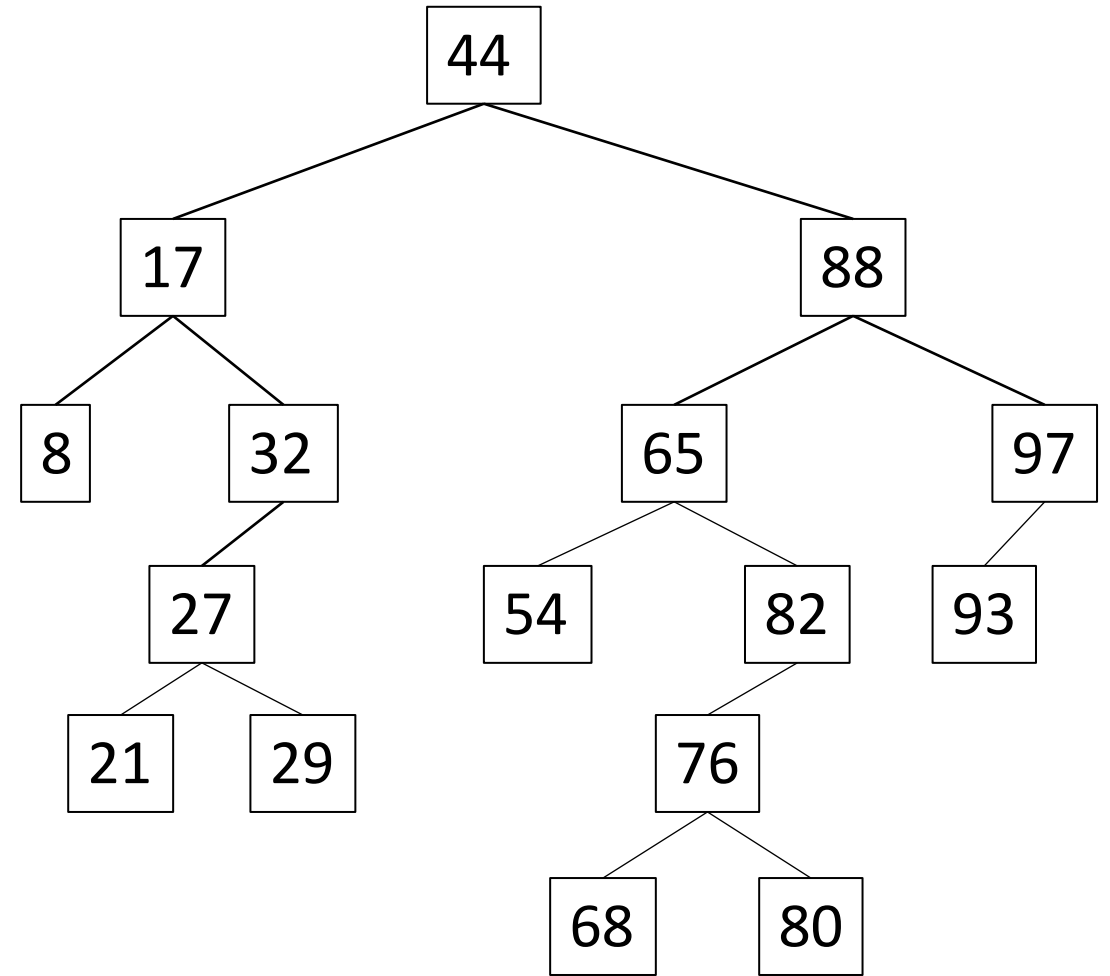


Binary Search Trees

CSCI 232

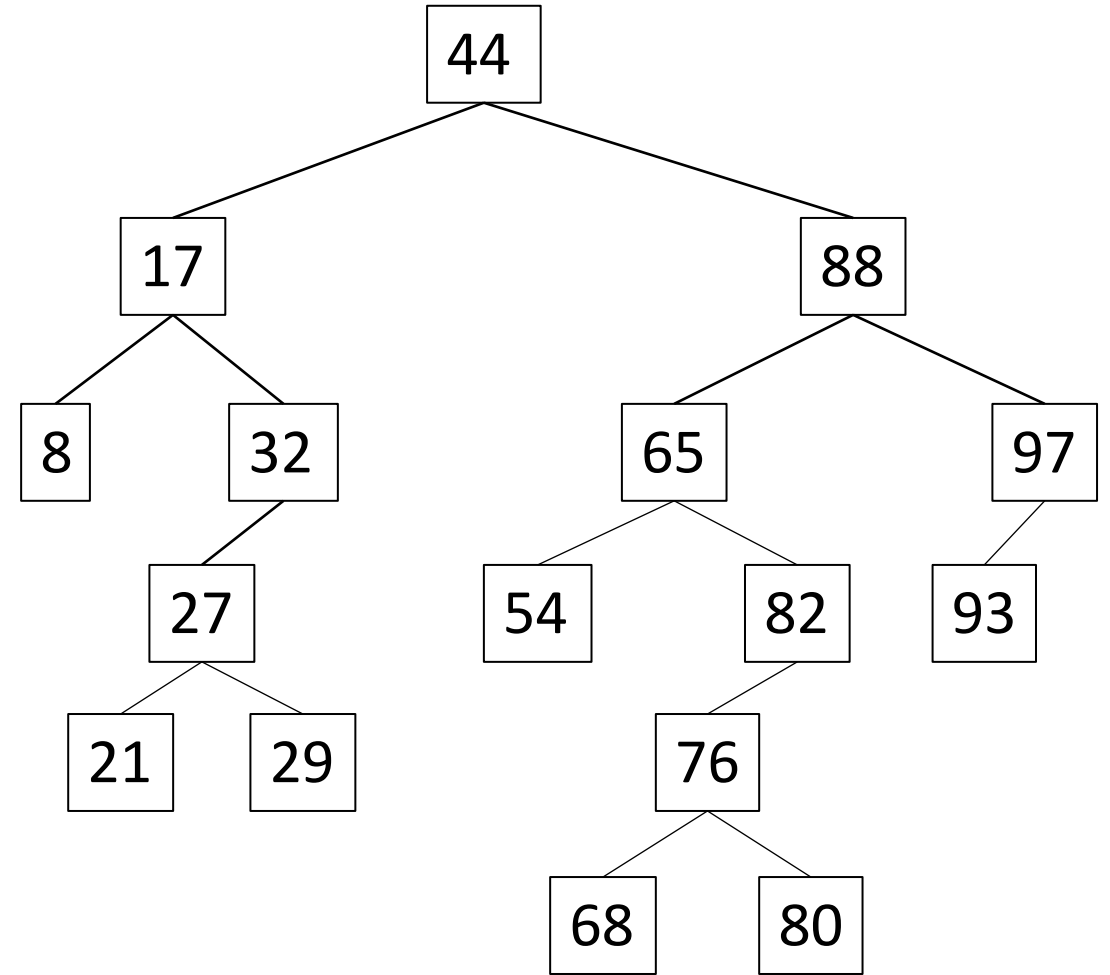
Binary Search Tree



Binary Search Tree

Binary Search Tree (BST) properties:

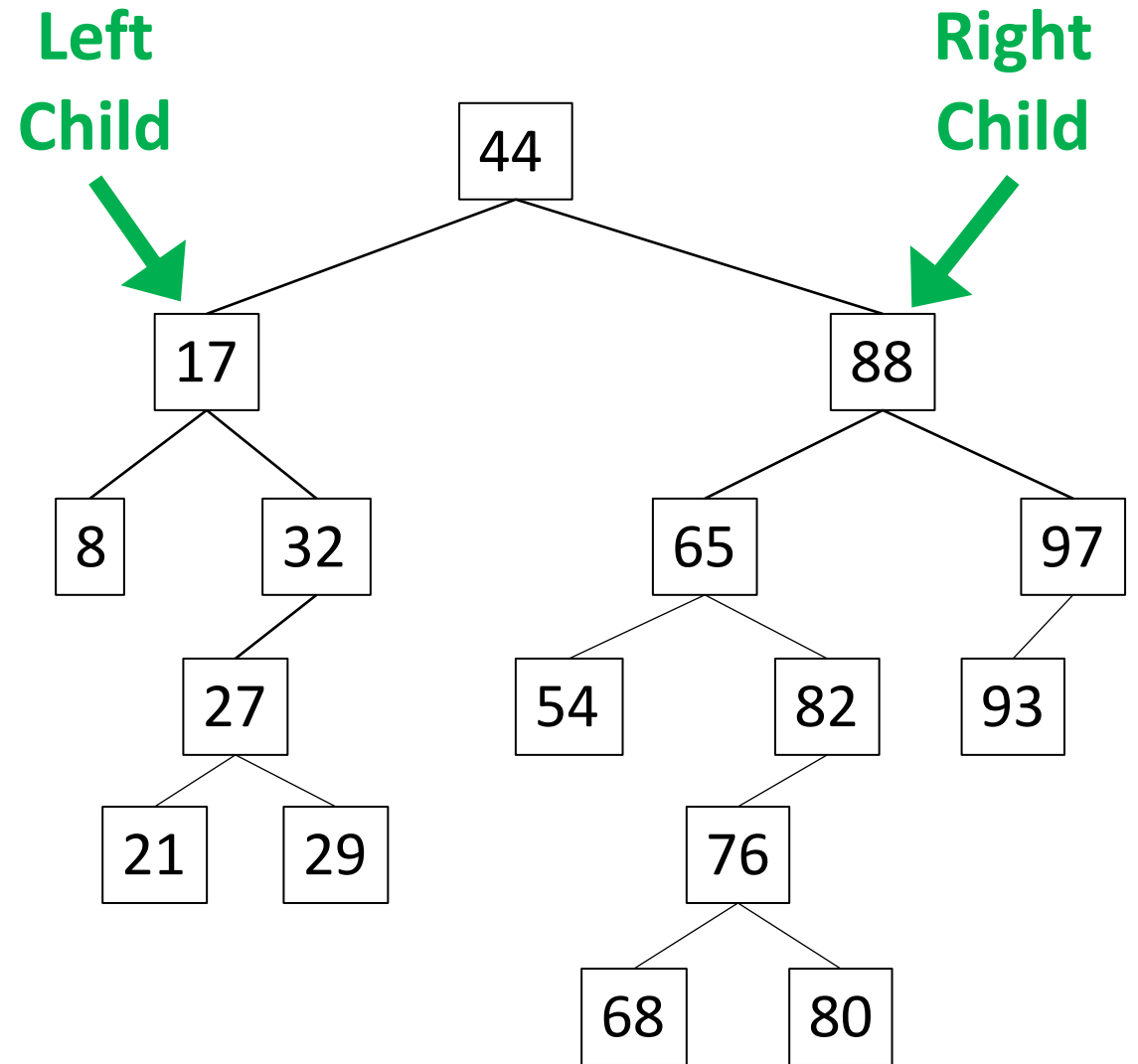
- A BST is composed of Comparable data elements.



Binary Search Tree

Binary Search Tree (BST) properties:

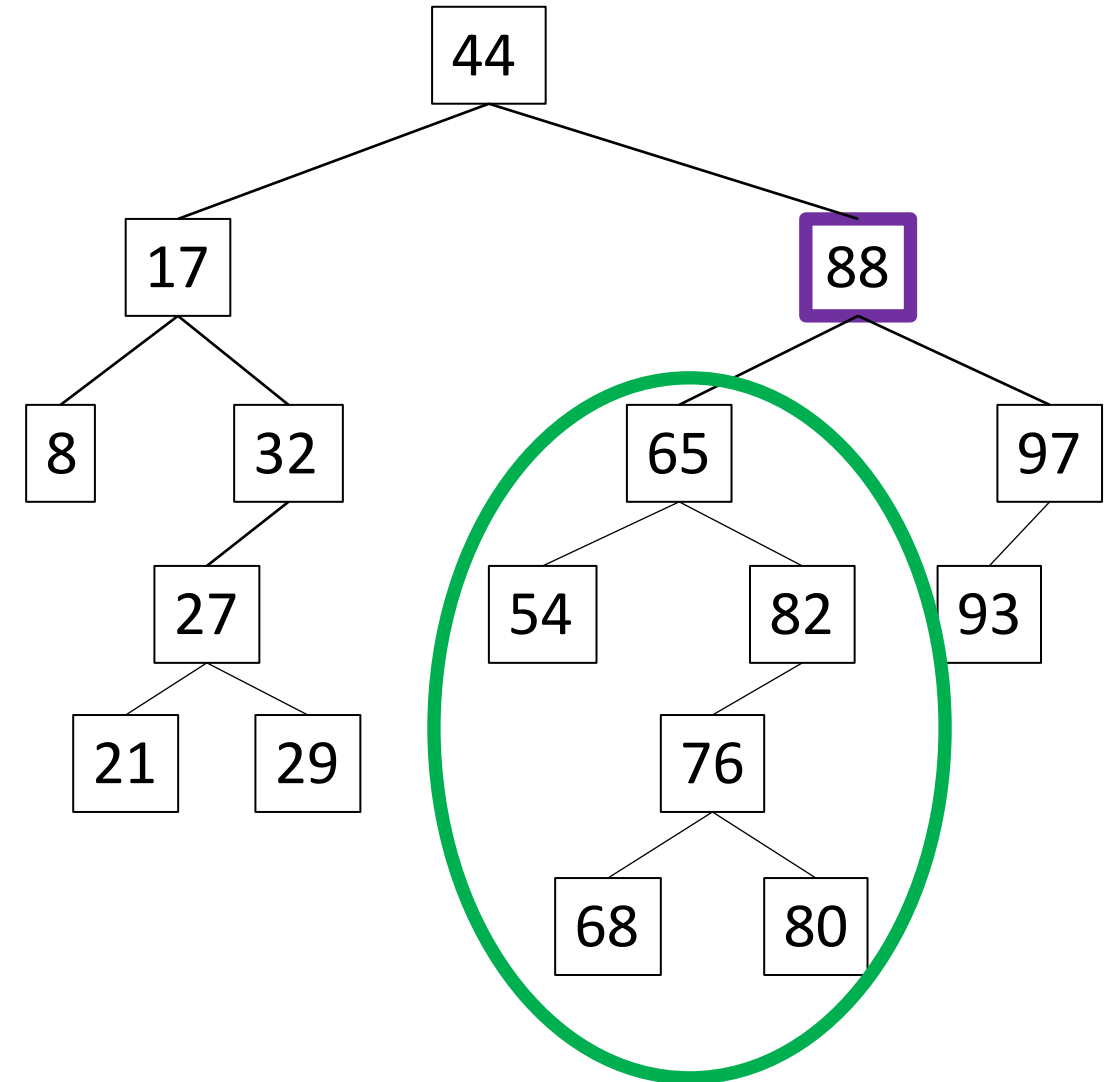
- A BST is composed of Comparable data elements.
- A BST is a binary tree (each node has at most two children).



Binary Search Tree

Binary Search Tree (BST) properties:

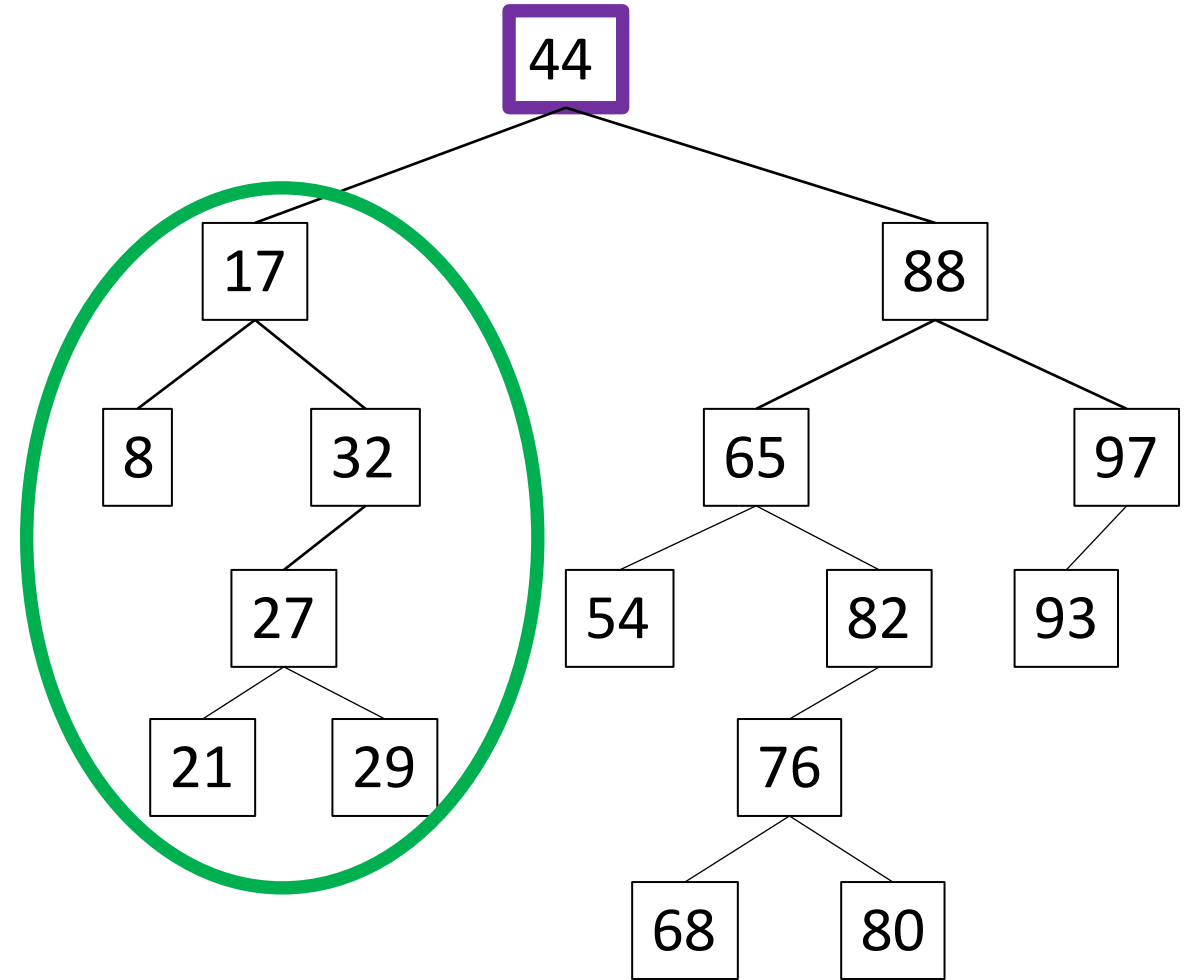
- A BST is composed of Comparable data elements.
- A BST is a binary tree (each node has at most two children).
- For each **node**, all **left-hand descendants** have values that are less than the node.



Binary Search Tree

Binary Search Tree (BST) properties:

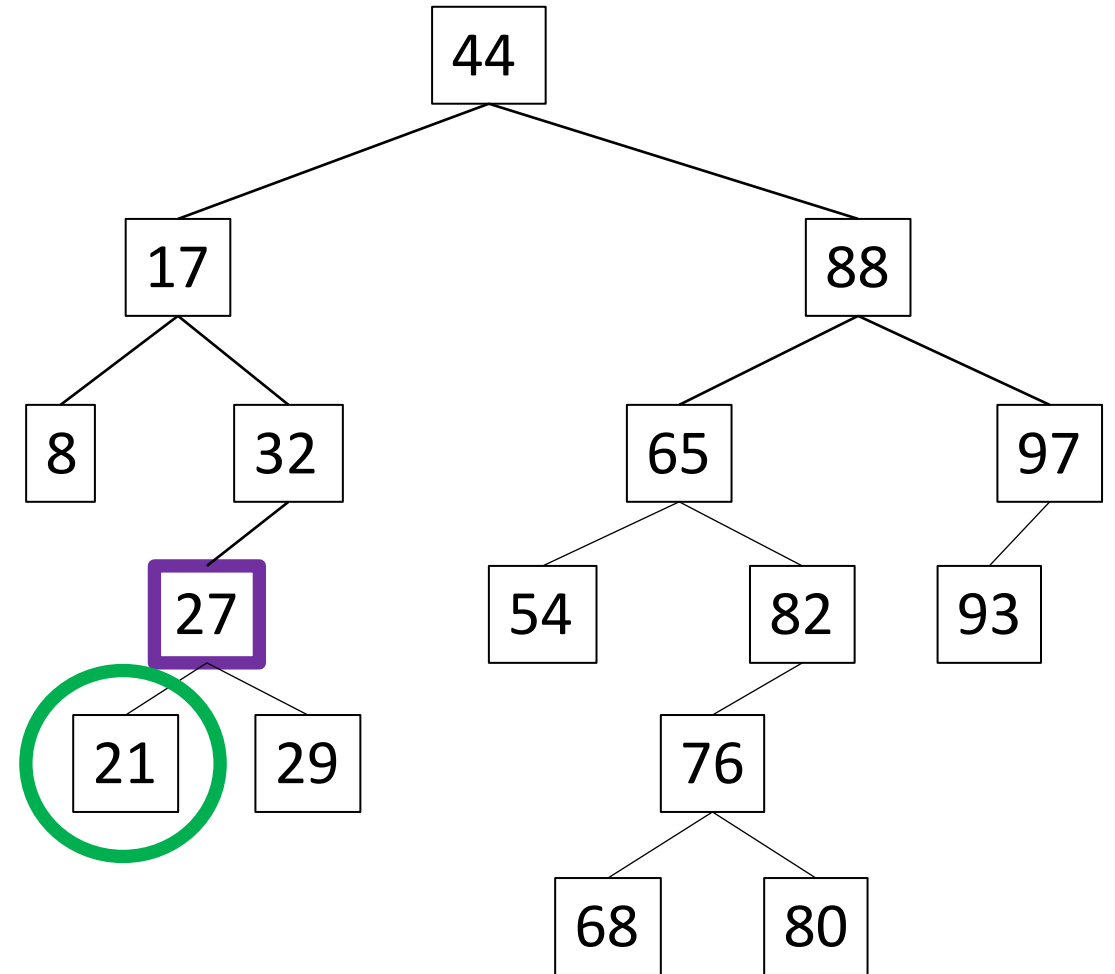
- A BST is composed of Comparable data elements.
- A BST is a binary tree (each node has at most two children).
- For each **node**, all **left-hand descendants** have values that are less than the node.



Binary Search Tree

Binary Search Tree (BST) properties:

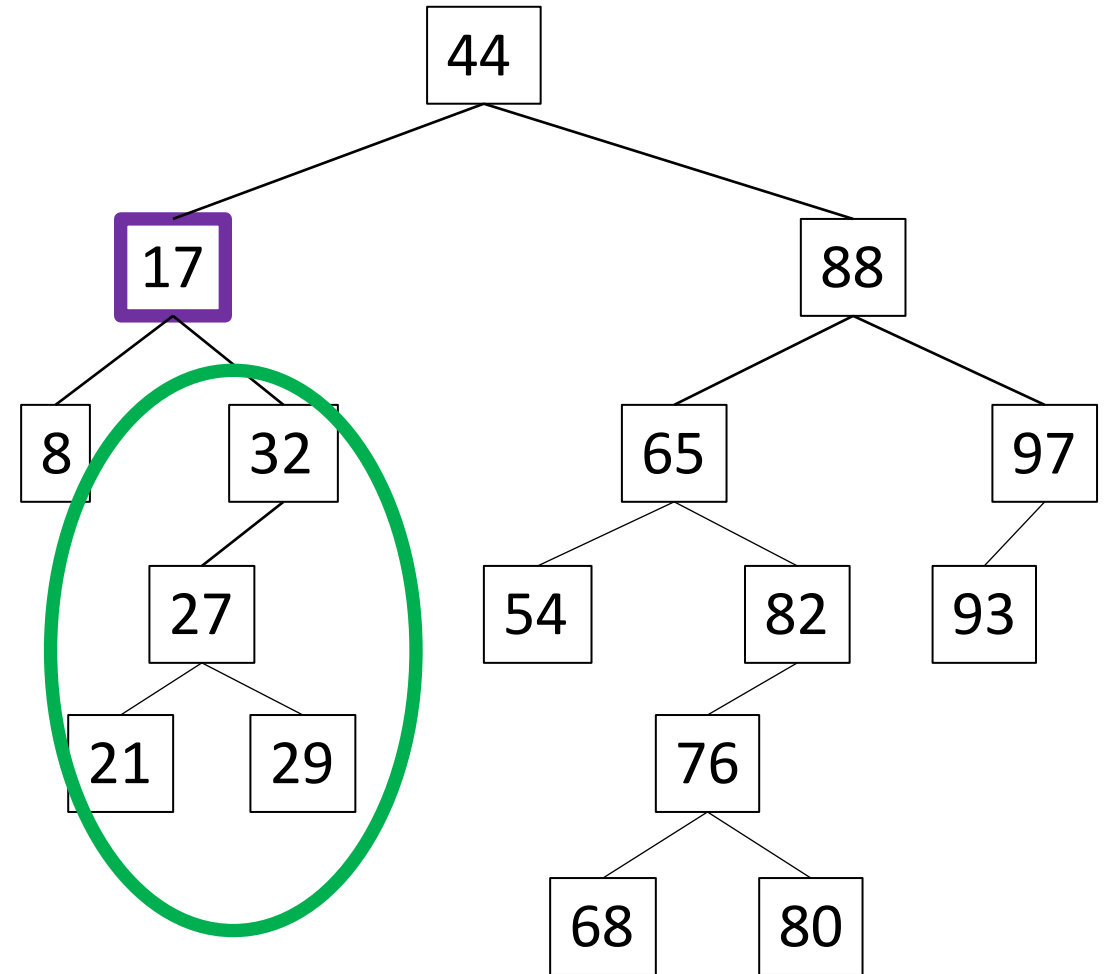
- A BST is composed of Comparable data elements.
- A BST is a binary tree (each node has at most two children).
- For each **node**, all **left-hand descendants** have values that are less than the node.



Binary Search Tree

Binary Search Tree (BST) properties:

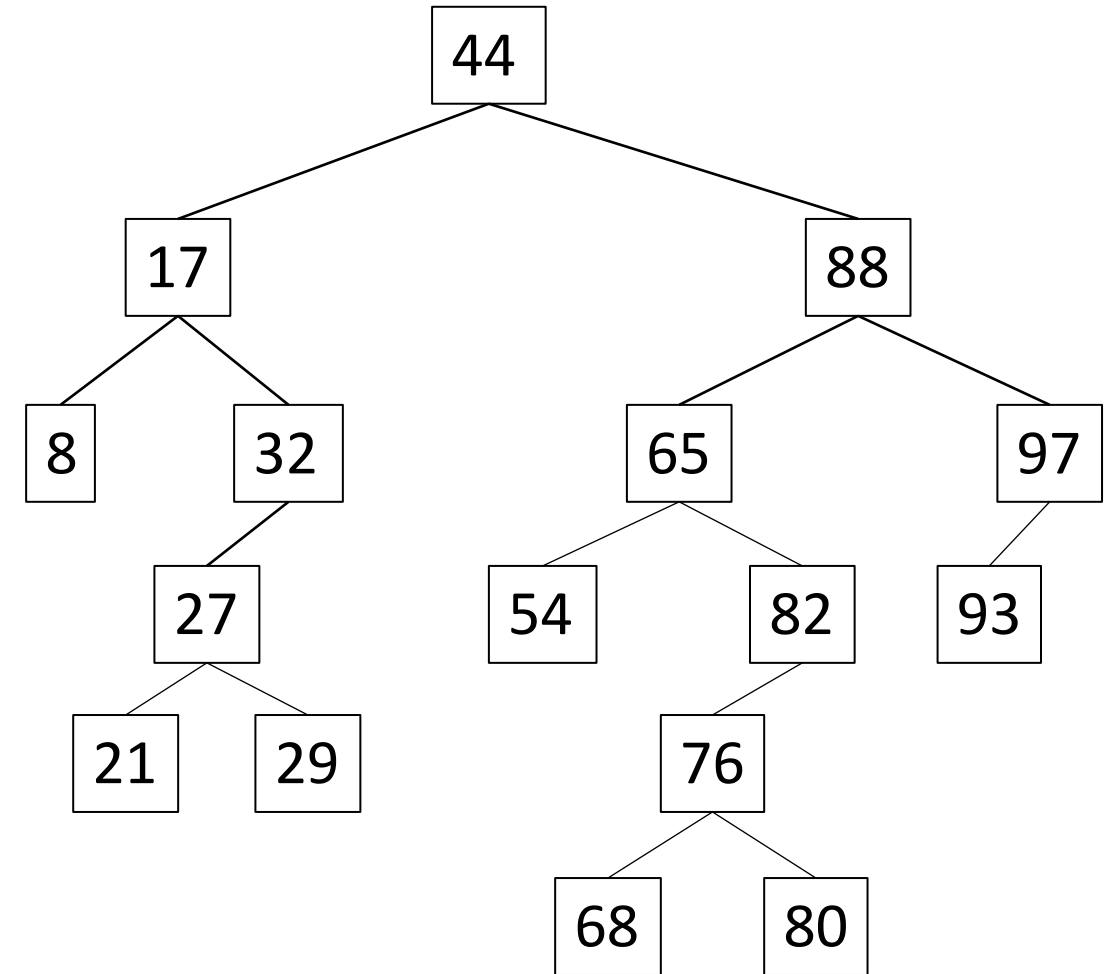
- A BST is composed of Comparable data elements.
- A BST is a binary tree (each node has at most two children).
- For each node, all left-hand descendants have values that are less than the node.
- For each node, all right-hand descendants have values that are larger than the node.



Binary Search Tree

Binary Search Tree (BST) properties:

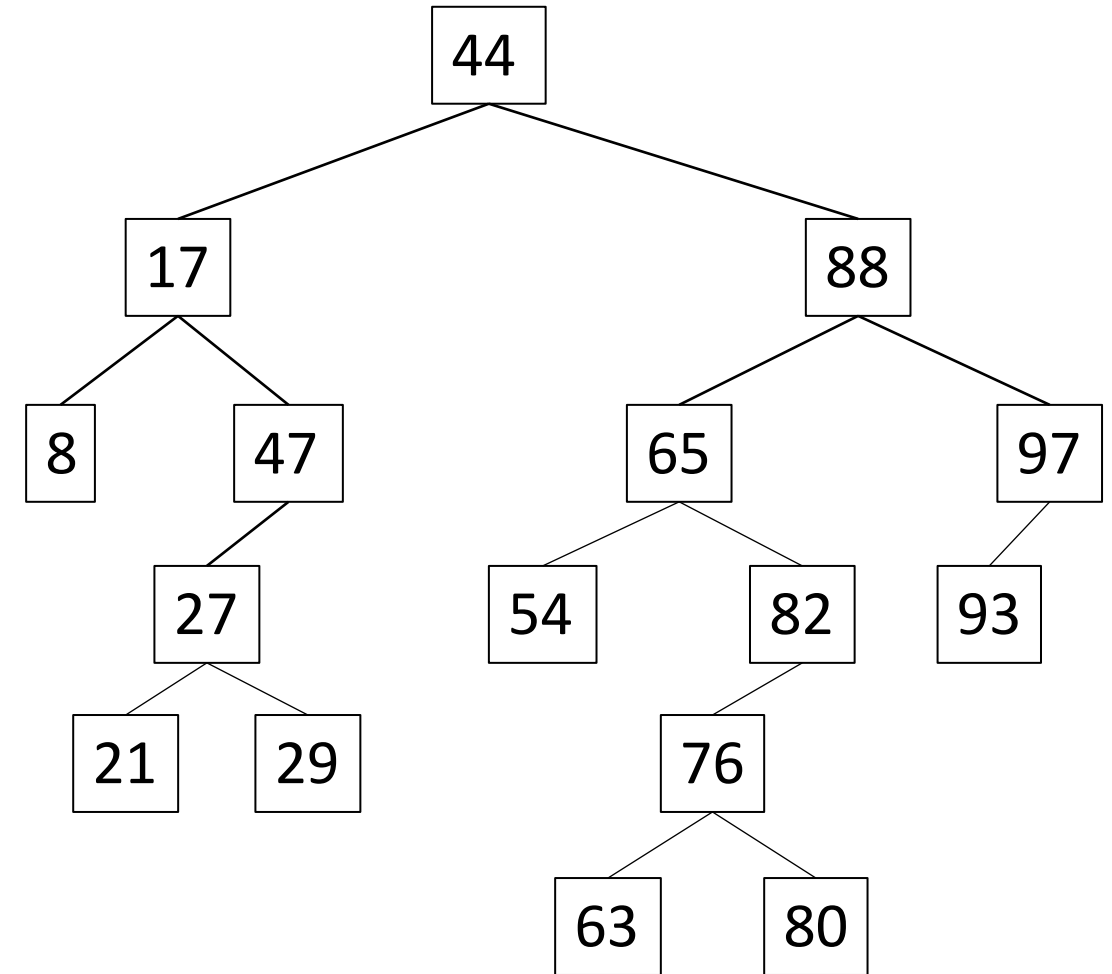
- A BST is composed of Comparable data elements.
- A BST is a binary tree (each node has at most two children).
- For each node, all left-hand descendants have values that are less than the node.
- For each node, all right-hand descendants have values that are larger than the node.
- There are no duplicate values (definitions vary).



Binary Search Tree

Binary Search Tree (BST) properties:

- **A BST is composed of Comparable data elements.**
- A BST is a binary tree (each node has at most two children).
- For each node, all left-hand descendants have values that are less than the node.
- For each node, all right-hand descendants have values that are larger than the node.
- There are no duplicate values (definitions vary).

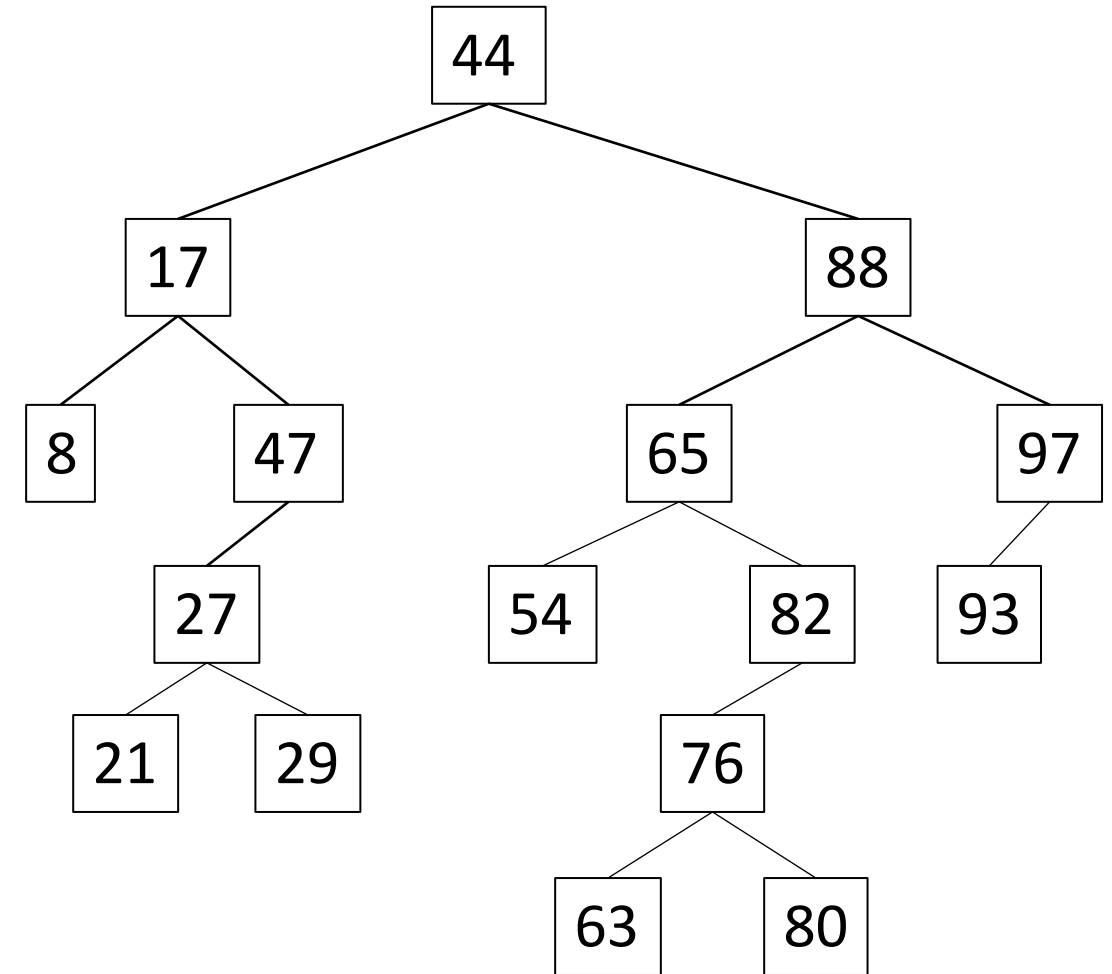


Is it a BST?

