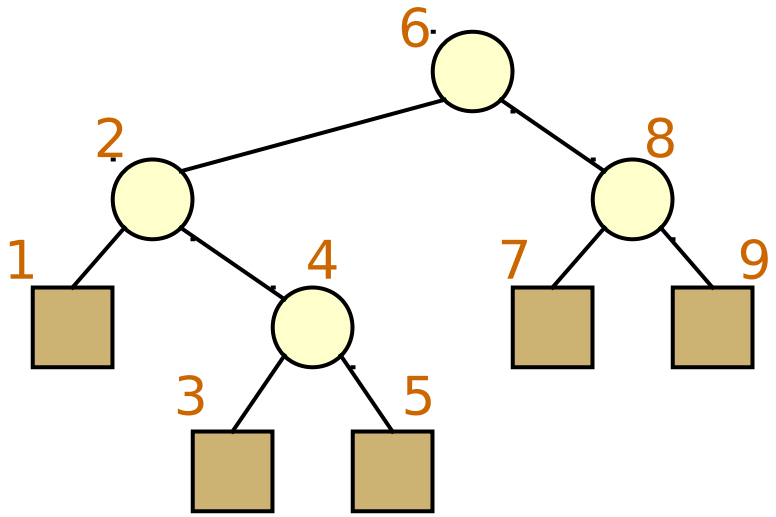
Binary Tree Traversals

- Let l, R, and r stand for moving left, visiting the node, and moving right.
- There are six possible combinations of traversal
  - lRr, lrR, Rlr, Rrl, rRl, rlR
- Adopt convention that we traverse left before right, only 3 traversals remain
  - l**R**r, lr**R**, **R**lr
  - in**order**, post**order**, pre**order**

Inorder Traversal

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



Algorithm ***inOrder(v)***

```
if isInternal (v)
    inOrder (leftChild (v))
visit(v)
if isInternal (v)
    inOrder (rightChild (v))
```

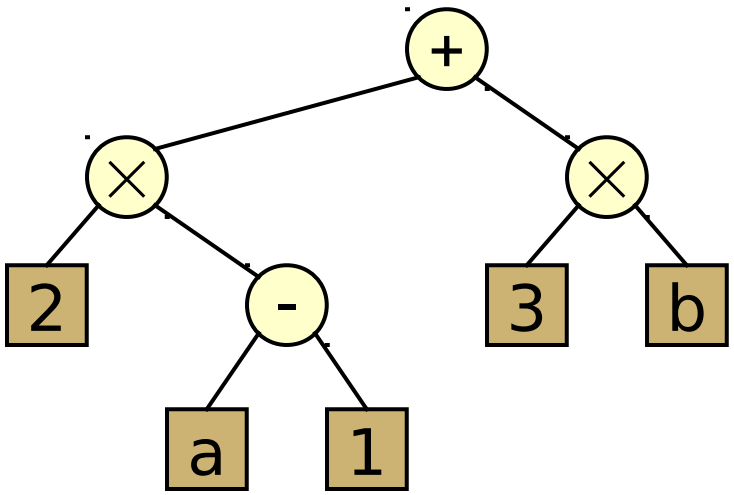
If the left subtree is non empty, do a inorder traversal on it.

Visit the root

If the right subtree is non empty, do a inorder traversal on it.

Print Arithmetic Expressions

- Specialization of an inorder traversal
  - print operand or operator when visiting node
  - print "(" before traversing left subtree
  - print ")" after traversing right subtree



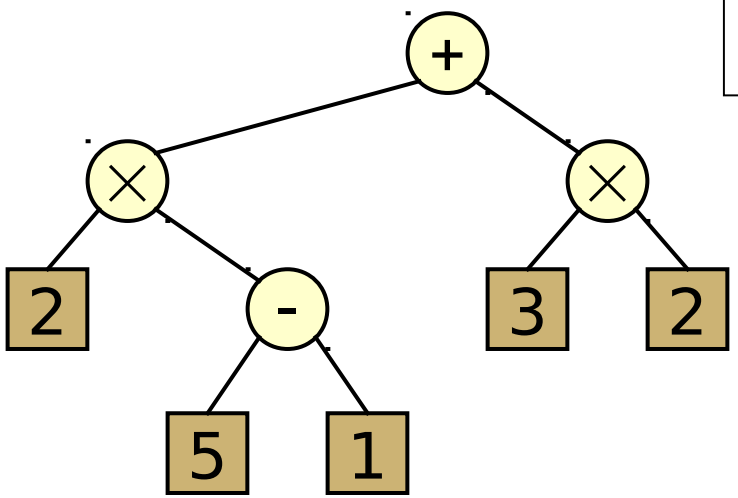
Algorithm ***inOrder*** **(*v*)**

```
if isInternal (v){
    print("(")
    inOrder (leftChild (v))}
print(v.element ())
if isInternal (v){
    inOrder (rightChild (v))
    print (")")}
```

