CS151 Intro to Data Structures

Trees

Announcements

- Lab7 and HW5 due next Friday (Nov 7th)
  - START EARLY

Outline

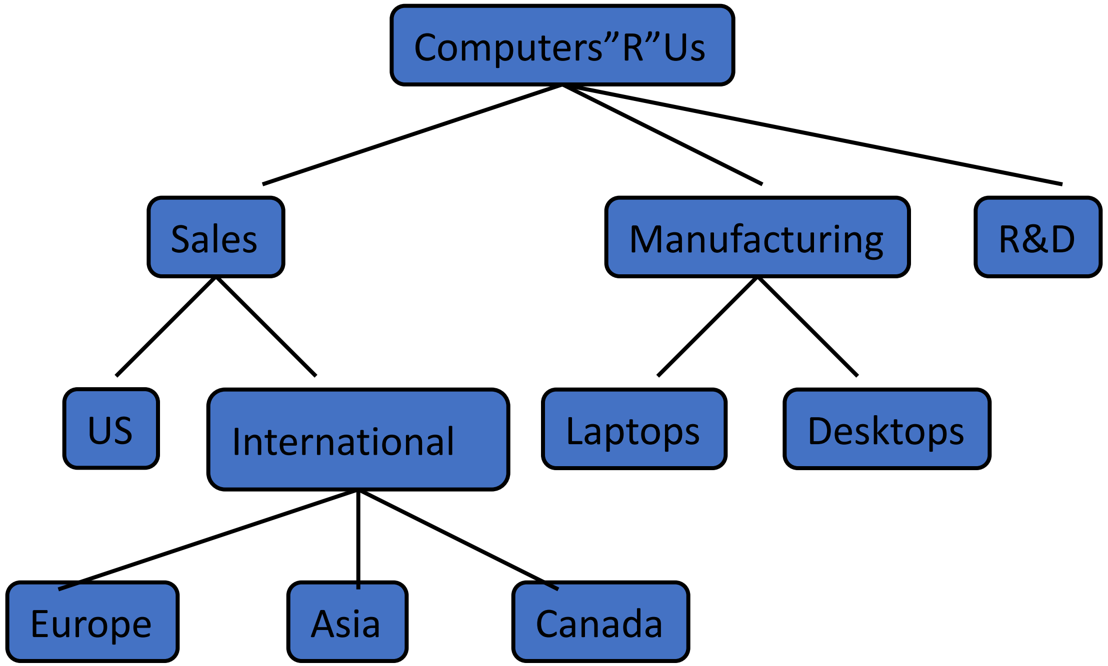
- Trees:
  - Binary Trees
  - Binary Search Trees
    - Inserting
    - Searching

Tree

A tree is a **hierarchical** data structure

Node: individual elements in the tree

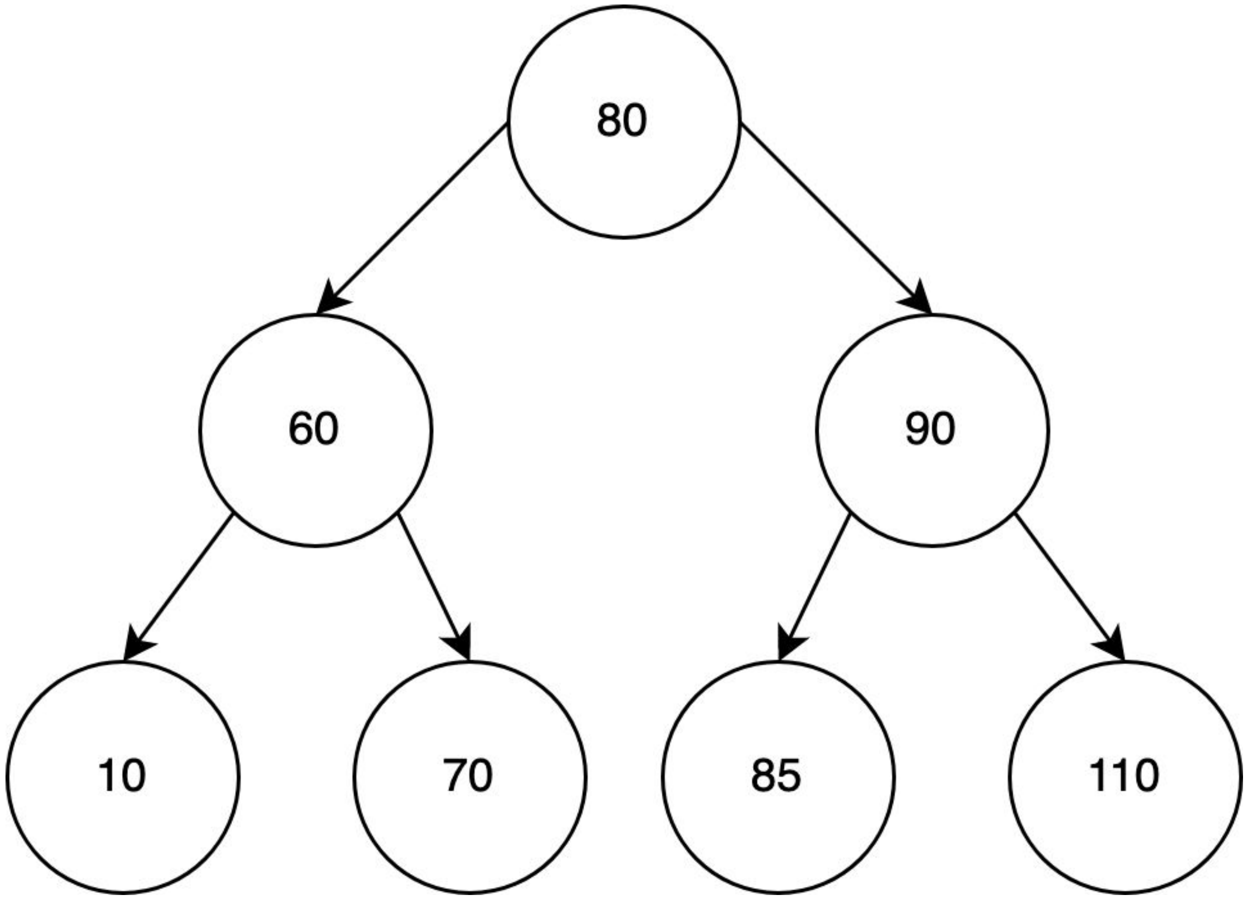
Nodes have a parent-child relation



Trees: Nodes

```
class Node {
    int key;
    Node left;
    Node right;

    public Node(int item) {
        key = item;
        left = null;
        right = null;
    }
}
```



Terminology

root: no parent

A

external/leaf node: no children

E, I, J, K, G, H, D

internal node: - node with at least one child

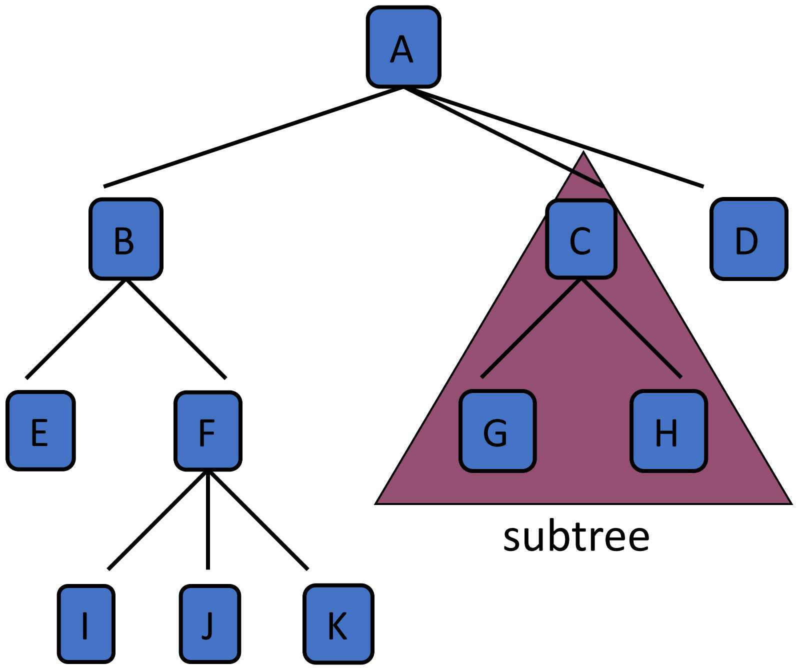
A, B, C, F

parent/child

depth - # of ancestors

Height - Maximum number of edges from a leaf node to the root

- **Subtree**: tree consisting of a node and its descendants



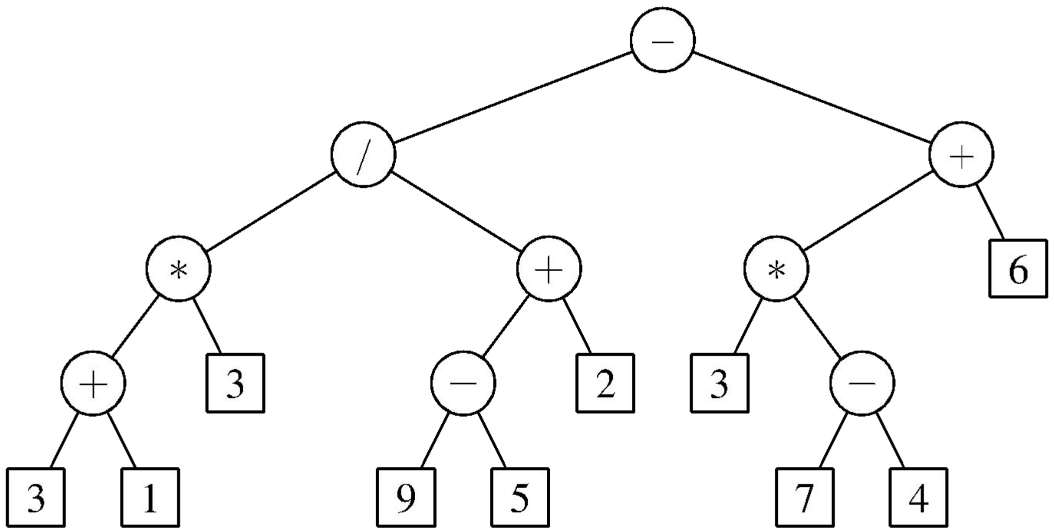
Binary Tree

Each node in a **binary tree** has at most two children

Recursive definition:

Each node has at most two children

- Both subtrees are **binary trees**



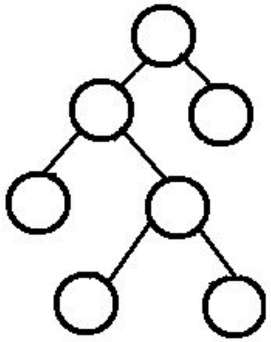
Types of Binary Trees

A binary tree is **full** (or proper) if each node has **zero or two children**

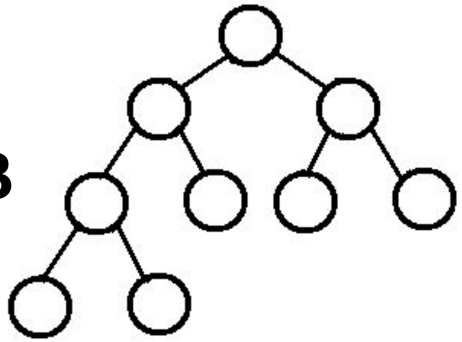
A binary tree is **complete** if every level (except possibly the last) is fully filled and, if the last level of the tree is not fully filled, the nodes of that level are filled from left to right

