CSCI 232:
Data Structures and Algorithms

Binary Search Trees (Part 1)

Reese Pearsall

Summer 2025

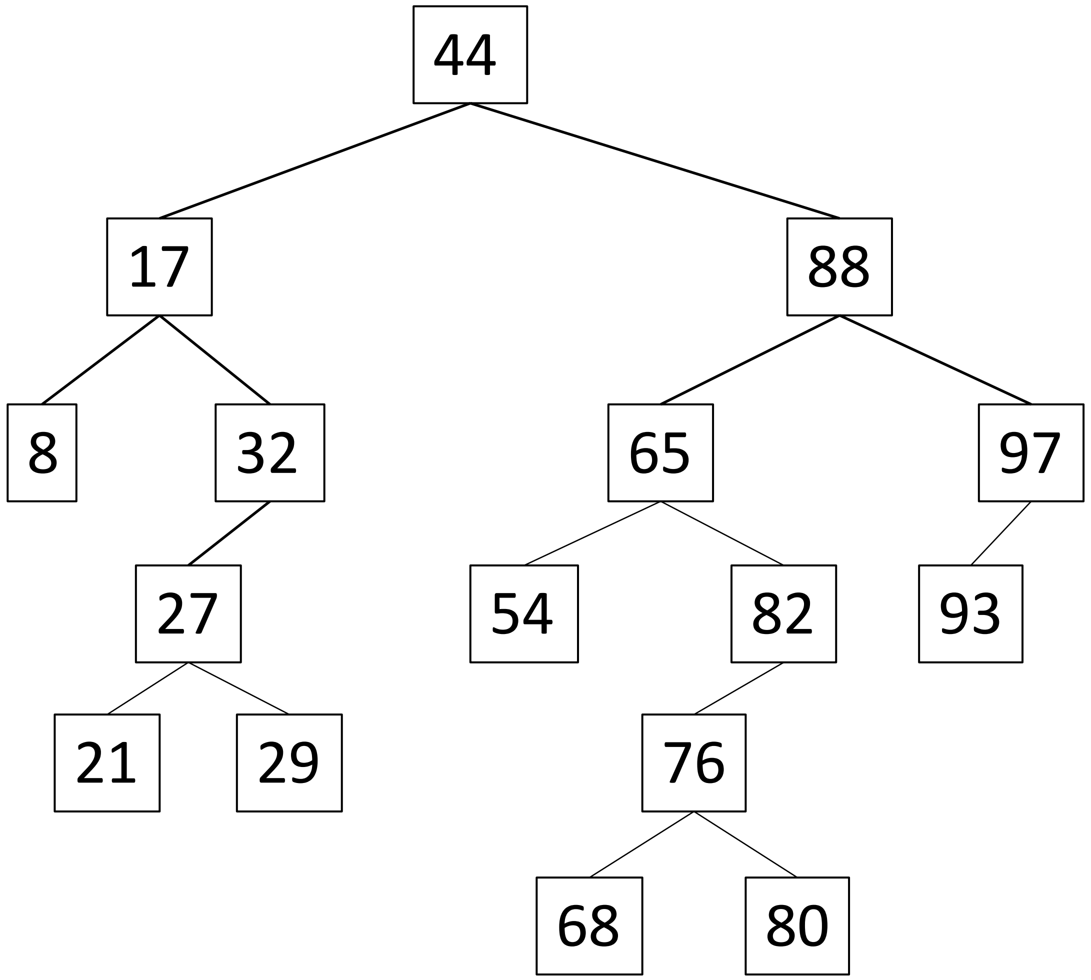
Announcements

Lab 2 due tonight at 11:59 PM

Lab 3 due Sunday at 11:59 PM

Program 1 posted. Due Wednesday June 4th

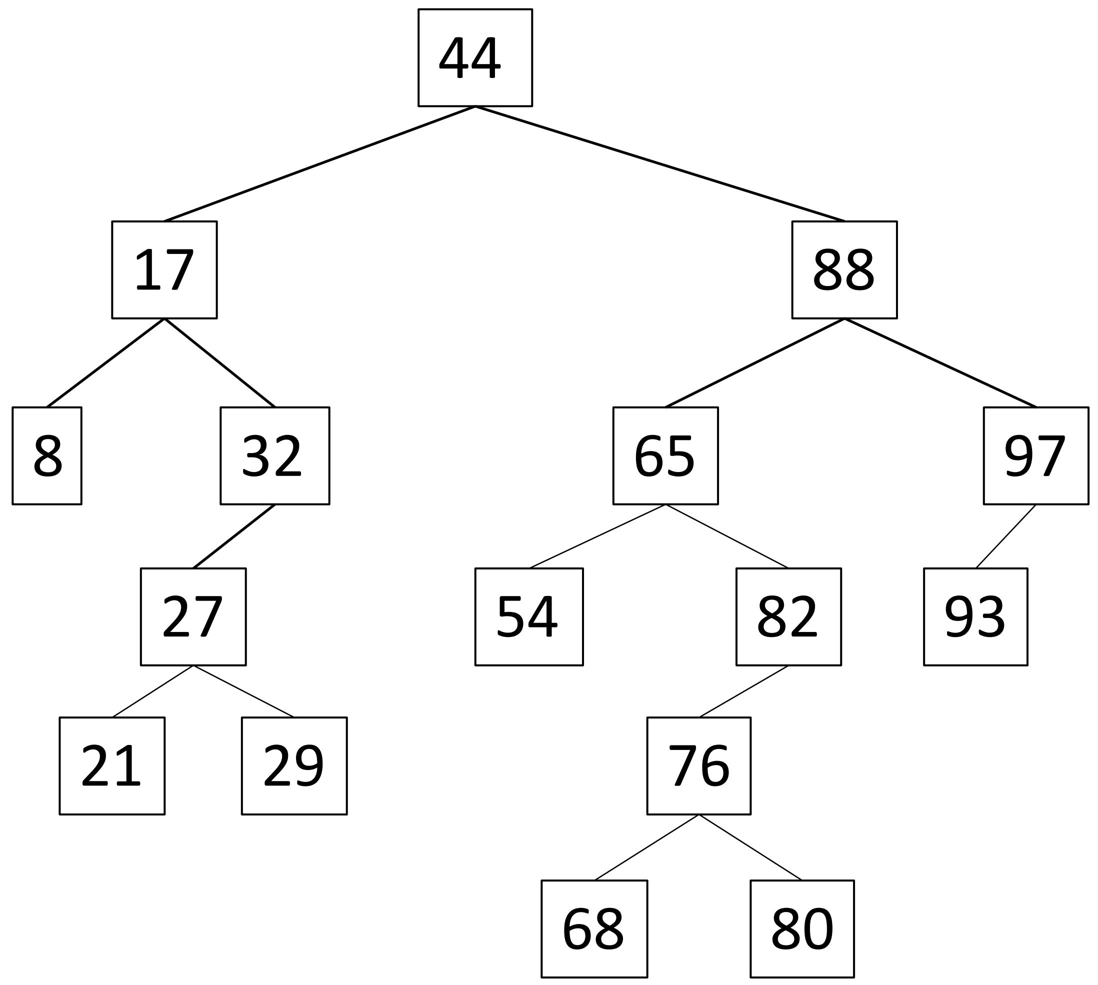
Binary Search Tree



Binary Search Tree

Binary Search Tree (BST) properties:

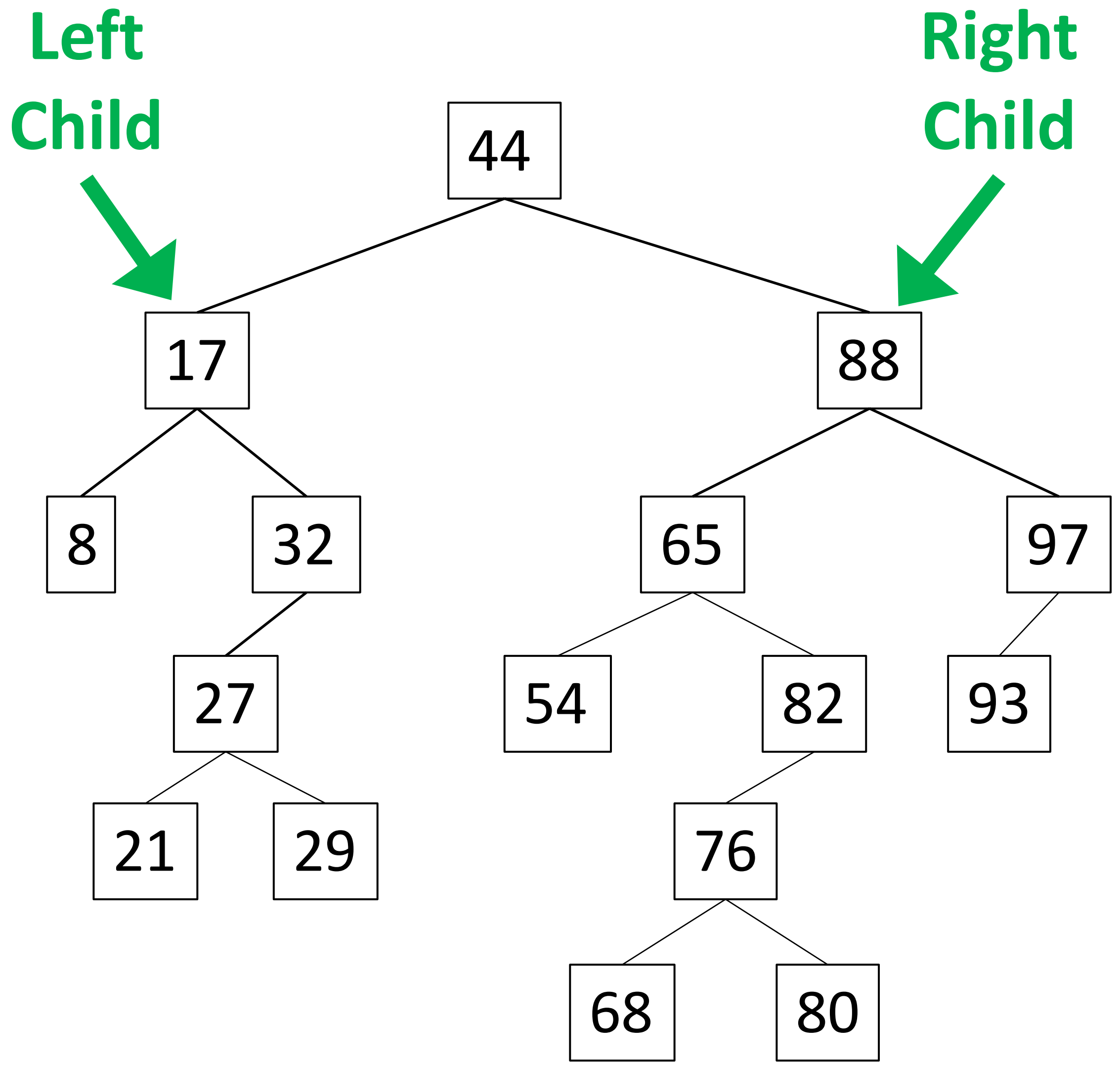
- A BST is composed of `Comparable` data elements.



Binary Search Tree

Binary Search Tree (BST) properties:

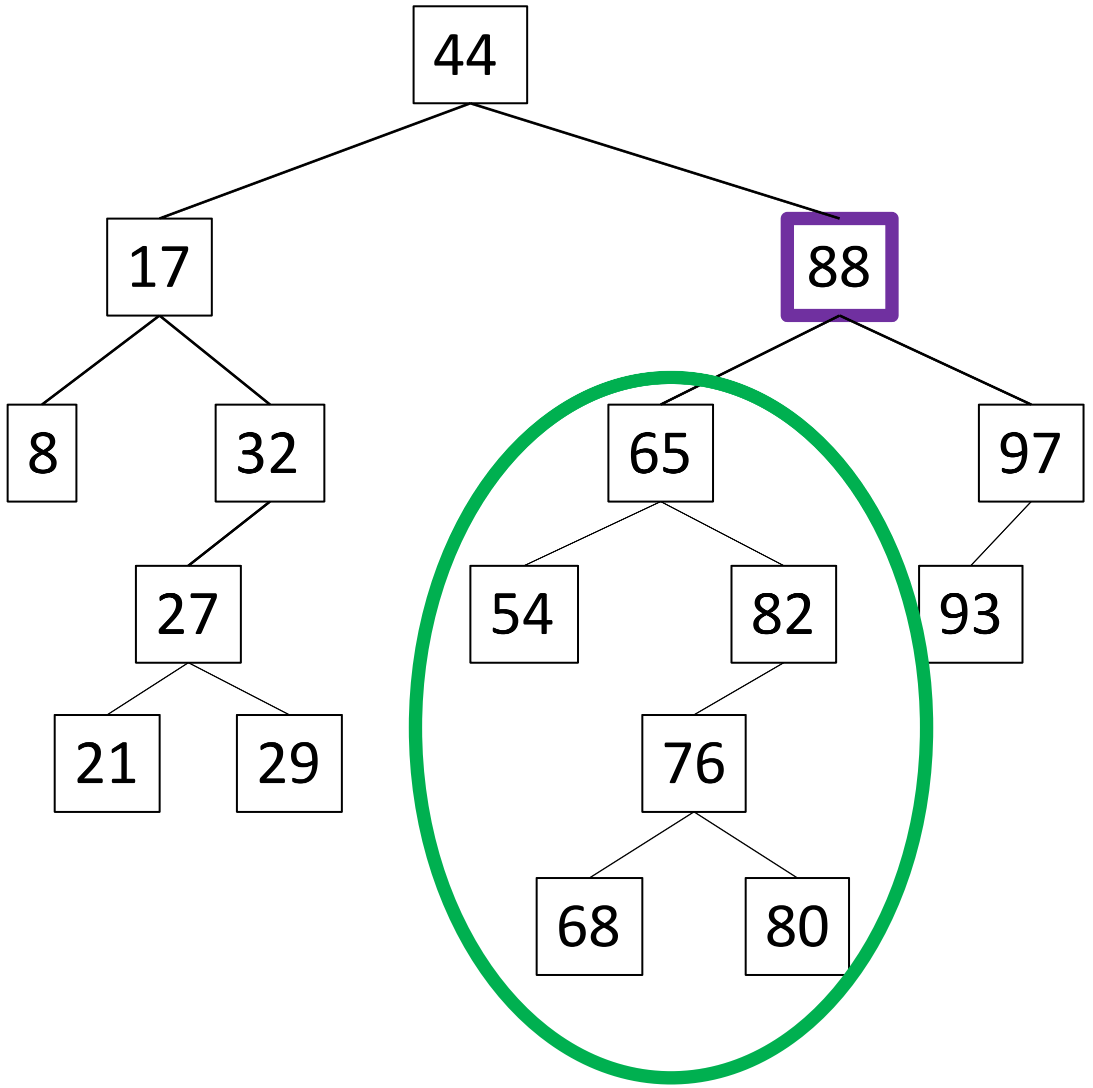
- A BST is composed of `Comparable` data elements.
- A BST is a binary tree (each node has at most two children).



Binary Search Tree

Binary Search Tree (BST) properties:

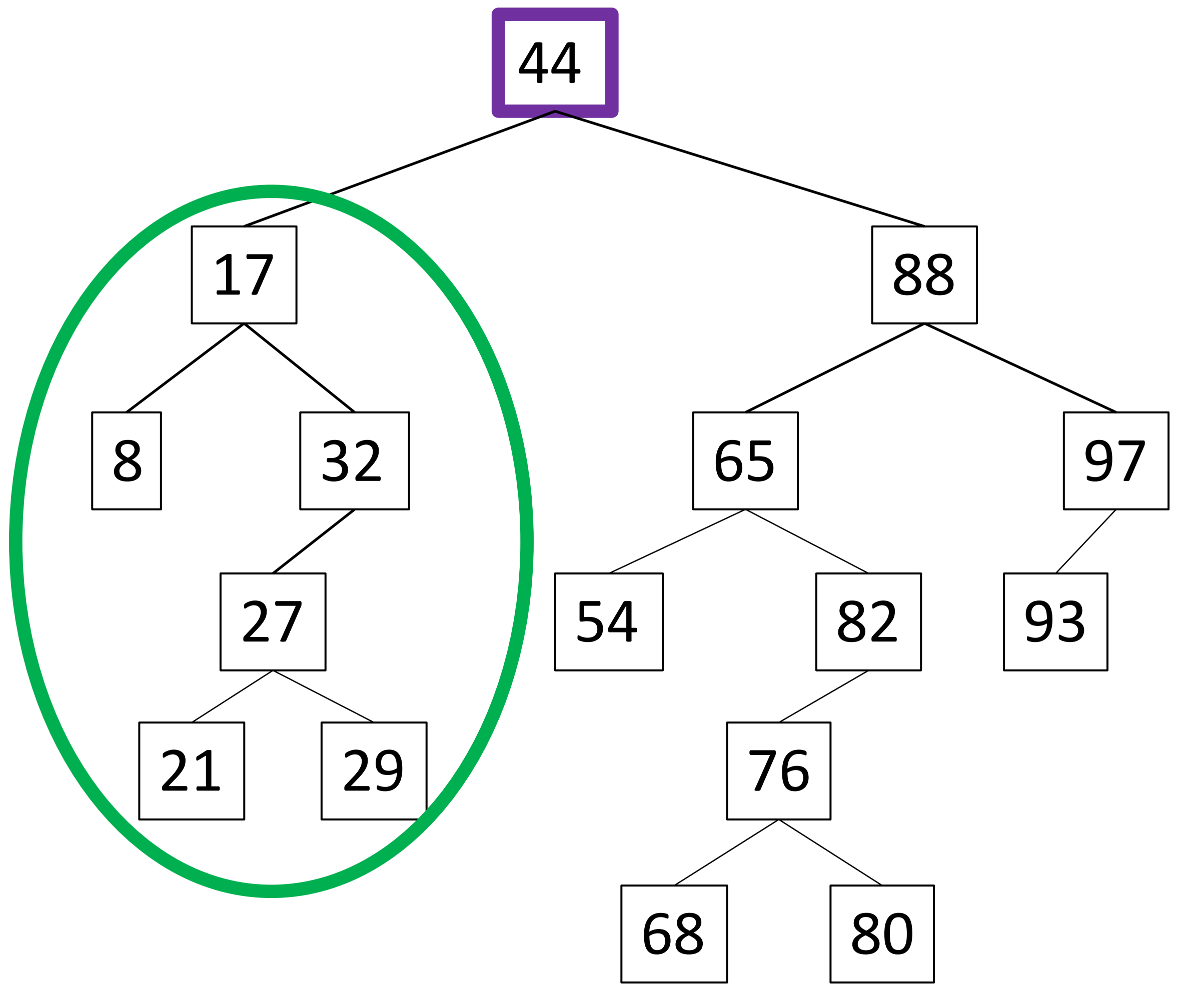
- A BST is composed of `Comparable` data elements.
- A BST is a binary tree (each node has at most two children).
- For each **node**, all **left-hand descendants** have values that are less that the node.



