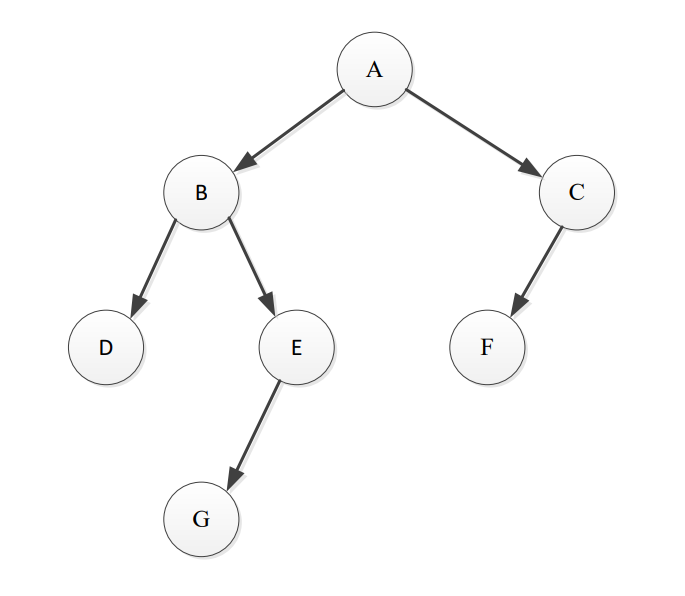
Question 1: Binary Tree Traversal

Node[] order;



Root: A

**Pre-order**

order.Add(current node)

preOrder(left child)

preOrder(right child)

Order: A B D E G C F

**In-order**

inOrder(left child)

order.Add(current node)

inOrder(right child)

inOrder(A)

inOrder(A.leftChild = B)

inOrder(B.leftChild = D)

inOrder(D.leftChild) can’t because D.leftChild is null

**order.Add(D)**

**order.Add(B)**

inOrder(B.rightChild = E)

inOrder(E.leftChild = G)

inOrder(G.leftChild) can’t. G.leftChild = null

**order.Add(G)**

**order.Add(E)**

inOrder(E.rightChild) can’t. E.rightChild = null

**order.Add(A)**

inOrder(A.rightChild = C)

inOrder(C.leftChild = F)

inOrder(F.leftChild) can’t. F.leftChild is null

**order.Add(F)**

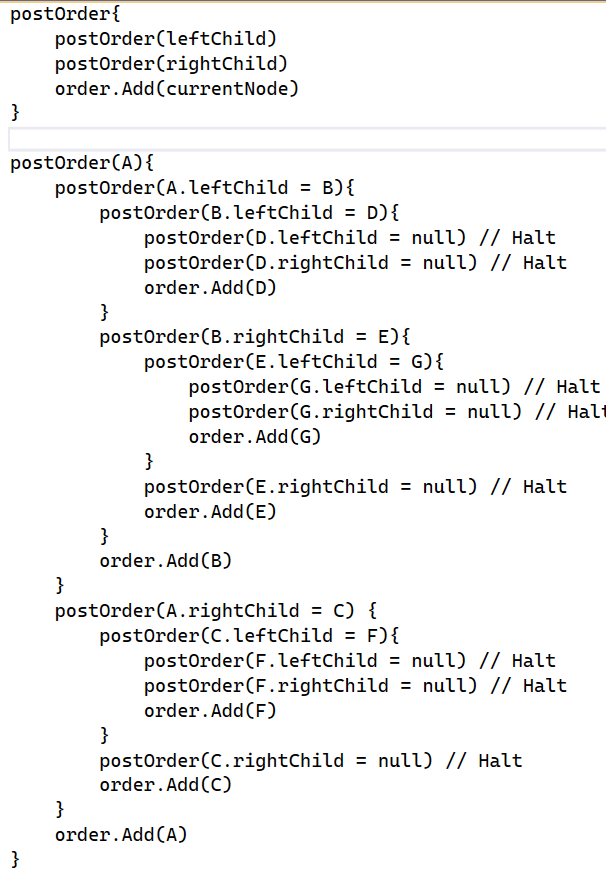
inOrder(F.rightChild) can’t

**order.Add(C)**

inOrder(C.rightChild) can’t

**Order: D B G E A F C**

Post-order



**Order: D G E B F C A**

**Question 2:**

*A binary search tree is a binary tree, in which every node has a left child less than it and a right child greater than or equal to it.*

**A.** Not a Binary Search Tree (BST) because on Node 25, the right child is 24, which is not greater than or equal to 25.

