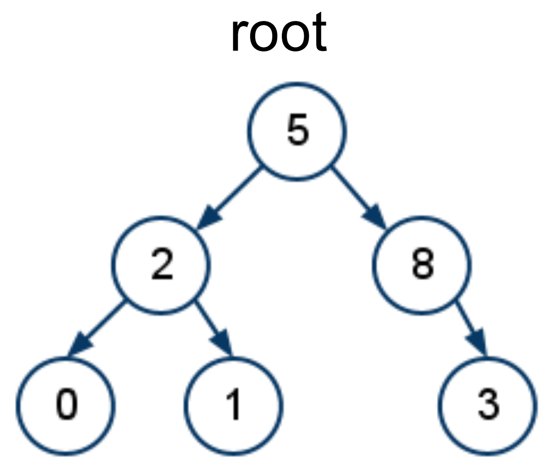
CMSC 132: Object-Oriented Programming II

Binary Search Trees

Quiz 1: What is the output?

```
void m(Node r){
  if(r==null) return;
  print(r.key+",");
  m(r.left);
  m(r.right);
}
m(root);
```

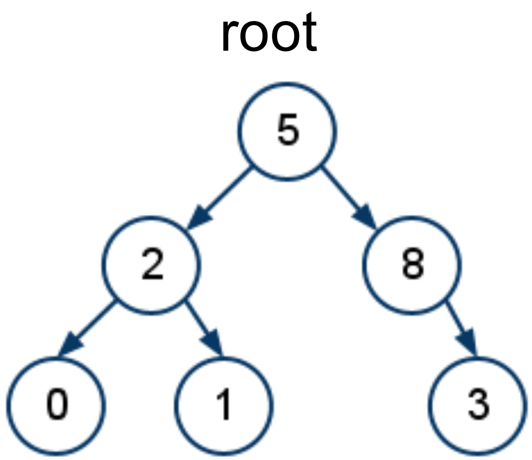


A. 5,2,0,1,3,8
B. 5,2,0,1,8,3
C. 5,2,8
D. 5,2,0,1

Quiz 1: What is the output?

```
void m(Node r){
  if(r==null) return;
  print(r.key+",");
  m(r.left);
  m(r.right);
}
m(root);
```

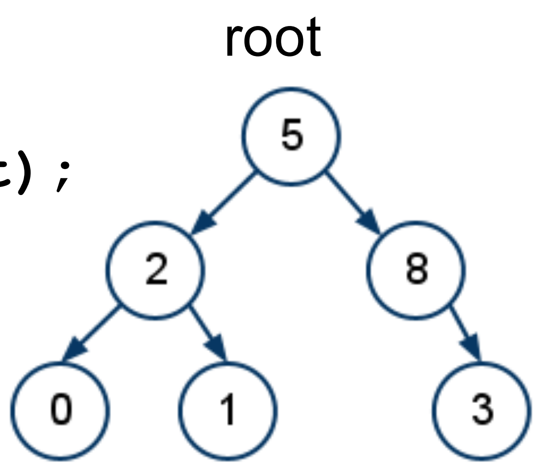
preOrder traversal



A. 5,2,0,1,3,8
B. 5,2,0,1,8,3
C. 5,2,8
D. 5,2,0,1

Quiz 2: What is the output?

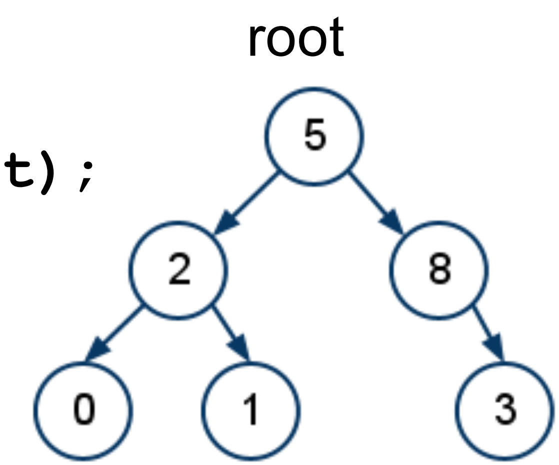
```
int m(Node r){
  if(r==null) return 0;
  return r.key  + m(r.left) + m(r.right);
}
m(root);
```



A. 6
B. 19
C. 15
D. 5

Quiz 2: What is the output?

