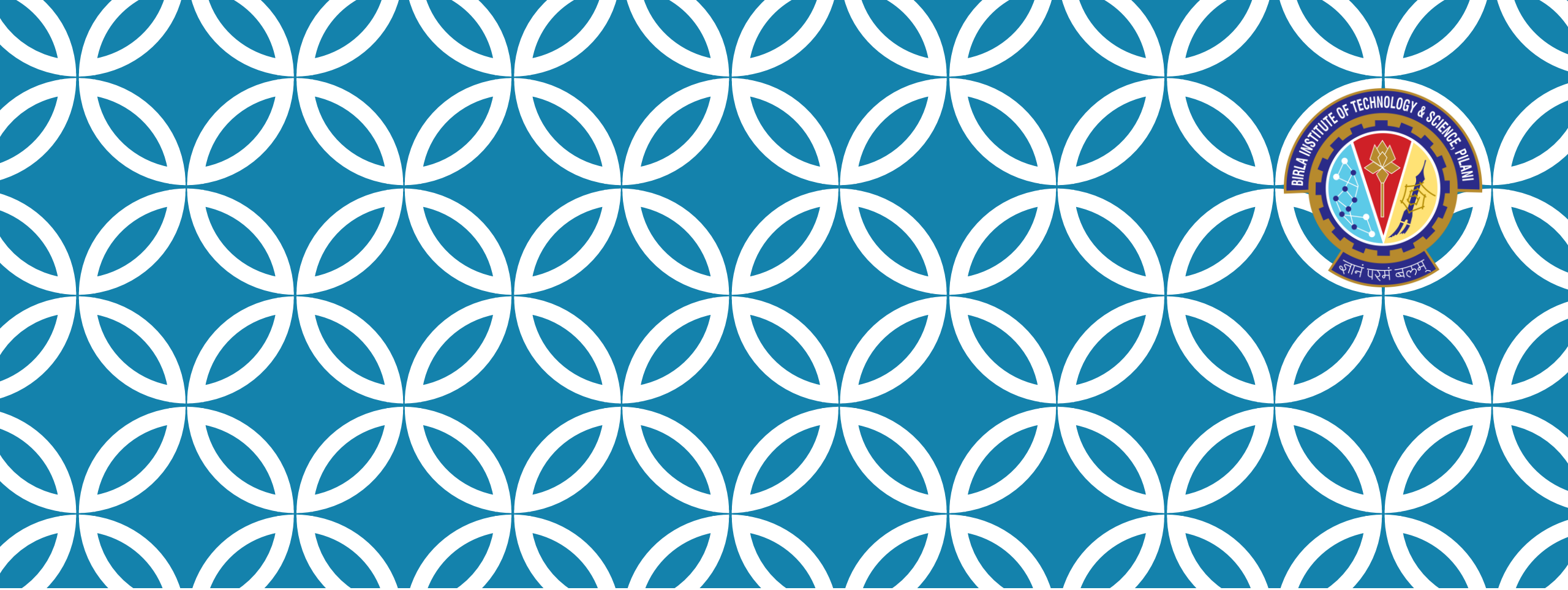
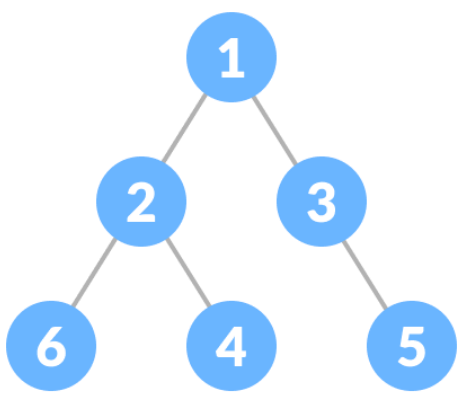


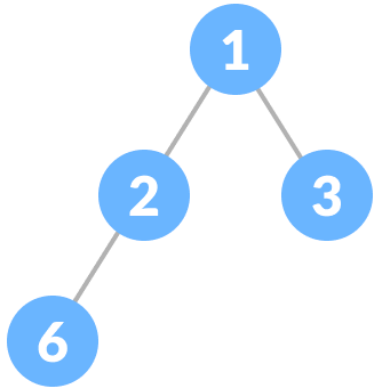
BITS F232: FOUNDATIONS OF DATA STRUCTURES & ALGORITHMS (1ST SEMESTER 2023-24) TREE ADT CONTINUED…

Chittaranjan Hota, PhD
Sr. Professor of Computer Sc.
BITS-Pilani Hyderabad Campus
hota[AT]hyderabad.bits-pilani.ac.in

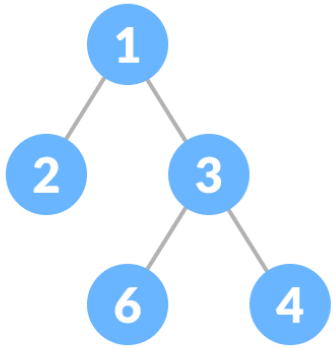
FULL VS COMPLETE: RECAP



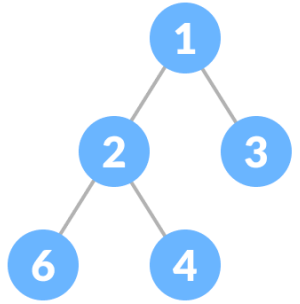
Full (X), or complete (X)



Full (X), or complete (Y)



Full (Y), or complete (N)