If a complete binary tree is filled at every level, it is **perfect**

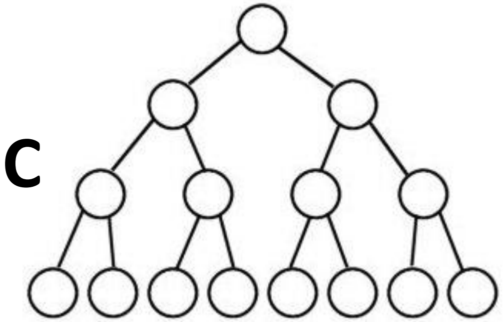
A



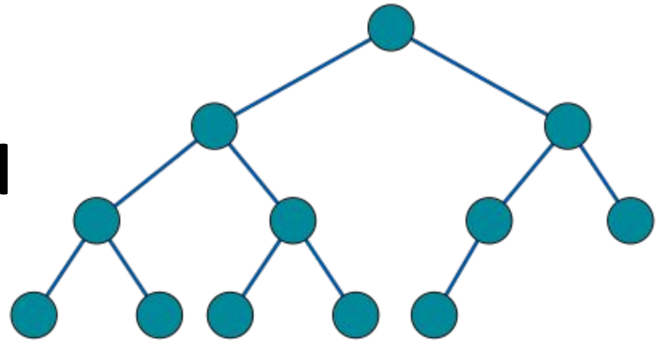
B



C



d



Types of Binary Trees

A binary tree is **full** (or proper) if each node has **zero or two children**

A binary tree is **complete** if every level (except possibly the last) is fully filled and leftmost aligned

If a complete binary tree is filled at every level, it is **perfect**

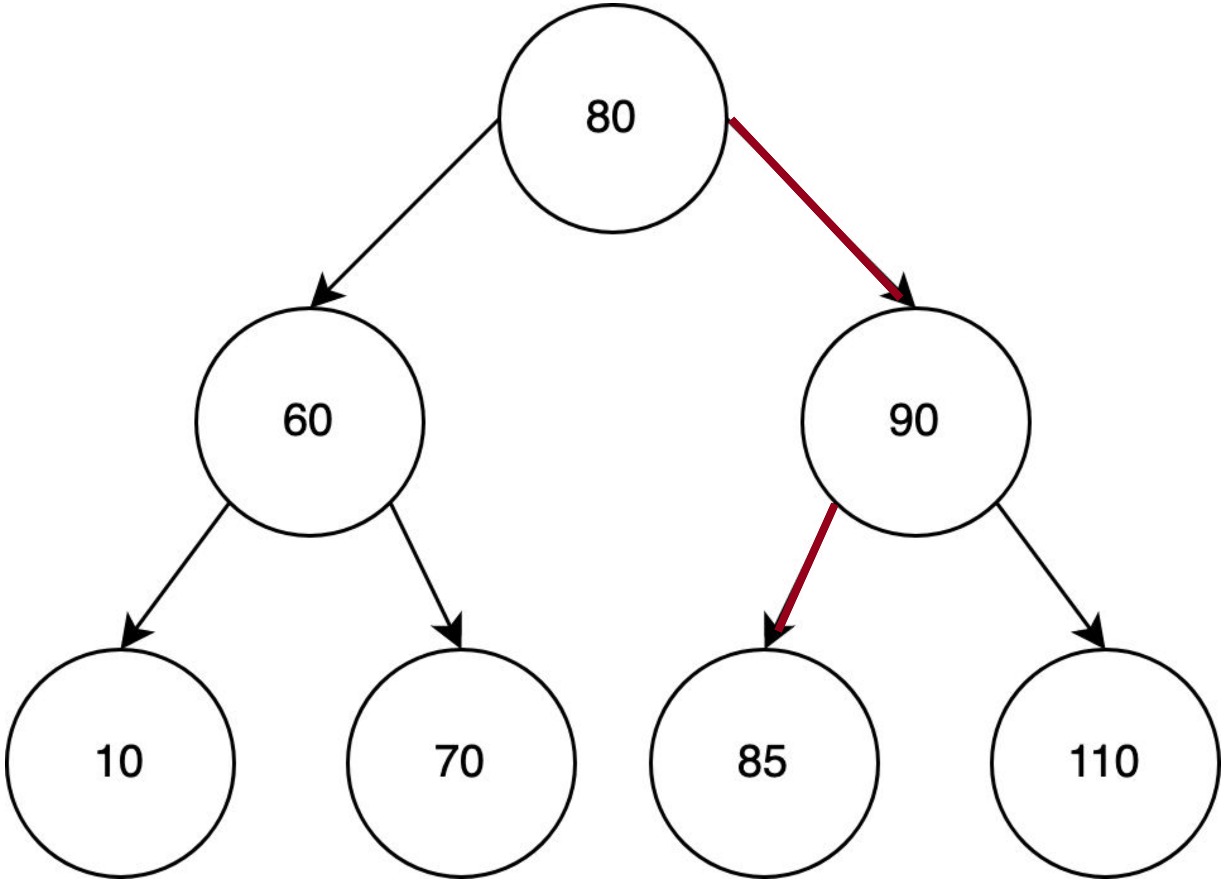
Q1: Is every full binary tree a complete binary tree?

Q2: Is every complete binary tree a full binary tree?
Q3: Is every perfect binary tree a full binary tree?

Binary Trees: Height

Height of a tree:

Maximum number of edges from a leaf node to the root



Height? 2

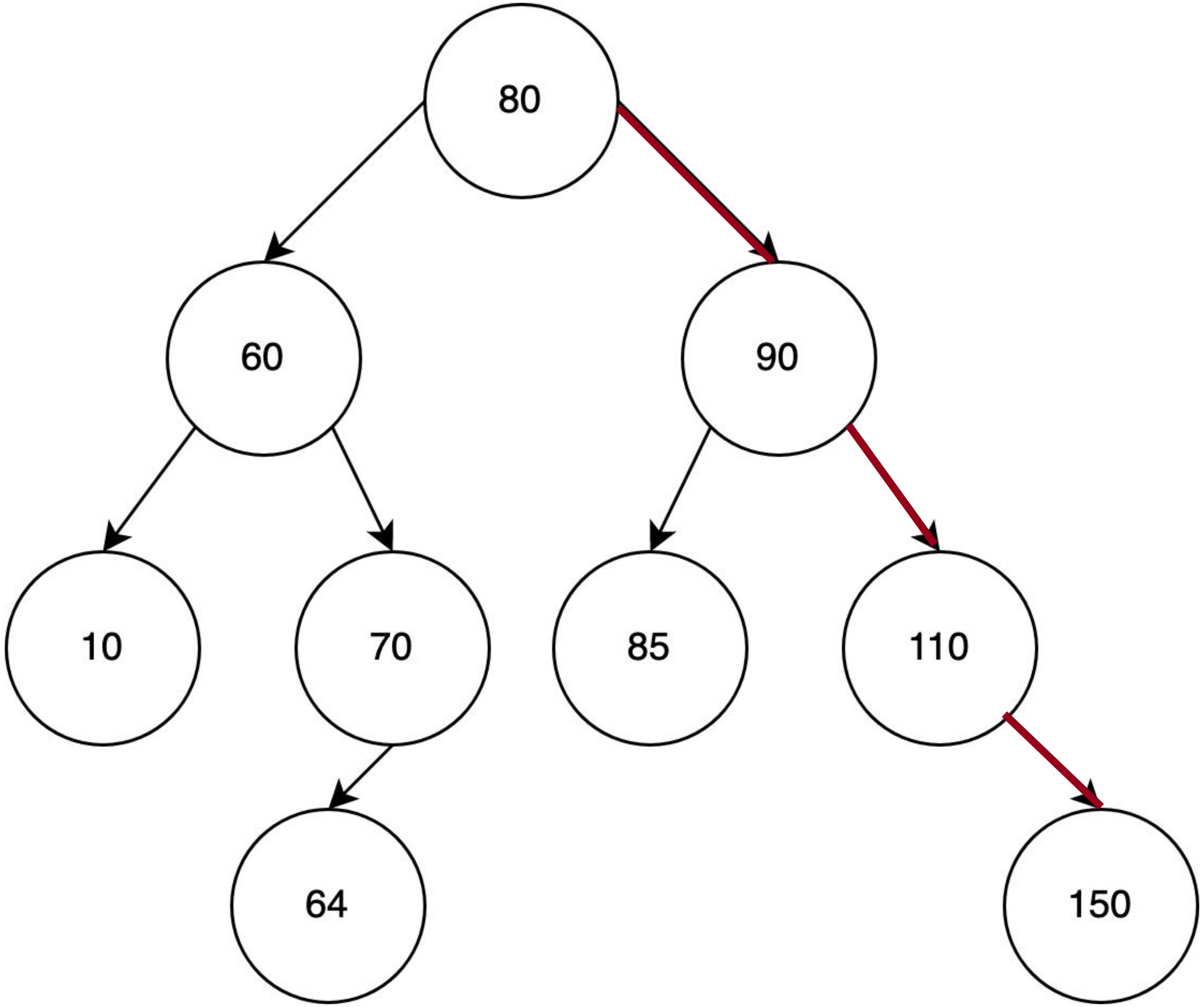
$\log_2(7) \approx 2$

Tree Review

Height? 3

$\log_2(9) \approx 3$

Height of a binary tree is roughly log(n) where n is number of nodes

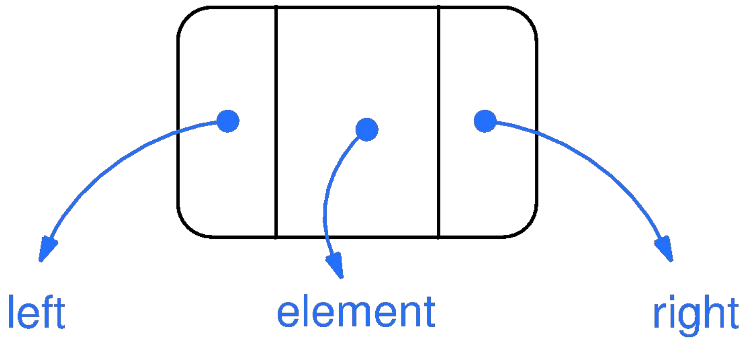


Binary Tree Interface

```
public interface BinaryTree<E extends
Comparable<E>> {
   int size();
   boolean isEmpty();
   void insert(E element);
   boolean contains(E element);
   ...
}
```

Node Implementation

```
public class Node<E> {
   private E element;
   private Node<E> left;
   private Node<E> right;
   //constructors, getters, setters
   }
}
```



Class

```
public class LinkedBinaryTree<E extends
Comparable<E>> implements BinaryTree<E> {
    // what instance variables?
    // nested Node class



}
```

