Algorithms and Data Structures

Binary Trees

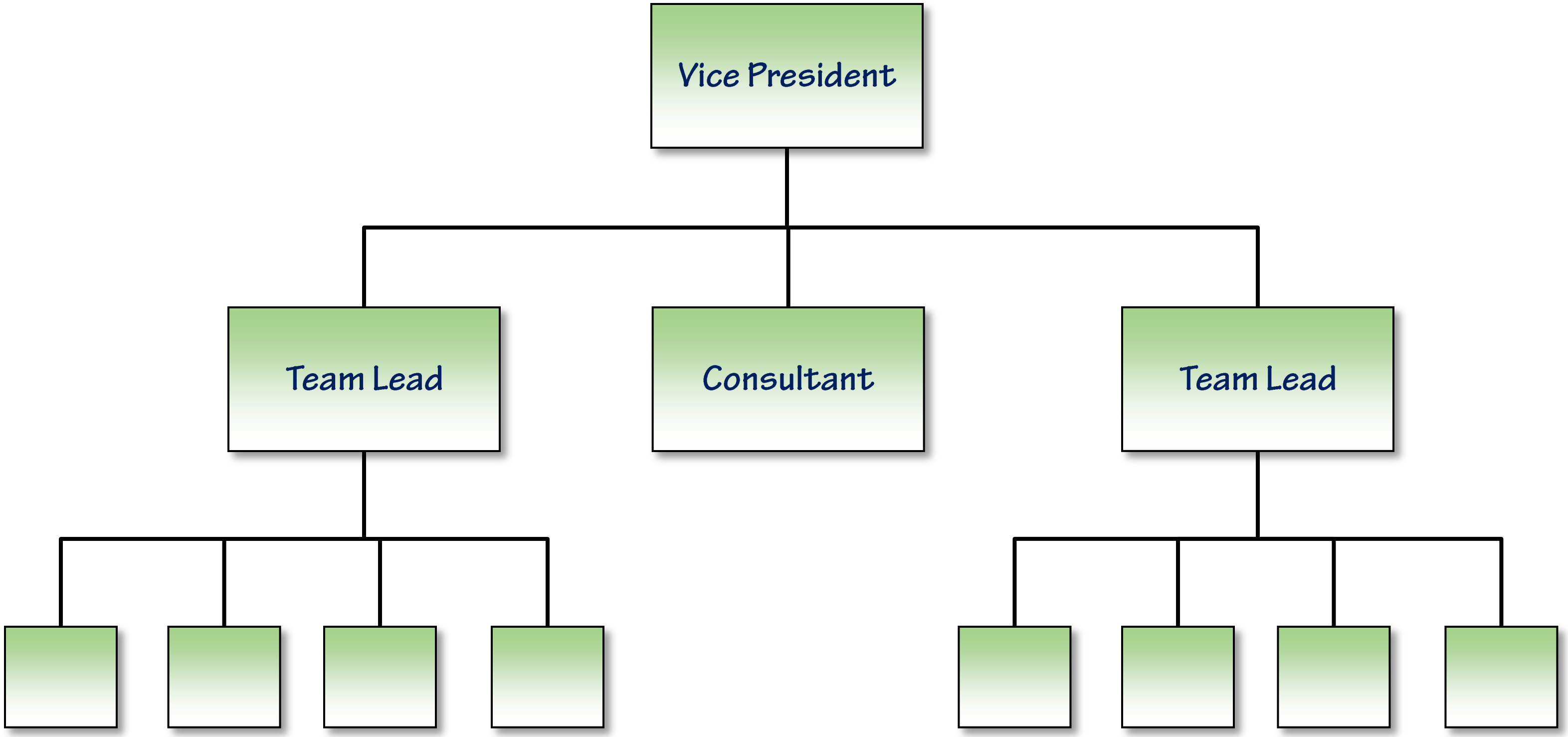
Robert Horvick

www.pluralsight.com

Outline

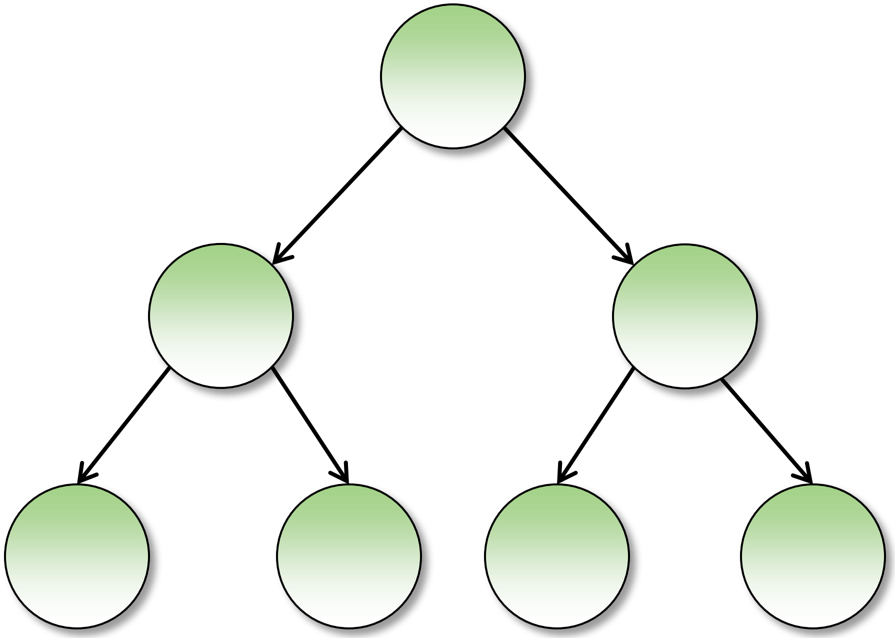
- Tree overview
- Binary Tree
  - Binary Search Tree
- Add and Remove
- Searching
- Traversals
  - Pre-Order
  - In-Order
  - Post-Order