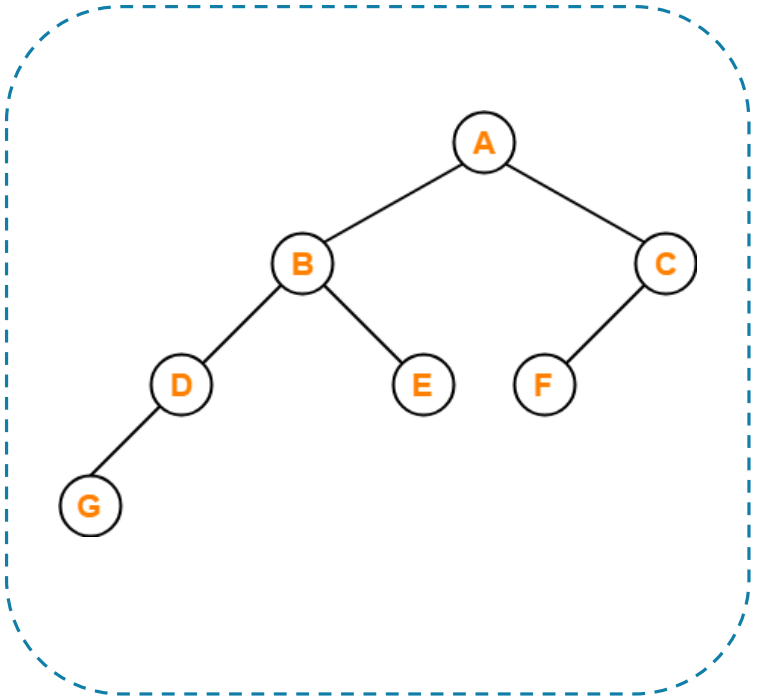
Full (Y), or complete (Y)

PROPERTIES OF A BINARY TREE

Notation:

n: number of nodes, ***e:*** number of external nodes, ***i:*** number of internal nodes, and ***h:*** height

- Minimum number of nodes in a binary tree with height h = h + 1
- Maximum number of nodes in a binary tree with height h = 2^{h+1} - 1
  - Let us draw the tree for h = 2.
- Total number of leaf nodes in a binary tree = total number of nodes with 2 children + 1
- Maximum number of nodes at any level 'l' in a binary tree = 2^l.
  - Let us draw it for level 2.

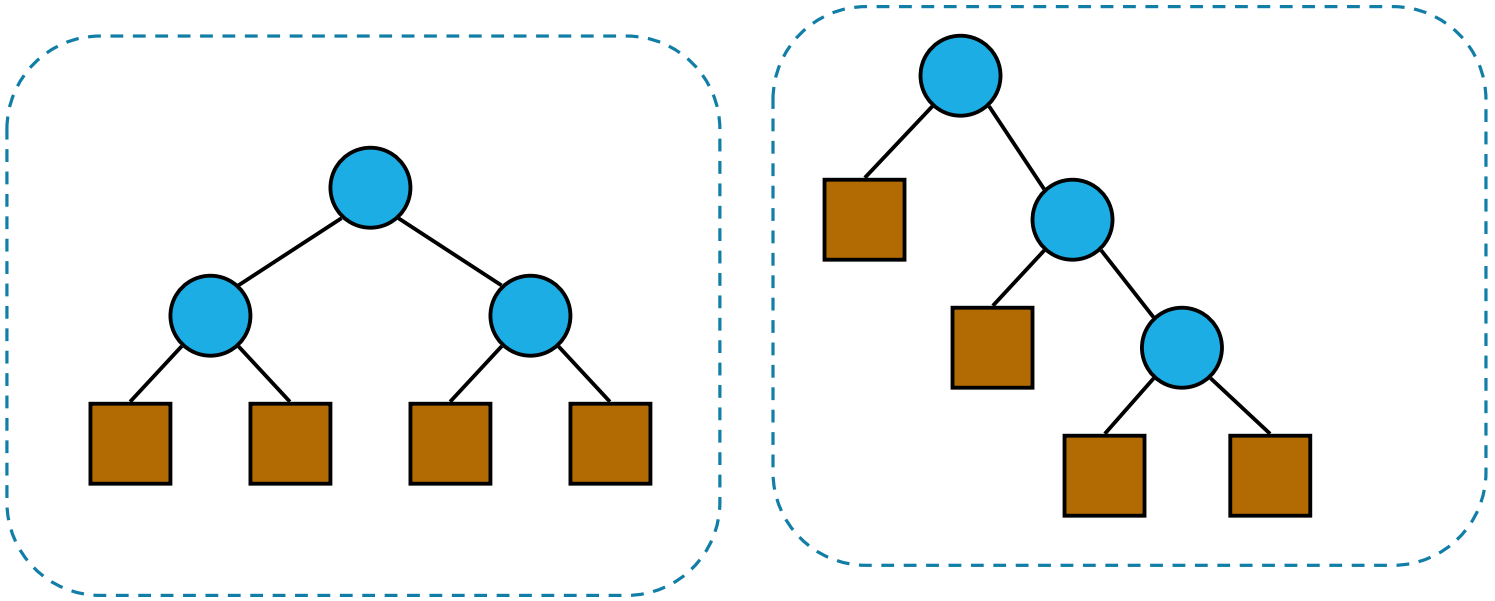


PROPERTIES OF A PROPER BINARY TREE

Notation:

n: number of nodes, ***e:*** number of external nodes, ***i:*** number of internal nodes, and ***h:*** height

- Properties:
  - $e = i + 1$
  - $n = 2e - 1$
  - $h \leq i$
  - $h \leq (n - 1)/2$
  - $e \leq 2^h$
  - $h \geq \log_2 e$
  - $h \geq \log_2 (n + 1) - 1$