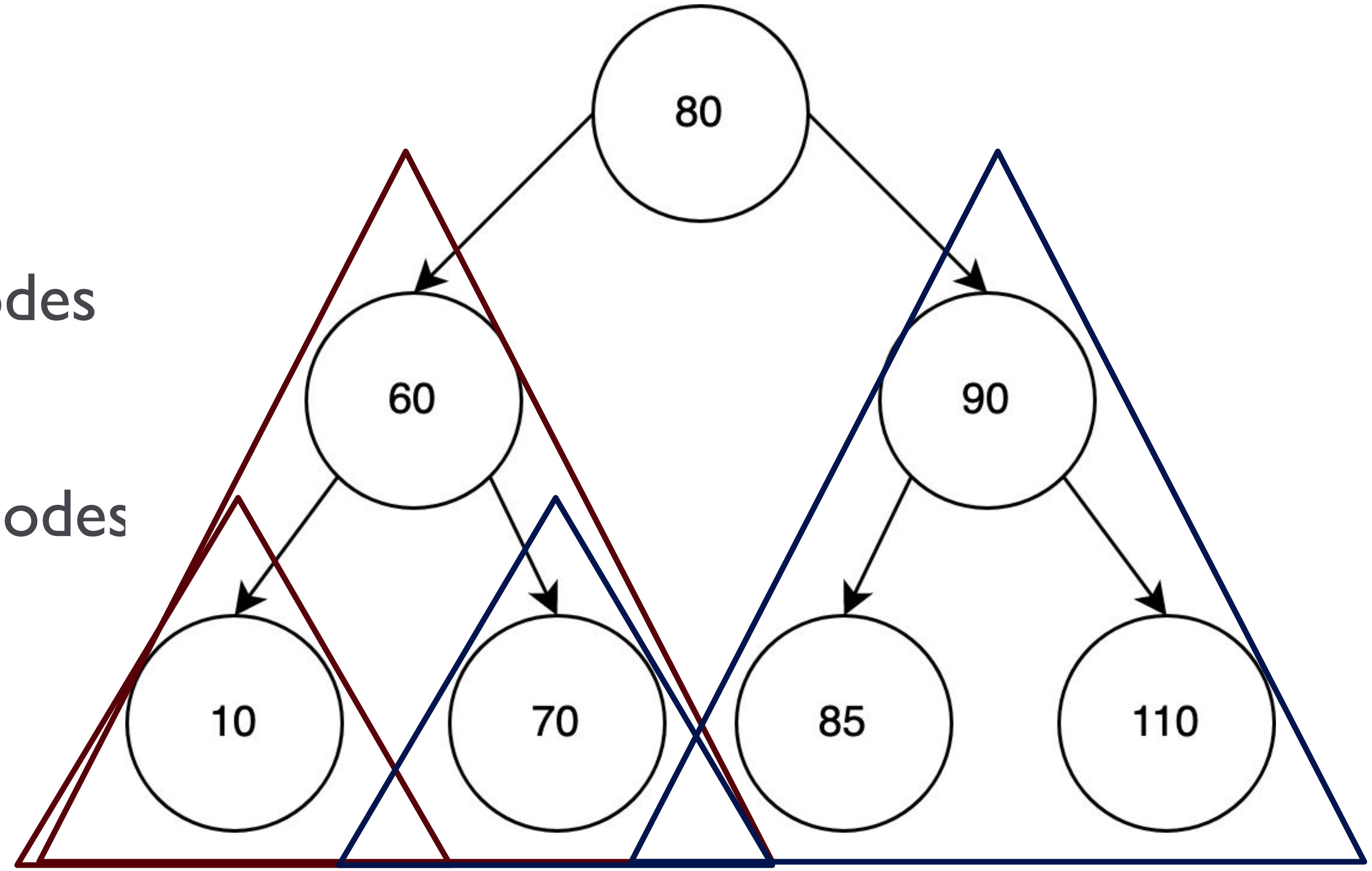
Binary Search Trees

Binary Search Trees

Definition:

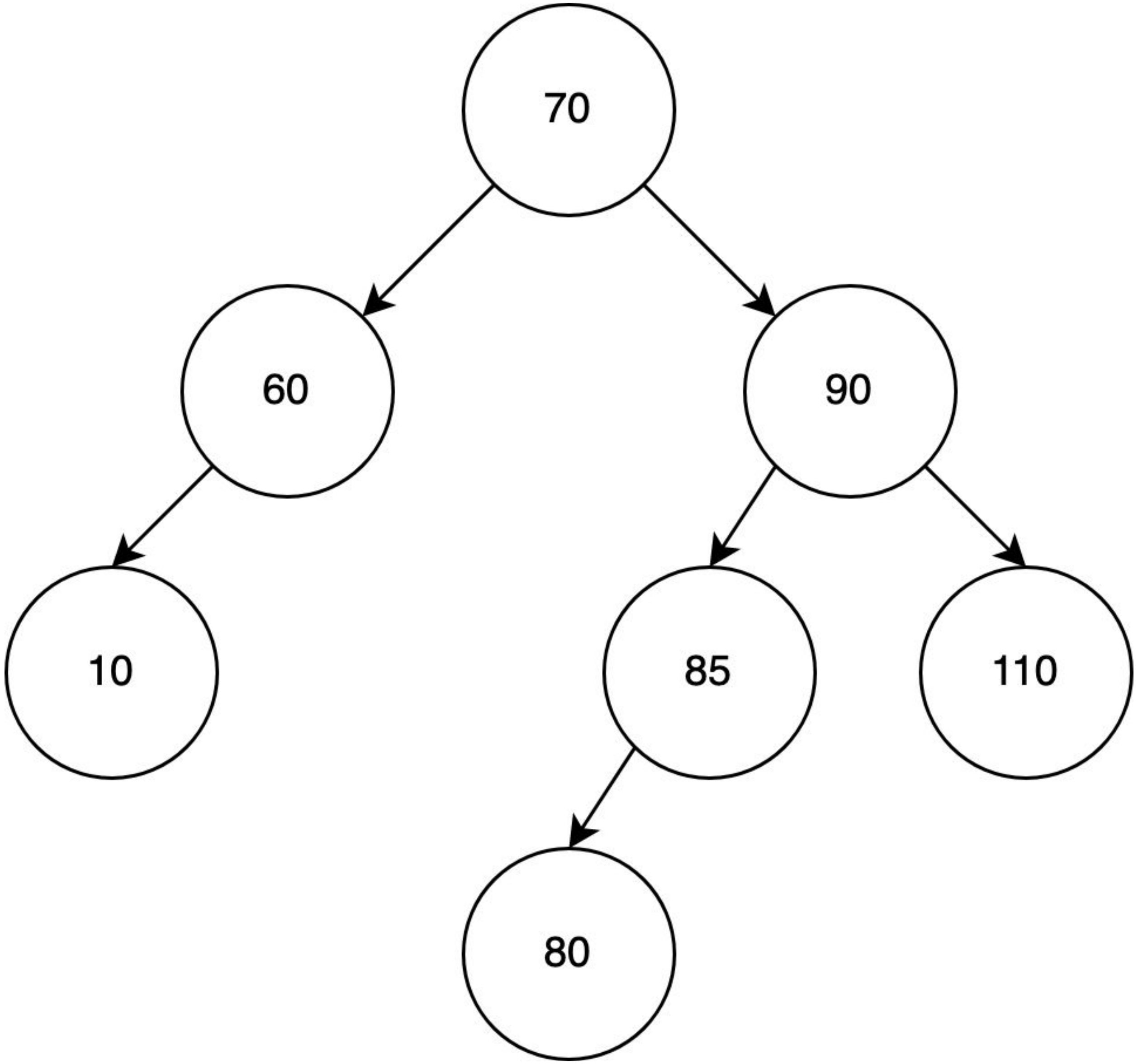
At **each node** with value **k**

- Left subtree contains only nodes with value **lesser** than **k**
- Right subtree contains only nodes with value **greater** than **k**
- Both subtrees are a **binary search tree**



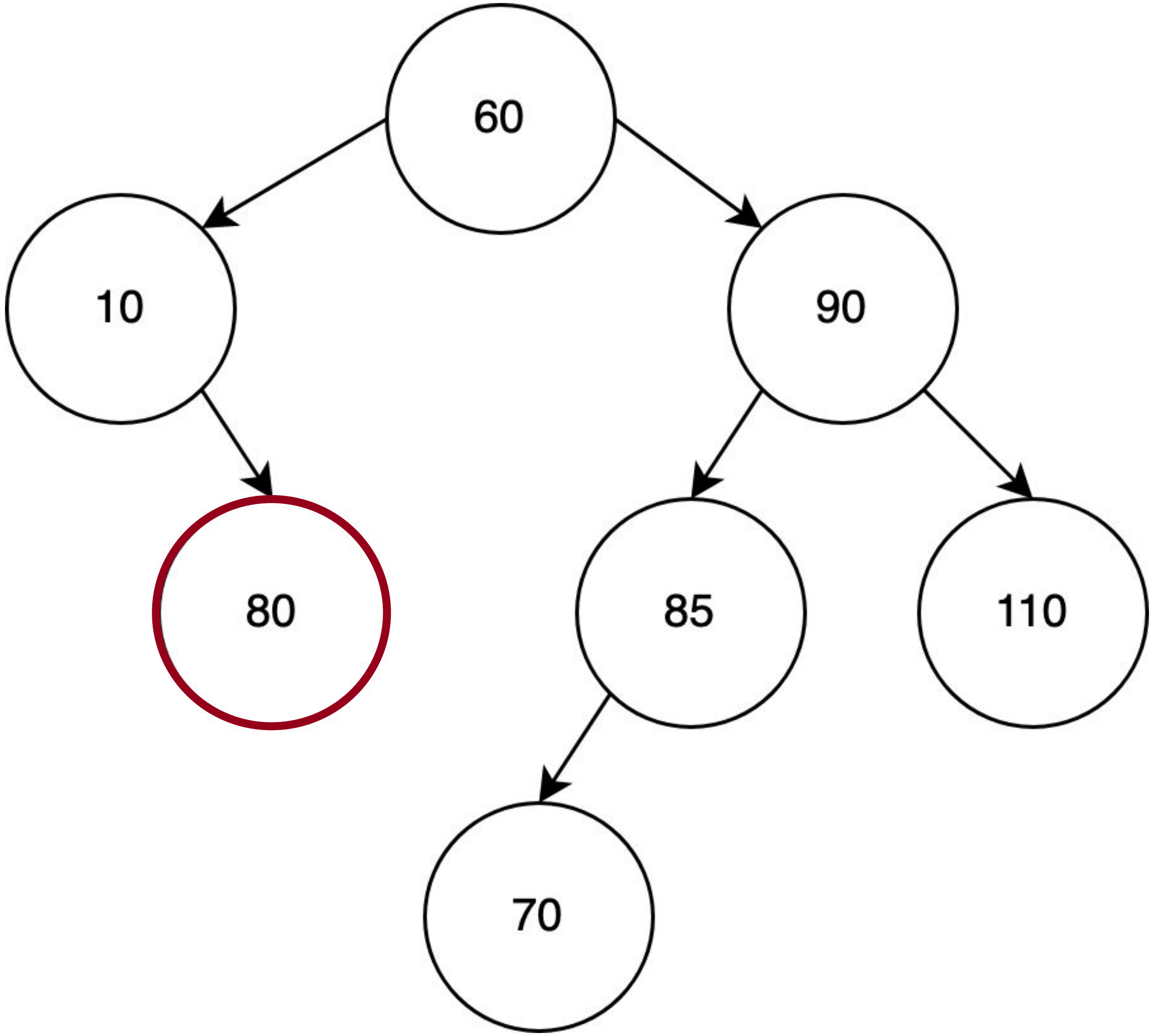
Exercise One: Binary Search Trees

Is this a binary search tree?



