COMP 2611, DATA STRUCTURES
LECTURE 10

BINARY SEARCH TREES

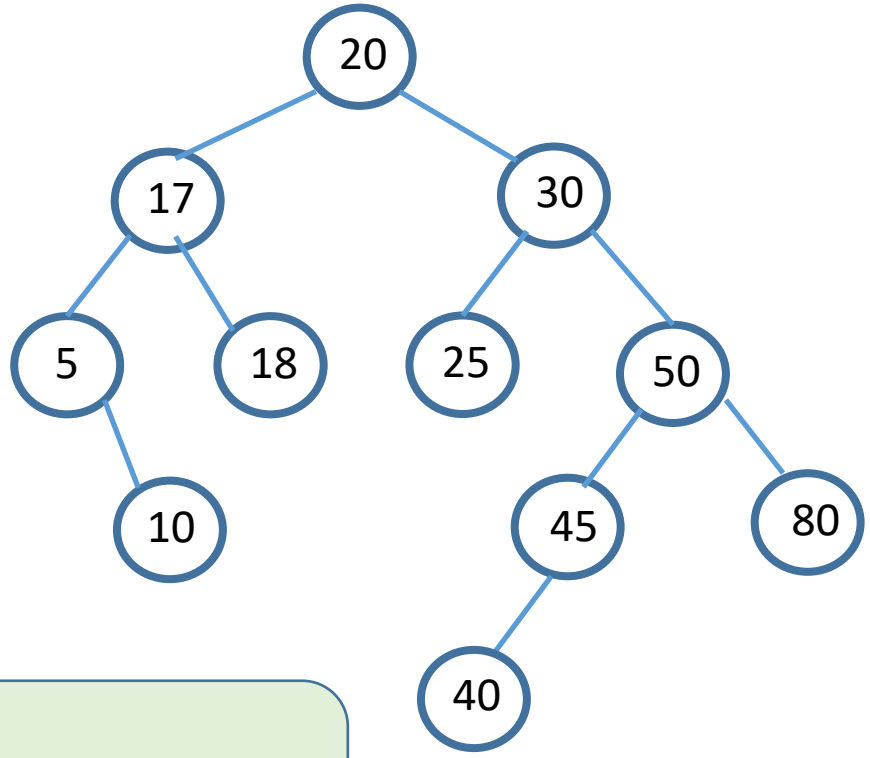
- **Inorder successor**
- **Degenerate BSTs**
- **Deleting a Node from a BST**

RETURN TO BINARY TREES

- **Performing a Level Order Traversal**

Inorder Successor

- Give the inorder traversal of the following BST:

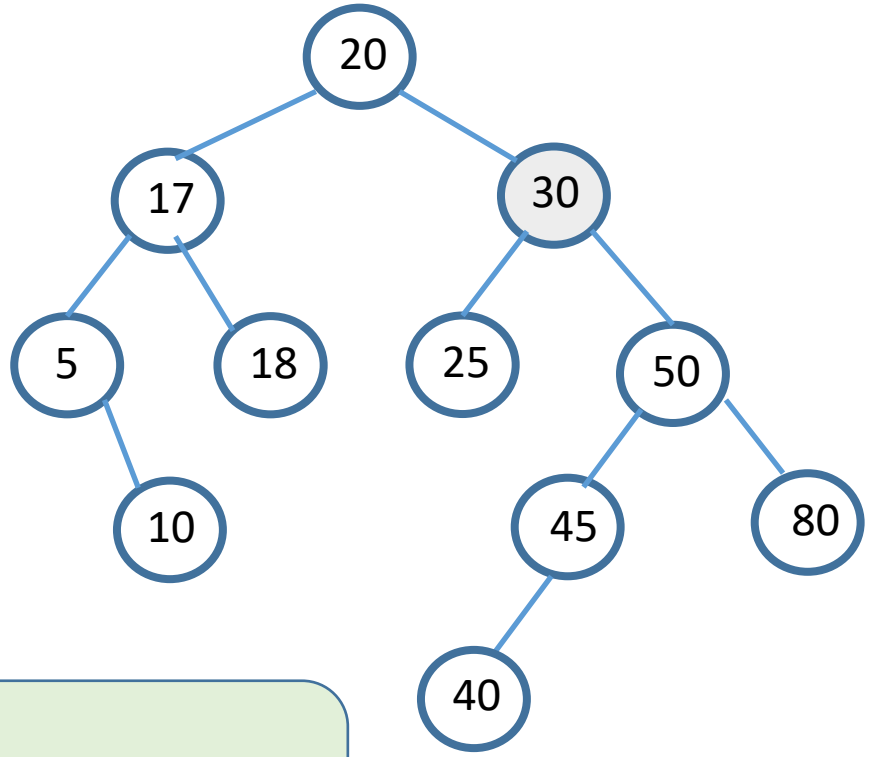


What's the inorder successor of 30?

5, 10, 17, 18, 20, 25, 30, 40, 45, 50, 80

Inorder Successor

- Give the inorder traversal of the following BST:

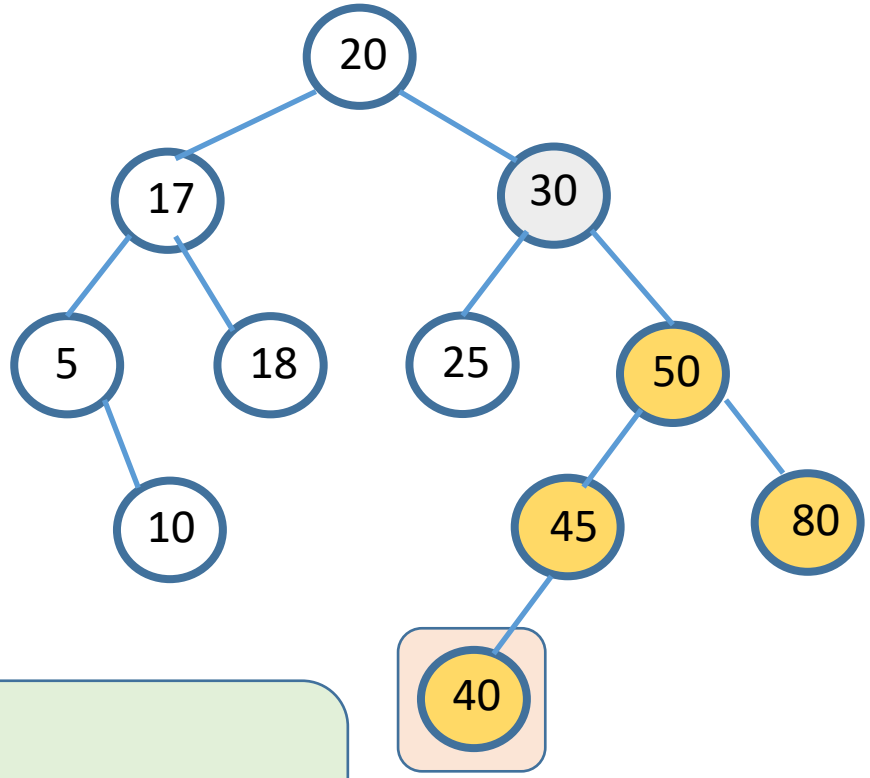


What's the inorder successor of 30?

5, 10, 17, 18, 20, 25, 30, 40, 45, 50, 80

Inorder Successor

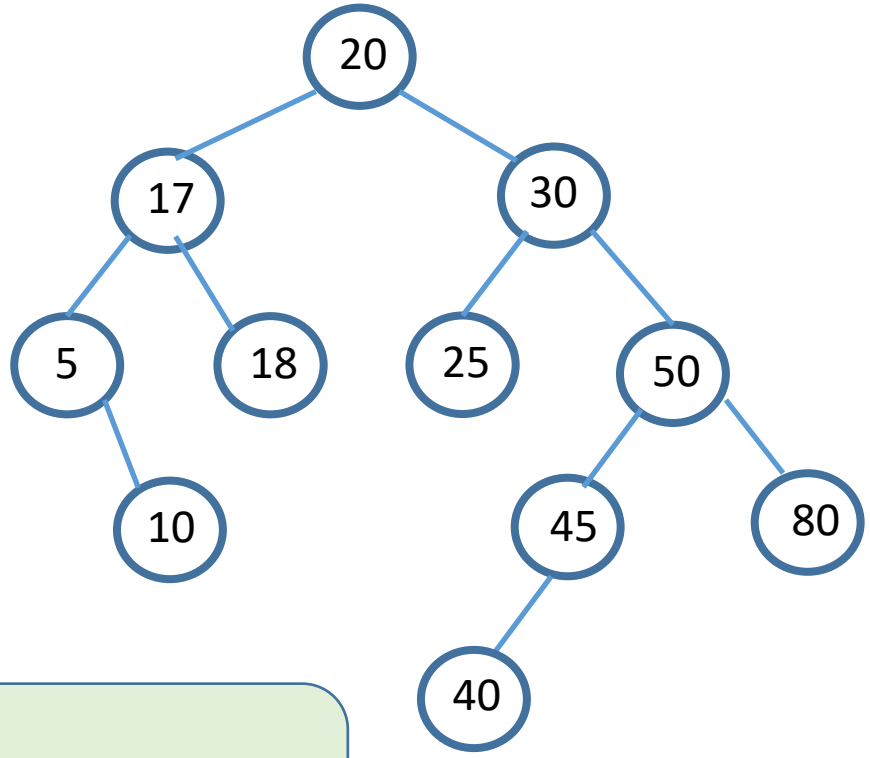
- Give the inorder traversal of the following BST:



How to find the inorder successor of 30?