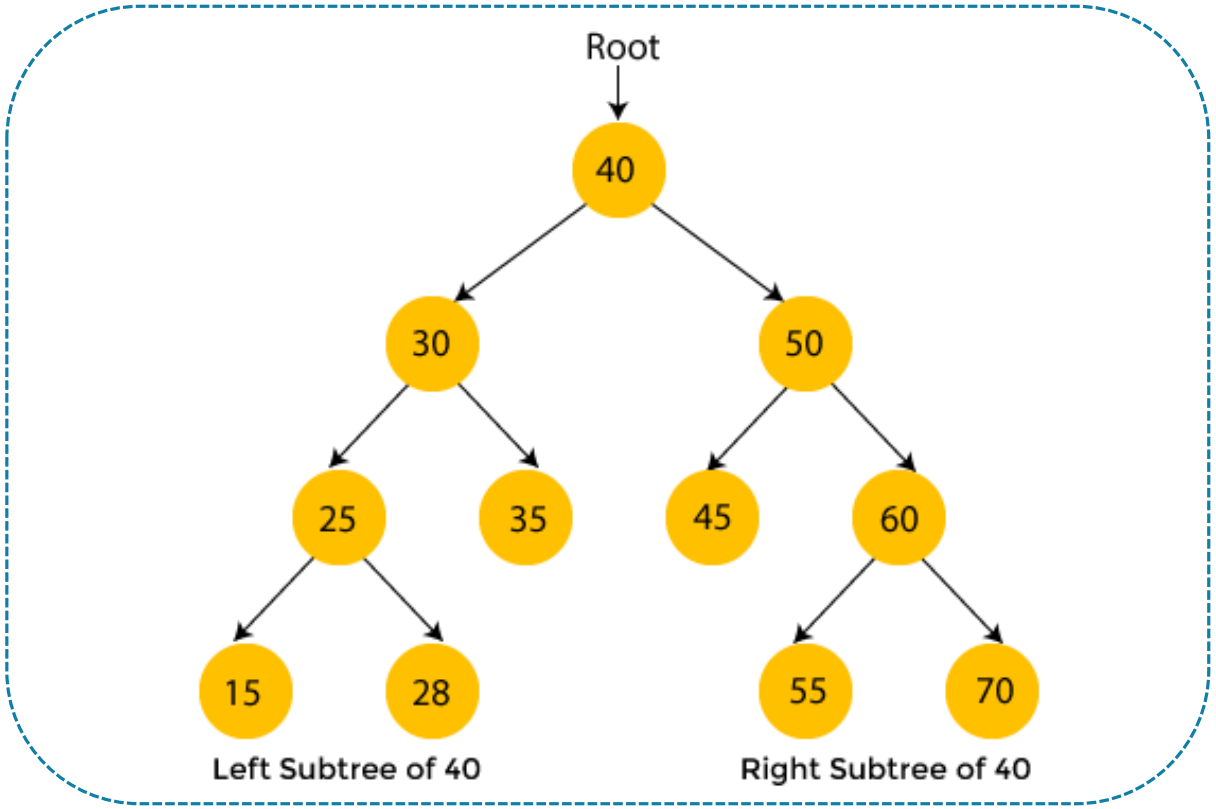


INORDER TRAVERSAL

- In an inorder traversal a node is visited after its left subtree and before its right subtree

Application: draw a binary tree

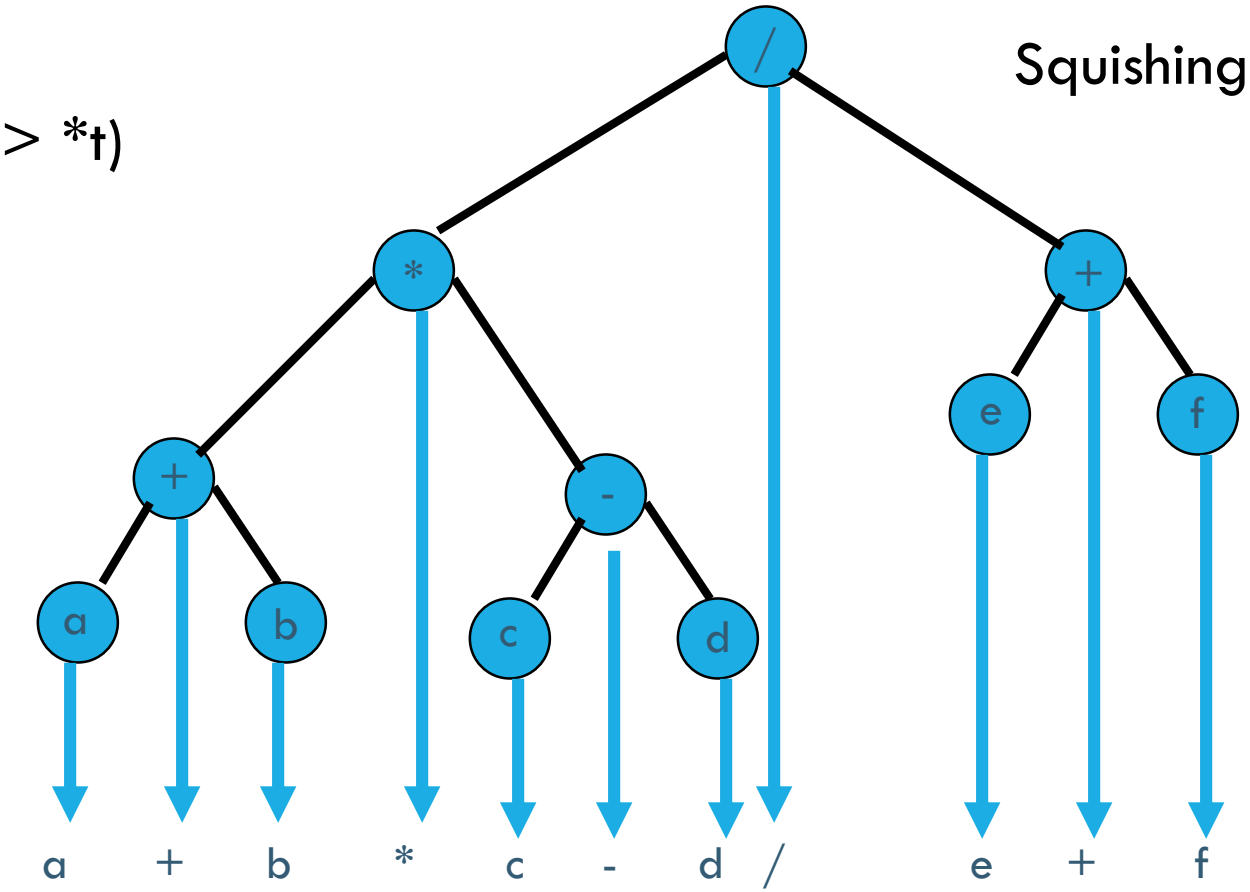
```
Algorithm inOrder(v)
    if ¬ v.isExternal()
        inOrder(v.left())
    visit(v)
    if ¬ v.isExternal()
        inOrder(v.right())
```