Binary Search Tree

Binary Search Tree (BST) properties:

- **A BST is composed of Comparable data elements.**
- **A BST is a binary tree (each node has at most two children).**
- For each node, all left-hand descendants have values that are less than the node.
- For each node, all right-hand descendants have values that are larger than the node.
- There are no duplicate values (definitions vary).

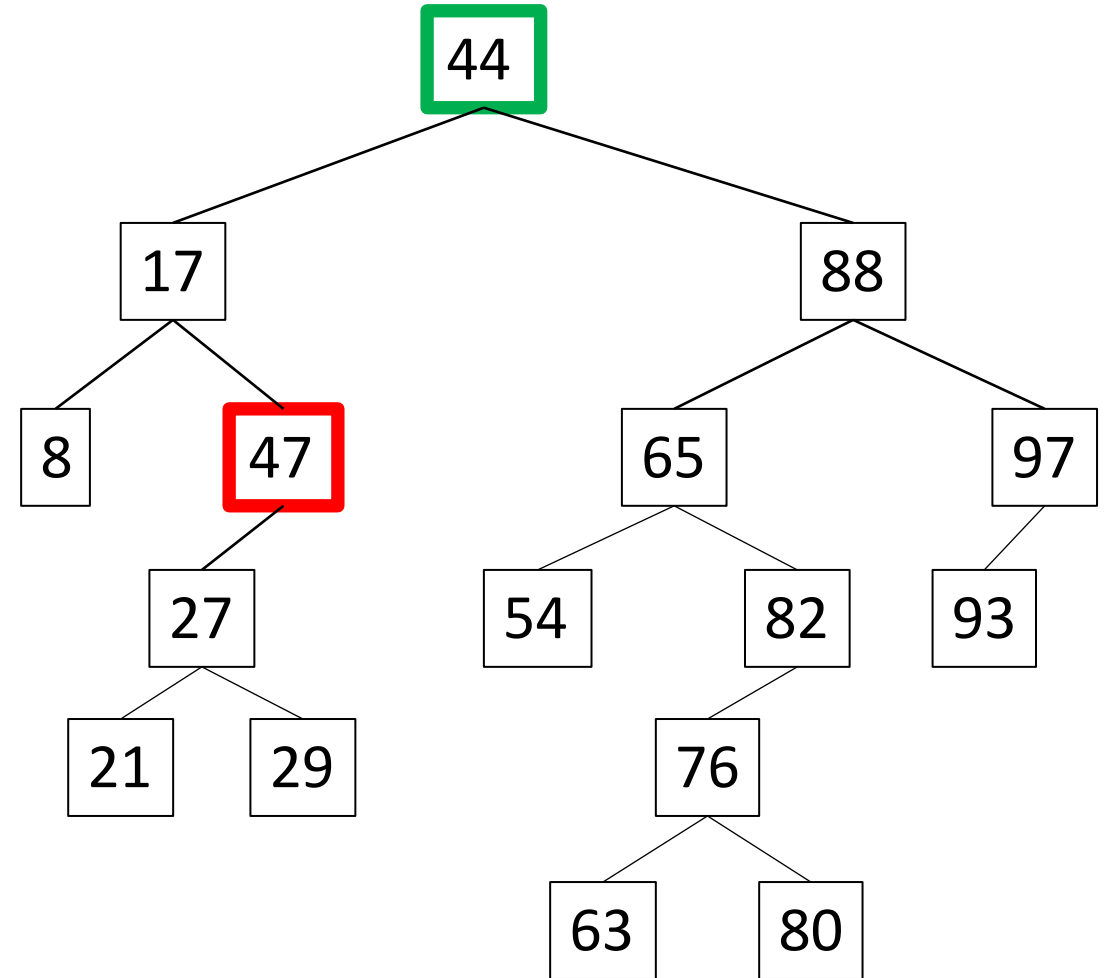


Is it a BST?

Binary Search Tree

Binary Search Tree (BST) properties:

- A BST is composed of Comparable data elements.
- A BST is a binary tree (each node has at most two children).
- For each node, all left-hand descendants have values that are less than the node.
- For each node, all right-hand descendants have values that are larger than the node.
- There are no duplicate values (definitions vary).

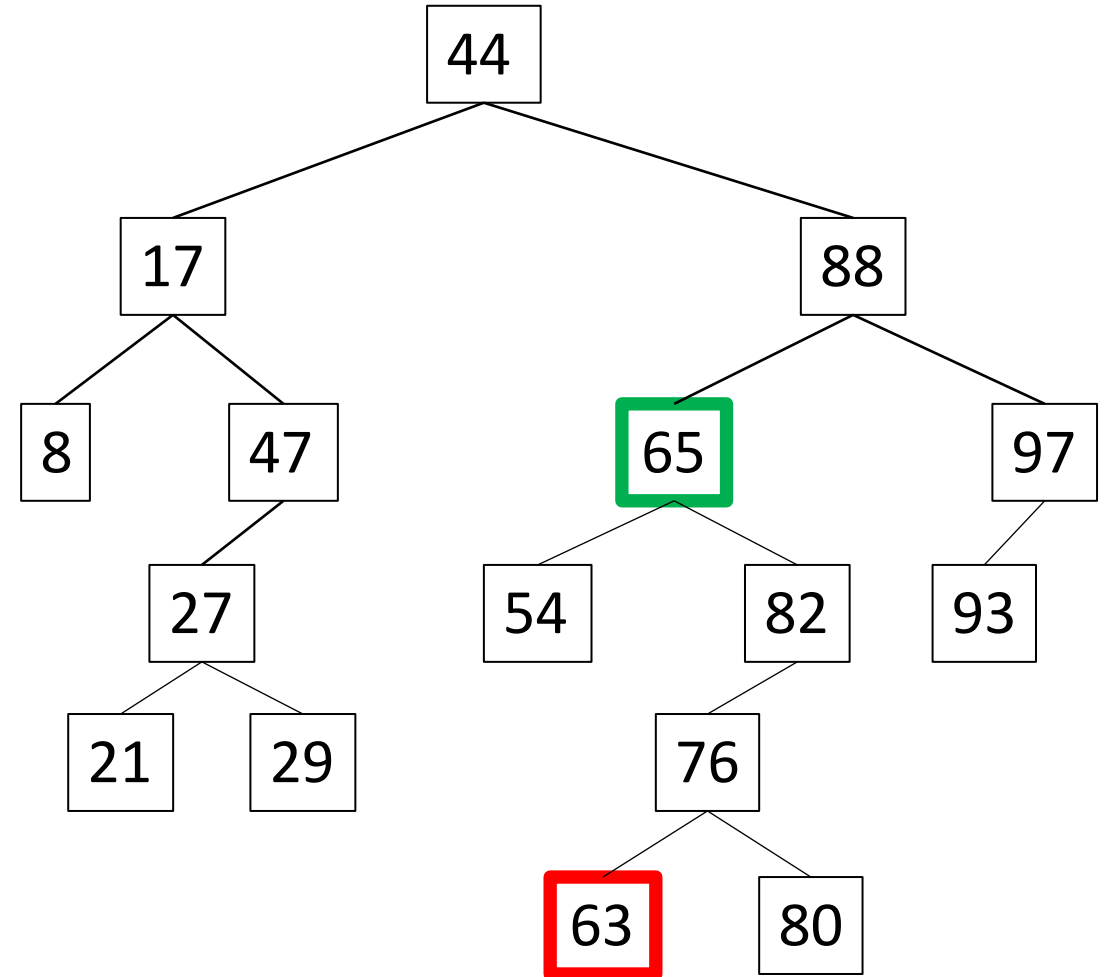


Is it a BST?

Binary Search Tree

Binary Search Tree (BST) properties:

- A BST is composed of Comparable data elements.
- A BST is a binary tree (each node has at most two children).
- For each node, all left-hand descendants have values that are less than the node.
- For each node, all right-hand descendants have values that are larger than the node.
- There are no duplicate values (definitions vary).

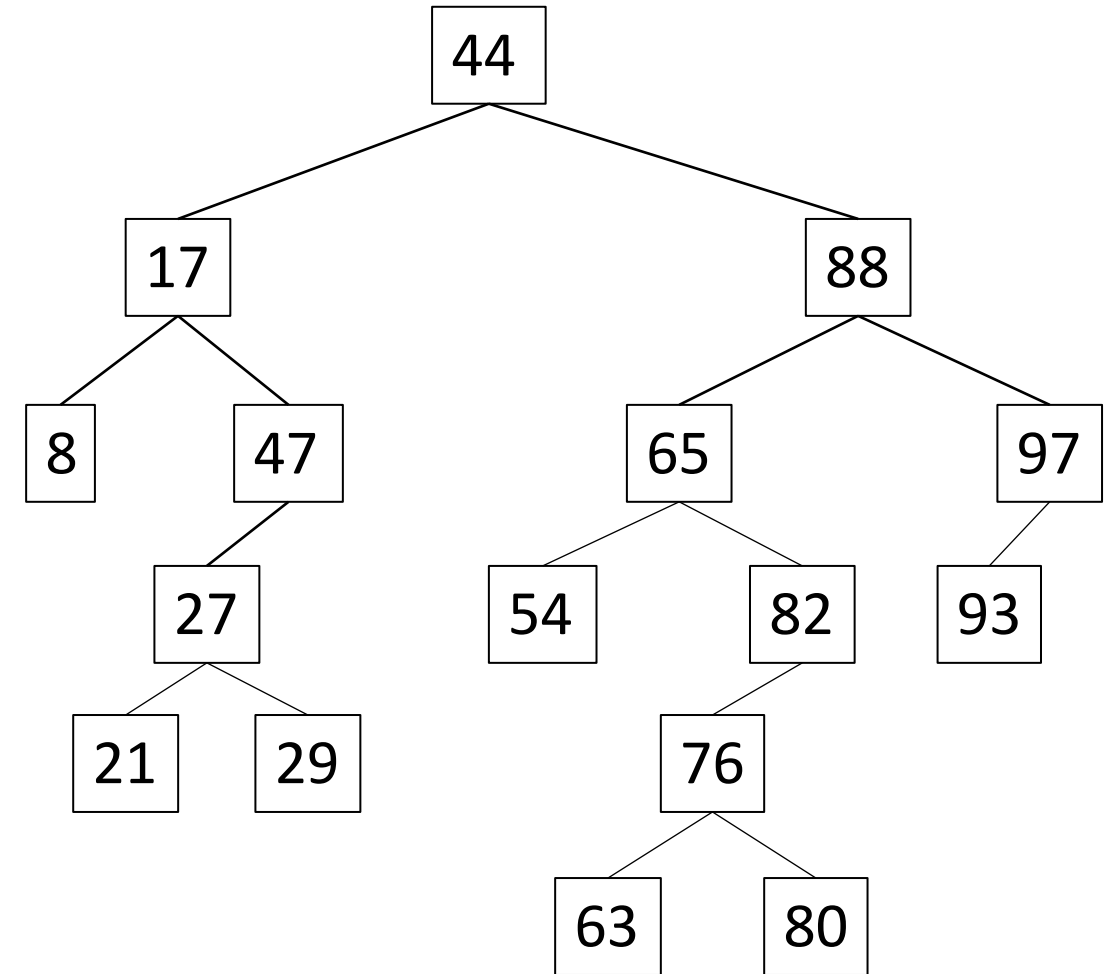


Is it a BST?

Binary Search Tree

Binary Search Tree (BST) properties:

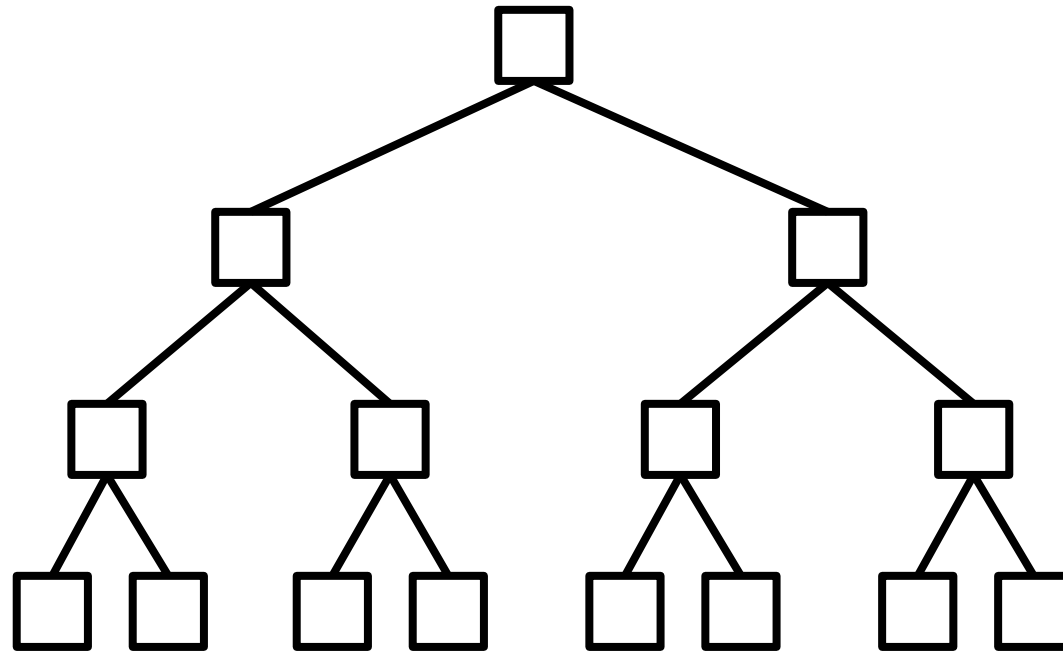
- A BST is composed of Comparable data elements.
- A BST is a binary tree (each node has at most two children).
- For each node, all left-hand descendants have values that are less than the node.
- For each node, all right-hand descendants have values that are larger than the node.
- There are no duplicate values (definitions vary).



Is it a BST?

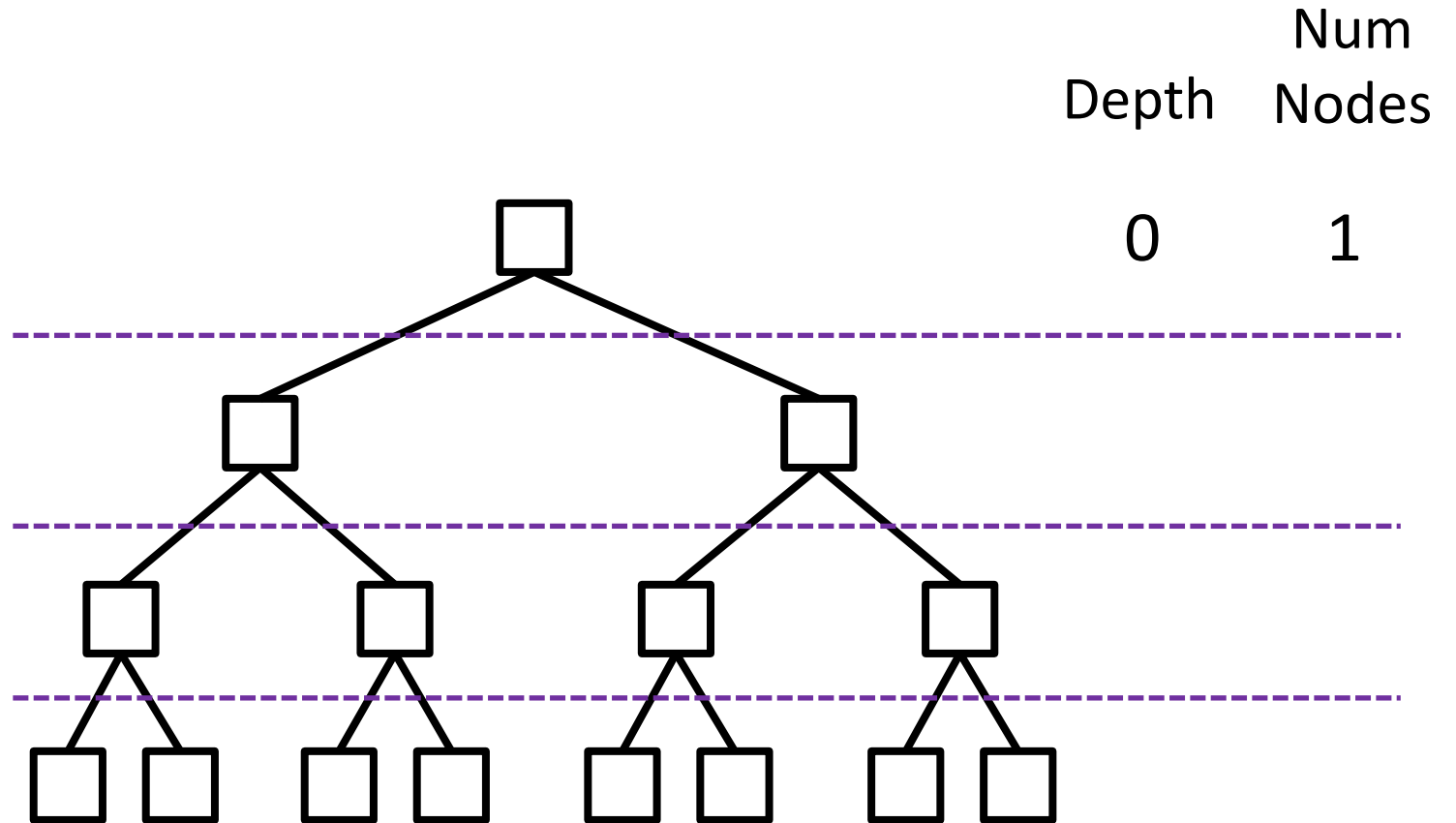
Binary Search Tree

What is the point? Why use a BST?



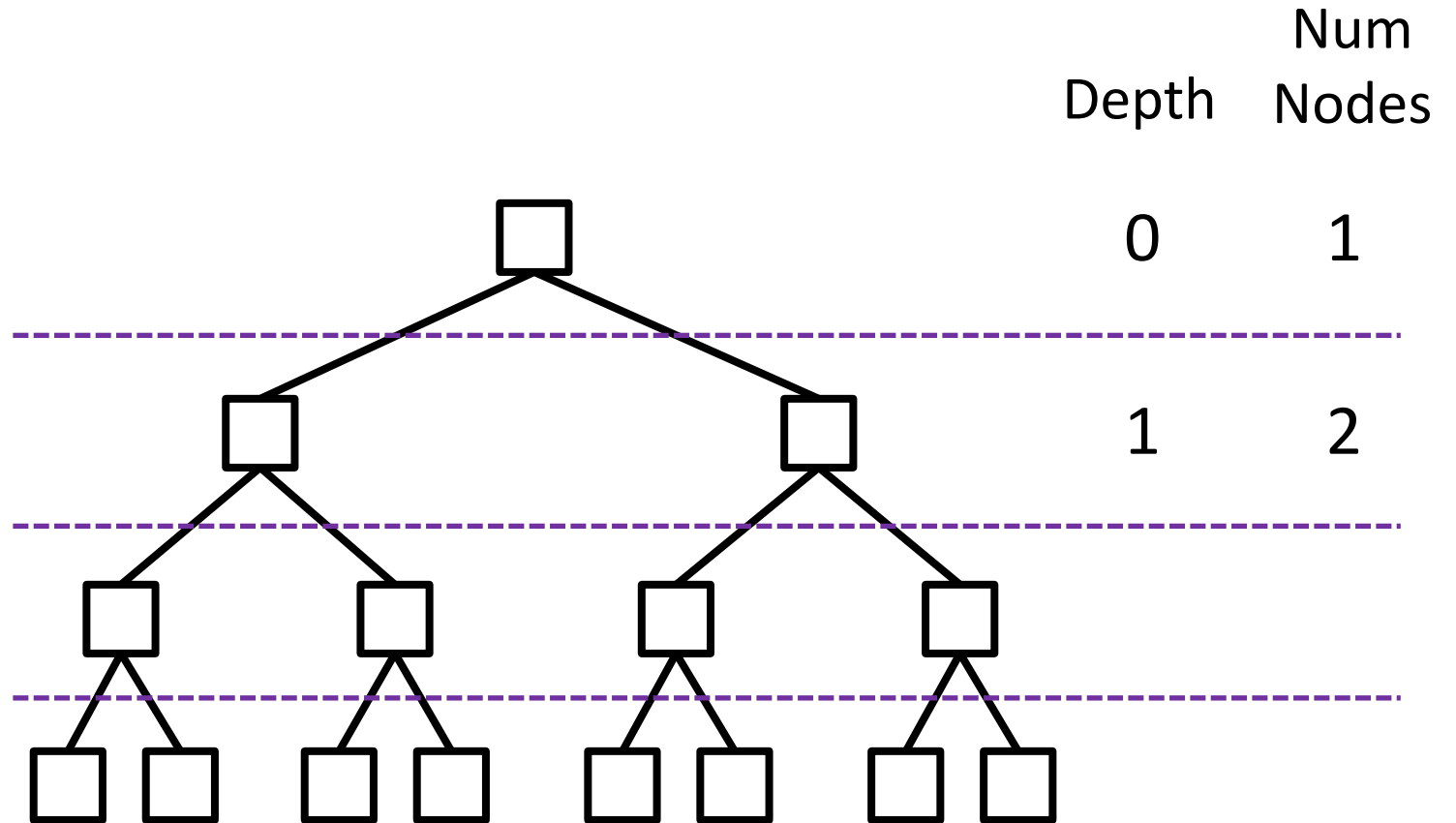
Binary Search Tree

What is the point? Why use a BST?



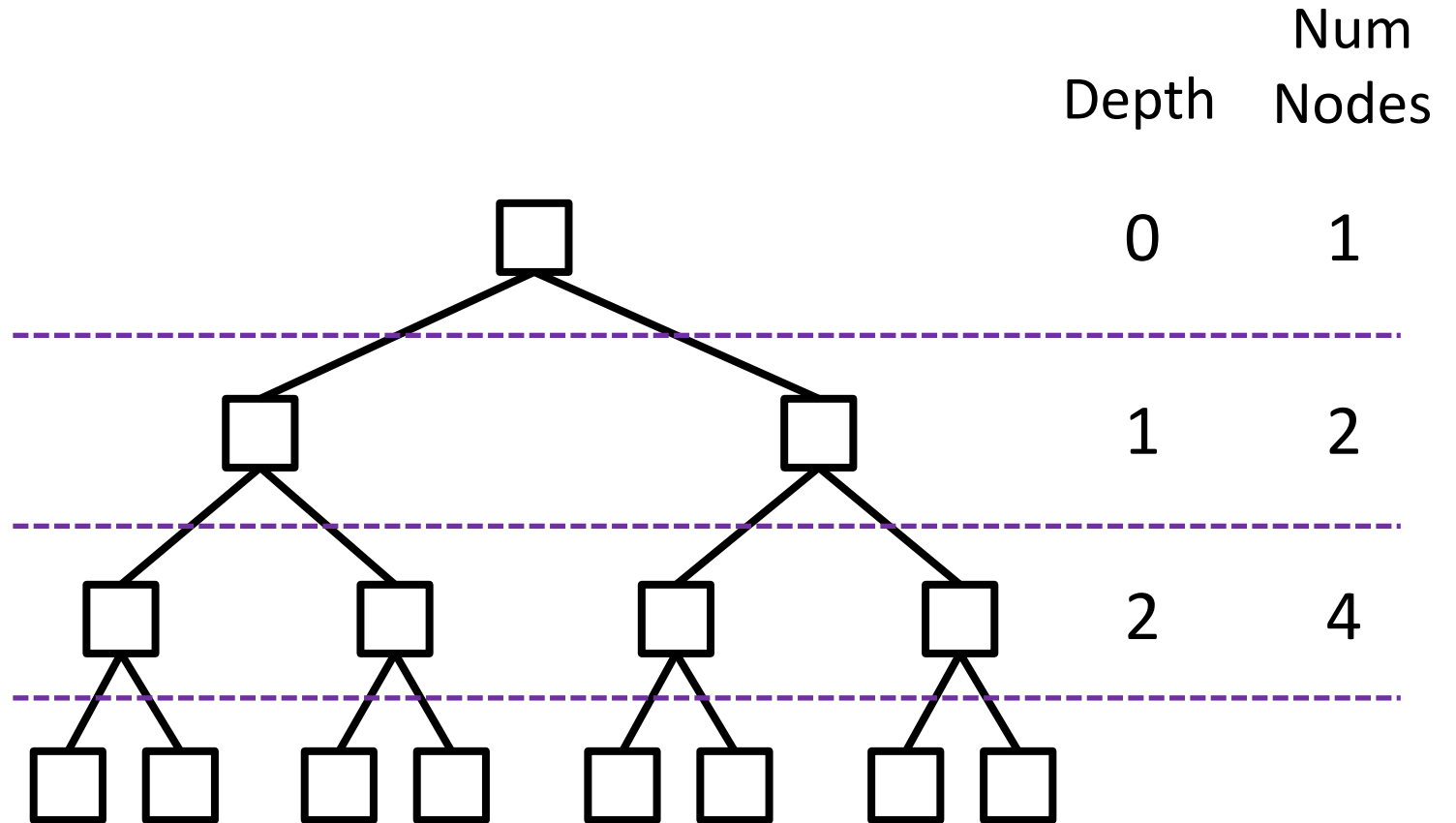
Binary Search Tree

What is the point? Why use a BST?



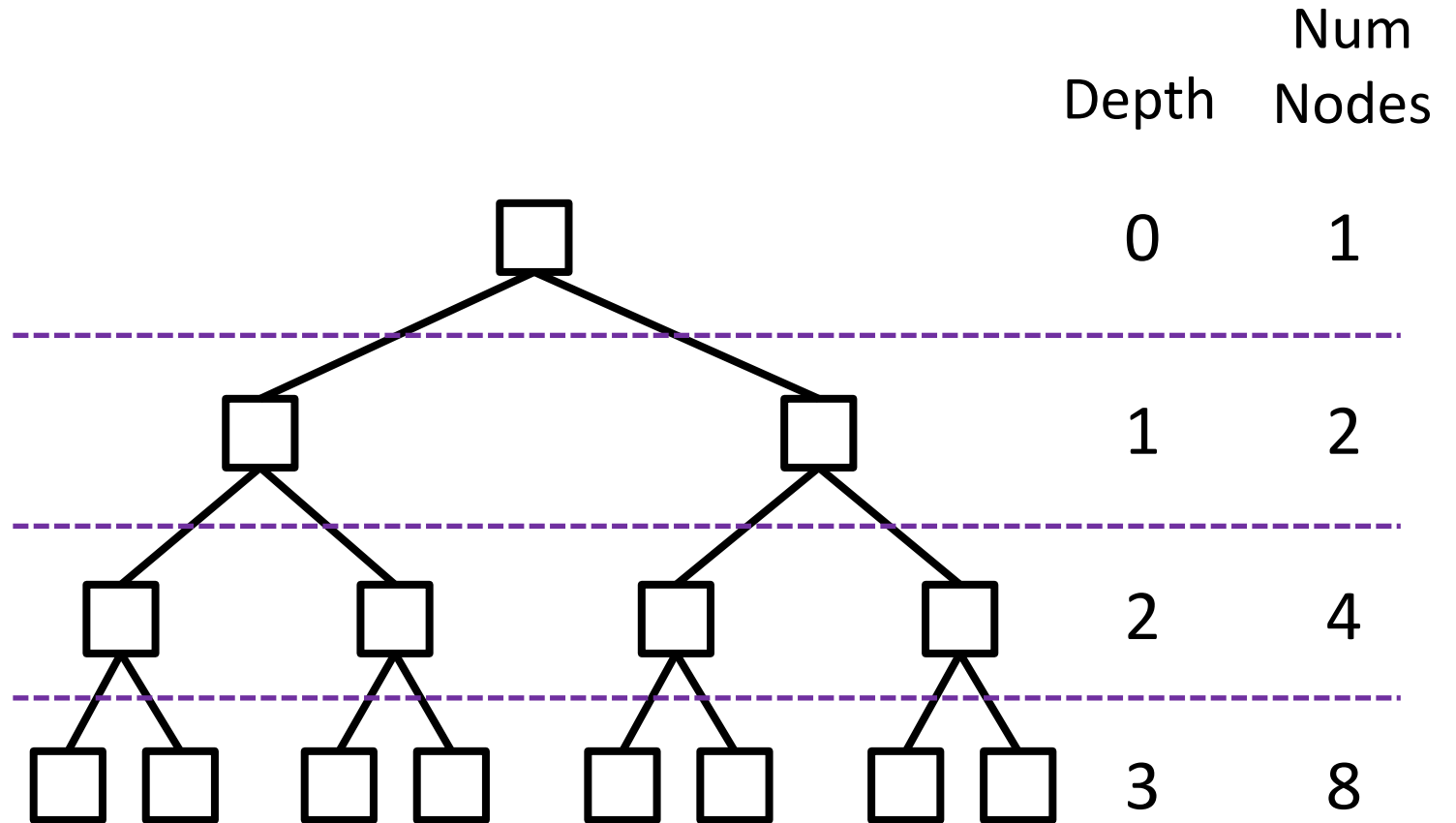
Binary Search Tree

What is the point? Why use a BST?



Binary Search Tree

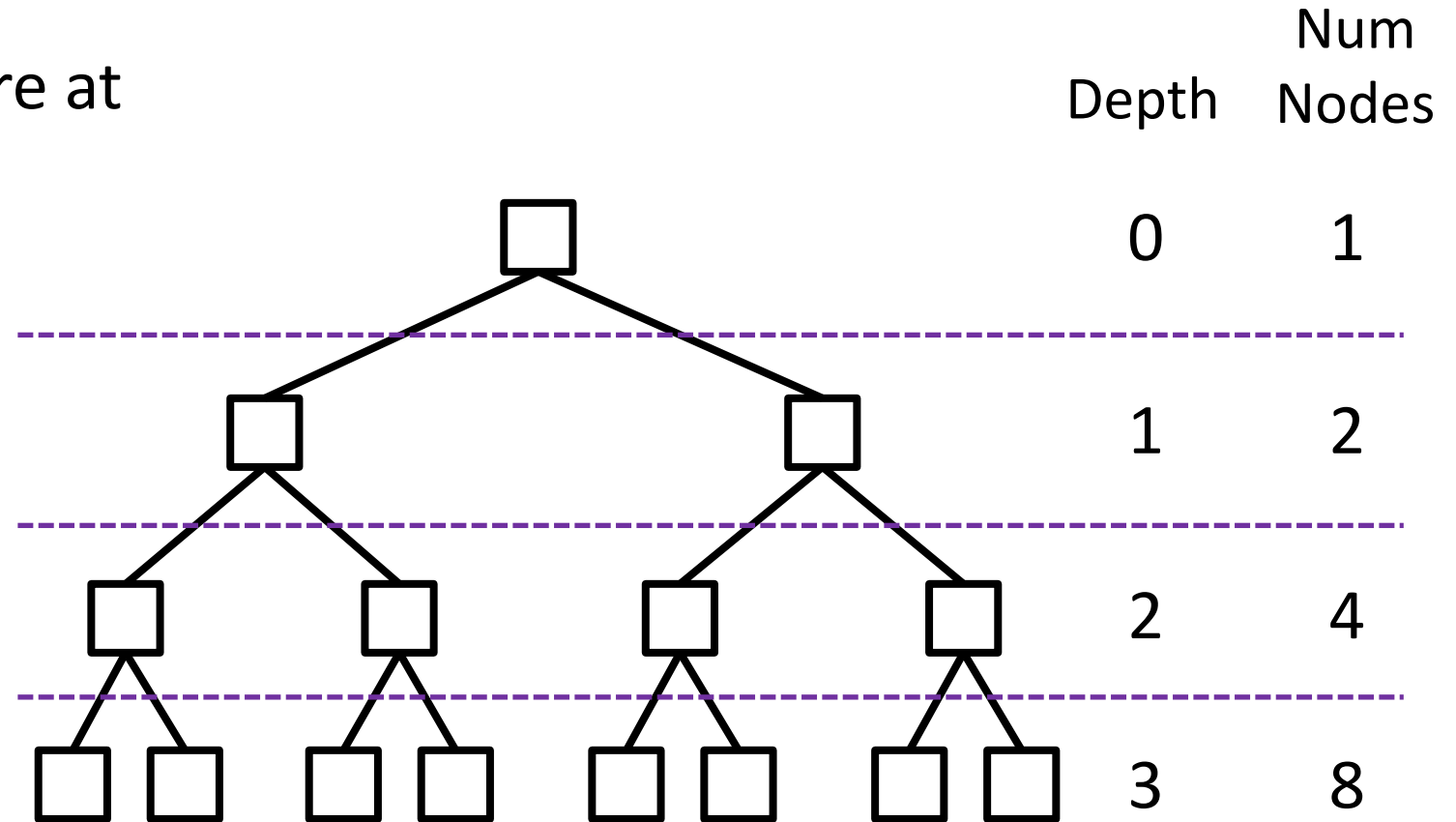
What is the point? Why use a BST?



Binary Search Tree

What is the point? Why use a BST?

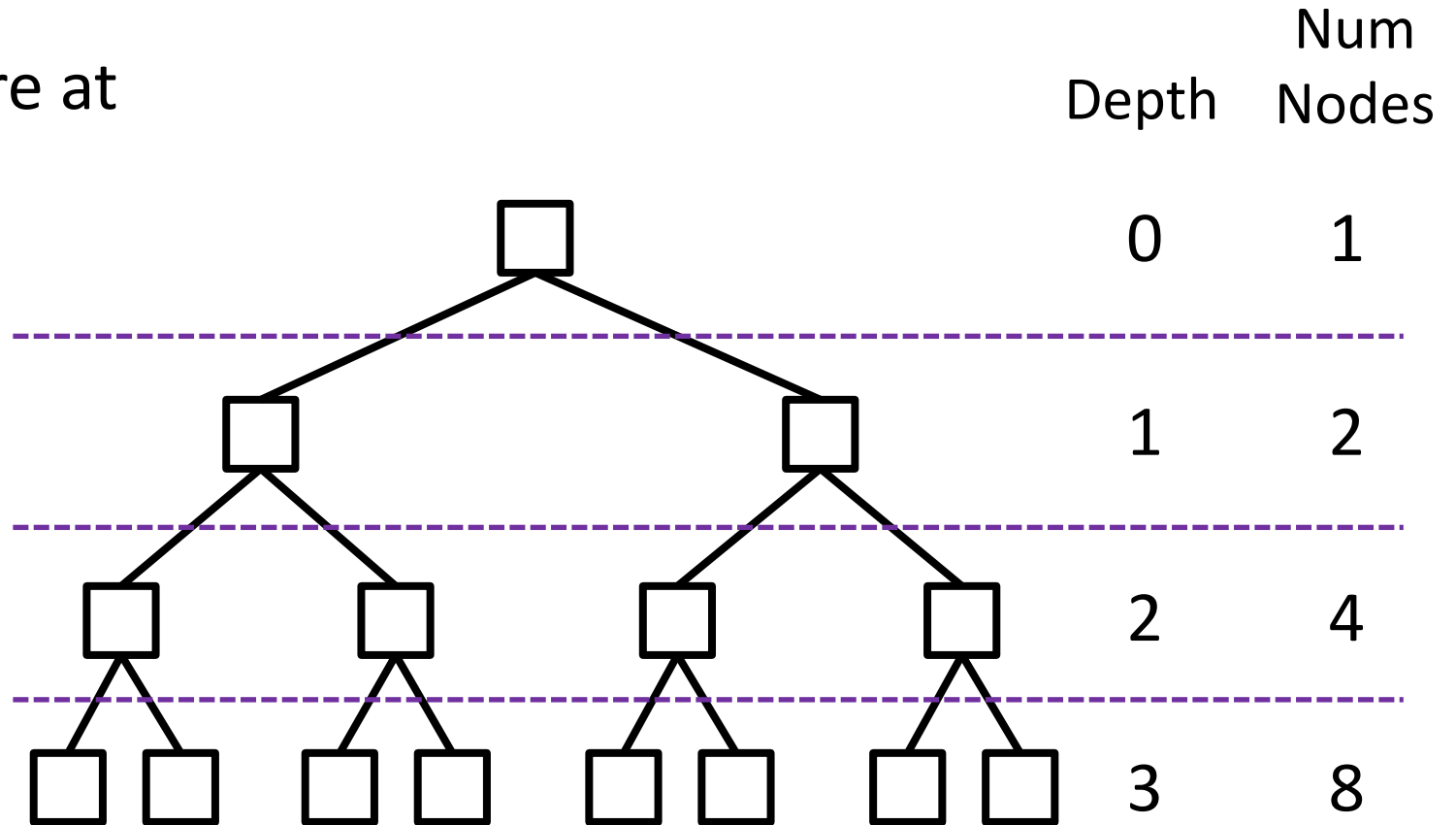
In general, at depth d , there are at most ?? nodes.



Binary Search Tree

What is the point? Why use a BST?

In general, at depth d , there are at most 2^d nodes.