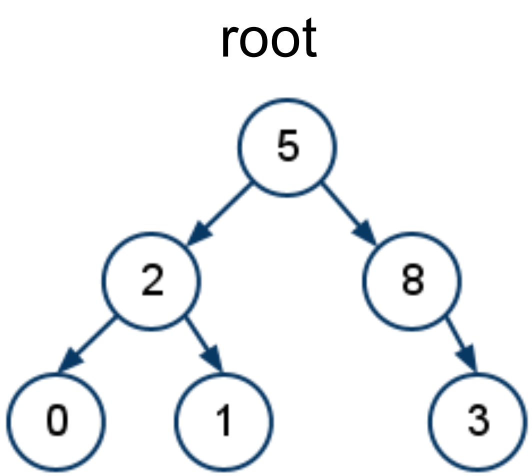
```
int m(Node r){
  if(r==null) return 0;
  return r.key  + m(r.left) + m(r.right);
}
m(root);
```



A. 6
B. 19
C. 15
D. 5

Quiz 3: What is the output?

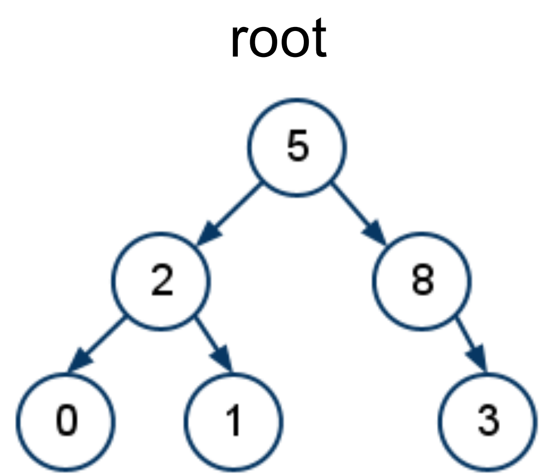
```
int m(Node r){
  if(r==null) return 0;
  if(r.left==null && r.right==null) return r.key;
  return m(r.left) + m(r.right);
}
m(root);
```



A. 6
B. 18
C. 4
D. 3

Quiz 3: What is the output?

```
int m(Node r){
  if(r==null) return 0;
  if(r.left==null && r.right==null) return r.key;
  return m(r.left) + m(r.right);
}
m(root);
```

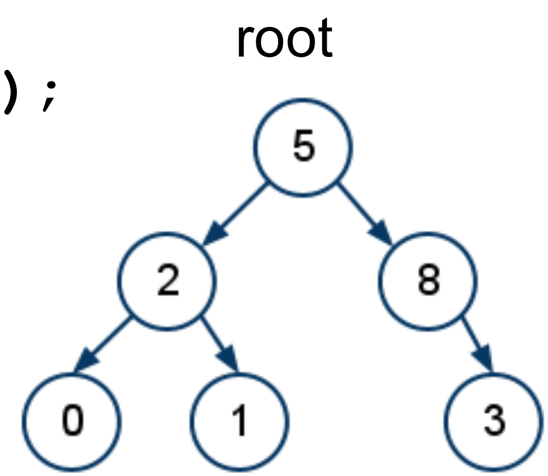


A. 6
B. 18
C. 4
D. 3

Sum of keys in leaf nodes

Quiz 4: What is the output?

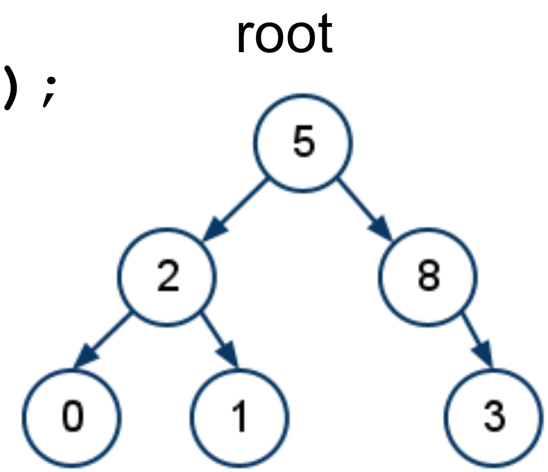
```
void m(){
  Queue<Node> q = new LinkedList();
  if(root==null) return;
  q.offer(root);
  while(!q.isEmpty()){
    Node t = q.poll();
    if(t.right!=null) q.offer(t.right);
    if(t.left!=null) q.offer(t.left);
    print(t.key+",");
  }
}
m();
```