**B.** Not a Binary Search Tree (BST) because on Node 85, the right child is 79, which is less than 85.

**C.** Is a Binary Search Tree

**Question 3:**

Search(node, key):

If (node == key):

Return node

If (node < key):

Search(node.left)

Else:

Search(node.right)

Return null

**Search 52:**

52 50: Search the right sub tree (from 55):

52 < 55: Search the left sub tree (from 52):

52 == 52: return Node

Nodes visited: 50, 55, 52

**Search 46:**

46 < 50: Search the left sub tree (from 45):

46 > 45: Search the right sub tree (from 46):

46 == 46: return Node

Nodes visited: 50, 45, 46

**Search 75:**

75 > 50: Search the right sub tree (from 55):

75 > 55: Search the right sub tree (from 80):

75 < 80: Search from the left sub tree (from 79):

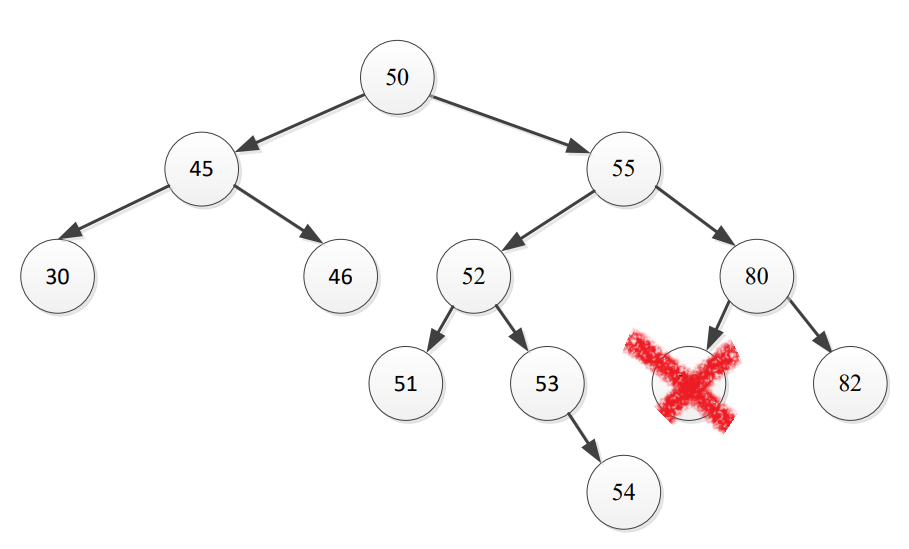
75 < 79: Search from the left sub tree (null):