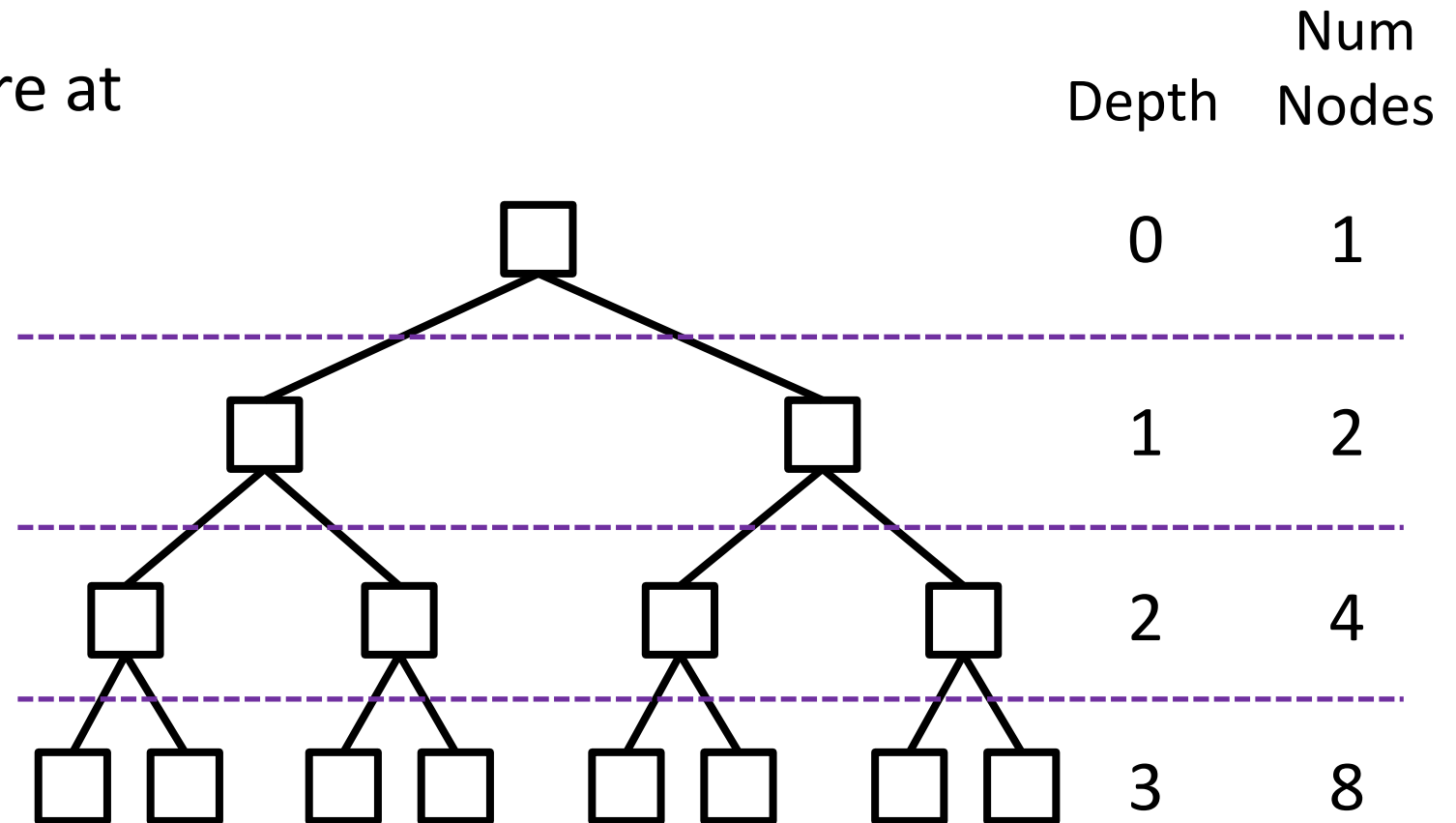


Binary Search Tree

What is the point? Why use a BST?

In general, at depth d , there are at most 2^d nodes.

Given a BST with n nodes, what is the greatest number of edges we would have to traverse to go from the root to a leaf?

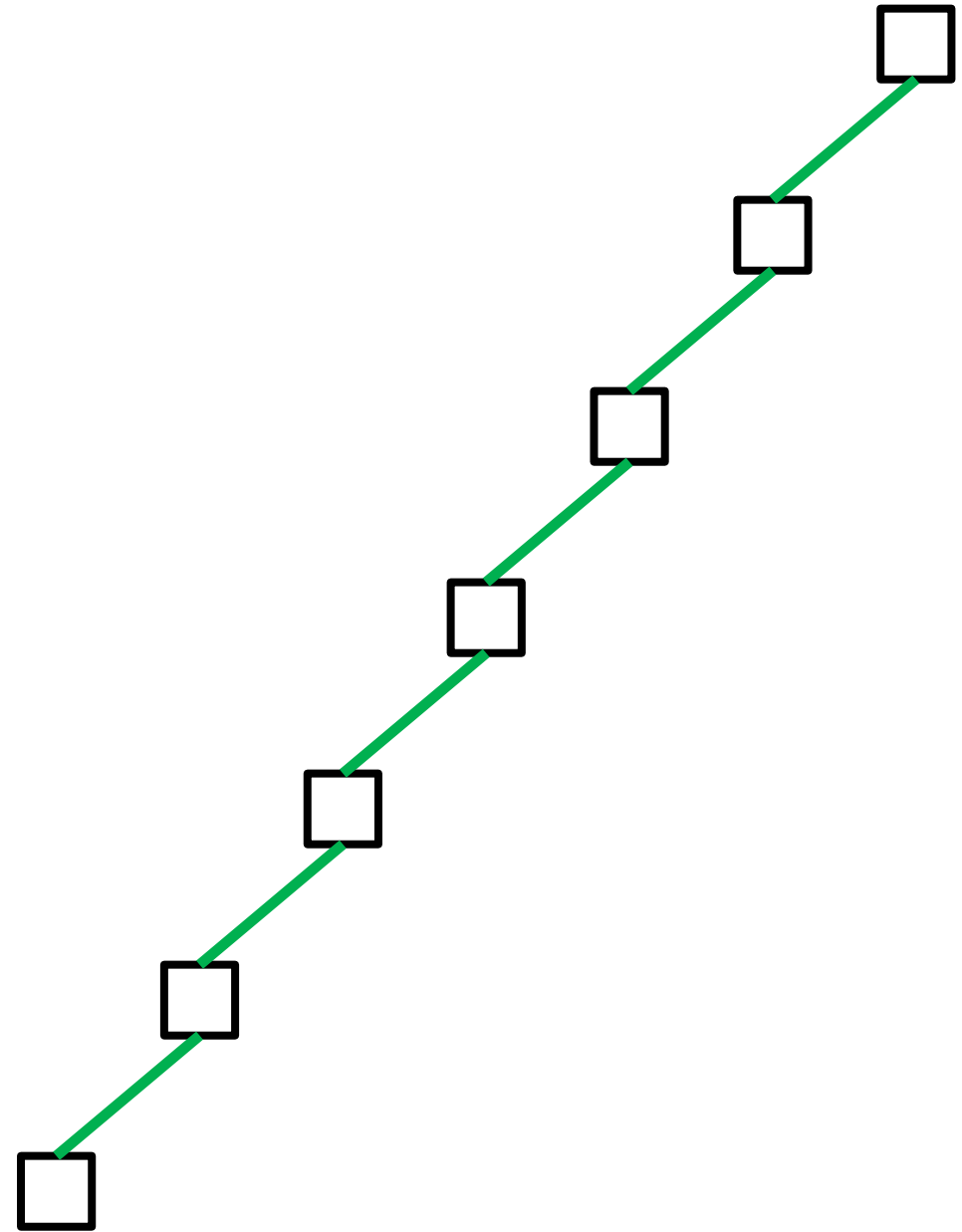


Binary Search Tree

What is the point? Why use a BST?

In general, at depth d , there are at most 2^d nodes.

Given a BST with n nodes, what is the greatest number of edges we would have to traverse to go from the root to a leaf? $n - 1$

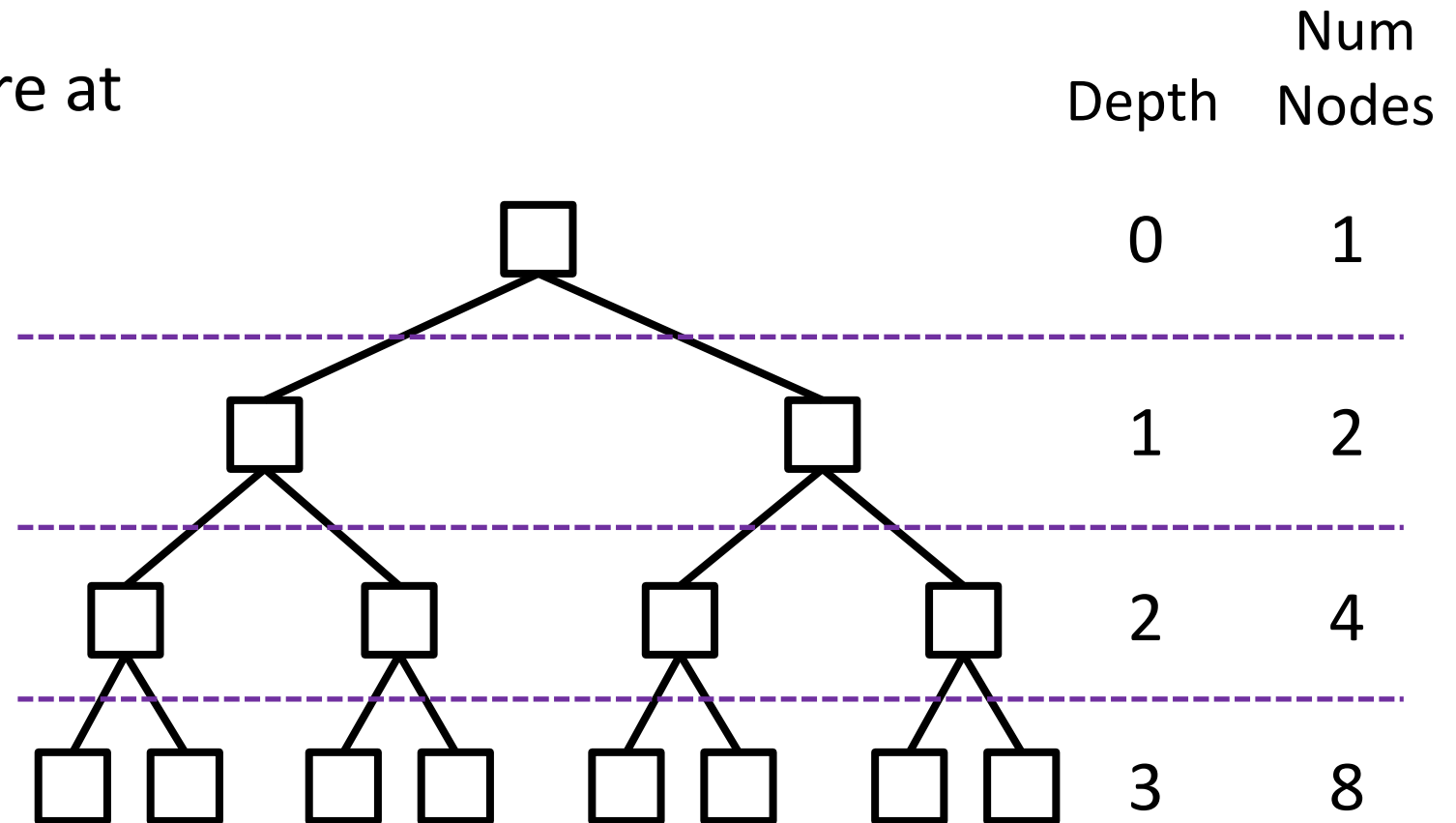


Binary Search Tree

What is the point? Why use a BST?

In general, at depth d , there are at most 2^d nodes.

Given a BST with n nodes, what is the greatest number of edges we would have to traverse to go from the root to a leaf?

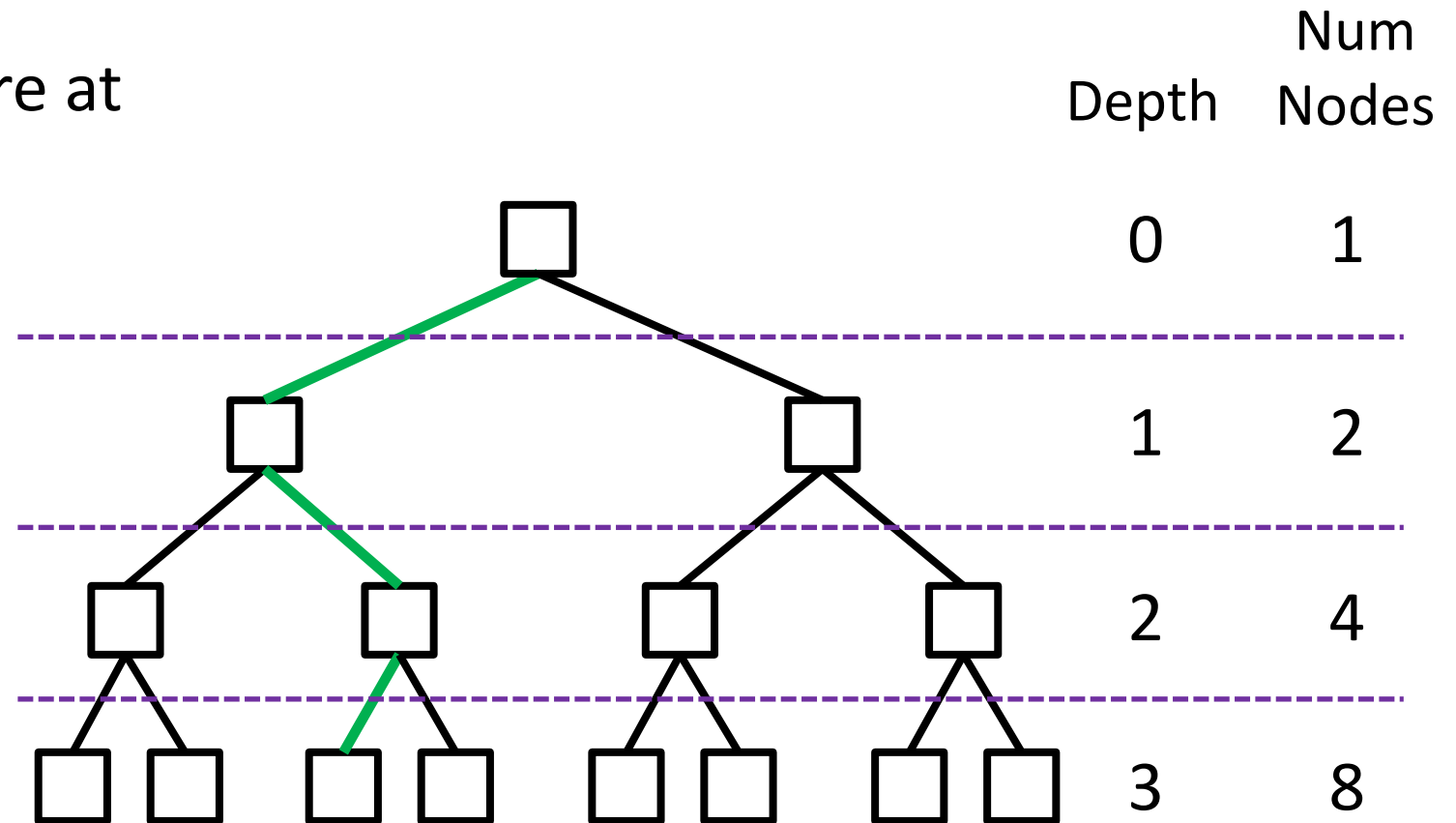


Binary Search Tree

What is the point? Why use a BST?

In general, at depth d , there are at most 2^d nodes.

Given a BST with n nodes, what is the greatest number of edges we would have to traverse to go from the root to a leaf? *height of tree*.



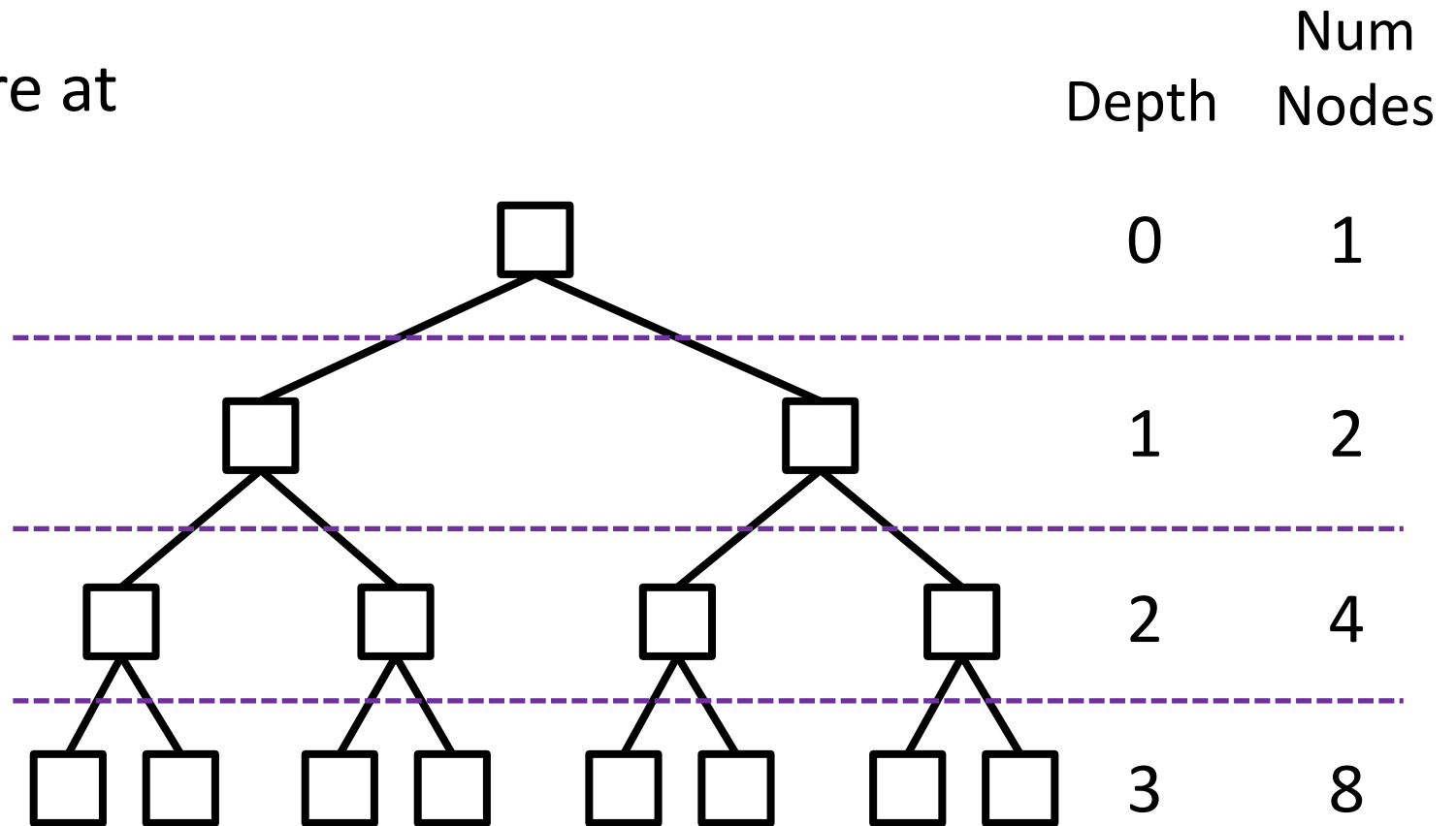
Binary Search Tree

What is the point? Why use a BST?

In general, at depth d , there are at most 2^d nodes.

Given a BST with n nodes, what is the greatest number of edges we would have to traverse to go from the root to a leaf? *height of tree*.

Given n nodes, what is the smallest height (h) of the BST?



Binary Search Tree

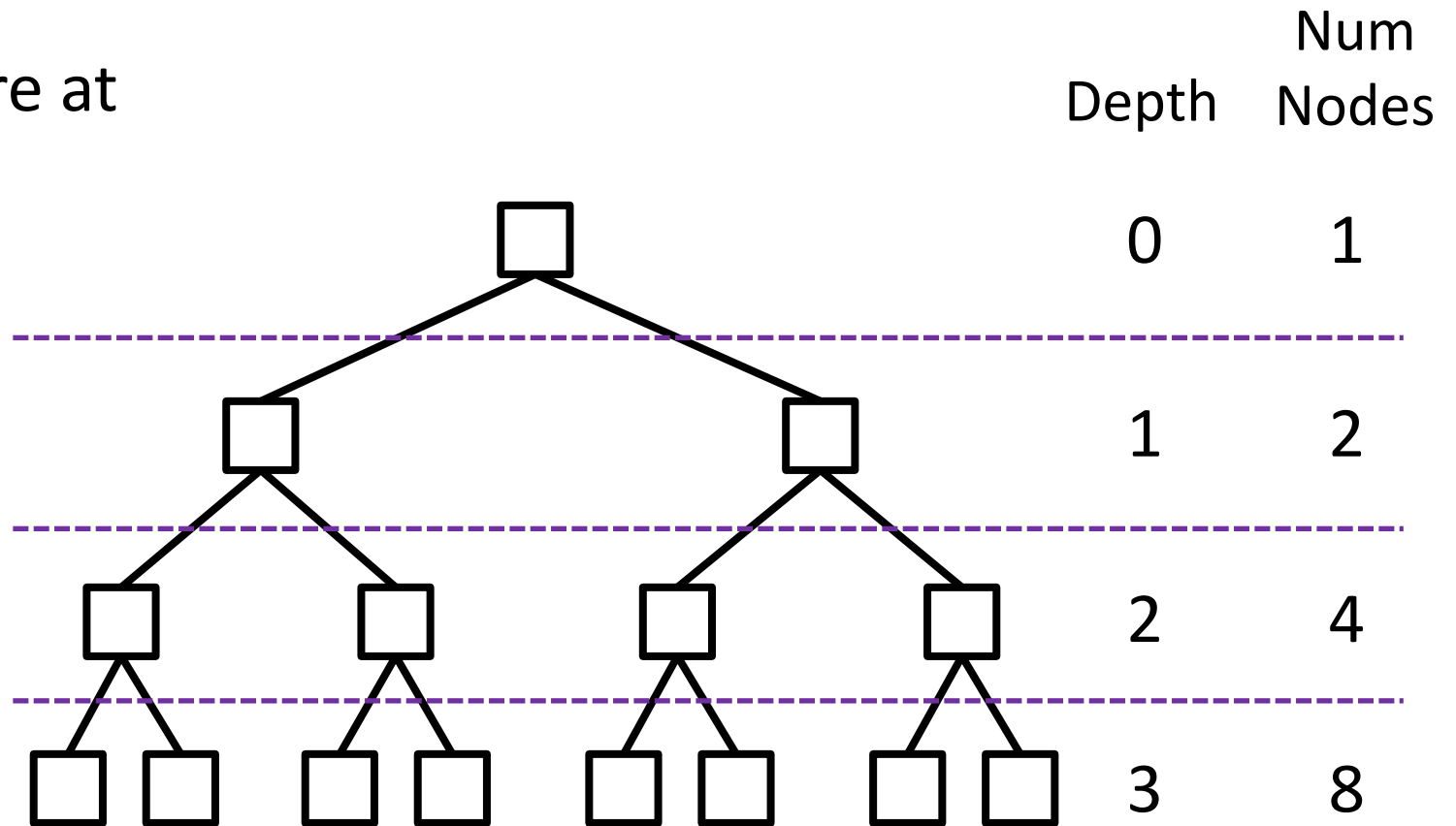
What is the point? Why use a BST?

In general, at depth d , there are at most 2^d nodes.

Given a BST with n nodes, what is the greatest number of edges we would have to traverse to go from the root to a leaf? *height of tree*.

Given n nodes, what is the smallest height (h) of the BST?

$$n = 2^0 + 2^1 + 2^2 + \dots + 2^h$$



Binary Search Tree

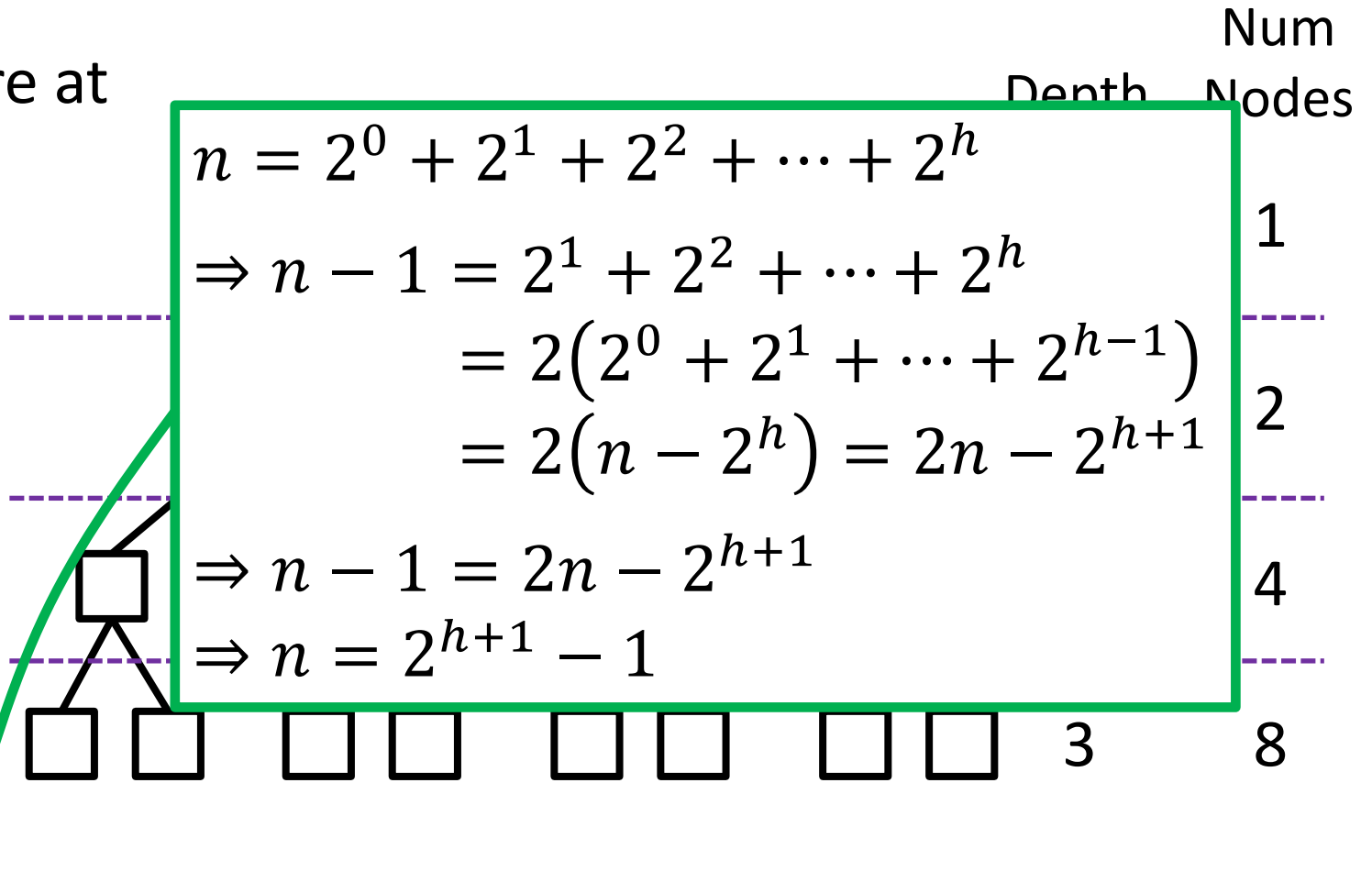
What is the point? Why use a BST?

In general, at depth d , there are at most 2^d nodes.

Given a BST with n nodes, what is the greatest number of edges we would have to traverse to go from the root to a leaf? *height of tree*.

Given n nodes, what is the smallest height (h) of the BST?

$$n = 2^0 + 2^1 + 2^2 + \dots + 2^h = 2^{h+1} - 1$$



Binary Search Tree

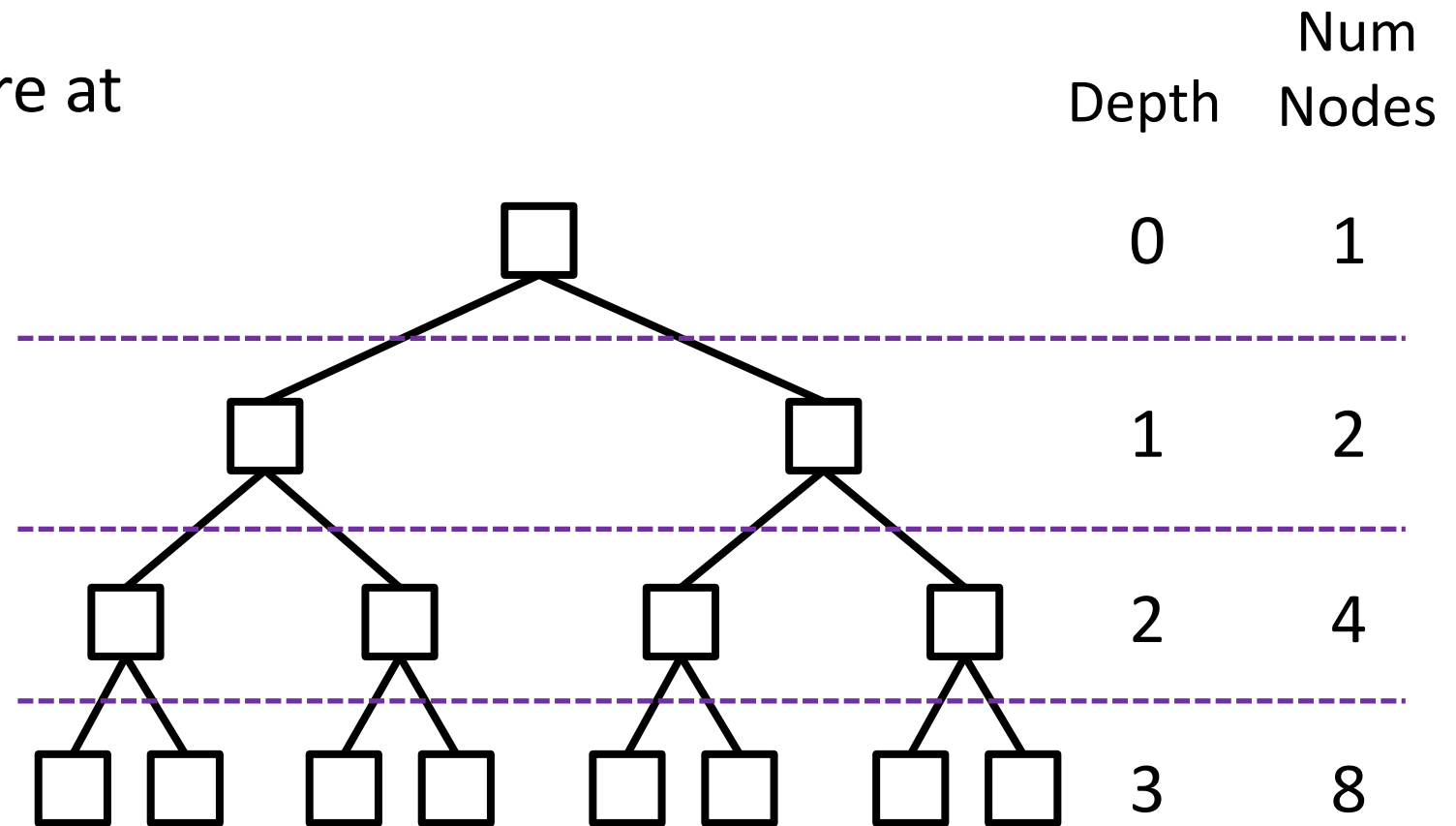
What is the point? Why use a BST?

In general, at depth d , there are at most 2^d nodes.

Given a BST with n nodes, what is the greatest number of edges we would have to traverse to go from the root to a leaf? *height of tree*.

Given n nodes, what is the smallest height (h) of the BST?

$$n = 2^0 + 2^1 + 2^2 + \dots + 2^h = 2^{h+1} - 1 \Rightarrow n + 1 = 2^{h+1}$$



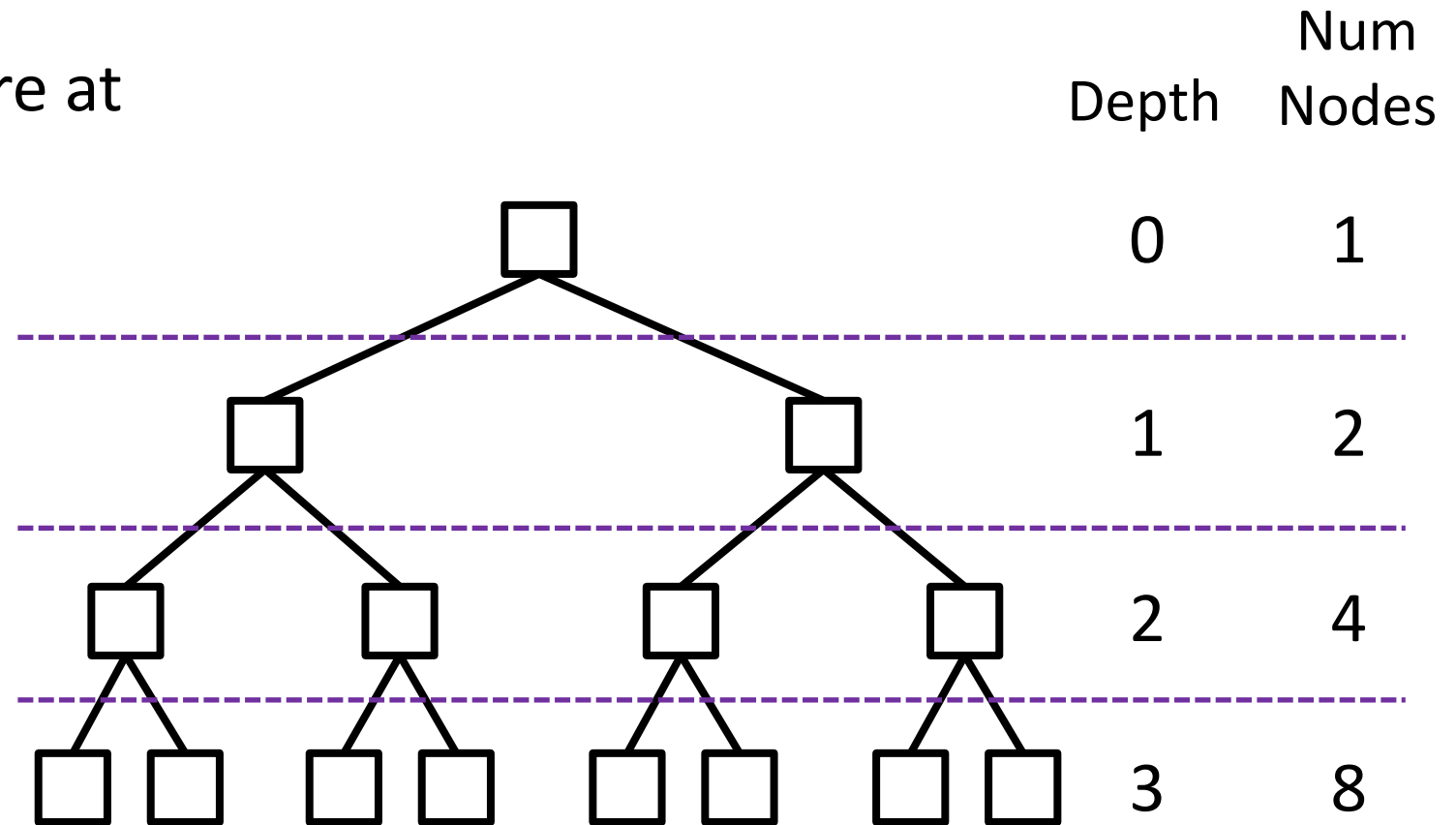
Binary Search Tree

What is the point? Why use a BST?

In general, at depth d , there are at most 2^d nodes.

Given a BST with n nodes, what is the greatest number of edges we would have to traverse to go from the root to a leaf? *height of tree*.