A. 5,2,8,0,1,3
B. 5,8,2,3,1,0
C. 0,1,3,2,8,5
D. 3,1,0,8,2,5

Quiz 4: What is the output?

```
void m(){
  Queue<Node> q = new LinkedList();
  if(root==null) return;
  q.offer(root);
  while(!q.isEmpty()){
    Node t = q.poll();
    if(t.right!=null) q.offer(t.right);
    if(t.left!=null) q.offer(t.left);
    print(t.key+",");
  }
}
m();
```

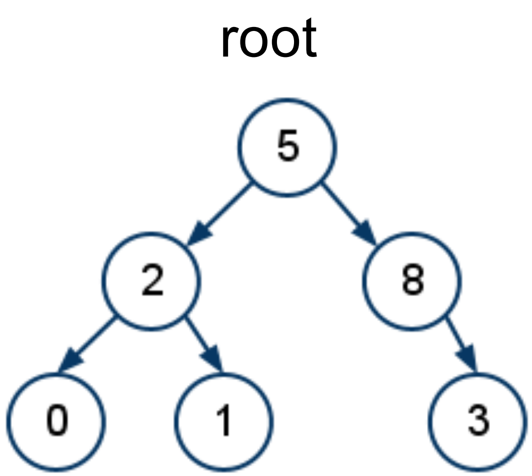


A. 5,2,8,0,1,3
B. 5,8,2,3,1,0
C. 0,1,3,2,8,5
D. 3,1,0,8,2,5

Level order right to left

Quiz 5: What is the output?

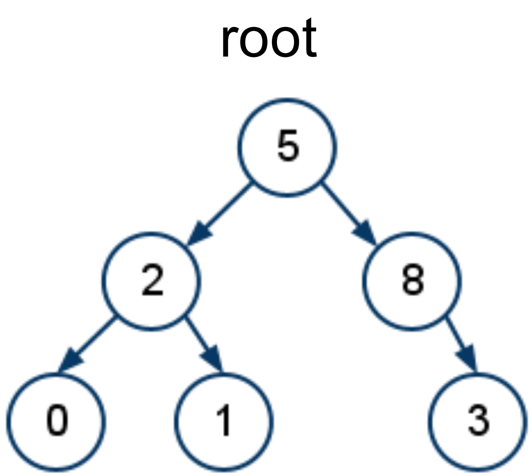
```
void m(Node r, int n, int c){
  if(r==null) return;
  if(c == n){
    System.out.print(r.key+",");
  }
  m(r.left, n, c+1);
  m(r.right, n, c+1);
}
m(root,3,1);
```



A. 2,8
B. 0,1,3
C. 5,2,8,0,1,3
D. 5,2,0,1,8,3

Quiz 5: What is the output?

```
void m(Node r, int n, int c){
  if(r==null) return;
  if(c == n){
    System.out.print(r.key+",");
  }
  m(r.left, n, c+1);
  m(r.right, n, c+1);
}
m(root,3,1);
```



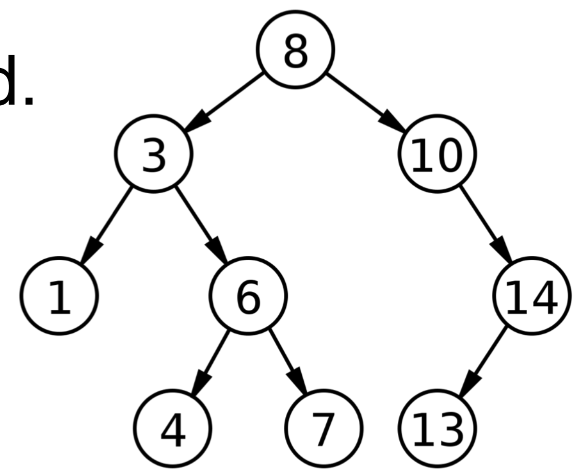
A. 2,8
B. 0,1,3
C. 5,2,8,0,1,3
D. 5,2,0,1,8,3