((2 × (a - 1)) + (3 × b))

Evaluate Arithmetic Expressions

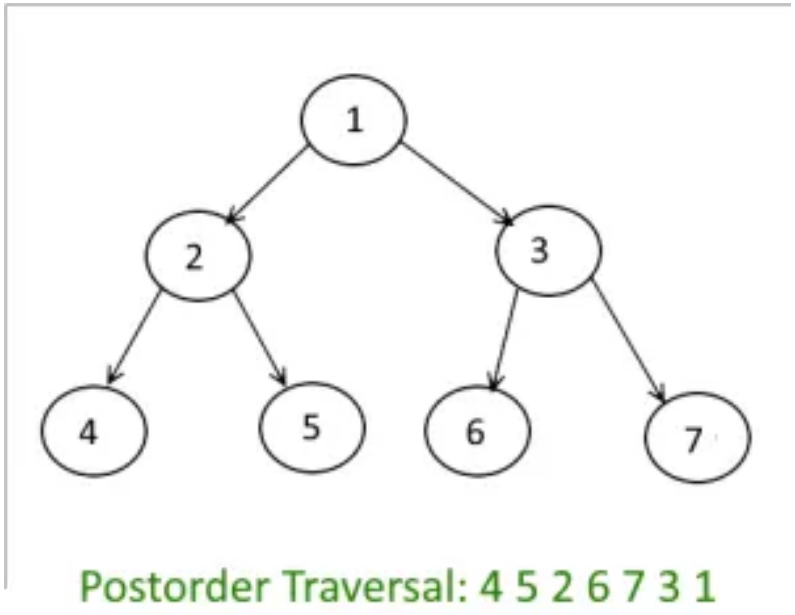
- Recursive method returning the value of a subtree
- When visiting an internal node, combine the values of the subtrees



Algorithm *evalExpr*(*v*)
    if *isExternal* (*v*)
        return *v.element* ()
    else
        x ← *evalExpr*(*leftChild* (*v*))
        y ← *evalExpr*(*rightChild* (*v*))
        ◊ ← operator stored at *v*
        return *x* ◊ *y*

Postorder Traversal

- In Postorder traversal left subtree is visited first, next right subtree is visited and only after that node is visited.