Given n nodes, what is the smallest height (h) of the BST?



$$n = 2^0 + 2^1 + 2^2 + \dots + 2^h = 2^{h+1} - 1 \Rightarrow \begin{aligned} n + 1 &= 2^{h+1} \\ \log_2(n + 1) &= h + 1 \end{aligned}$$

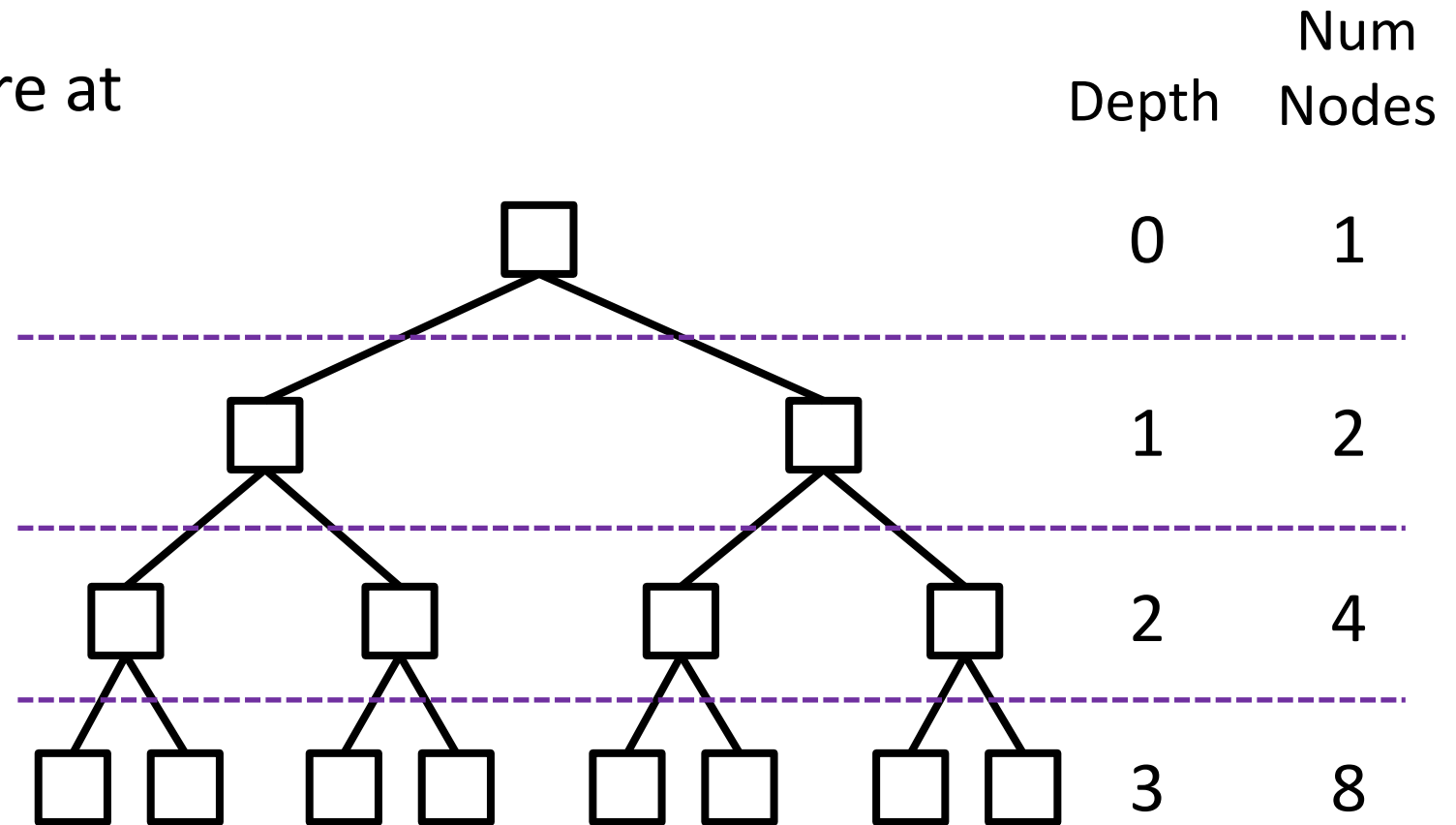
Binary Search Tree

What is the point? Why use a BST?

In general, at depth d , there are at most 2^d nodes.

Given a BST with n nodes, what is the greatest number of edges we would have to traverse to go from the root to a leaf? *height of tree*.

Given n nodes, what is the smallest height (h) of the BST?

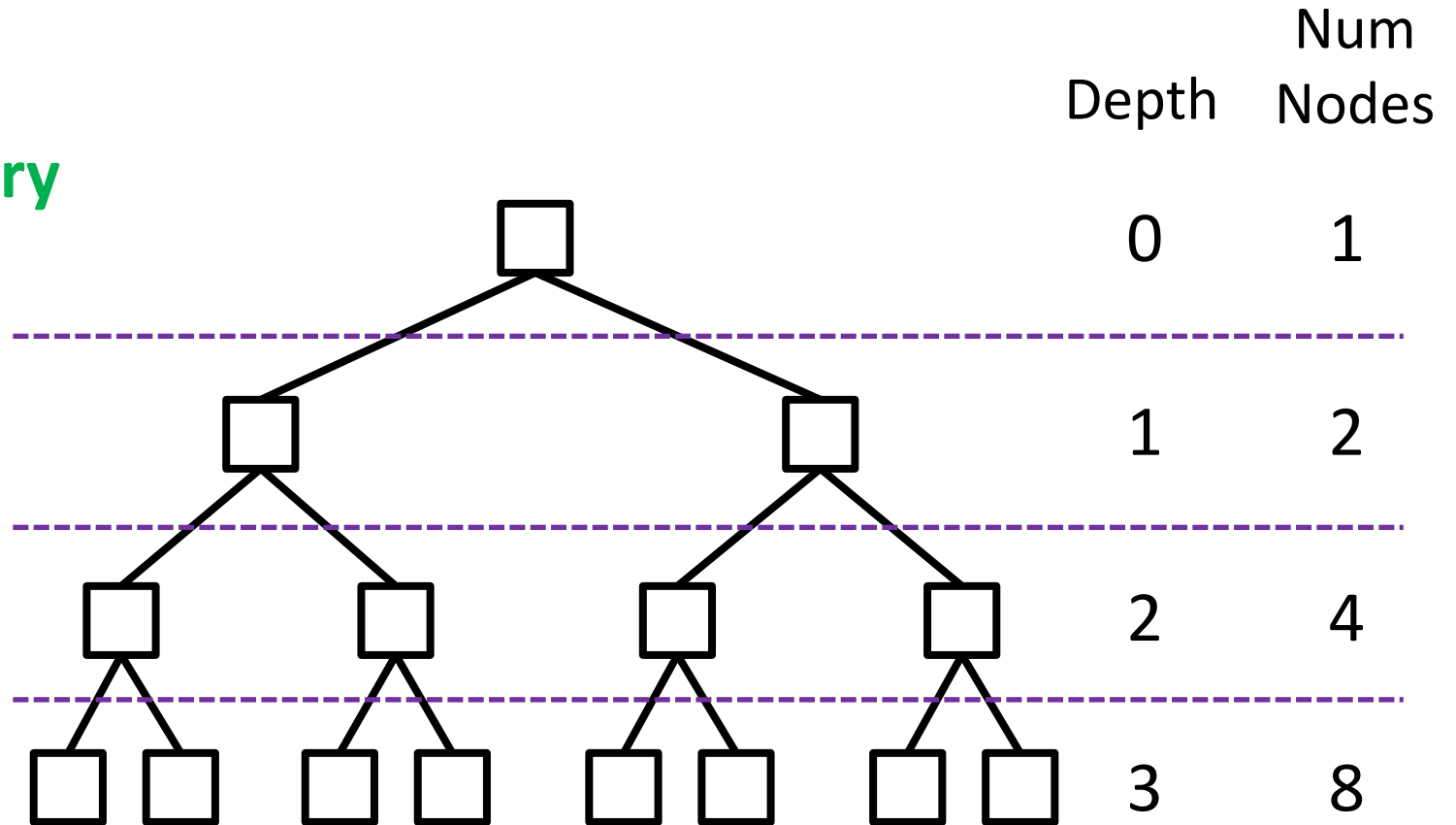


$$n = 2^0 + 2^1 + 2^2 + \dots + 2^h = 2^{h+1} - 1 \Rightarrow n + 1 = 2^{h+1} \\ \log_2(n + 1) = h + 1 \Rightarrow h \in O(\log n)$$

Binary Search Tree

What is the point? Why use a BST?

This means we can access any node in a specific type of binary tree in $\log n$ time.

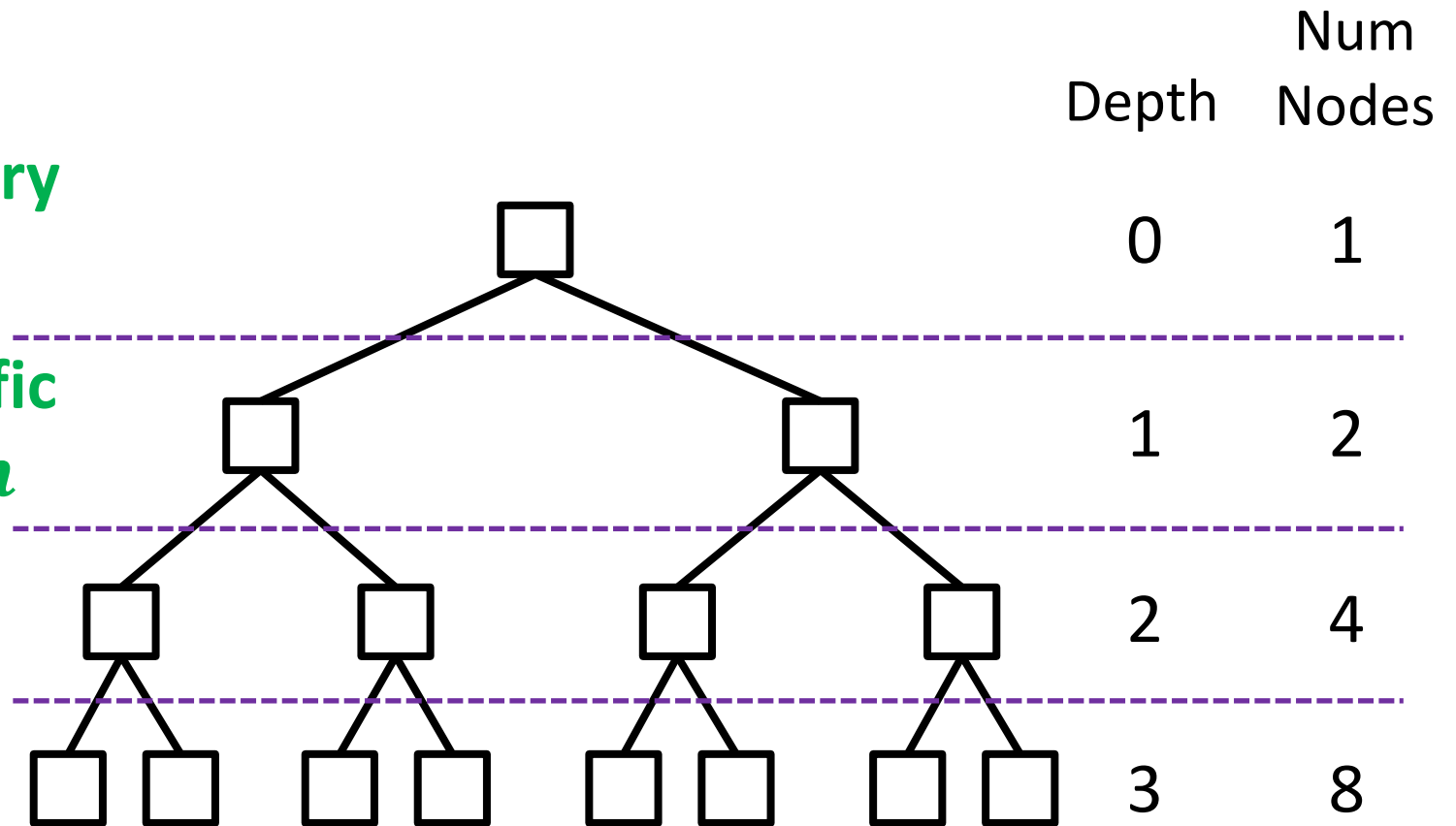


Binary Search Tree

What is the point? Why use a BST?

This means we can access any node in a specific type of binary tree in $\log n$ time.

Of note, we can test if a specific value is in a collection in $\log n$ time.



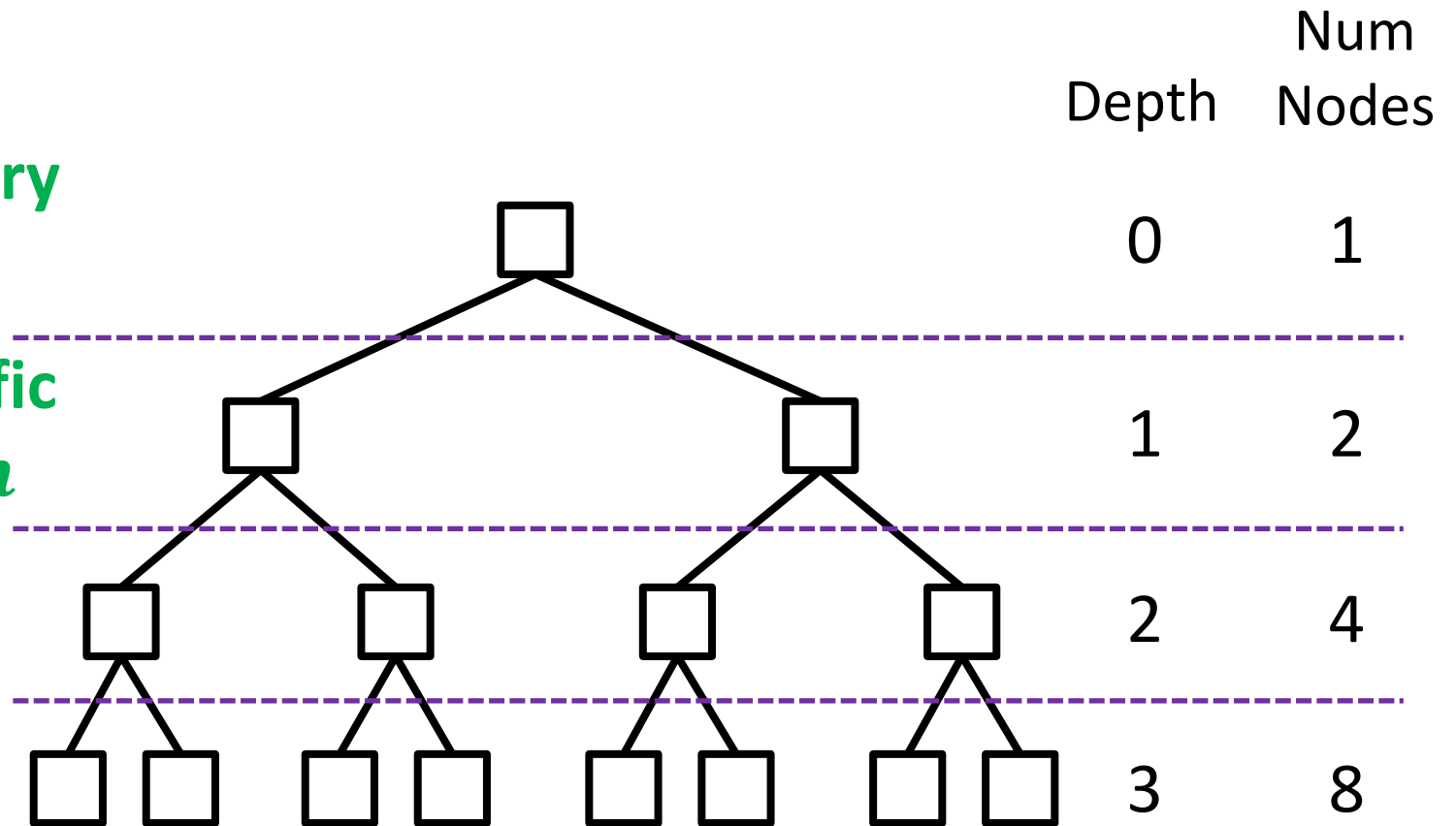
Binary Search Tree

What is the point? Why use a BST?

This means we can access any node in a specific type of binary tree in $\log n$ time.

Of note, we can test if a specific value is in a collection in $\log n$ time.

But we can already do that with a sorted array and Binary Search!



Binary Search Tree

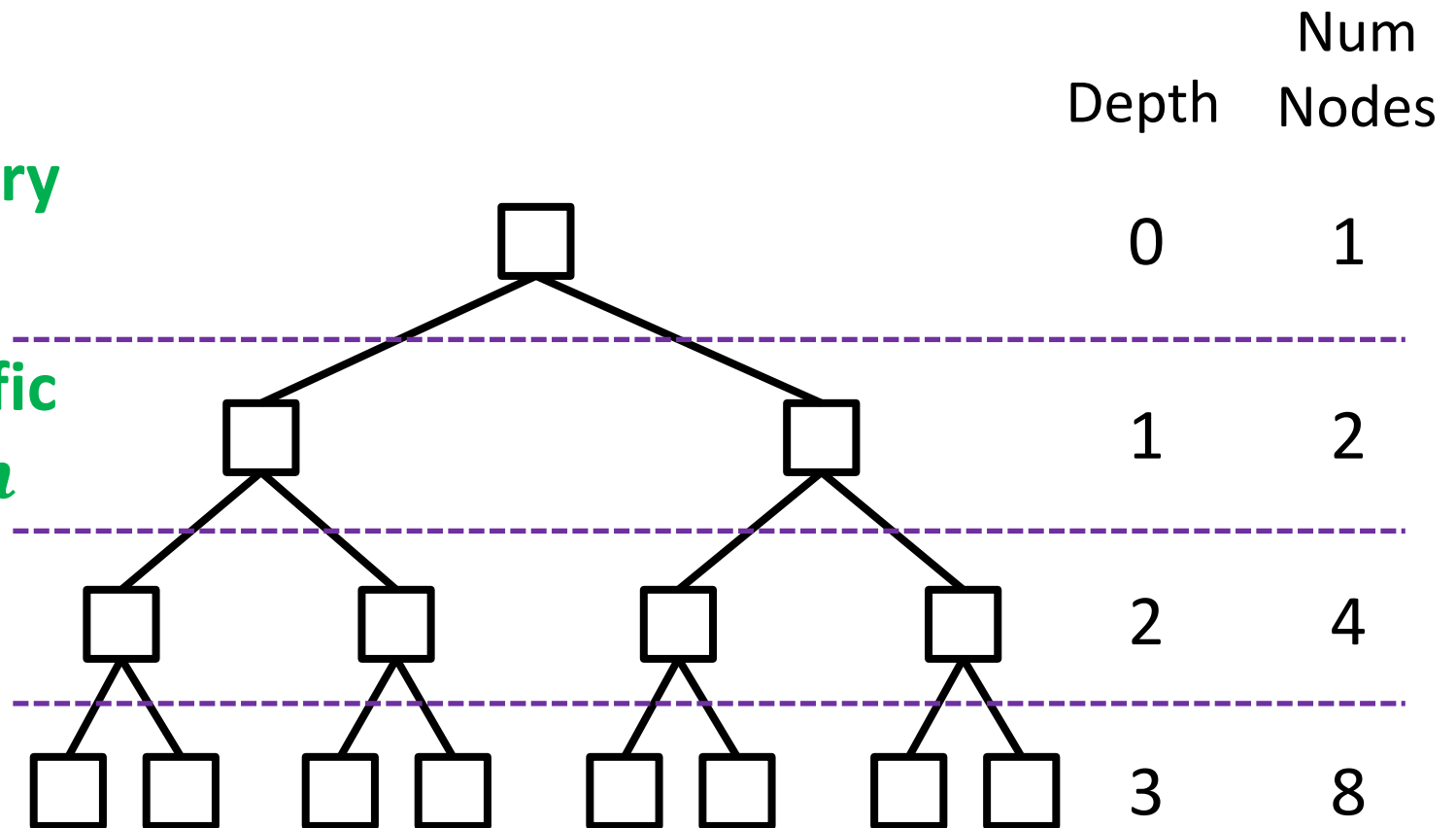
What is the point? Why use a BST?

This means we can access any node in a specific type of binary tree in $\log n$ time.

Of note, we can test if a specific value is in a collection in $\log n$ time.

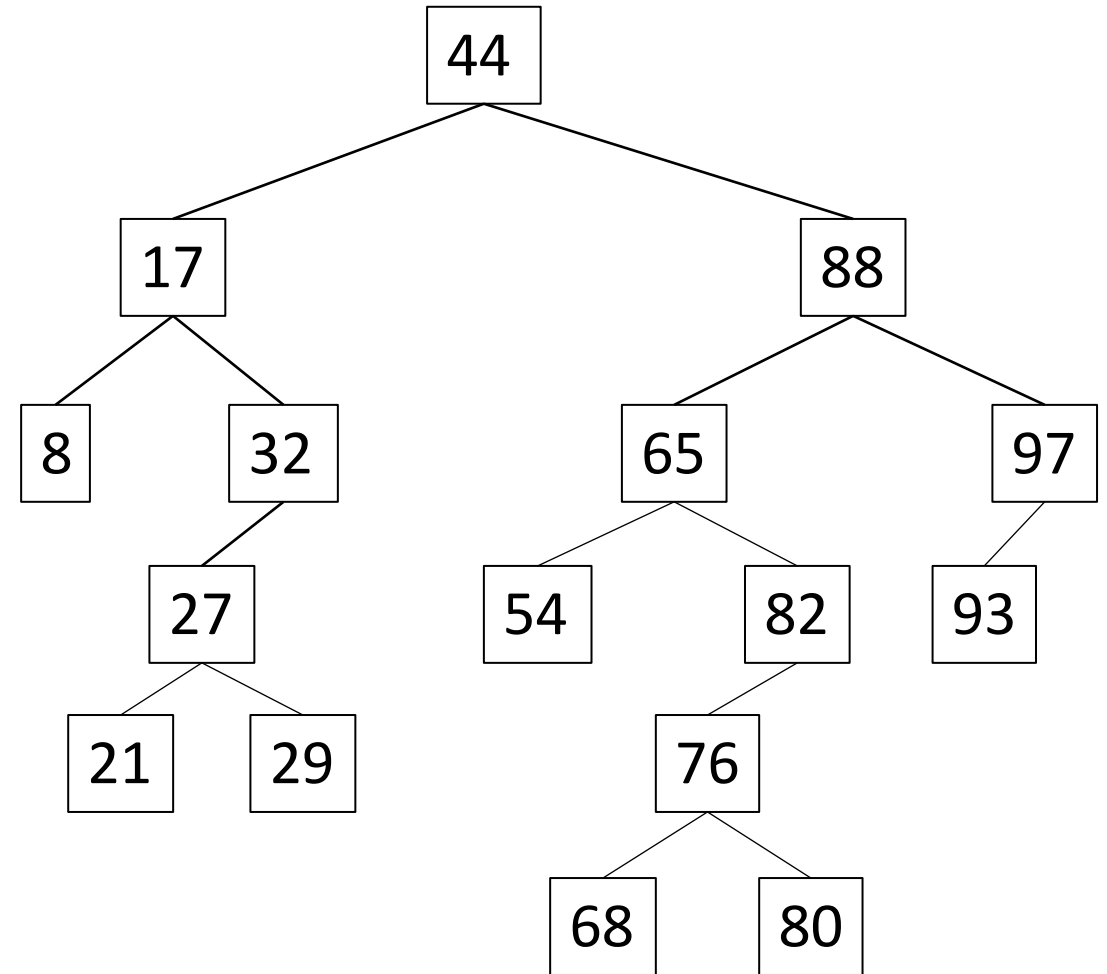
But we can already do that with a sorted array and Binary Search!

Perhaps managing a BST is more efficient than managing an array.



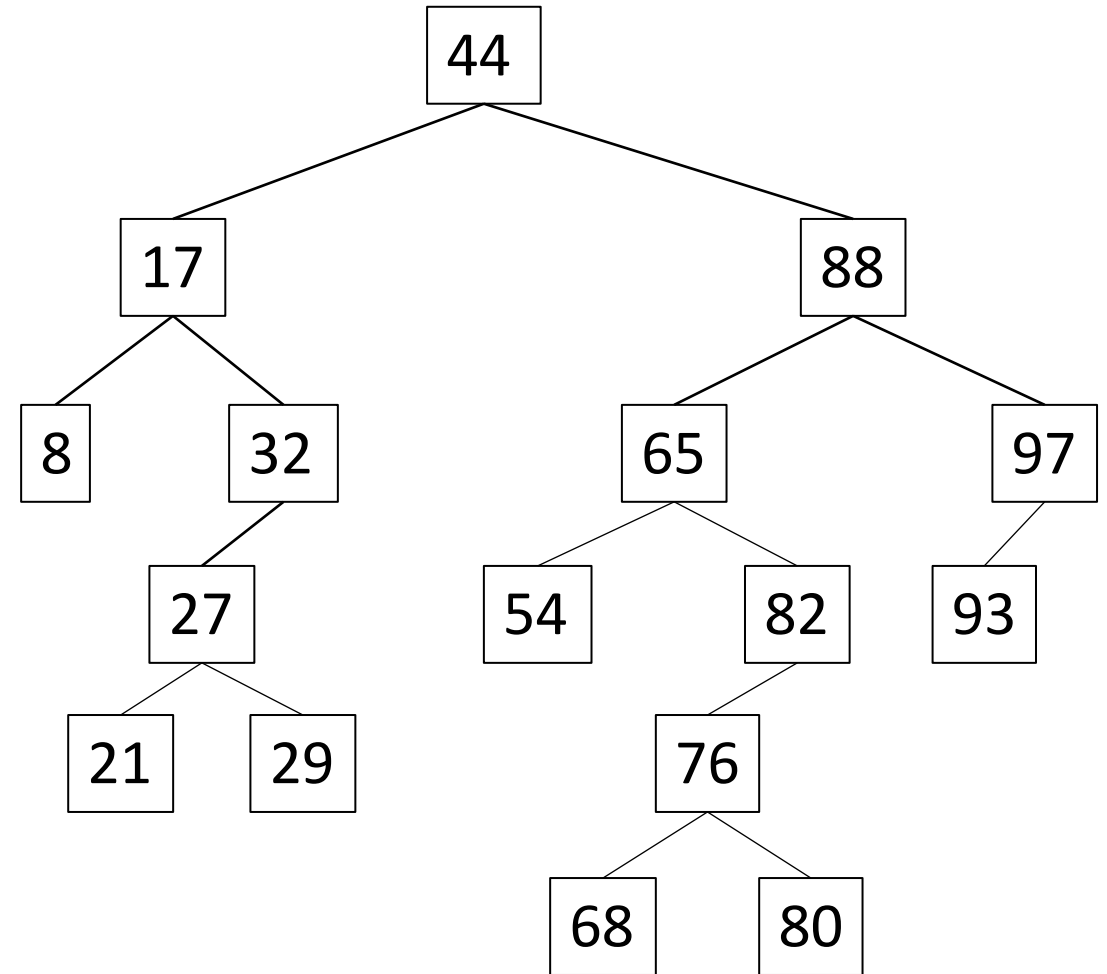
Binary Search Tree - Insertion

```
public class Node {  
  
    private int value;  
    private Node left;  
    private Node right;  
    private Node parent;  
  
    public Node(int value) {  
        this.value = value;  
    }  
  
    // getValue()  
    // getLeft(), getRight()  
    // getParent()  
  
    // setLeft(), setRight()  
    // setParent()  
}
```



Binary Search Tree - Insertion

`insert(31);`

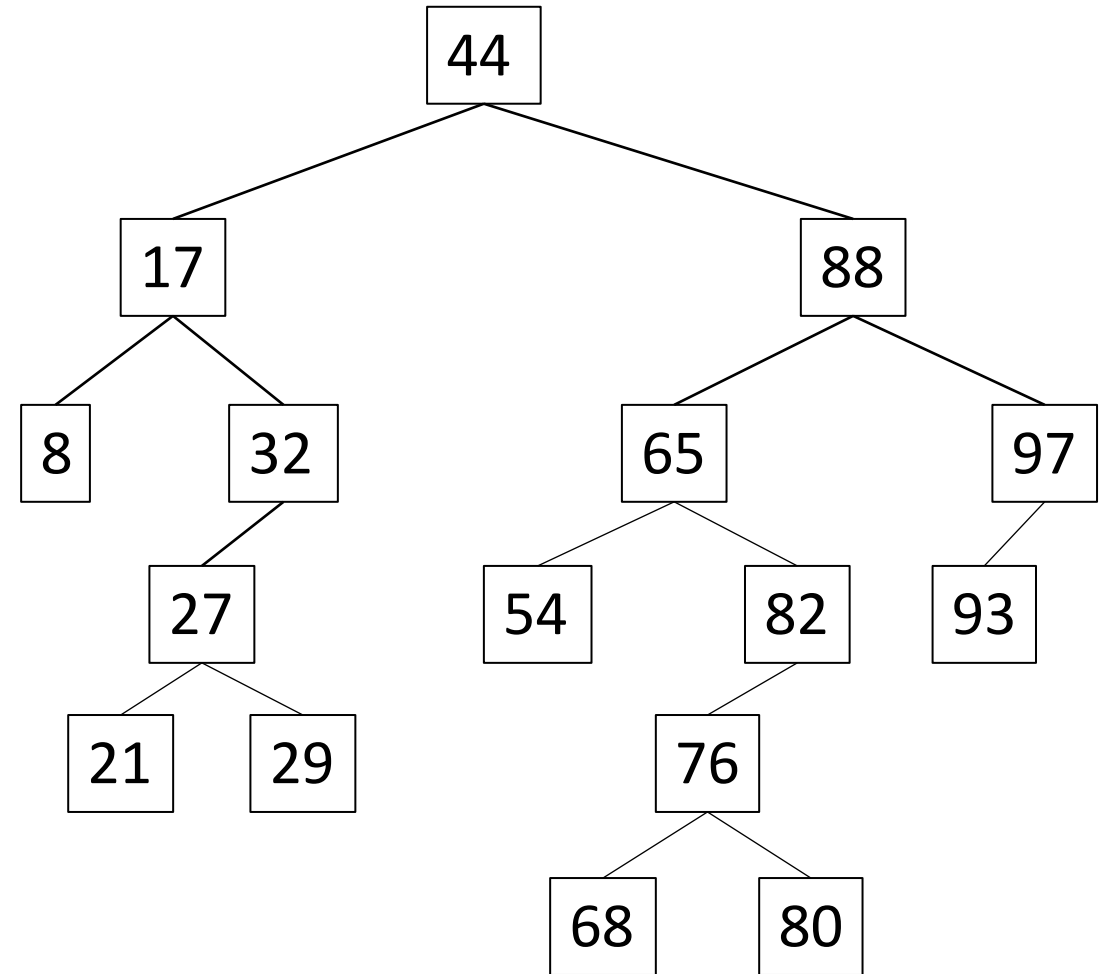


Binary Search Tree - Insertion

`insert(31);`

Step 1: Find where it should go.

Step 2: Modify pointers.

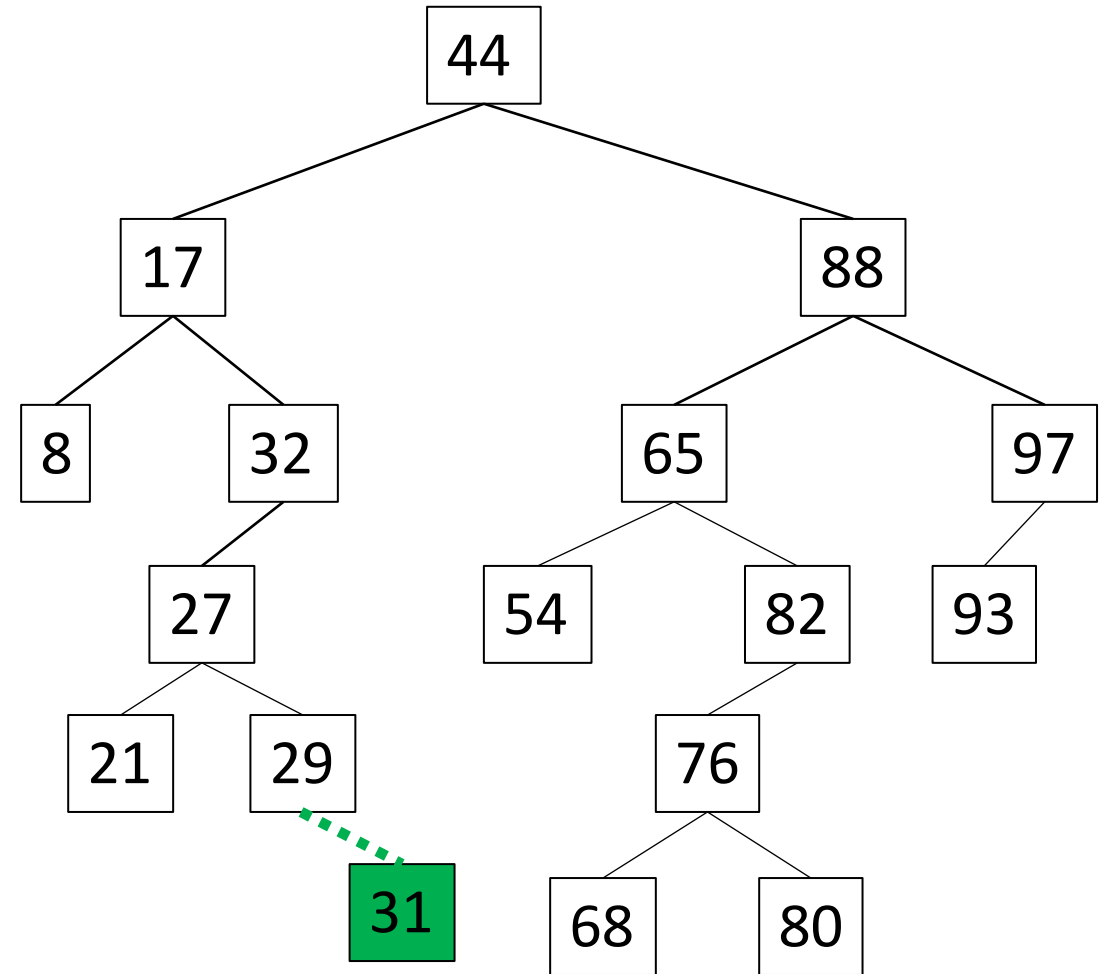


Binary Search Tree - Insertion

`insert(31);`

Step 1: Find where it should go.

Step 2: Modify pointers.

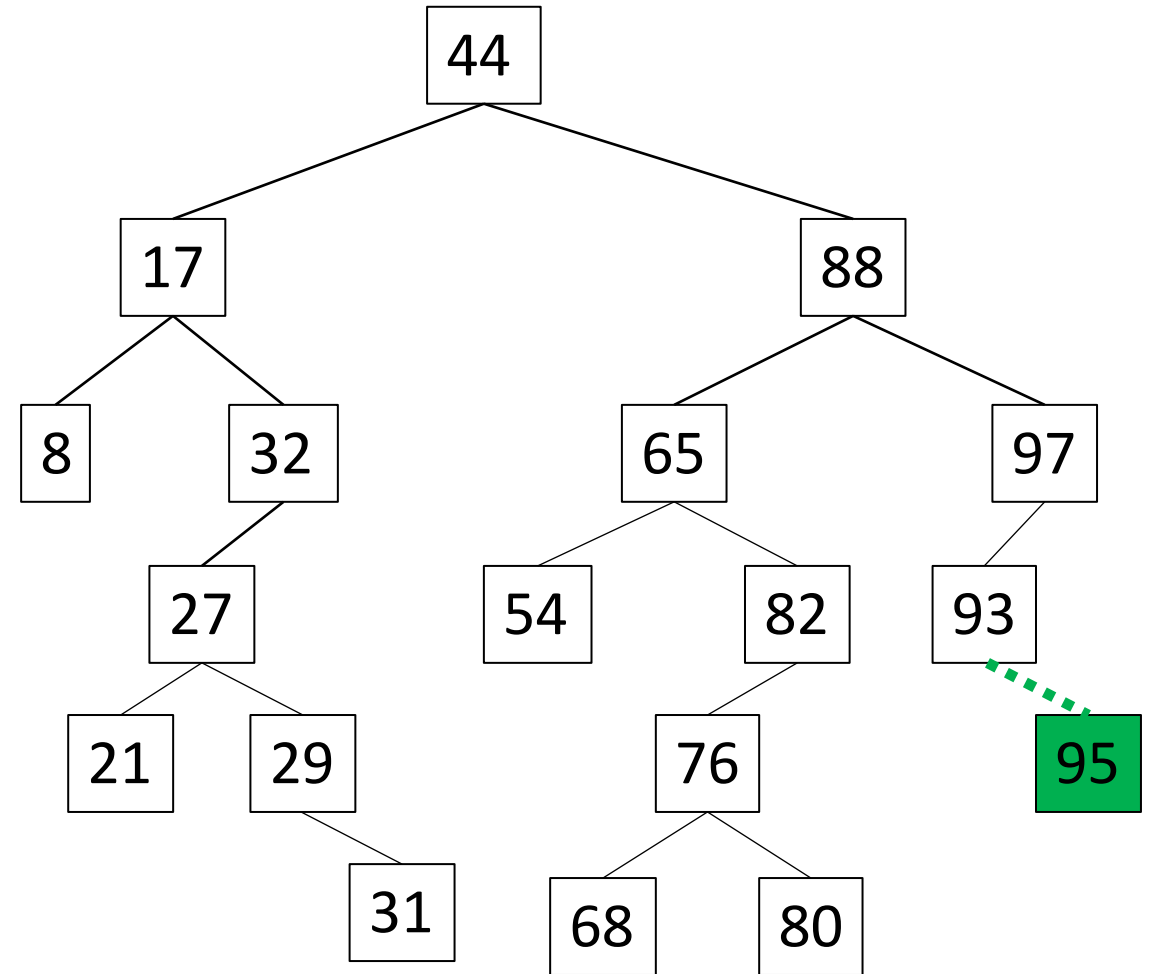


Binary Search Tree - Insertion

`insert(95);`

Step 1: Find where it should go.