What is a Tree?



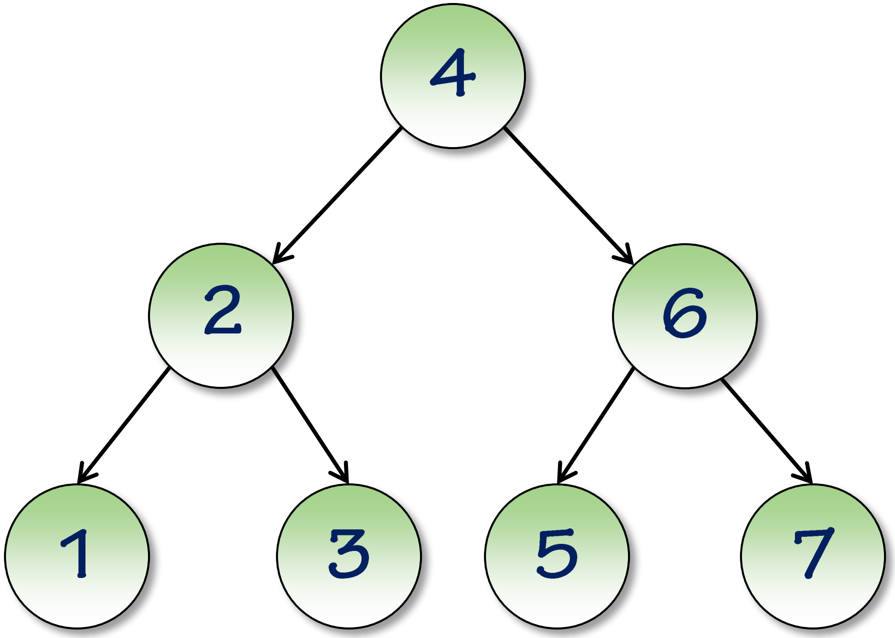
Binary Tree

- Hierarchy of Data
- A Root Node
- 0-2 Children
- Left Child
- Right Child
- Each child is itself a tree
  - Left Child
  - Right Child



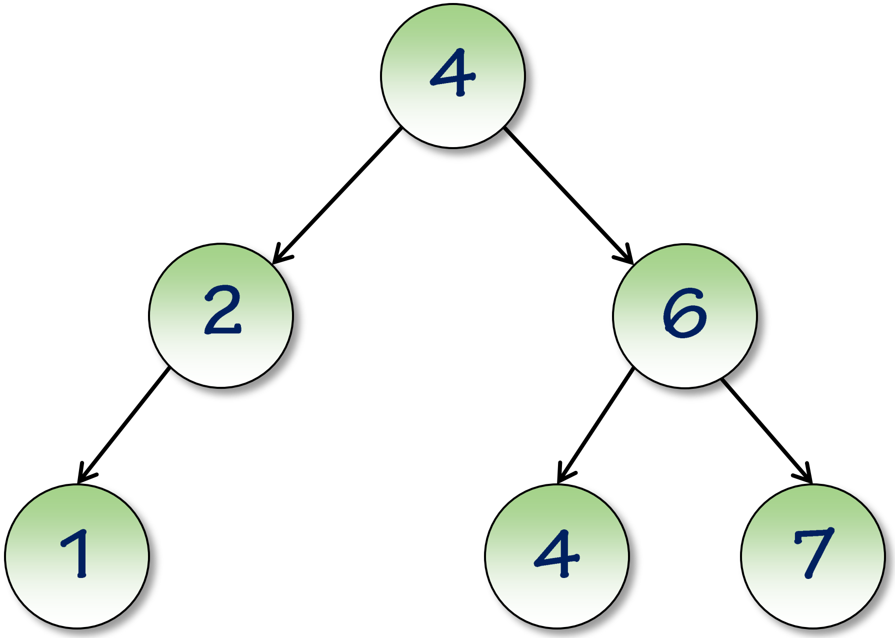
Binary Search Tree

- Sorted Hierarchy of Data
- A Root Node
- Left Child
  - Less than parent
- Right Child
  - Greater than parent
- All children follow the same rules



