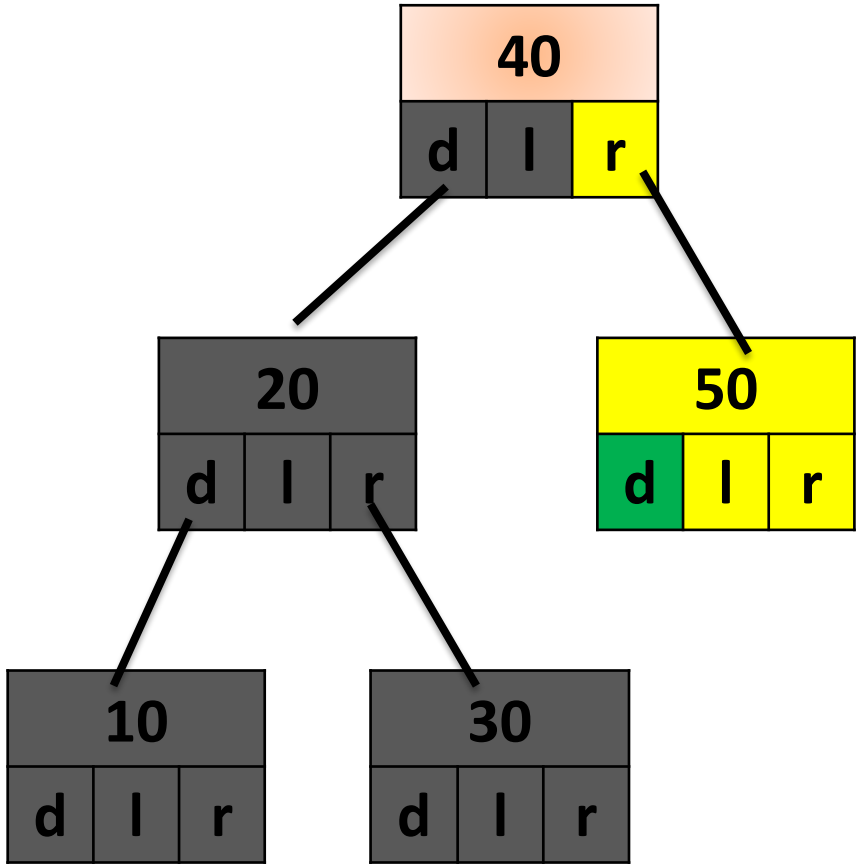
preorder



OUTPUT

40	20	10	30	

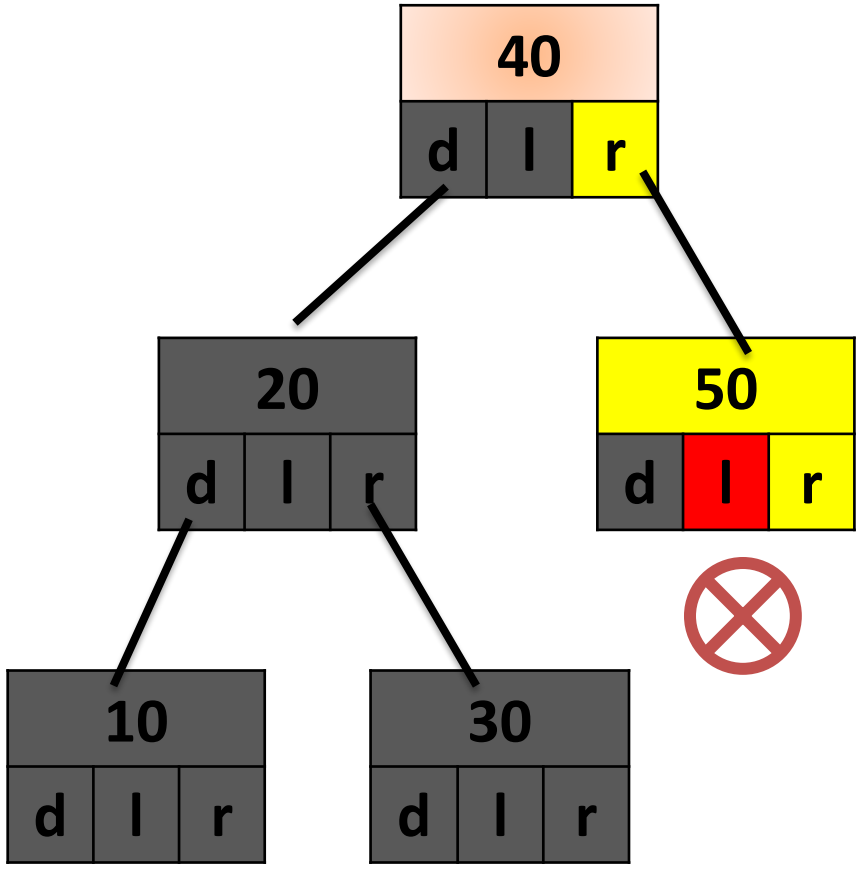
preorder



OUTPUT

40	20	10	30	50

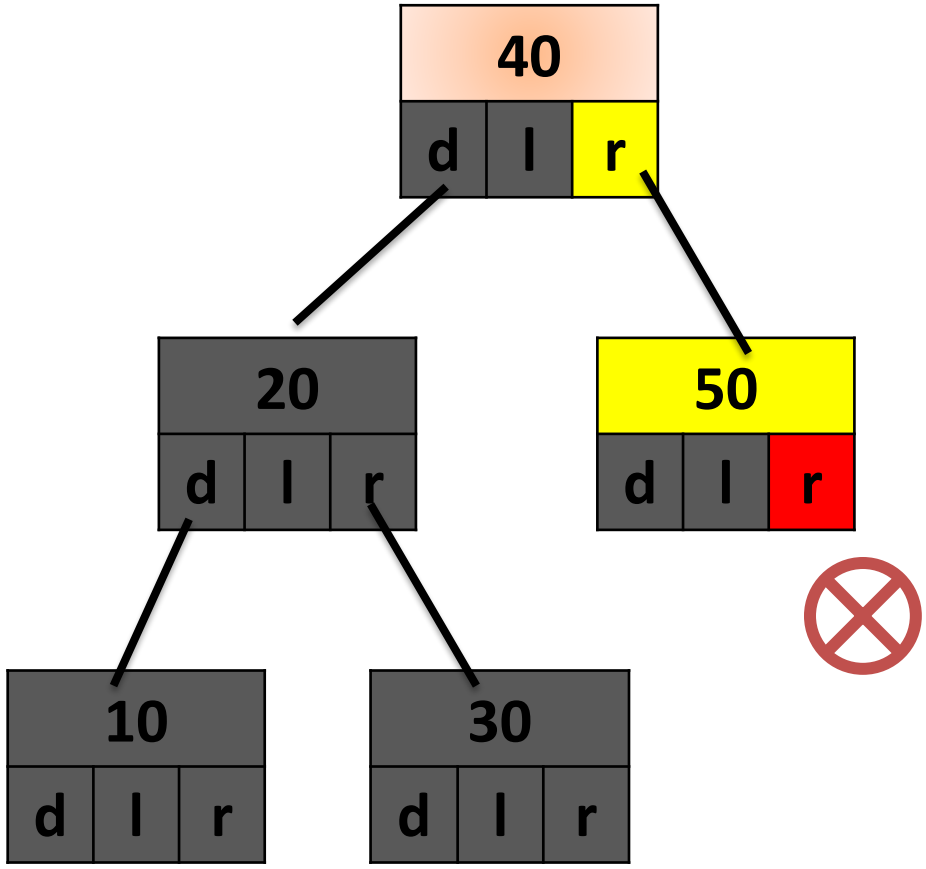
preorder



OUTPUT

