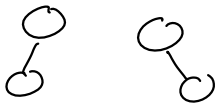
Tree? Kind of data structure

Has nodes, each node has at most 1 parent, and some children

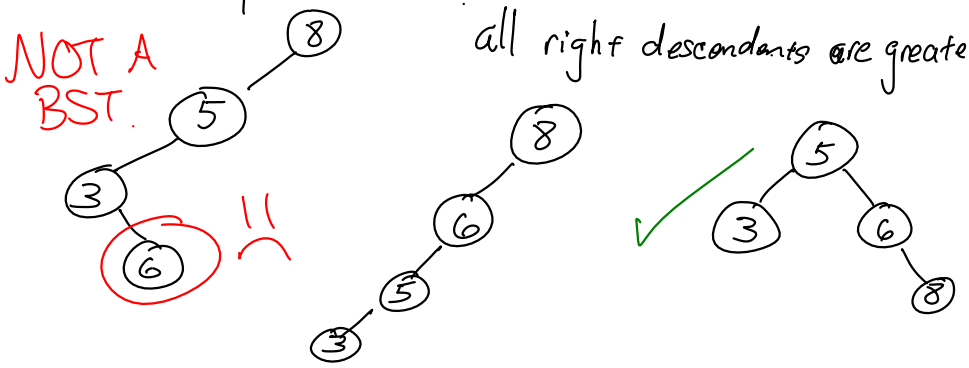
also nodes

Binary Tree? Tree where each node has
at most 1 left child and
at most 1 right child



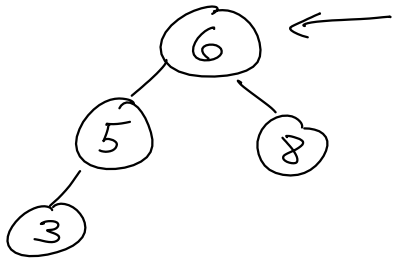
Binary Search Tree?

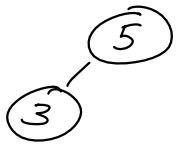
Binary Tree where all left descendents are less than a given node,
all right descendants are greater



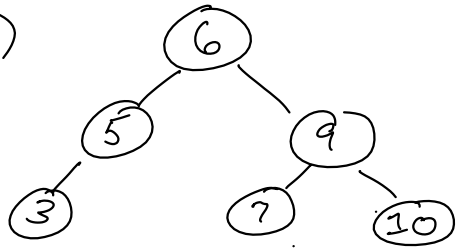
```
Method get(key):
    If root == null:
        (throw exception)
    End If
    node ← findInSubtree(key, root)
    Return node.value
End Method

Method findInSubtree(key, root)
    If root == null:
        
    If root.key == key:
        Return root
    Else If root.key < key:
        Return findInSubtree(key, root.right)
    Else
        Return findInSubtree(key, root.left)
    End If
End Method
```





```
Function InOrder Traversal (node, list)
    If node.left ≠ null:
        InOrderTraversal(node.left, list)
    End If
    list.insertAt Tail (node.key)
    If node.right ≠ null:
        InOrder Traversal(node.right, list)
    End If
End Function
```



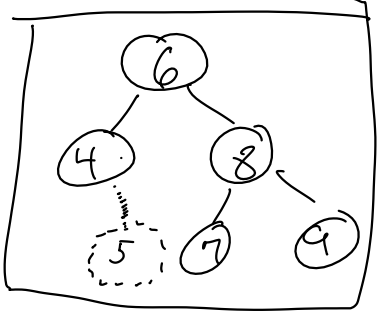
3, 5, 6, 7, 9, 10

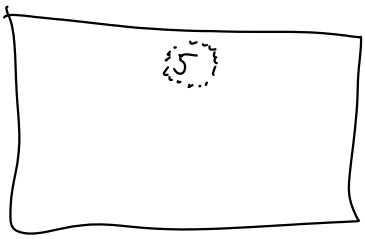
Pre Order	6, 5, 3, 9, 7, 10
Post Order	3, 5, 7, 10, 9, 6
In Order	3, 5, 6, 7, 9, 10

```
Function Post Order Traversal (node, list)
    If node.left ≠ null:
        Post OrderTraversal(node.left, list)
    End If
    If node.right ≠ null:
        Post Order Traversal(node.right, list)
    End If
    list.insertAt Tail (node.key)
End Function
```

LinkedBST

bst . insert (5, ···)






```
Method insert(key, value):
    this.root ← insertInSubtree(this.root, key, value)
EndMethod
```

insertInSubtree returns the replacement for node

```
Method insertInSubtree(node, key, value)
    If node == null:
        newnode ← new LinkedBSTNode(key, value, null, null)
        Return newnode
    Else If key < node.key:
        node.left ← insertInSubtree(node.left, key, value)
    Else If key > node.key:
        node.right ← insertInSubtree(node.right, key, value)
    Else
        throw !!
    End If
    Return node
End Method
```

```
Method findMinInSubtree(node):
    If node.left ≠ null
        Return findMinInSubtree(node.left)
    Else
        Return node
    End If
End Method

Method removeInSubtree(node, key):
    If node == null:
        throw !!
    Else If key < node.key:
        node.left ← removeInSubtree(node.left, key)
    Else If key > node.key:
        node.right ← removeInSubtree(node.right, key)
    Else:
        If node.left == null and node.right == null:
            Return null
        Else If node.right == null:
            Return node.left
        Else If node.left == null:
            Return node.right
        Else:
            smallest ← findMinInSubtree(node.right)
            node.key ← smallest.key
            node.value ← smallest.value
            node.right removeInSubtree(node.right, smallest)
            Return node
        End If
    End If
End Method
```