Step 2: Modify pointers.



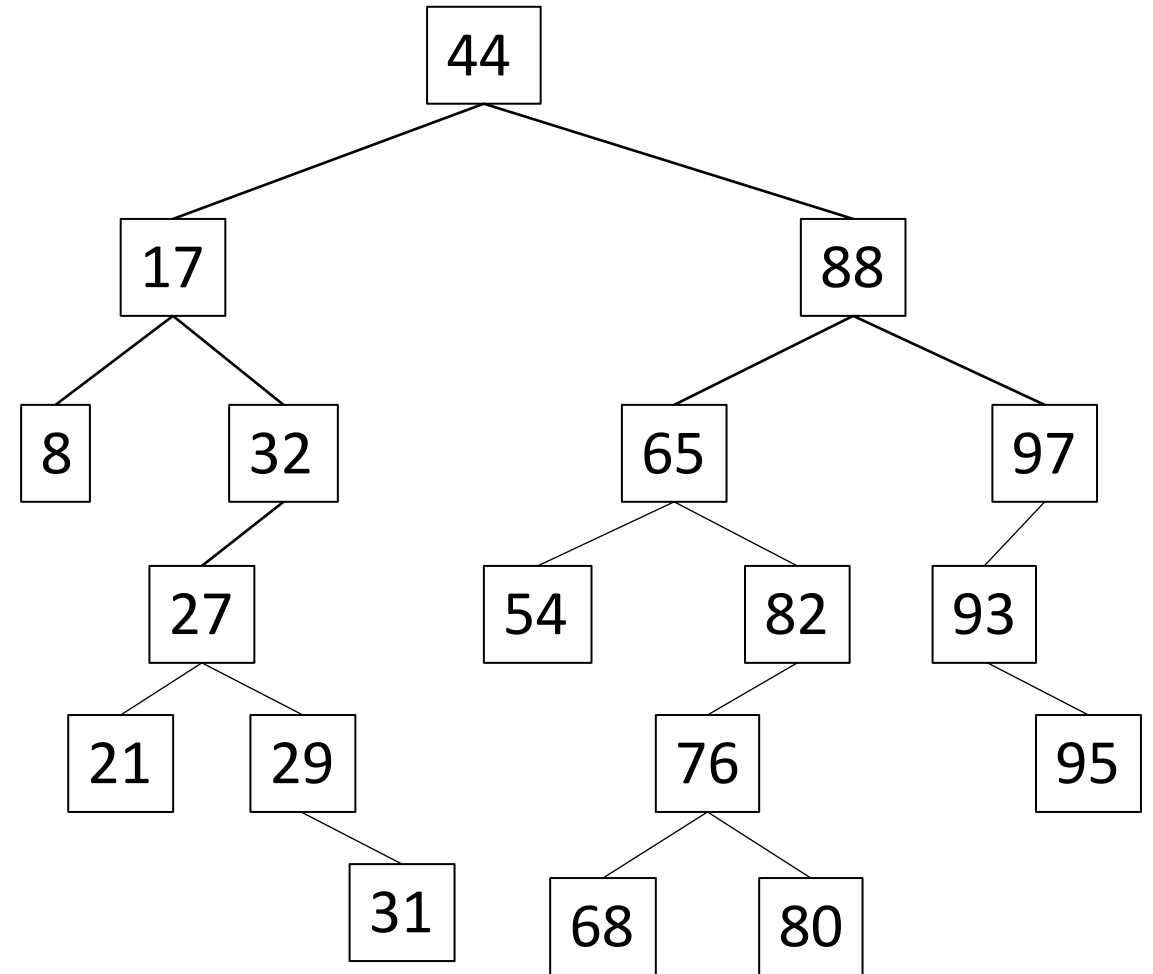
Binary Search Tree - Insertion

`insert(95);`

Step 1: Find where it should go.

Step 2: Modify pointers.

Any trends??



Binary Search Tree - Insertion

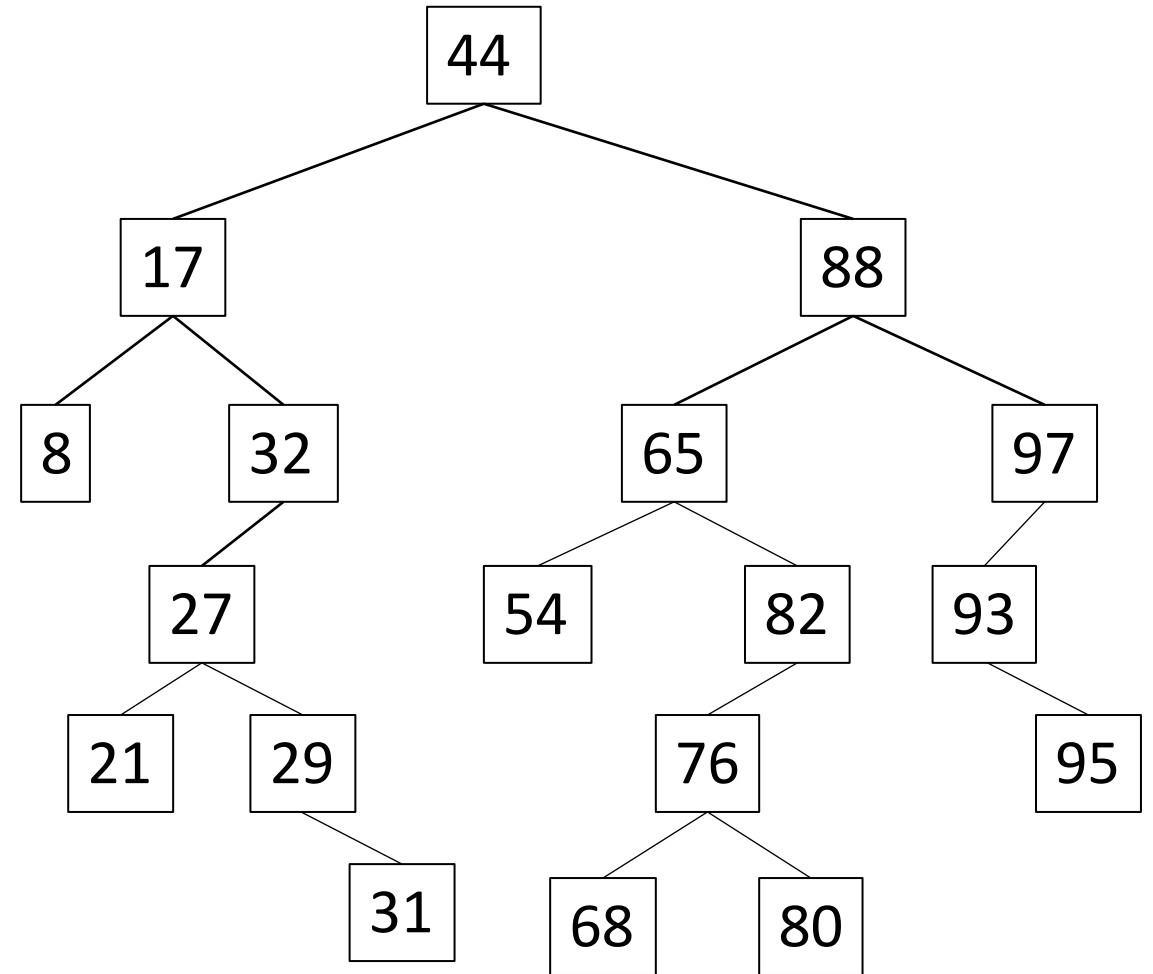
`insert(95);`

Step 1: Find where it should go.

Step 2: Modify pointers.

Any trends??

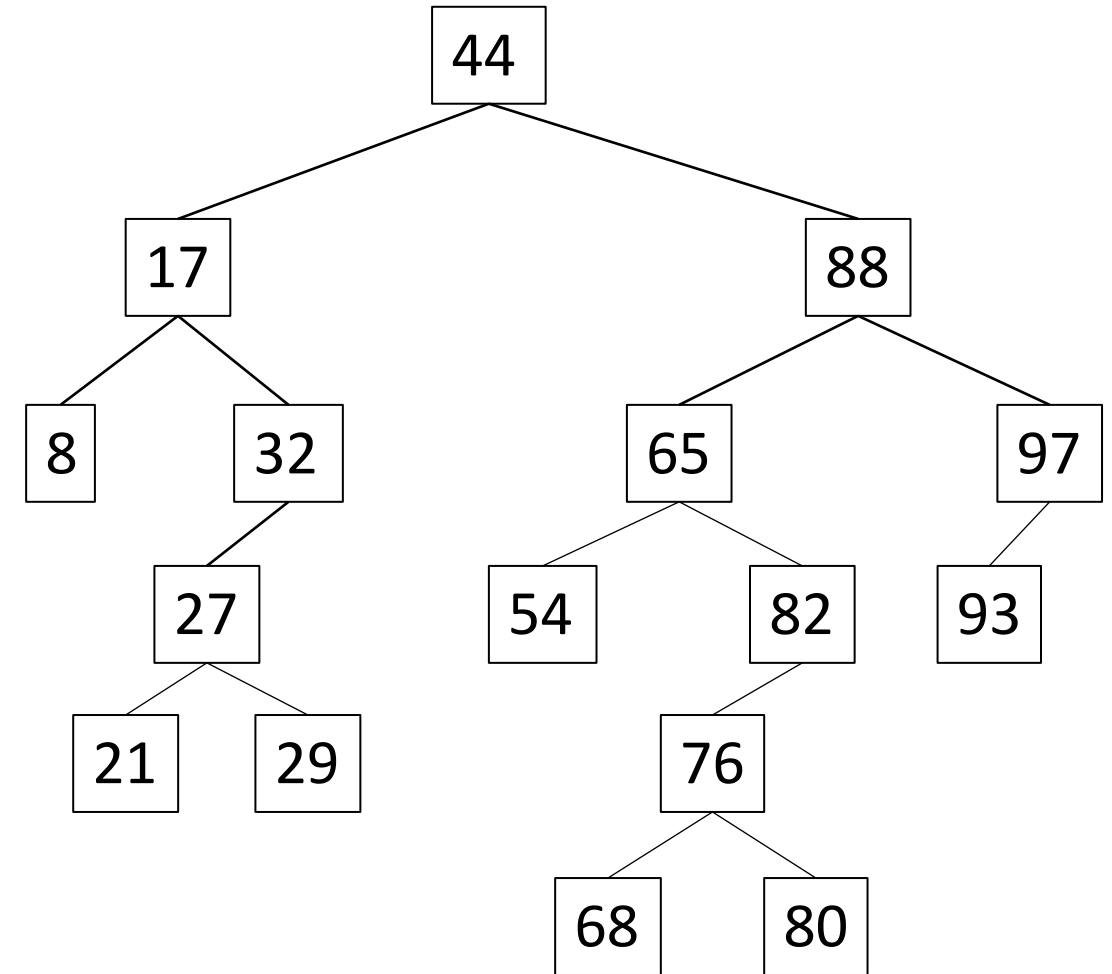
Always insert a new leaf!



Binary Search Tree - Insertion

insert(28);

```
public void insert(int newValue) {
```



```
}
```

Binary Search Tree - Insertion

insert(28);

```
public void insert(int newValue) {  
    if (root == null) {  
        } else {
```

root  **null**

```
}
```

Binary Search Tree - Insertion

insert(28);

```
public void insert(int newValue) {  
    if (root == null) {  
        root = new Node(newValue);  
    } else {
```

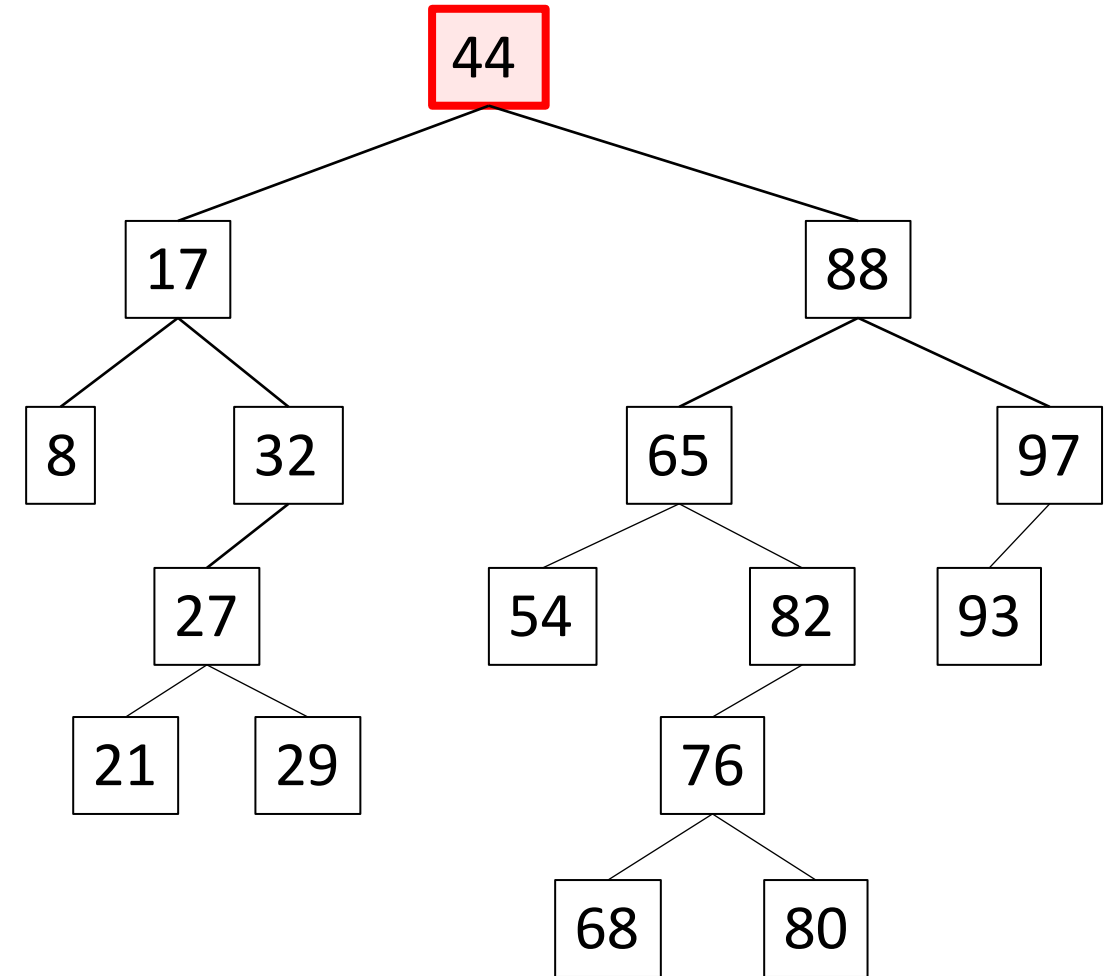


```
}
```

Binary Search Tree - Insertion

insert(28);

```
public void insert(int newValue) {  
    if (root == null) {  
        root = new Node(newValue);  
    } else {  
        Node currentNode = root;
```

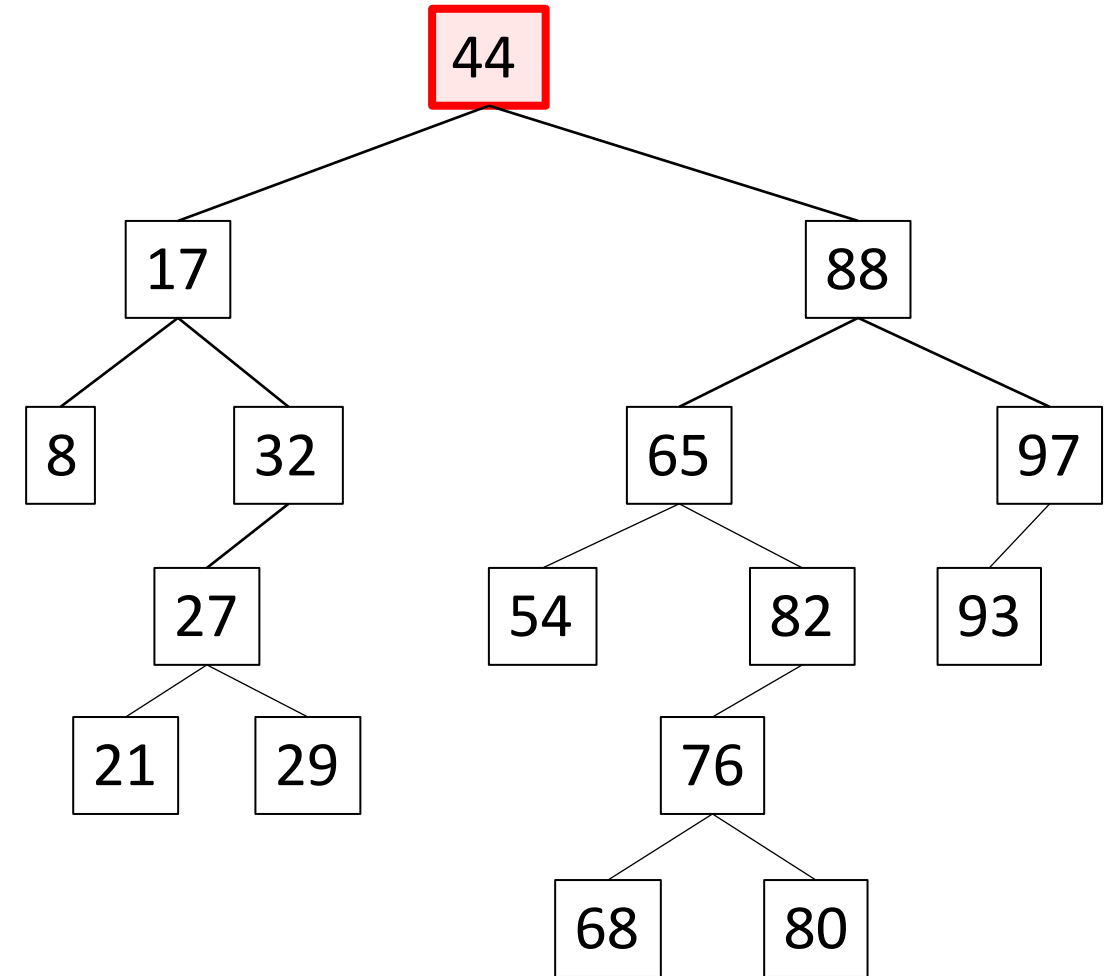


```
}
```

Binary Search Tree - Insertion

insert(28);

```
public void insert(int newValue) {  
    if (root == null) {  
        root = new Node(newValue);  
    } else {  
        Node currentNode = root;  
        boolean placed = false;
```

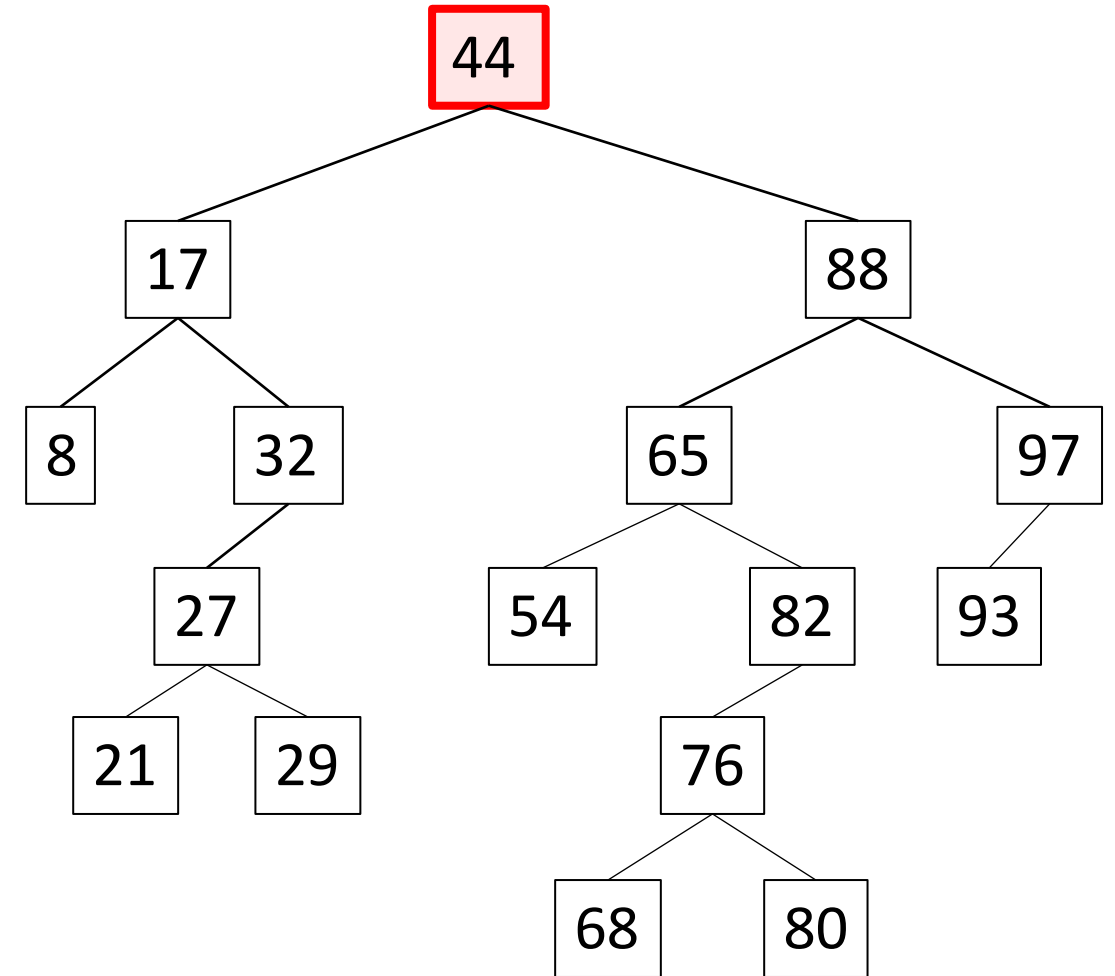


```
}
```

Binary Search Tree - Insertion

insert(28);

```
public void insert(int newValue) {  
    if (root == null) {  
        root = new Node(newValue);  
    } else {  
        Node currentNode = root;  
        boolean placed = false;  
        while (!placed) {
```

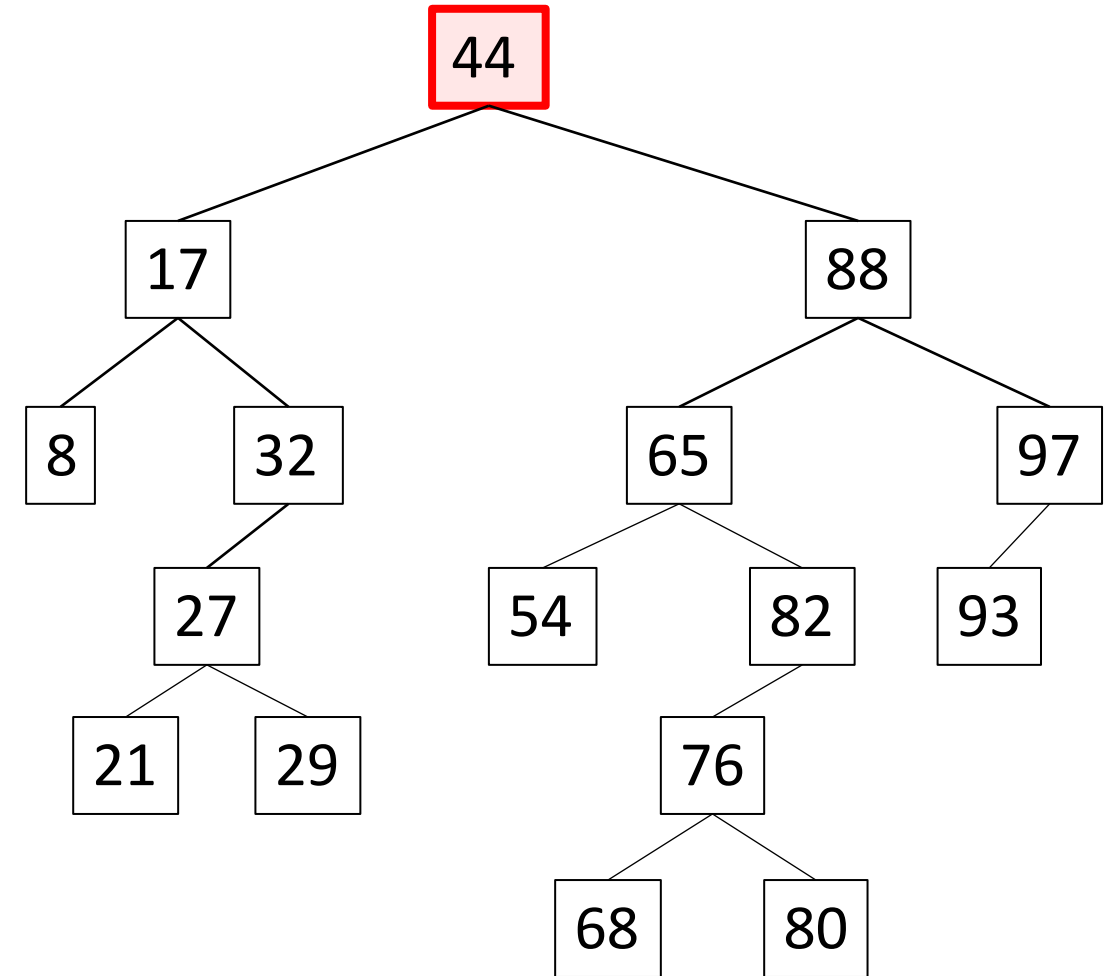


```
}
```


Binary Search Tree - Insertion

insert(28);

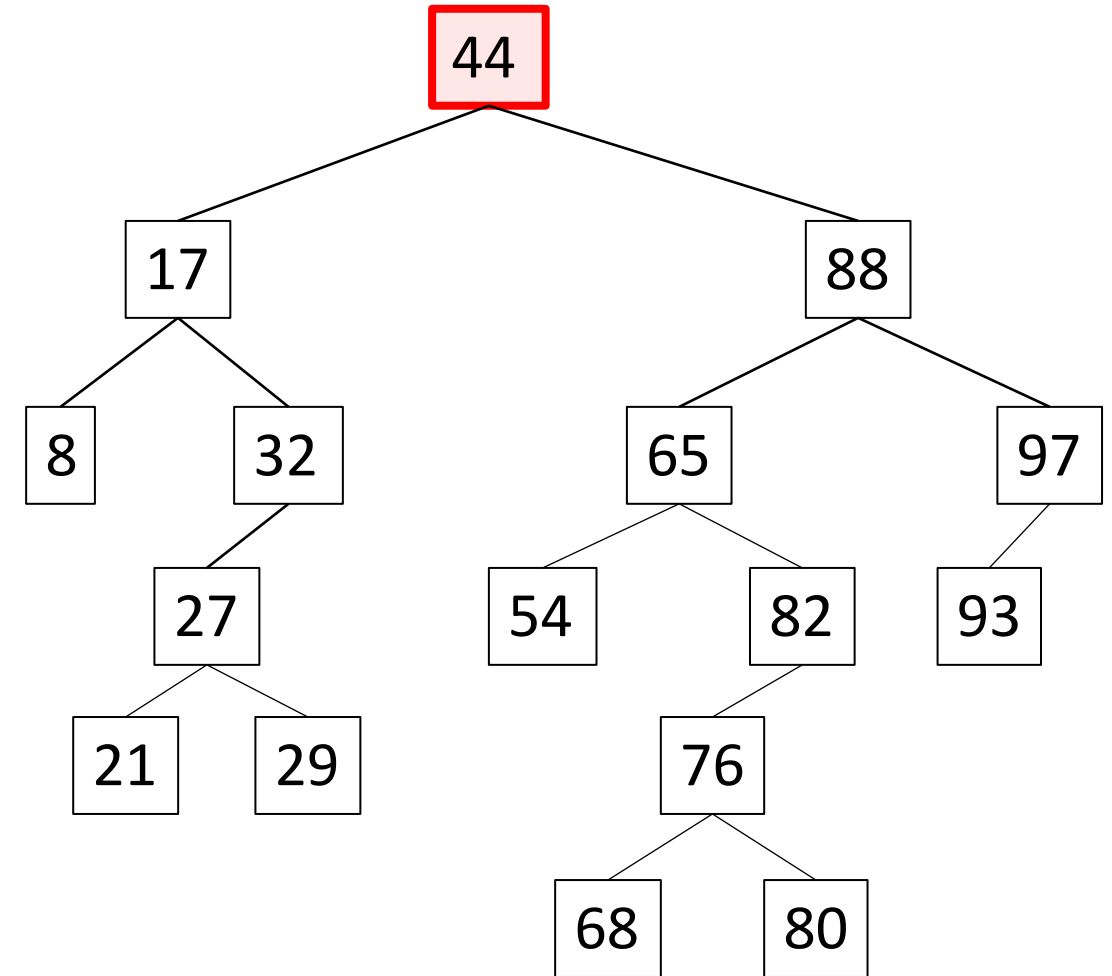
```
public void insert(int newValue) {  
    if (root == null) {  
        root = new Node(newValue);  
    } else {  
        Node currentNode = root;  
        boolean placed = false;  
        while (!placed) {  
            if (newValue < currentNode.getValue()) {  
  
            } else {  
  
            }  
        }  
    }  
}
```



Binary Search Tree - Insertion

insert(28);

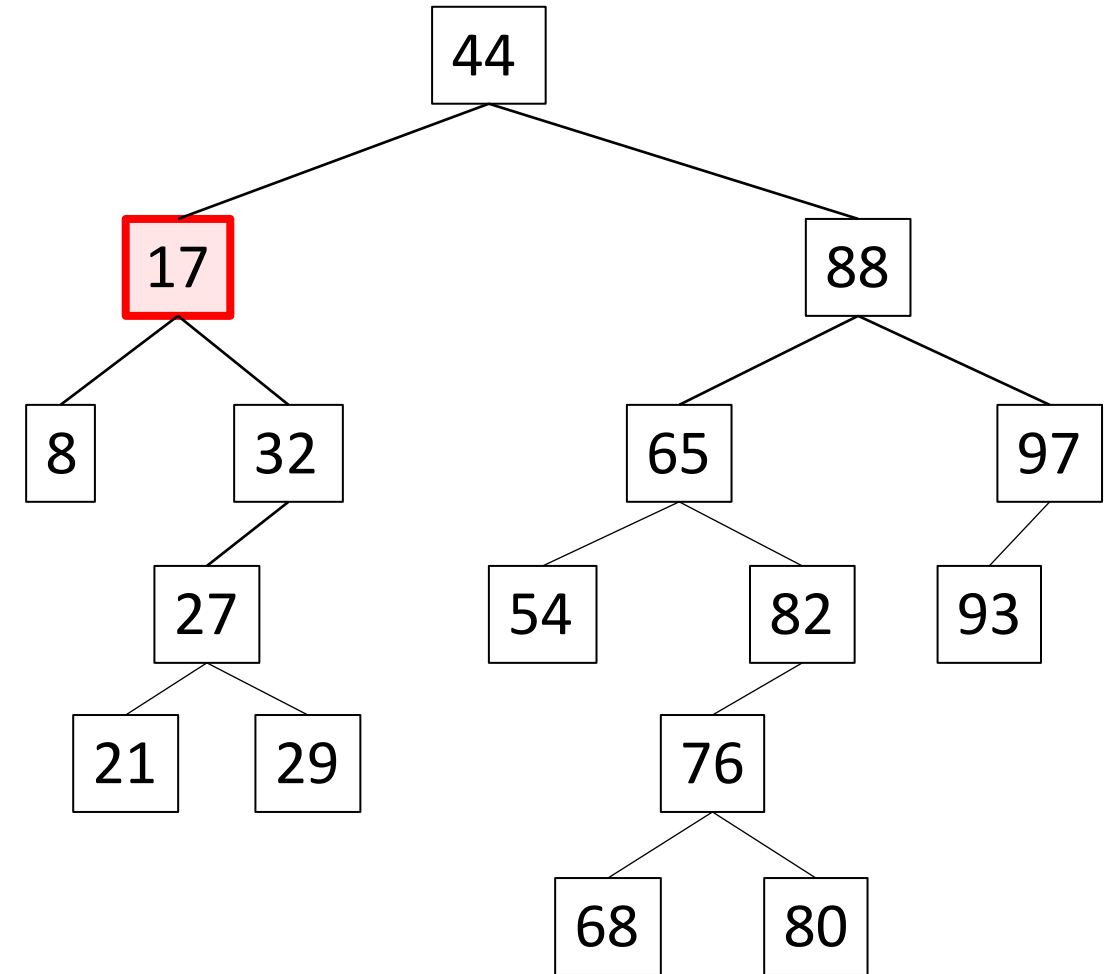
```
public void insert(int newValue) {  
    if (root == null) {  
        root = new Node(newValue);  
    } else {  
        Node currentNode = root;  
        boolean placed = false;  
        while (!placed) {  
            if (newValue < currentNode.getValue()) {  
                if (currentNode.getLeft() != null) {  
  
                } else {  
  
                }  
            } else {  
  
            }  
        }  
    }  
}
```



Binary Search Tree - Insertion

insert(28);

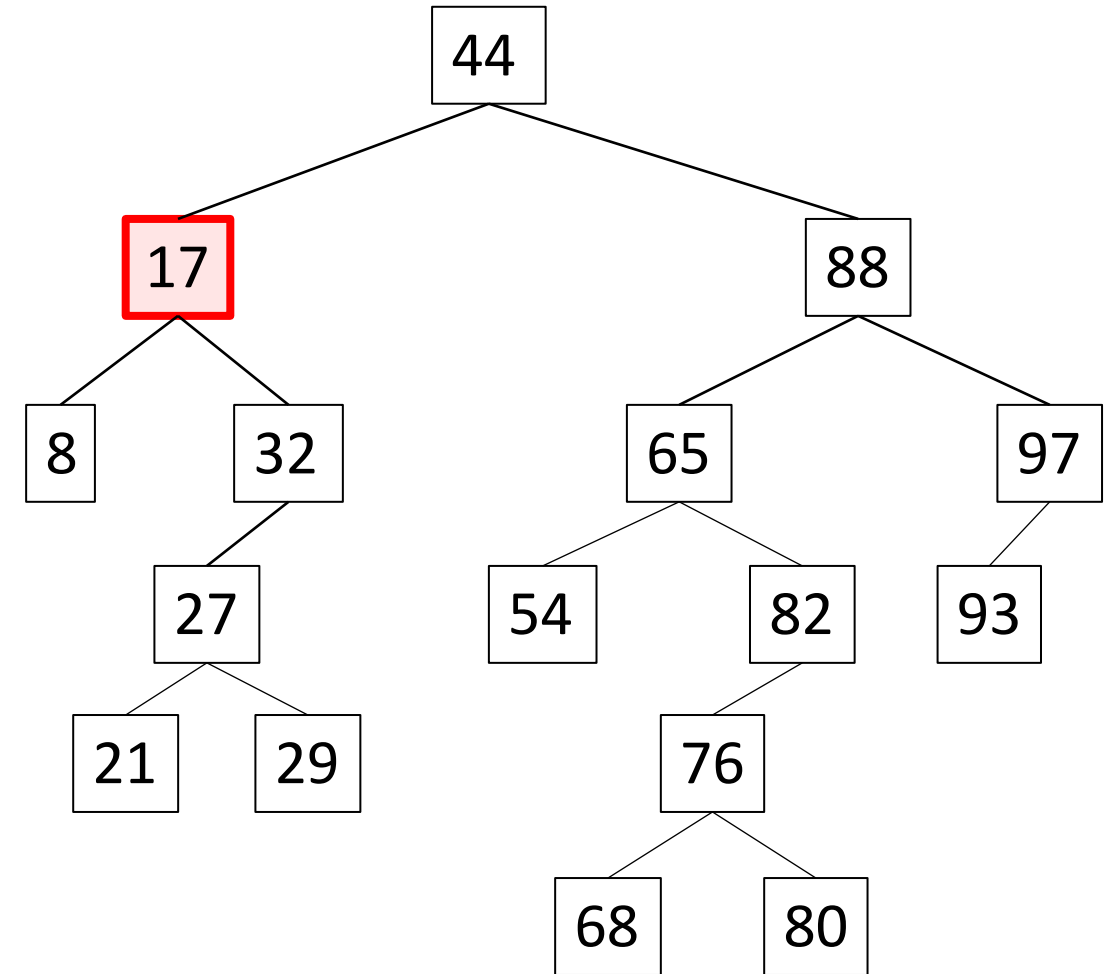
```
public void insert(int newValue) {  
    if (root == null) {  
        root = new Node(newValue);  
    } else {  
        Node currentNode = root;  
        boolean placed = false;  
        while (!placed) {  
            if (newValue < currentNode.getValue()) {  
                if (currentNode.getLeft() != null) {  
                    currentNode = currentNode.getLeft();  
                } else {  
                    }  
                } else {  
                    }  
            }  
        }  
    }  
}
```