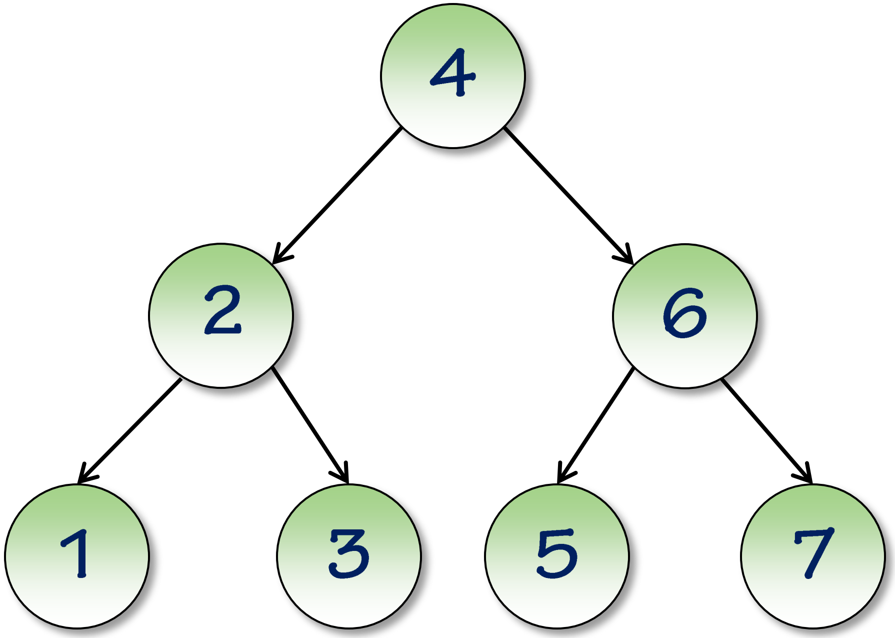
Adding Data

- **Recursive Algorithm**
- **Case 1: Empty Tree**
  - Becomes the root node
- **Case 2: Smaller Value**
  - Recursively Add to Left
- **Case 3: Larger Value**
  - Recursively Add to Right
- **Equal Values?**
  - Treat as larger value



Searching

```
Find(Node current, Data value) {
        if (current == null) {
                return null;
         }
        if (current.Value == value) {
                return current;
         }
        if (value  < current.Value) {
                return Find(current.Left, value);
         }
        return Find(current.Right, value);
}
```



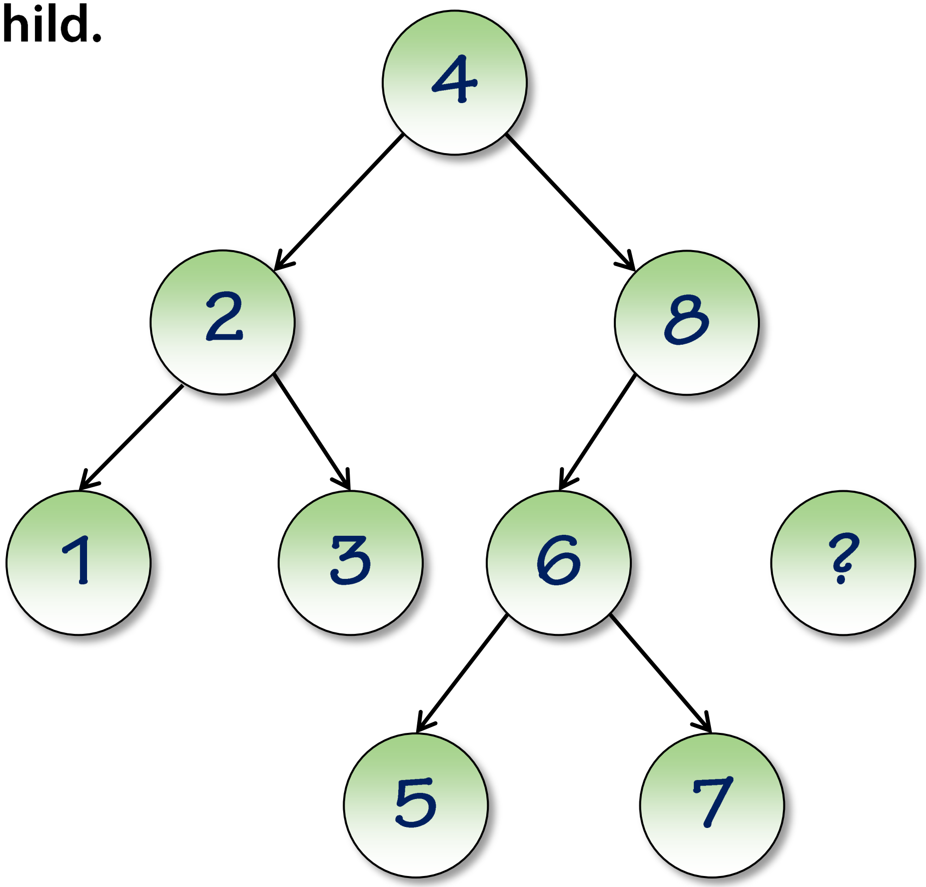
- Find(Root, 3)
- Find(Root, 5)
- Find(Root, 8)