Binary Search Tree

Binary Search Tree (BST) properties:

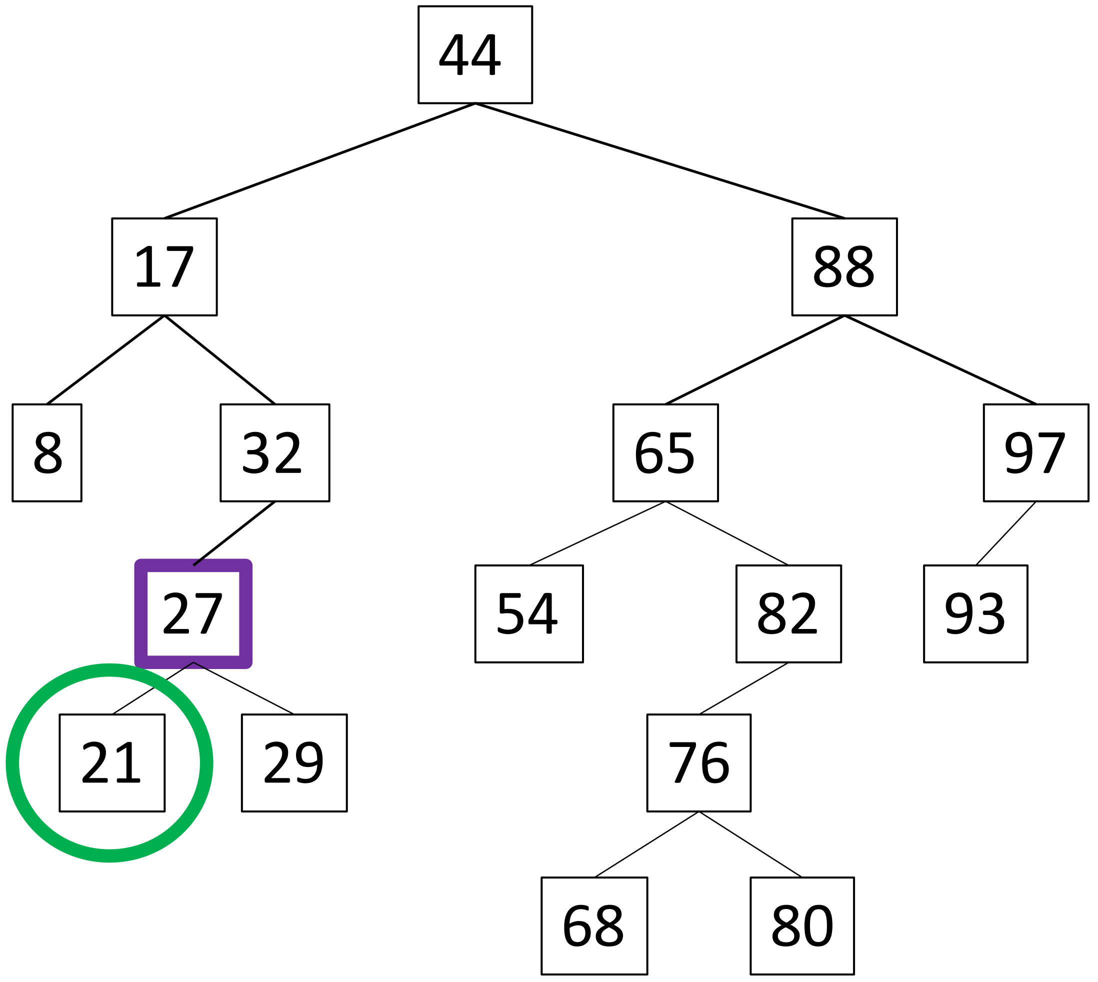
- A BST is composed of `Comparable` data elements.
- A BST is a binary tree (each node has at most two children).
- For each **node**, all **left-hand descendants** have values that are less that the node.



Binary Search Tree

Binary Search Tree (BST) properties:

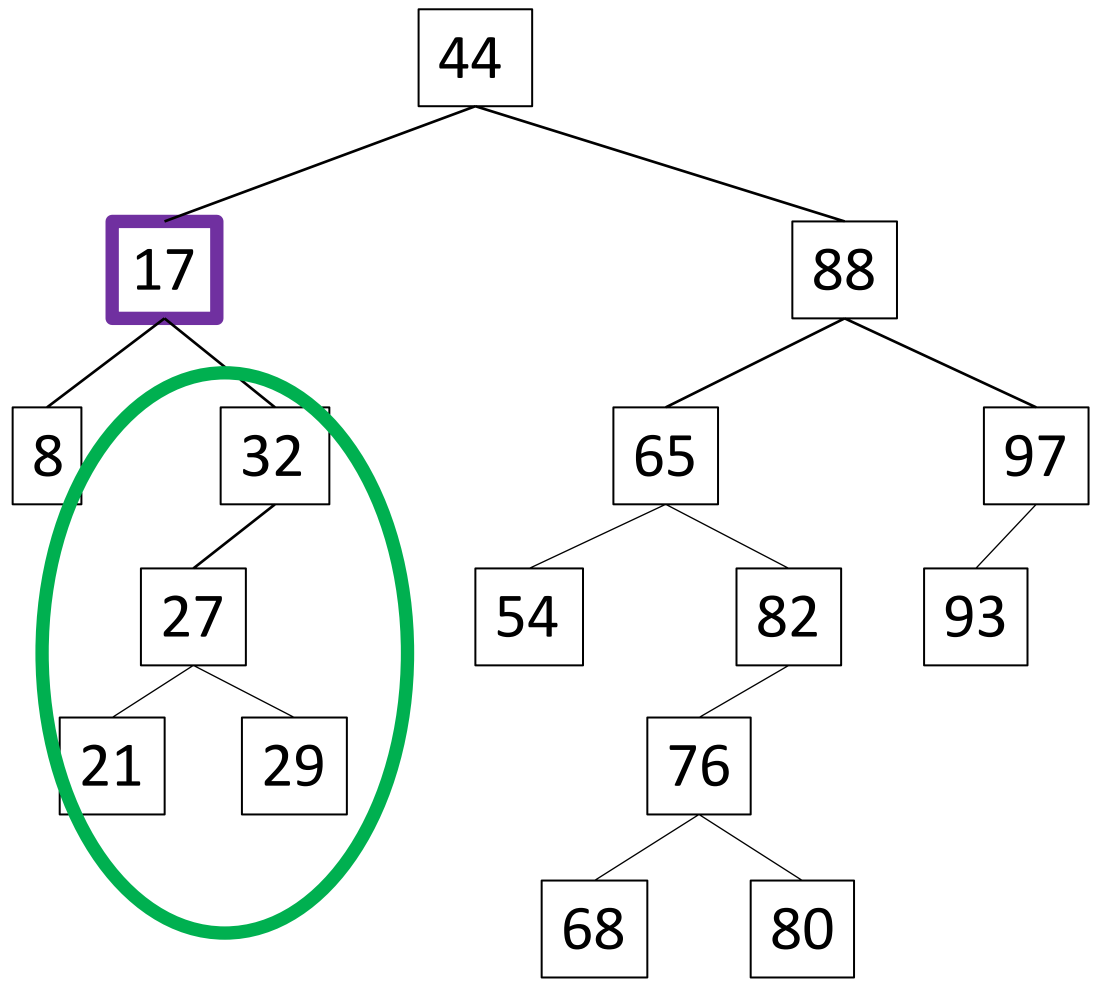
- A BST is composed of `Comparable` data elements.
- A BST is a binary tree (each node has at most two children).
- For each **node**, all **left-hand descendants** have values that are less that the node.



Binary Search Tree

Binary Search Tree (BST) properties:

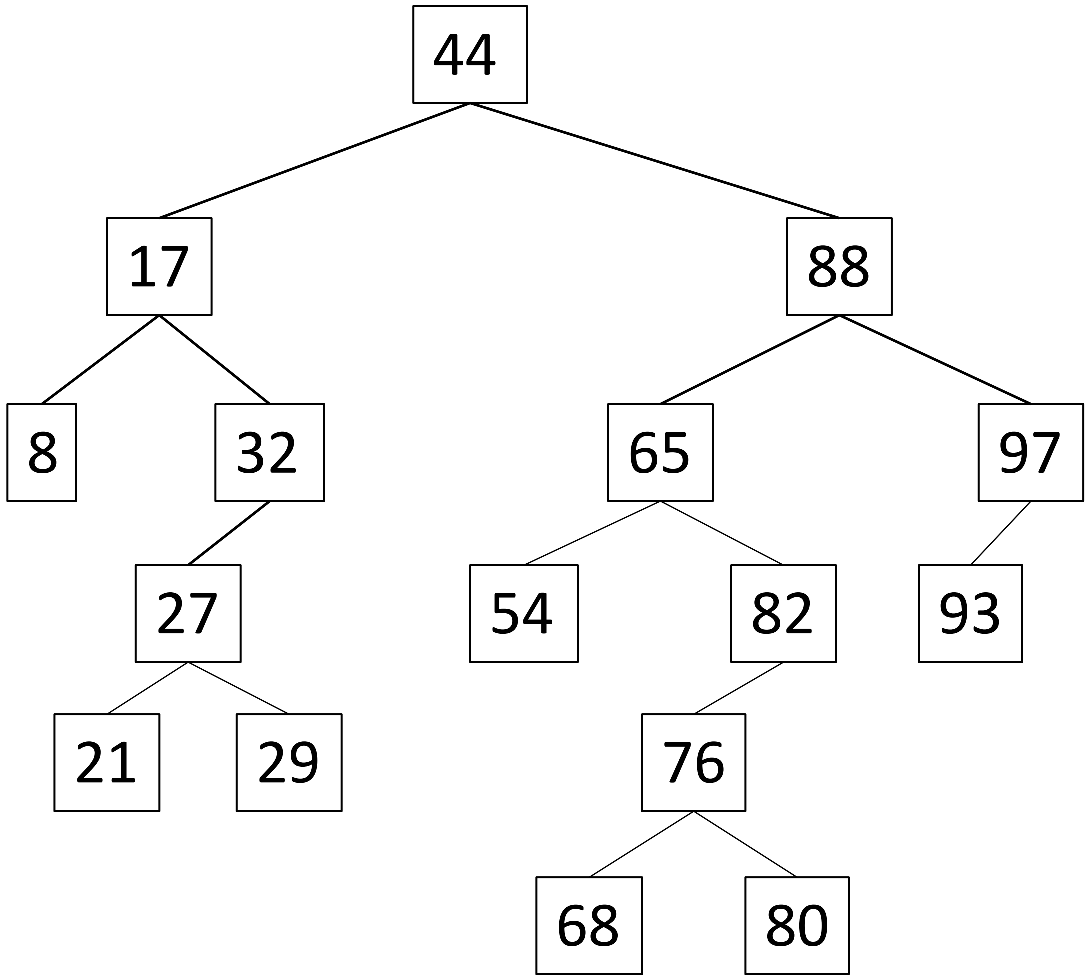
- A BST is composed of `Comparable` data elements.
- A BST is a binary tree (each node has at most two children).
- For each node, all left-hand descendants have values that are less that the node.
- For each node, all right-hand descendants have values that are larger than the node.



Binary Search Tree

Binary Search Tree (BST) properties:

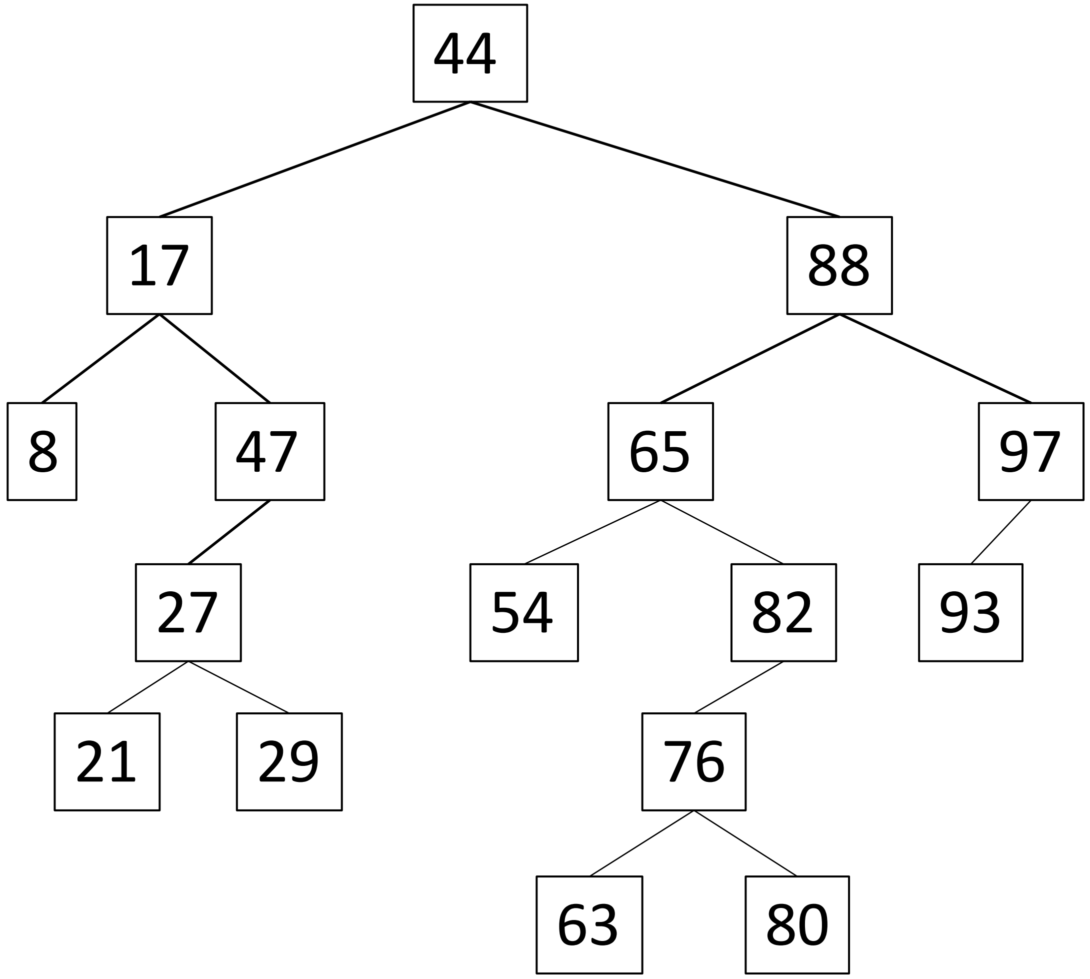
- A BST is composed of `Comparable` data elements.
- A BST is a binary tree (each node has at most two children).
- For each node, all left-hand descendants have values that are less that the node.
- For each node, all right-hand descendants have values that are larger than the node.
- There are no duplicate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

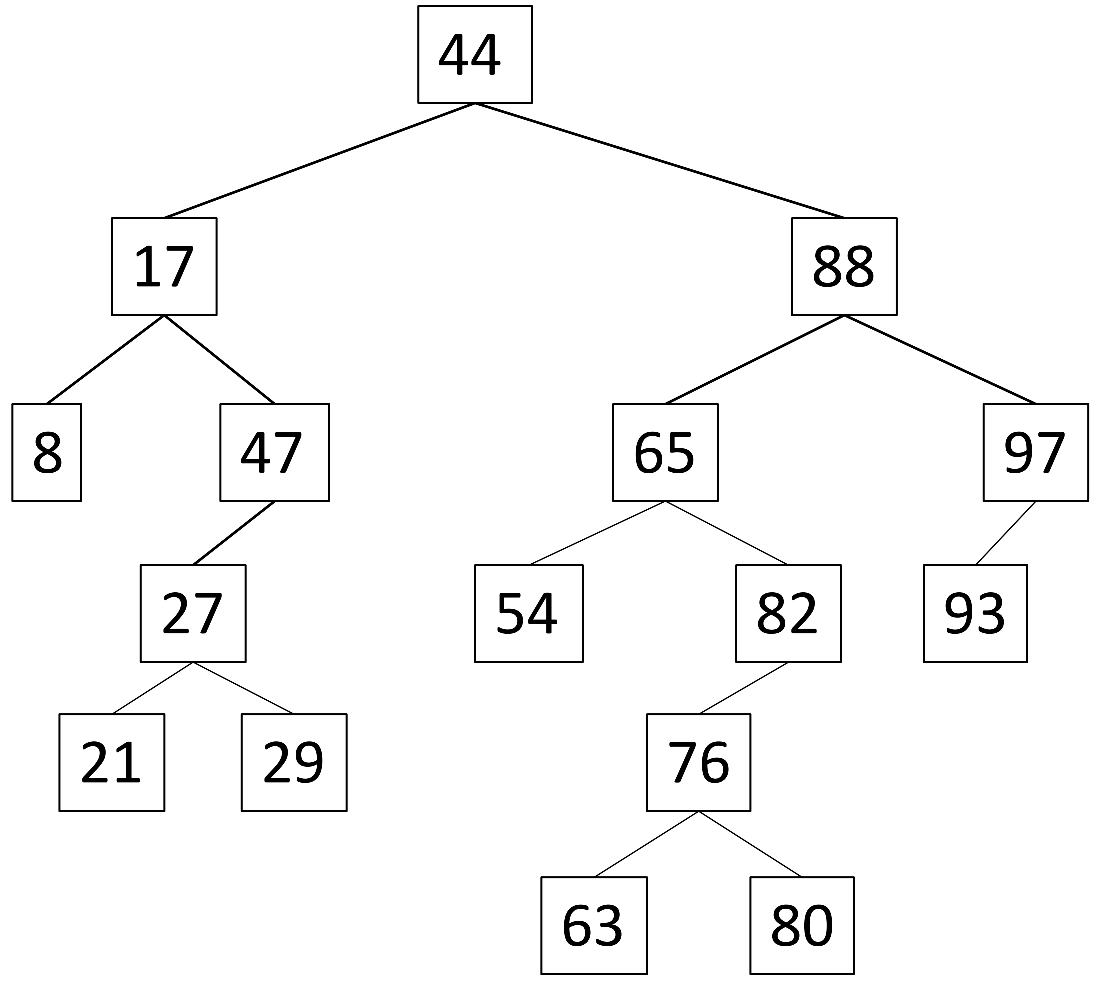


Is it a BST?