Binary Search Tree - Insertion

insert(28);

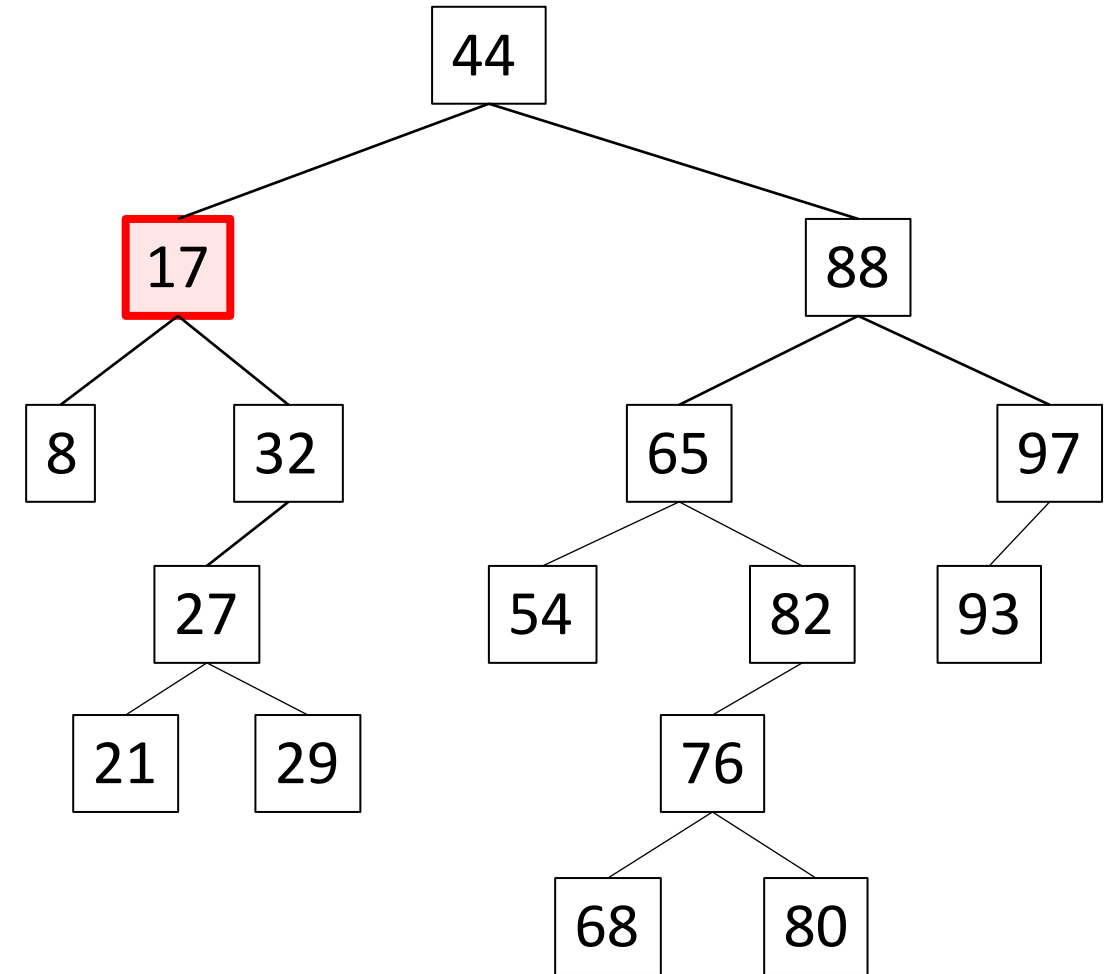
```
public void insert(int newValue) {  
    if (root == null) {  
        root = new Node(newValue);  
    } else {  
        Node currentNode = root;  
        boolean placed = false;  
        while (!placed) {  
            if (newValue < currentNode.getValue()) {  
                if (currentNode.getLeft() != null) {  
                    currentNode = currentNode.getLeft();  
                } else {  
                    }  
                } else {  
                    }  
            }  
        }  
    }  
}
```



Binary Search Tree - Insertion

insert(28);

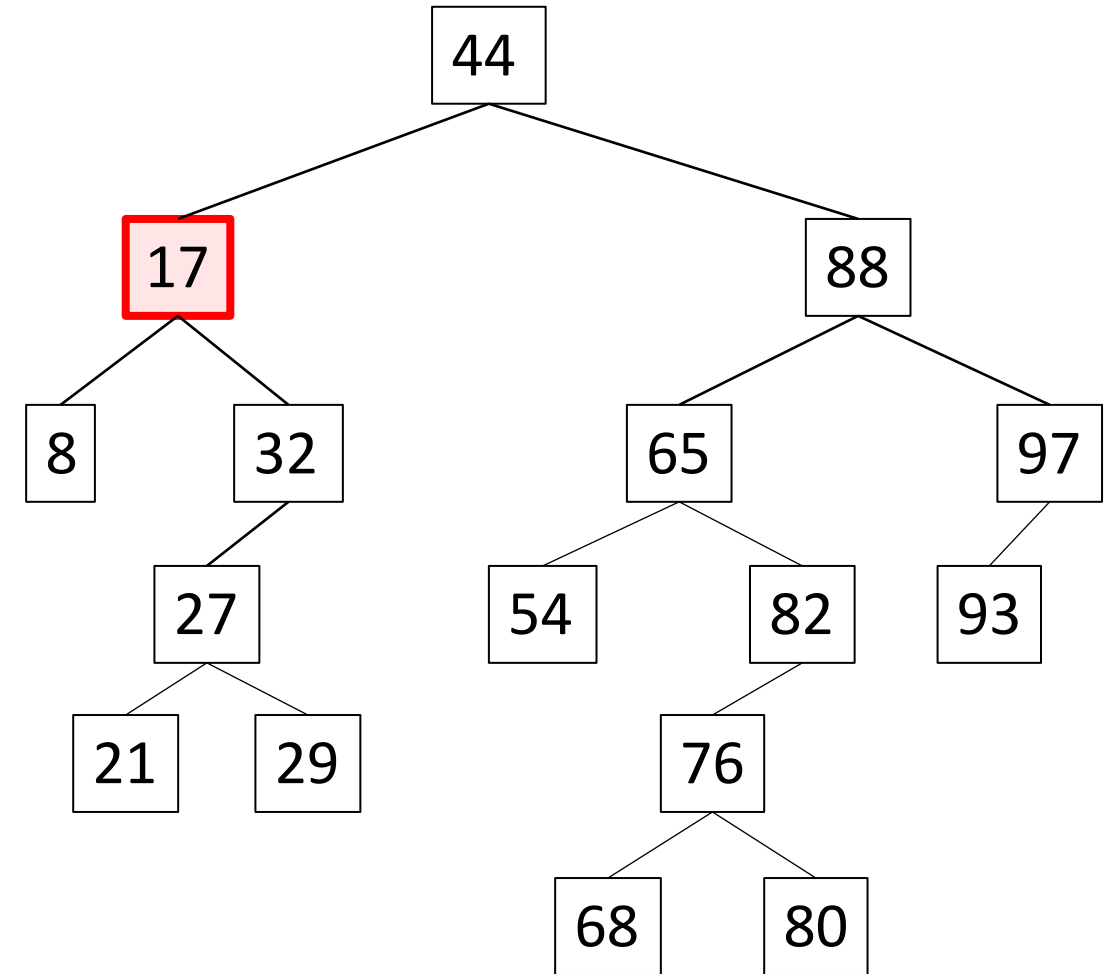
```
public void insert(int newValue) {  
    if (root == null) {  
        root = new Node(newValue);  
    } else {  
        Node currentNode = root;  
        boolean placed = false;  
        while (!placed) {  
            if (newValue < currentNode.getValue()) {  
                if (currentNode.getLeft() != null) {  
                    currentNode = currentNode.getLeft();  
                } else {  
                    }  
                } else {  
                    }  
            }  
        }  
    }  
}
```



Binary Search Tree - Insertion

insert(28);

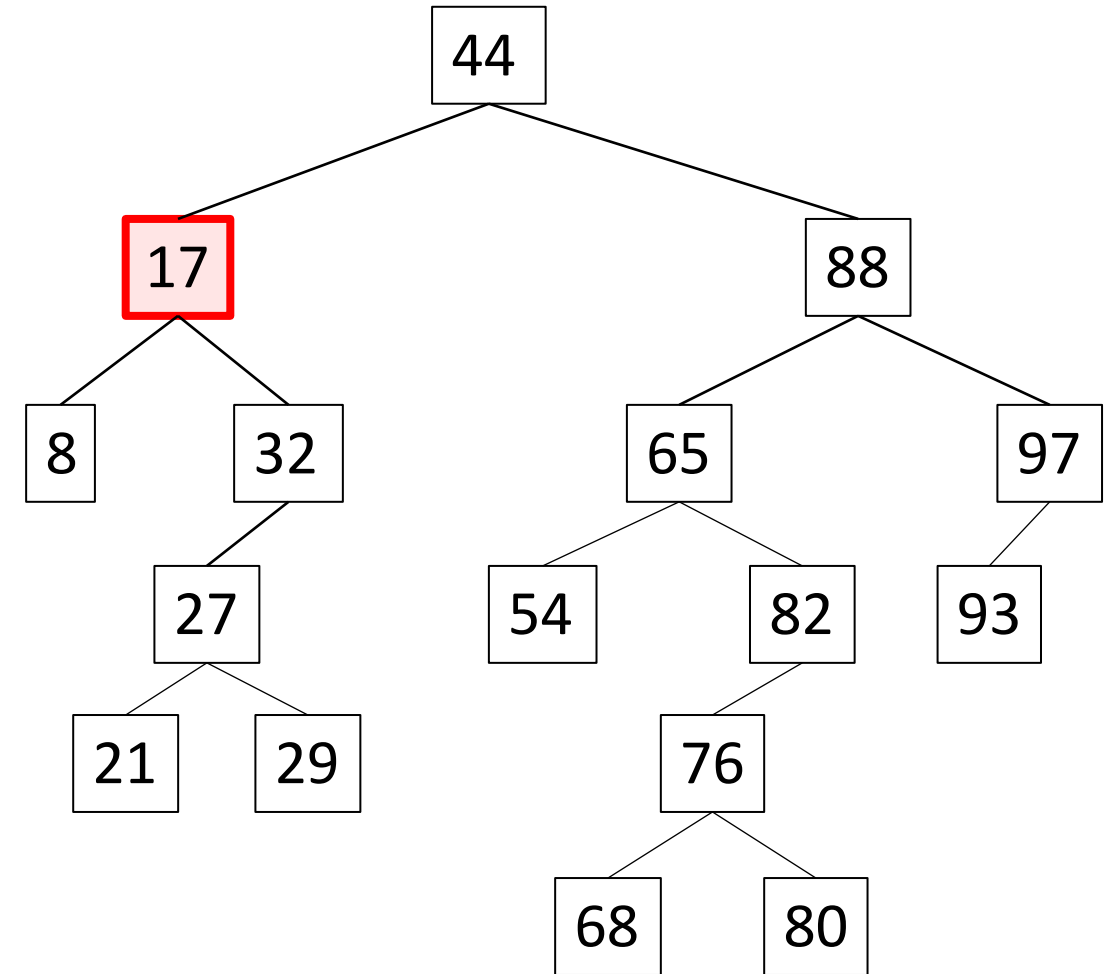
```
public void insert(int newValue) {  
    if (root == null) {  
        root = new Node(newValue);  
    } else {  
        Node currentNode = root;  
        boolean placed = false;  
        while (!placed) {  
            if (newValue < currentNode.getValue()) {  
                if (currentNode.getLeft() != null) {  
                    currentNode = currentNode.getLeft();  
                } else {  
                    } else {  
                }  
            }  
        }  
    }  
}
```



Binary Search Tree - Insertion

insert(28);

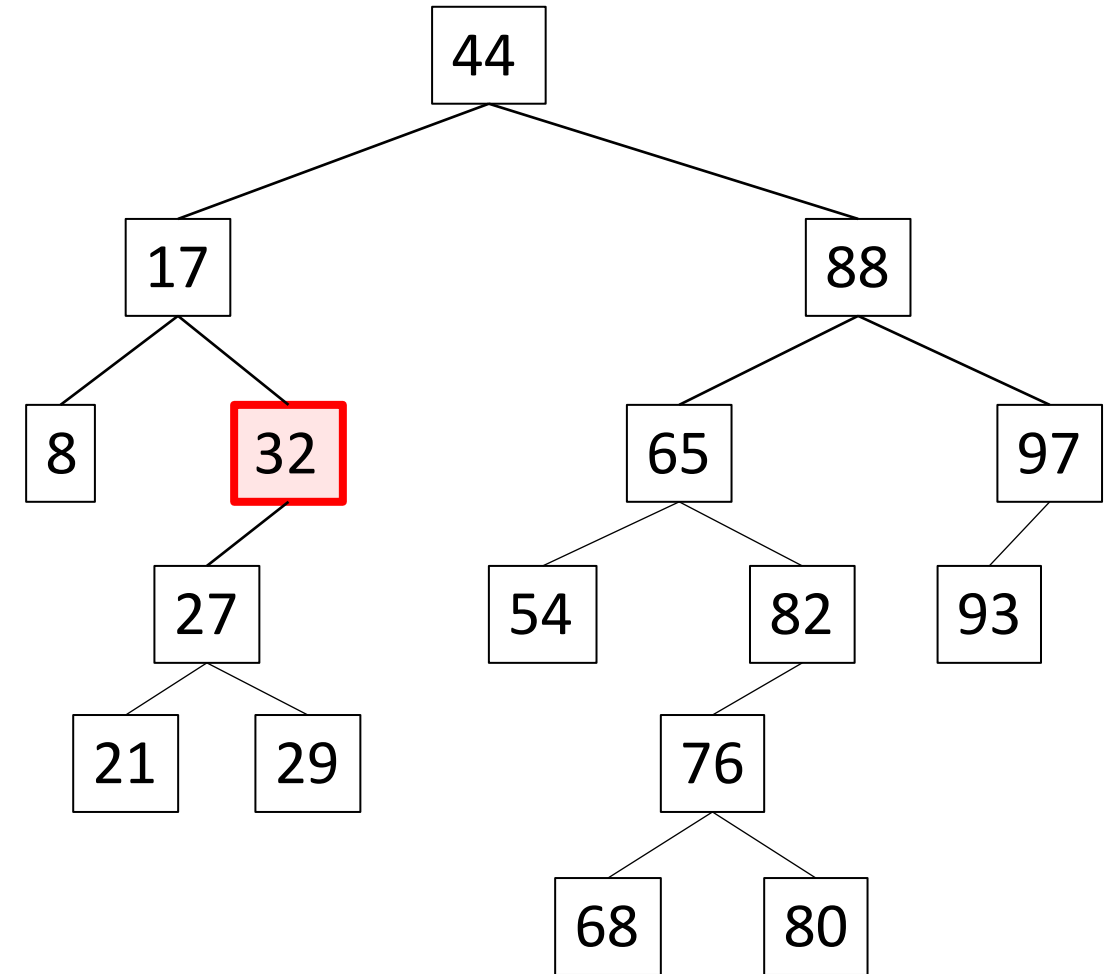
```
public void insert(int newValue) {  
    if (root == null) {  
        root = new Node(newValue);  
    } else {  
        Node currentNode = root;  
        boolean placed = false;  
        while (!placed) {  
            if (newValue < currentNode.getValue()) {  
                if (currentNode.getLeft() != null) {  
                    currentNode = currentNode.getLeft();  
                } else {  
                    }  
                } else {  
                    if (currentNode.getRight() != null) {  
                        }  
                    } else {  
                        }  
                }  
            }  
        }  
    }  
}
```



Binary Search Tree - Insertion

insert(28);

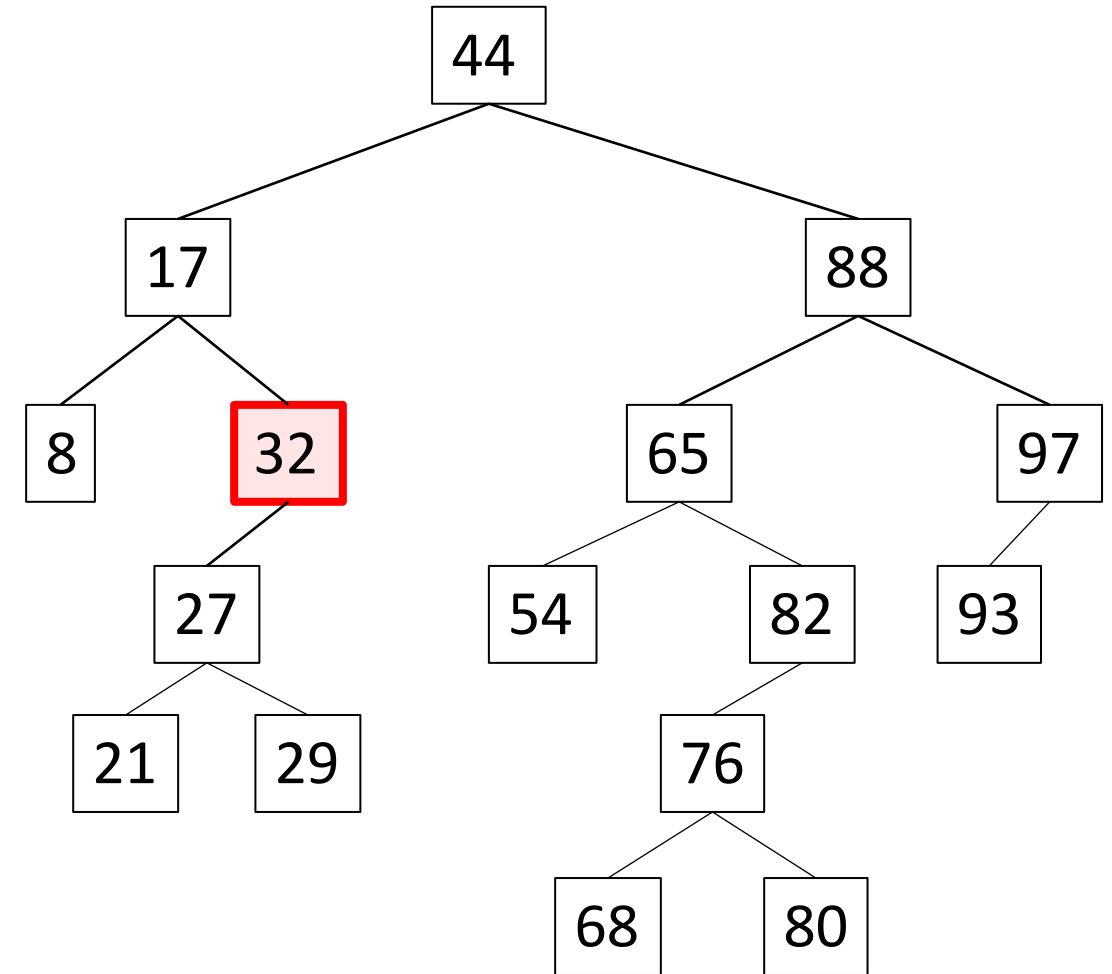
```
public void insert(int newValue) {  
    if (root == null) {  
        root = new Node(newValue);  
    } else {  
        Node currentNode = root;  
        boolean placed = false;  
        while (!placed) {  
            if (newValue < currentNode.getValue()) {  
                if (currentNode.getLeft() != null) {  
                    currentNode = currentNode.getLeft();  
                } else {  
                    }  
                } else {  
                    if (currentNode.getRight() != null) {  
                        currentNode = currentNode.getRight();  
                    } else {  
                        }  
                    }  
            }  
        }  
    }  
}
```



Binary Search Tree - Insertion

insert(28);

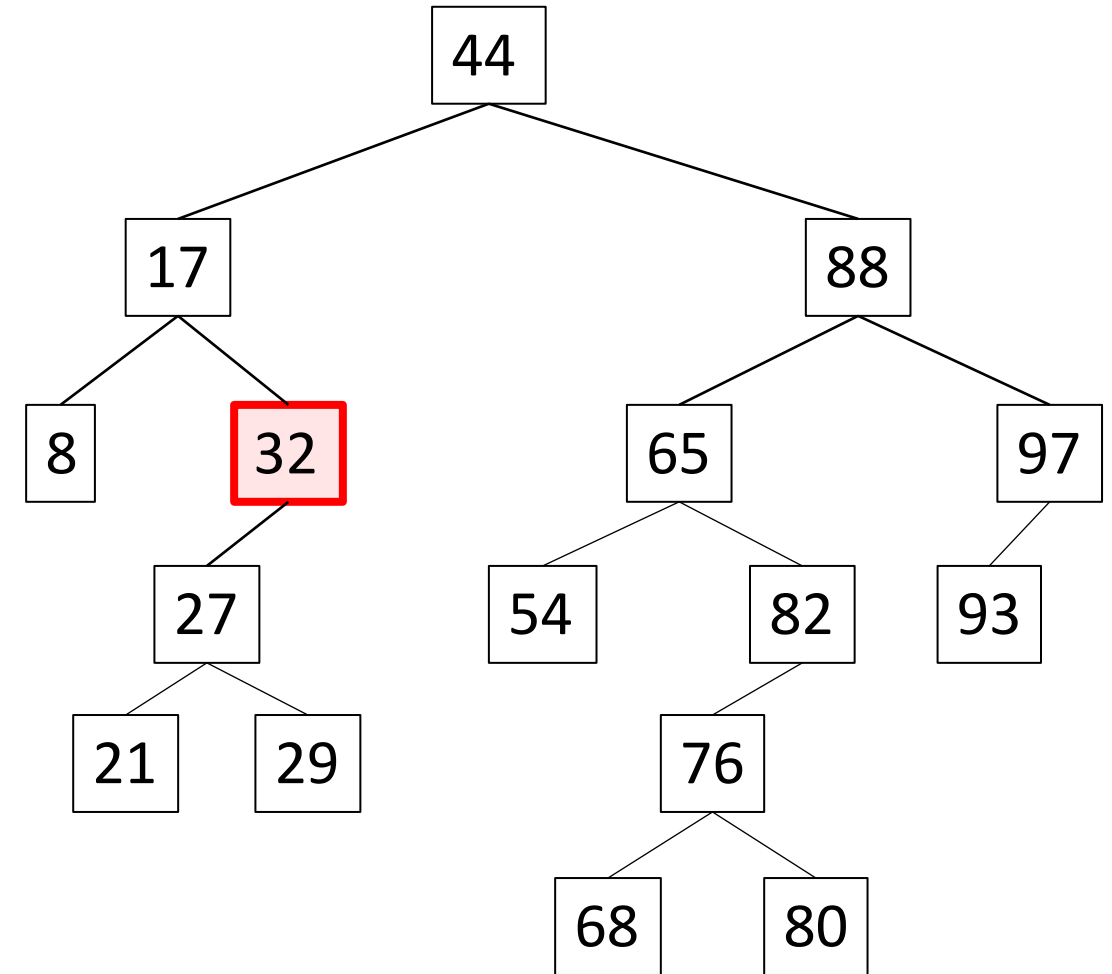
```
public void insert(int newValue) {  
    if (root == null) {  
        root = new Node(newValue);  
    } else {  
        Node currentNode = root;  
        boolean placed = false;  
        while (!placed) {  
            if (newValue < currentNode.getValue()) {  
                if (currentNode.getLeft() != null) {  
                    currentNode = currentNode.getLeft();  
                } else {  
                    }  
                } else {  
                    if (currentNode.getRight() != null) {  
                        currentNode = currentNode.getRight();  
                    } else {  
                        }  
                    }  
            }  
        }  
    }  
}
```



Binary Search Tree - Insertion

`insert(28);`

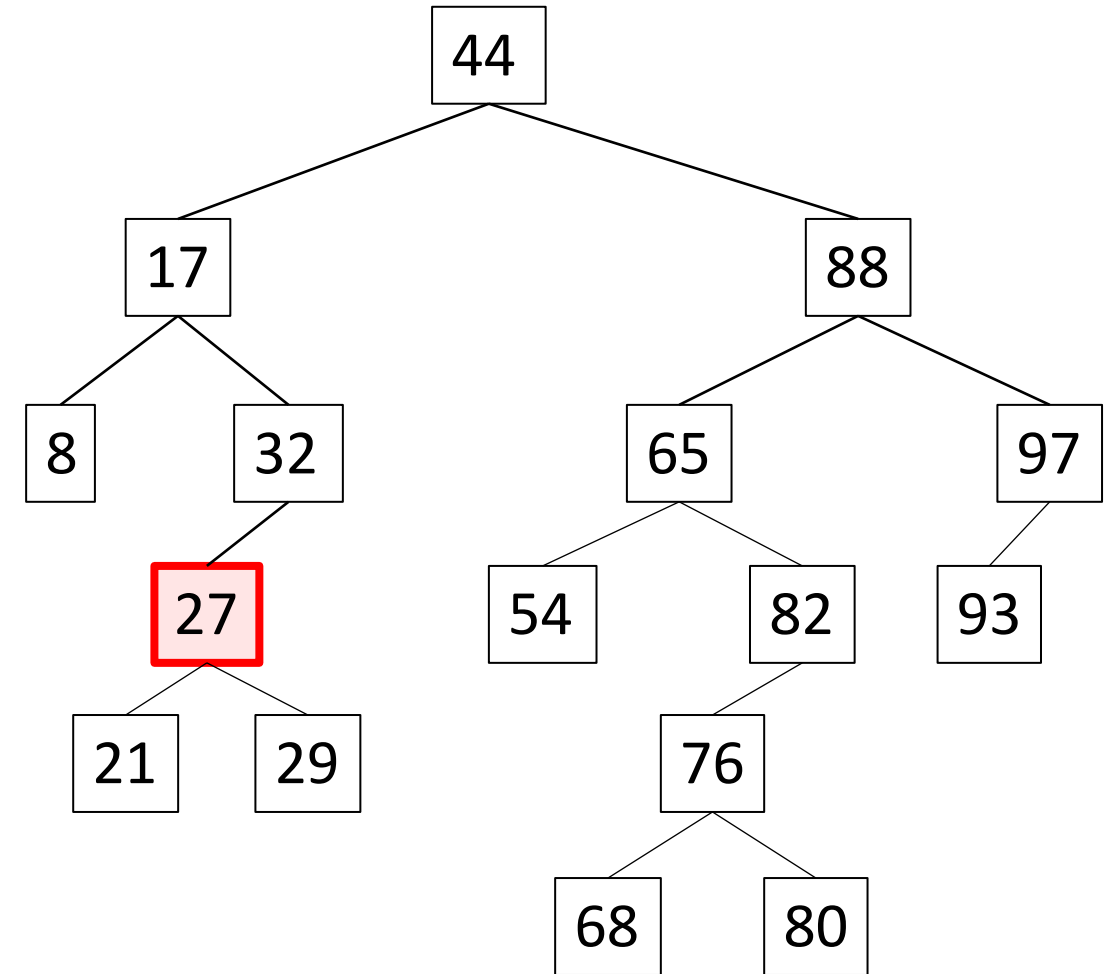
```
public void insert(int newValue) {  
    if (root == null) {  
        root = new Node(newValue);  
    } else {  
        Node currentNode = root;  
        boolean placed = false;  
        while (!placed) {  
            if (newValue < currentNode.getValue()) {  
                if (currentNode.getLeft() != null) {  
                    currentNode = currentNode.getLeft();  
                } else {  
                    }  
                } else {  
                    if (currentNode.getRight() != null) {  
                        currentNode = currentNode.getRight();  
                    } else {  
                        }  
                    }  
                }  
            }  
        }  
    }  
}
```



Binary Search Tree - Insertion

insert(28);

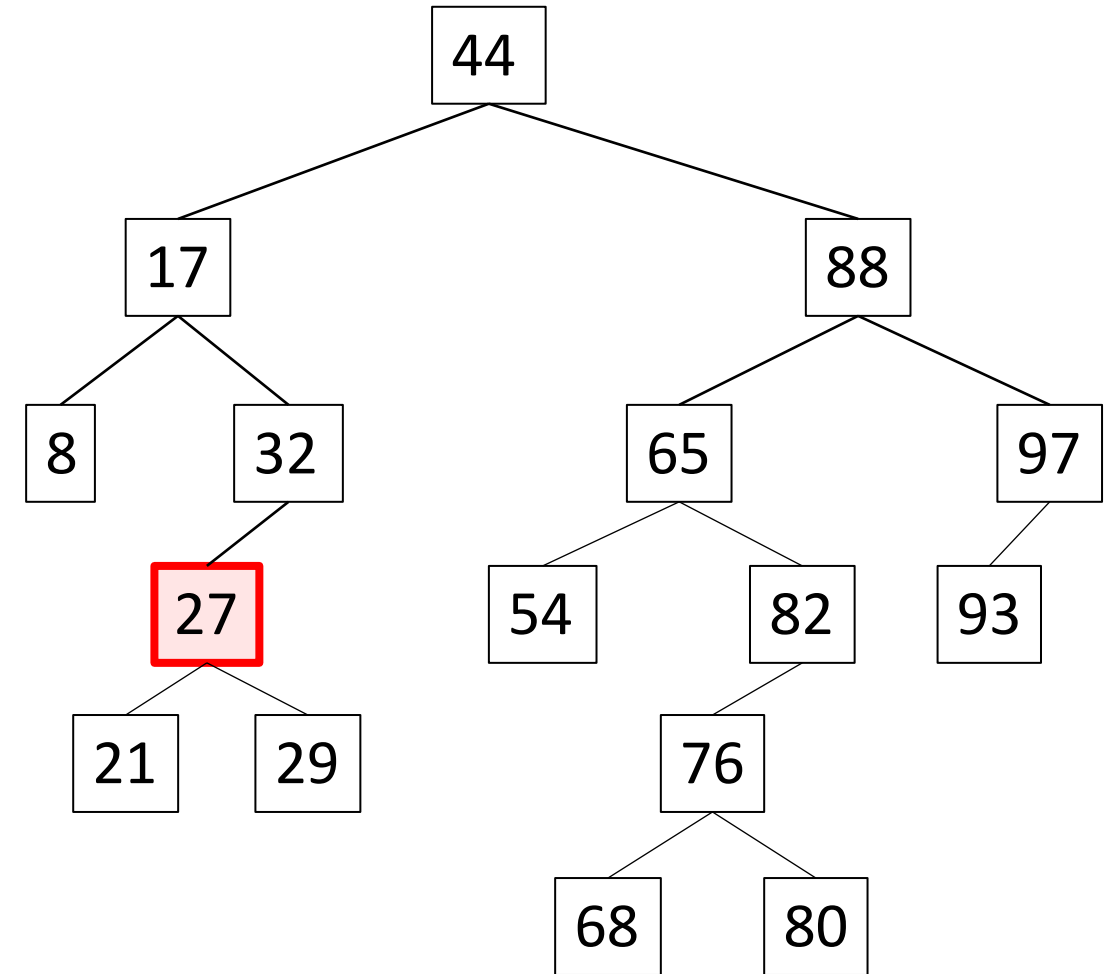
```
public void insert(int newValue) {  
    if (root == null) {  
        root = new Node(newValue);  
    } else {  
        Node currentNode = root;  
        boolean placed = false;  
        while (!placed) {  
            if (newValue < currentNode.getValue()) {  
                if (currentNode.getLeft() != null) {  
                    currentNode = currentNode.getLeft();  
                } else {  
                    }  
                } else {  
                    if (currentNode.getRight() != null) {  
                        currentNode = currentNode.getRight();  
                    } else {  
                        }  
                    }  
                }  
            }  
        }  
    }  
}
```



Binary Search Tree - Insertion

`insert(28);`

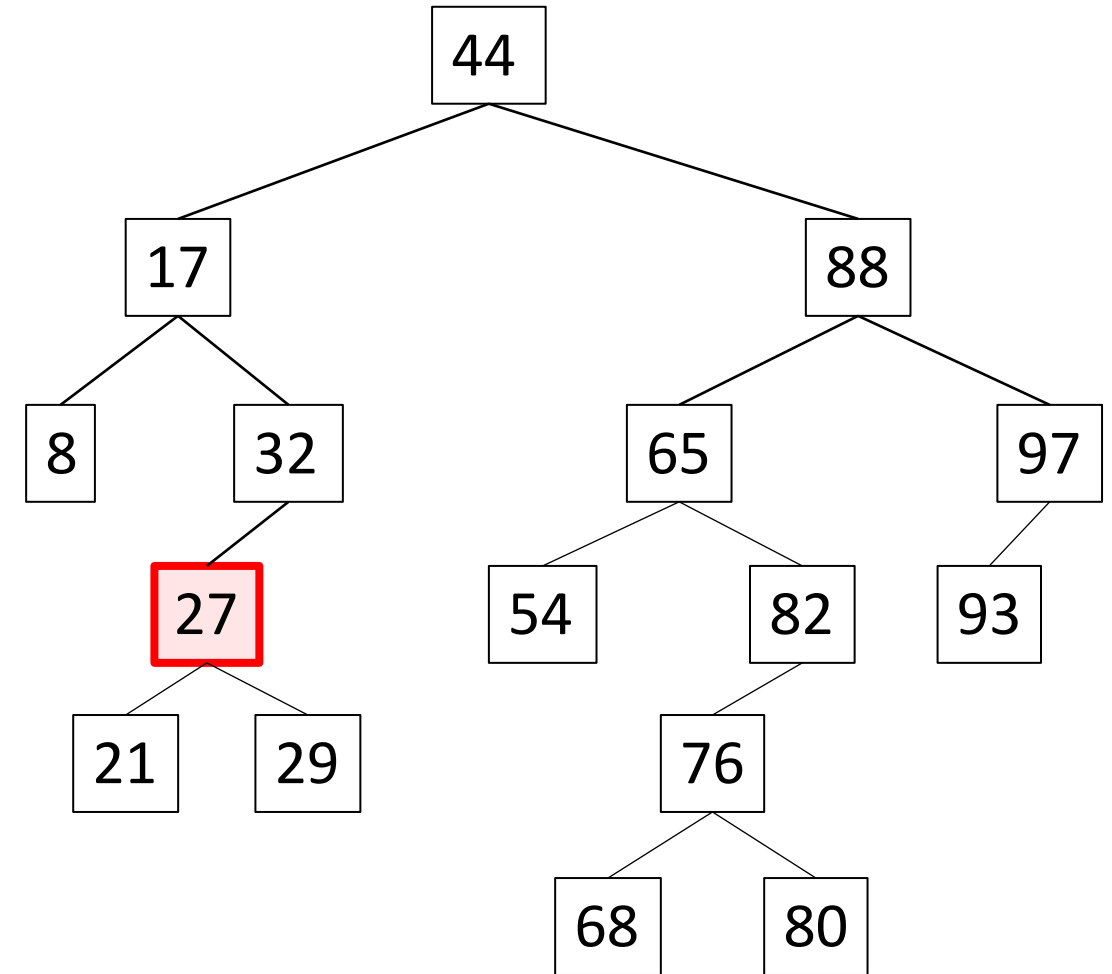
```
public void insert(int newValue) {  
    if (root == null) {  
        root = new Node(newValue);  
    } else {  
        Node currentNode = root;  
        boolean placed = false;  
        while (!placed) {  
            if (newValue < currentNode.getValue()) {  
                if (currentNode.getLeft() != null) {  
                    currentNode = currentNode.getLeft();  
                } else {  
                    }  
                } else {  
                    if (currentNode.getRight() != null) {  
                        currentNode = currentNode.getRight();  
                    } else {  
                        }  
                    }  
            }  
        }  
    }  
}
```



Binary Search Tree - Insertion

insert(28);