Remove

- **Find the node to be deleted**
  - □ If the node does not exist, exit
- **Leaf (terminal) node**
  - □ Remove parent’s pointer to deleted node
- **Non-Leaf node**
  - □ Find the child to replace the deleted node
  - □ Three scenarios

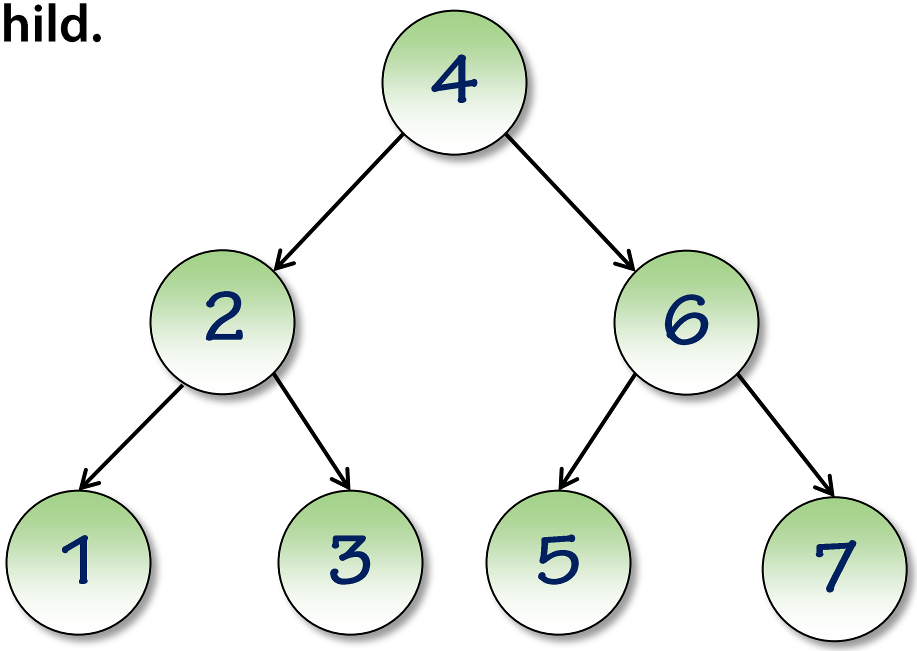
Remove (Case 1)

- **Removed node has no right child.**
  - Left child replaces removed
- **Remove(8)**
  - Find Node to remove
  - Has no right child
  - Promote left child



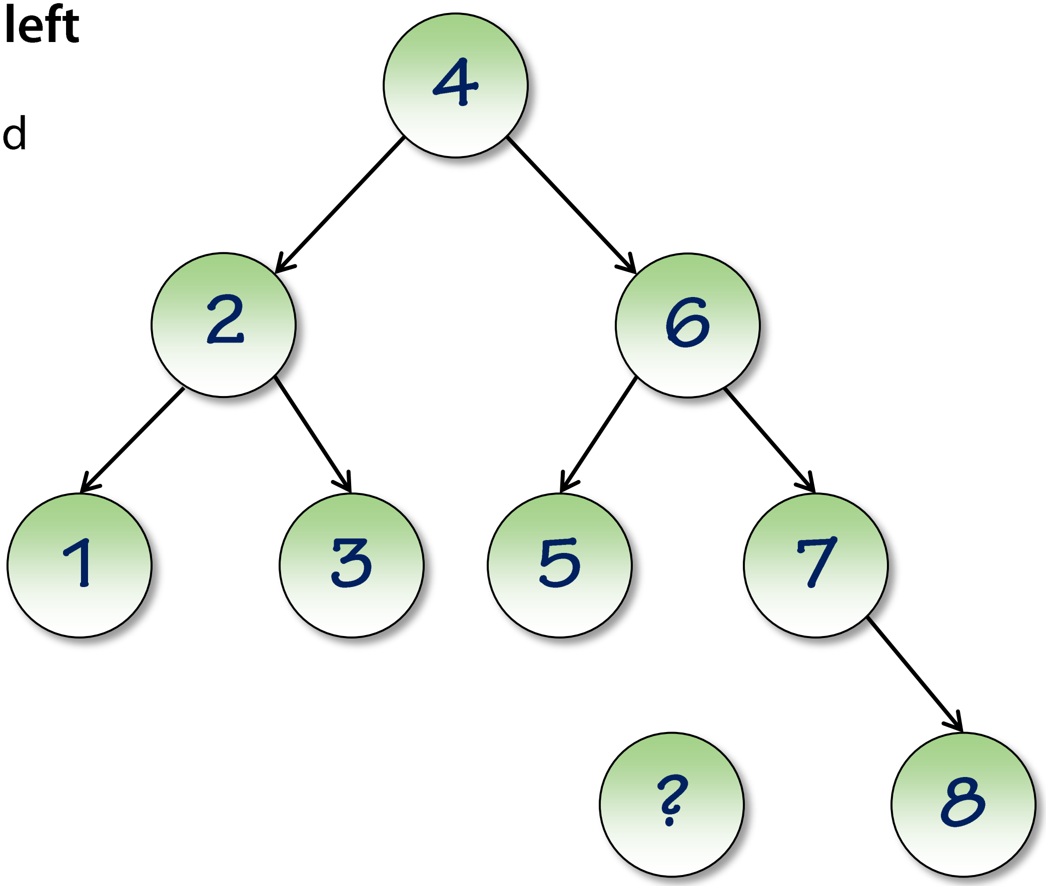
Remove (Case 1)