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 that the node.
- For each node, all right-hand descendants have values that are larger than the node.
- There are no duplicate values (definitions vary).

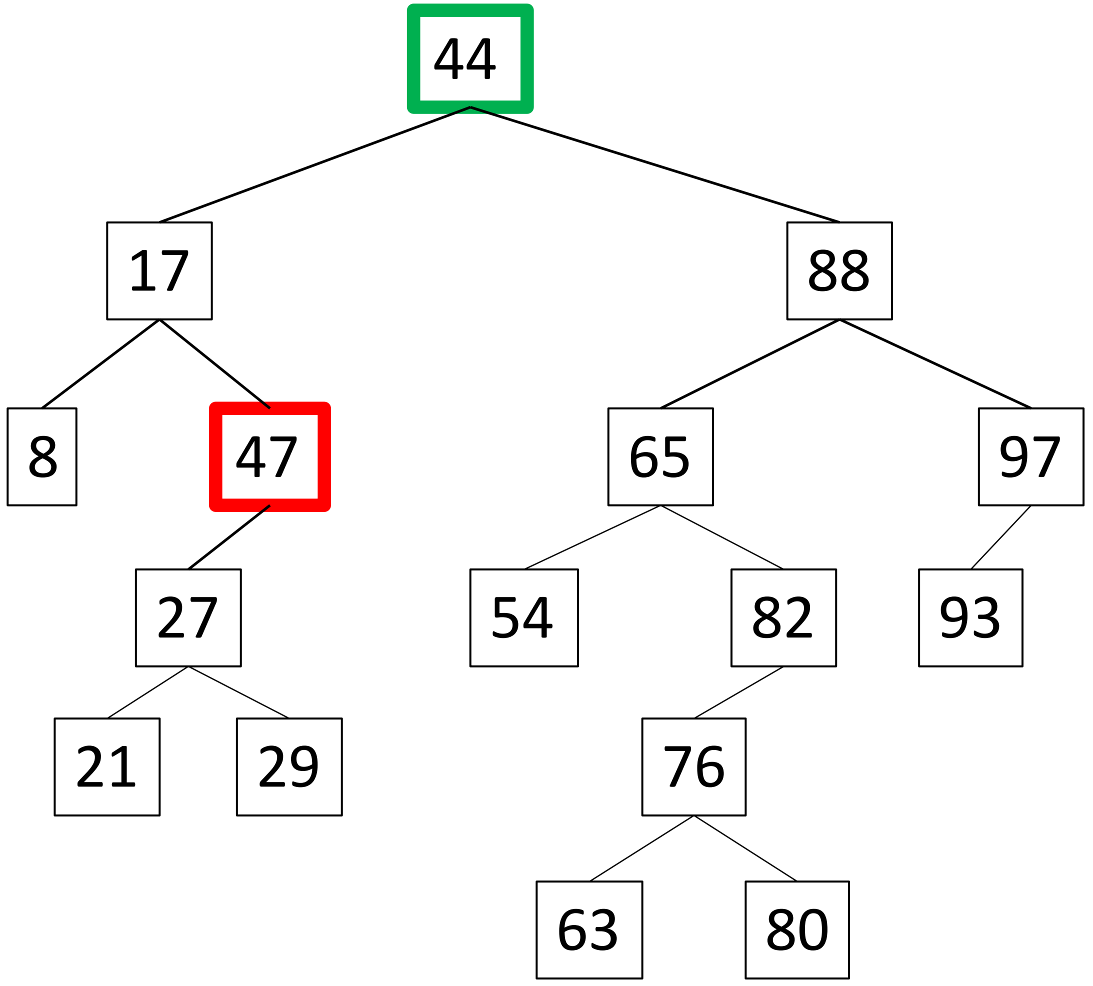


Is it a BST?

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 that the node.**
- For each node, all right-hand descendants have values that are larger than the node.
- There are no duplicate values (definitions vary).

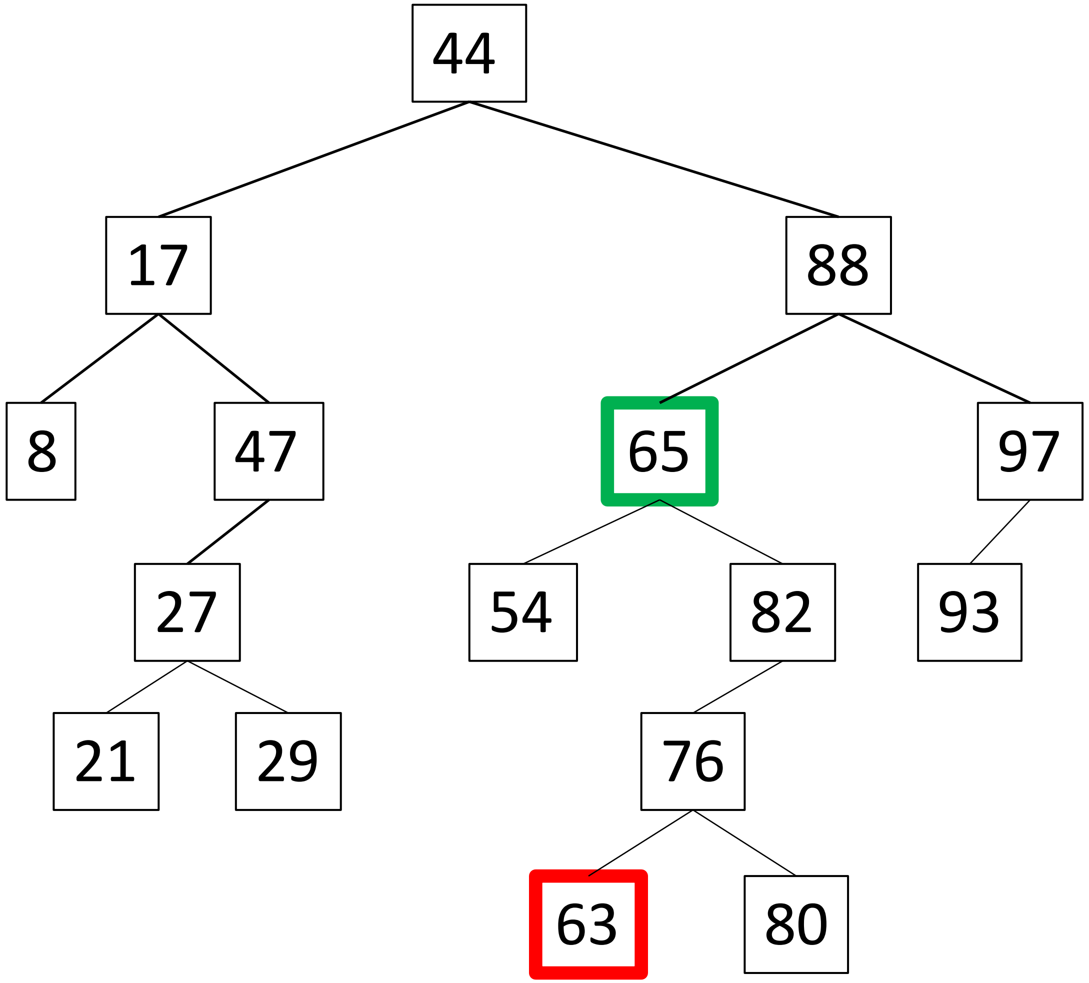


Is it a BST?

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 that the node.
- For each node, all right-hand descendants have values that are larger than the node.
- There are no duplicate values (definitions vary).

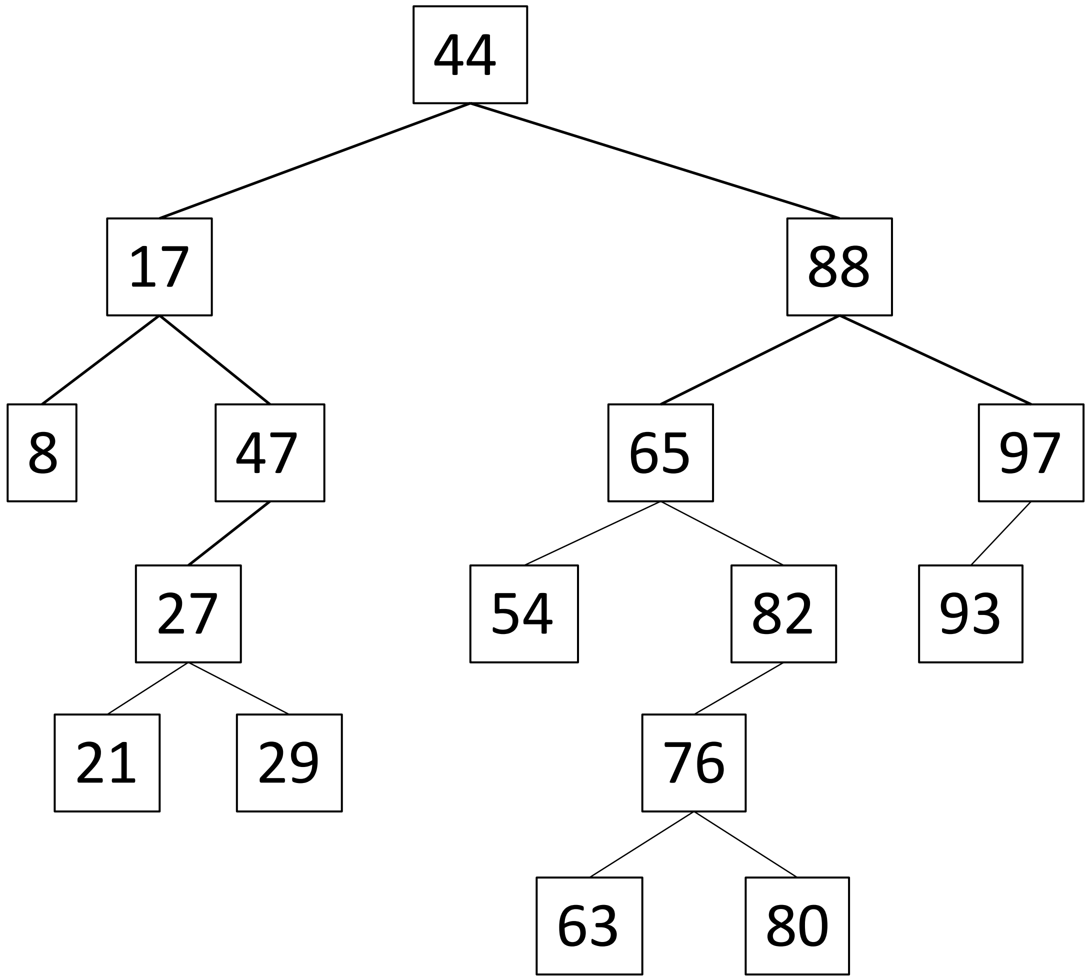


Is it a BST?

Binary Search Tree

Binary Search Tree (BST) properties:

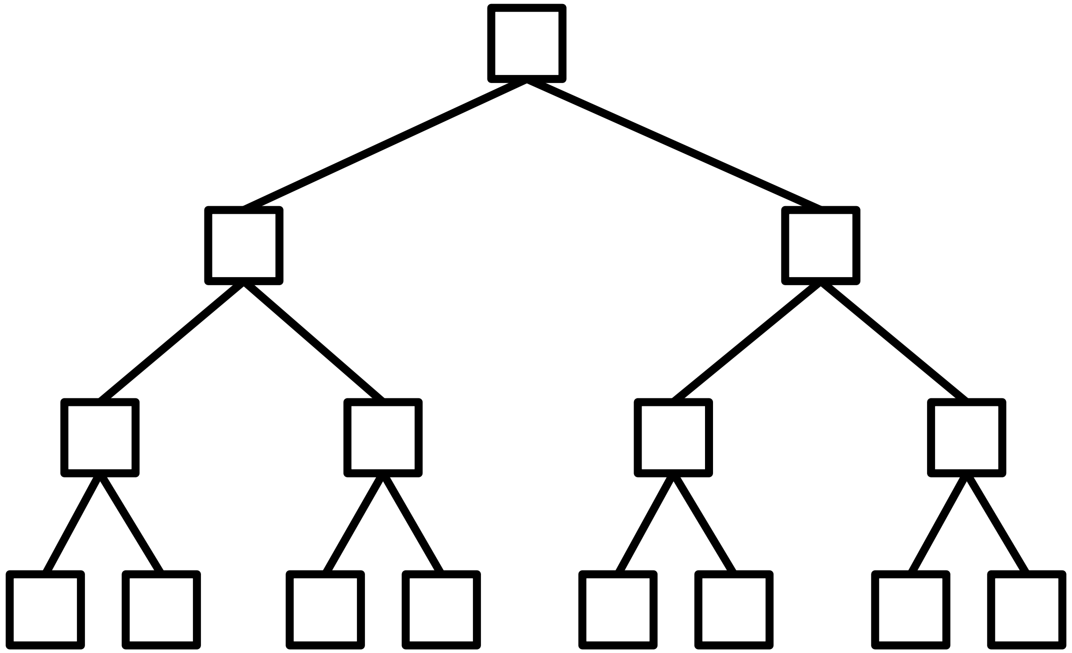
- **A BST is composed of `Comparable` data elements.**
- **A BST is a binary tree (each node has at most two children).**
- **For each node, all left-hand descendants have values that are less that the node.**
- **For each node, all right-hand descendants have values that are larger than the node.**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



Is it a BST?

Binary Search Tree

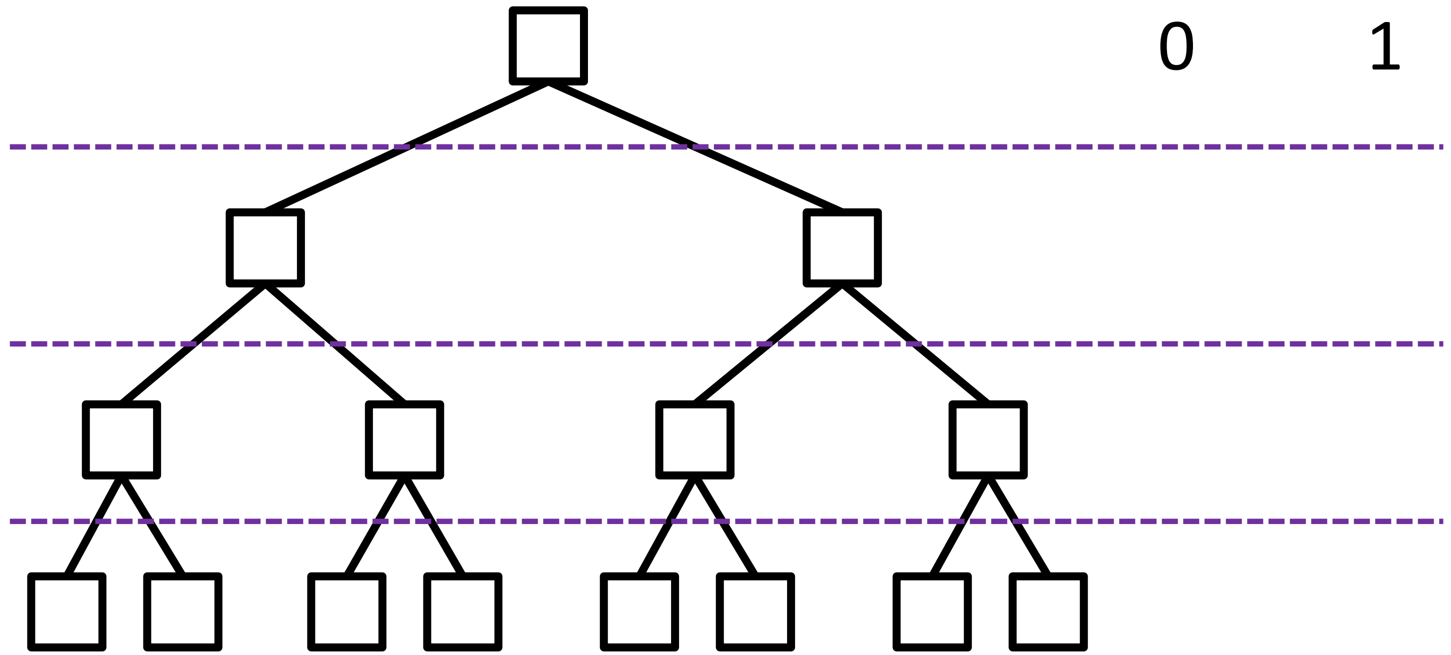
What is the point? Why use a BST?