Print nodes at level n

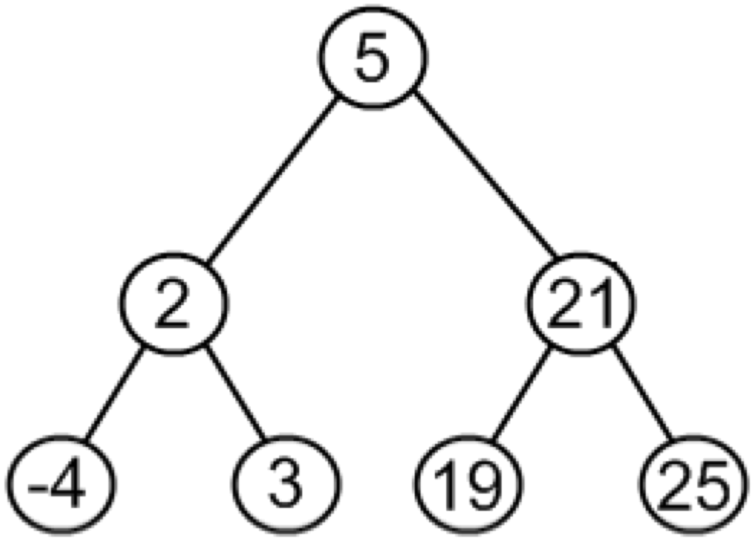
Binary Search Tree

- A BST is a binary tree

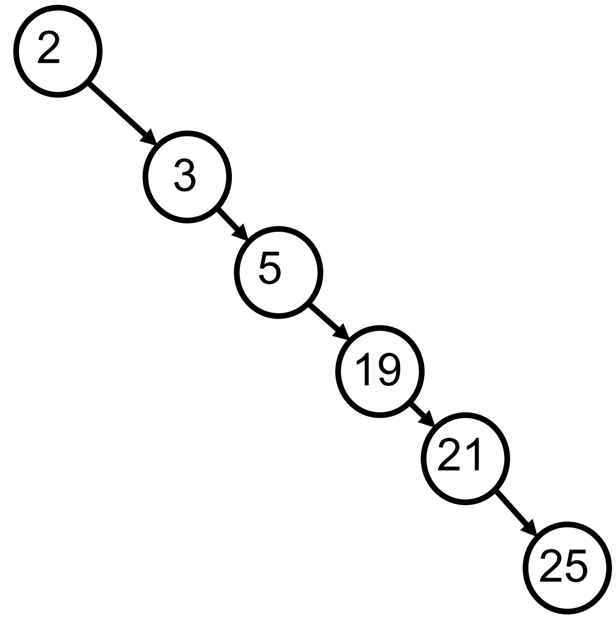
- Nodes are ordered in the following way:
  - Each node contains one key (also known as data
  - Keys in the left subtree are less then the key in its parent node
  - Keys in the right subtree are greater the key in its parent node
  - Duplicate keys are not allowed.



Balanced vs Not Balanced BST



Balanced
Height: O(log n)



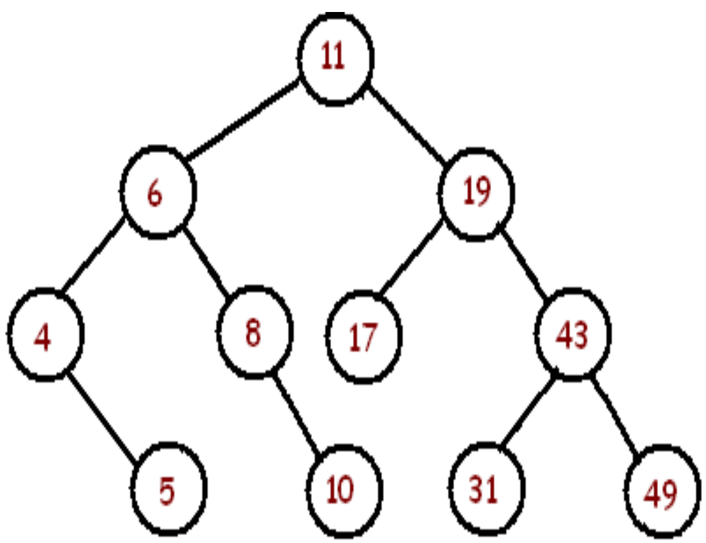
Not Balanced
Height: O(n)