Exercise One: Binary Search Trees

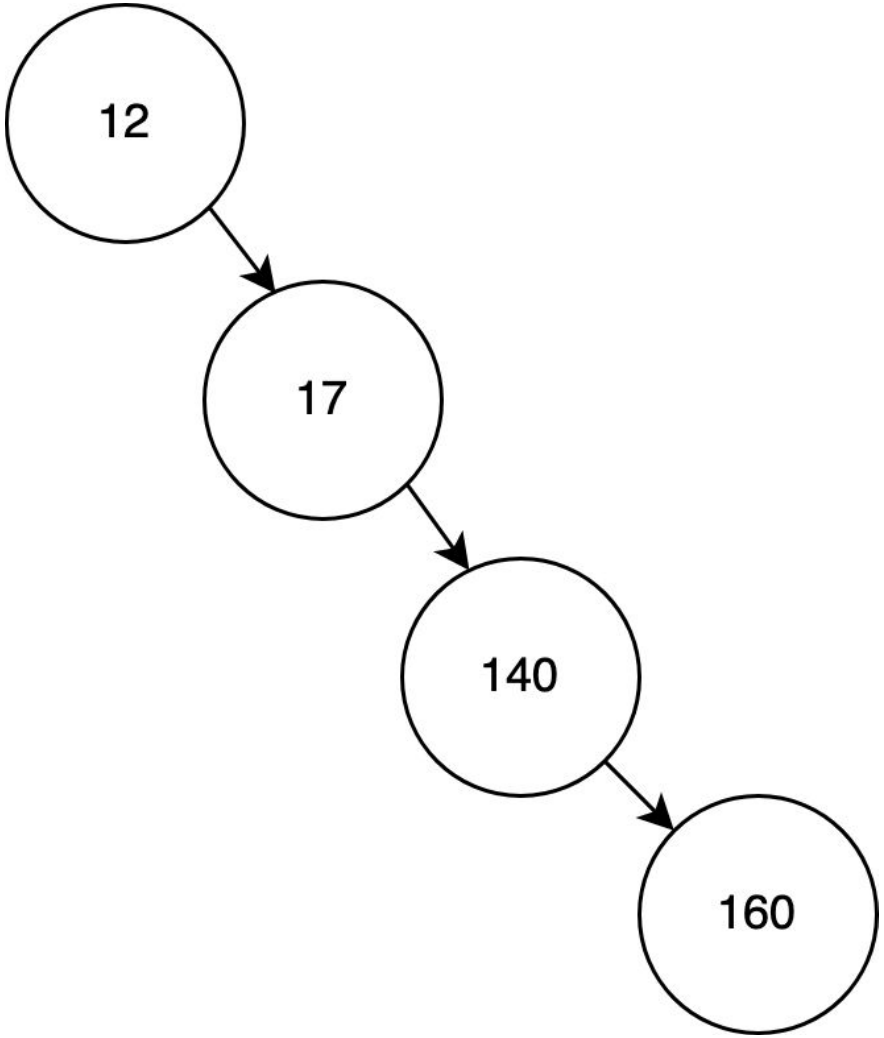
Is this a binary search tree?



Exercise One: Binary Search Trees

Is this a binary search tree?

✓



Today's Lecture

1. Binary Search Trees
2. **Search**
3. Insertion
4. Removal
5. Summary

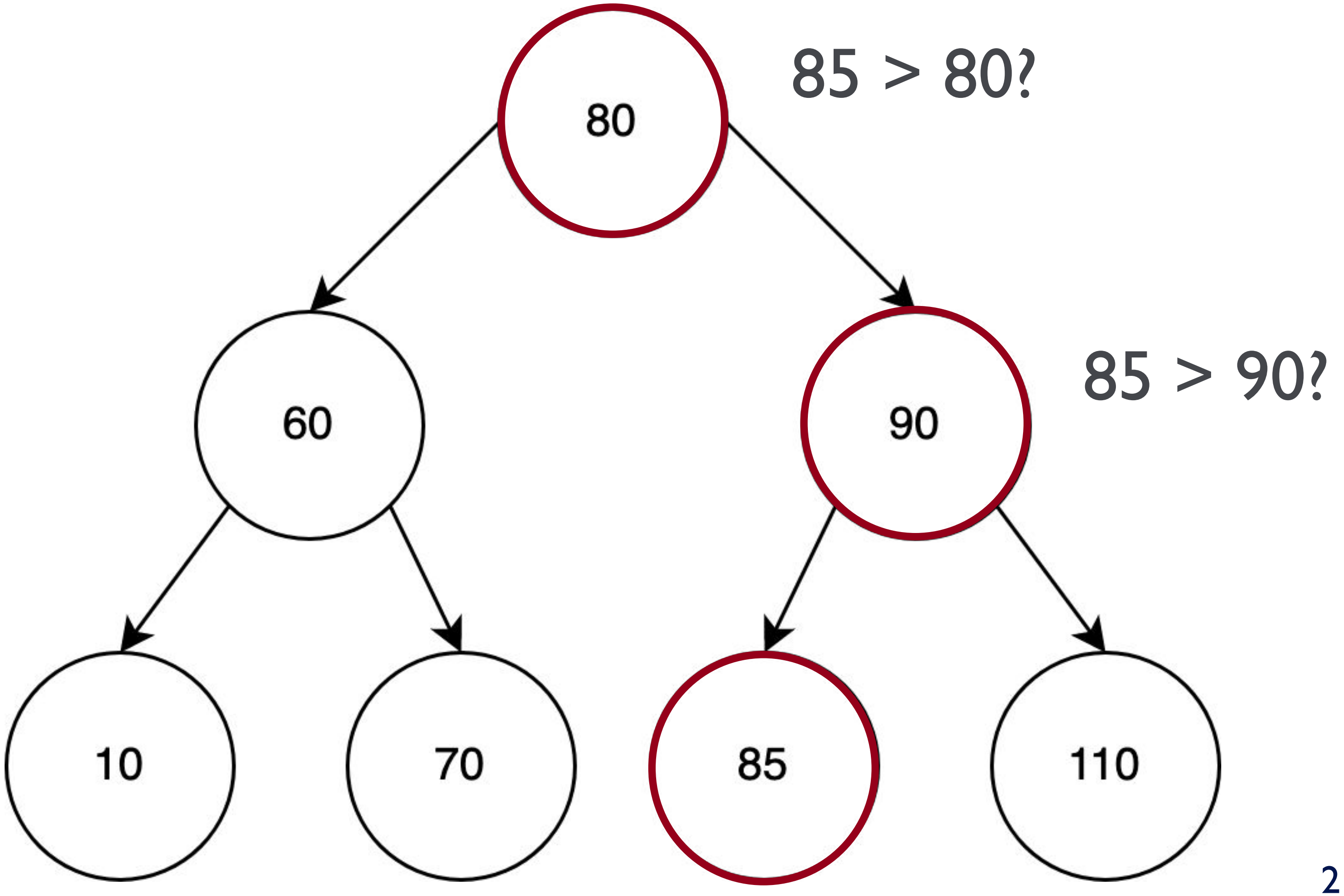
Binary Search Trees: Efficient Search

Goal: Report if a value exists in the tree

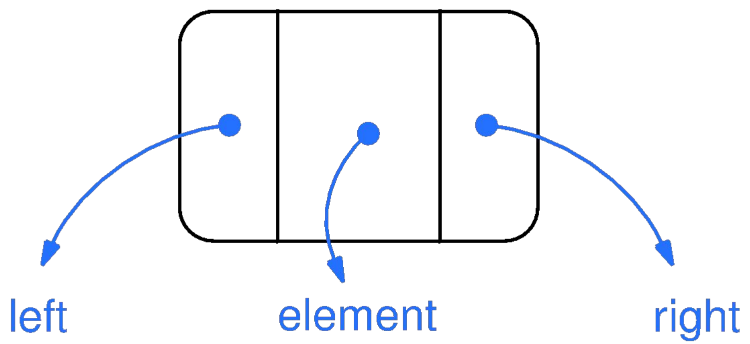
Target: 85

if **target** > **k**:
    Move right
else:
    Move Left

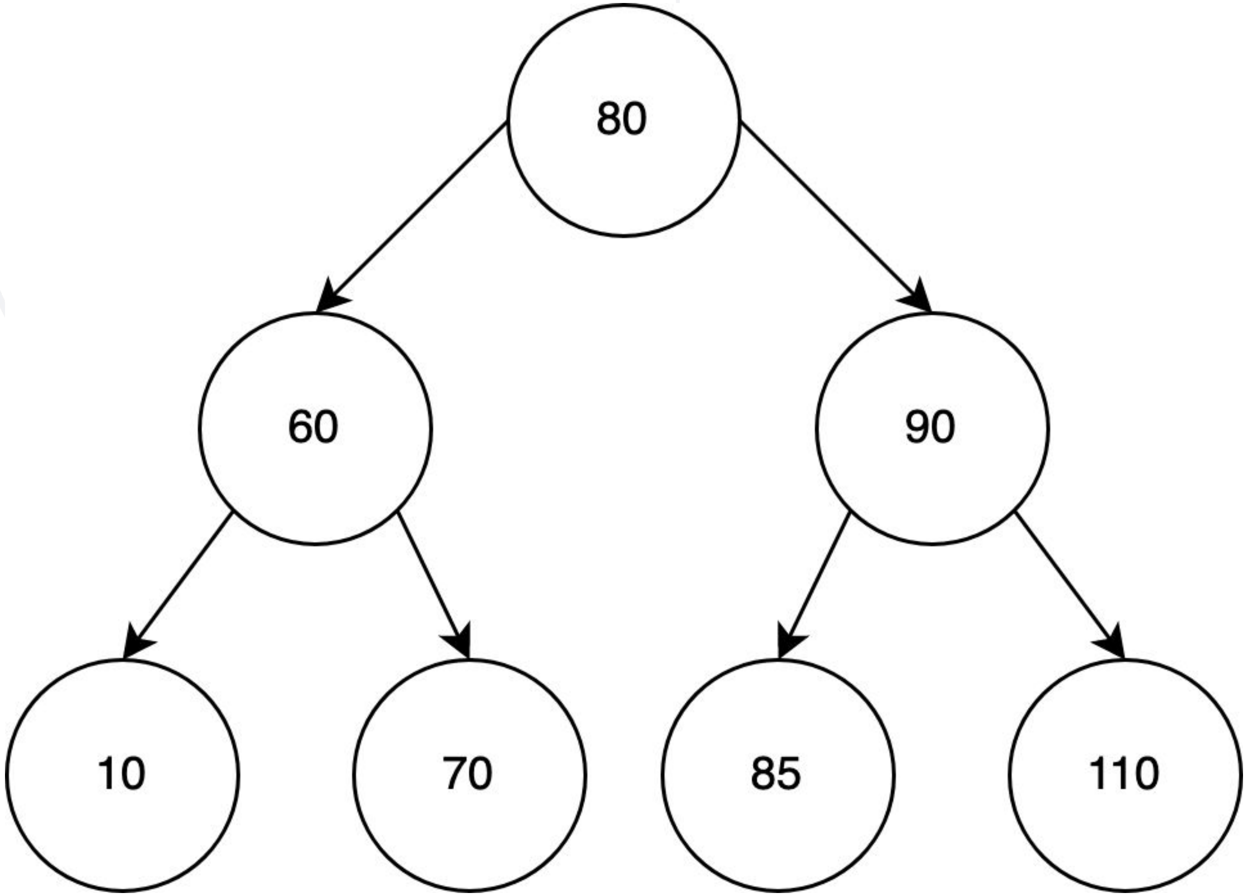
Complexity?
O(log n)