Binary Search Tree

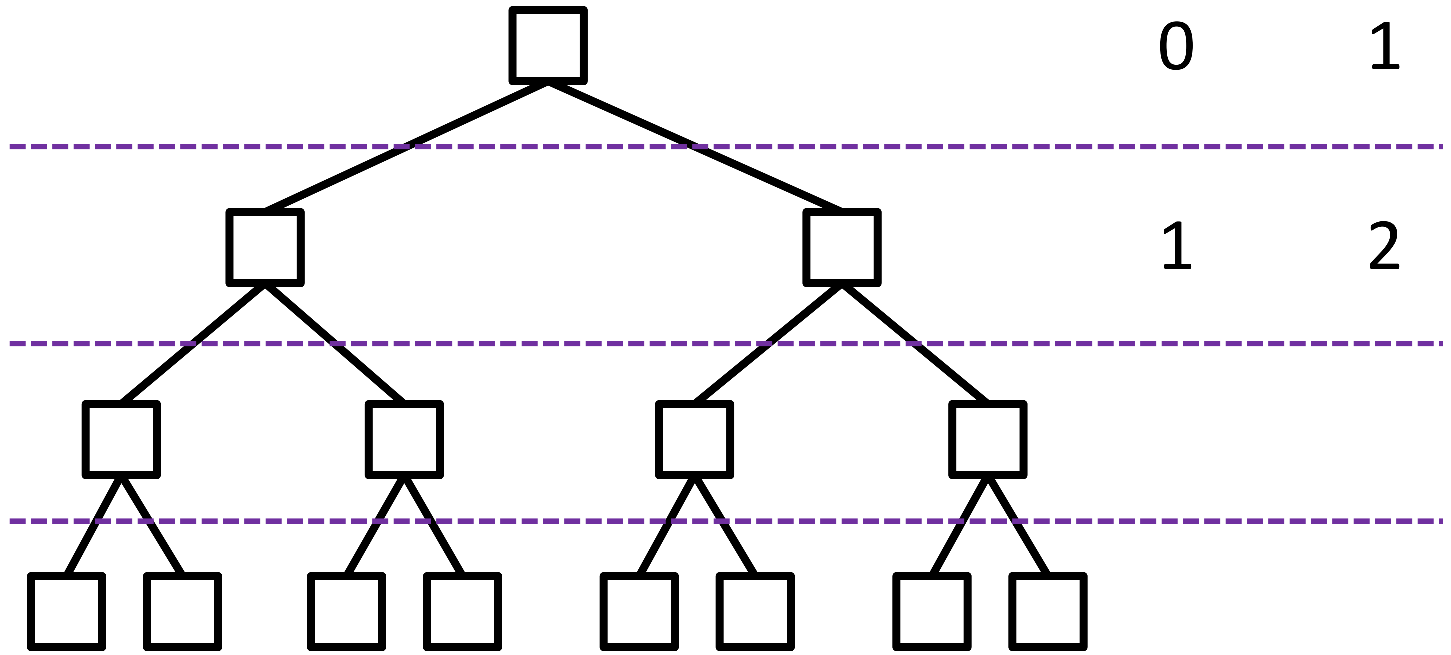
What is the point? Why use a BST?

Depth	Num Nodes
0	1



Binary Search Tree

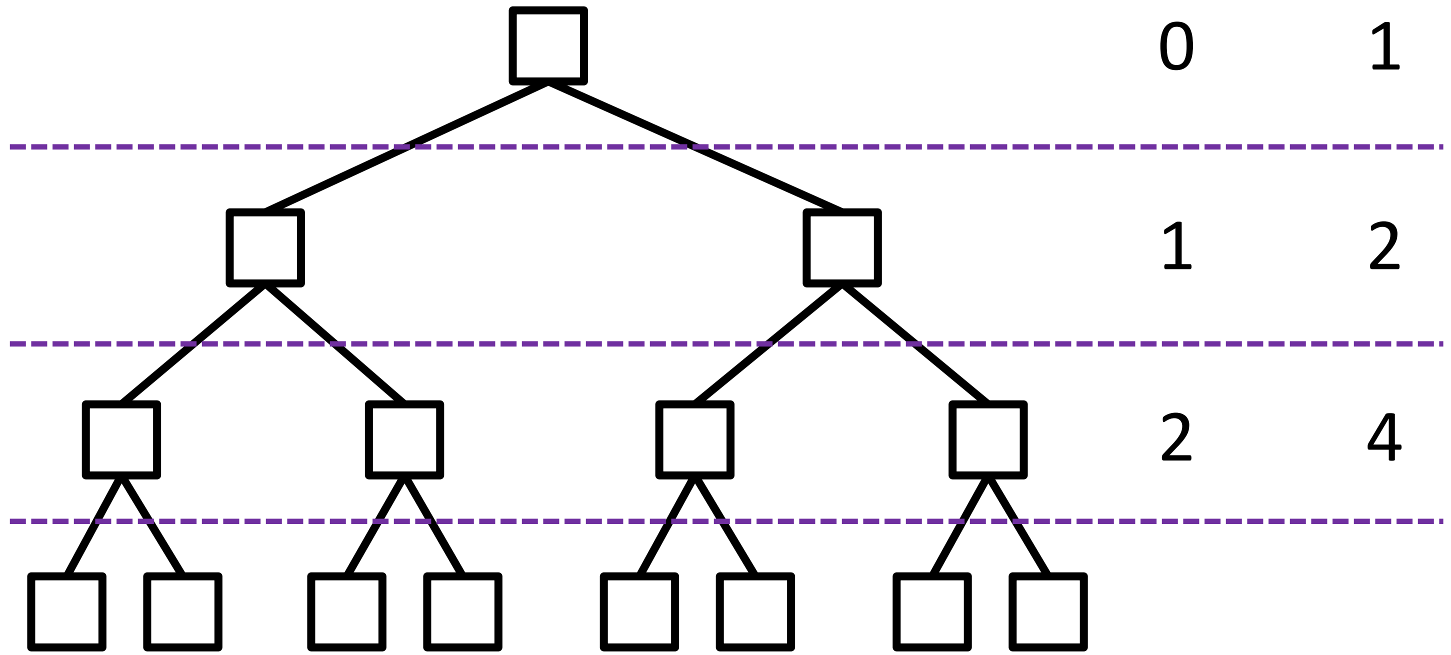
What is the point? Why use a BST?



Depth	Num Nodes
0	1
1	2

Binary Search Tree

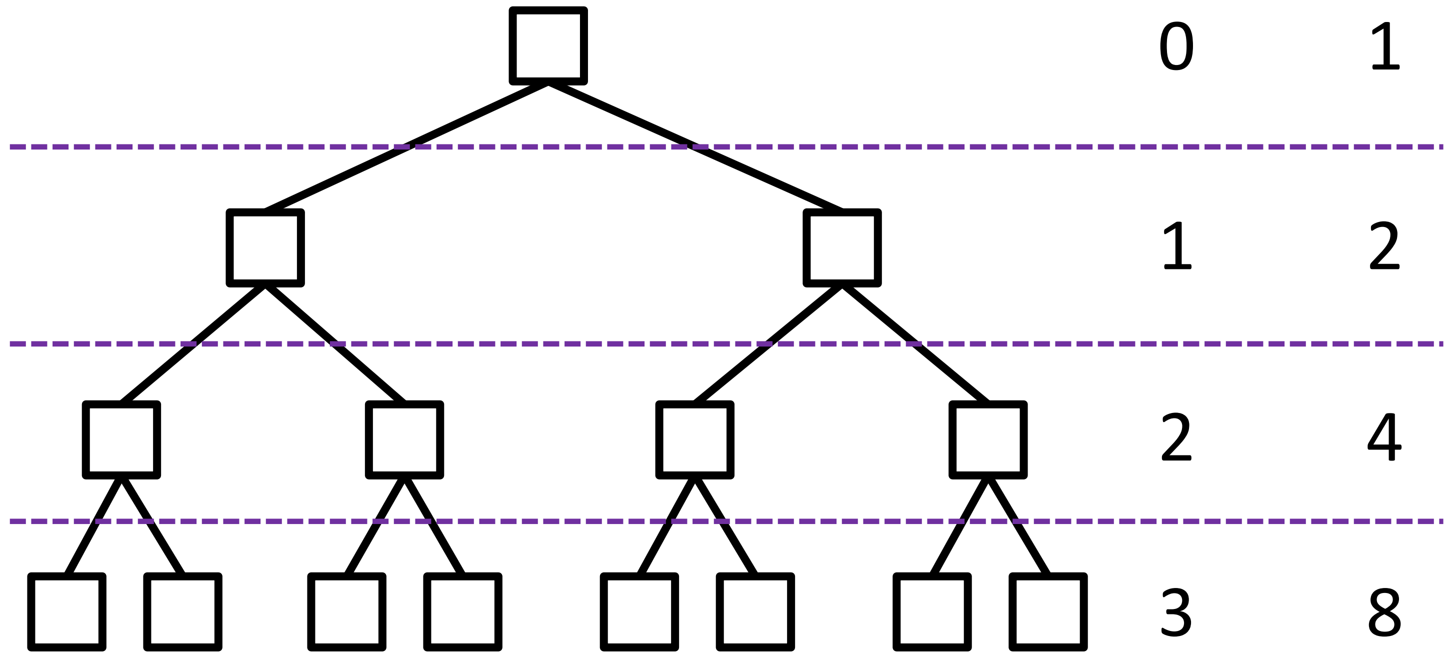
What is the point? Why use a BST?



Depth	Num Nodes
0	1
1	2
2	4

Binary Search Tree

What is the point? Why use a BST?

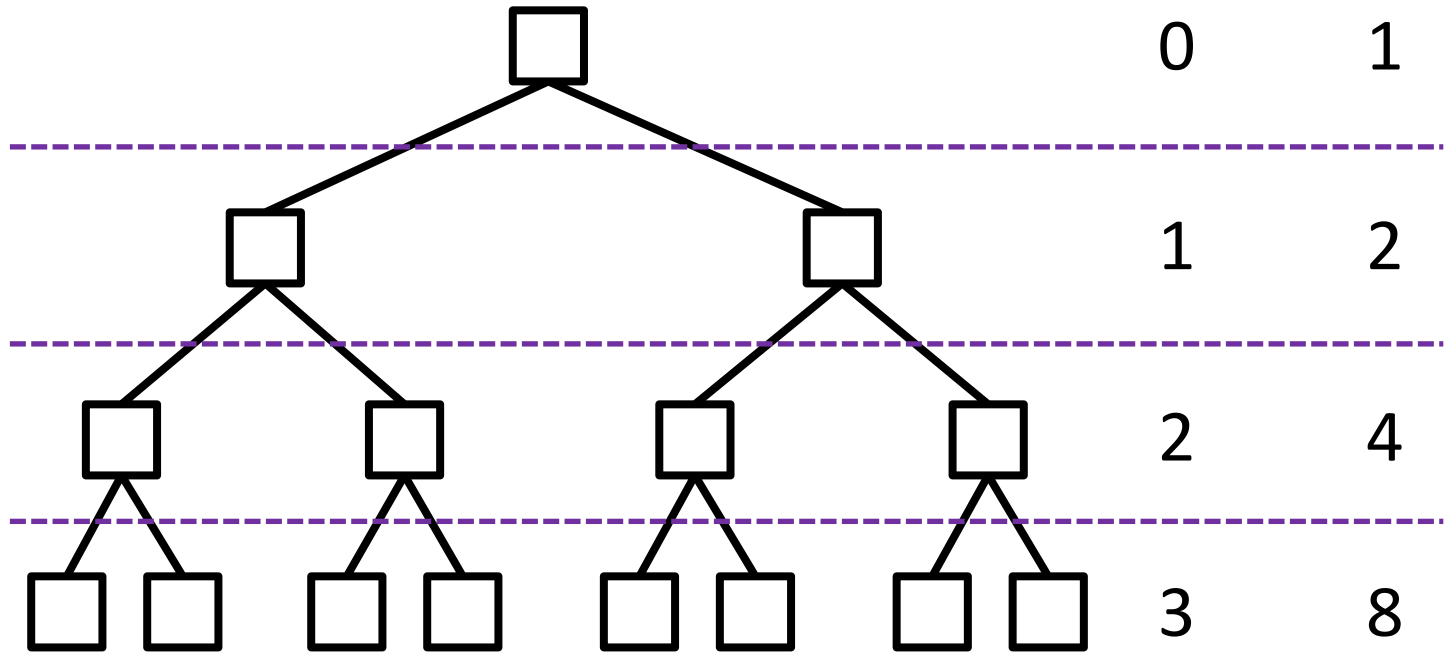


Depth	Num Nodes
0	1
1	2
2	4
3	8

Binary Search Tree

What is the point? Why use a BST?

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

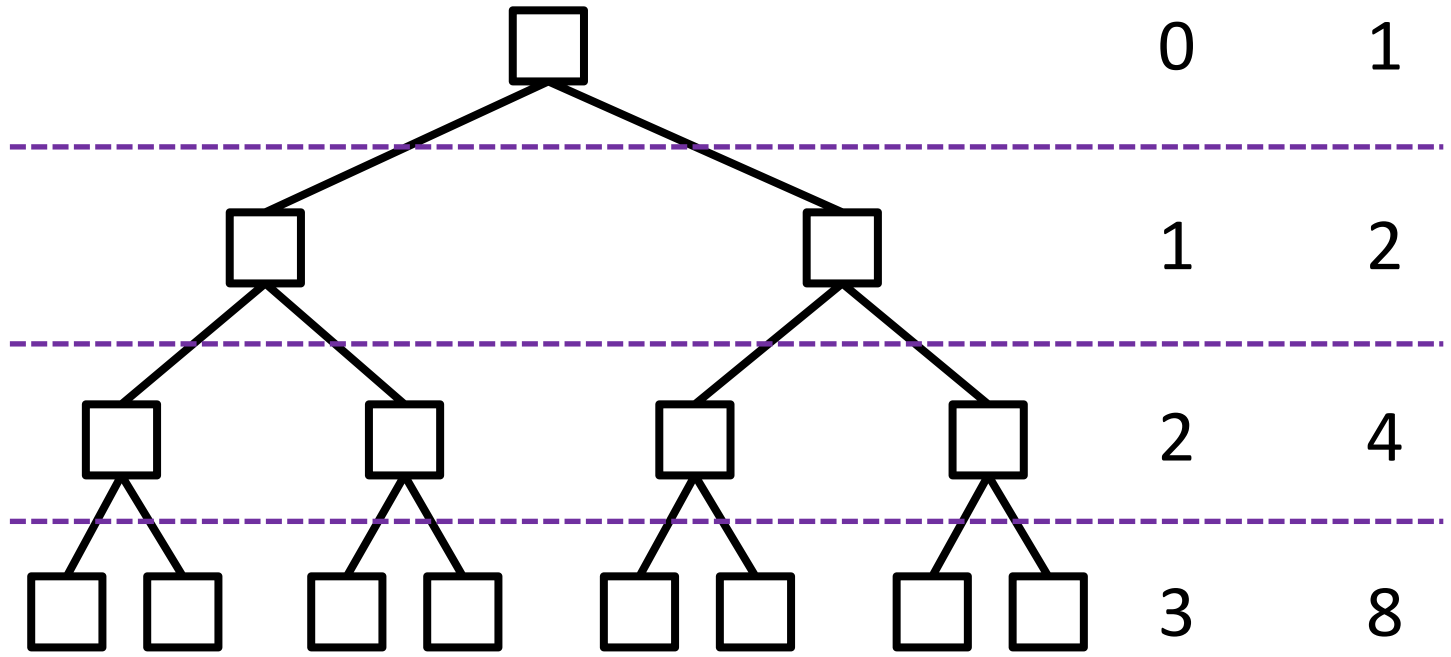


Depth	Num Nodes
0	1
1	2
2	4
3	8

Binary Search Tree

What is the point? Why use a BST?