40	20	10	30	50

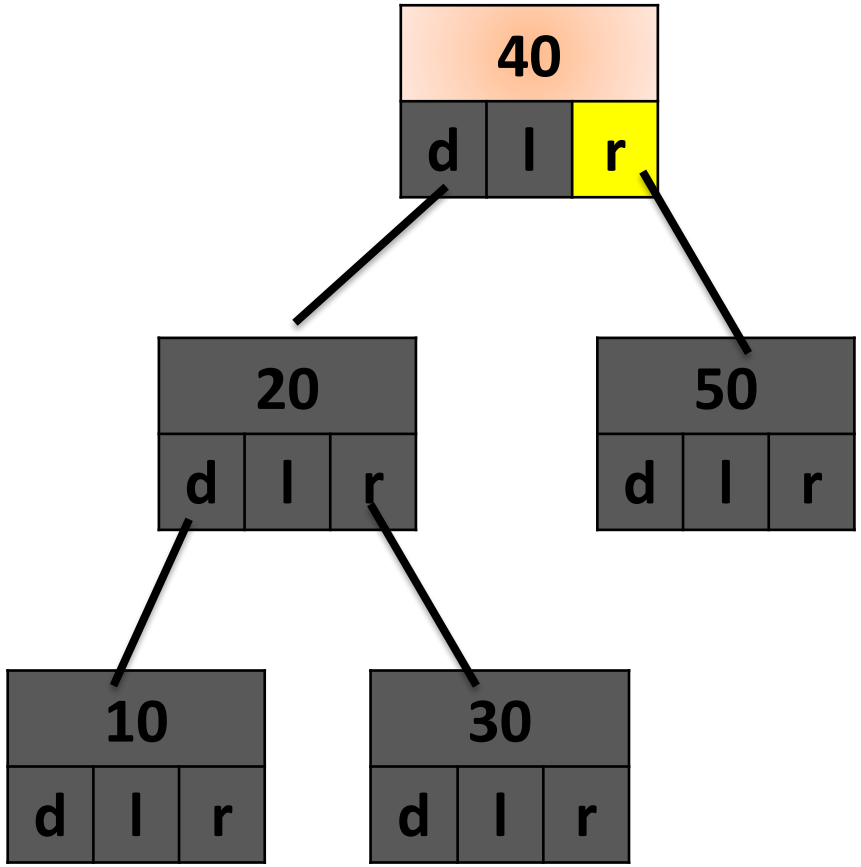
preorder



OUTPUT

40	20	10	30	50

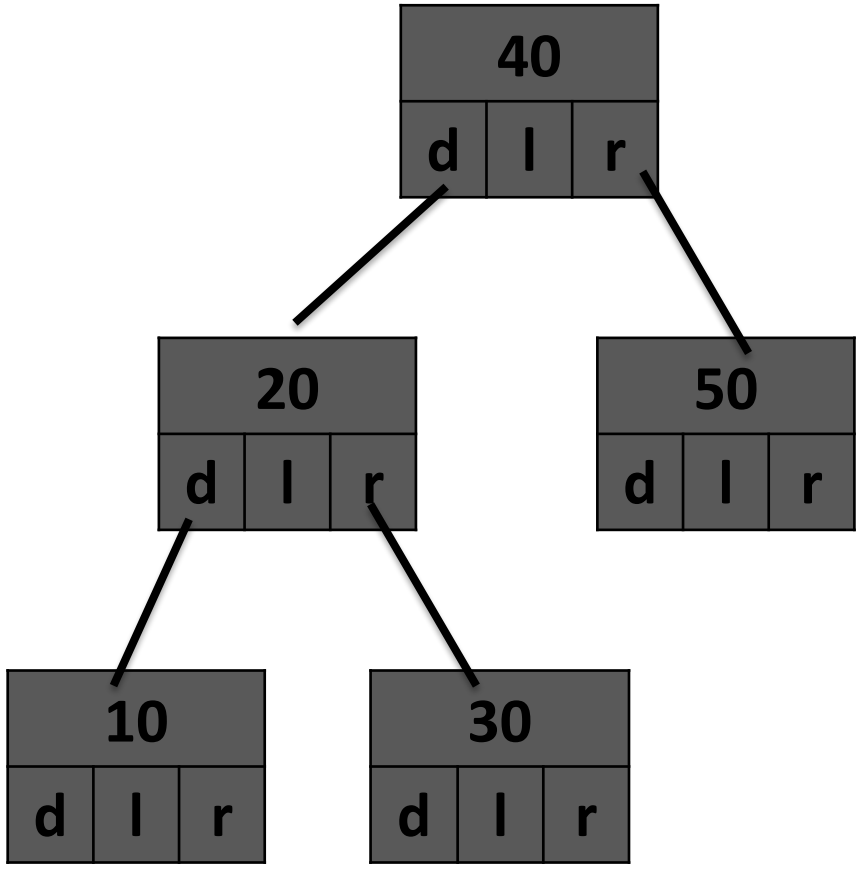
preorder



OUTPUT

40	20	10	30	50

preorder

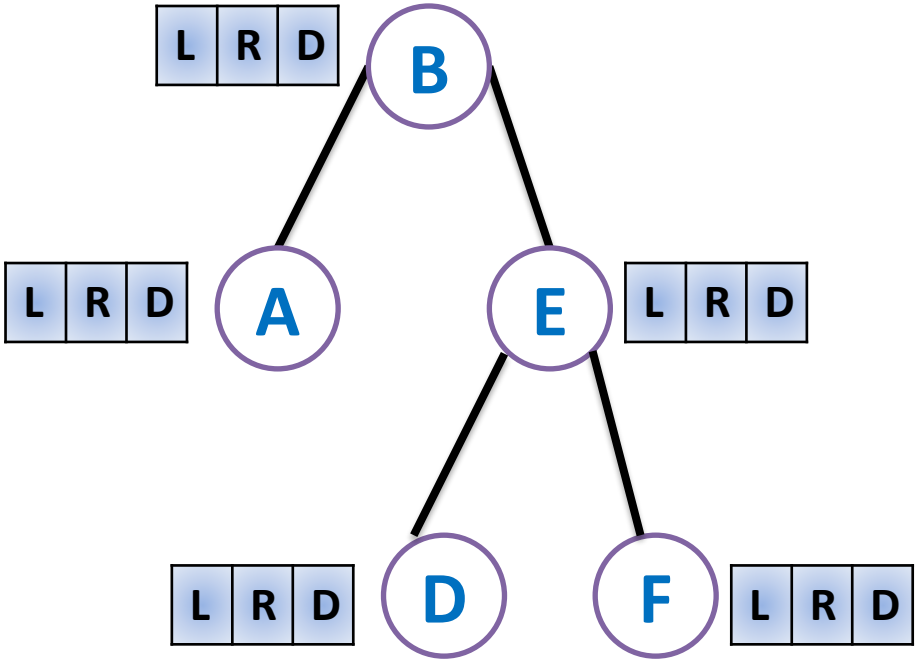


OUTPUT

40	20	10	30	50

Postorder traversal

1. Traverse the left subtree
2. Traverse the right subtree
3. Print data.



OUTPUT

A	D	F	E	B

Find preorder & postorder traversal sequence of this tree.