BSTs: Search Implementation



BSTs: Search Implementation



search(Node(80), 85)

search(Node(90), 85)

search(Node(85), 85)

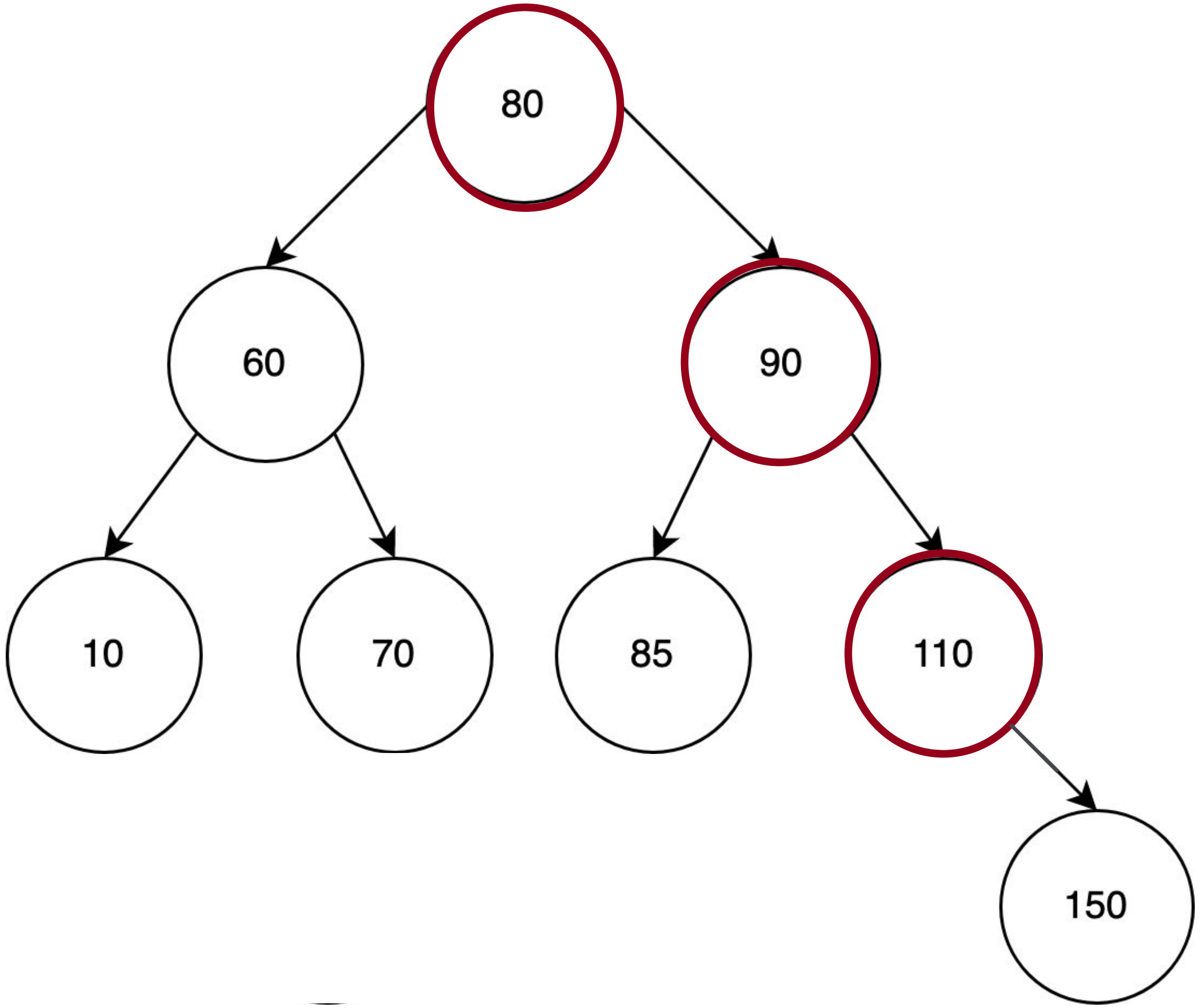
Today's Lecture

1. Binary Search Trees
2. Search
3. **Insertion**
4. Removal
5. Summary

Binary Search Trees: Insertion

Insertion must maintain the properties of a BST!

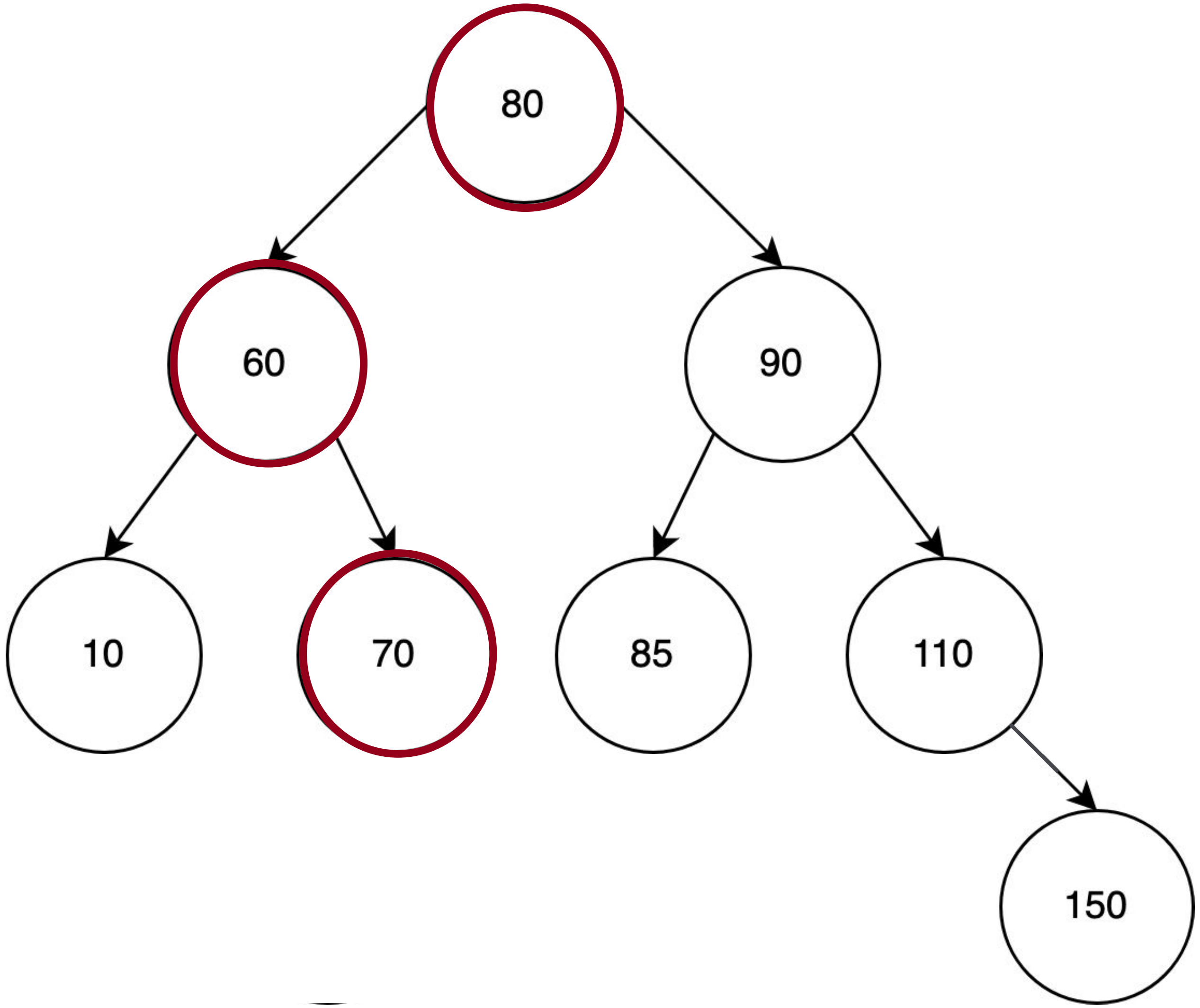
Insert: 150



Binary Search Trees: Insertion

Insertion must maintain the properties of a BST!

Insert: 64



Complexity?
O(log n)