- **Removed node has no right child.**
  - Left child replaces removed
- **Remove(8)**
  - Find Node to remove
  - Has no right child
  - Promote left child



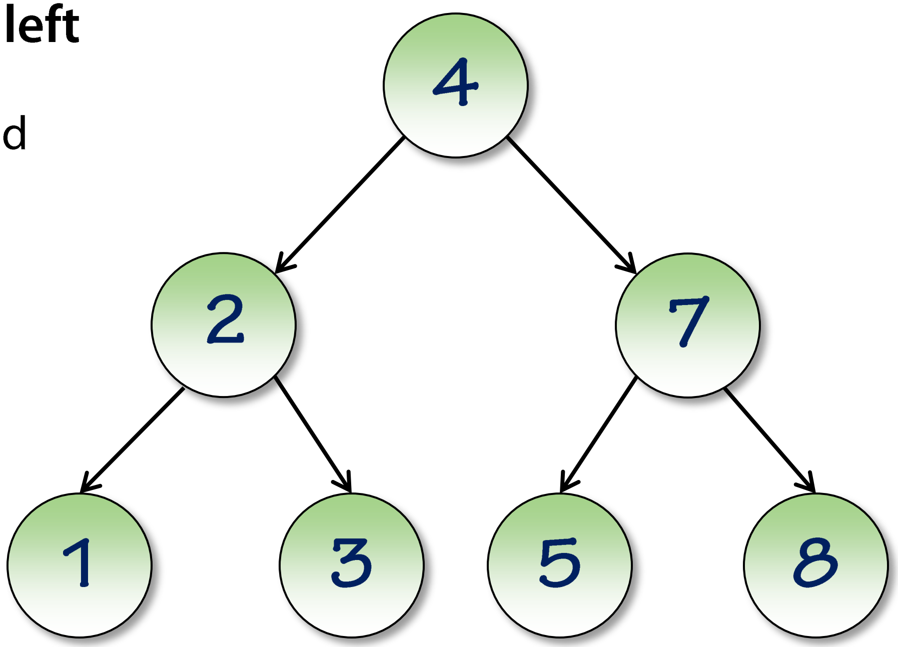
Remove (Case 2)

- **Removed right child has no left**
  - Right child replaces removed
- **Remove(6)**
  - Find Node to remove
  - Node right has no left
  - Promote right child



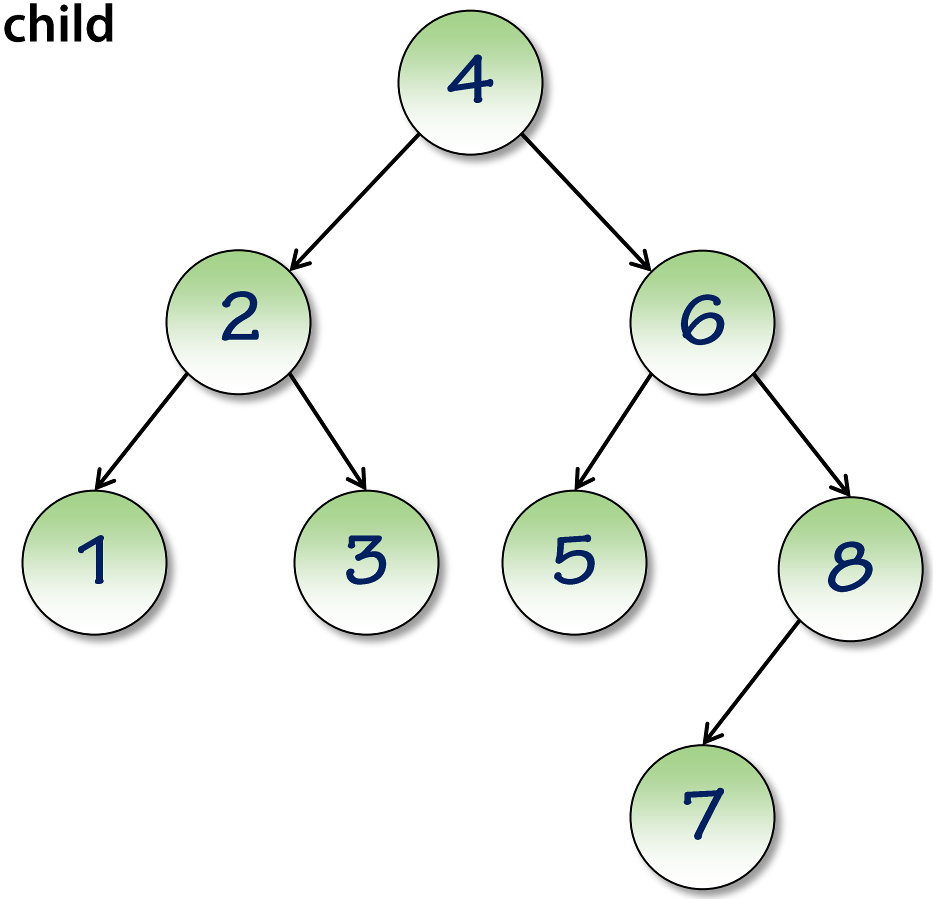
Remove (Case 2)