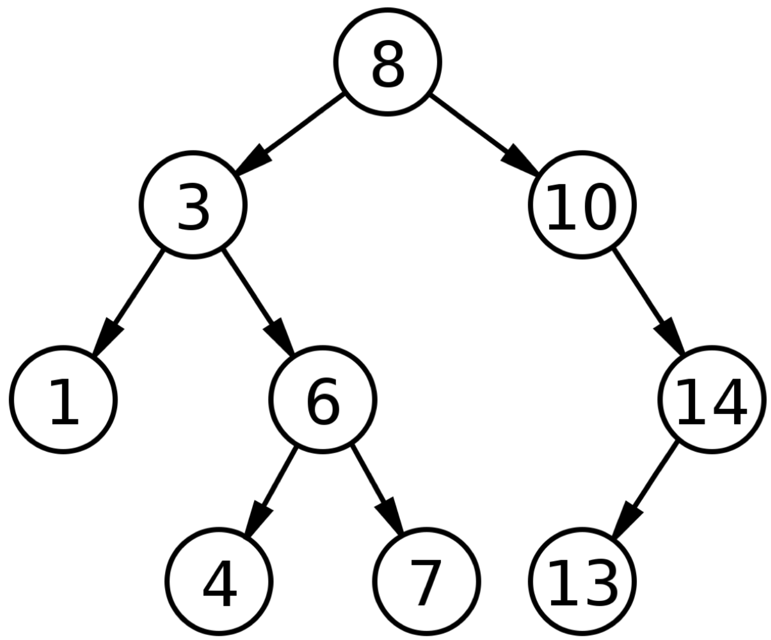
BST Class

```
public class BST<Key extends Comparable<Key>, Value>{
      private Node root;
      private class Node{
         private Key key;
         private Value value;
         private Node left, right;
         public Node(Key k, Value v){
            key = k;
            value = v;
      }
}
```

Search

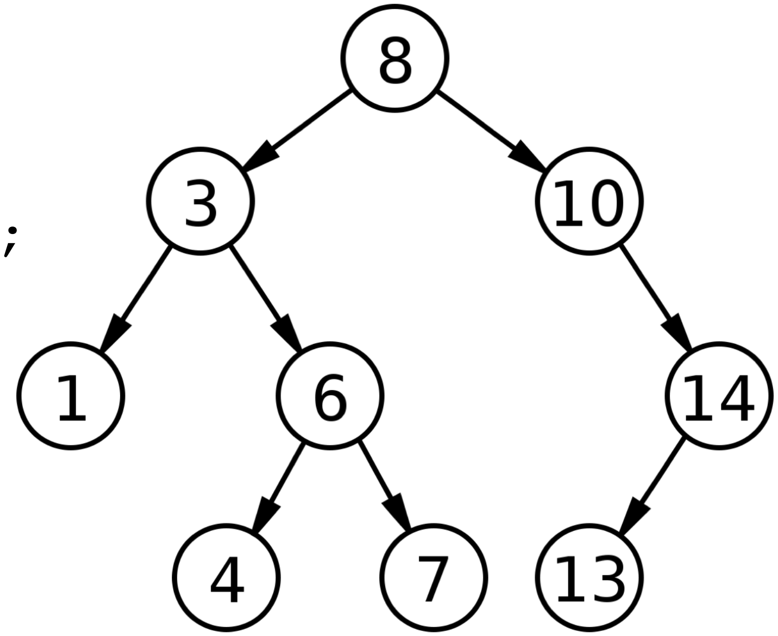
- Since s binary search tree with n nodes has a minimum of O(log n) levels, it takes at least O(log n) comparisons to find a particular node.
- Unfortunately, a binary search tree can degenerate to a linked list, reducing the search time to O(n).