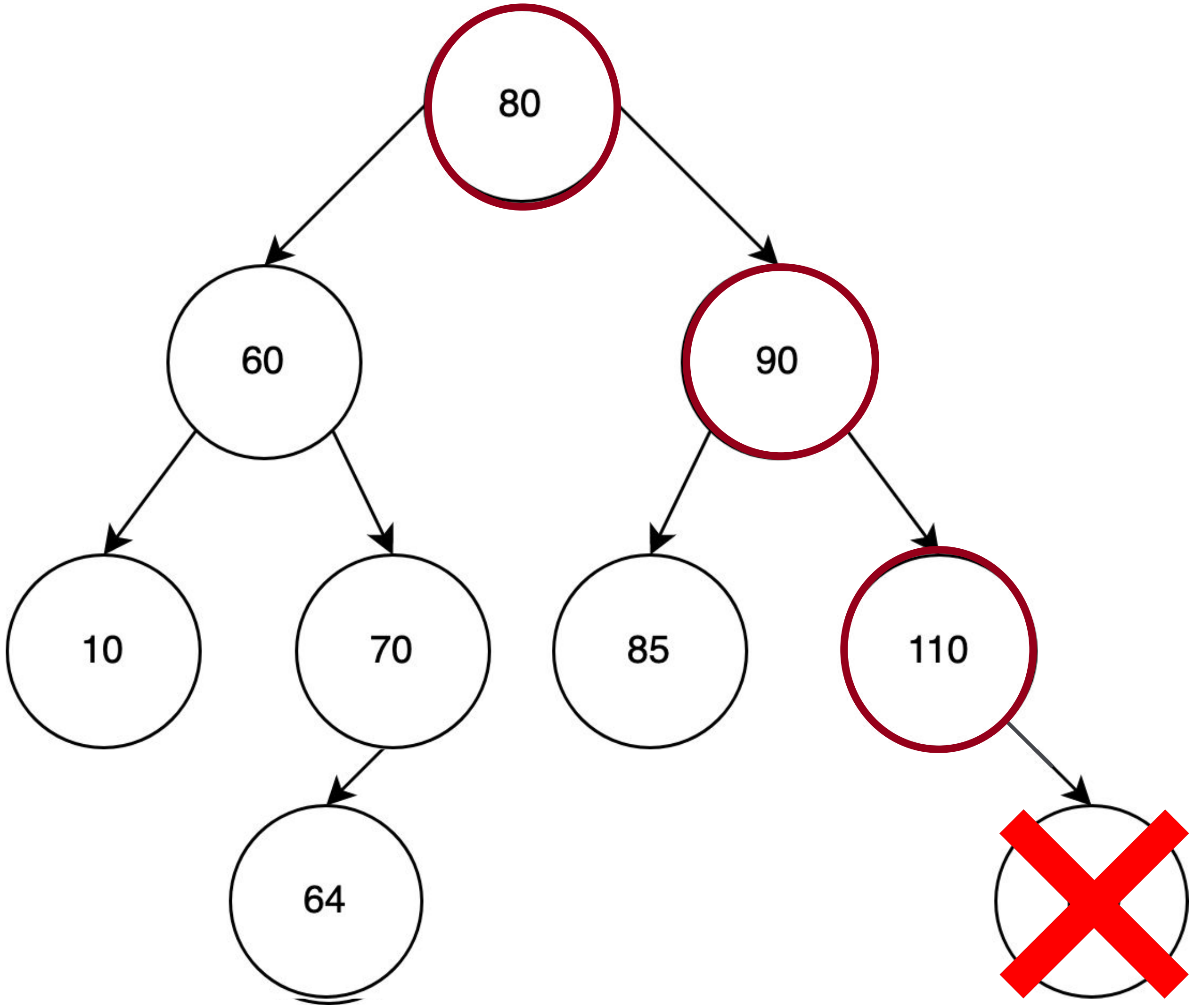
Today's Lecture

1. Binary Search Trees
2. Search
3. Insertion
4. **Removal**
5. Summary

Binary Search Trees: Deletion

Deletion must maintain the properties of a BST!

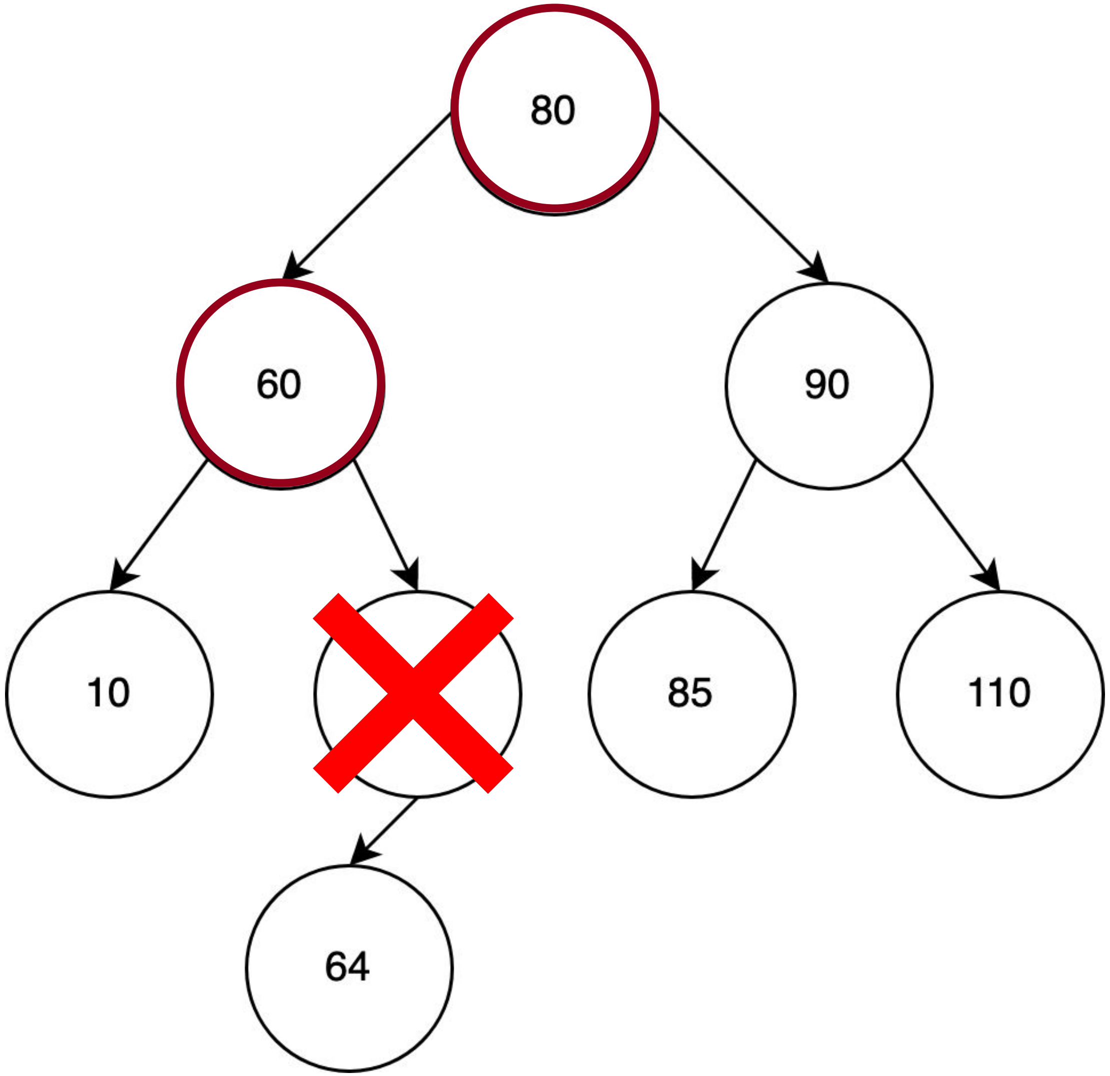
Delete: <u>150</u>



Binary Search Trees: Deletion

Deletion must maintain the properties of a BST!

Delete: <u>70</u>



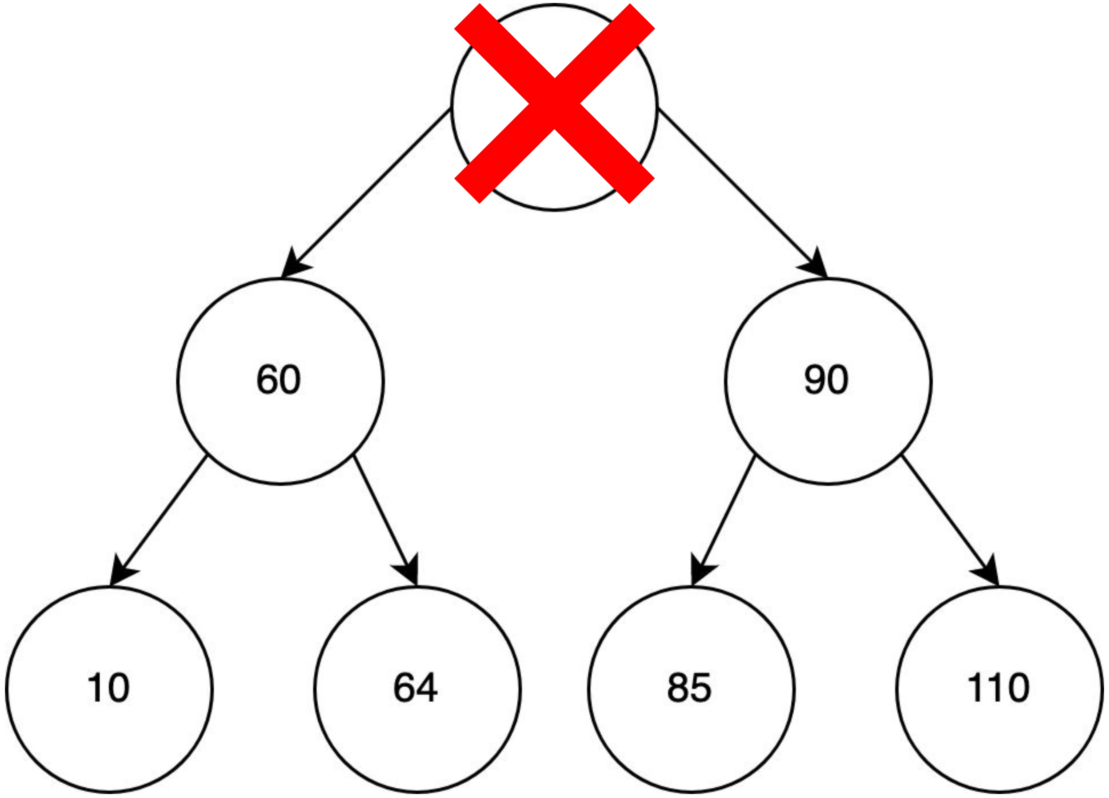
Binary Search Trees: Deletion

Deletion must maintain the properties of a BST!

Delete: 80

At **each node** with value **k**

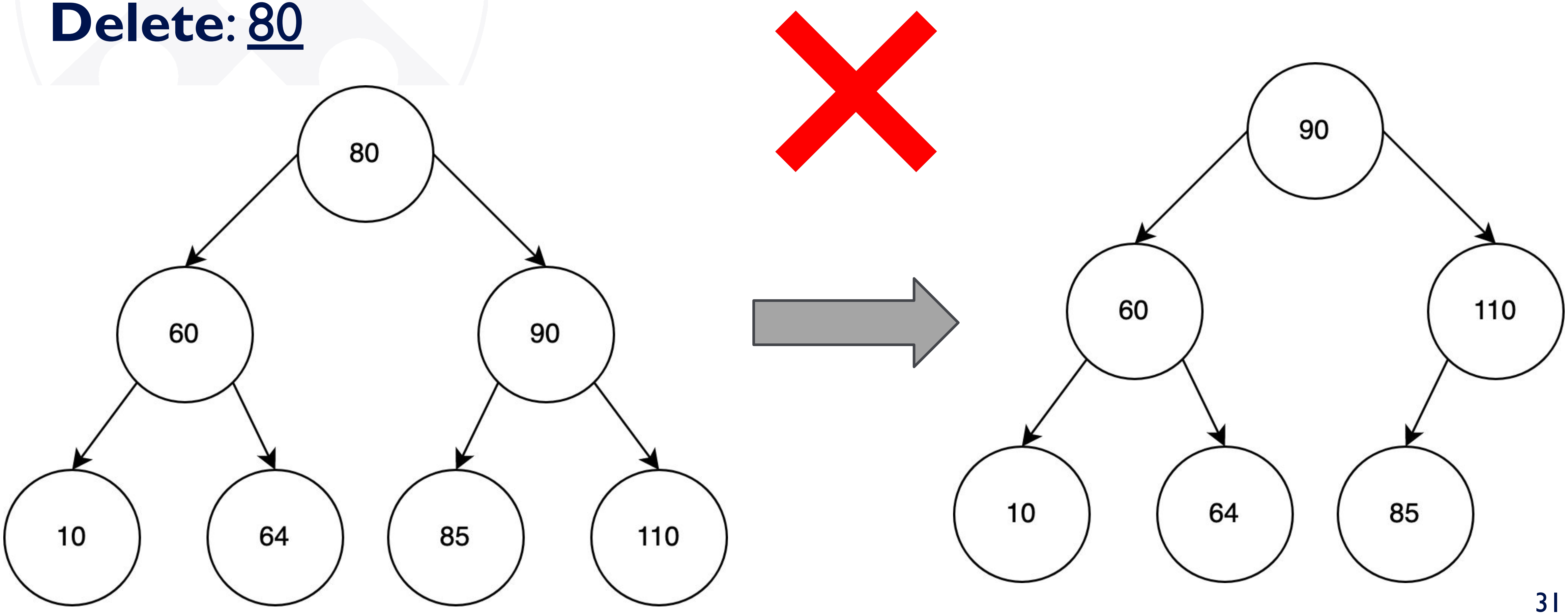
- Left subtree contains only nodes with value **lesser** than **k**
- Right subtree contains only nodes with value **greater** than **k**
- Both subtrees are a **binary search tree**



Binary Search Trees: Deletion

Replace with 90?