- **Removed right child has no left**
  - Right child replaces removed
- **Remove(6)**
  - Find Node to remove
  - Node right has no left
  - Promote right child



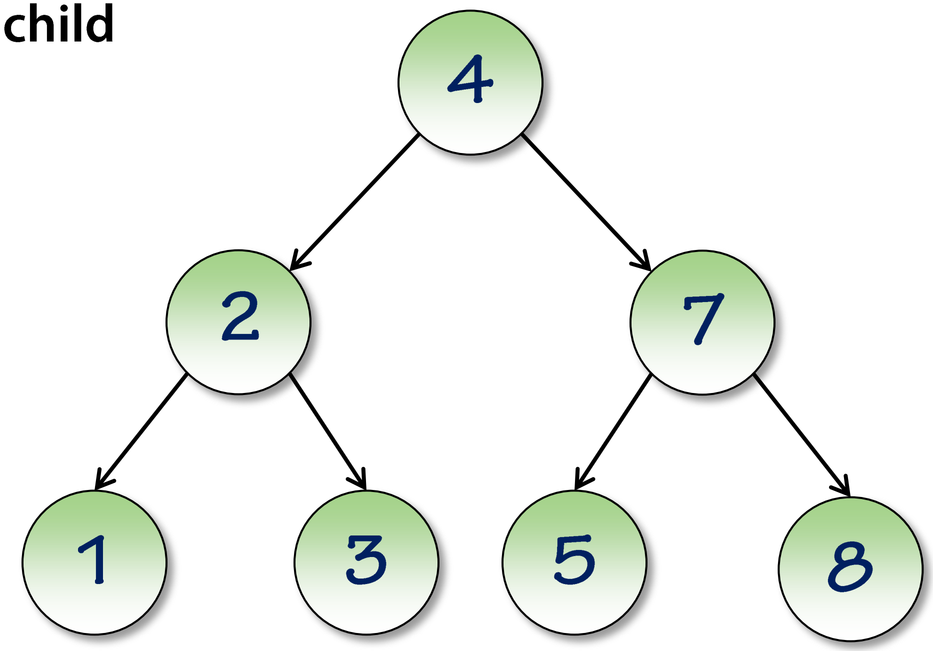
Remove (Case 3)

- **Removed right child has left child**
  - Right child's left-most child replaces removed
- **Remove(6)**
  - Find Node to remove
  - Node right has left
  - Find right's left-most child
  - Promote left-most child



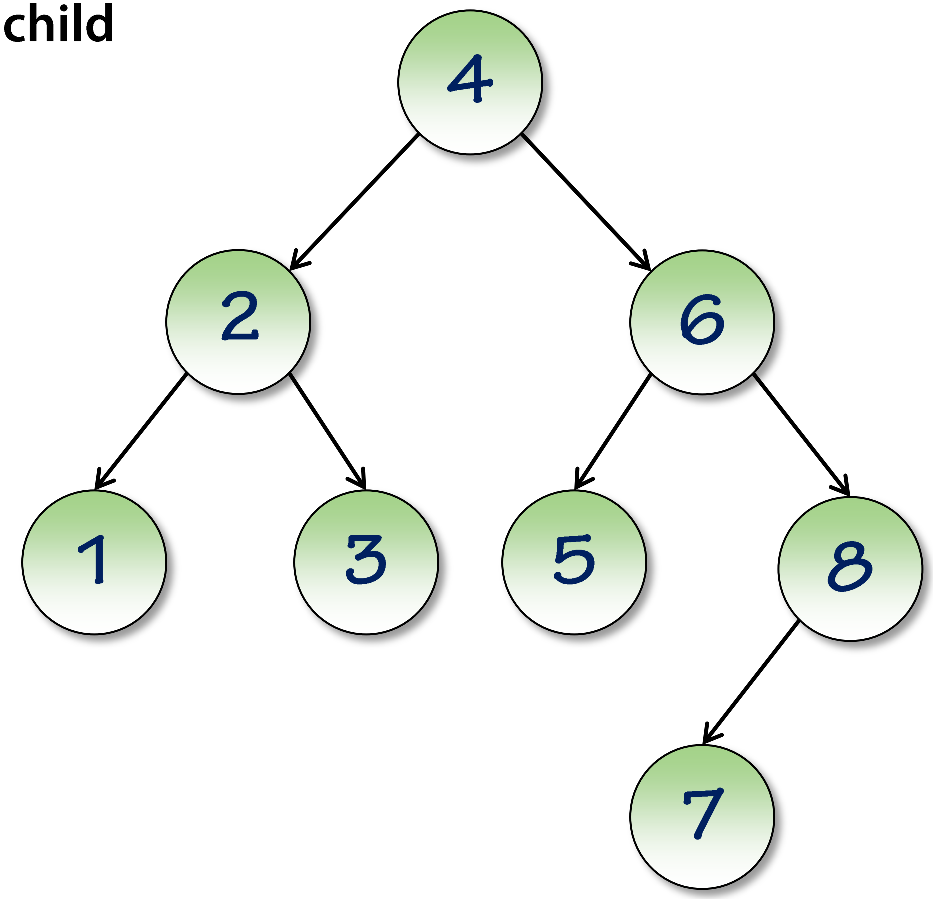
Remove (Case 3)