Algorithm Post***Order(v)***

if ***isInternal*** **(*v*)**

postOrder **(*leftChild*** **(*v*))**

postOrder (right***Child*** **(*v*))**

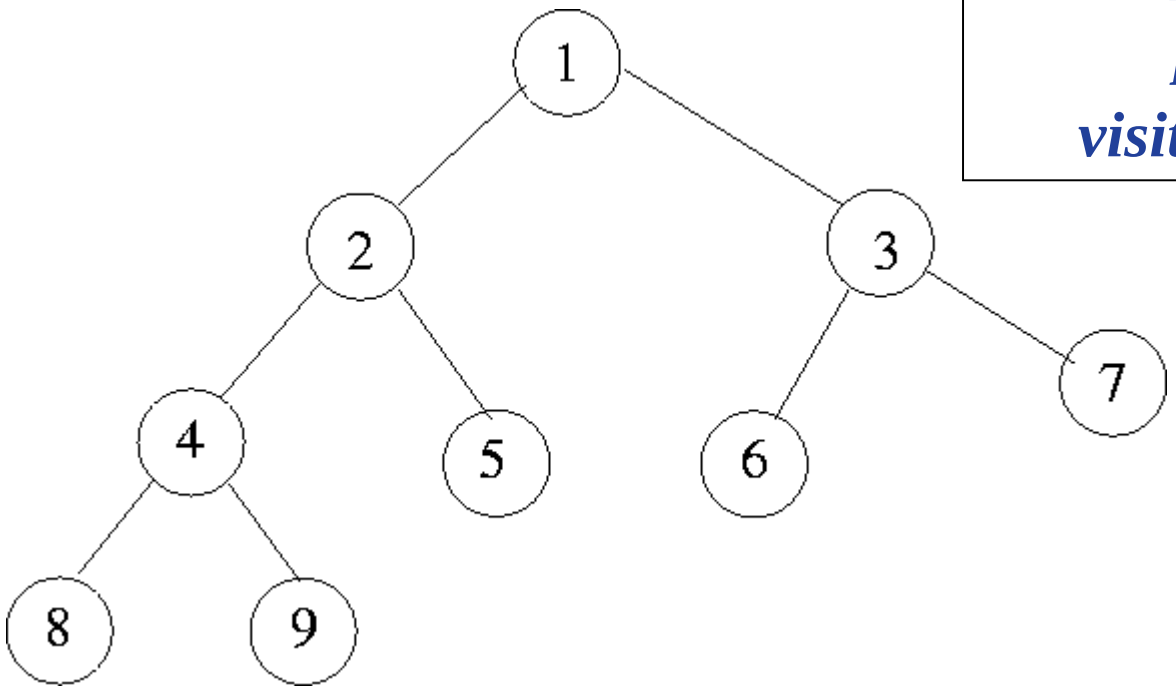
visit**(*v*)**

If the left subtree is non empty, do a postorder traversal on it.

If the right subtree is non empty, do a postorder traversal on it.

Visit the root

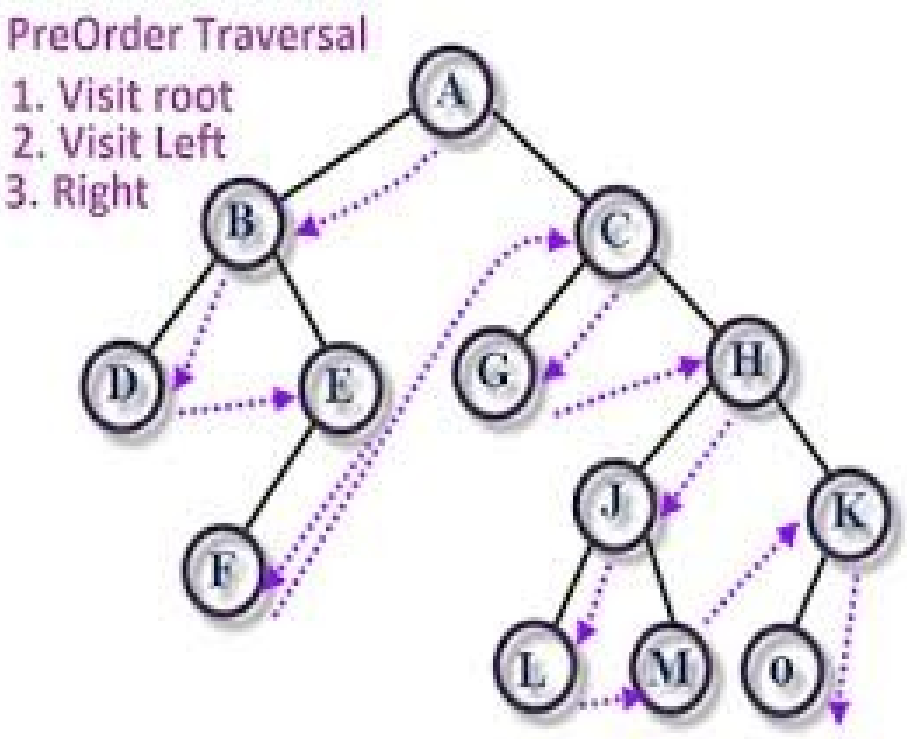
Postorder Traversal




```
Algorithm PostOrder(v)
    if isInternal (v)
        postOrder (leftChild (v))
        postOrder (rightChild (v))
    visit(v)
```

Preorder Traversal

- In Preorder traversal, first visit node, process left subtree then process right subtree.



Algorithm Pre*Order*(*v*)

visit (*v*)

PreOrder (*leftChild* (*v*))

PreOrder (**right***Child* (*v*))

Visit the node

If the left subtree is non empty, do a preorder traversal on it.

If the right subtree is non empty, do a preorder traversal on it.

Preorder Traversal

Algorithm Pre***Order(v)***

visit **(*v*)**

PreOrder **(*leftChild*** **(*v*))**

PreOrder **(right*Child*** **(*v*))**