**KATHMANDU UNIVERSITY**

**Department of Computer Science and Engineering**

**Dhulikhel, Kavre**

****

**Mini Project on**

**Binary Tree Using Post-order and In order Traversal**

**Algorithm and Complexity**

**[Course Code: COMP 314]**

**Submitted by:**

Sajag Silwal (Roll no: 48)

CE(3rd Year/ 2nd Semester)

**Submitted to:**

Dr. Rajani Chulyadyo

Department of Computer Science and Engineering

# Introduction

A binary tree is a non-linear tree-type data structure with a maximum of two children for each node or parent. Since each node of a binary tree can have a maximum of 2 children, we basically name them left child and right child. The example

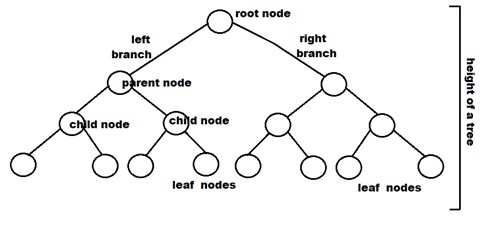


Figure 1: Binary Tree Diagram

## Post Order and In Order Traversal for constructing a binary tree

The Order of post-order in constructing a binary tree is

Left 🡪 Right 🡪 Root

The order of post-order in constructing a binary tree is:

Left 🡪 Root 🡪 Right

The approach for constructing a binary tree using Post-order and In-order traversal are:

1. Find the root node using post-order traversal.
2. Find the left subtree and the right subtree using in-order traversal by finding the index of the root node of the corresponding subtrees.
3. Once the root node is found, the recursion can be implemented on right and left subtree. The right and left subarray split at respective root index to repeat the same process until a single element is in either sub-array.

## Example:

The example of implementation of post-order and in-order to construct a binary tree is given below:

**Input:**  
   
In-order Traversal : { 4, 2, 1, 7, 5, 8, 3, 6 }  
Post-order Traversal : { 4, 2, 7, 8, 5, 6, 3, 1 }

Output: Binary Tree given below

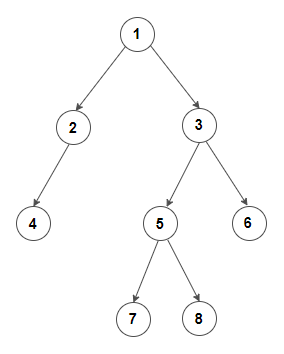


Figure 2: Binary Tree Using Post Order and In Order

For the root node, Post-Order Traversal is considered (i.e. ‘1’ is taken as a root element). Similarly, the recursive process is continued in the bit sides. The Input set of nodes should be in traversed in the form of Post-Order and In-Order.

## Algorithm

The pseudocode for construction of binary tree using post-order and in-order traversal are:

1. If startIndex > endIndex or postIndex < 0
2. Return NIL
3. End if
4. RootIndex 🡨 findIndex(inOrder, postOrder[postIndex])
5. Root 🡨 new Node(inOrder[rootIndex])
6. Root.right 🡨 constructBinaryTree(inOrder, postOrder, rootIndex+1, endIndex)
7. Root.left 🡨 constructBinaryTree(inOrder, postOrder, startIndex, rootIndex-1)
8. Return Root
9. End

## Time Complexity:

The time complexity of the Binary tree using post-order and in-order traversal is O(n2).

## Source Code:

The Source code for the building of a binary tree using post-order and in-order traversal is given below:

from prettytable import PrettyTable

class Node:

    def \_\_init\_\_(self, x):

        self.data = x

        self.left = None

        self.right = None

def build(inn, post, start, end):

    global dc, index, lst, lstIndex

    # print(dc)

    if (start > end):

        return None

    curr = post[index]

    node = Node(curr)

    lst[lstIndex] = curr

    lstIndex += 1

    index -= 1

    #If node doesn't have any children

    if (start == end):

        return node

    iIndex = dc[curr]

    node.right = build(inn, post,

                           iIndex + 1, end)

    node.left = build(inn, post, start,

                          iIndex - 1)

    return node

def constructTree(inn, post, lenn):

    global index

    global lst, lstIndex

    for i in range(lenn):

        dc[inn[i]] = i

    # Index in postorder

    index = lenn - 1

    return build(inn, post, 0, lenn - 1)

#preorder Traversal

def preOrder(node):

    if (node == None):

        return

    print(node.data, end = " ")

    preOrder(node.left)

    preOrder(node.right)

#postorder Traversal

def postOrder(node):

    if(node==None):

        return

    postOrder(node.left)

    postOrder(node.right)

    print(node.data, end=" ")

#Inorder Traversal

def inOrder(node):

    if(node==None):

        return

    inOrder(node.left)

    print(node.data, end=" ")

    inOrder(node.right)

# Driver Code

if \_\_name\_\_ == '\_\_main\_\_':

    inn = [ 4, 8, 2, 5, 1, 6, 3, 7 ]

    post = [ 8, 4, 5, 2, 6, 7, 3, 1 ]

    n = len(inn)

    dc, index, lst, lstIndex = {}, 0, {}, 0

    root = constructTree(inn, post, n)

    # Prettytable

    print("\n The index of the node and its correspondig value are: \n")

    tb = PrettyTable()

    tb.field\_names = ["index", "value of node"]

    for i in lst:

        tb.add\_row([i, lst[i]])

    print(tb)

    print("\nThe array in pre order traversal is :\n")

    preOrder(root)

    print("\nThe array in post order traversal is : \n")

    postOrder(root)

    print("\nThe array in in order traversal is : \n")

    inOrder(root)

## Output:

