ICS 240
Introduction to Data Structures

Jessica Maistrovich

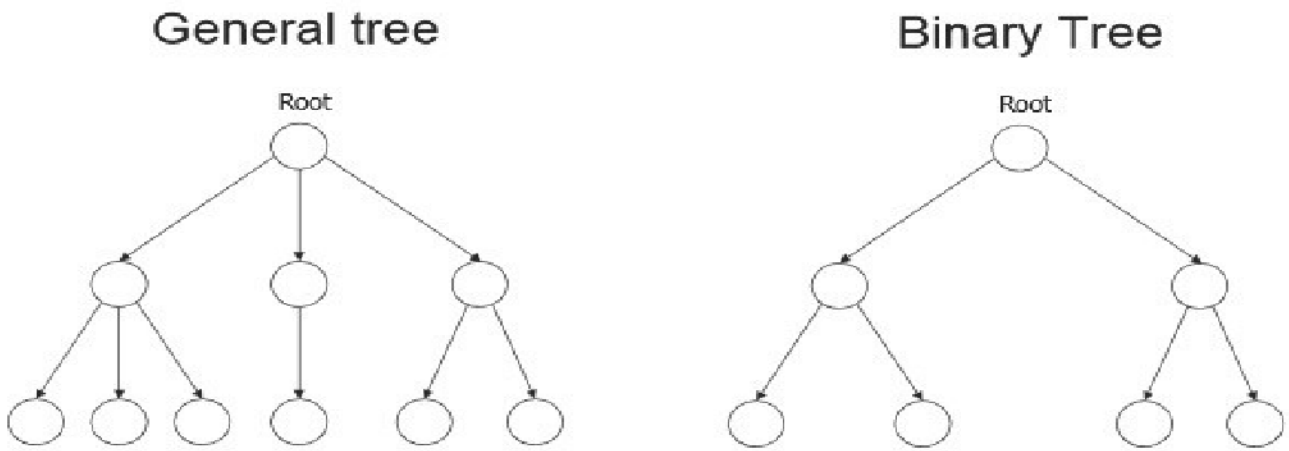
Metropolitan State University

Implementing a Binary Tree

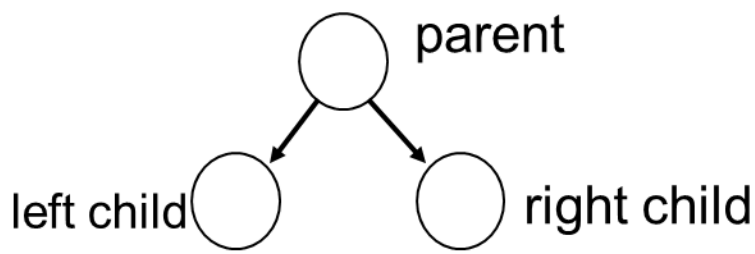
Two possibilities: Array or Nodes

Recall Binary Trees

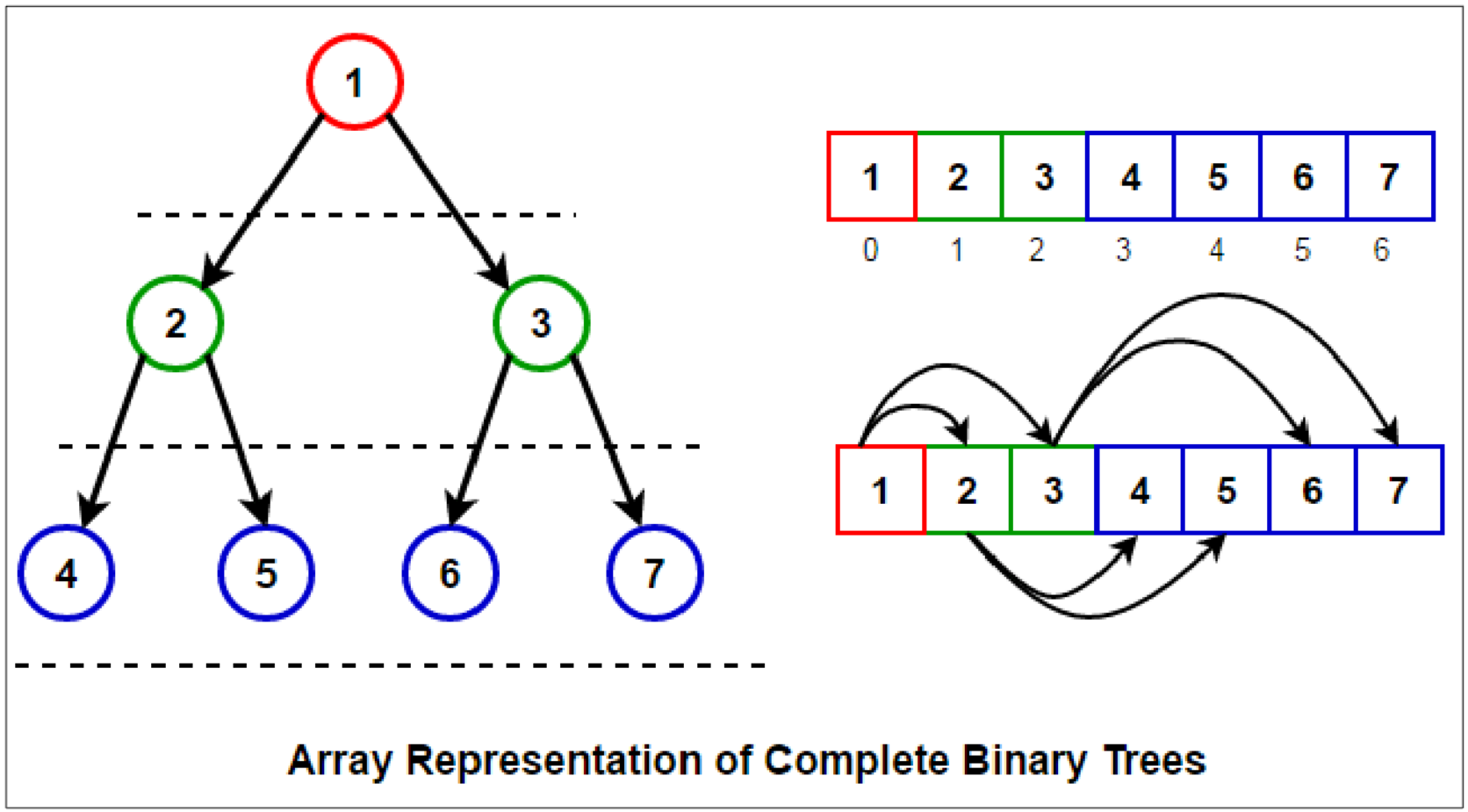
- A binary tree is a special type of tree where each node can have **two** children at most



- Usually called leftChild and rightChild

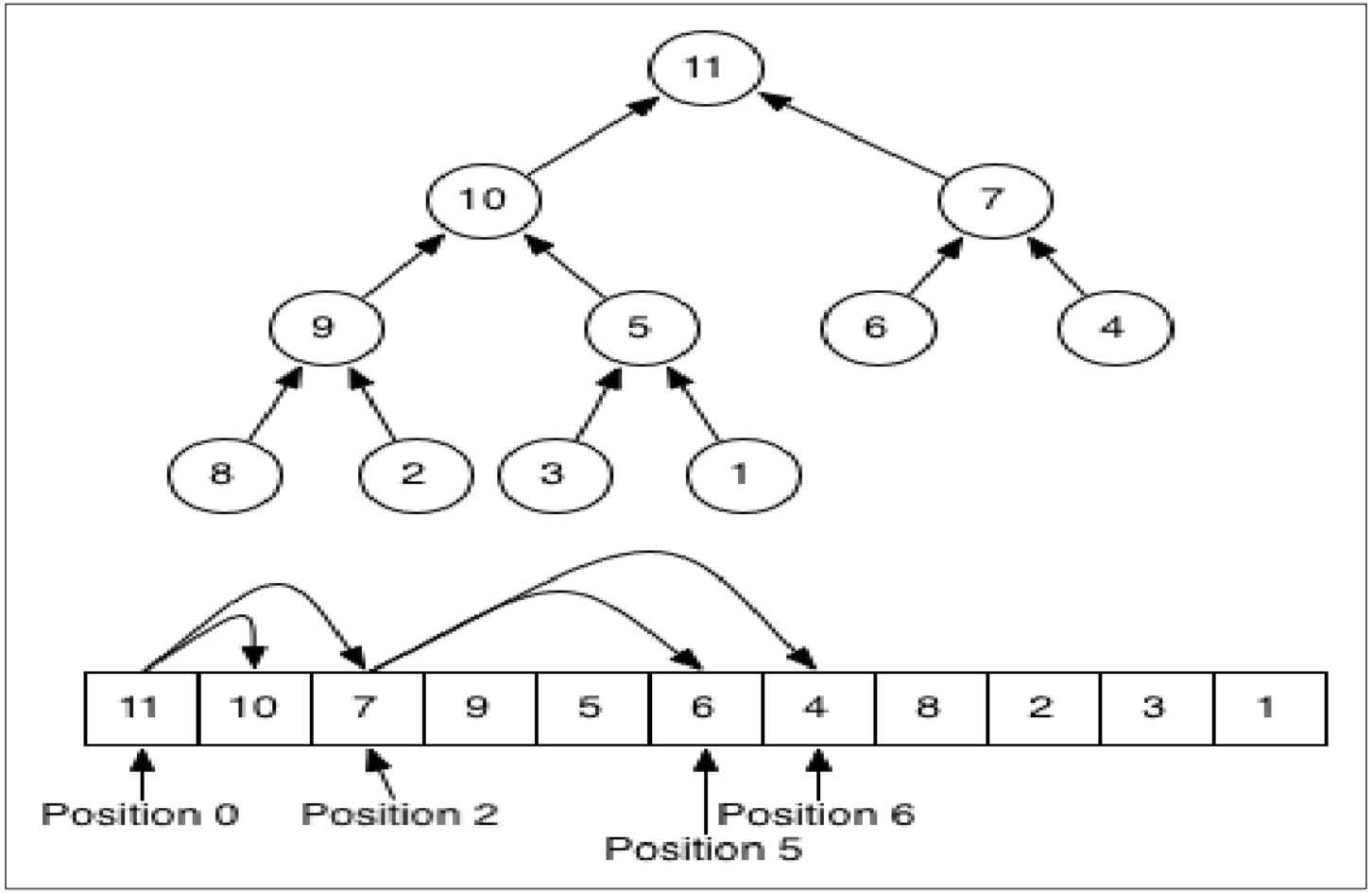


Array Representation of **Complete** Binary Trees



- For any node at index `i`:
  - left child is at `[2i+1]`
  - Right child is at `[2i+2]`
  - parent is at `[(i-1)/2]`

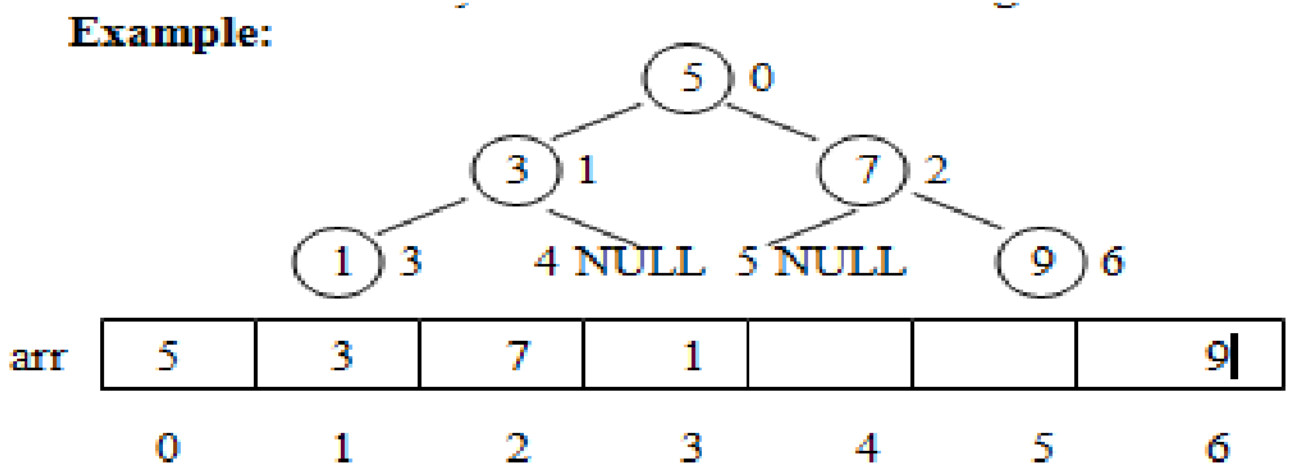
Example



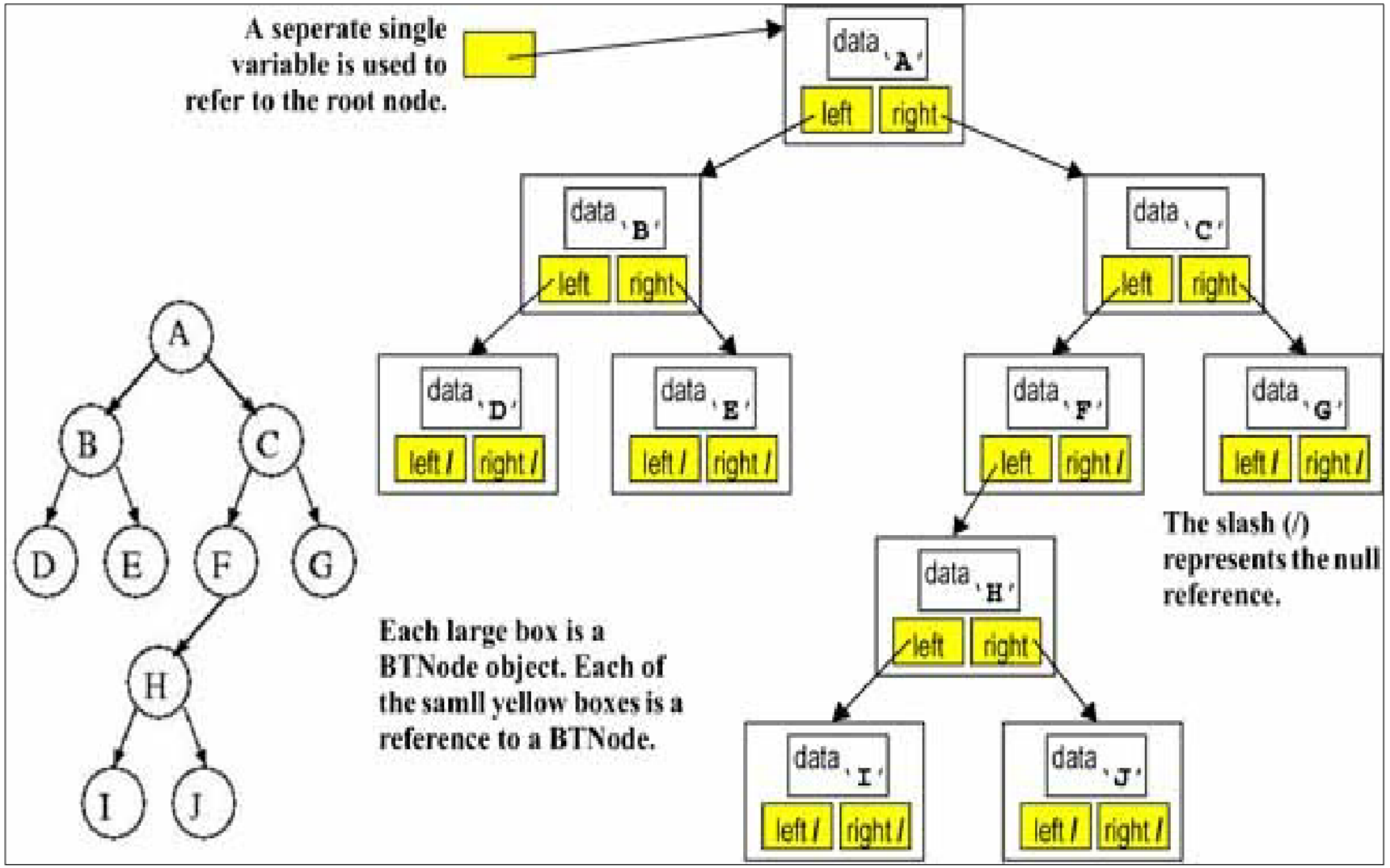
- Consider node with value 7:
  - 7 is stored at array index 2
  - Its left child is at `2*2+1 = 5`
  - Its right child is at `2*2+2 = 6`
  - Its parent is at `(2-1)/2 = 0`

Can Arrays Be Used to Represent Incomplete Trees?

- It is hard to use arrays to represent incomplete trees because there will be many empty slots



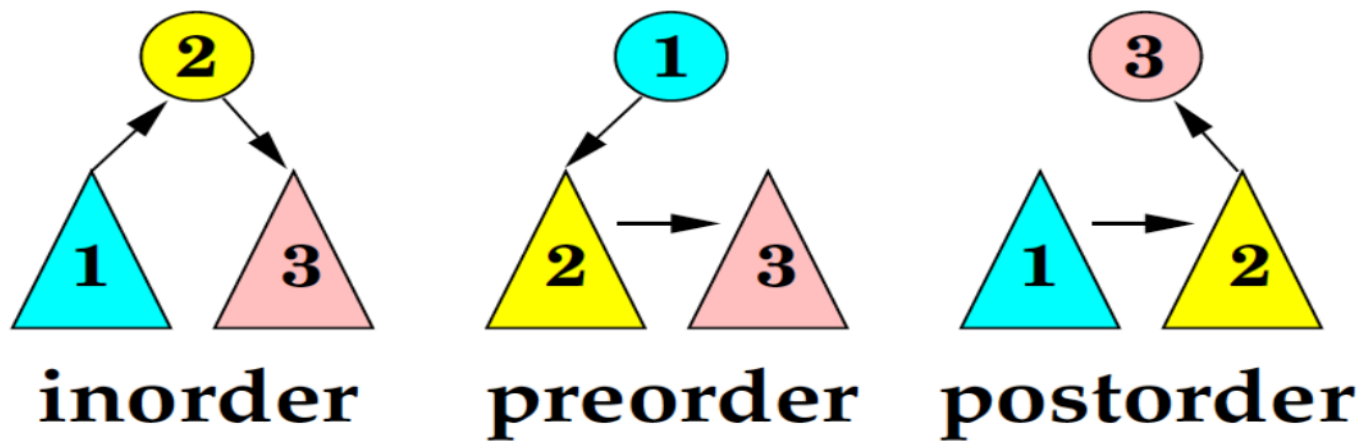
Tree Representation using Nodes



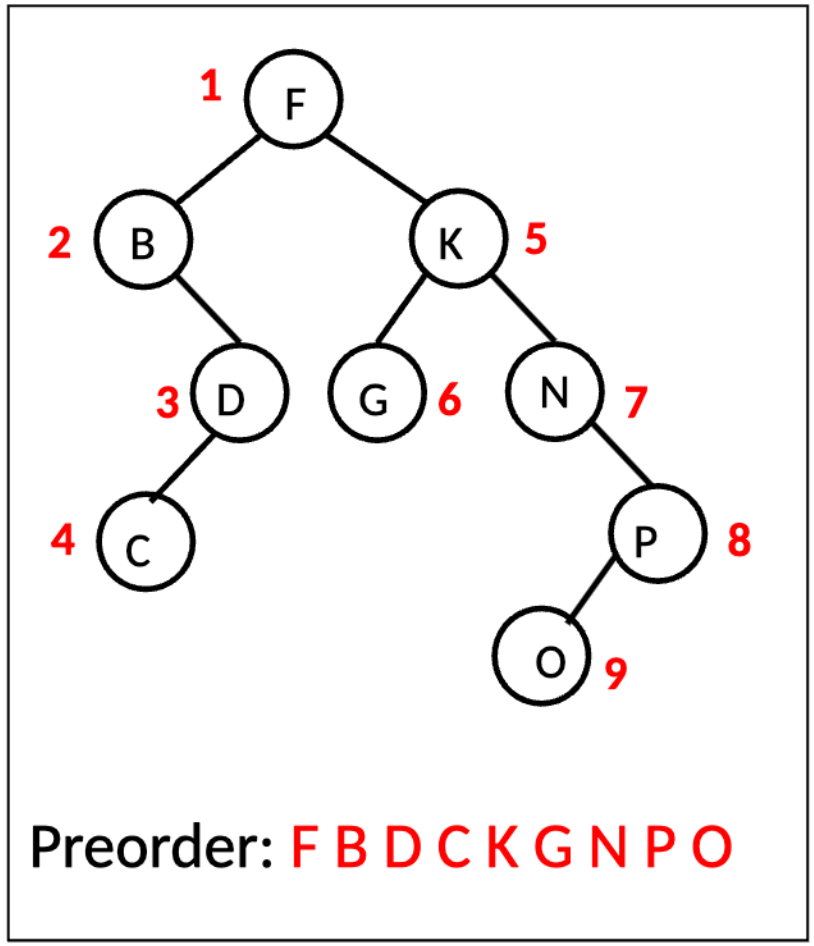
Binary Tree Traversals

Binary Tree Traversals

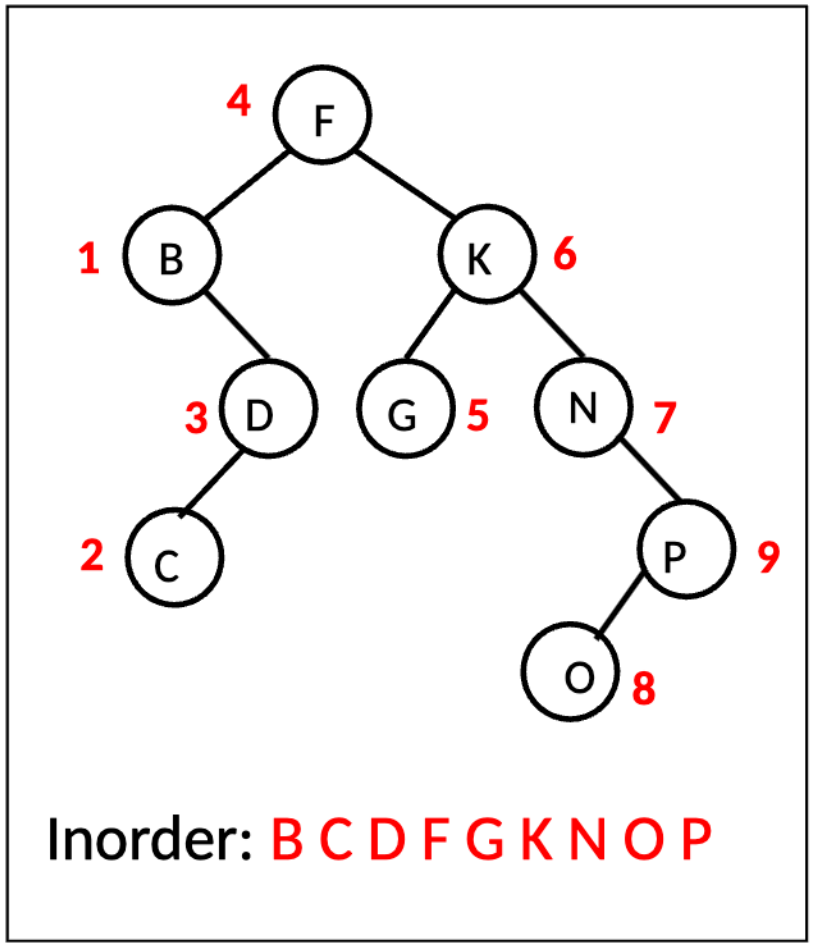
- Tree traversal is the process of visiting each node in the tree exactly once
- There are three commonly used types of tree traversals
  - **preorder** traversal = root, left, right
  - **inorder** traversal = left, root, right
  - **postorder** traversal = left, right, root



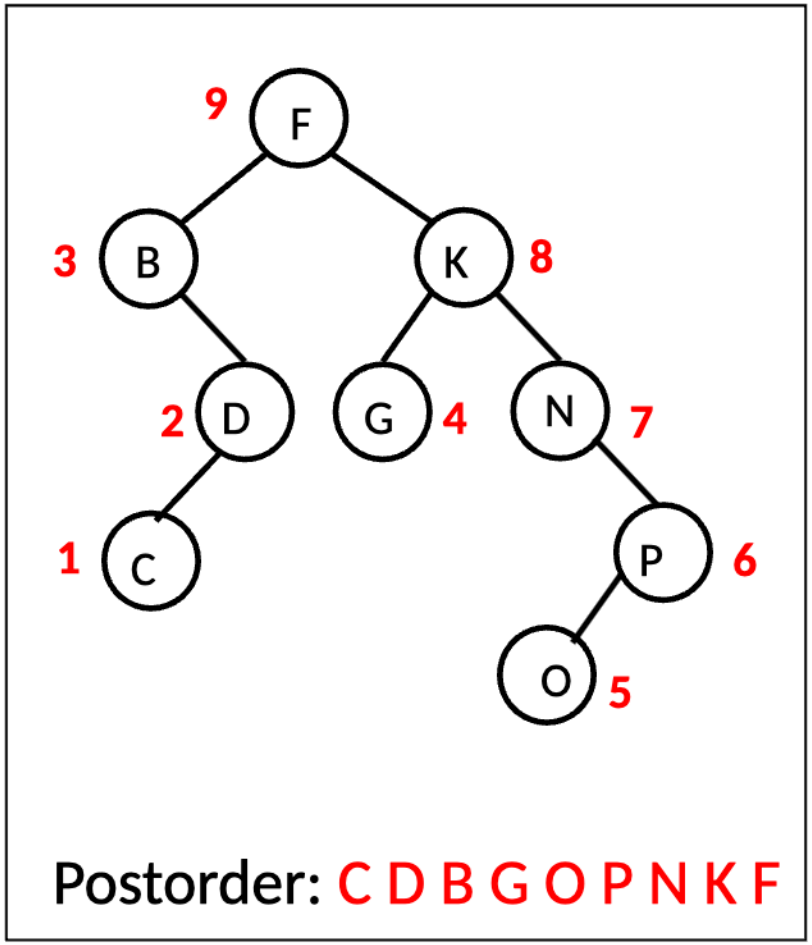
Preorder Traversal – node, left, right



Inorder Traversal – left, node, right



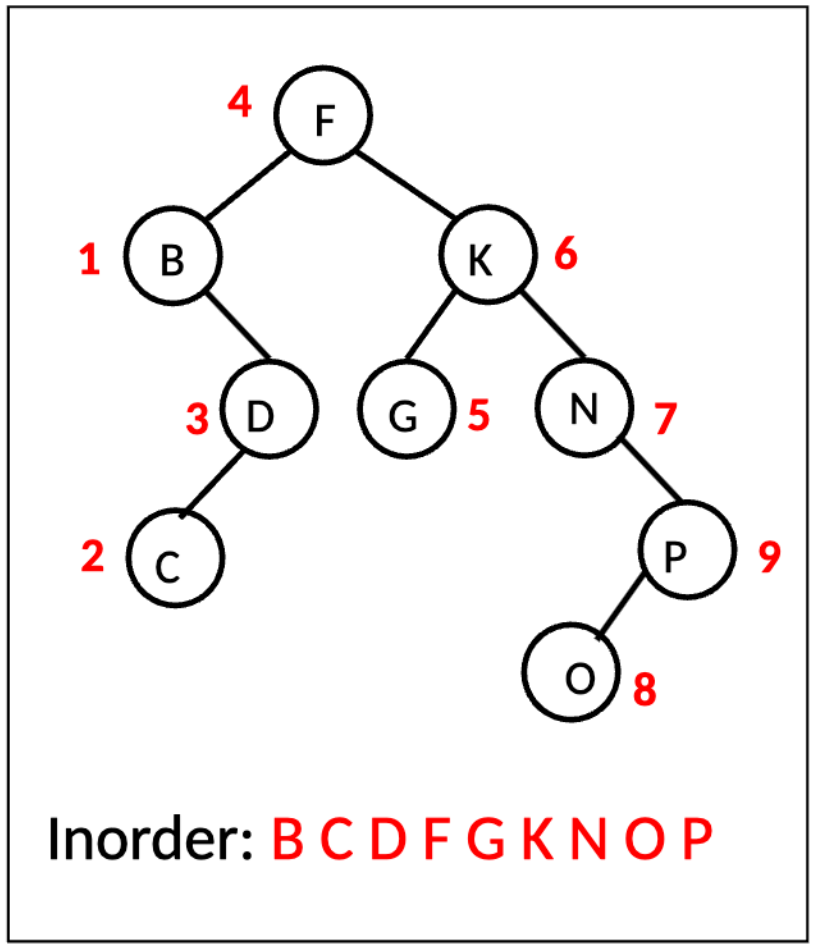
Postorder Traverals – left, right, node



Tree Traversal – `inorder()`

```
public void inorder()   {
      inorder(root);
}
```

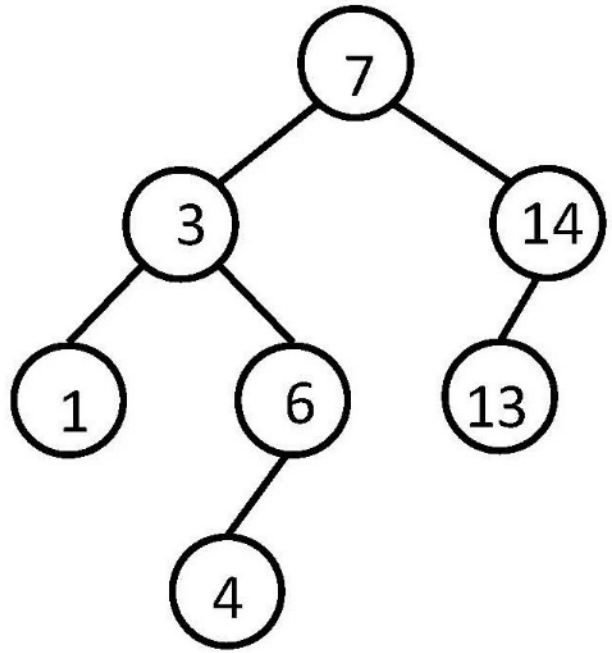
```
private void inorder(IntBTNode cur) {
   if (cur != null)     {
      inorder(cur.getLeft());

System.out.println(cur.getData());
      inorder(cur.getRight());
   }
}
```



Tree Traversal – `inorder()`

```
public void inorder()   {
      inorder(0);
}
```

```
private void inorder(int i){
   if (i < tree.length && tree[i] != null)
   {
      inorder (2*i+1);
      System.out.println(tree[i]);
      inorder (2*i+2);
   }
}
```



Inorder traversal: 1 3 4 6 7 13 14

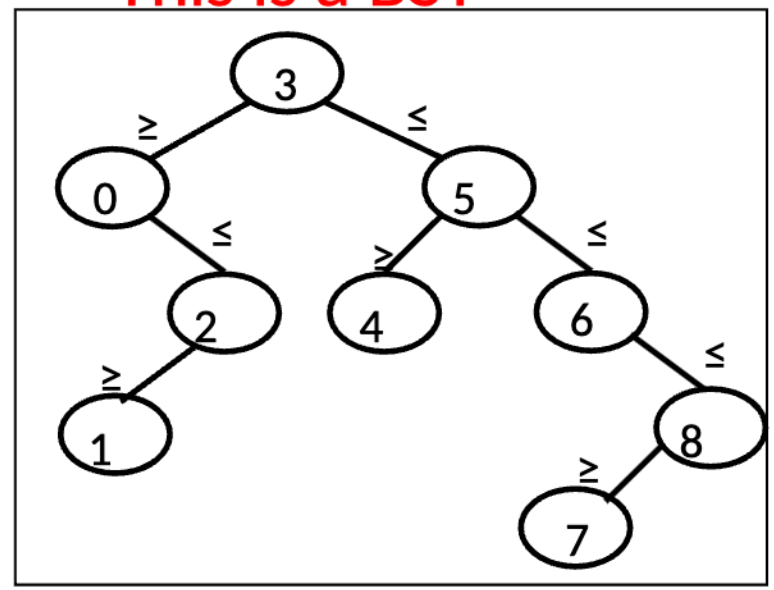
index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
contents	7	3	14	1	6	13	null	null	null	4	null	null	null	null	null	null

Binary Search Trees

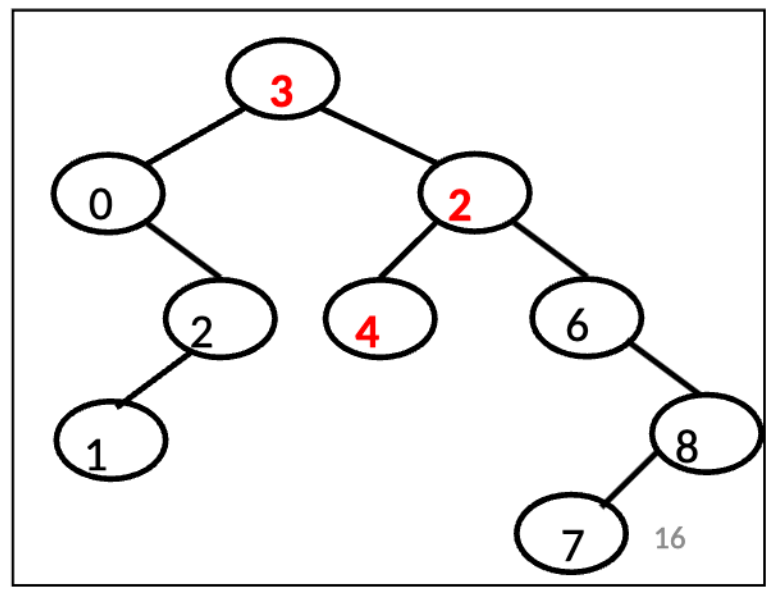
Binary Search Tree (BST)

- A Binary Search Tree (BST) is a binary tree such that, for any given node `n`:
  - All values in `n`'s left subtree are less than or equal to the value in `n`
  - All values in `n`'s right subtree are greater than the value in `n`
- BST's are usually used to implement sorted lists of elements
  - In a BST, we can implement `add()`, `remove()` and `search()` in `log(n)`

This is a BST



This is NOT a BST



Binary Search Tree Operations

- BST operations
  - `search()` or `countOccurrences()`
  - `add()`
  - `remove()`
- All BST operations are done in `O(BST height)`
  - Start at the root and then go down the tree
  - For a **balanced** tree with `height = log n` all operations can be performed in `O(log n)`
  - For a **degenerate** tree with `height = n` all operations are done in `O(n)`

Contrasting Various List Implementations

	`add`	`remove`	`search`
Array (unordered)	O(1)	O(n)	O(n)
Sorted array	O(n)	O(n)	O(log n)
Linked list (unordered)	O(1)	O(n)	O(n)
BST	O(h)	O(h)	O(h)

Searching a BST

BST Search Operation

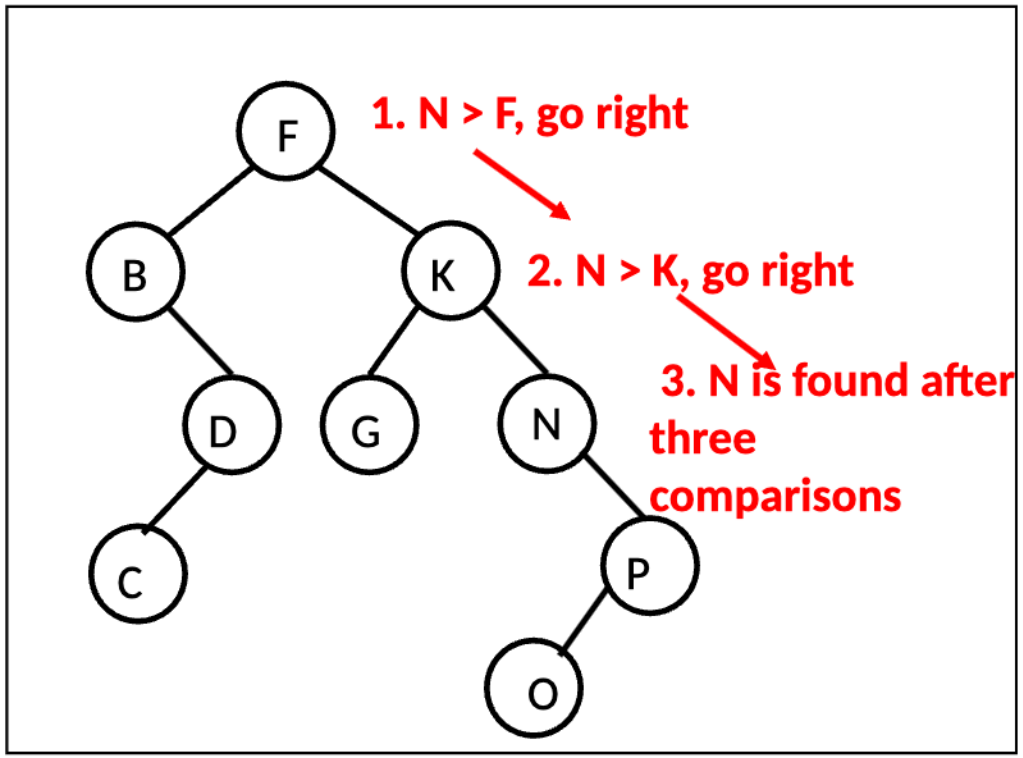
- To search for a value in a BST, start from the root and scan down until the value is found or you arrive at an empty subtree
- Descend using comparisons to make left/right decisions

```
if (search_value == node_value) return true (found)
else if (search_value < node_value)
    go to the left child
else if (search_value > node_value)
    go to the right child
```

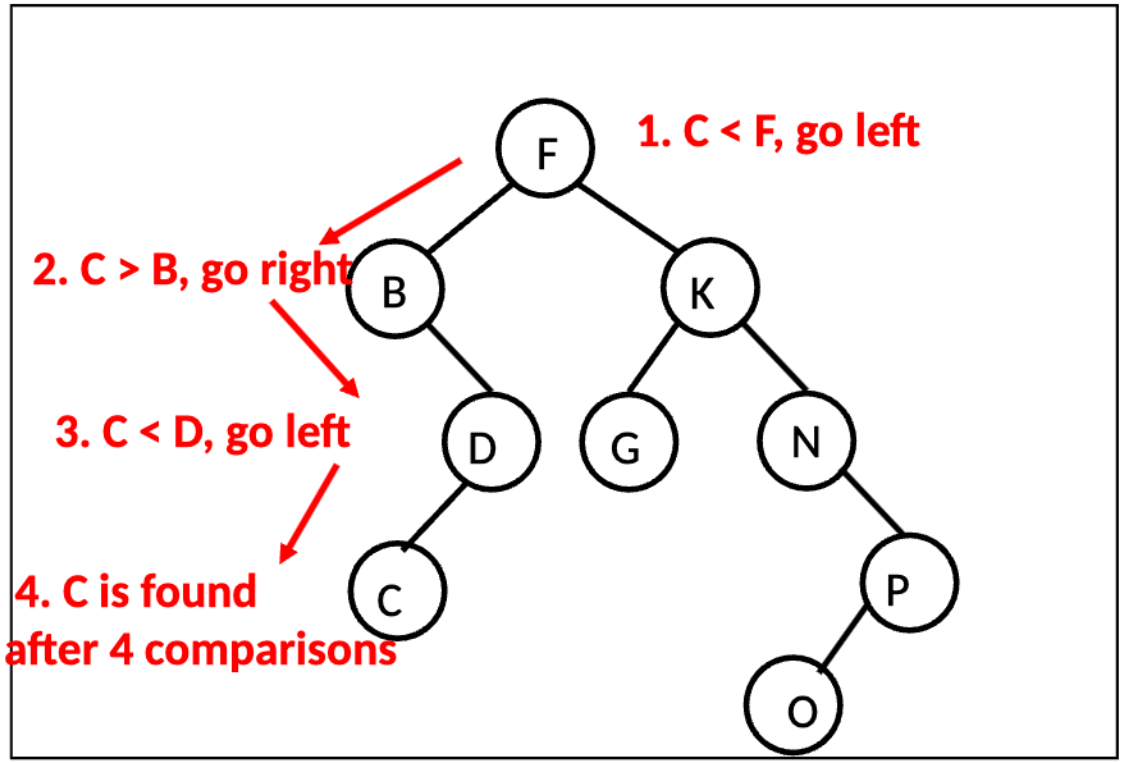
- Stop descending when move is impossible (no left or right child) and return false (or not found)

Examples of BST Search

Search for N



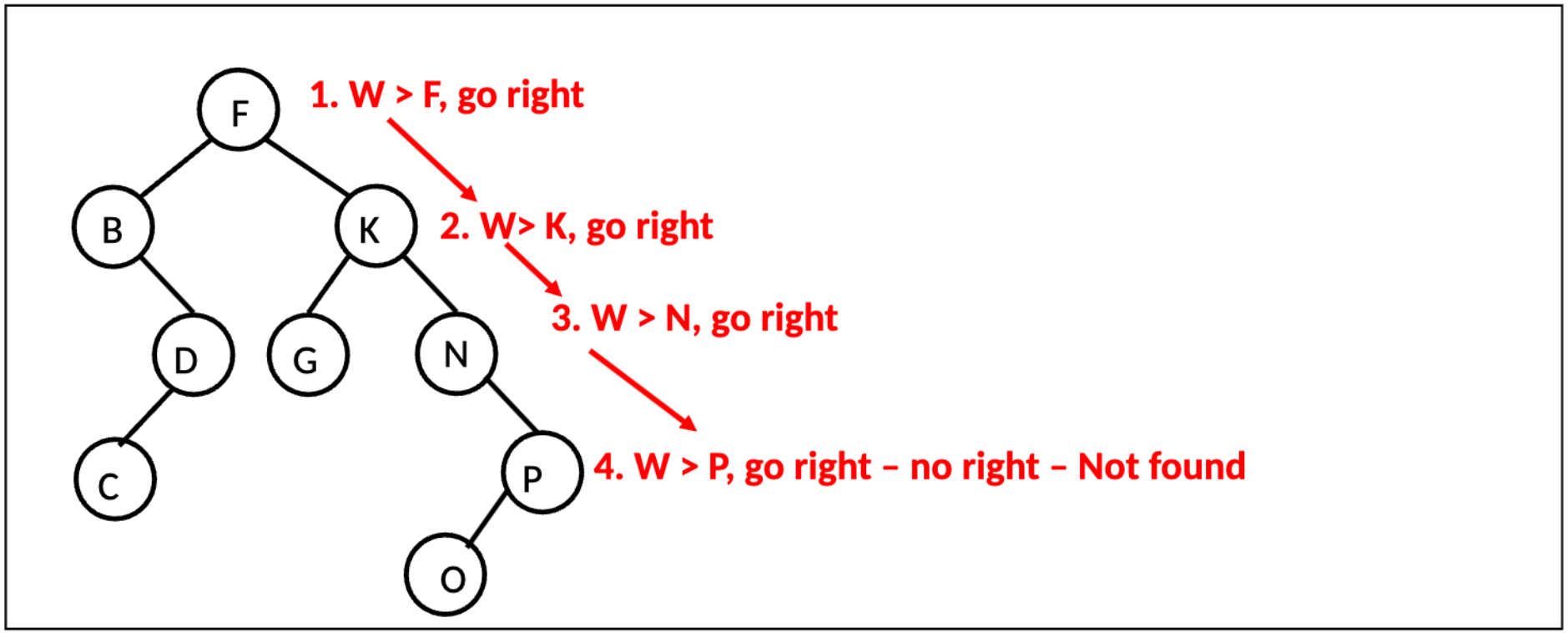
Search for C



The maximum number of comparisons to search a BST = tree height
Worst case **O(n)** when the tree is degenerate
Average and best case is **O(log n)** when the tree is balanced

Example of BST Search

Search for W

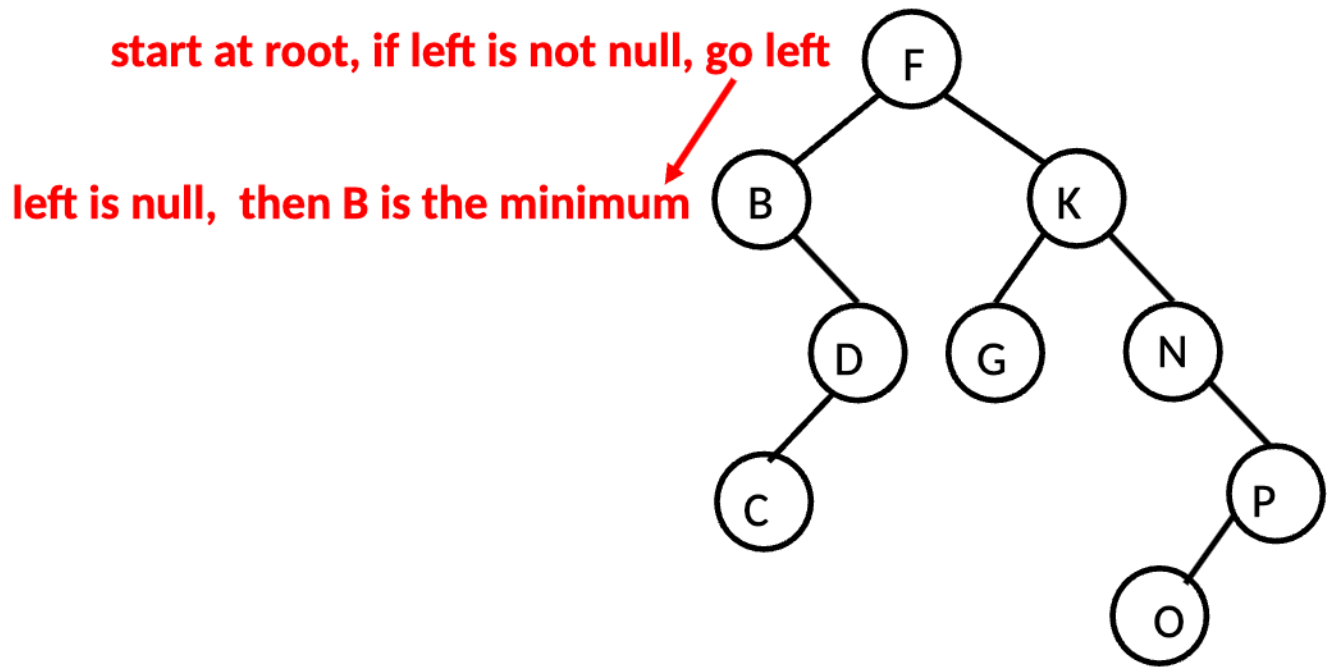


Finding the Minimum Value in a BST

- The minimum value is in the **left-most** node in the tree

- To find minimum:

```
start at the root
while (left != null)
      go left
```

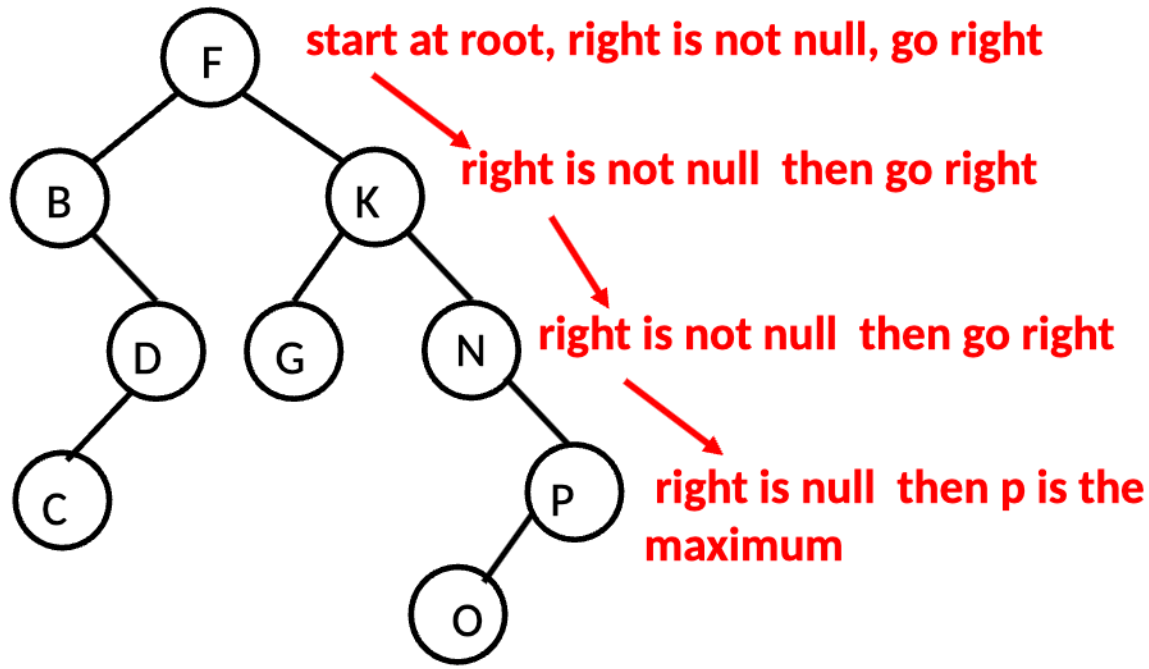


Finding the Maximum Value in a BST

- The maximum value is in the **right-most** node in the tree.

- To find maximum:

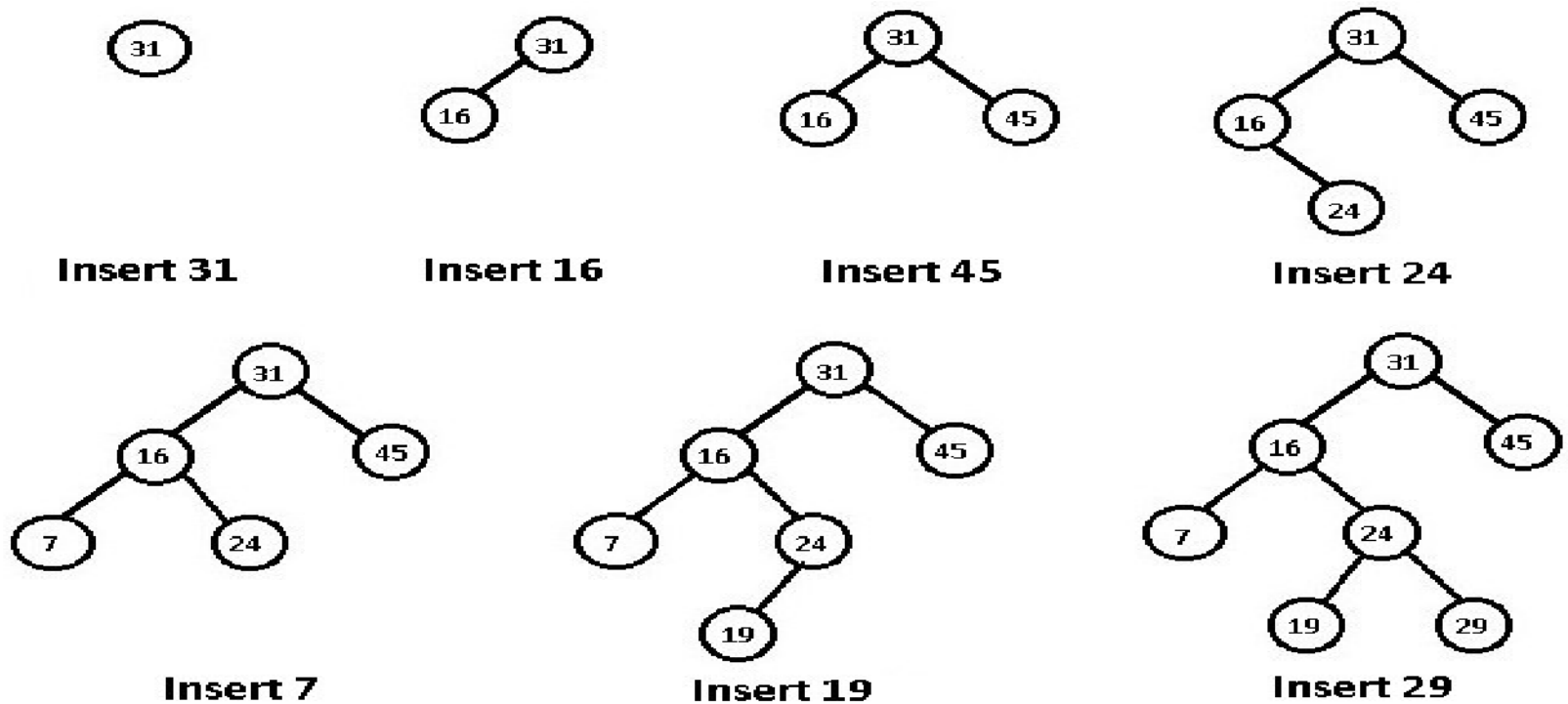
```
start at the root
while (right != null)
   go right
```



Add element to BST

BST Add Operation

- Build a BST by inserting the following values:
  - `31,16,45,24,7,19,29`

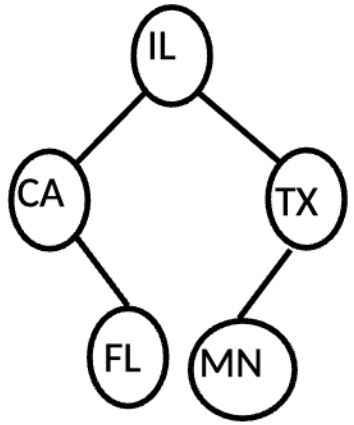


Will you get a different BST if you insert the values in a different order?

- Assume you want to build a BST from the following values:
  - MN, IL, CA, TX, FL

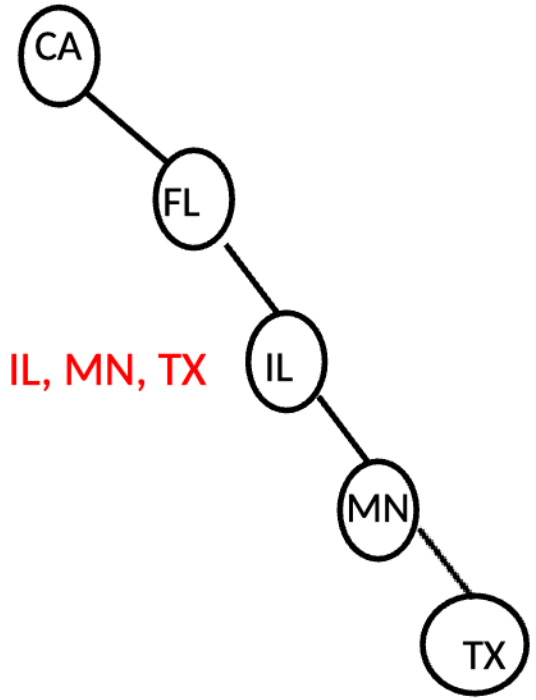
- See the resulting BSTs if the values are inserted in the following orders:
  - Case 1: IL,CA,TX,MN,FL
  - Case 2: CA, FL, IL, MN, TX
  - Case 3: TX,MN, IL,FL,CA
  - Case 4: MN, CA, FL, TX, IL

Case 1: IL,CA,TX,MN,FL



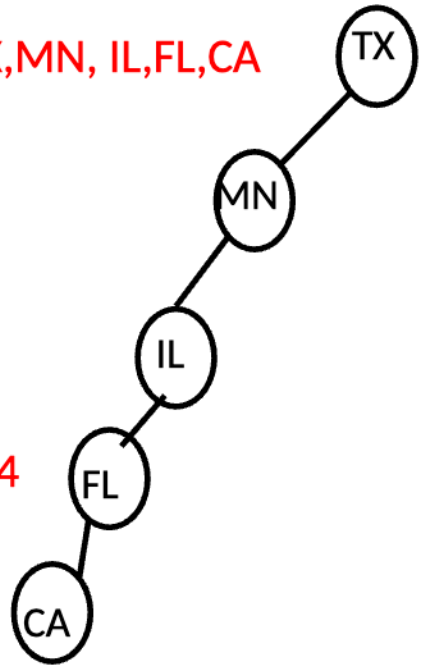
Height = 2

Case 2: CA, FL, IL, MN, TX



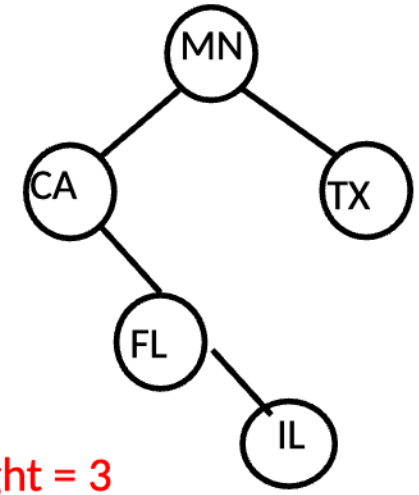
Height = 4

Case 3: TX,MN, IL,FL,CA



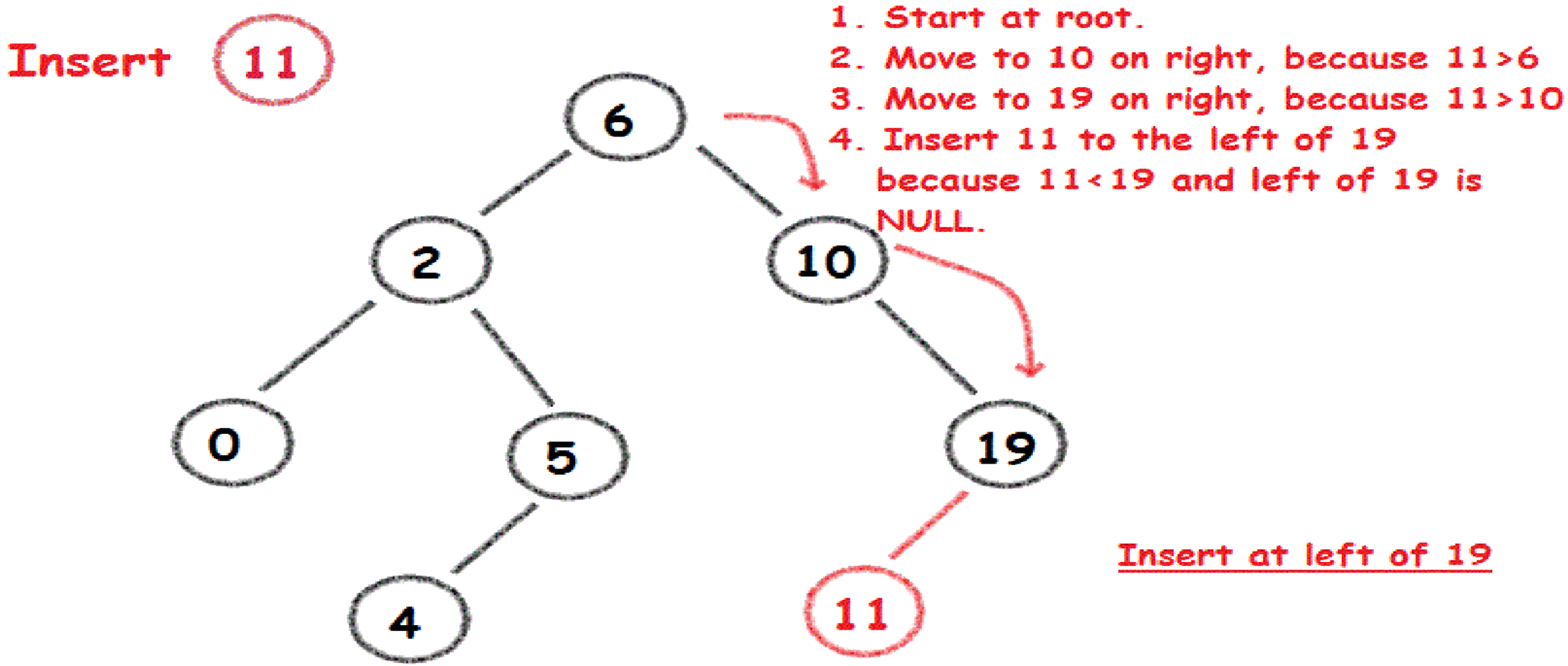
Height = 4

Case 4: MN, CA, FL, TX, IL



Height = 3

BST Add Operation

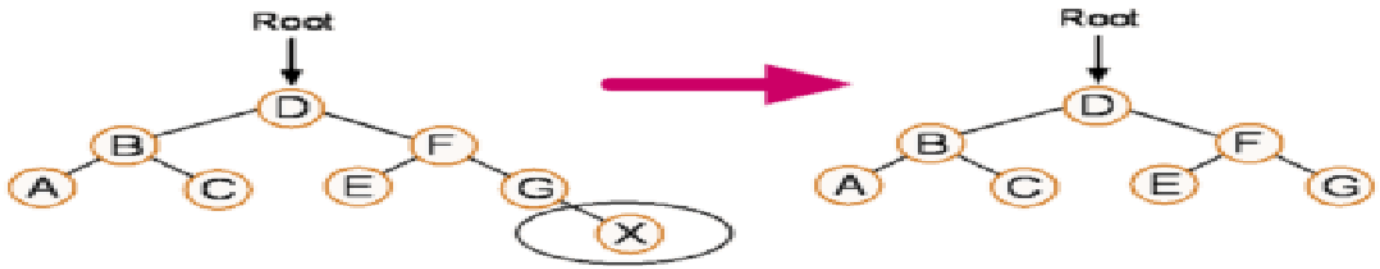


Delete element from BST

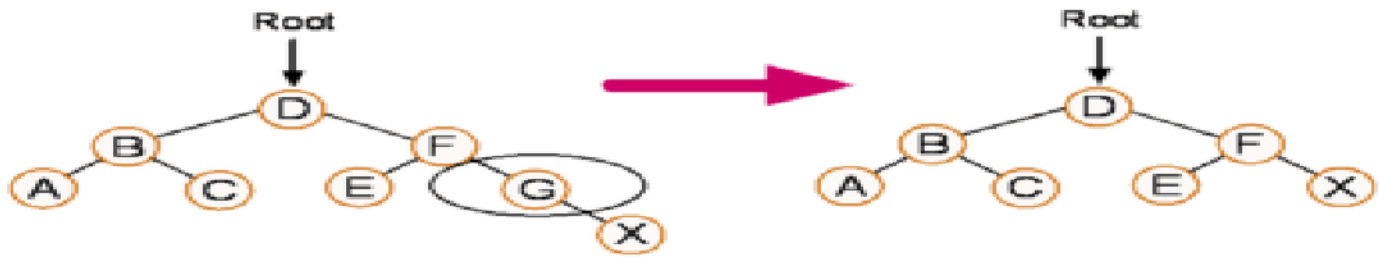
BST Delete Operation

- The node to be deleted can be one of the following three types:
  - CASE 1: leaf node:
    - Just remove the node from the tree
  - CASE 2: has one child:
    - Replace the node with its child
  - CASE 3: has two children (left and right):
    - Depending on the implementation, replace the node with either:
      - the minimum element in its right subtree, or
      - the maximum element in its left subtree

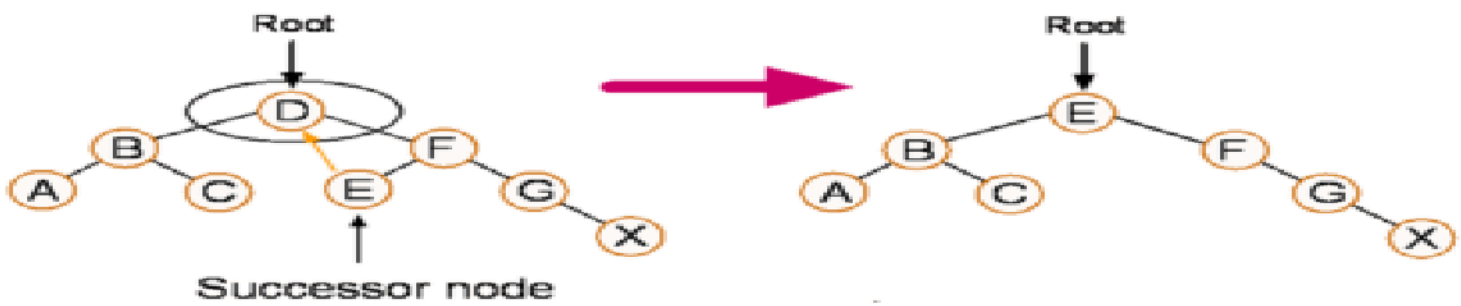
Leaf Deletion



Deleting a node with a single child



Deleting a node with two children, locate the successor node on the right-hand side (or predecessor on the left) and replace the deleted node (D) with the successor (E). Finally remove the successor node.



Extra

Binary Search Tree Animation

- https://www.cs.usfca.edu/~galles/visualization/BST.html