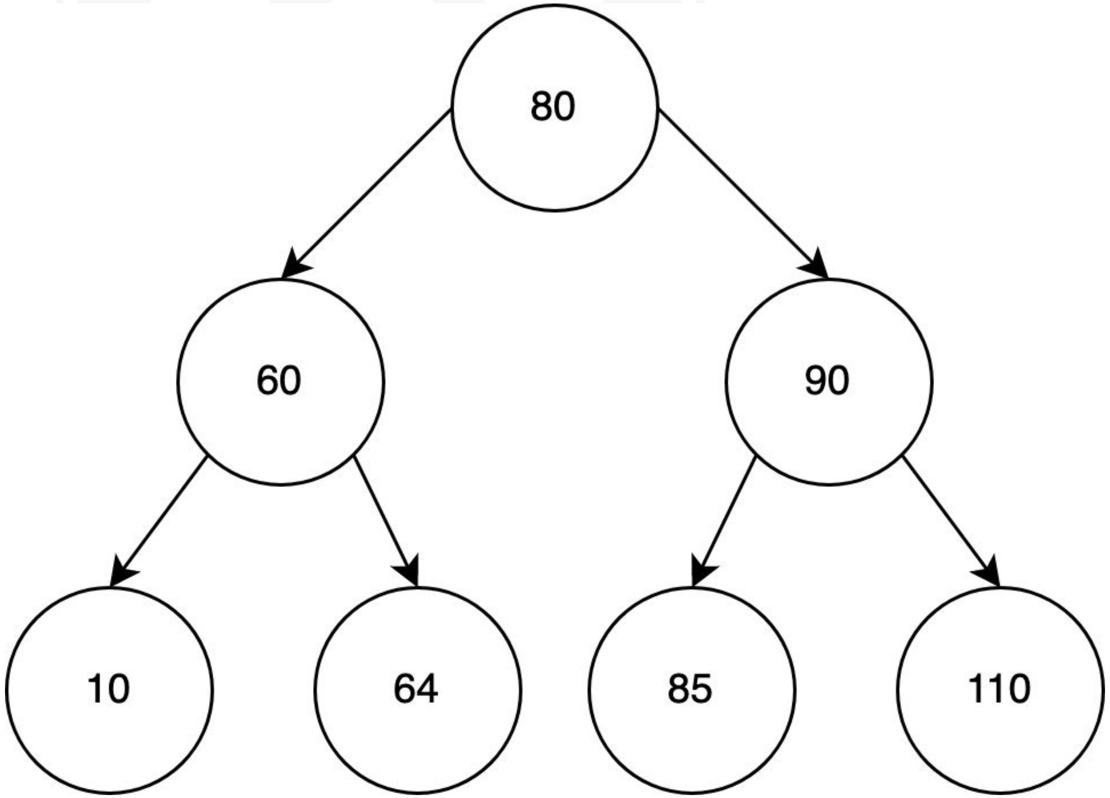
Delete: 80

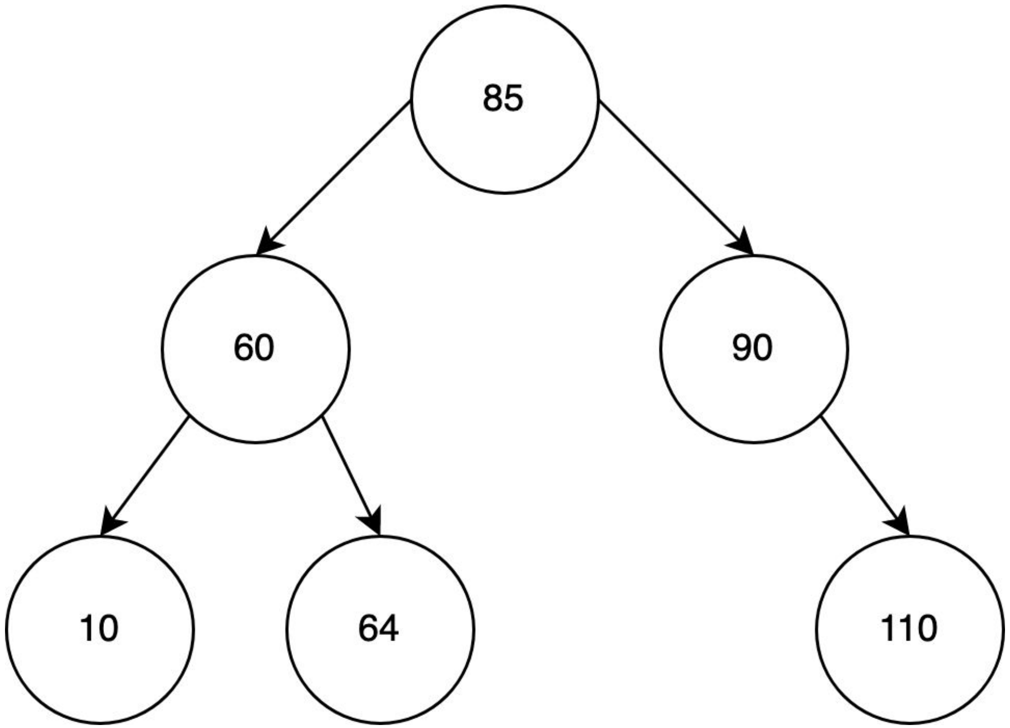


Binary Search Trees: Deletion

Replace with 85?

Delete: <u>80</u>

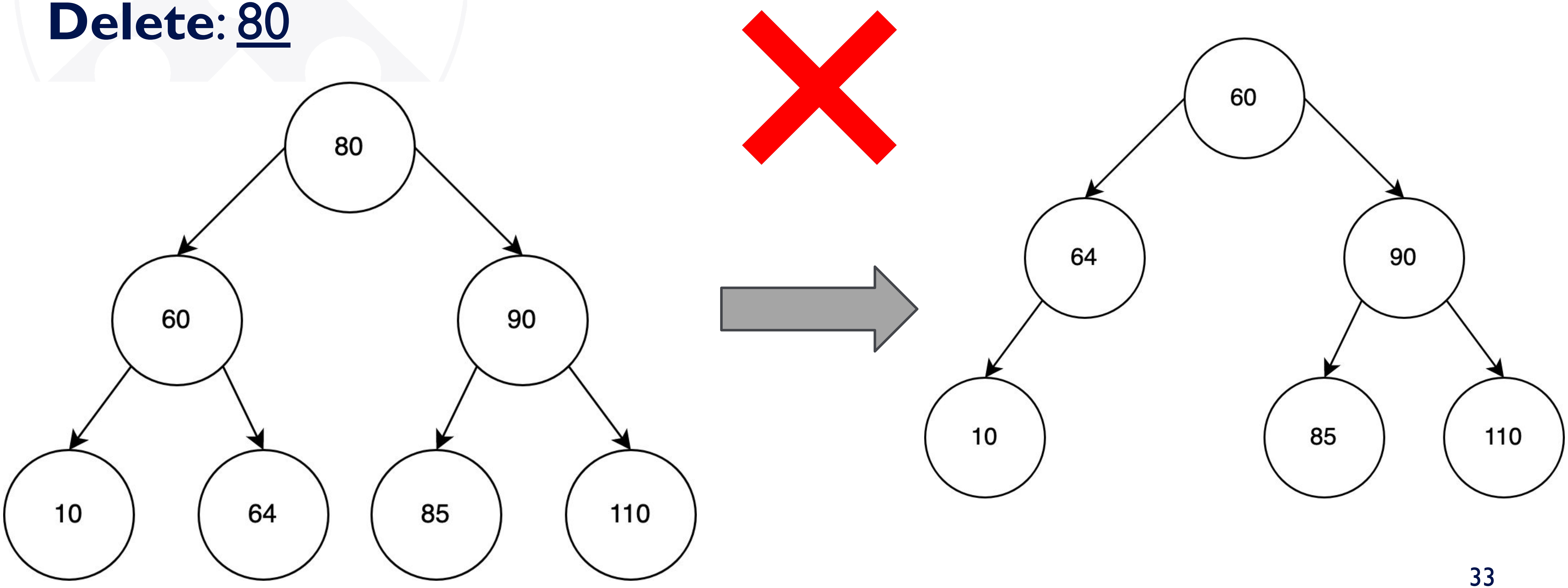




Binary Search Trees: Deletion

Replace with 60?

