

PreOrder(root):

visit root.item

PreOrder(root.left)

PreOrder(root.right)

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

InOrder(root):

InOrder(root.left)

visit root.item

InOrder(root.right)

Visited: **D, B, G, E, A, F, C**

PostOrder(root):

PostOrder(root.left)

PostOrder(root.right)

visit root.item

Visited: **D, G, E, B, F, C, A**

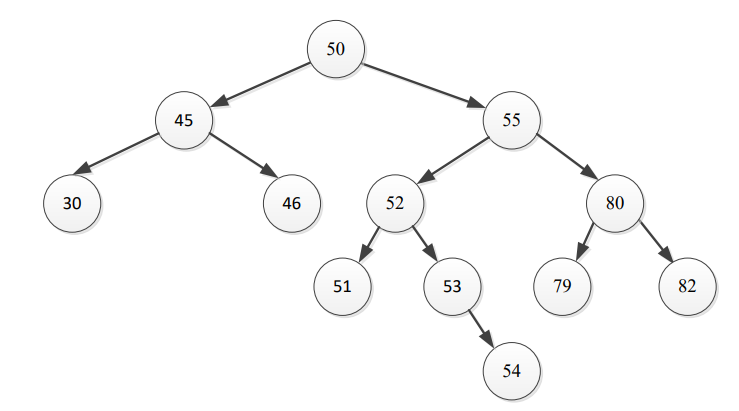
Question 2:

A: Not a Binary Search Tree (BST) because 24 < 25, so 24 should be on the left subtree of 25.

B: Not a BST because 79 < 85, so 79 should be on the left subtree of 85

C: Is a BST, because for each node, every item in its left subtree is less than that node, and every item in its right subtree is less than that node.

Traverse the tree in C:



**Pre-order: 50, 45, 30, 46, 55, 52, 51, 53, 54, 80, 79, 82**

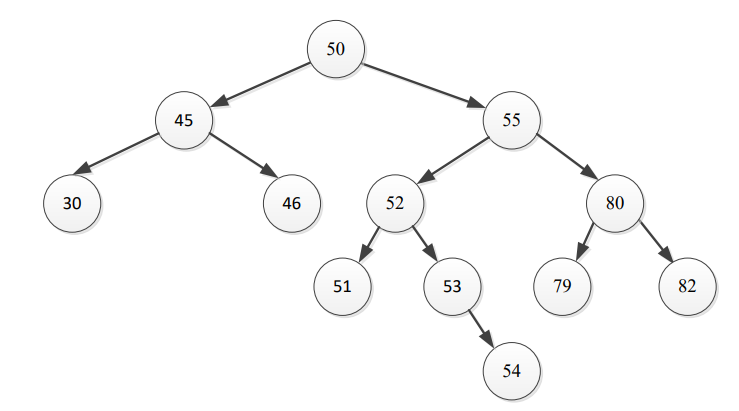
* If we traverse an insert to a new tree in pre-order, we get an exact copy

In-order: 30, 45, 46, 50, 51, 52, 53, 54, 55, 79, 80, 82

* This is sorted in non-descending order. If traverse and insert, we will get somewhat of a sorted linked list.

Post-order: 30, 46, 45, 51, 54, 53, 79, 82, 80, 55, 50

Question 3:



Searching:

**52:**

Went through 50, 55 and 52

**46:**

Search(K, root):

if root.item == K: return true

else:

if root.item < K: return Search(K, root.right)

else: return Search(K, root.left)

Search(46, 50)

50 > 46: return Search(46, 50.left => 45):

45 < 46: return Search(46, 45.right => 46):

46 == 46: return true

Visited 50, 45, 46

**75:**

Search(75, 50):

50 < 75: return Search(75, 50.right => 55):

55 < 75: return Search(75, 50.right => 80):

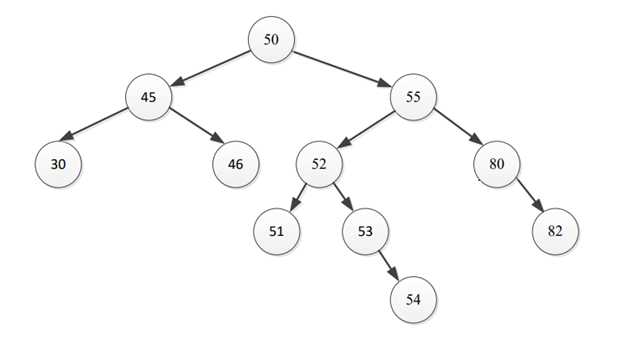
80 > 75: return Search(75, 80.left => 79):