BINARY TREE TRAVERSAL: INORDER

```
template <class T>
void inOrder(binaryTreeNode<T> *t)
{
   if (t != NULL)
   {
      inOrder(t->leftChild);
      visit(t);
      inOrder(t->rightChild);
   }
}
```

Lab: after midterm (Lab no:8)



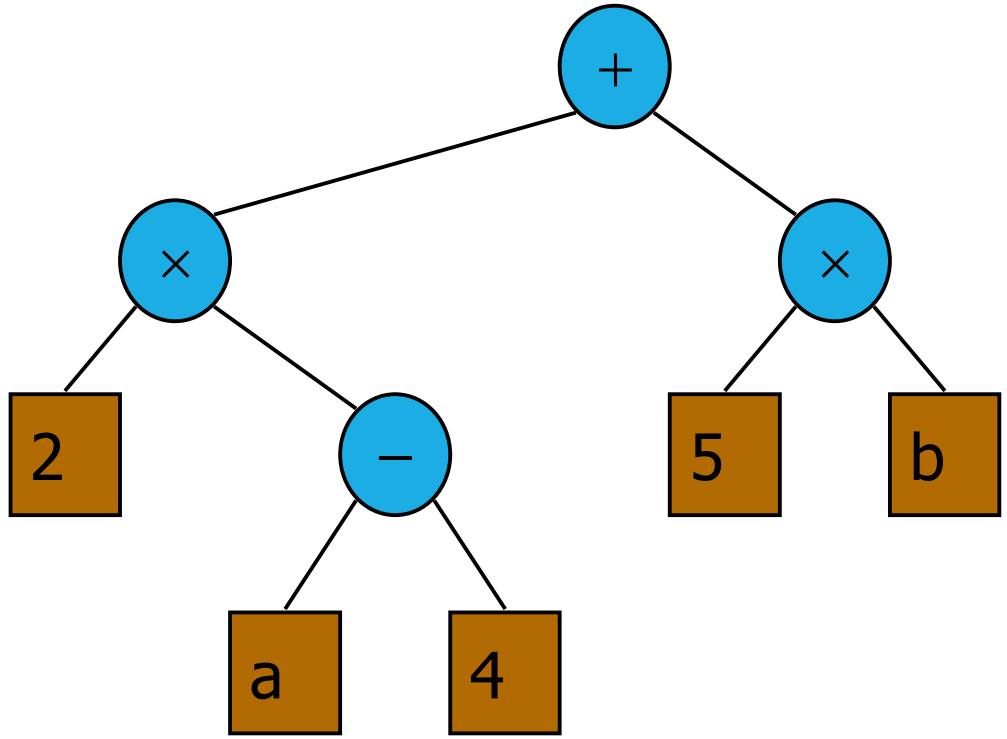
Gives infix form of expression!

PRINT ARITHMETIC EXPRESSION

Specialization of an inorder traversal

- print operand or operator when visiting node
- print "(" before traversing left subtree
- print ")" after traversing right subtree

```
Algorithm printExpression(v)
    if ¬v.isExternal()
        print("(")
        inOrder(v.left())
    print(v.element())
    if ¬v.isExternal()
        inOrder(v.right())
        print (")")
```

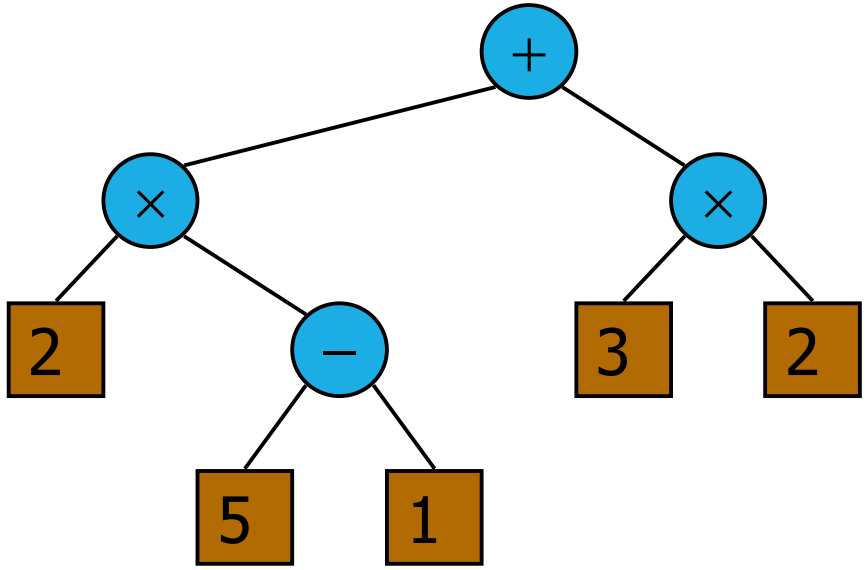


Let us write it out…

EVALUATE ARITHMETIC EXPRESSION

Specialization of a postorder traversal

- recursive method returning the value of a subtree
- when visiting an internal node, combine the values of the subtrees