Search

```
public Value get(Key key){
  return get(root, key);
}

private Value get(Node x, Key key){
    if(x == null) return null;
    int cmp = key.compareTo(x.key);
    if(cmp < 0)
        return get(x.left, key);
    else if(cmp > 0)
        return get(x.right, key);
    else return x.value;
}
```

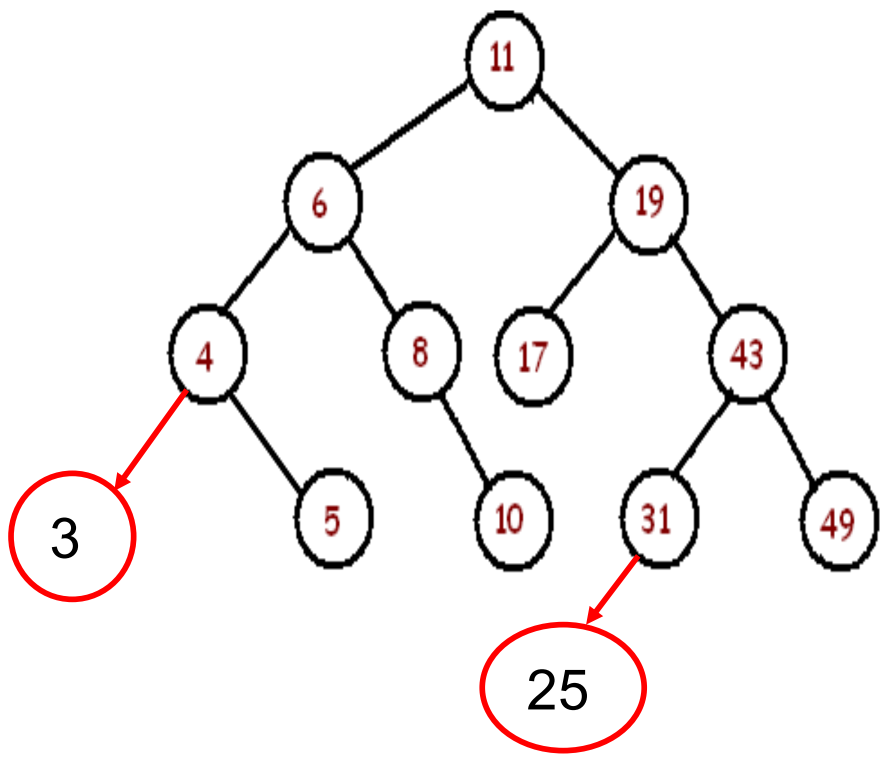


Average: O(log n)
Worst: O(n)

Insert

- The insertion procedure is quite similar to searching.
- We start at the root and recursively go down the tree searching for a location in a BST to insert a new node.
- New node always becomes a leaf node

insert(3)
insert(25)



Insert

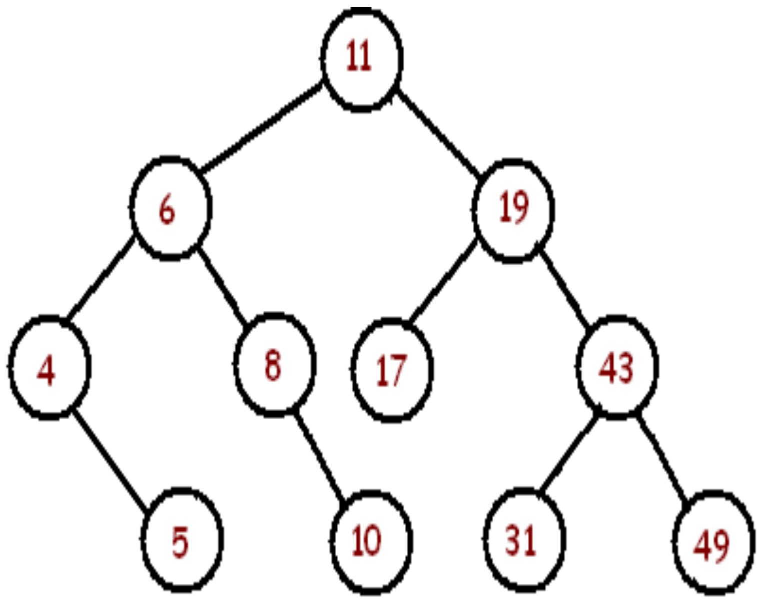
```
public void put(Key key, Value val){
   root = put(root, key, val);
}

private Node put(Node x, Key key, Value val){
   if(x == null) {return new Node(key,val);}
   int cmp = key.compareTo(x.key);
   if(cmp < 0) {x.left = put(x.left, key, val);}
   else if(cmp>0) {x.right = put(x.right, key, val);}
   else {x.value = val;}
   return x;
}
```