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



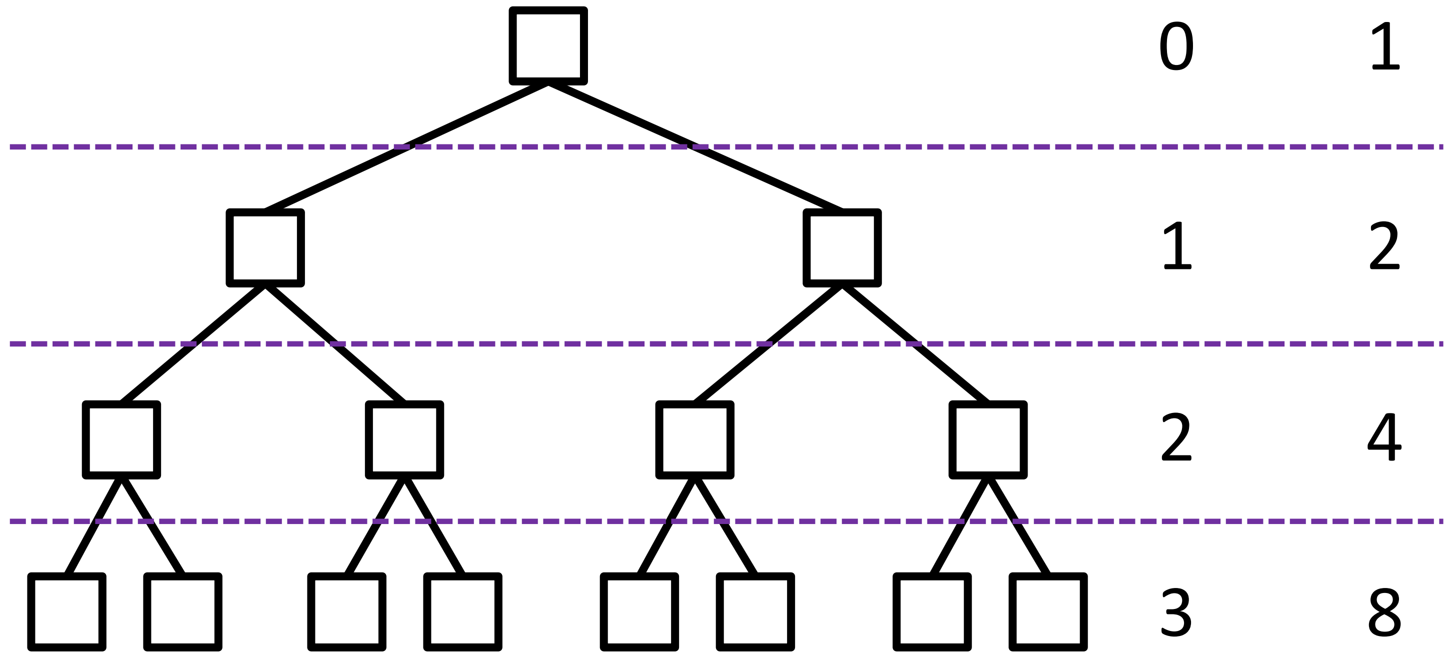
Depth	Num Nodes
0	1
1	2
2	4
3	8

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?



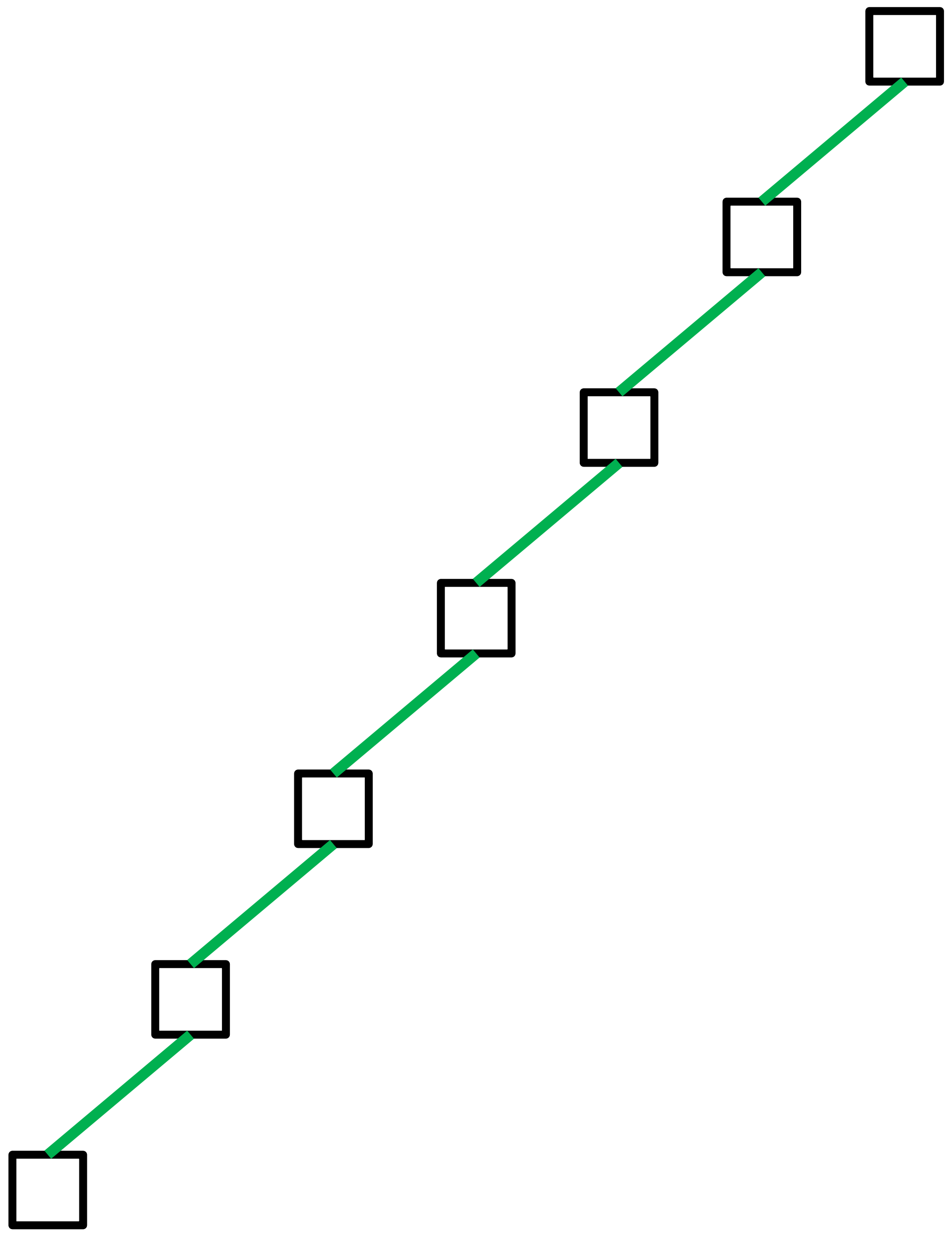
Depth	Num Nodes
0	1
1	2
2	4
3	8

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? $n - 1$

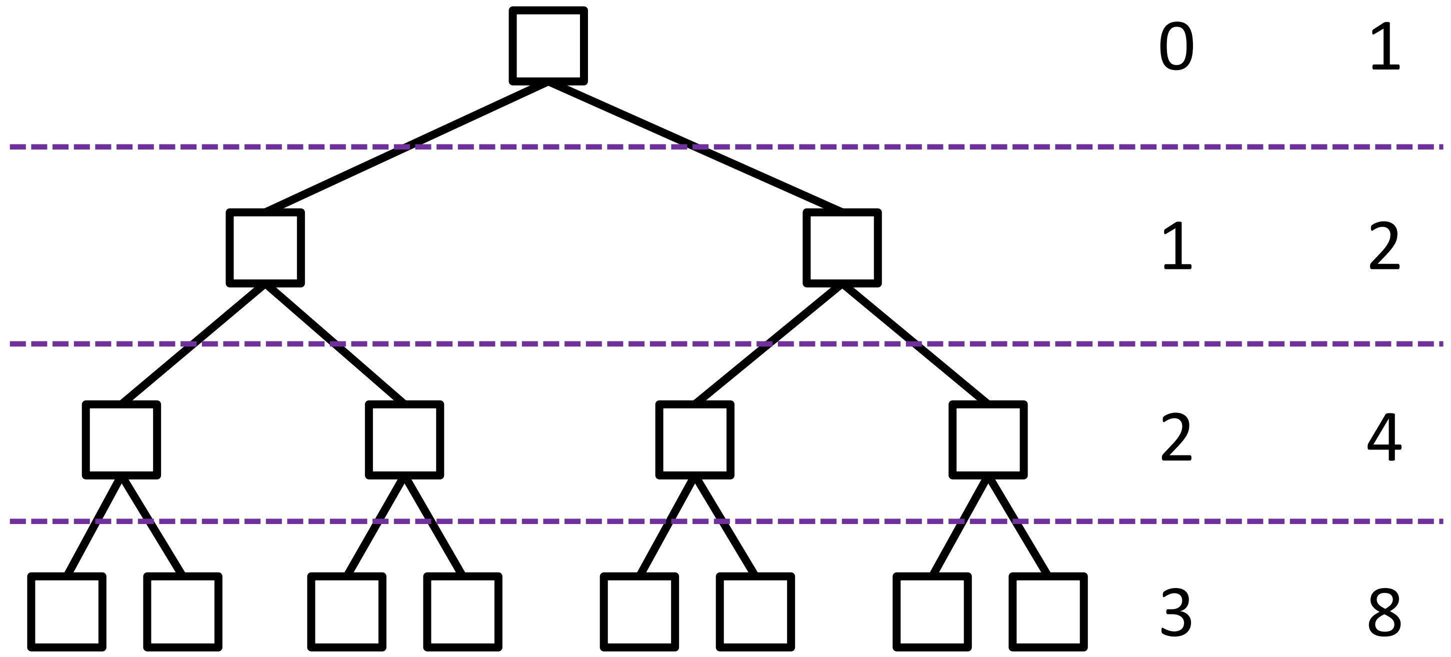


Binary Search Tree

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?



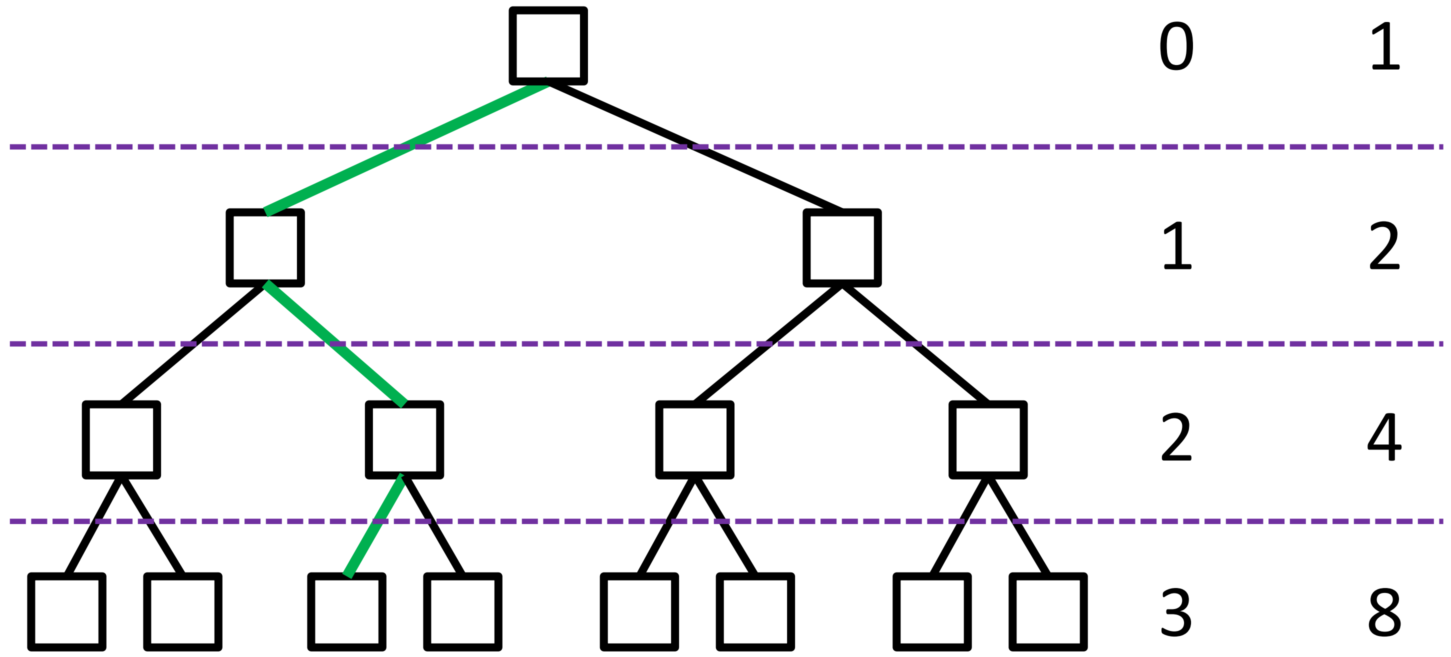
Depth	Num Nodes
0	1
1	2
2	4
3	8

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.*



Depth	Num Nodes
0	1
1	2
2	4
3	8

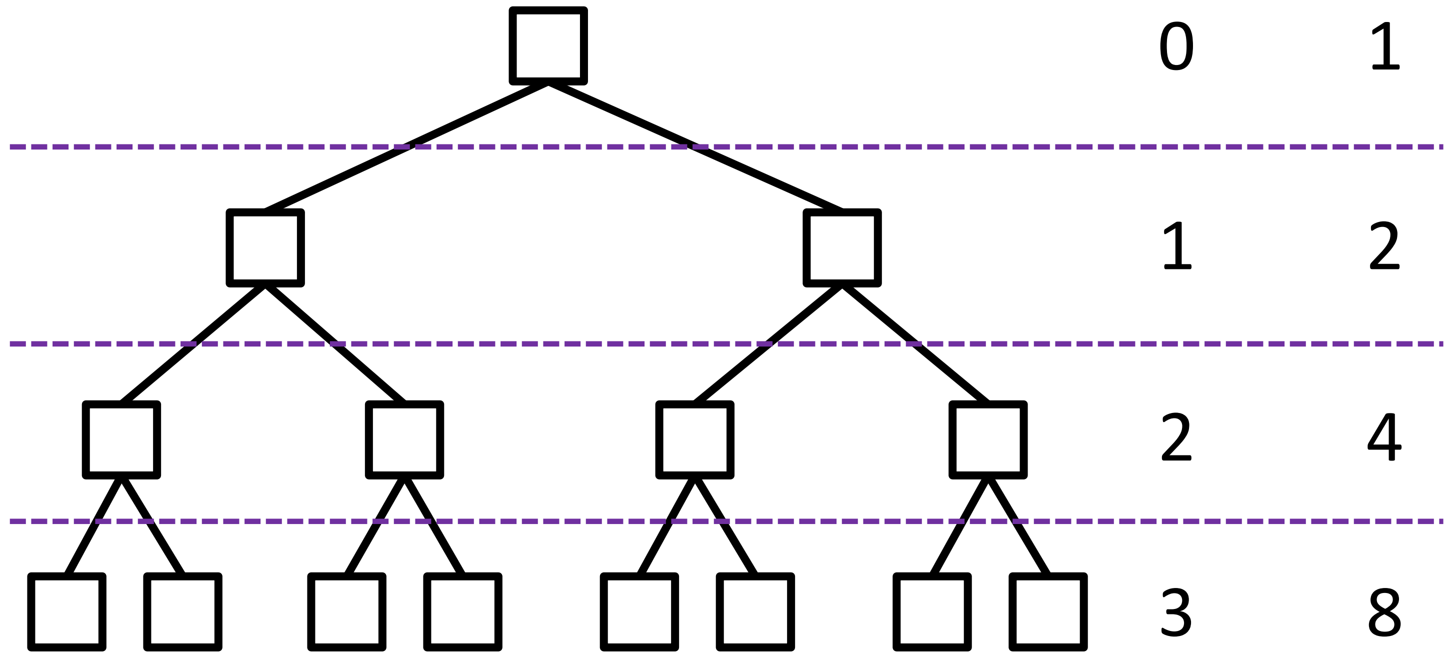
Binary Search Tree

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?



Depth	Num Nodes
0	1
1	2
2	4
3	8

Binary Search Tree

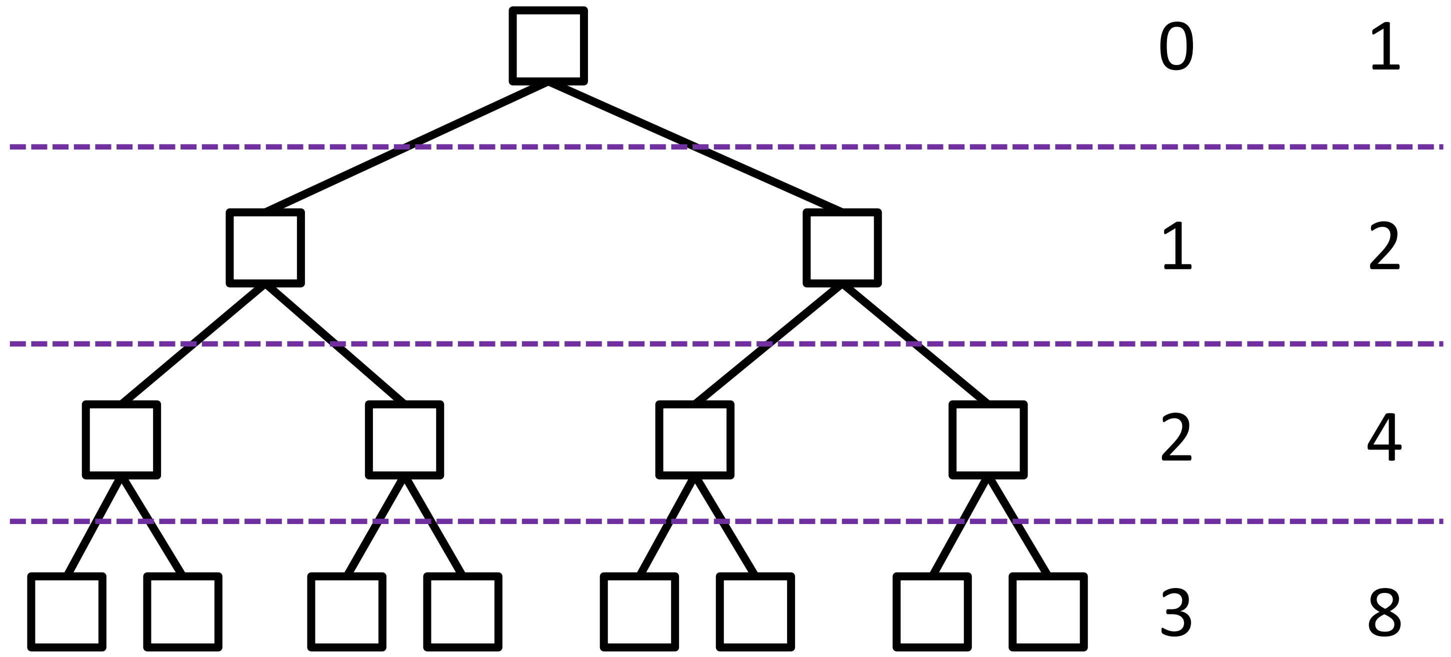
What is the point? Why use a BST?