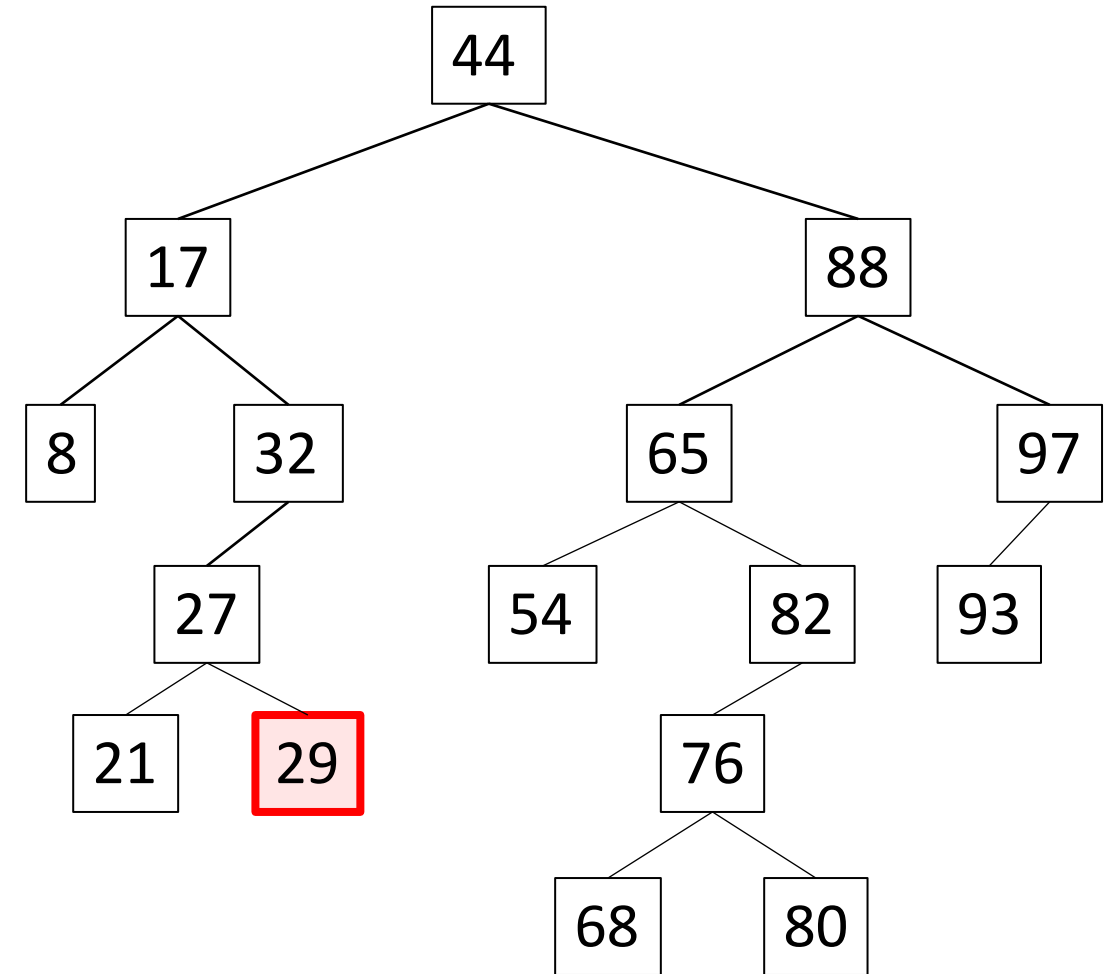
```
public void insert(int newValue) {  
    if (root == null) {  
        root = new Node(newValue);  
    } else {  
        Node currentNode = root;  
        boolean placed = false;  
        while (!placed) {  
            if (newValue < currentNode.getValue()) {  
                if (currentNode.getLeft() != null) {  
                    currentNode = currentNode.getLeft();  
                } else {  
                    }  
                } else {  
                    if (currentNode.getRight() != null) {  
                        currentNode = currentNode.getRight();  
                    } else {  
                        }  
                    }  
                }  
            }  
        }  
    }  
}
```



Binary Search Tree - Insertion

`insert(28);`

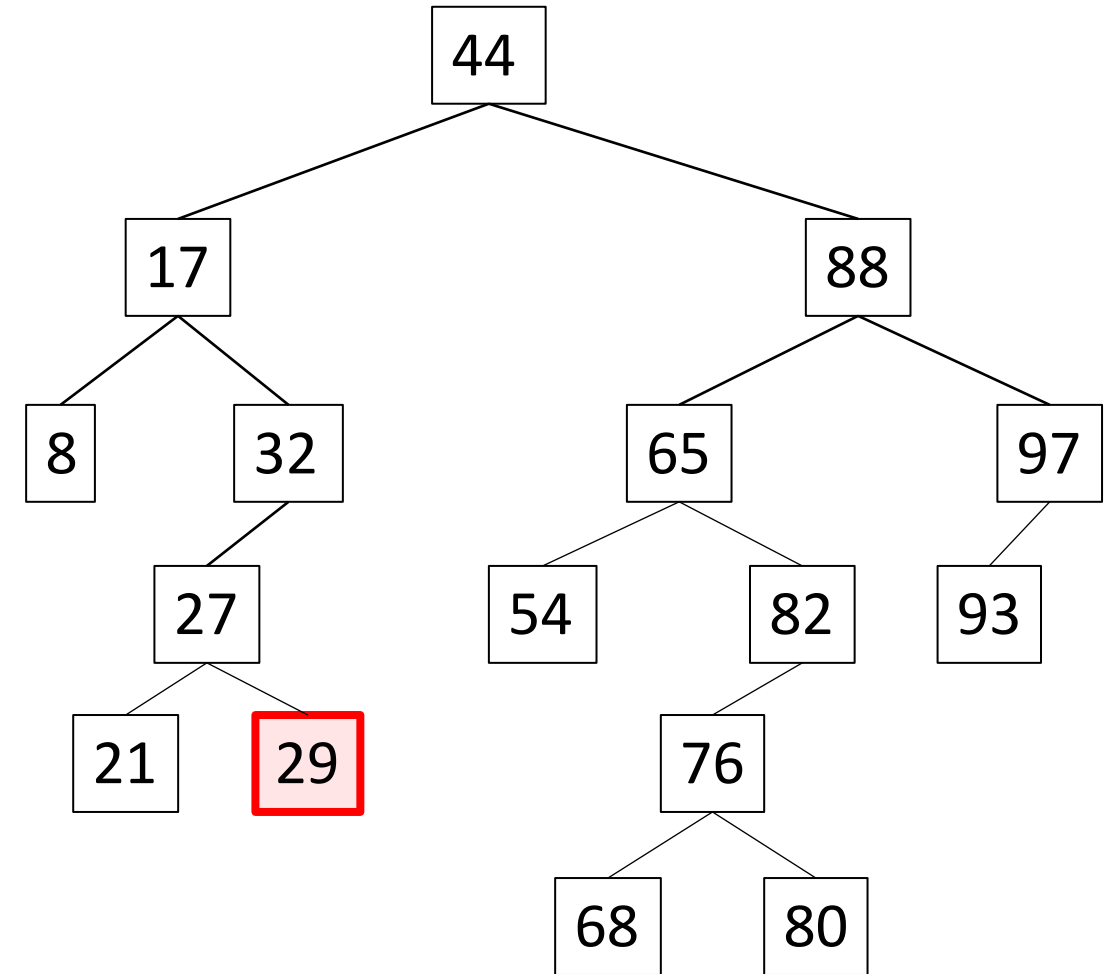
```
public void insert(int newValue) {  
    if (root == null) {  
        root = new Node(newValue);  
    } else {  
        Node currentNode = root;  
        boolean placed = false;  
        while (!placed) {  
            if (newValue < currentNode.getValue()) {  
                if (currentNode.getLeft() != null) {  
                    currentNode = currentNode.getLeft();  
                } else {  
                    }  
                } else {  
                    if (currentNode.getRight() != null) {  
                        currentNode = currentNode.getRight();  
                    } else {  
                        }  
                    }  
                }  
            }  
        }  
    }  
}
```



Binary Search Tree - Insertion

`insert(28);`

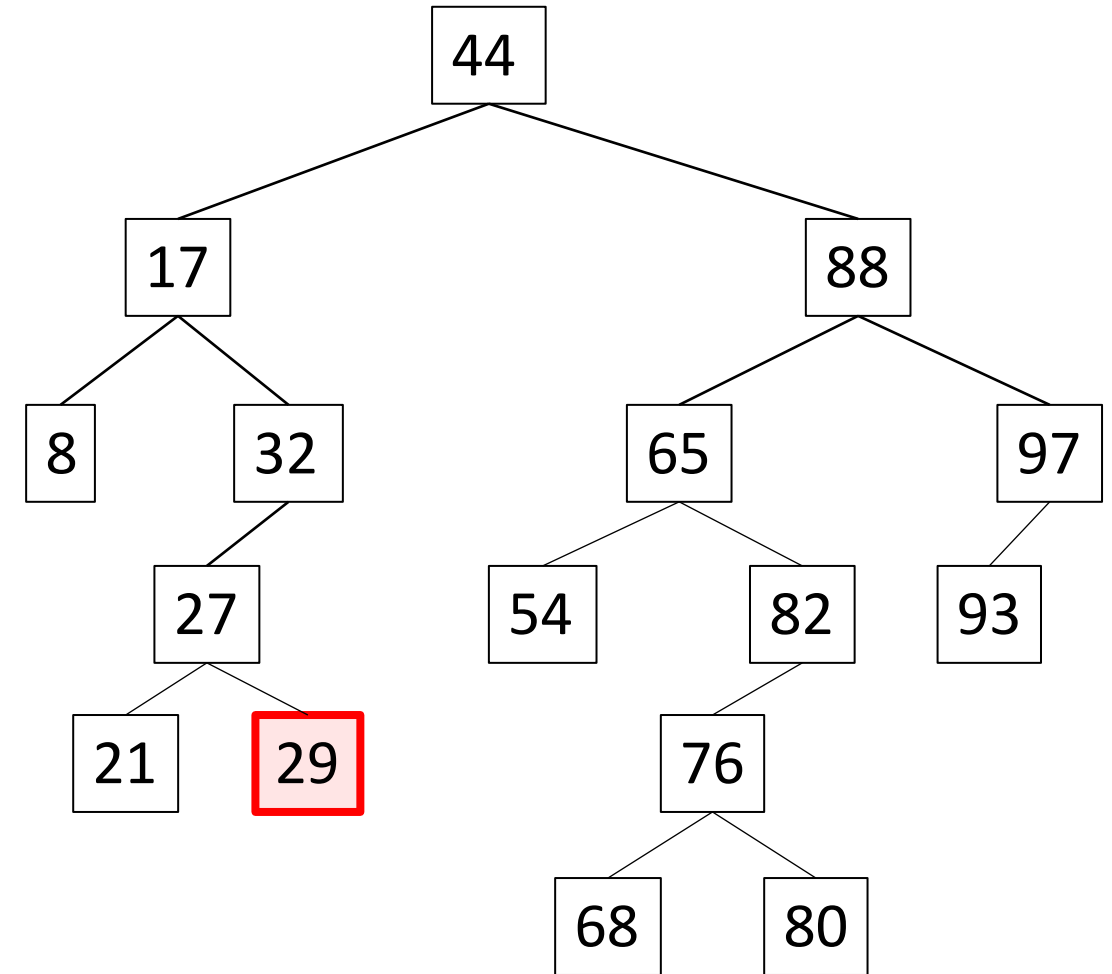
```
public void insert(int newValue) {  
    if (root == null) {  
        root = new Node(newValue);  
    } else {  
        Node currentNode = root;  
        boolean placed = false;  
        while (!placed) {  
            if (newValue < currentNode.getValue()) {  
                if (currentNode.getLeft() != null) {  
                    currentNode = currentNode.getLeft();  
                } else {  
                    }  
                } else {  
                    if (currentNode.getRight() != null) {  
                        currentNode = currentNode.getRight();  
                    } else {  
                        }  
                    }  
            }  
        }  
    }  
}
```



Binary Search Tree - Insertion

insert(28);

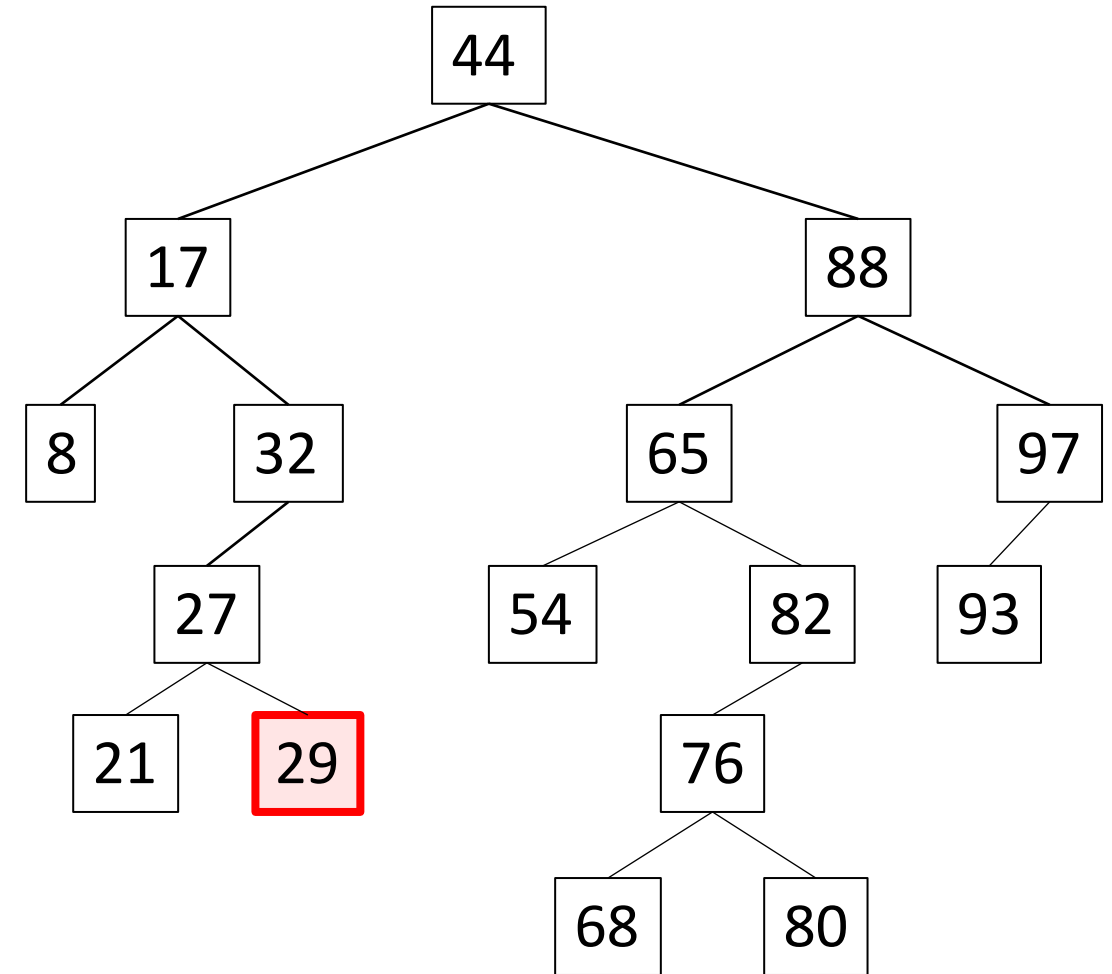
```
public void insert(int newValue) {  
    if (root == null) {  
        root = new Node(newValue);  
    } else {  
        Node currentNode = root;  
        boolean placed = false;  
        while (!placed) {  
            if (newValue < currentNode.getValue()) {  
                if (currentNode.getLeft() != null) {  
                    currentNode = currentNode.getLeft();  
                } else {  
                    }  
                } else {  
                    if (currentNode.getRight() != null) {  
                        currentNode = currentNode.getRight();  
                    } else {  
                        }  
                    }  
                }  
            }  
        }  
    }  
}
```



Binary Search Tree - Insertion

insert(28);

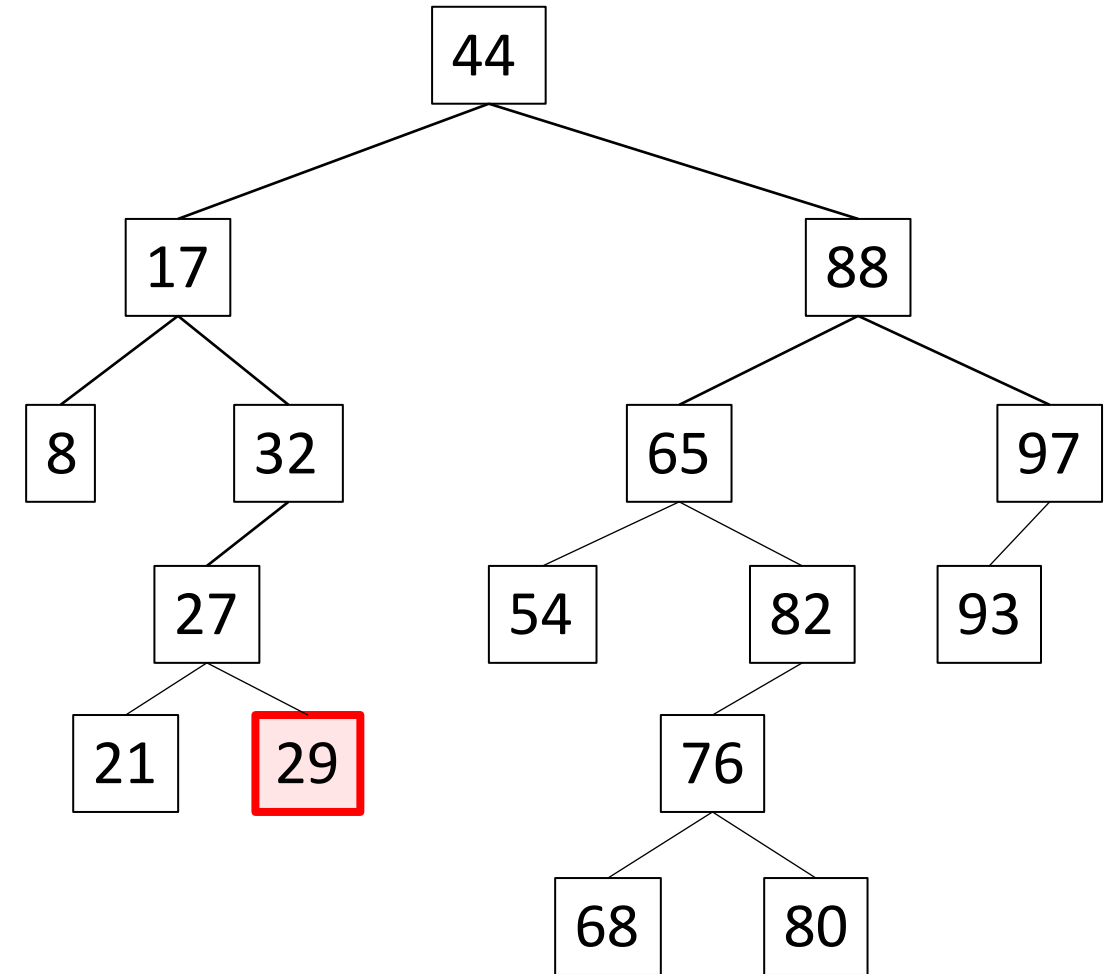
```
public void insert(int newValue) {  
    if (root == null) {  
        root = new Node(newValue);  
    } else {  
        Node currentNode = root;  
        boolean placed = false;  
        while (!placed) {  
            if (newValue < currentNode.getValue()) {  
                if (currentNode.getLeft() != null) {  
                    currentNode = currentNode.getLeft();  
                } else {  
                    }  
                } else {  
                    if (currentNode.getRight() != null) {  
                        currentNode = currentNode.getRight();  
                    } else {  
                        }  
                    }  
        }  
    }  
}
```



Binary Search Tree - Insertion

`insert(28);`

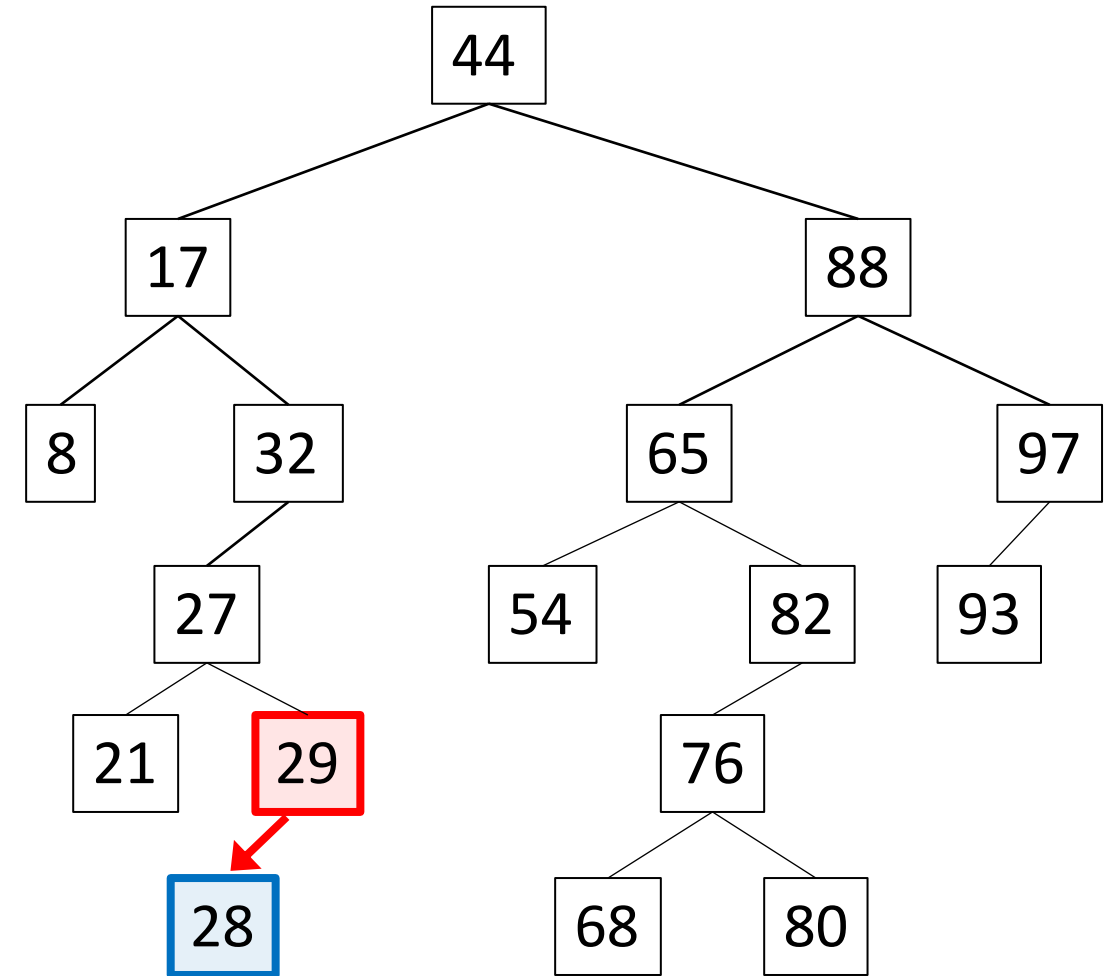
```
public void insert(int newValue) {  
    if (root == null) {  
        root = new Node(newValue);  
    } else {  
        Node currentNode = root;  
        boolean placed = false;  
        while (!placed) {  
            if (newValue < currentNode.getValue()) {  
                if (currentNode.getLeft() != null) {  
                    currentNode = currentNode.getLeft();  
                } else {  
                    // Insertion logic for left child  
                }  
            } else {  
                if (currentNode.getRight() != null) {  
                    currentNode = currentNode.getRight();  
                } else {  
                    // Insertion logic for right child  
                }  
            }  
        }  
    }  
}
```



Binary Search Tree - Insertion

insert(28);

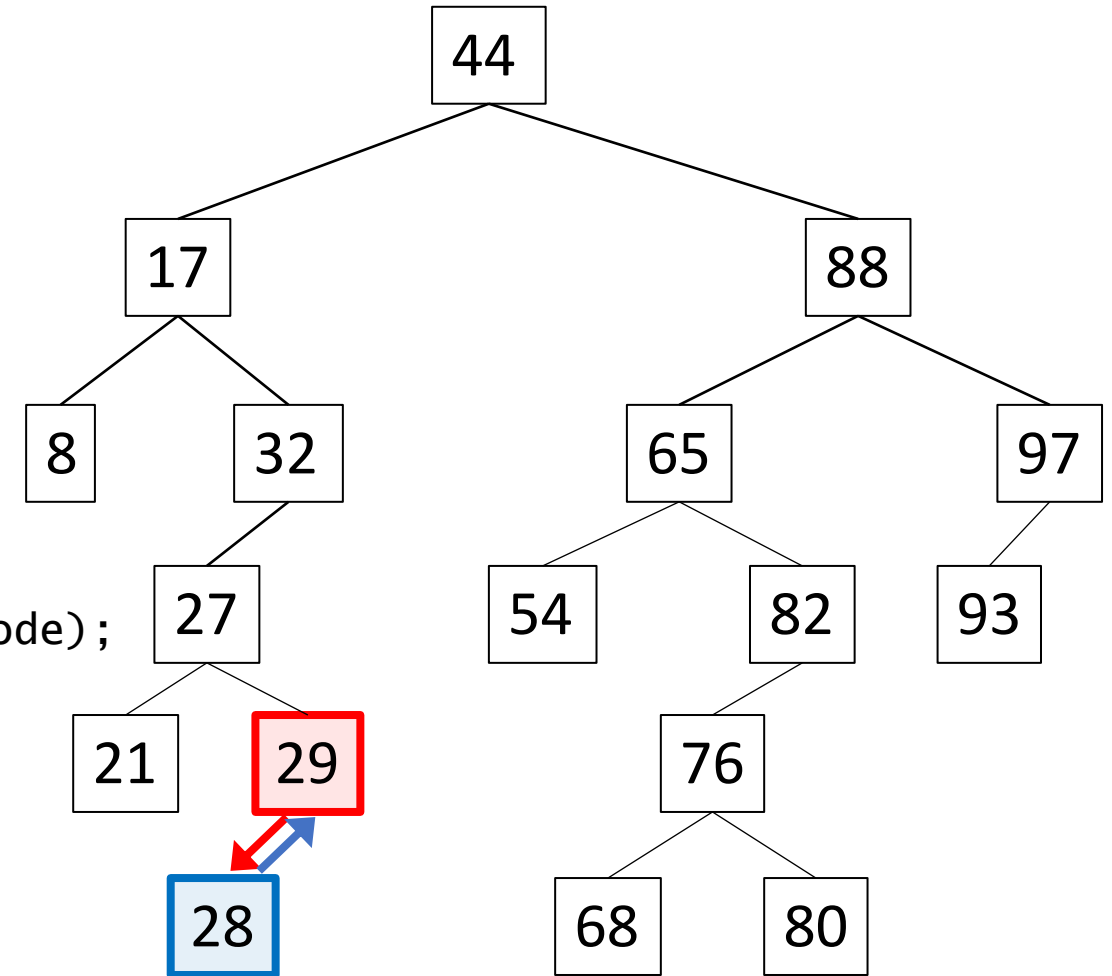
```
public void insert(int newValue) {  
    if (root == null) {  
        root = new Node(newValue);  
    } else {  
        Node currentNode = root;  
        boolean placed = false;  
        while (!placed) {  
            if (newValue < currentNode.getValue()) {  
                if (currentNode.getLeft() != null) {  
                    currentNode = currentNode.getLeft();  
                } else {  
                    currentNode.setLeft(new Node(newValue));  
                }  
            } else {  
                if (currentNode.getRight() != null) {  
                    currentNode = currentNode.getRight();  
                } else {  
                    currentNode.setRight(new Node(newValue));  
                }  
            }  
        }  
    }  
}
```



Binary Search Tree - Insertion

insert(28);

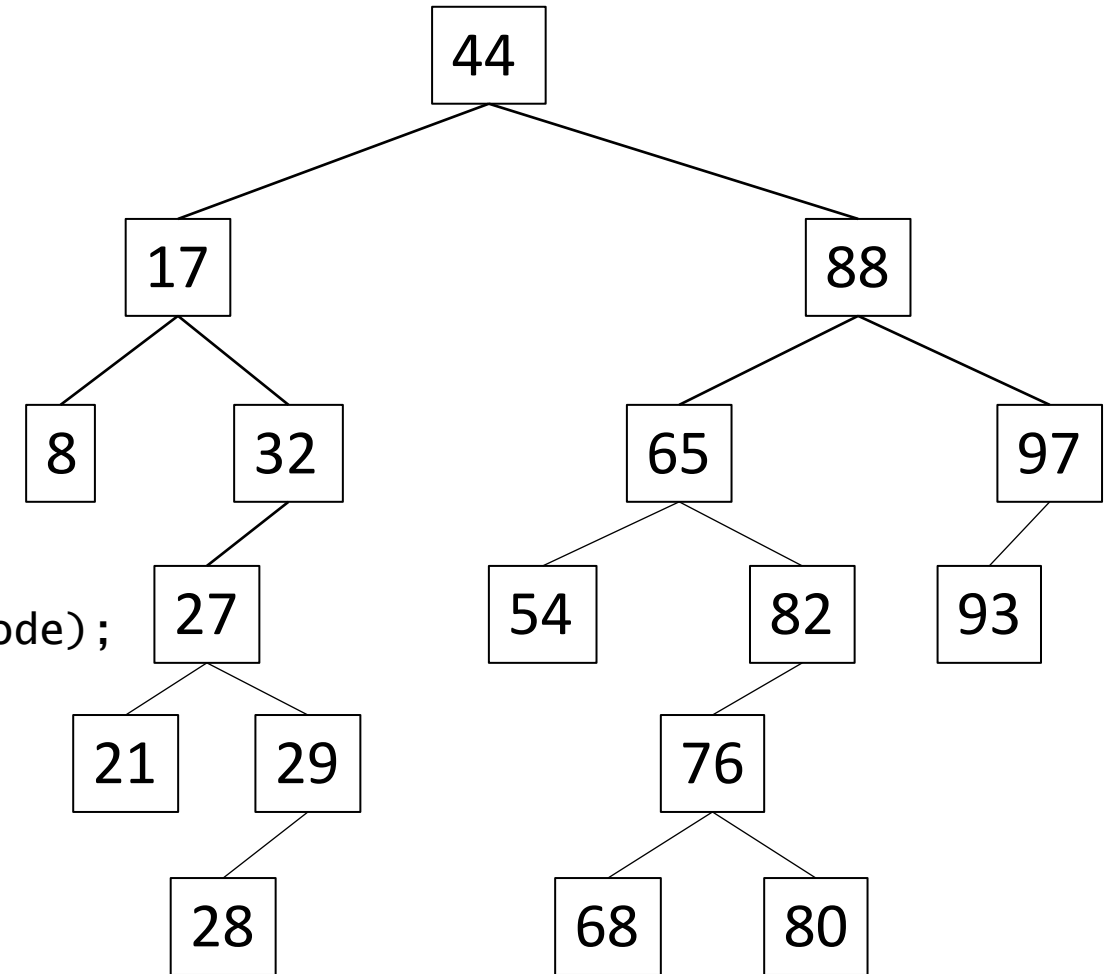
```
public void insert(int newValue) {  
    if (root == null) {  
        root = new Node(newValue);  
    } else {  
        Node currentNode = root;  
        boolean placed = false;  
        while (!placed) {  
            if (newValue < currentNode.getValue()) {  
                if (currentNode.getLeft() != null) {  
                    currentNode = currentNode.getLeft();  
                } else {  
                    currentNode.setLeft(new Node(newValue));  
                    currentNode.getLeft().setParent(currentNode);  
                }  
            } else {  
                if (currentNode.getRight() != null) {  
                    currentNode = currentNode.getRight();  
                } else {  
                    currentNode.setRight(new Node(newValue));  
                    currentNode.getRight().setParent(currentNode);  
                }  
            }  
        }  
    }  
}
```



Binary Search Tree - Insertion

insert(28);

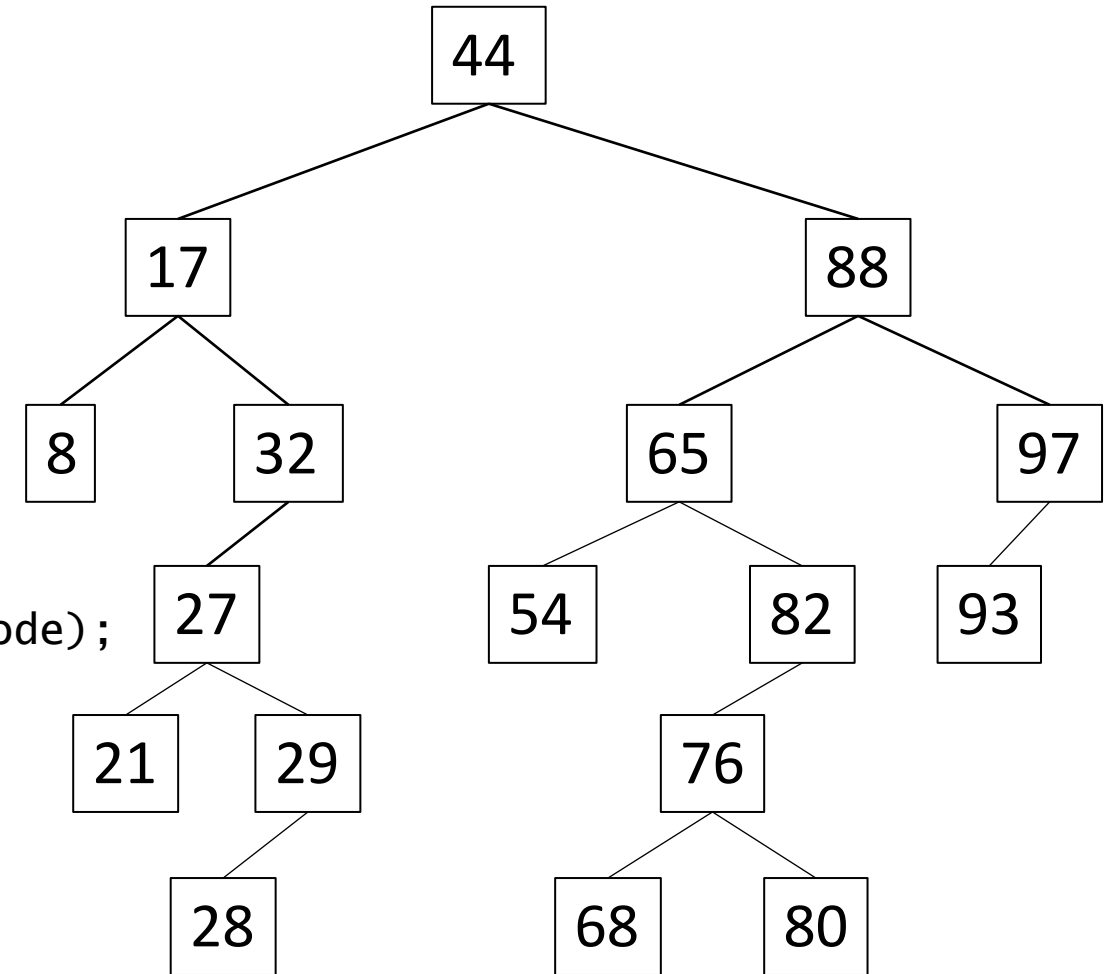
```
public void insert(int newValue) {  
    if (root == null) {  
        root = new Node(newValue);  
    } else {  
        Node currentNode = root;  
        boolean placed = false;  
        while (!placed) {  
            if (newValue < currentNode.getValue()) {  
                if (currentNode.getLeft() != null) {  
                    currentNode = currentNode.getLeft();  
                } else {  
                    currentNode.setLeft(new Node(newValue));  
                    currentNode.getLeft().setParent(currentNode);  
                    placed = true;  
                }  
            } else {  
                if (currentNode.getRight() != null) {  
                    currentNode = currentNode.getRight();  
                } else {  
                    currentNode.setRight(new Node(newValue));  
                    currentNode.getRight().setParent(currentNode);  
                    placed = true;  
                }  
            }  
        }  
    }  
}
```



Binary Search Tree - Insertion

insert(28);