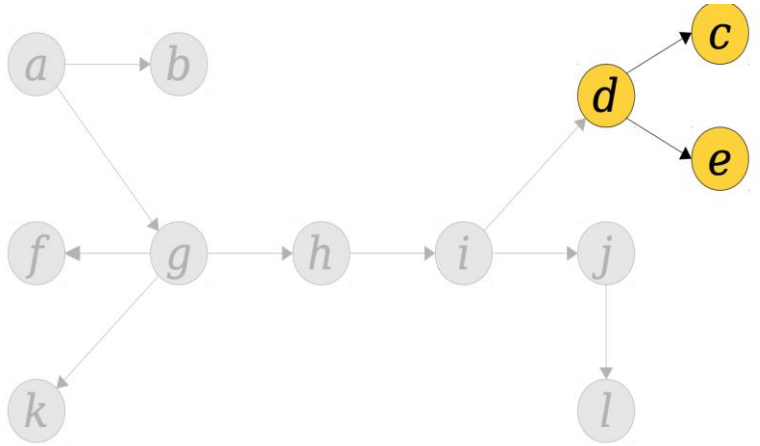



```
Algorithm evalExpr(v)
    if v.isExternal()
        return v.element()
    else
        x ← evalExpr(v.left())
        y ← evalExpr(v.right())
        ◊ ← operator stored at v
        return x ◊ y
```

EULER TOUR TRAVERSAL

- Generic traversal of a binary tree
- We can **unify** the tree-traversal algorithms (in-order, pre-order, and post-order) into a single framework by relaxing the requirement that each node be visited exactly once.
- Walk around the tree and visit each node three times:
  - on the left (preorder)
  - from below (inorder)
  - on the right (postorder)

+: It allows for more general kinds of algorithms to be expressed easily.



a b a g h i **d c d e d** i j l j i h g f g k g a

```
template <typename E, typename R>                // do the tour
int EulerTour<E, R>::eulerTour(const Position& p) const {
  Result r = initResult();
  if (p.isExternal()) {                          // external node
    visitExternal(p, r);
  }
  else {                                         // internal node
    visitLeft(p, r);
    r.leftResult = eulerTour(p.left());          // recurse on left
    visitBelow(p, r);
    r.rightResult = eulerTour(p.right());        // recurse on right
    visitRight(p, r);
  }
  return result(r);
}
```

Applications: finding no. of descendants, level of each node, lowest common ancestor, etc.

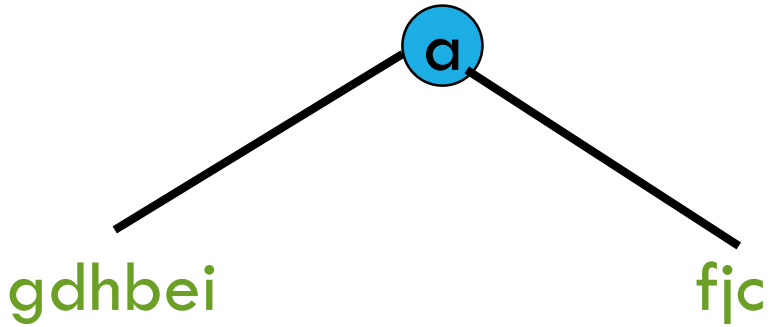
BINARY TREE CONSTRUCTION FROM TRAVERSAL ORDER

Can you construct the binary tree, given two traversal sequences?

preorder = ab

postorder = ba

Try:
Postorder: DEBFCA
Inorder: DBEAFC

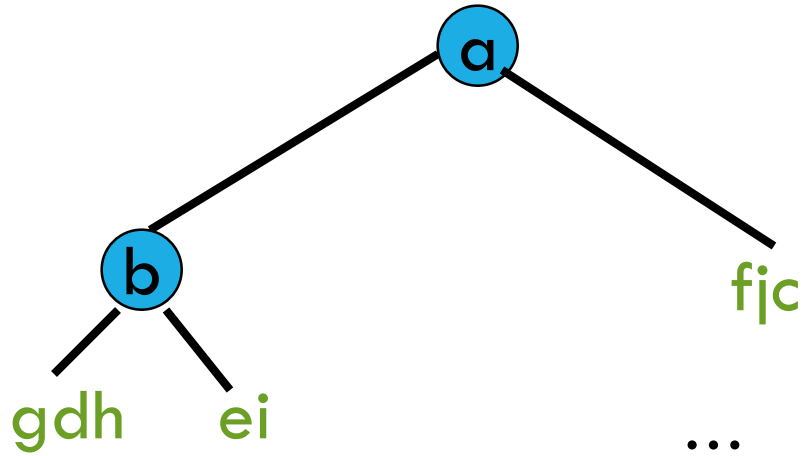


do they uniquely define a binary tree?

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

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

preorder = b d g h e i c f j



Similarly, Scan postorder from right to left using inorder.

…

BINARY TREE ADT

p.left(): Return the left child of p; an error condition occurs if p is an external node.

p.right(): Return the right child of p; an error condition occurs if p is an external node.

p.parent(): Return the parent of p; an error occurs if p is the root.

p.isRoot(): Return true if p is the root and false otherwise.

p.isExternal(): Return true if p is external and false otherwise.

size(): Return the number of nodes in the tree.

empty(): Return true if the tree is empty and false otherwise.