return null

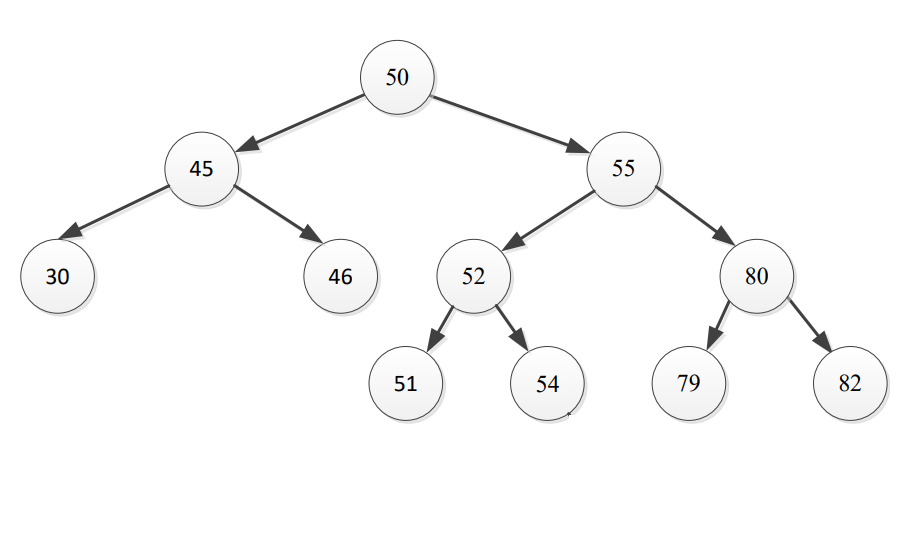
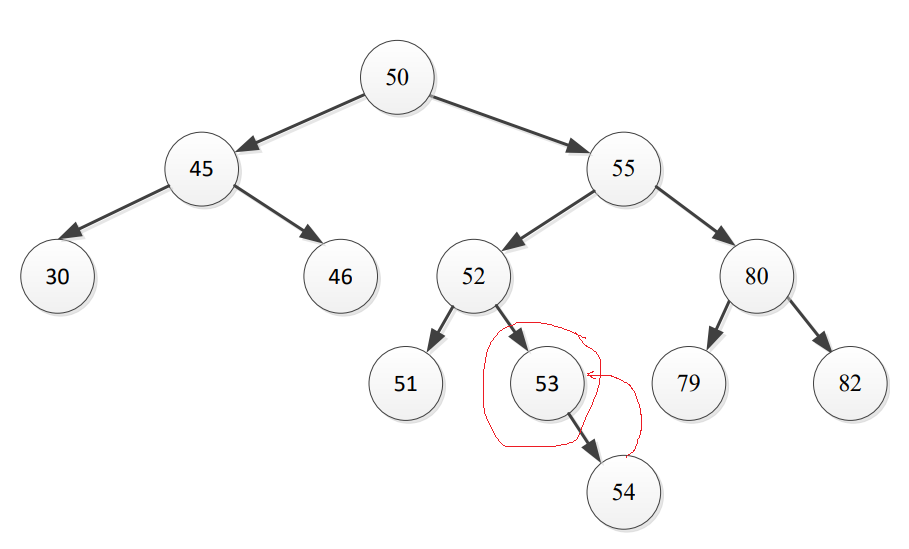
Nodes visited: 50, 55, 80, 79

**Delete:**

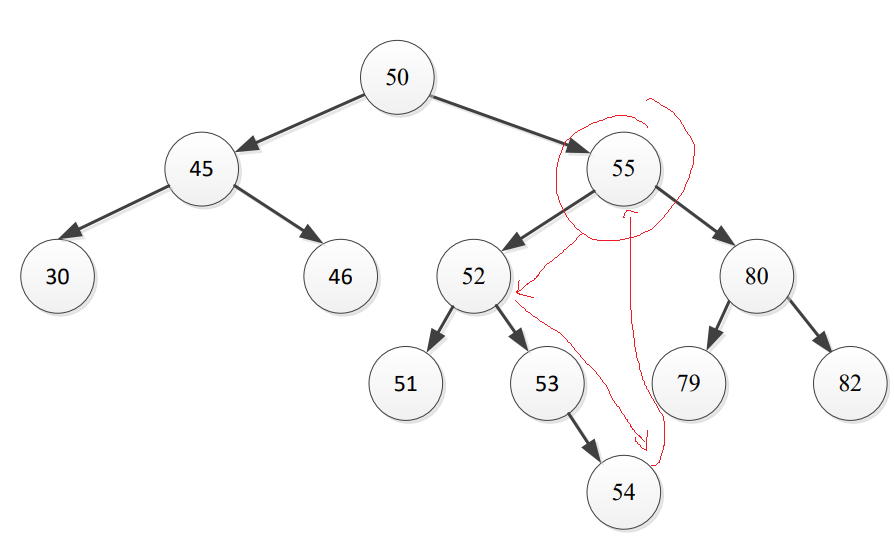
**79**: Has no child, so just remove it.