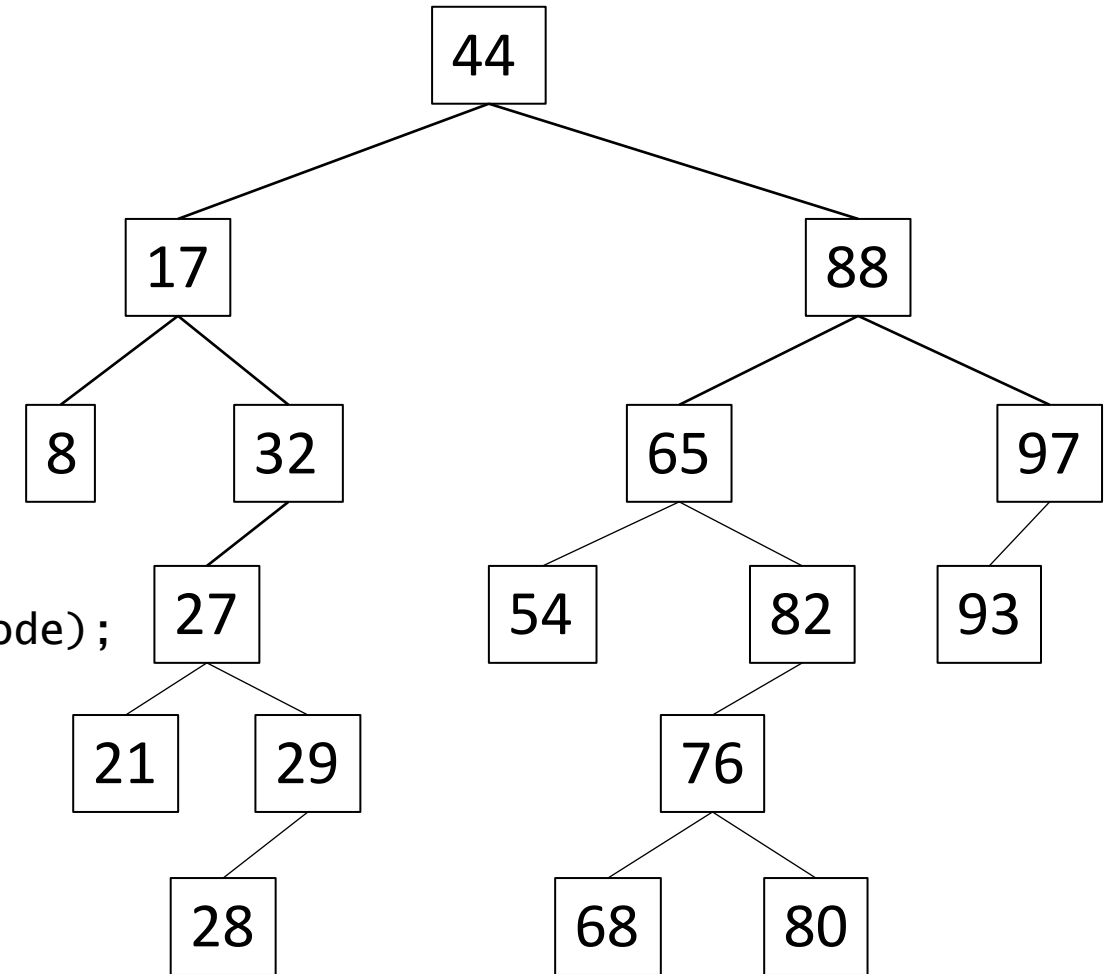
```
public void insert(int newValue) {  
    if (root == null) {  
        root = new Node(newValue);  
    } else {  
        Node currentNode = root;  
        boolean placed = false;  
        while (!placed) {  
            if (newValue < currentNode.getValue()) {  
                if (currentNode.getLeft() != null) {  
                    currentNode = currentNode.getLeft();  
                } else {  
                    currentNode.setLeft(new Node(newValue));  
                    currentNode.getLeft().setParent(currentNode);  
                    placed = true;  
                }  
            } else {  
                if (currentNode.getRight() != null) {  
                    currentNode = currentNode.getRight();  
                } else {  
                    currentNode.setRight(new Node(newValue));  
                    currentNode.getRight().setParent(currentNode);  
                    placed = true;  
                }  
            }  
        }  
    }  
}
```



Binary Search Tree - Insertion

insert(28);

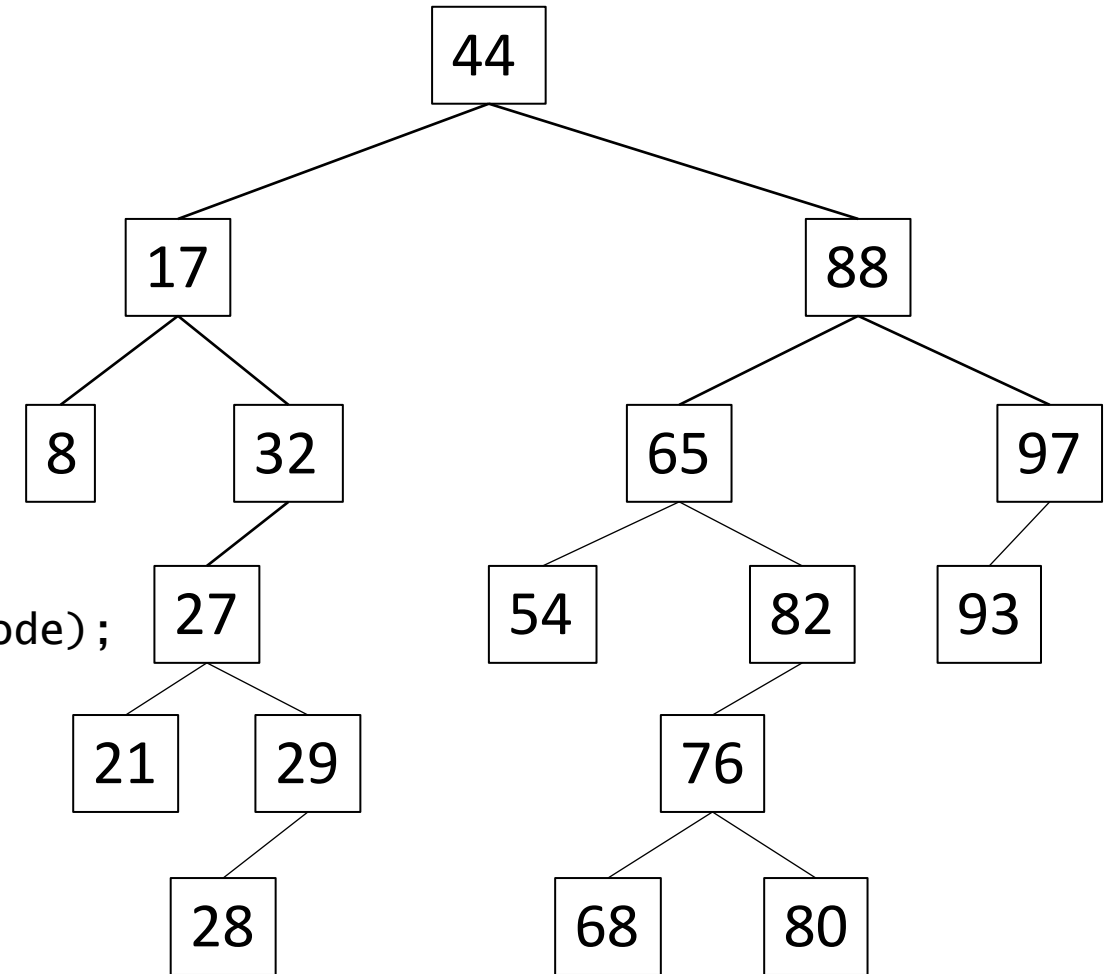
```
public void insert(int newValue) {  
    if (root == null) {  
        root = new Node(newValue);  
    } else {  
        Node currentNode = root;  
        boolean placed = false;  
        while (!placed) {  
            if (newValue < currentNode.getValue()) {  
                if (currentNode.getLeft() != null) {  
                    currentNode = currentNode.getLeft();  
                } else {  
                    currentNode.setLeft(new Node(newValue));  
                    currentNode.getLeft().setParent(currentNode);  
                    placed = true;  
                }  
            } else {  
                if (currentNode.getRight() != null) {  
                    currentNode = currentNode.getRight();  
                } else {  
                    currentNode.setRight(new Node(newValue));  
                }  
            }  
        }  
    }  
}
```



Binary Search Tree - Insertion

insert(28);

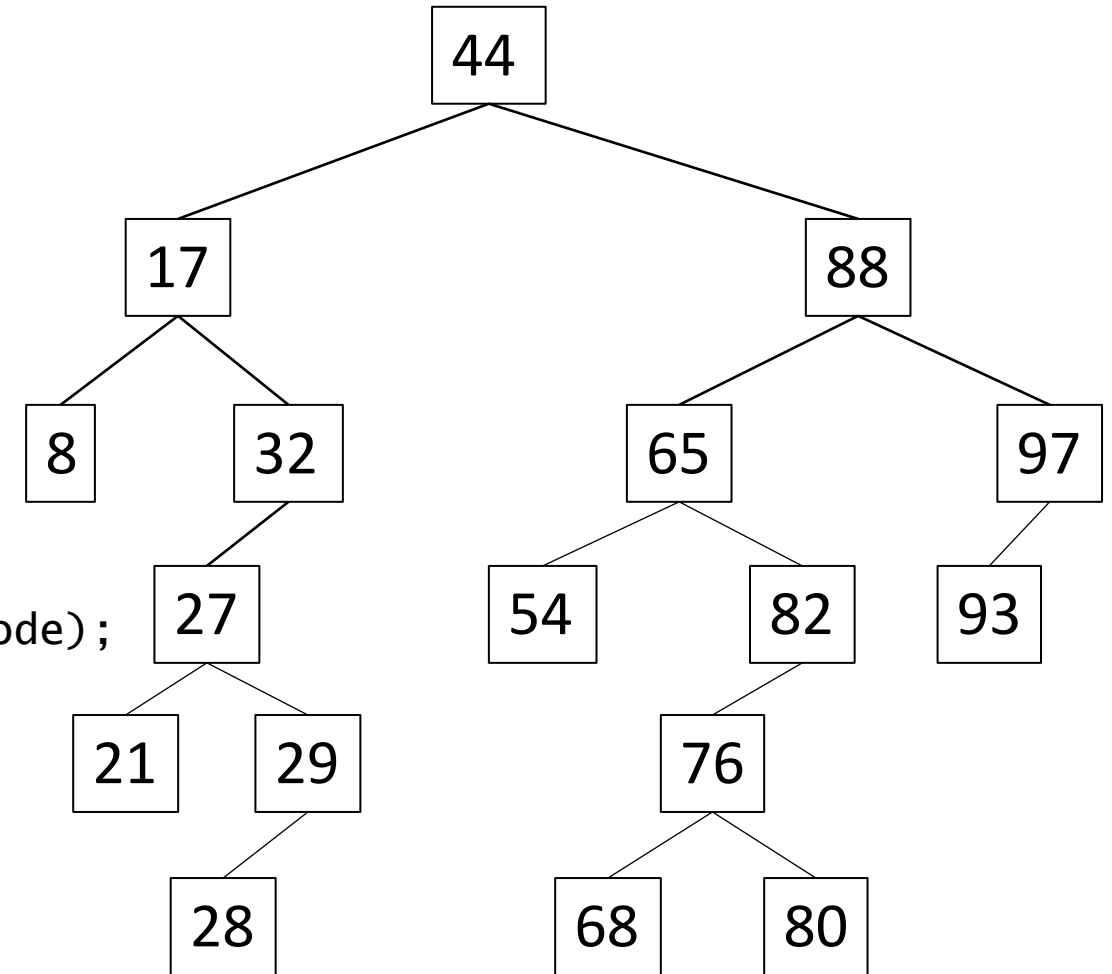
```
public void insert(int newValue) {  
    if (root == null) {  
        root = new Node(newValue);  
    } else {  
        Node currentNode = root;  
        boolean placed = false;  
        while (!placed) {  
            if (newValue < currentNode.getValue()) {  
                if (currentNode.getLeft() != null) {  
                    currentNode = currentNode.getLeft();  
                } else {  
                    currentNode.setLeft(new Node(newValue));  
                    currentNode.getLeft().setParent(currentNode);  
                    placed = true;  
                }  
            } else {  
                if (currentNode.getRight() != null) {  
                    currentNode = currentNode.getRight();  
                } else {  
                    currentNode.setRight(new Node(newValue));  
                    currentNode.getRight().setParent(currentNode);  
                    placed = true;  
                }  
            }  
        }  
    }  
}
```



Binary Search Tree - Insertion

insert(28);

```
public void insert(int newValue) {  
    if (root == null) {  
        root = new Node(newValue);  
    } else {  
        Node currentNode = root;  
        boolean placed = false;  
        while (!placed) {  
            if (newValue < currentNode.getValue()) {  
                if (currentNode.getLeft() != null) {  
                    currentNode = currentNode.getLeft();  
                } else {  
                    currentNode.setLeft(new Node(newValue));  
                    currentNode.getLeft().setParent(currentNode);  
                    placed = true;  
                }  
            } else {  
                if (currentNode.getRight() != null) {  
                    currentNode = currentNode.getRight();  
                } else {  
                    currentNode.setRight(new Node(newValue));  
                    currentNode.getRight().setParent(currentNode);  
                    placed = true;  
                }  
            }  
        }  
    }  
}
```

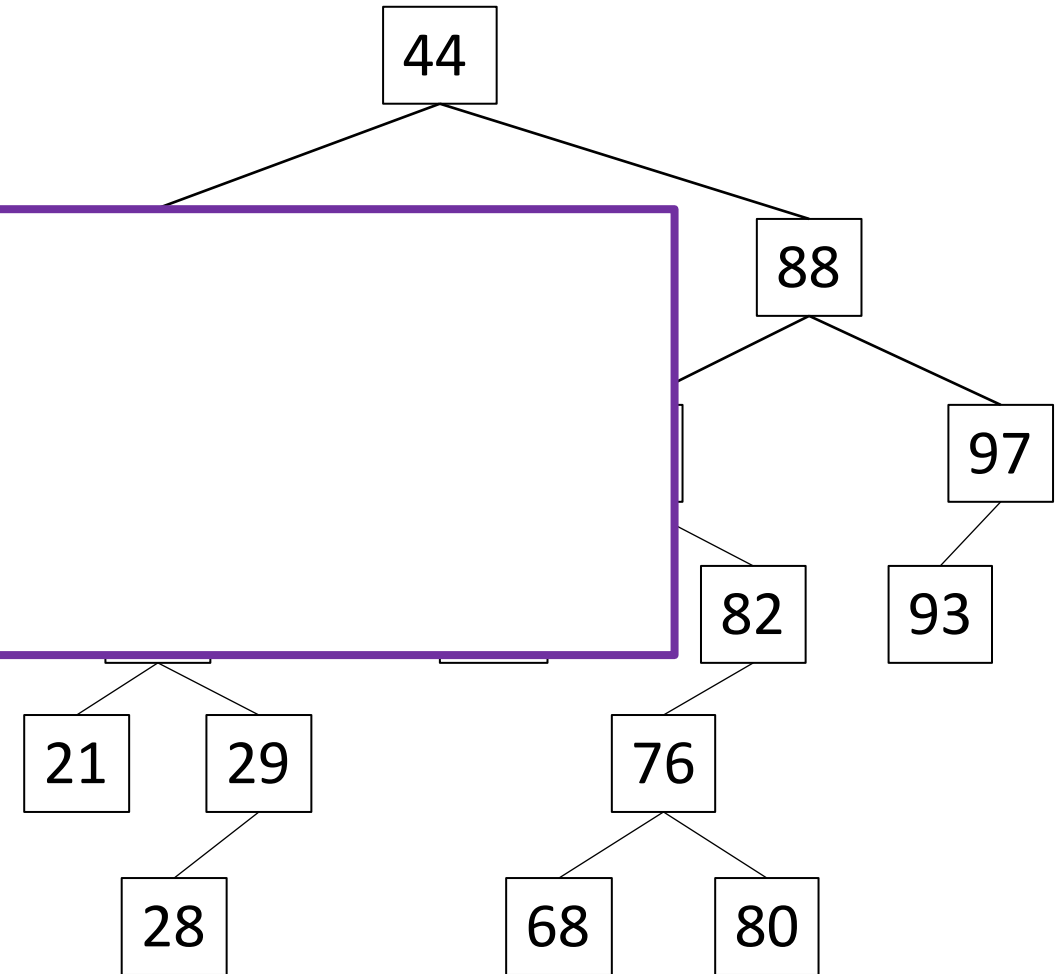


Binary Search Tree - Insertion

insert(28);

```
public void insert(int newValue) {  
    if (root == null) {  
        root = new Node(newValue);  
    } else {  
        Node currentNode = root;  
        boolean placed = false;  
        while (!placed) {  
            if (newValue < currentNode.getValue()) {  
                if (currentNode.getLeft() == null) {  
                    currentNode.setLeft(new Node(newValue));  
                    currentNode.getLeft().setParent(currentNode);  
                    placed = true;  
                }  
            } else {  
                if (currentNode.getRight() != null) {  
                    currentNode = currentNode.getRight();  
                } else {  
                    currentNode.setRight(new Node(newValue));  
                    currentNode.getRight().setParent(currentNode);  
                    placed = true;  
                }  
            }  
        }  
    }  
}
```

Running time?

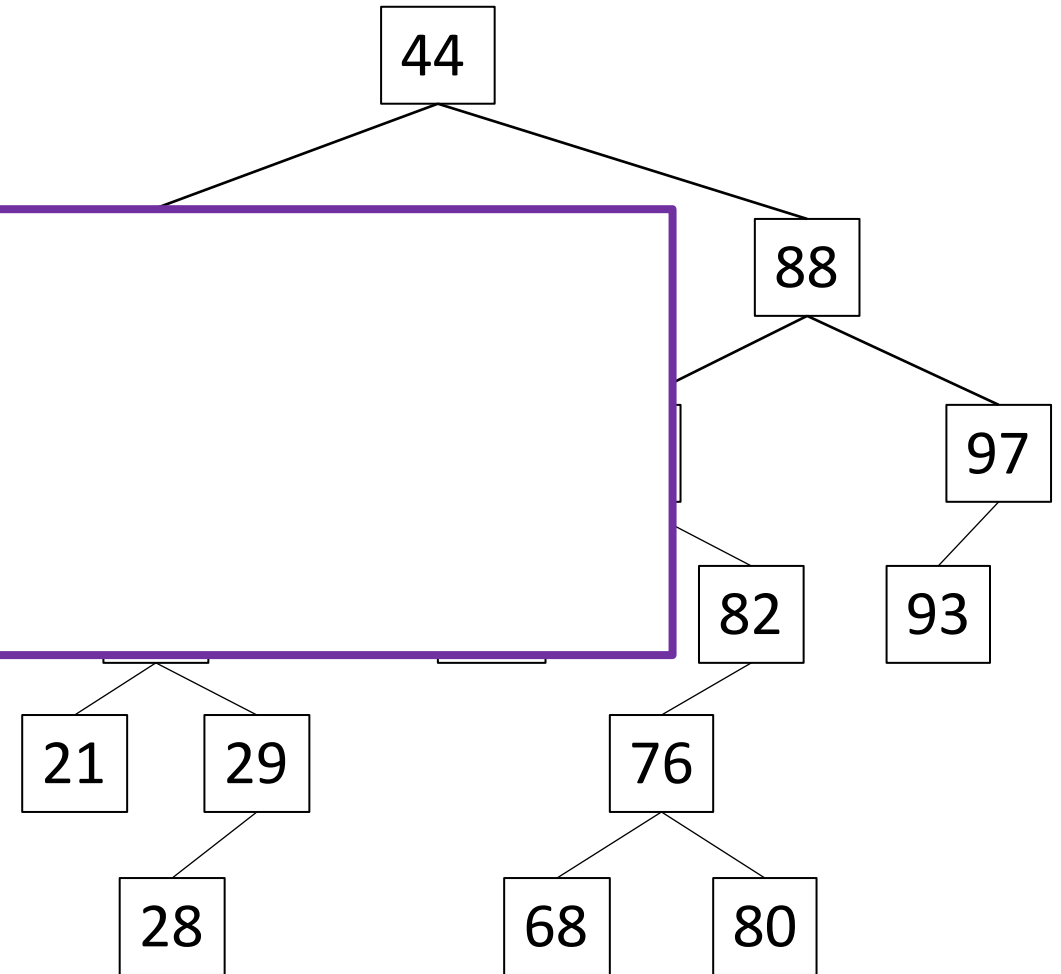


insert(28);

Binary Search Tree - Insertion

```
public void insert(int newValue) {  
    if (root == null) {  
        root = new Node(newValue);  
    } else {  
        Node currentNode = root;  
        boolean placed = false;  
        while (!placed) {  
            if (newValue < currentNode.getValue()) {  
                if (currentNode.getLeft() == null) {  
                    currentNode.setLeft(new Node(newValue));  
                    currentNode.getLeft().setParent(currentNode);  
                    placed = true;  
                }  
            } else {  
                if (currentNode.getRight() != null) {  
                    currentNode = currentNode.getRight();  
                } else {  
                    currentNode.setRight(new Node(newValue));  
                    currentNode.getRight().setParent(currentNode);  
                    placed = true;  
                }  
            }  
        }  
    }  
}
```

Running time?
“Bad” tree?

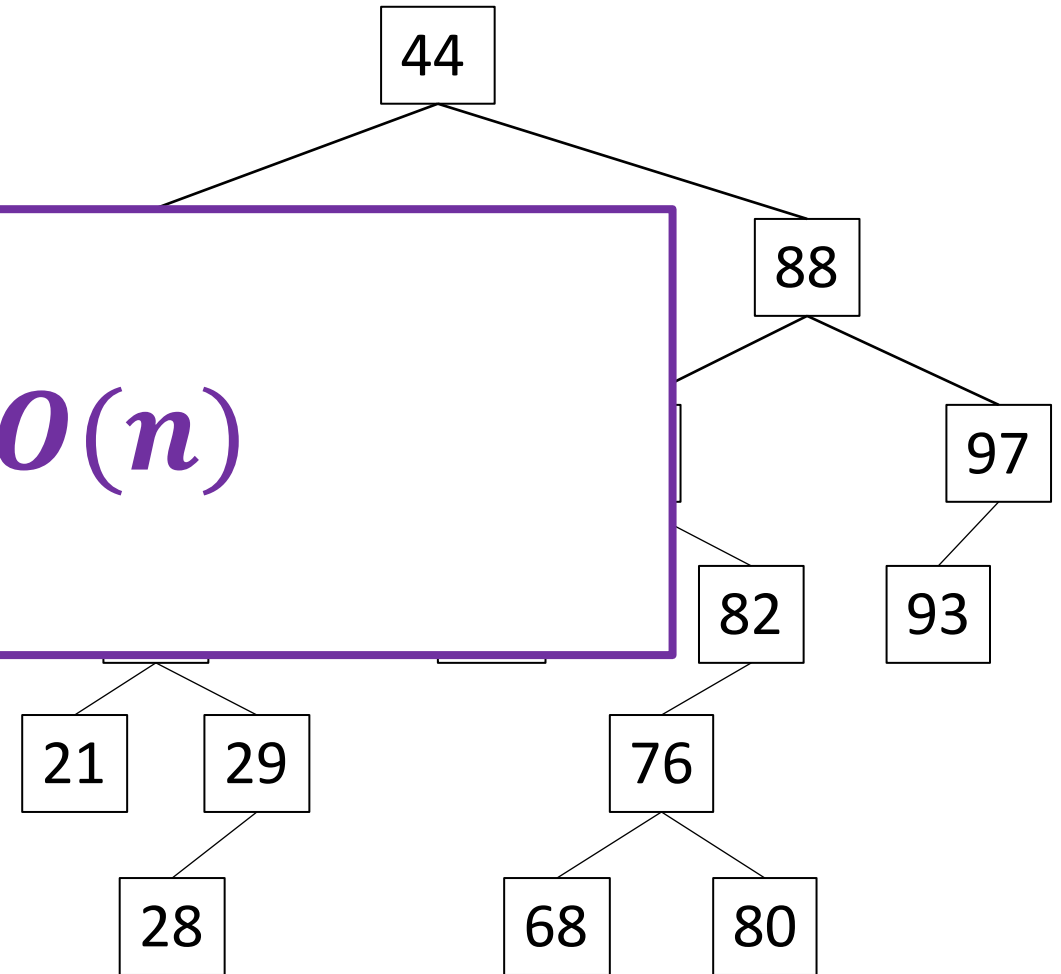


insert(28);

Binary Search Tree - Insertion

```
public void insert(int newValue) {  
    if (root == null) {  
        root = new Node(newValue);  
    } else {  
        Node currentNode = root;  
        boolean placed = false;  
        while (!placed) {  
            if (newValue < currentNode.getValue()) {  
                if (currentNode.getLeft() == null) {  
                    currentNode.setLeft(new Node(newValue));  
                    currentNode.getLeft().setParent(currentNode);  
                    placed = true;  
                }  
            } else {  
                if (currentNode.getRight() != null) {  
                    currentNode = currentNode.getRight();  
                } else {  
                    currentNode.setRight(new Node(newValue));  
                    currentNode.getRight().setParent(currentNode);  
                    placed = true;  
                }  
            }  
        }  
    }  
}
```

Running time?
“Bad” tree? $O(n)$



insert(28);

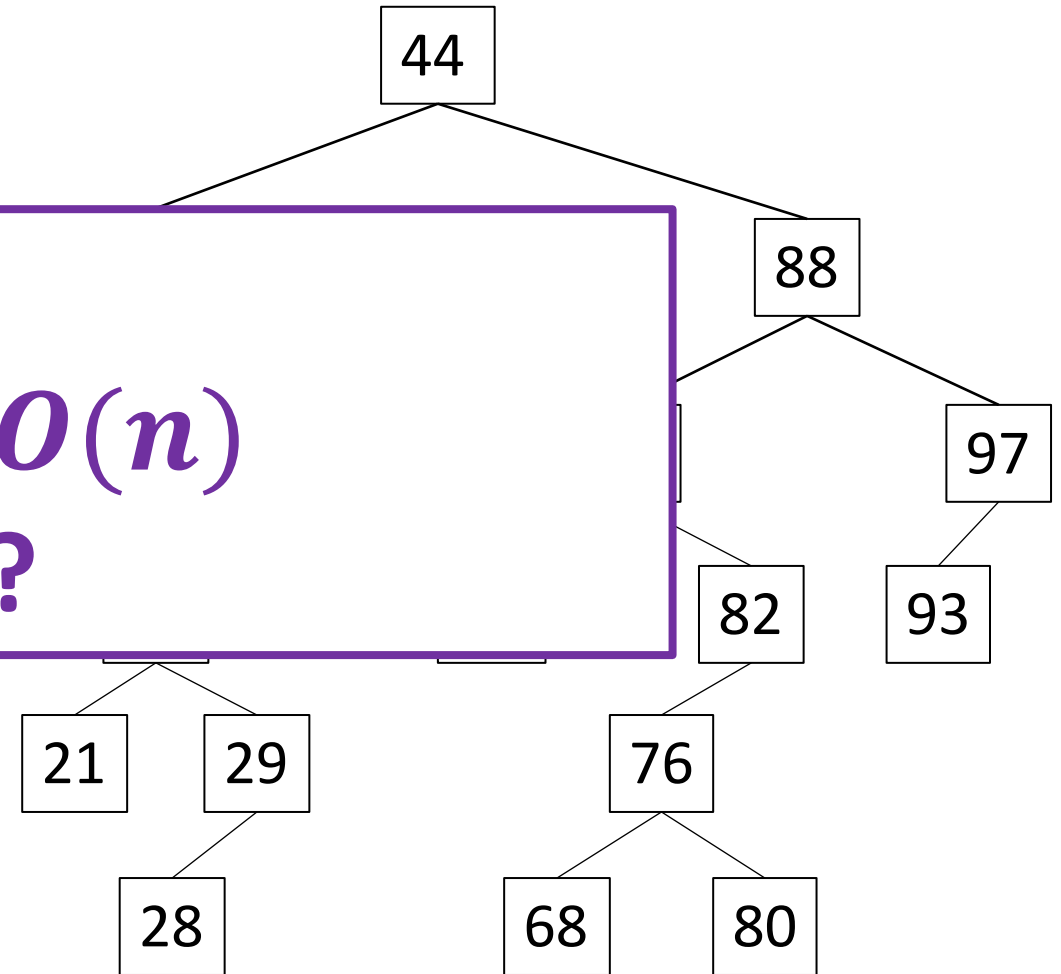
Binary Search Tree - Insertion

```
public void insert(int newValue) {  
    if (root == null) {  
        root = new Node(newValue);  
    } else {  
        Node currentNode = root;  
        boolean placed = false;  
        while (!placed) {  
            if (newValue < currentNode.getValue()) {  
                if (currentNode.getLeft() == null) {  
                    currentNode.setLeft(new Node(newValue));  
                    currentNode.getLeft().setParent(currentNode);  
                    placed = true;  
                }  
            } else {  
                if (currentNode.getRight() != null) {  
                    currentNode = currentNode.getRight();  
                } else {  
                    currentNode.setRight(new Node(newValue));  
                    currentNode.getRight().setParent(currentNode);  
                    placed = true;  
                }  
            }  
        }  
    }  
}
```

Running time?

“Bad” tree? $O(n)$

“Good” tree?



insert(28);

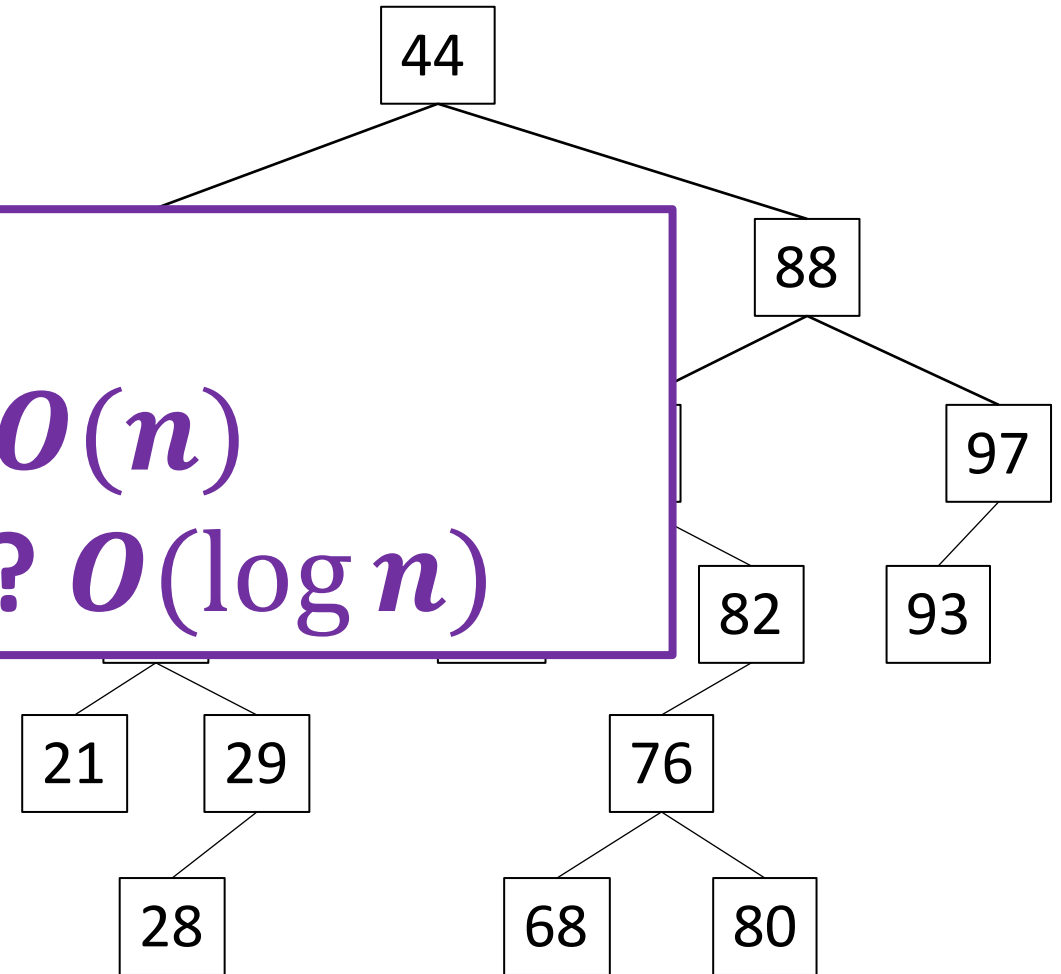
Binary Search Tree - Insertion

```
public void insert(int newValue) {  
    if (root == null) {  
        root = new Node(newValue);  
    } else {  
        Node currentNode = root;  
        boolean placed = false;  
        while (!placed) {  
            if (newValue < currentNode.getValue()) {  
                if (currentNode.getLeft() == null) {  
                    currentNode.setLeft(new Node(newValue));  
                    currentNode.getLeft().setParent(currentNode);  
                    placed = true;  
                }  
            } else {  
                if (currentNode.getRight() != null) {  
                    currentNode = currentNode.getRight();  
                } else {  
                    currentNode.setRight(new Node(newValue));  
                    currentNode.getRight().setParent(currentNode);  
                    placed = true;  
                }  
            }  
        }  
    }  
}
```

Running time?

“Bad” tree? $O(n)$

“Good” tree? $O(\log n)$



Binary Search Tree - Insertion

insert(28);

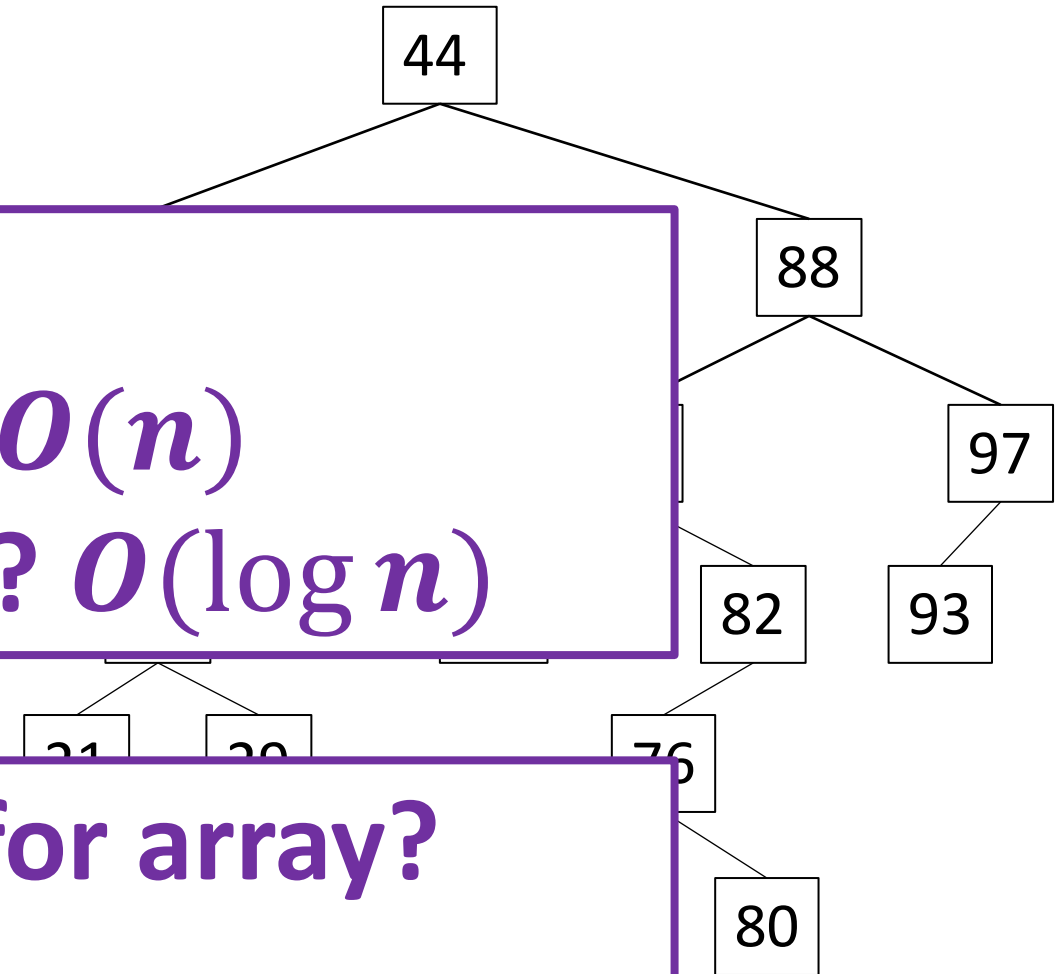
```
public void insert(int newValue) {  
    if (root == null) {  
        root = new Node(newValue);  
    } else {  
        Node currentNode = root;  
        boolean placed = false;  
        while (!placed) {  
            if (newValue < currentNode.value) {  
                if (currentNode.left == null) {  
                    currentNode.left = new Node(newValue);  
                    placed = true;  
                } else {  
                    currentNode = currentNode.left;  
                }  
            } else {  
                if (currentNode.right == null) {  
                    currentNode.right = new Node(newValue);  
                    placed = true;  
                } else {  
                    currentNode = currentNode.right;  
                }  
            }  
        }  
    }  
}
```

Running time?

“Bad” tree? $O(n)$

“Good” tree? $O(\log n)$

Running time for array?



insert(28);

Binary Search Tree - Insertion

```
public void insert(int newValue) {  
    if (root == null) {  
        root = new Node(newValue);  
    } else {  
        Node currentNode = root;  
        boolean placed = false;  
        while (!placed) {  
            if (newValue < currentNode.value) {  
                if (currentNode.left == null) {  
                    currentNode.left = new Node(newValue);  
                    placed = true;  
                } else {  
                    currentNode = currentNode.left;  
                }  
            } else {  
                if (currentNode.right == null) {  
                    currentNode.right = new Node(newValue);  
                    placed = true;  
                } else {  
                    currentNode = currentNode.right;  
                }  
            }  
        }  
    }  
}
```

Running time?

“Bad” tree? $O(n)$

“Good” tree? $O(\log n)$

Running time for array?

$O(n)$