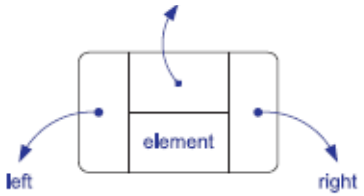
root(): Return a position for the tree's root; an error occurs if the tree is empty.

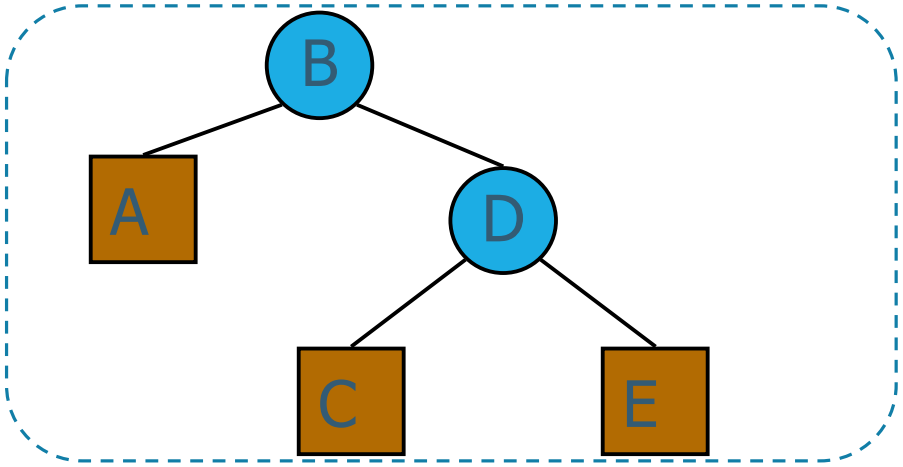
positions(): Return a position list of all the nodes of the tree.

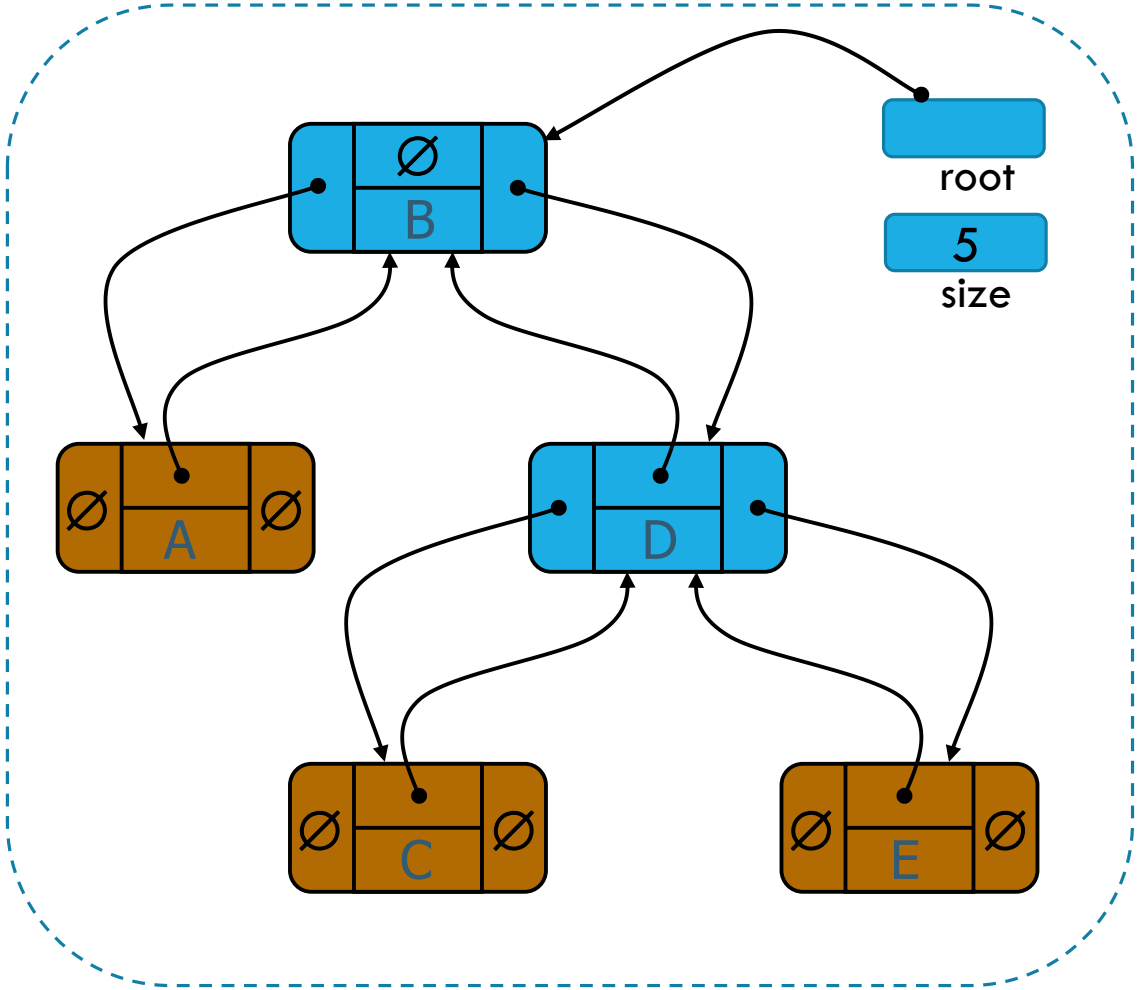
```
template <typename E>
class Position<E> {
public:
  E& operator*();
  Position left() const;
  Position right() const;
  Position parent() const;
  bool isRoot() const;
  bool isExternal() const;
};

template <typename E>
class BinaryTree<E> {
public:
  class Position;
  class PositionList;
public:
  int size() const;
  bool empty() const;
  Position root() const;
  PositionList positions() const;
};
```