- **Removed right child has left child**
  - □ Right child's left-most child replaces removed
- **Remove(6)**
  - □ Find Node to remove
  - □ Node right has left
  - □ Find right's left-most child
  - □ Promote left-most child



Remove (Case 3)

- **Removed right child has left child**
  - Right child's left-most child replaces removed
- **Remove(6)**
  - Find Node to remove
  - Node right has left
  - Find right's left-most child
  - Promote left-most child

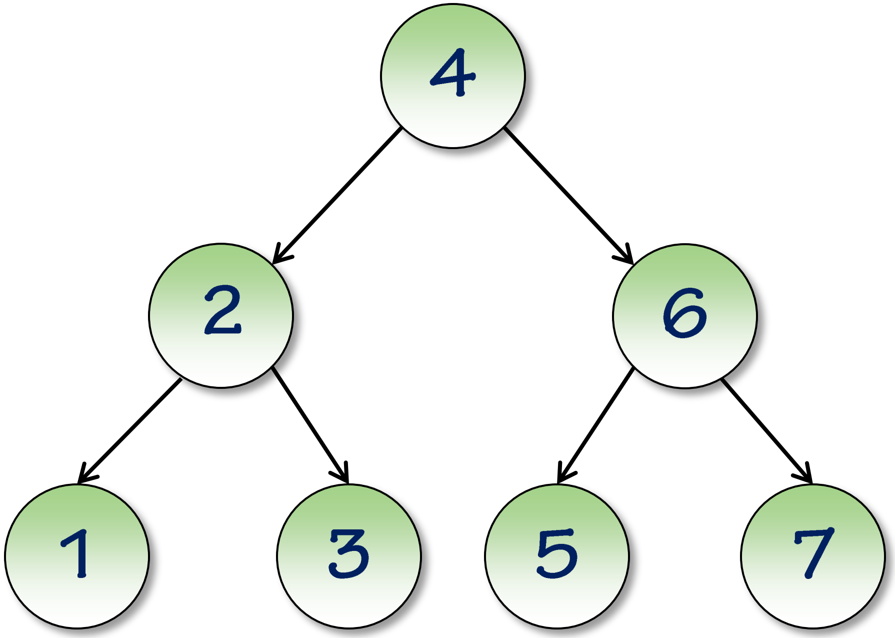


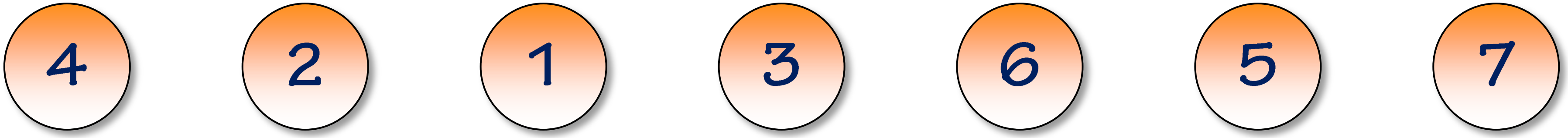
Tree Traversals

- **Enumerate nodes in a well-defined order**
- **Basic algorithm**
  - Process node
  - Visit Left
  - Visit Right
- **What varies is the order**
- **Three Common Orders**
  - Pre-Order
  - In-Order
  - Post-Order