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?

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



Depth	Num Nodes
0	1
1	2
2	4
3	8

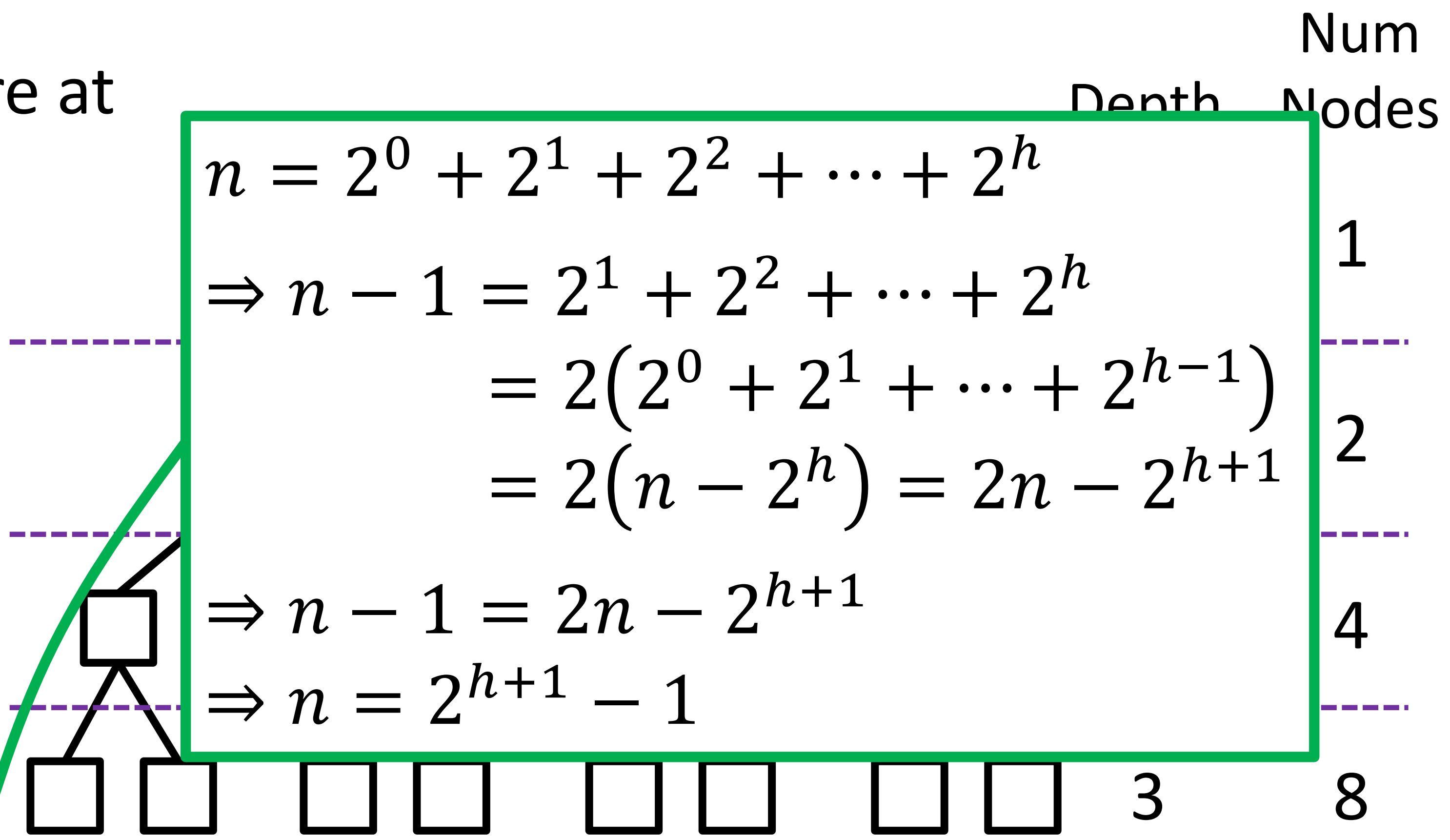
Binary Search Tree

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?



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

$$\Rightarrow n - 1 = 2^1 + 2^2 + \cdots + 2^h$$

$$= 2\left(2^0 + 2^1 + \cdots + 2^{h-1}\right)$$

$$= 2\left(n - 2^h\right) = 2n - 2^{h+1}$$

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

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

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

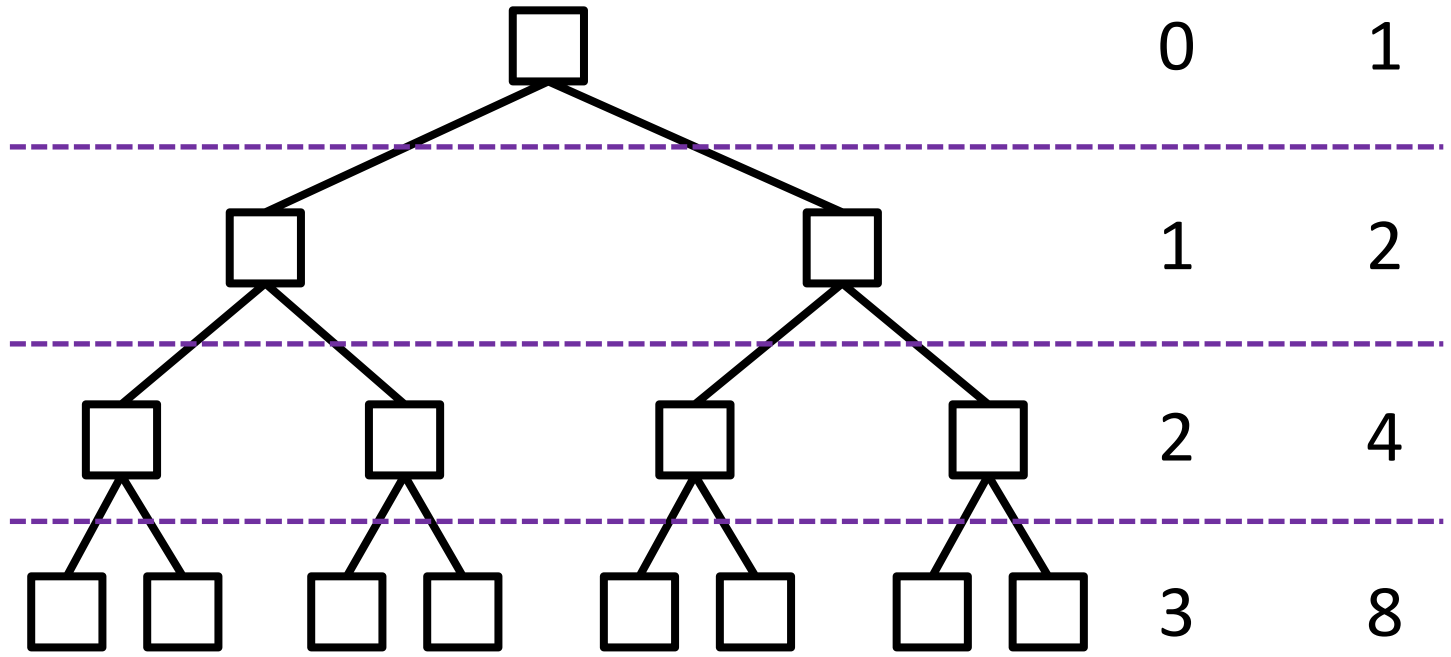
Binary Search Tree

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?



Depth	Num Nodes
0	1
1	2
2	4
3	8

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

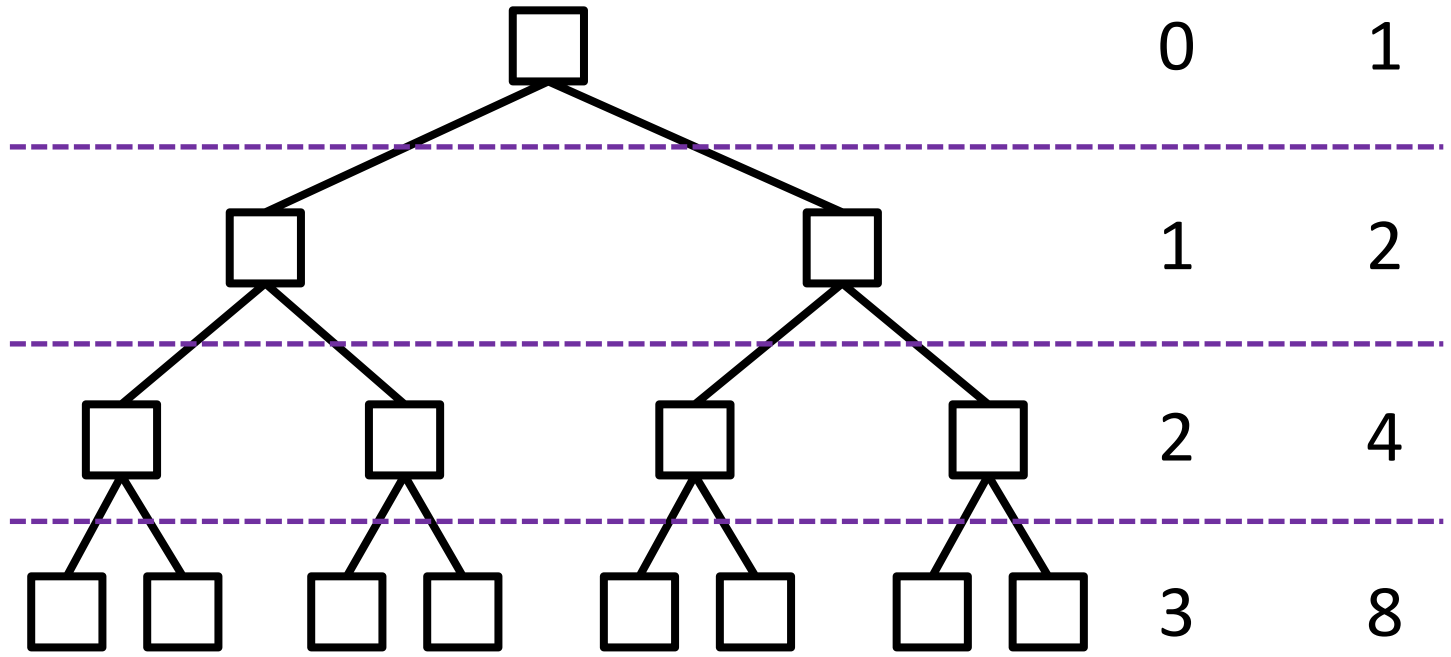
Binary Search Tree

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?



Depth	Num Nodes
0	1
1	2
2	4
3	8

$$n = 2^0 + 2^1 + 2^2 + \cdots + 2^h = 2^{h+1} - 1 \Rightarrow \begin{array}{l} n + 1 = 2^{h+1} \\ \log_2(n+1) = h + 1 \end{array}$$

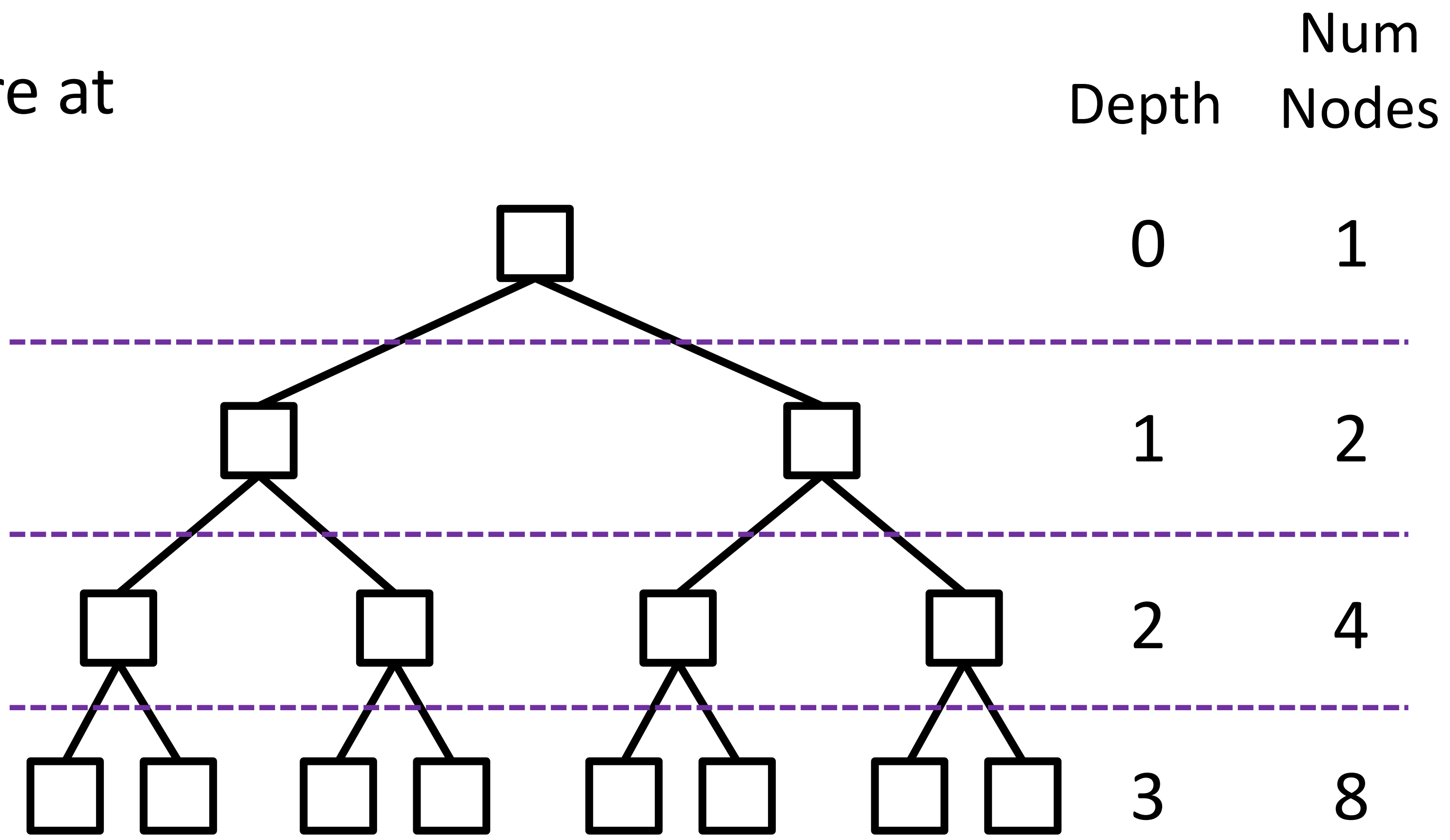
Binary Search Tree

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?

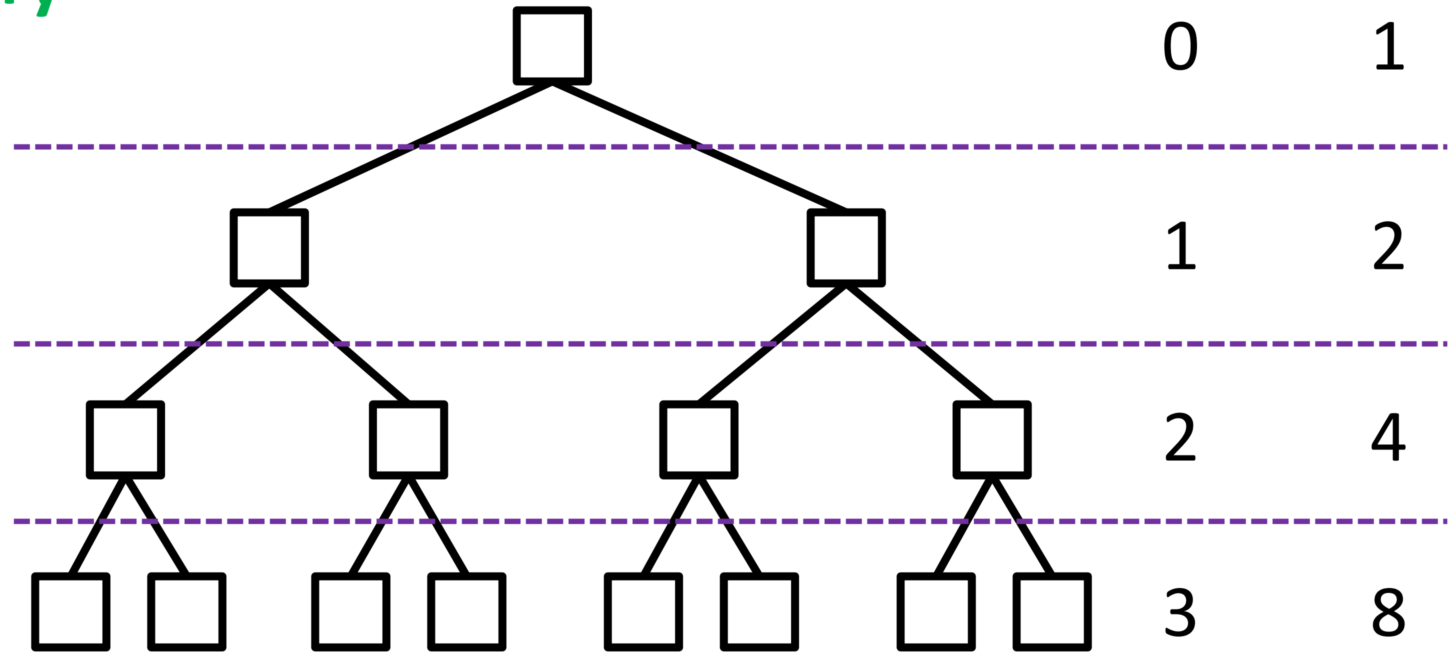


$$n = 2^0 + 2^1 + 2^2 + \cdots + 2^h = 2^{h+1} - 1 \Rightarrow \begin{matrix} n + 1 = 2^{h+1} \\ \log_2(n+1) = h + 1 \end{matrix} \Rightarrow \boldsymbol{h \in O(\log n)}$$

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.