LINKED STRUCTURE FOR BINARY TREES

```
struct Node{
    Node *left;
    Node *right;
    Node *par;
    int data;
    Node() : data(), par(NULL), left(NULL), right(NULL) { }
};
```

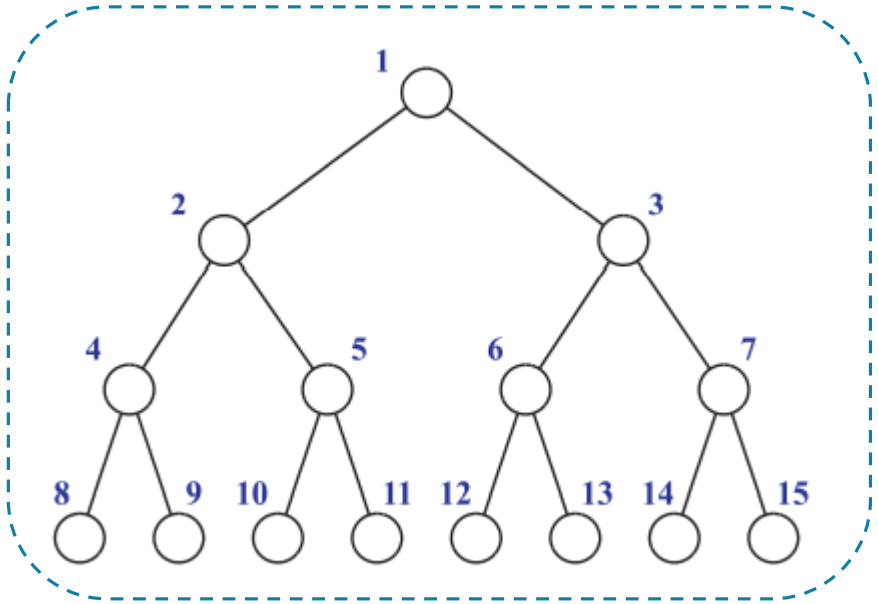




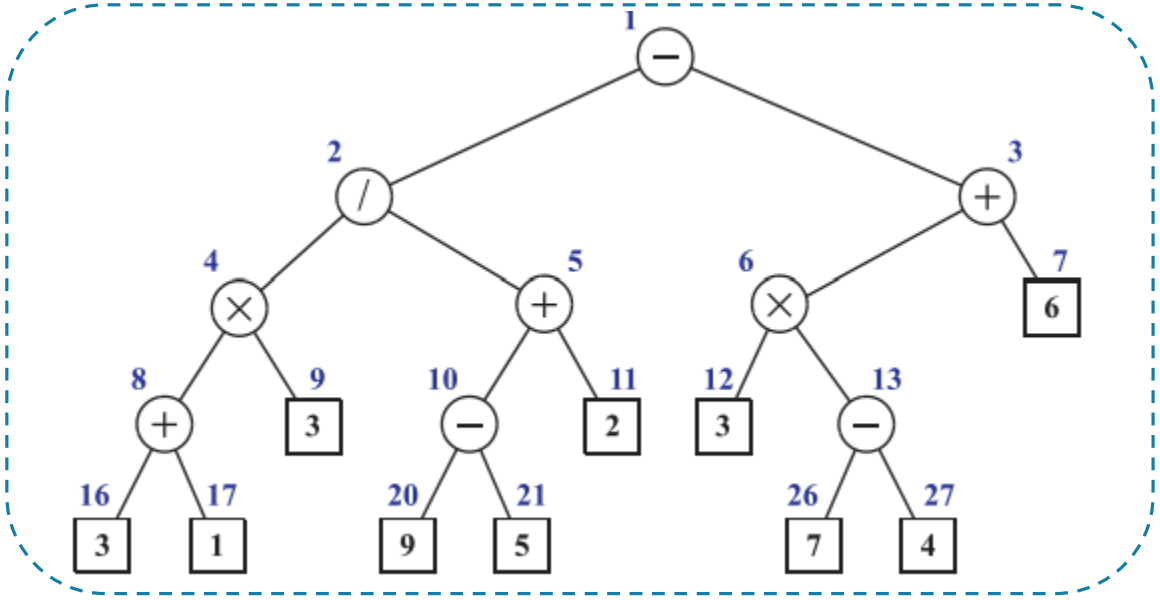


A VECTOR-BASED IMPLEMENTATION OF BINARY TREE

- A simple structure for representing a binary tree T is based on a way of numbering the nodes of T.
- If v is the root of T, then f(v) = 1; If v is the left child of node u, then f(v) = 2f(u); If v is the right child of node u, then f(v) = 2f(u) +1 ➔ function f is called level numbering function.