Pre-Order Traversal

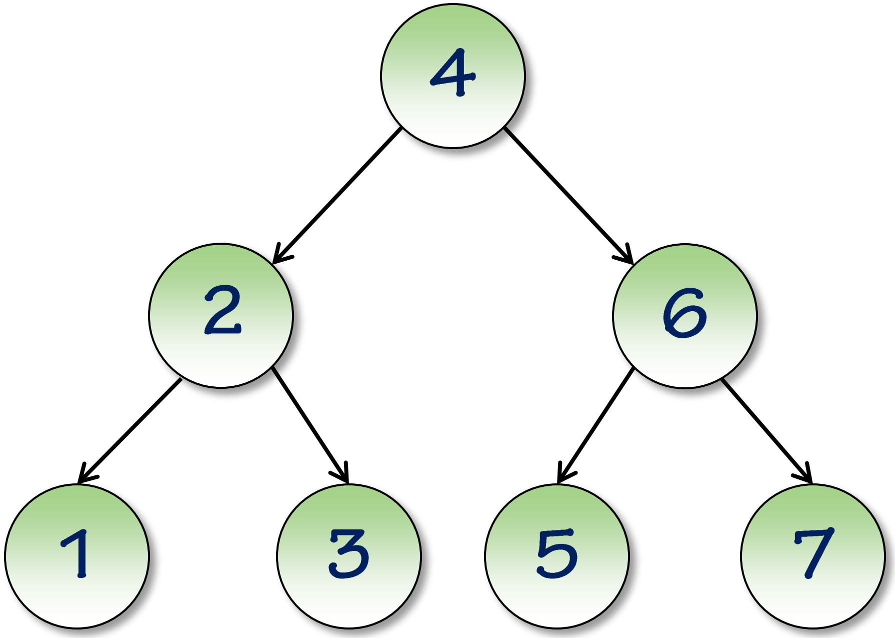
```
Visit(Node current) {
    if ( current == null ) {
        return;
    }
    Process(current.Value);
    Visit(current.Left);
    Visit(current.Right);
}
```

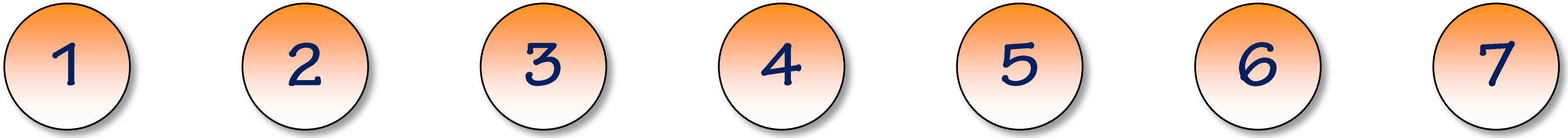




In-Order Traversal

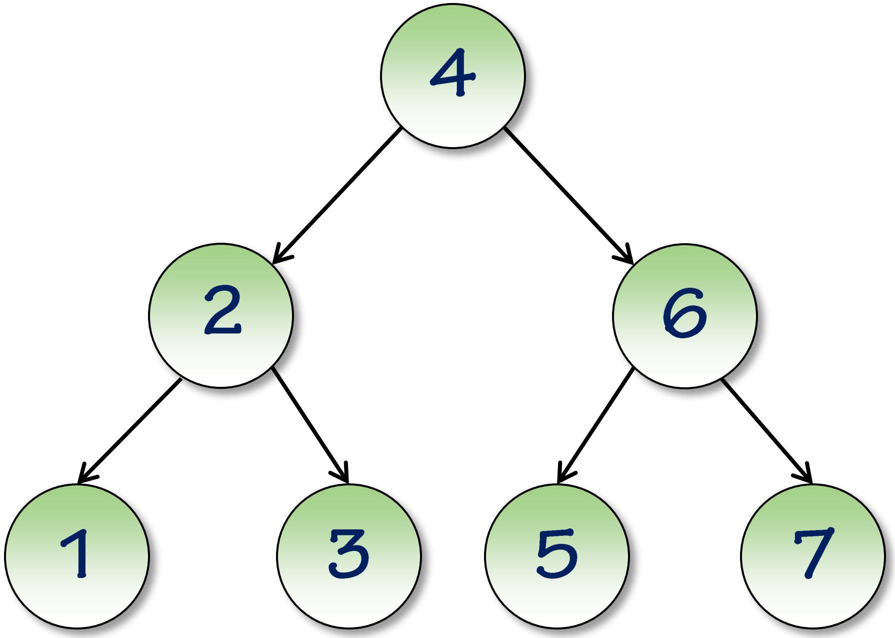
```
Visit(Node current) {
    if ( current == null ) {
        return;
    }
    Visit(current.Left);
    Process(current.Value);
    Visit(current.Right);
}
```

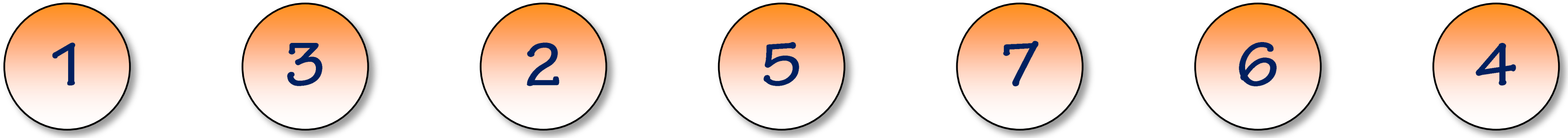




Post-Order Traversal

```
Visit(Node current) {
    if ( current == null ) {
        return;
    }
    Visit(current.Left);
    Visit(current.Right);
    Process(current.Value);
}
```





Summary

- **Binary Search Tree**
  - Smaller values on left
  - Larger values on right
- **Add and Remove**
- **Searching**
- **Traversals**
  - Pre-Order
  - In-Order
  - Post-Order