Delete

- There are several cases to consider.
- A node to be deleted:
  - is not in a tree: there is nothing to delete
  - is a leaf: remove the node
  - has only one child: same as deleting a node from a linked list
  - has two children:
    - Replace the node with:
      - Largest of left subtree
      - Smallest of Right subtree

Delete Example

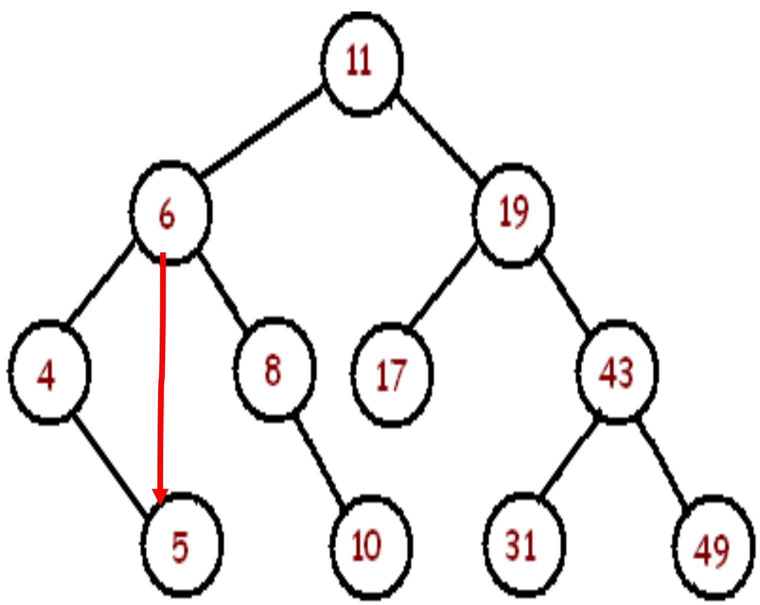
Delete 5



5 is a leaf node. It can be removed directly.

Delete Example

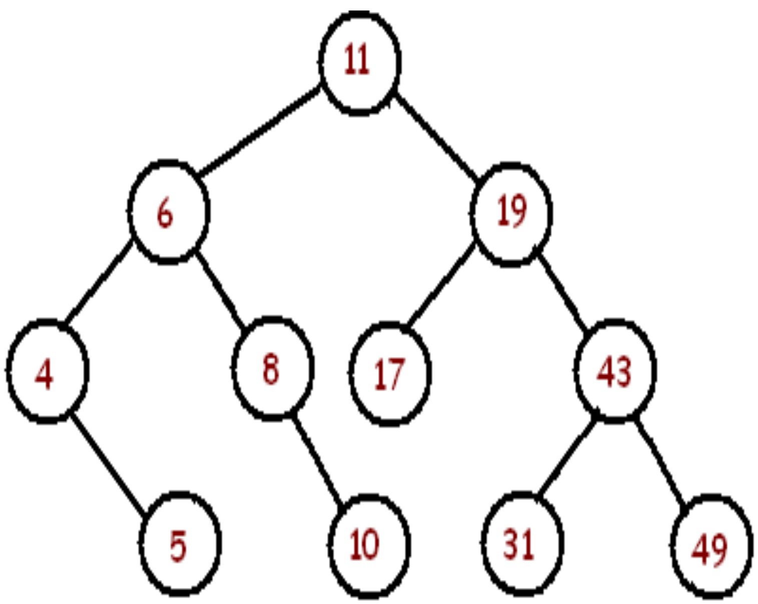
Delete 8



Node 8 has only one child. Parent, Node 6, will point to node 10, the child of node 8.

Delete Example

Delete 11



Node 10, the max of left subtree or Node 17, the min of right subtree will replace Node 11.