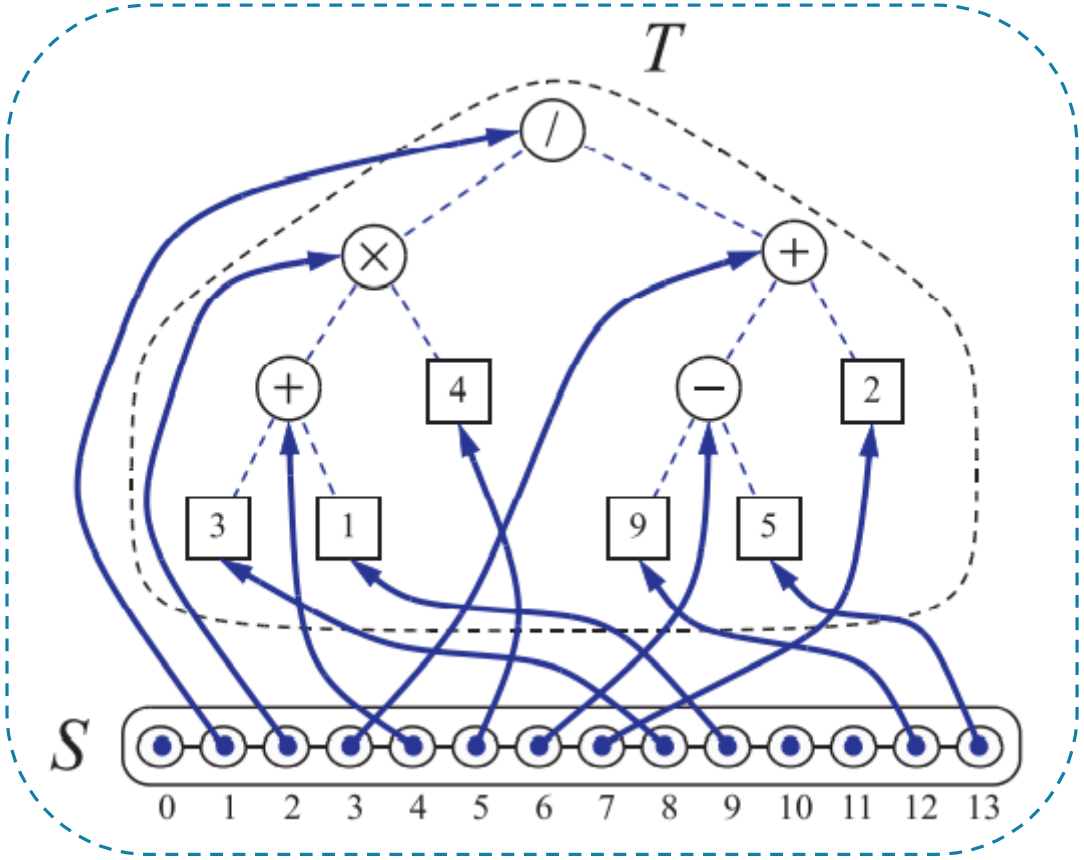


(General scheme)



(An Example Expression tree)

CONTINUED...



(Example binary tree using a vector)

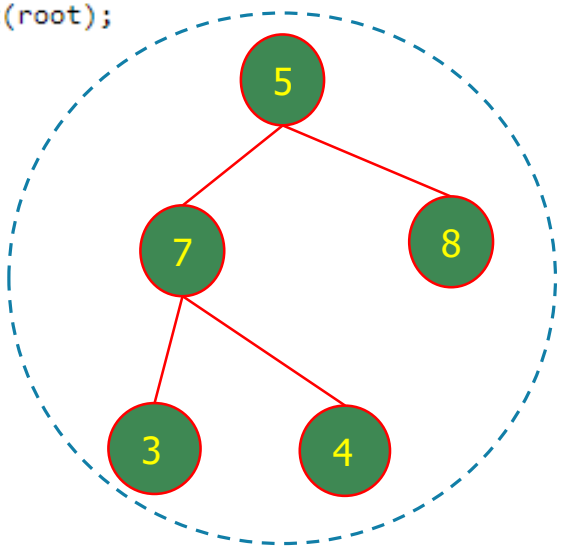
Operation	*Time*
left, right, parent, isExternal, isRoot	$O(1)$
size, empty	$O(1)$
root	$O(1)$
expandExternal, removeAboveExternal	$O(1)$
positions	$O(n)$

The vector implementation of a binary tree is a fast and easy way of realizing the binary-tree ADT, but it can be very space inefficient if the height of the tree is large. ➔ $O(2^n)$, where 'n' is no. of nodes in T.

BINARY TREE IMPLEMENTATION USING LINKED STRUCT

```
77  void BinaryTree::inorder(Node *ptr){
78      if(!ptr) return;
79      inorder(ptr->left);
80      cout<<" "<<ptr->data;
81      inorder(ptr->right);
82  }
83  void BinaryTree::postorder(Node *ptr){
84      if(!ptr) return;
85      postorder(ptr->left);
86      postorder(ptr->right);
87      cout<<" "<<ptr->data;
88  }
89  void BinaryTree::preorder(Node *ptr){
90      if(!ptr) return;
91      cout<<ptr->data<<" ";
92      preorder(ptr->left);
93      preorder(ptr->right);
94  }
```

```
98  Node* BinaryTree::createTree(vector<int> &v,Node *root,
99                               Node *parent,int i){
100     n = v.size();
101     if(i<v.size()){
102         Node *temp = new Node;
103         temp->data = v[i];
104         temp->par = parent;
105         root = temp;
106         root->left = createTree(v,root->left,root,2*i+1);
107         root->right = createTree(v,root->right,root,2*i+2);
108         all_nodes.insert(root);
109     }
110     main_root = root;
111     return root;
112 }
```




```
Enter size of input array : 6
Enter array : 5 7 8 3 4 9
1
Size : 6
2
Tree is not empty
3
Inorder traversal :  3 7 4 5 9 8
4
Preorder traversal : 5 7 3 4 8 9
5
Postorder traversal :  3 4 7 9 8 5
6
Height by height1 : 2
7
Height by height2 : 2
```

```
Enter size of input array : 5
Enter array : 5 7 8 3 4
1
Size : 5
2
Tree is not empty
3
Inorder traversal :  3 7 4 5 8
4
Preorder traversal : 5 7 3 4 8
5
Postorder traversal :  3 4 7 8 5
6
Height by height1 : 2
7
Height by height2 : 2
```