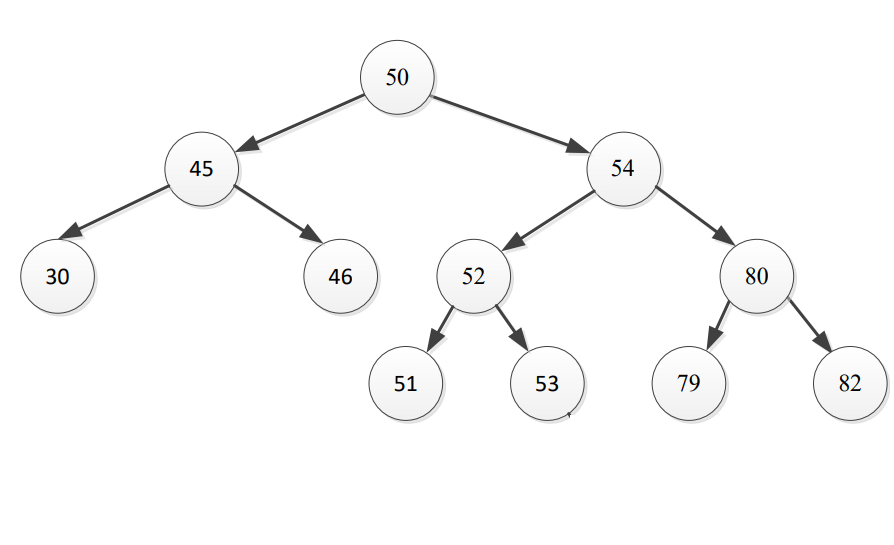


**53:** Has one child, so the child replaces it



**55:** Has two children, so look for the right-most leaf in the left sub-tree (Go left, then go right as far as you can)





**Insert:**

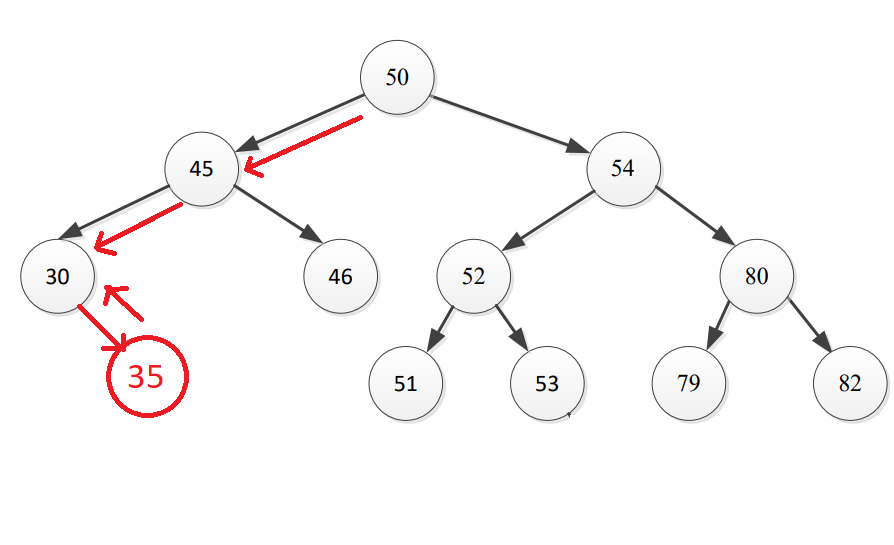
35

Insert (35, 50):

50 > 35 and 50.leftChild is not null: Insert(35, 50.leftChild = 45):

45 > 35 and 45.leftChild is not null: Insert(35, 30):

30 < 35 and 35.rightChild is null: Insert 35 to 30.right



**81**