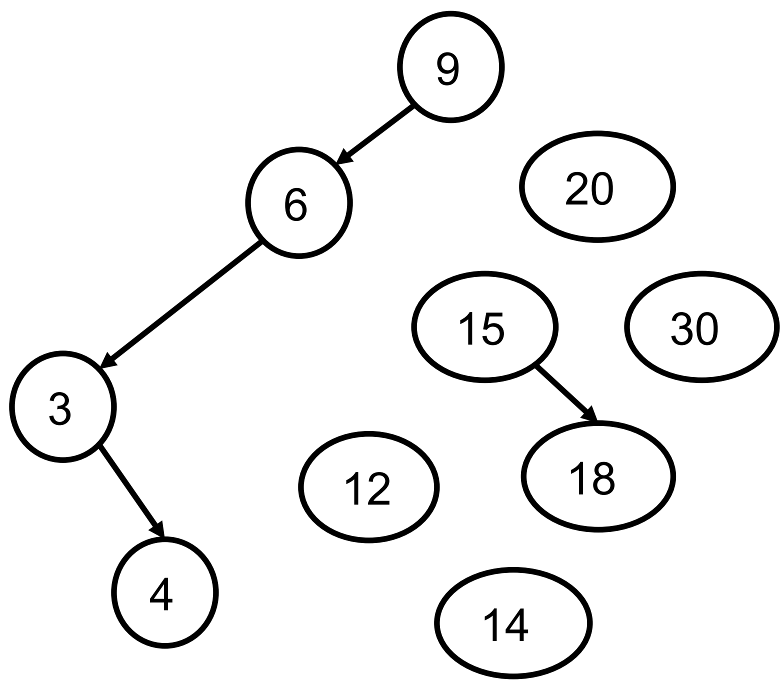
Delete

```
private Node delete(Node x, Key k){
   if(x == null) return null;
   int cmp = k.compareTo(x.key);
   if(cmp < 0) x.left = delete(x.left, k);
   else if (cmp > 0) x.right = delete(x.right, k);
   else{
     if(x.right == null) return x.left;
     if(x.left == null) return x.right;
     Node t = x;
     x = min(t.right);
     x.right = deleteMin(t.right);
     x.left = t.left;
   }
   return x;
}
```

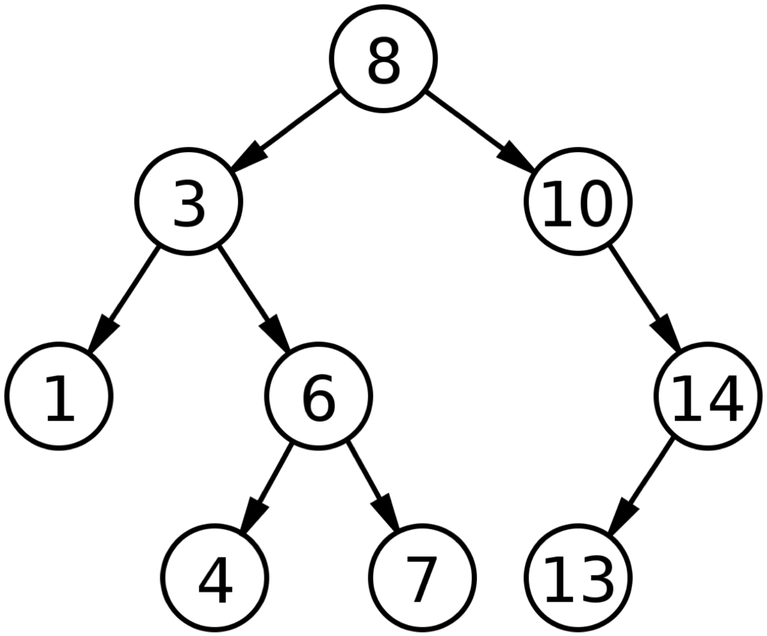
Delete Example

```
private Node delete(Node x, Key k){
   if(x == null) return null;
   int cmp = k.compareTo(x.key);
   if(cmp < 0) x.left = delete(x.left, k);
   else if (cmp > 0) x.right = delete(x.right, k);
   else{
     if(x.right == null) return x.left;
     if(x.left == null) return x.right;
     Node t = x;
     x = min(t.right);
     x.right = deleteMin(t.right);
     x.left = t.left;
   }
   return x;
}
```

Delete 9



isBST



isBST

- 1) inOrder , check if sorted
- 2)

```
boolean isBST(Node x, Key min, Key max) {
  if (x == null) return true;
  if (min != null && x.key.compareTo(min) <= 0)
      return false;
  if (max != null && x.key.compareTo(max) >= 0)
      return false;
  return isBST(x.left, min, x.key) &&
      isBST(x.right, x.key, max);
}
```

```
isBST(root, null, null);
```

Find min

Find max

Construct a BST

- **Construct a BST from given preorder traversal**
- preOrder: {10, 5, 1, 7, 40, 50}

LCA (Least Common Ancestor)

`LCA(5,10) = 6`

`5<6<10`