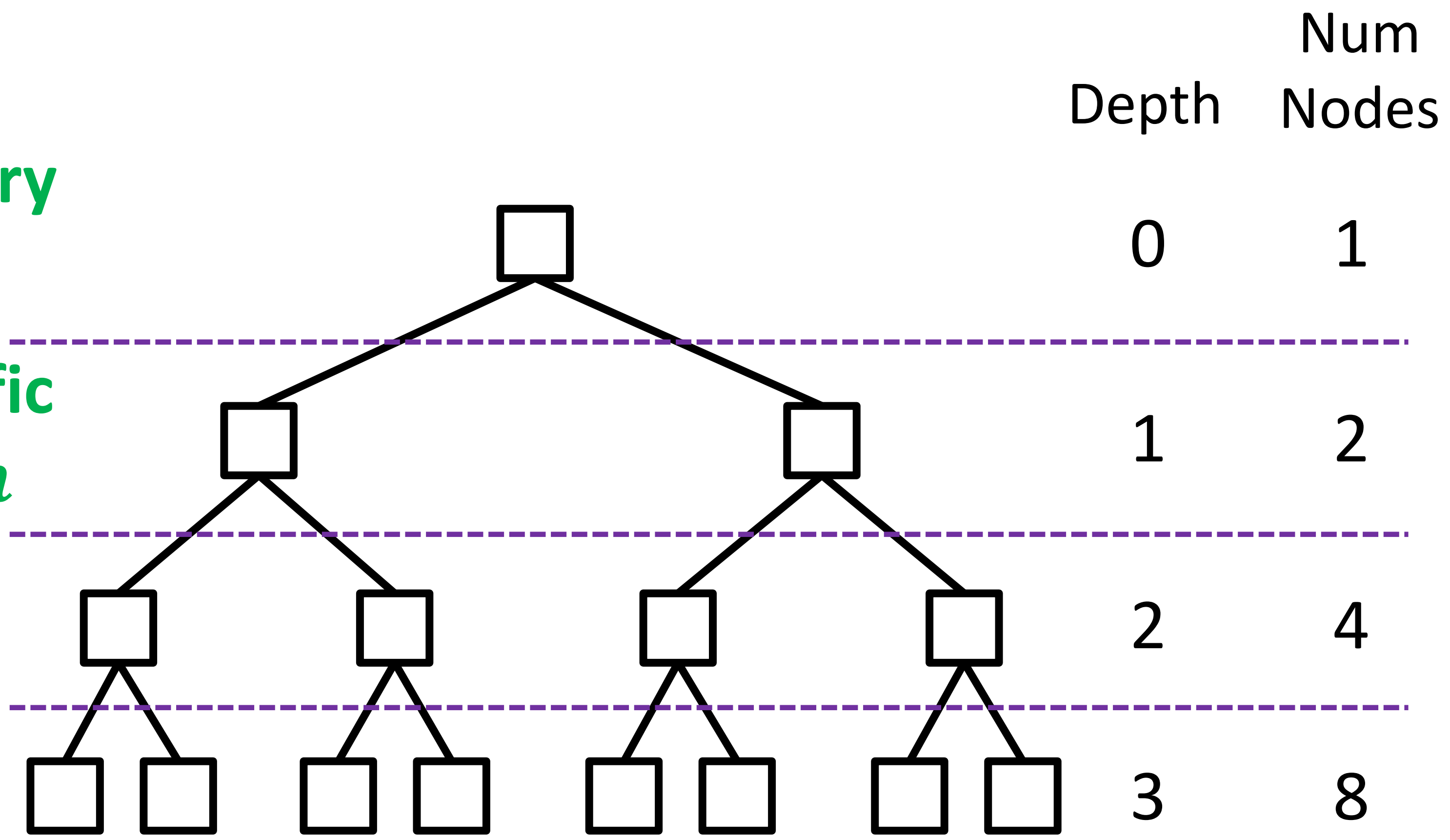
Depth	Num Nodes
0	1
1	2
2	4
3	8

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.



Depth	Num Nodes
0	1
1	2
2	4
3	8

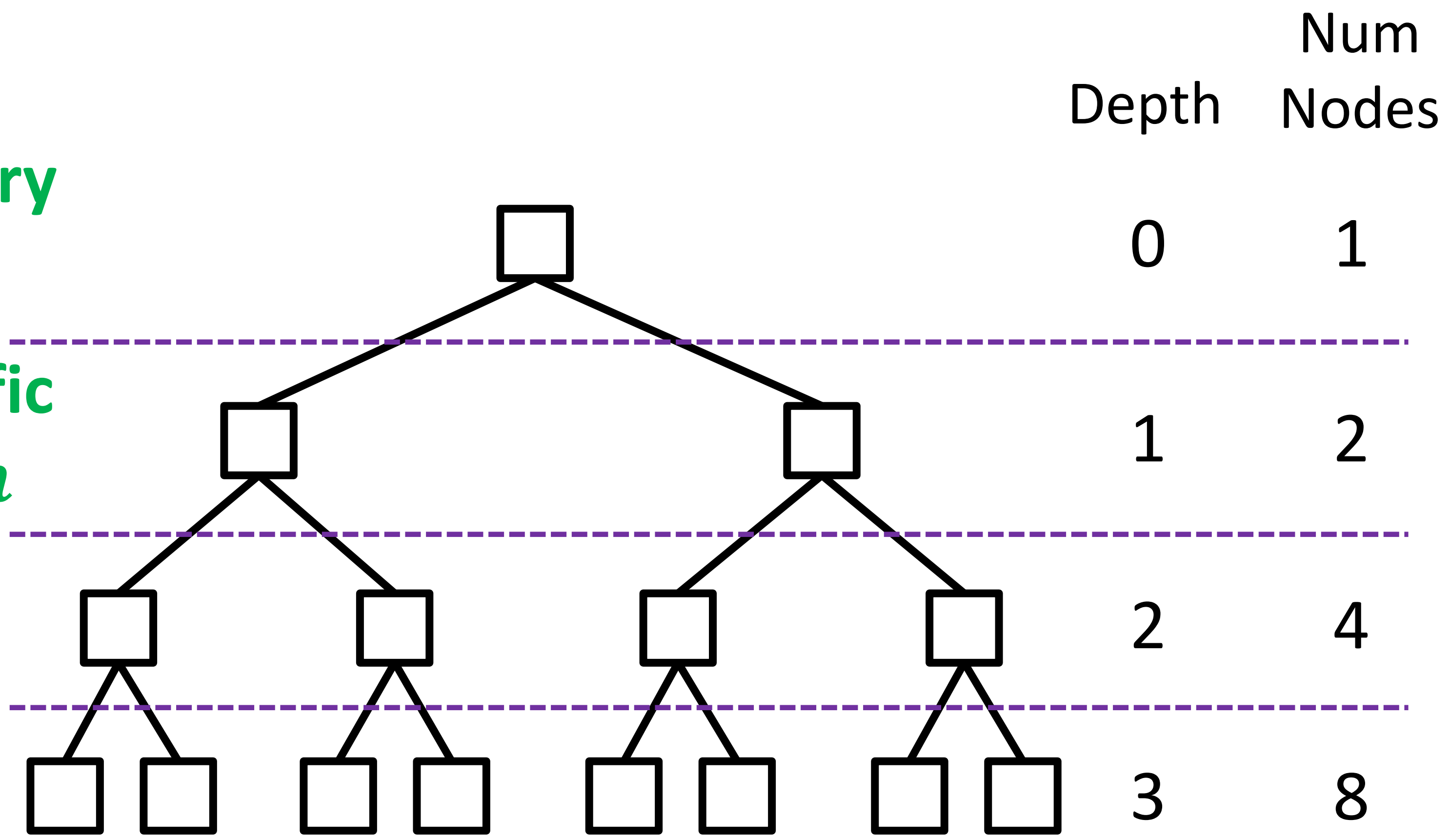
Binary Search Tree

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!



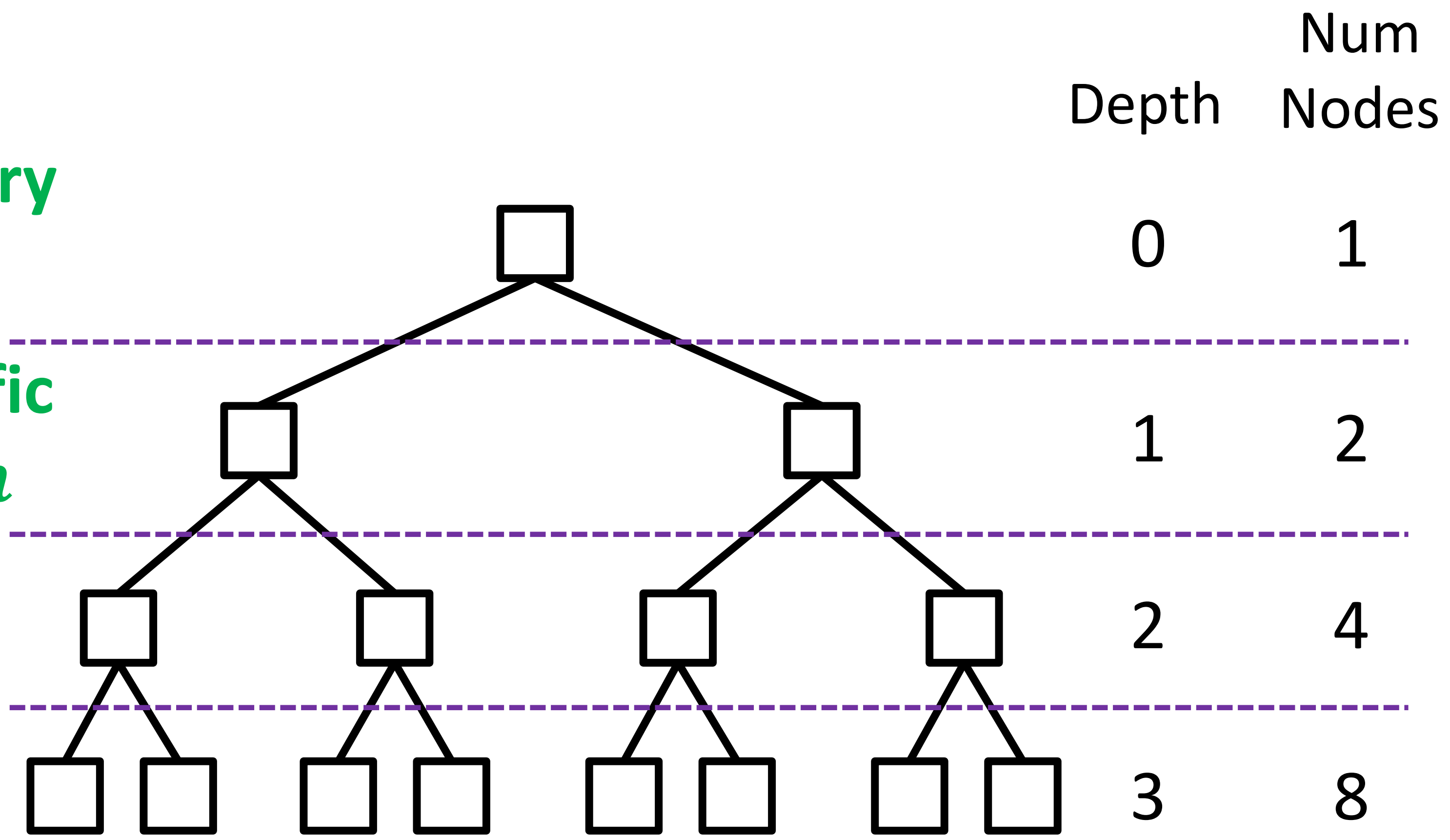
Binary Search Tree

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.

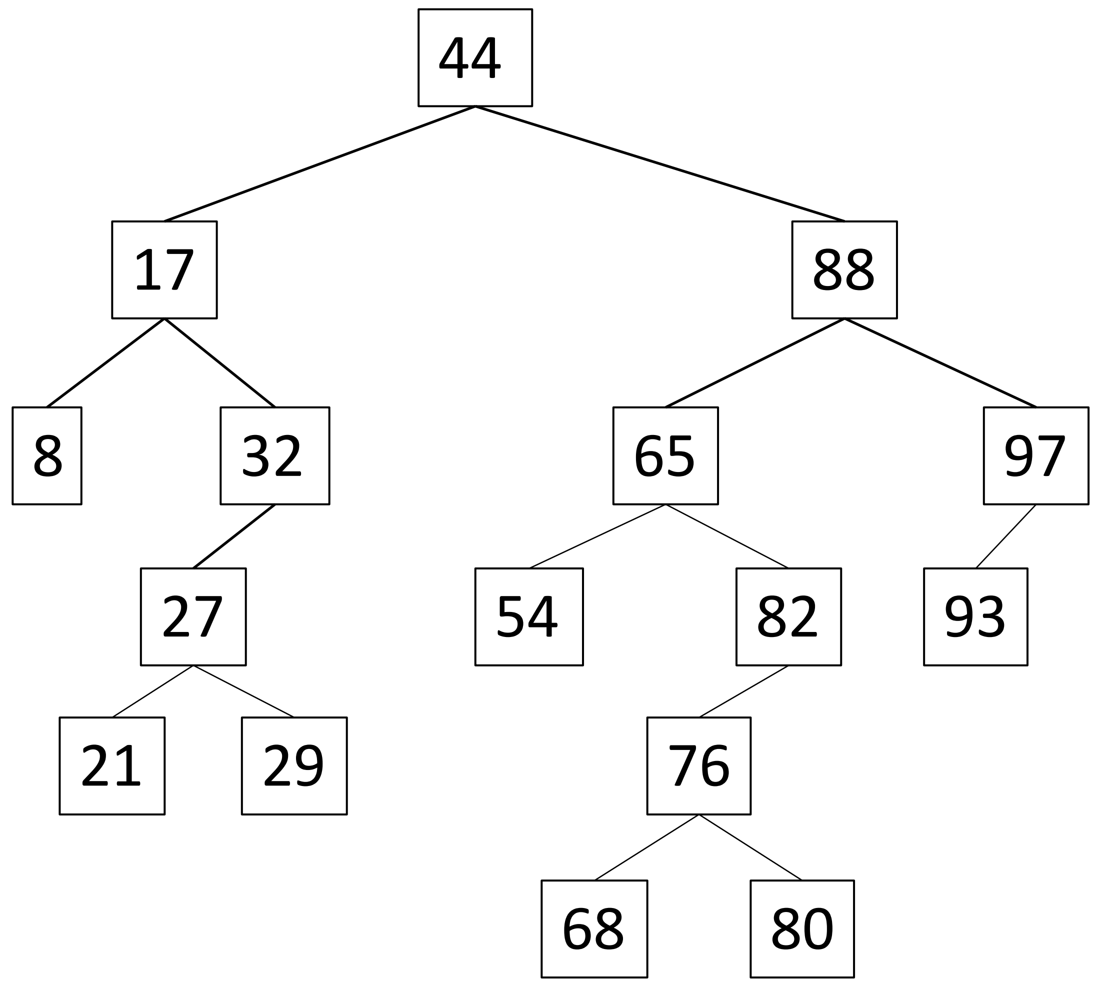
But we can already do that with a sorted array and Binary Search!