5, 10, 17, 18, 20, 25, 30, 40, 45, 50, 80

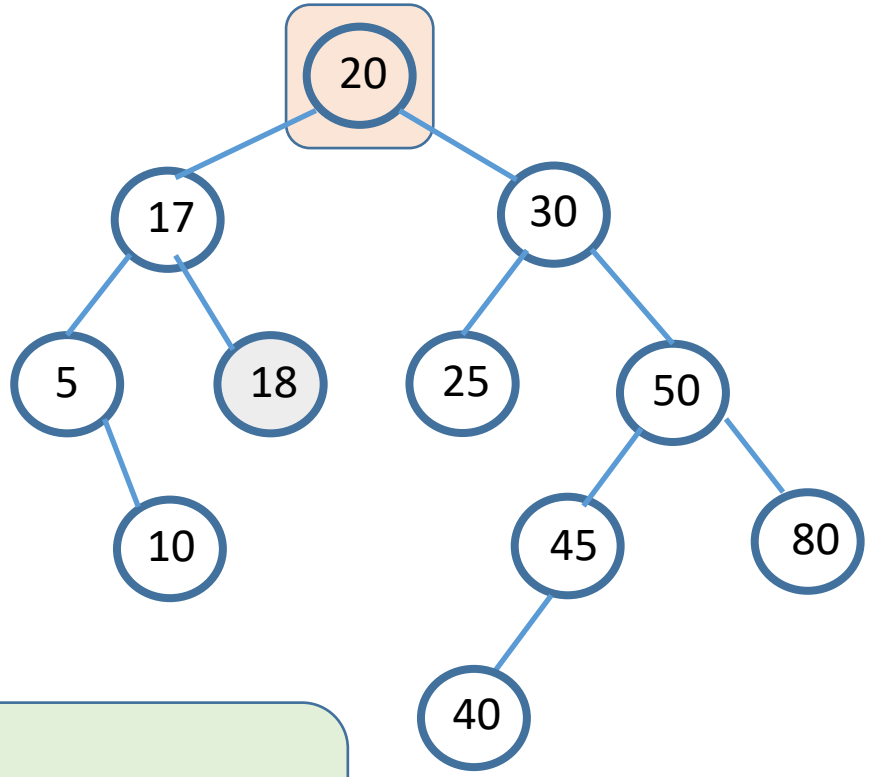
Inorder Successor



What's the inorder successor of 18?

5, 10, 17, 18, 20, 25, 30, 40, 45, 50, 80

Inorder Successor



What's the inorder successor of 18?

5, 10, 17, 18, 20, 25, 30, 40, 45, 50, 80

Finding the Inorder Successor

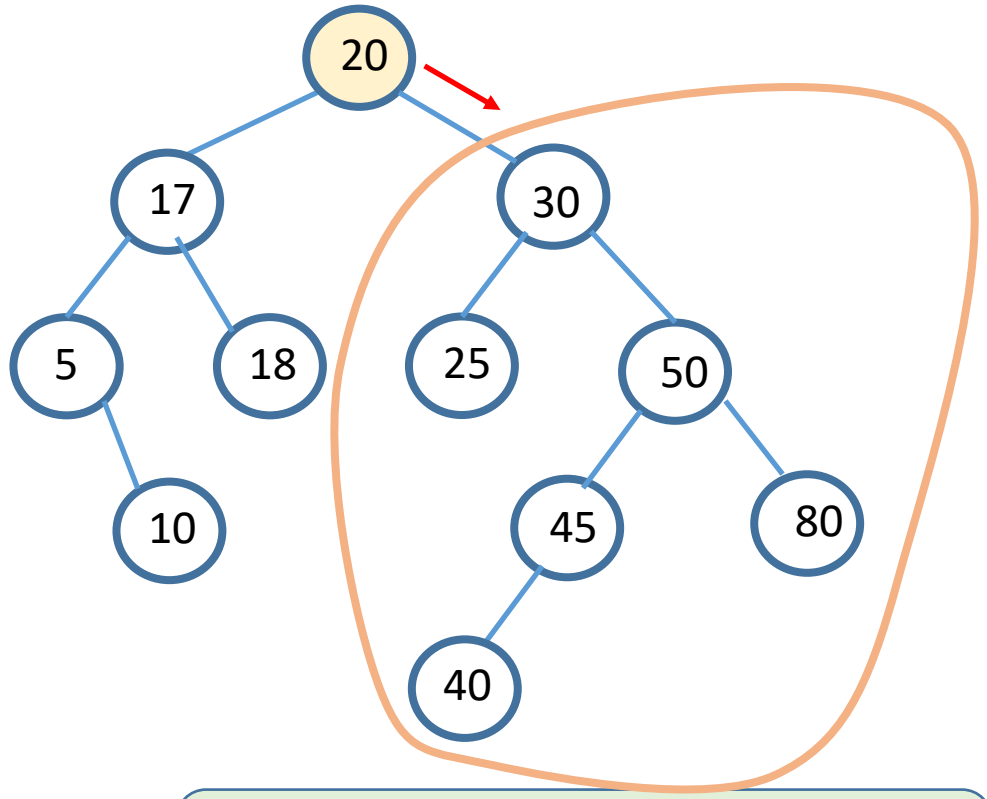
Case 1:

- Node has a non-empty right subtree (e.g., 5, 17, 20, 30, 50)

- The inorder successor is the first node in an inorder traversal of the right subtree. How to find this node?

```
int inOrderSuccessor (BTNode * node) {

        if (node->right != NULL) {
                return minimum (node->right);
        }
        …
}
```

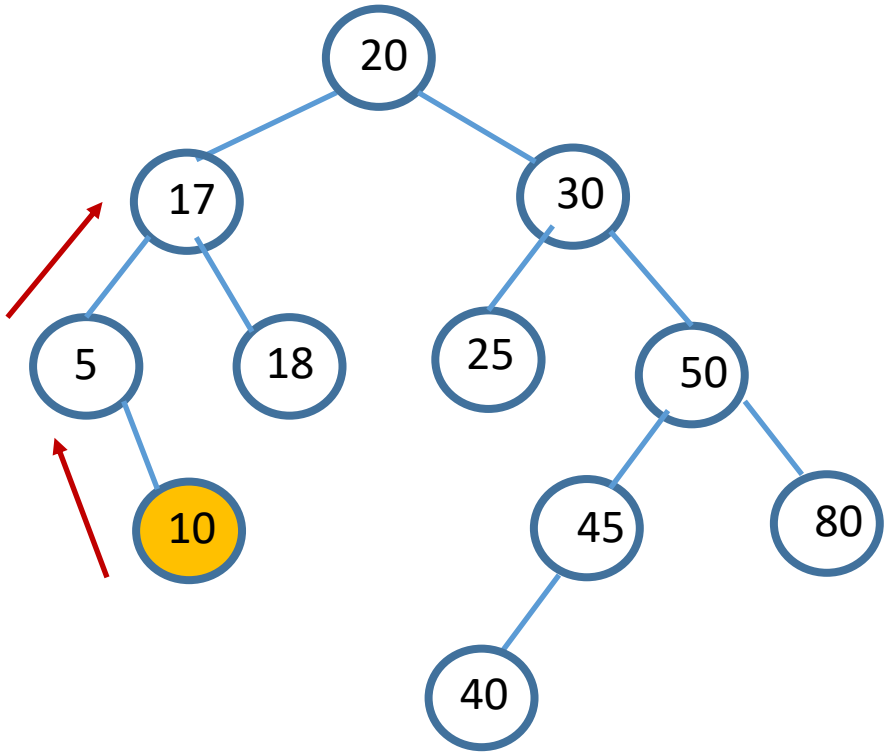


5, 10, 17, 18, 20, 25, 30, 40, 45, 50, 80

Finding the Inorder Successor

Case 2:

- Node has an empty right subtree e.g., 10, 18, 25, 45, 80)
- The inorder successor is one of its ancestors. Which one?
- Suppose that *x* has a successor *y*. Then, *y* is the lowest ancestor of *x* whose left child is also an ancestor of *x*.
- To find *y*, we go up the tree from *x* until we encounter a node that is the left child of its parent. If no such node is encountered, there is no successor.



Inorder successor of:

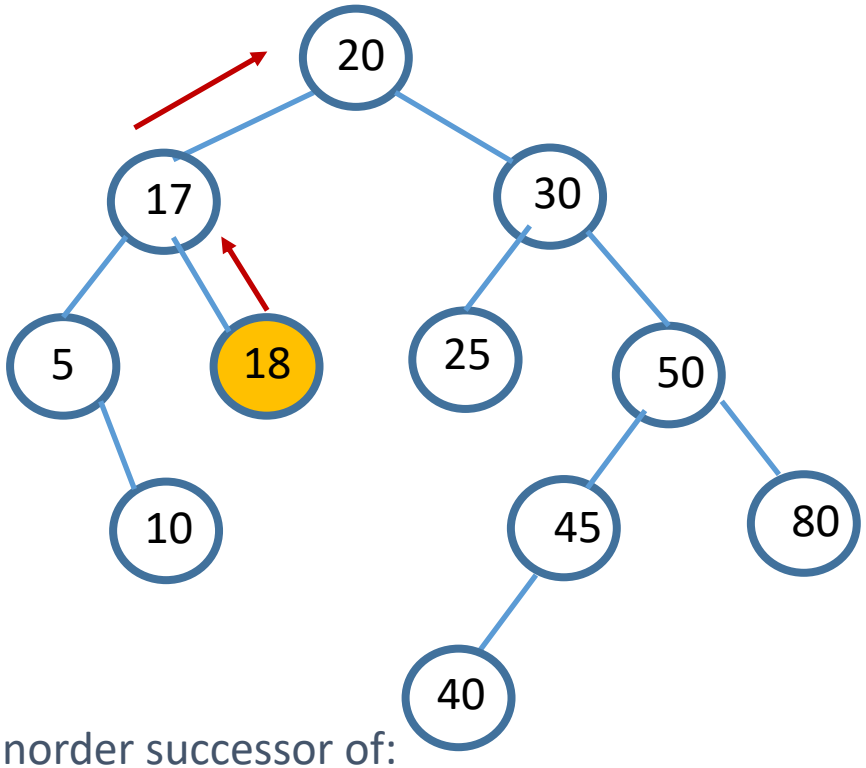
- 10 is 17

5, 10, 17, 18, 20, 25, 30, 40, 45, 50, 80

Finding the Inorder Successor

Case 2:

- Node has an empty right subtree e.g., 10, 18, 25, 45, 80)
- The inorder successor is one of its ancestors. Which one?
- Suppose that *x* has a successor *y*. Then, *y* is the lowest ancestor of *x* whose left child is also an ancestor of *x*.
- To find *y*, we go up the tree from *x* until we encounter a node that is the left child of its parent. If no such node is encountered, there is no successor.



Inorder successor of:

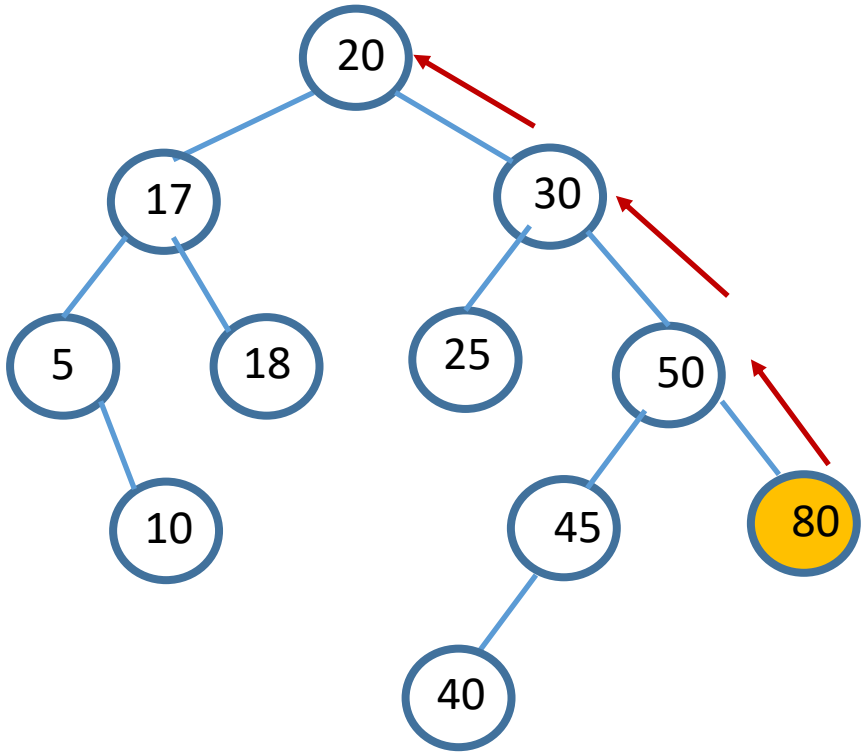
- 10 is 17
- 18 is 20

5, 10, 17, 18, 20, 25, 30, 40, 45, 50, 80

Finding the Inorder Successor

Case 2:

- Node has an empty right subtree e.g., 10, 18, 25, 45, 80)
- The inorder successor is one of its ancestors. Which one?
- Suppose that *x* has a successor *y*. Then, *y* is the lowest ancestor of *x* whose left child is also an ancestor of *x*.
- To find *y*, we go up the tree from *x* until we encounter a node that is the left child of its parent. If no such node is encountered, there is no successor.



Inorder successor of:

- 10 is 17
- 18 is 20
- 80 is ???

5, 10, 17, 18, 20, 25, 30, 40, 45, 50, 80

Code to Find the Inorder Successor

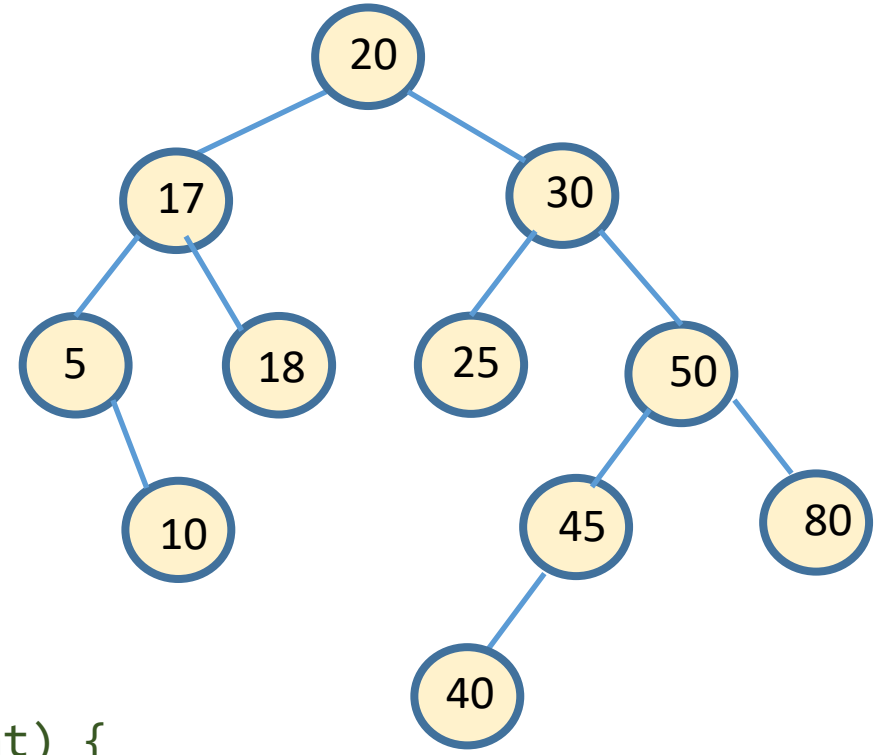
Case 1 and Case 2:

```
BTNode * inOrderSuccessor (BTNode * node) {
        if (node == NULL)
                return NULL;

        if (node->right != NULL)
                return minimum (node->right);

        BTNode * parent;

        parent = node->parent;
        while (parent != NULL && node == parent->right) {
                node = parent;
                parent = parent->parent;
        }
        return parent;
}
```



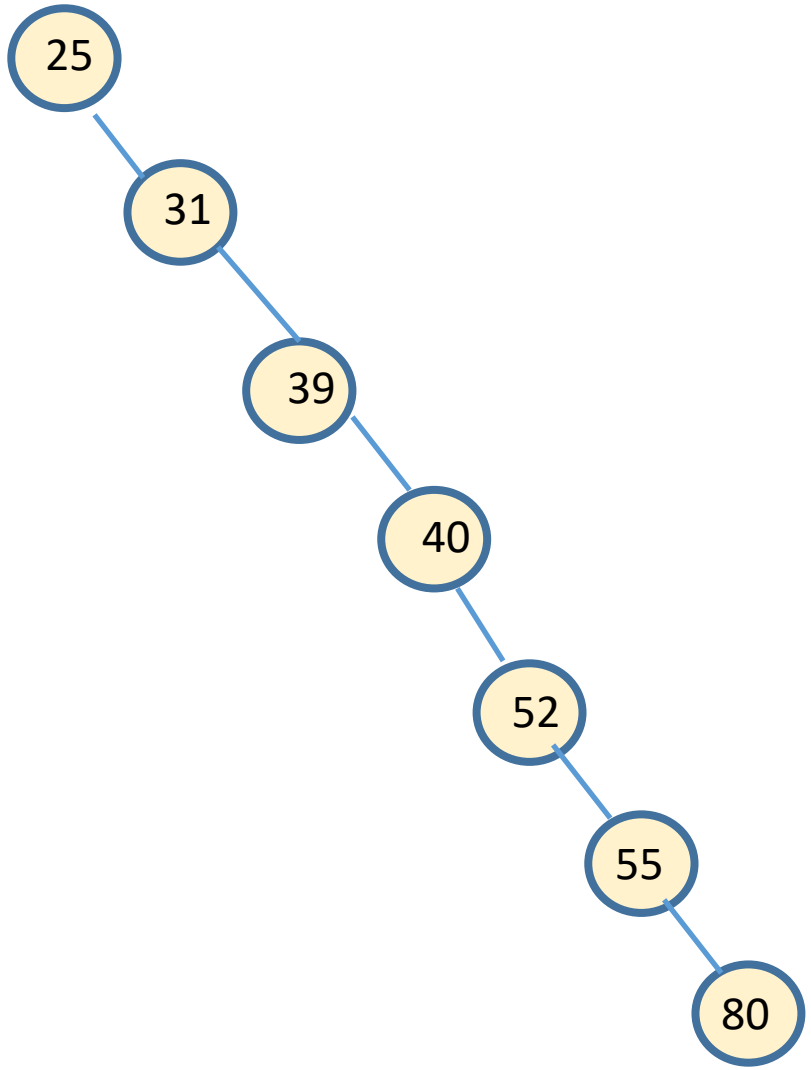
5, 10, 17, 18, 20, 25, 30, 40, 45, 50, 80

Degenerate BSTs

- Create a BST and insert the following elements in the order given:

```
25, 31, 39, 40, 52, 55, 80
```

- The resulting BST is called *degenerate*. Why do you think so?

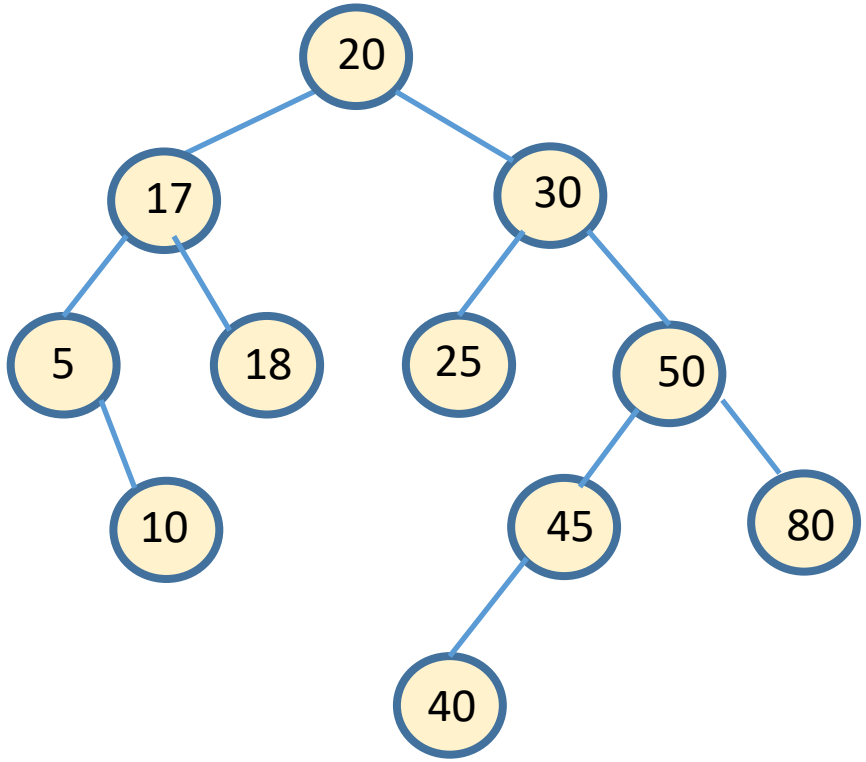


Deleting a Node from a BST

There are three cases to consider:

Case 1:

- Node is a leaf (e.g., 10, 18, 25, 40, 80)

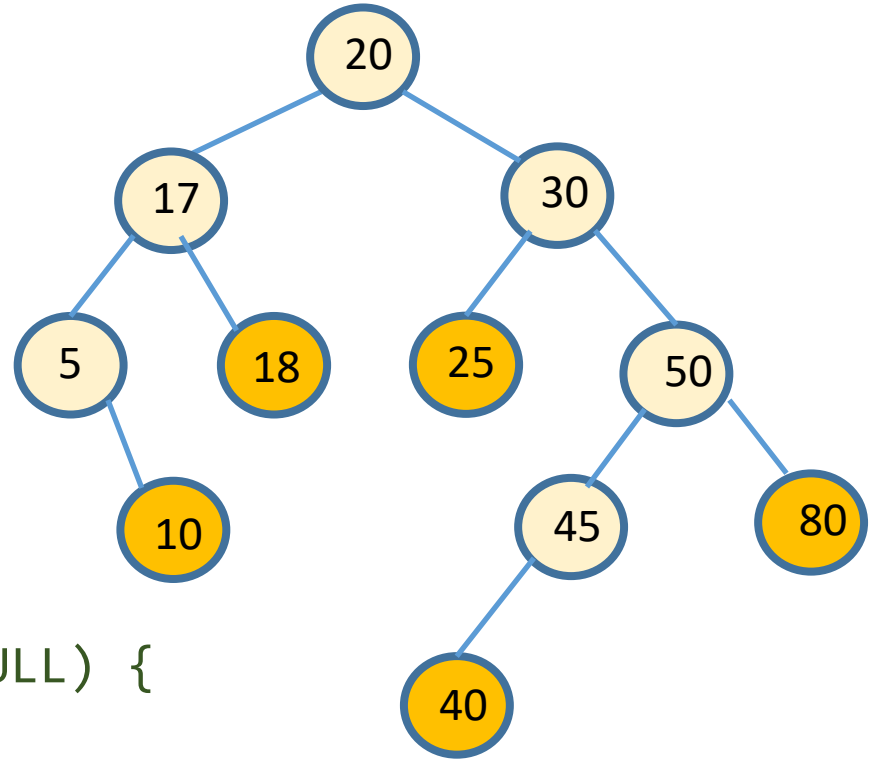


Deleting a Node from a BST

There are three cases to consider:

Case 1:

- Node is a leaf (e.g., 10, 18, 25, 40, 80)



- Go to the parent of the node. Set its left pointer or right pointer to NULL.

```
if (node->left == NULL && node->right == NULL) {
        BTNode * parent = node->parent;
        if (parent->left == node)
                parent->left = NULL;
        else
                parent->right = NULL;
}
```

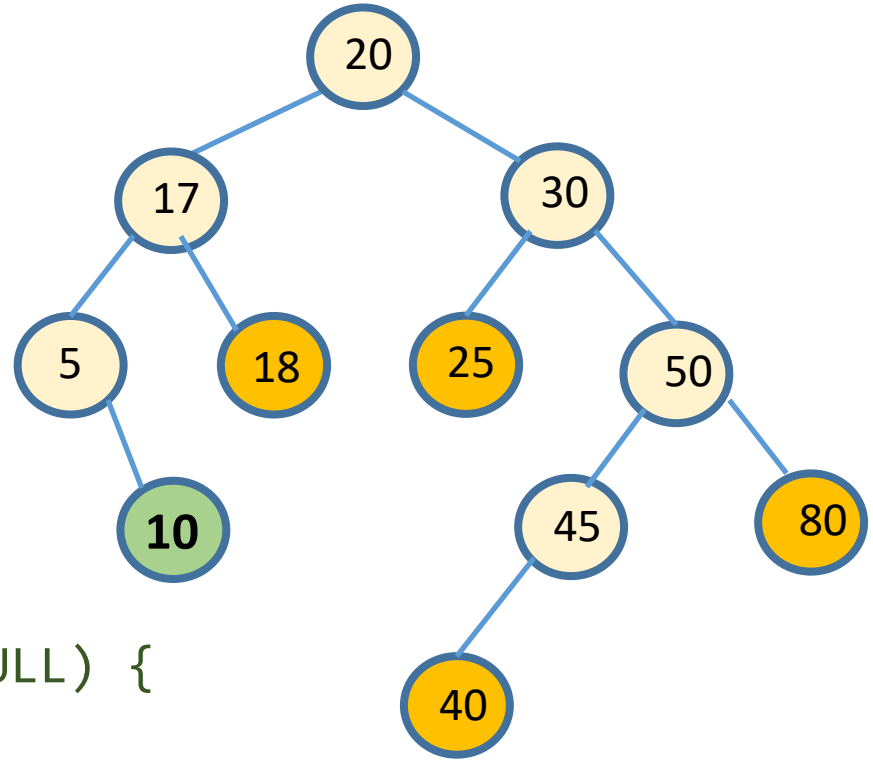
Deleting a Node from a BST

There are three cases to consider:

Case 1:

- Node is a leaf (e.g., 10, 18, 25, 40, 80)

- Go to the parent of the node. Set its left pointer or right pointer to NULL.




```
if (node->left == NULL && node->right == NULL) {
        BTNode * parent = node->parent;
        if (parent->left == node)
                parent->left = NULL;
        else
                parent->right = NULL;
}
```

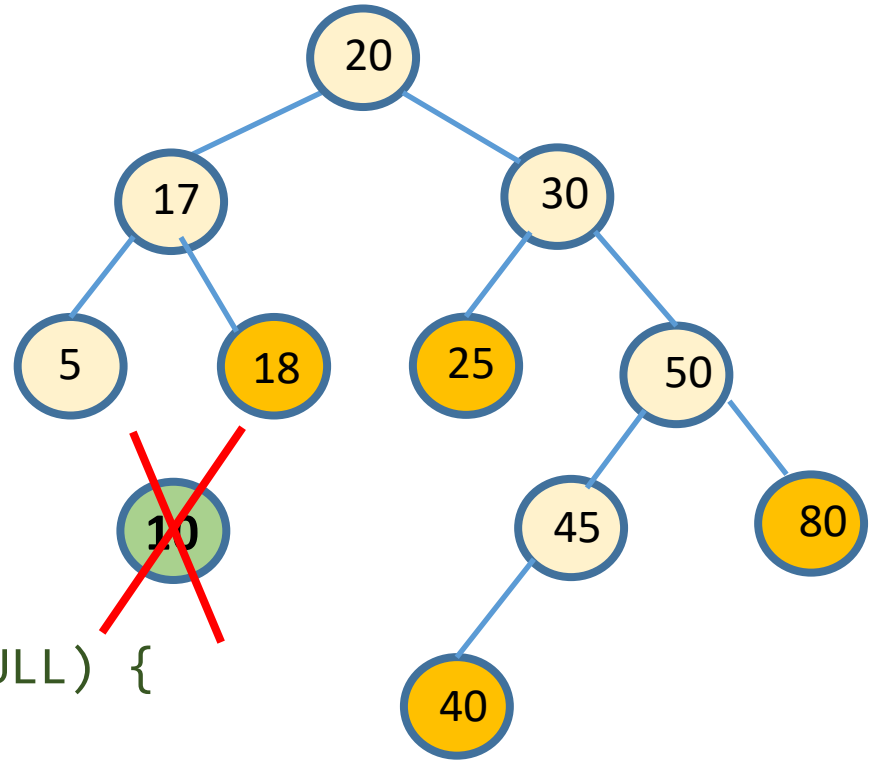
Deleting a Node from a BST

There are three cases to consider:

Case 1:

- Node is a leaf (e.g., 10, 18, 25, 40, 80)

- Go to the parent of the node. Set its left pointer or right pointer to NULL.



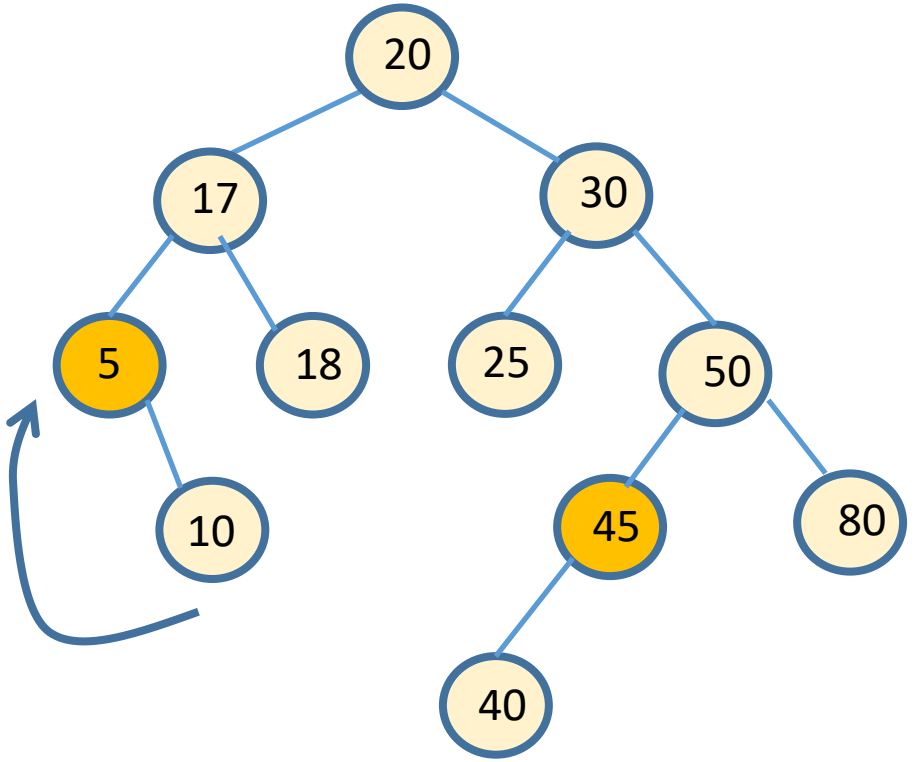

```
if (node->left == NULL && node->right == NULL) {
        BTNode * parent = node->parent;
        if (parent->left == node)
                parent->left = NULL;
        else
                parent->right = NULL;
}
```

Deleting a Node from a BST

Case 2:

- (a) Node has no left subtree (e.g., 5)
- (b) Node has no right subtree (e.g., 45)

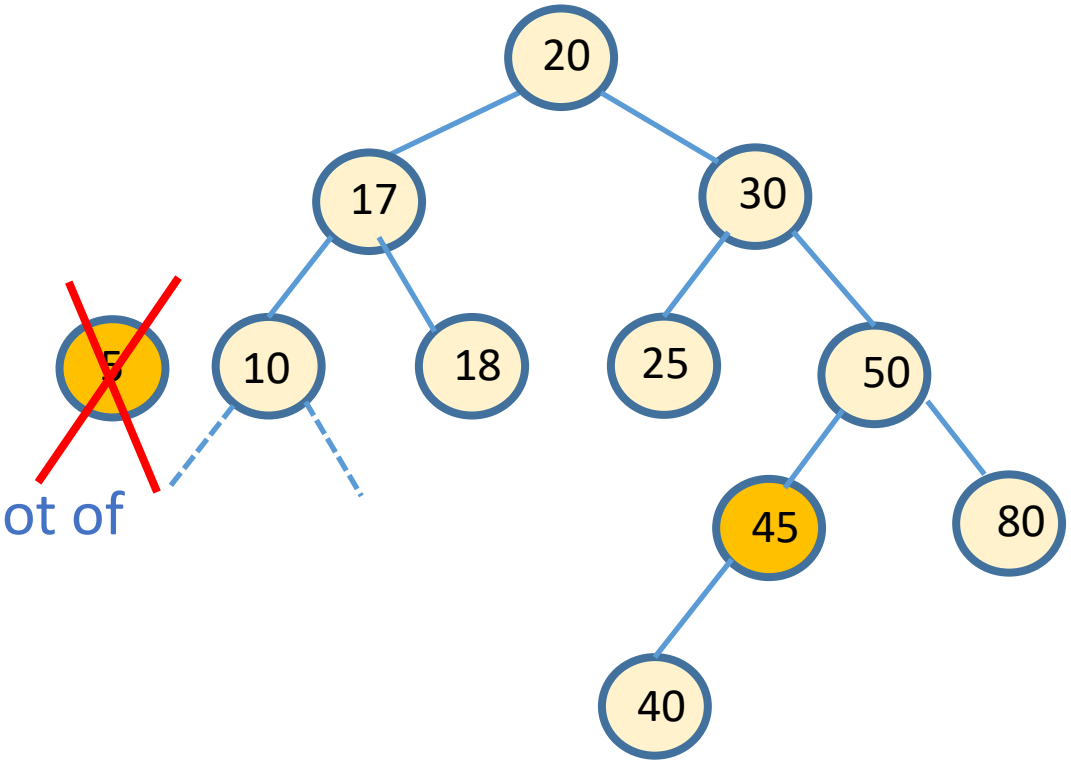
- Deletion of (a): Replace node with the root of its right subtree (e.g., 10 replaces 5)



Deleting a Node from a BST

Case 2:

- (a) Node has no left subtree (e.g., 5)
- (b) Node has no right subtree (e.g., 45)



- Deletion of (a): Replace node with the root of its right subtree (e.g., 10 replaces 5)

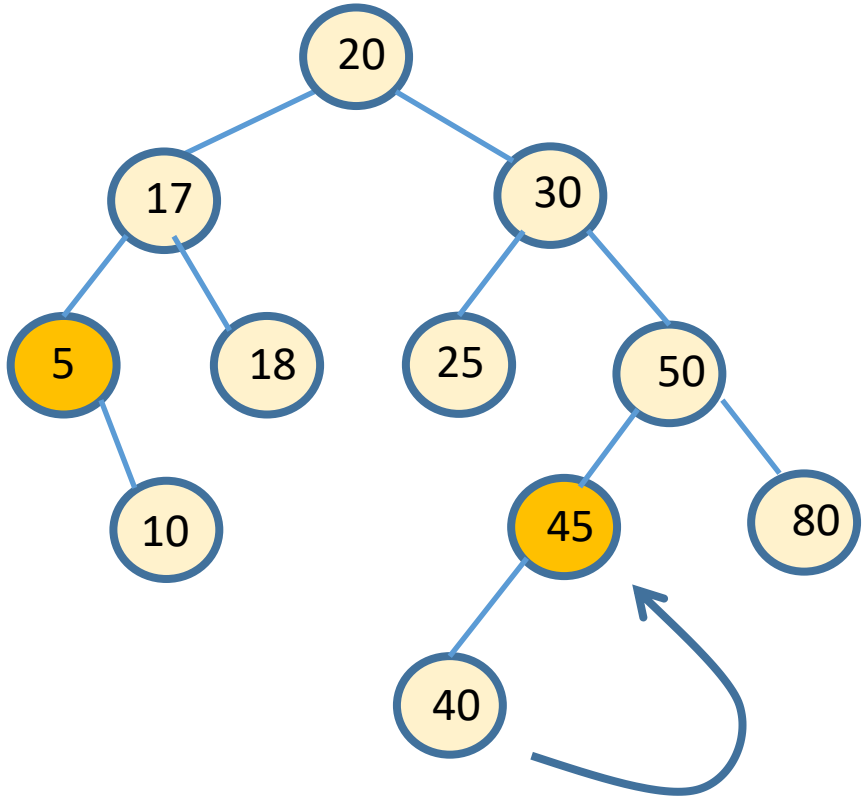
- This works, even if the node with 10 has children.

Deleting a Node from a BST

Case 2:

- (a) Node has no left subtree (e.g., 5)
- (b) Node has no right subtree (e.g., 45)

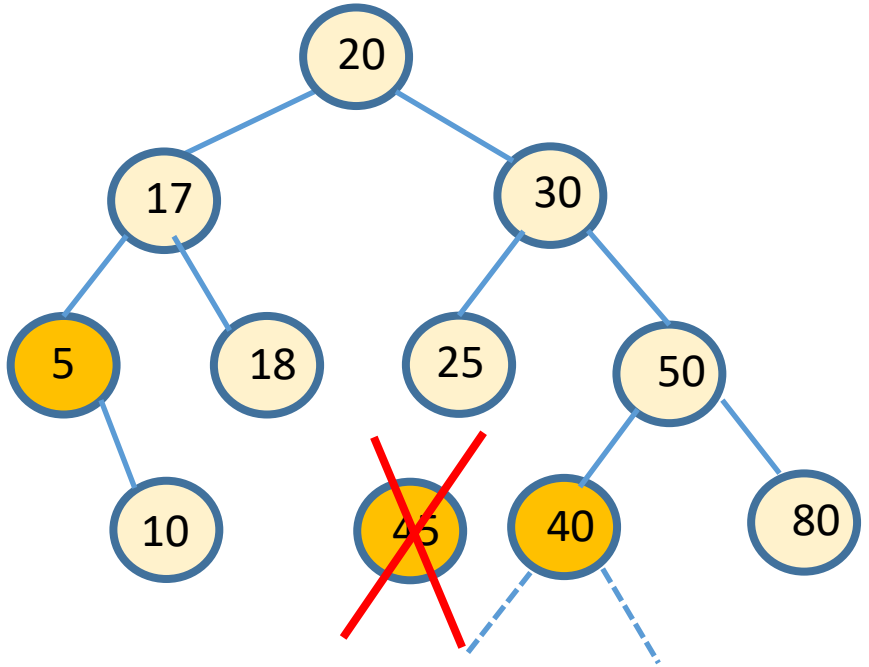
- Deletion of (b): Replace node with the root of its left subtree (e.g., 40 replaces 45)



Deleting a Node from a BST

Case 2:

- (a) Node has no left subtree (e.g., 5)
- (b) Node has no right subtree (e.g., 45)



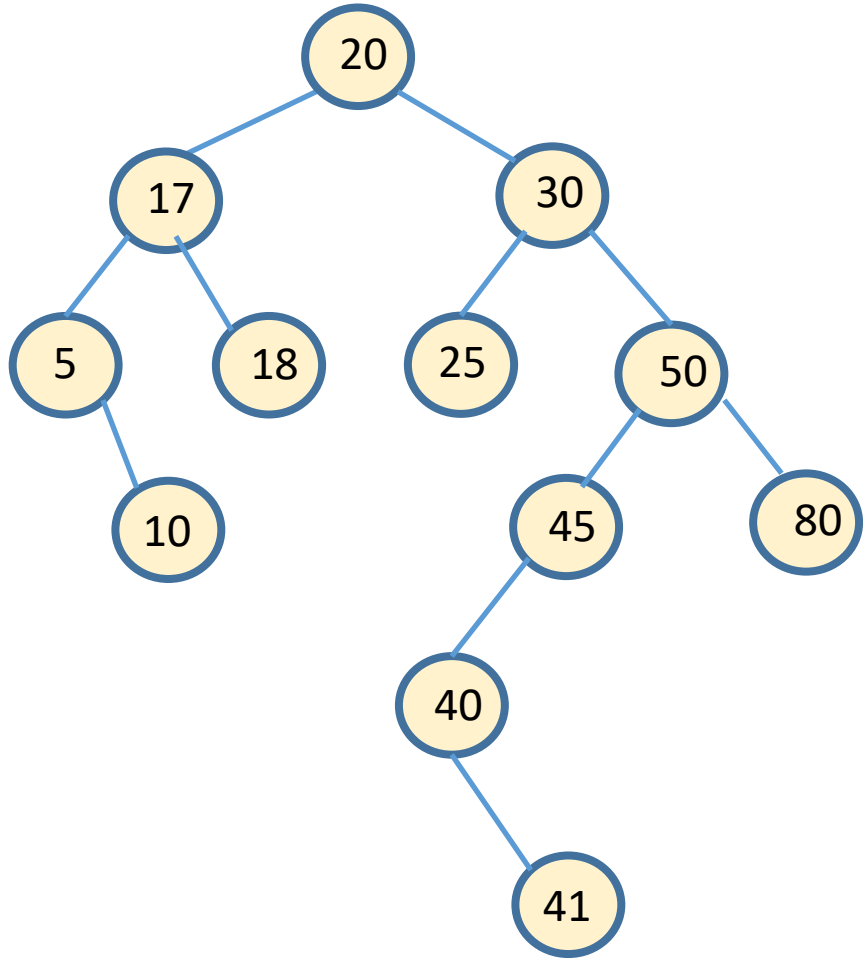
- Deletion of (b): Replace node with the root of its left subtree (e.g., 40 replaces 45)

- This works, even if the node with 40 has children.

Deleting a Node from a BST

Case 3:

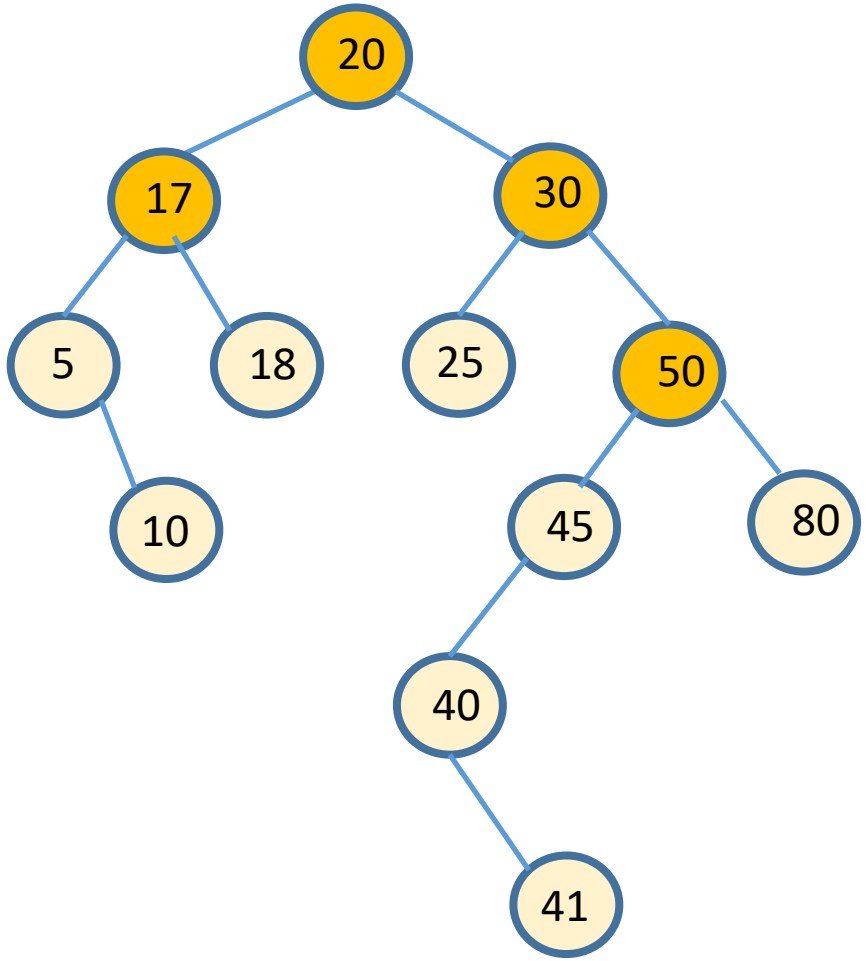
- Node has non-empty left and right subtrees (e.g., 17, 20, 30, 50)



Deleting a Node from a BST

Case 3:

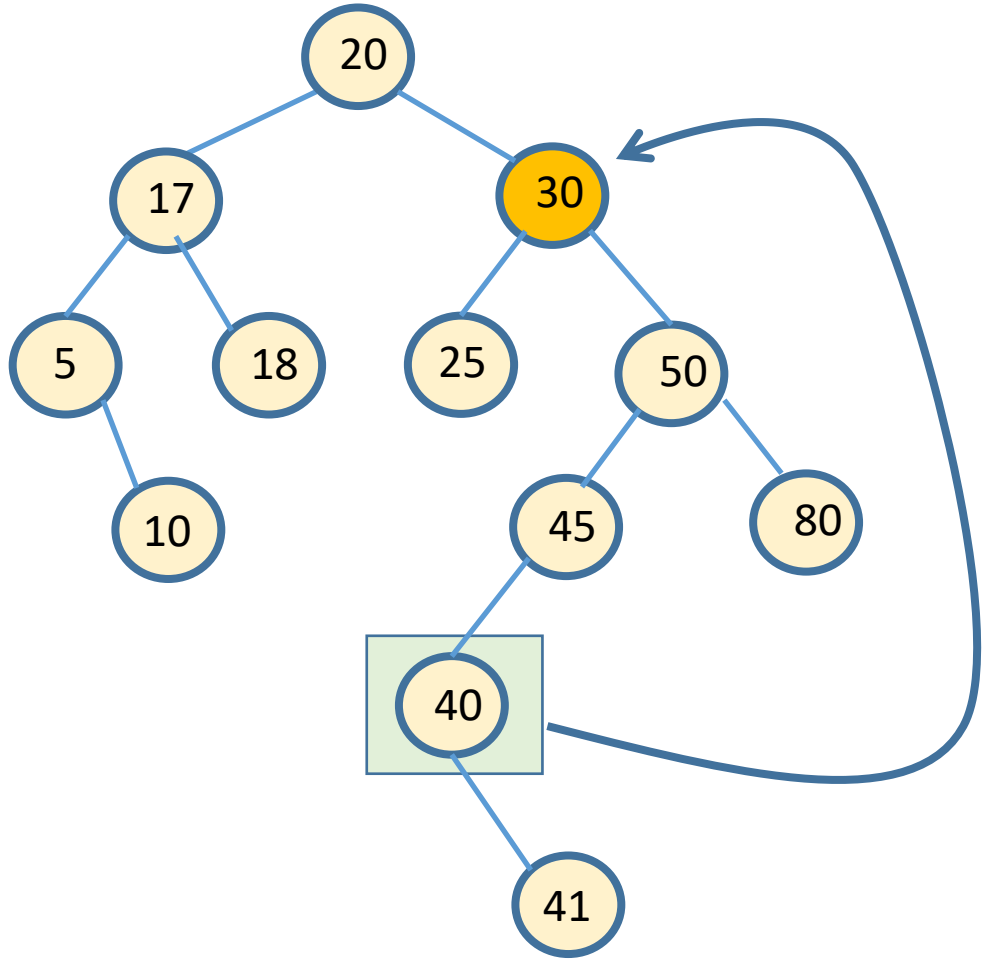
- Node has non-empty left and right subtrees (e.g., 17, 20, 30, 50)
- Suppose 30 is to be deleted.



Deleting a Node from a BST

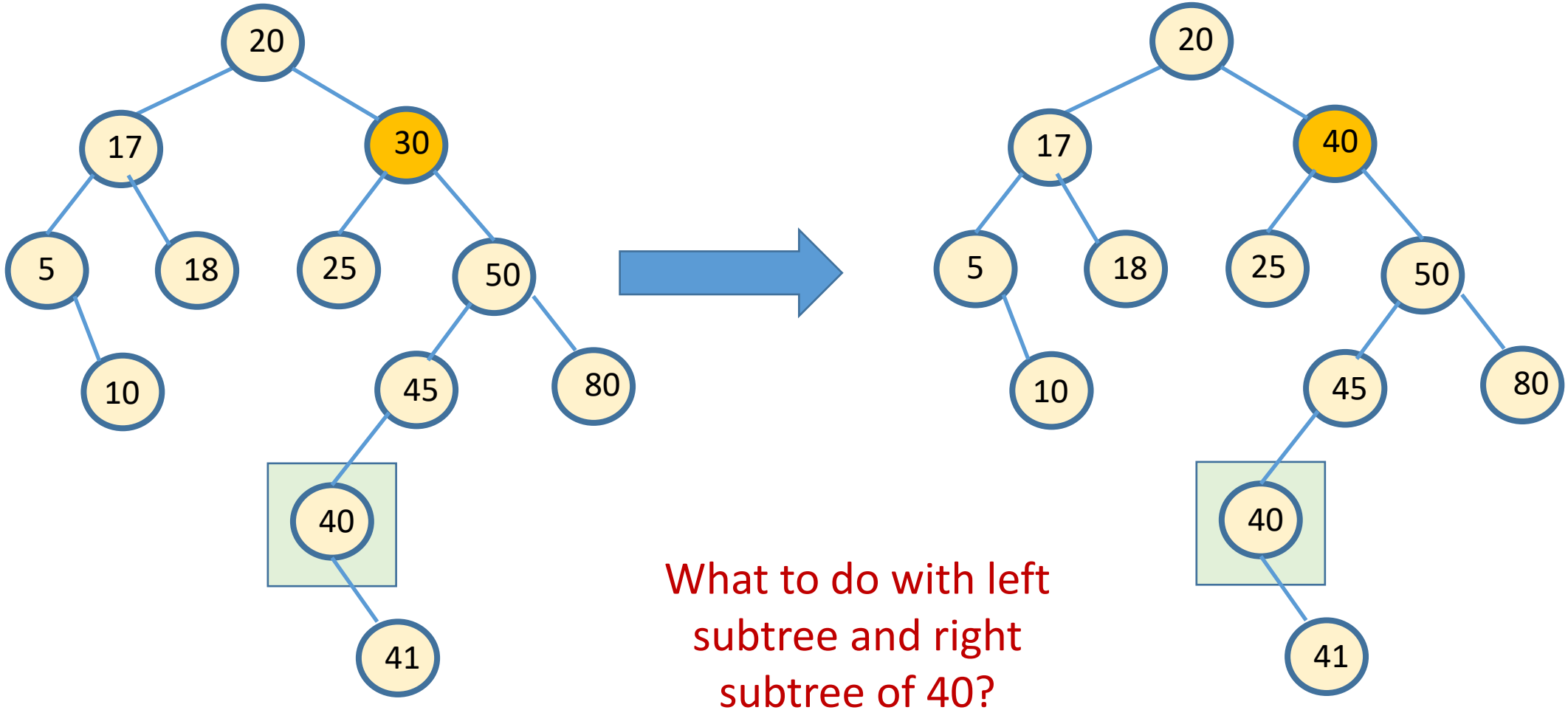
Case 3:

- Node has non-empty left and right subtrees (e.g., 17, 20, 30, 50)
- Suppose 30 is to be deleted.
- Find the inorder successor of 30. What is it?
- Copy the data in 40 to 30



Deleting a Node from a BST

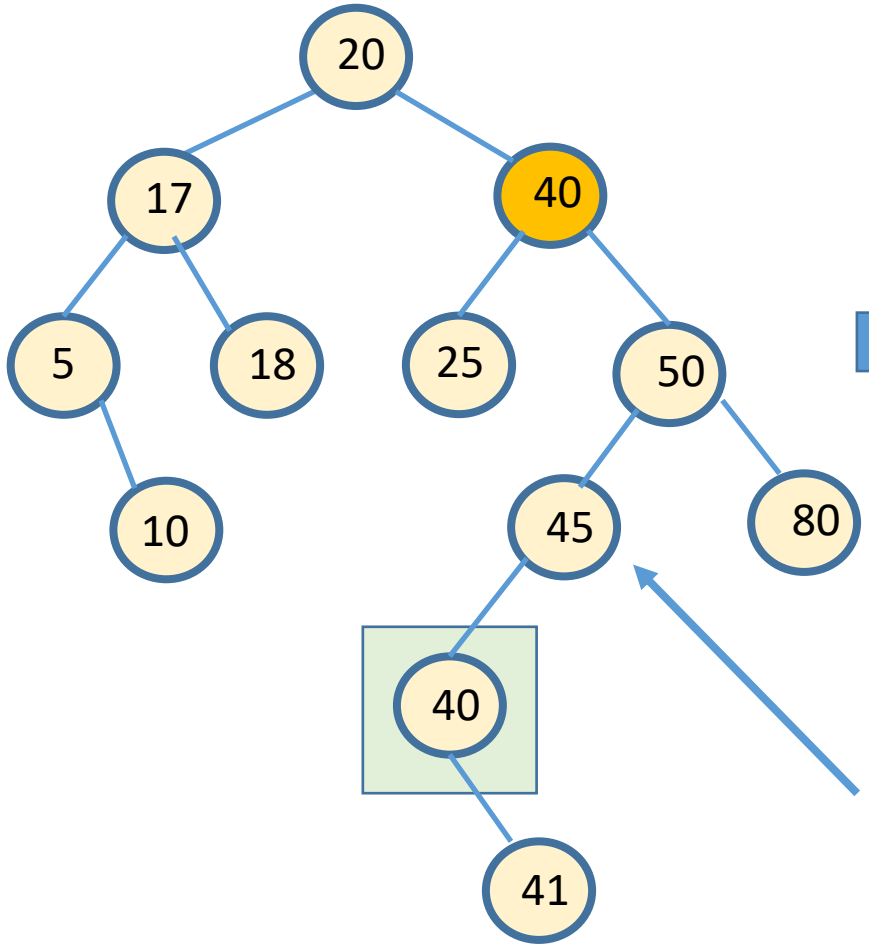
Case 3: Suppose 30 has to be deleted.

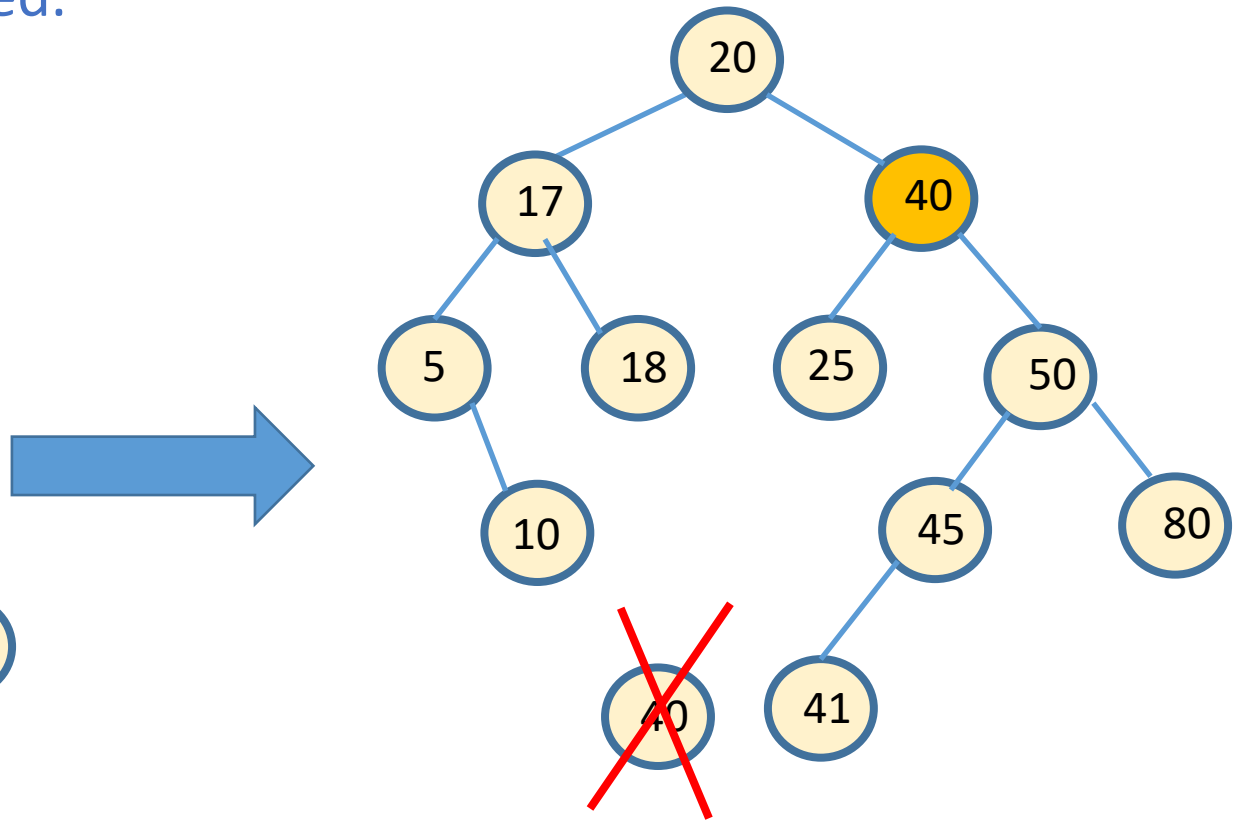


What to do with left subtree and right subtree of 40?

Deleting a Node from a BST

Case 3: Suppose 30 has to be deleted.

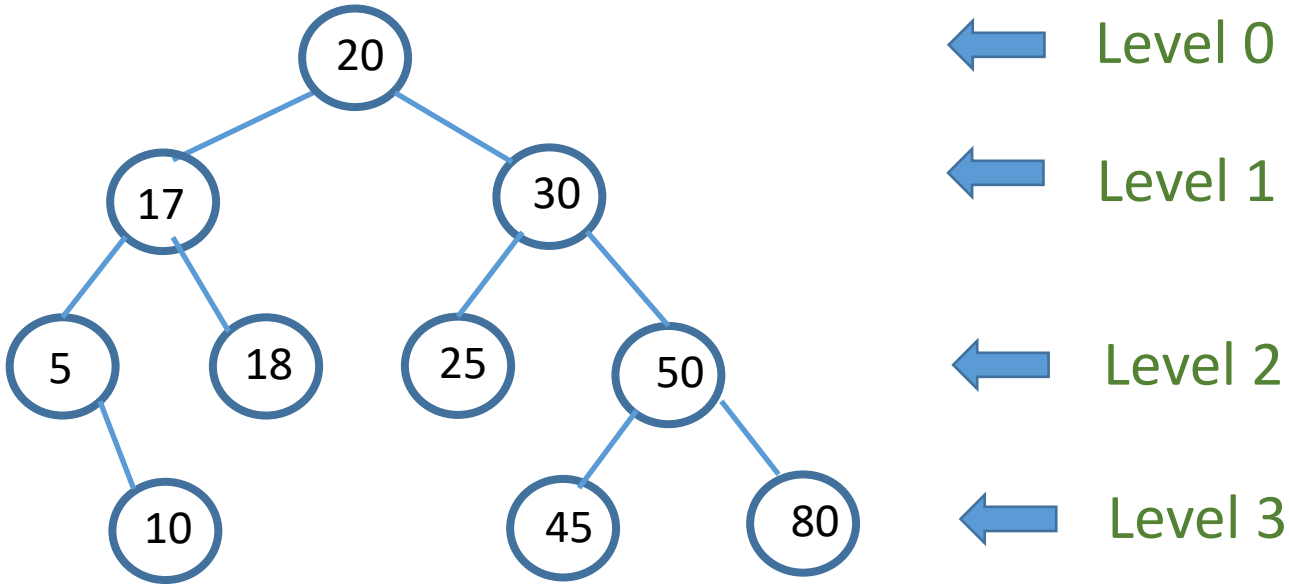




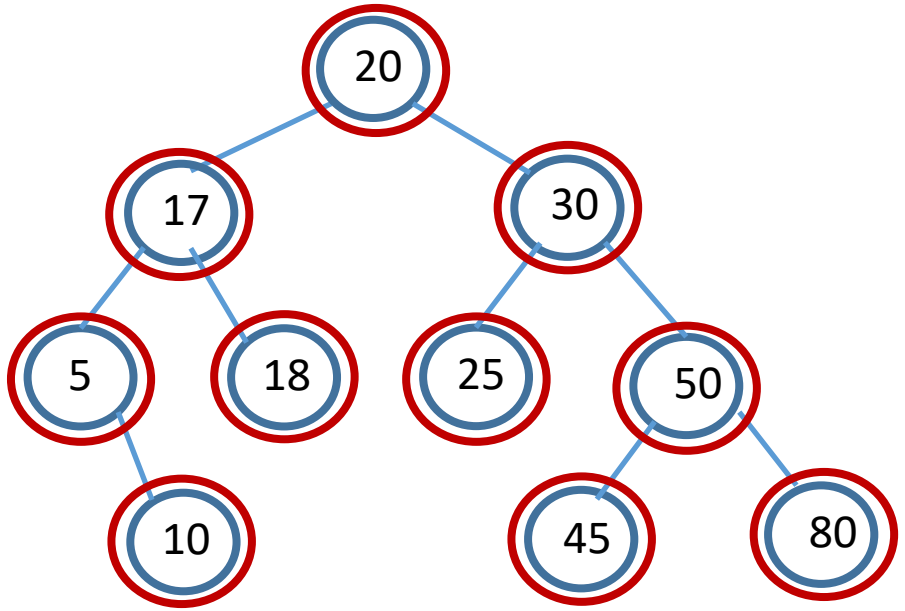
Connect right subtree of 40 to parent of 40.

Delete the original 40

Level Order Traversal of a Binary Tree



Level Order Traversal of a Binary Tree



A level order traversal would visit the nodes in the order: 20, 17, 30, 5, 18, 25, 50, 10, 45, 80.