79 > 75: return Search(75, 79.left => null): return false

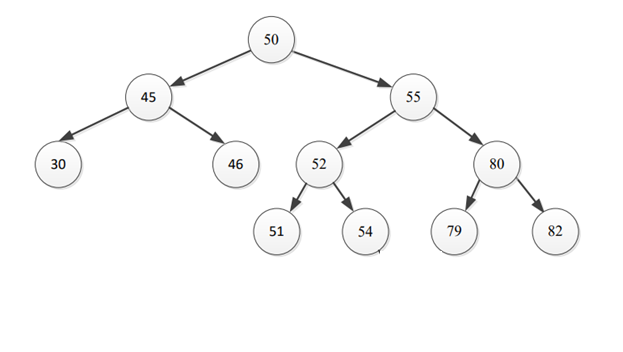
Visited 50, 55, 80, 79

Deleting:

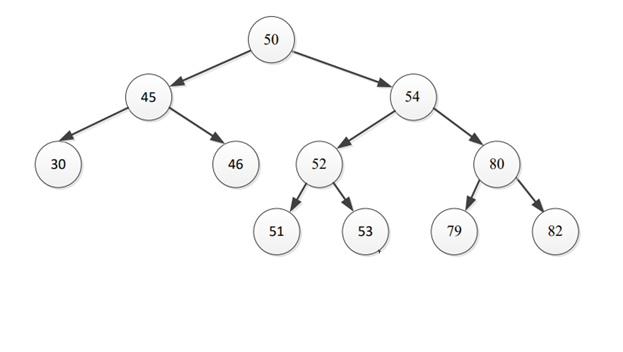
**79:** A leaf, so just remove it.



**53**: A node with one child, so have the child take its place



**55**: Has two children and some grand-children. Replace it with the right most element in its left subtree.



Insert:

Insert(K, root):

if root is null, make it K

else:

if root.item > K: try to insert to the left

if root.left is null, make it K

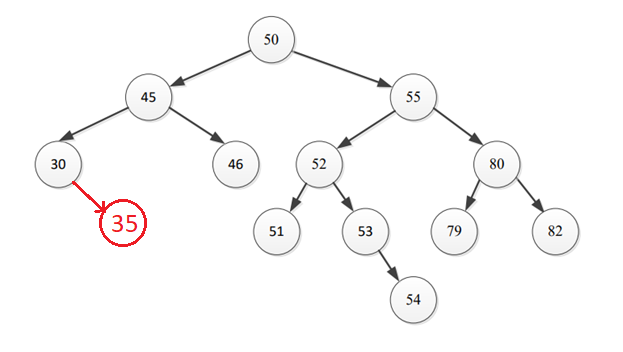
else Insert(K, root.left)

else:

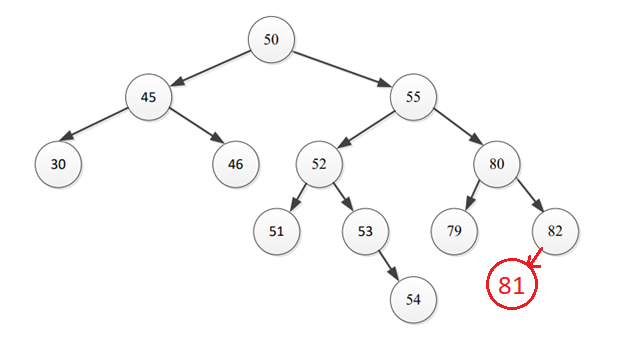
if root.right is null, make it K

else Insert(K, root.right)

**35**: 50 -> 45 (since 35 < 50) -> 30 (since 35 < 45) -> Insert 35 to the right of 30 (currently null)



**81:** 50 -> 55 (since 50 < 81) -> 80 (since 55 < 81) -> 82 (since 80 < 81) -> Insert to the left of 82 (currently, null; since 82 > 81)

****