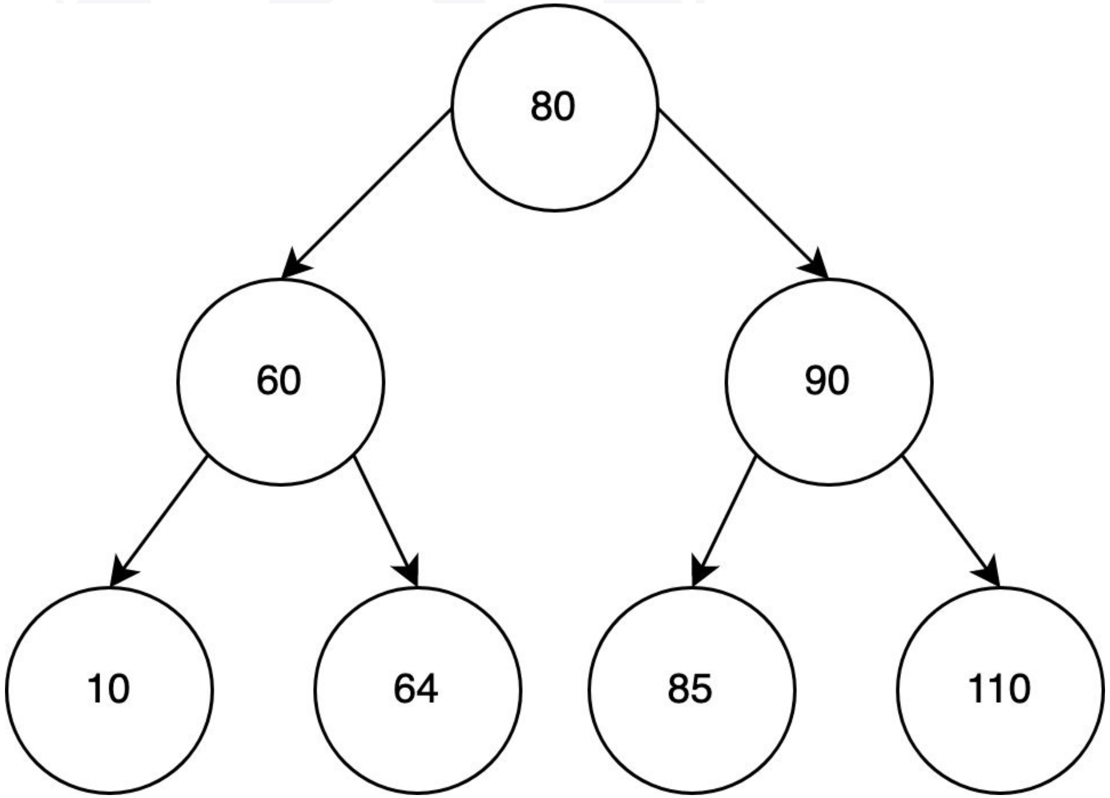
Delete: <u>80</u>

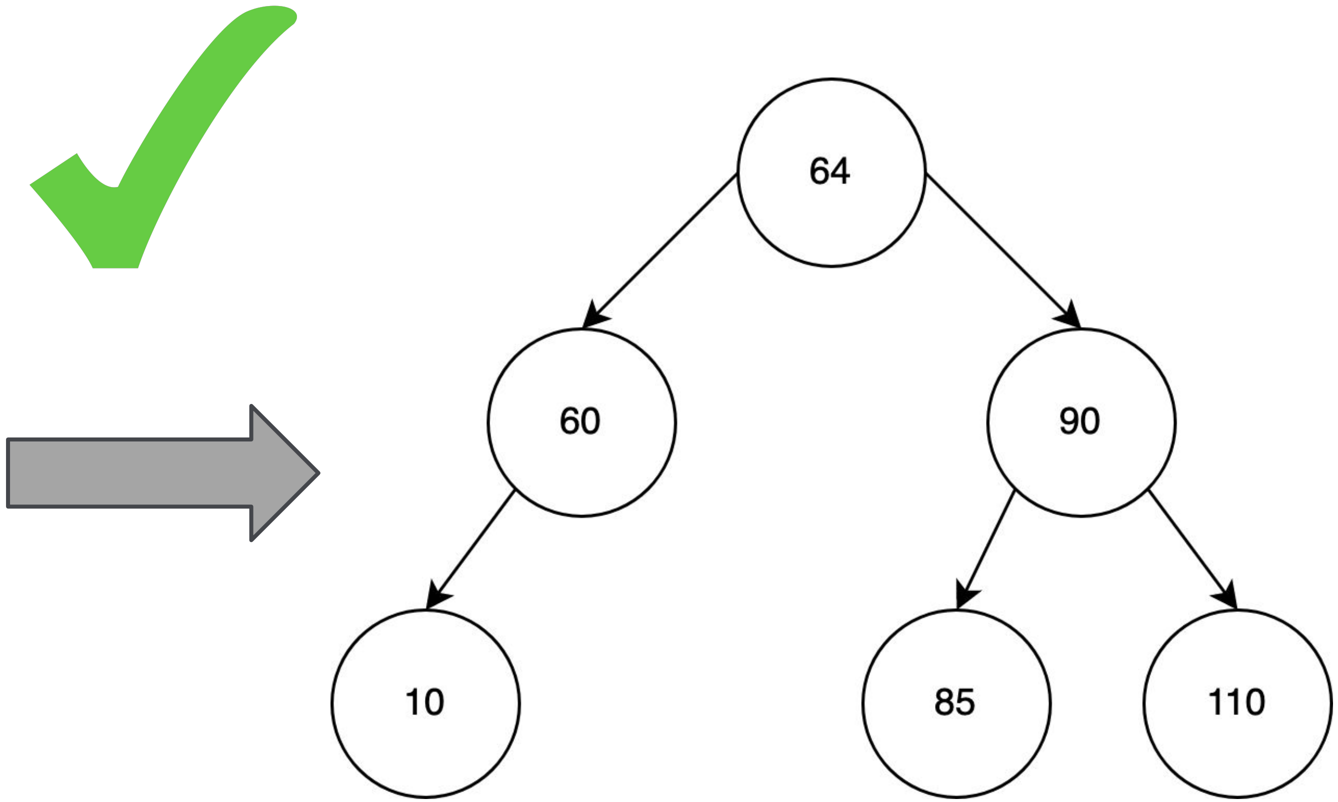


Binary Search Trees: Deletion

Replace with 64?

Delete: <u>80</u>





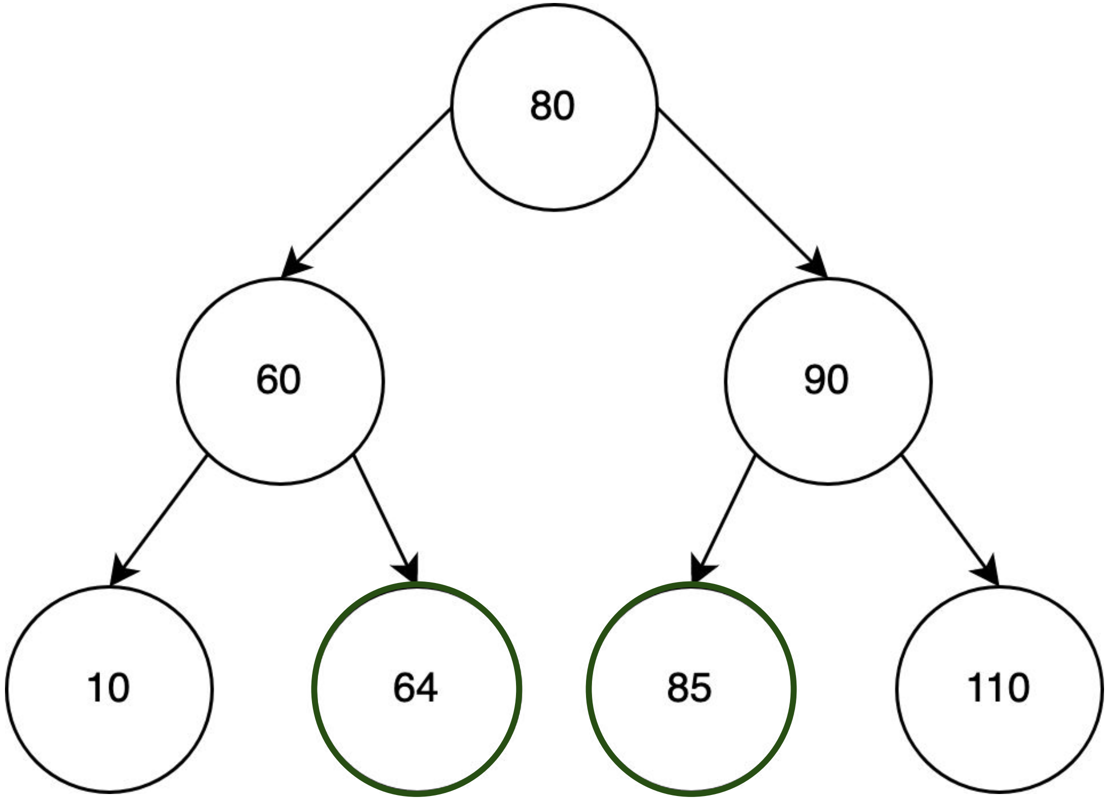
Binary Search Trees: Deletion

Deletion must maintain the properties of a BST!

Delete: <u>80</u>

Replace deleted node with either:

1. Smallest value in right subtree
2. Largest value in left subtree



Binary Search Trees: Deletion

Complexity?

Case 1: Removing a **leaf node**

O(log n)

Case 2: Removing a **node with one child**

O(log n)

Case 3: Removing a **node with two children**

O(log n)

What can go wrong?

Complexity?

Search

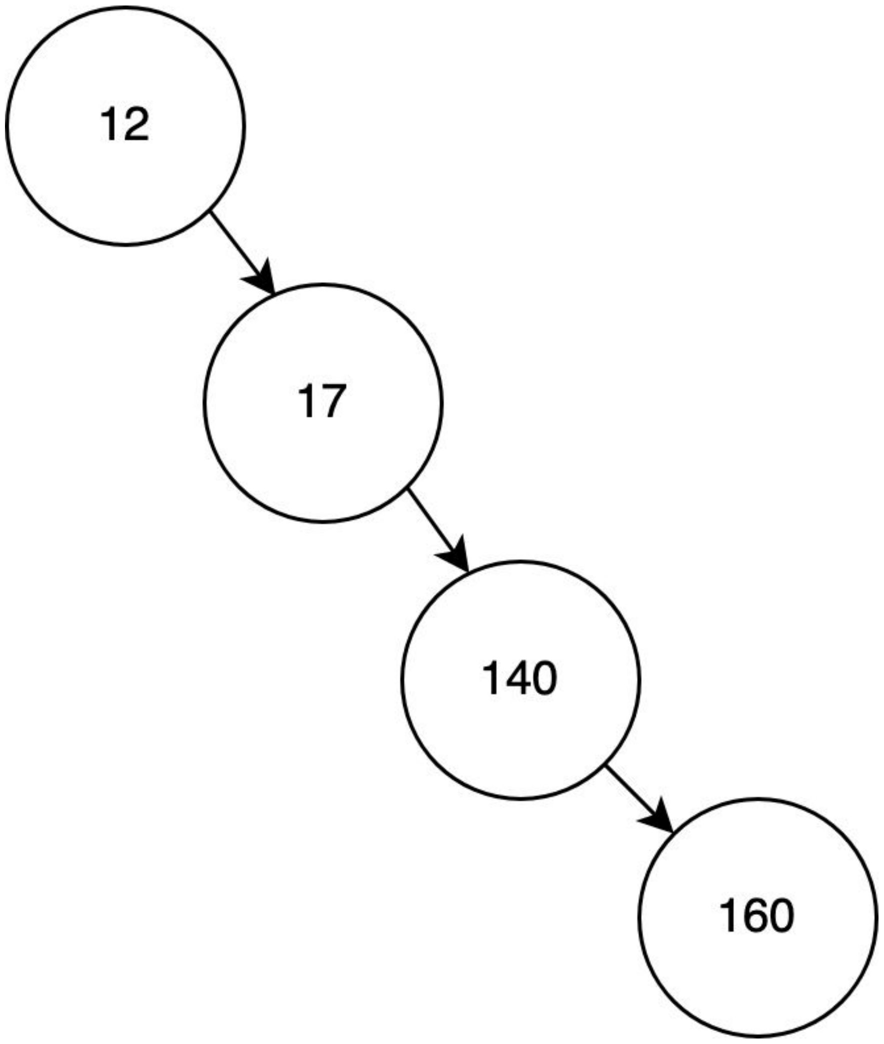
O(n)

Insertion:

O(n)

Deletion:

O(n)



Today's Lecture

1. Binary Search Trees
2. Search
3. Insertion
4. Removal
5. **Summary**

Summary

Takeaways:

Binary search trees are an efficient data structure for search

For a *balanced* binary search tree:

- Search: O(log n)
- Insertion: O(log n)
- Removal: O(log n)