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

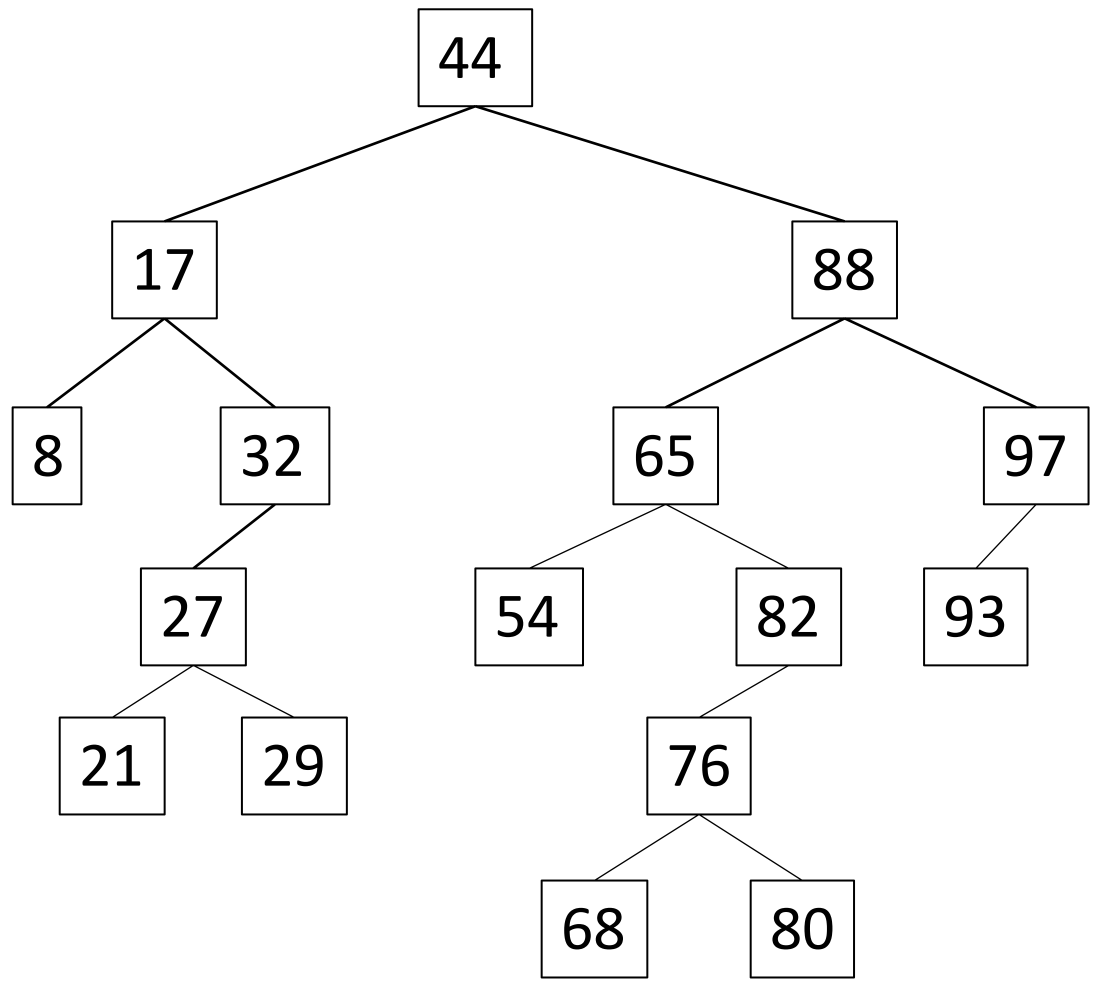
Binary Search Tree - Insertion

```
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);
```

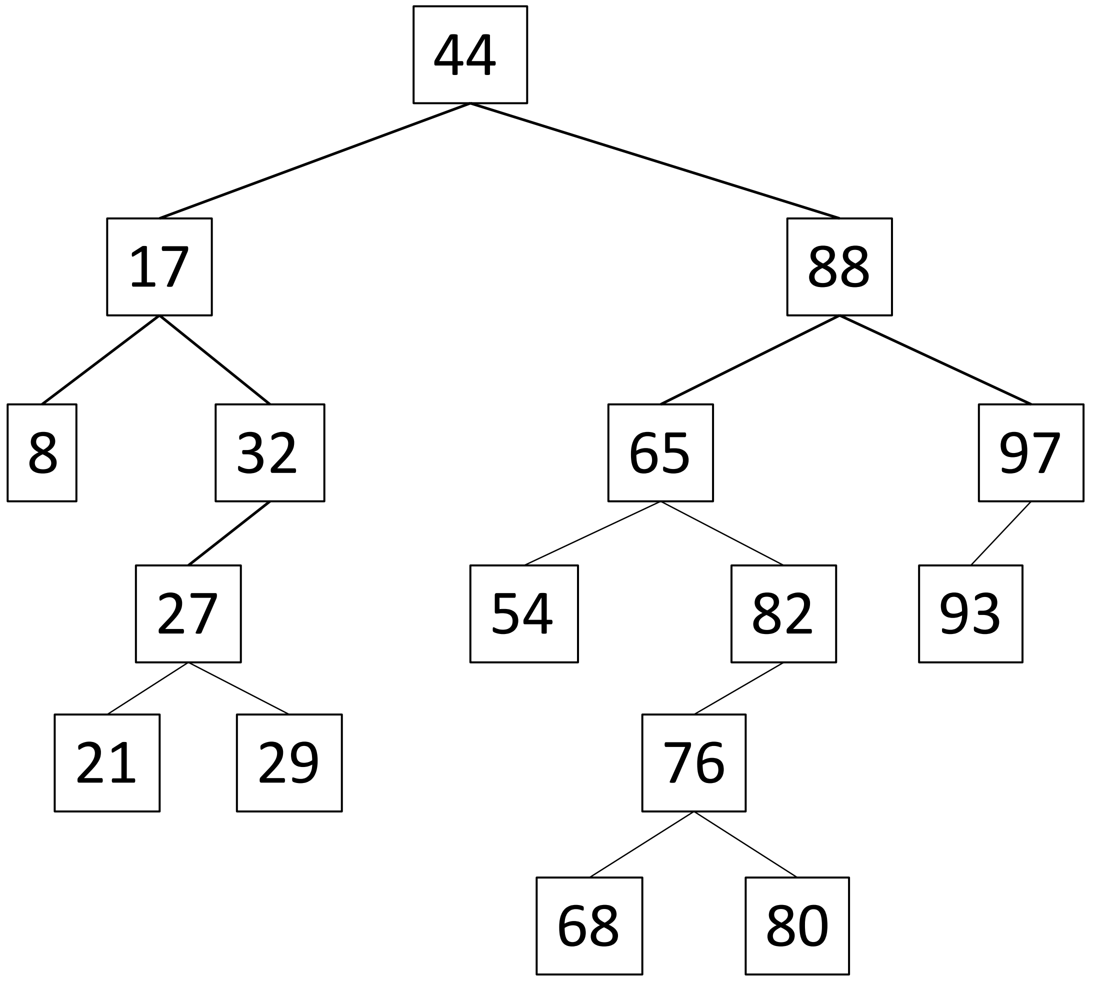


Binary Search Tree - Insertion

```
insert(31);
```

Step 1: Find where it should go.

Step 2: Modify pointers.

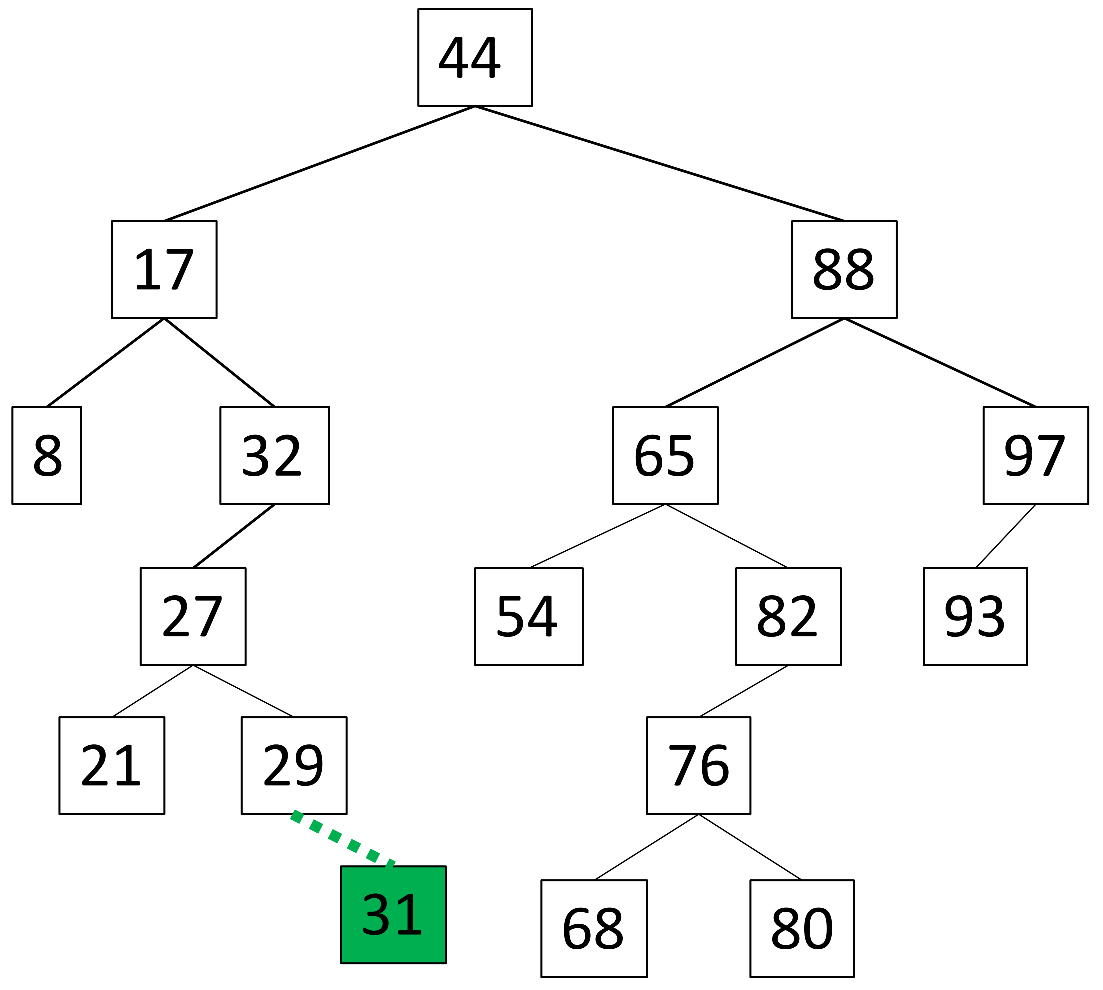


Binary Search Tree - Insertion

```
insert(31);
```

Step 1: Find where it should go.

Step 2: Modify pointers.

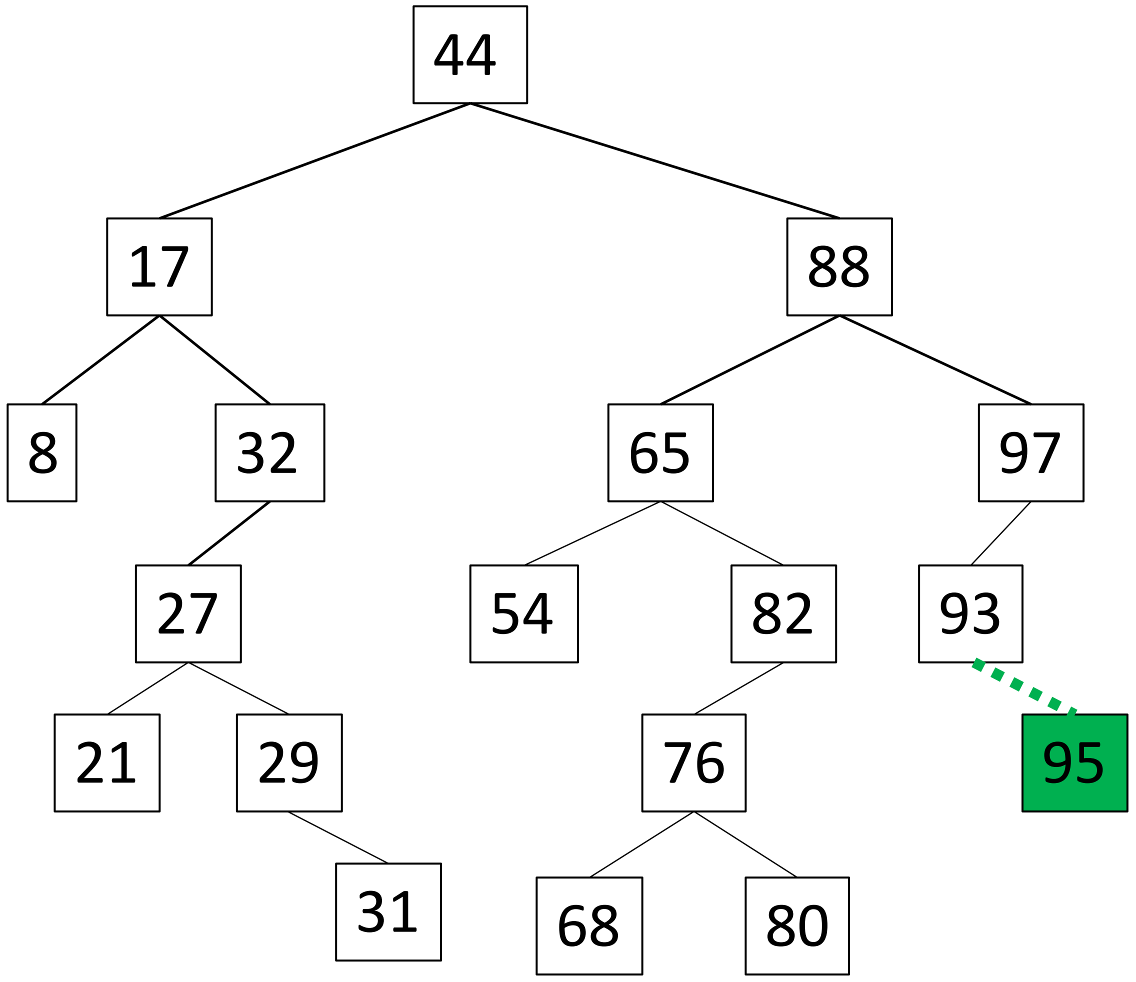


Binary Search Tree - Insertion

```
insert(95);
```

Step 1: Find where it should go.

Step 2: Modify pointers.



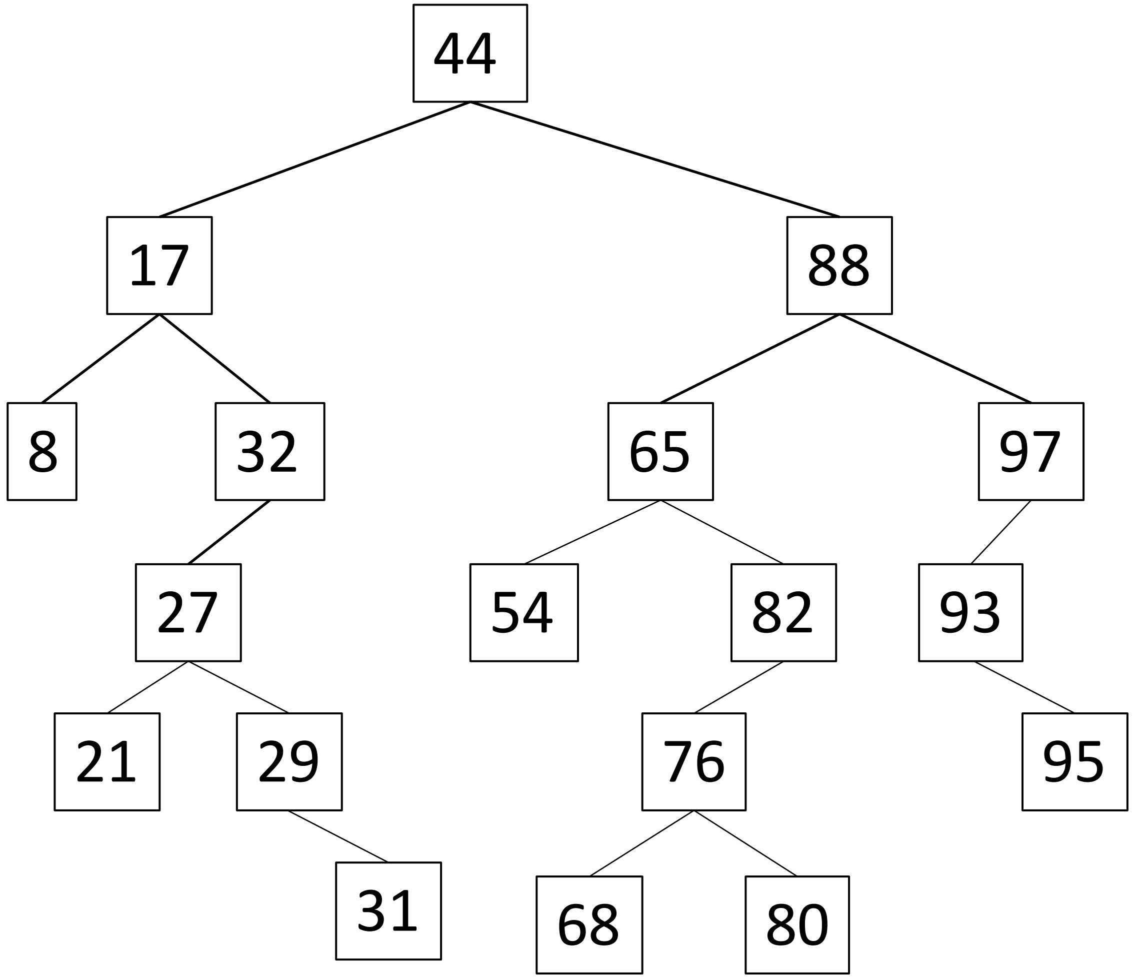
Binary Search Tree - Insertion

```
insert(95);
```

Step 1: Find where it should go.

Step 2: Modify pointers.

Any trends??



Binary Search Tree - Insertion

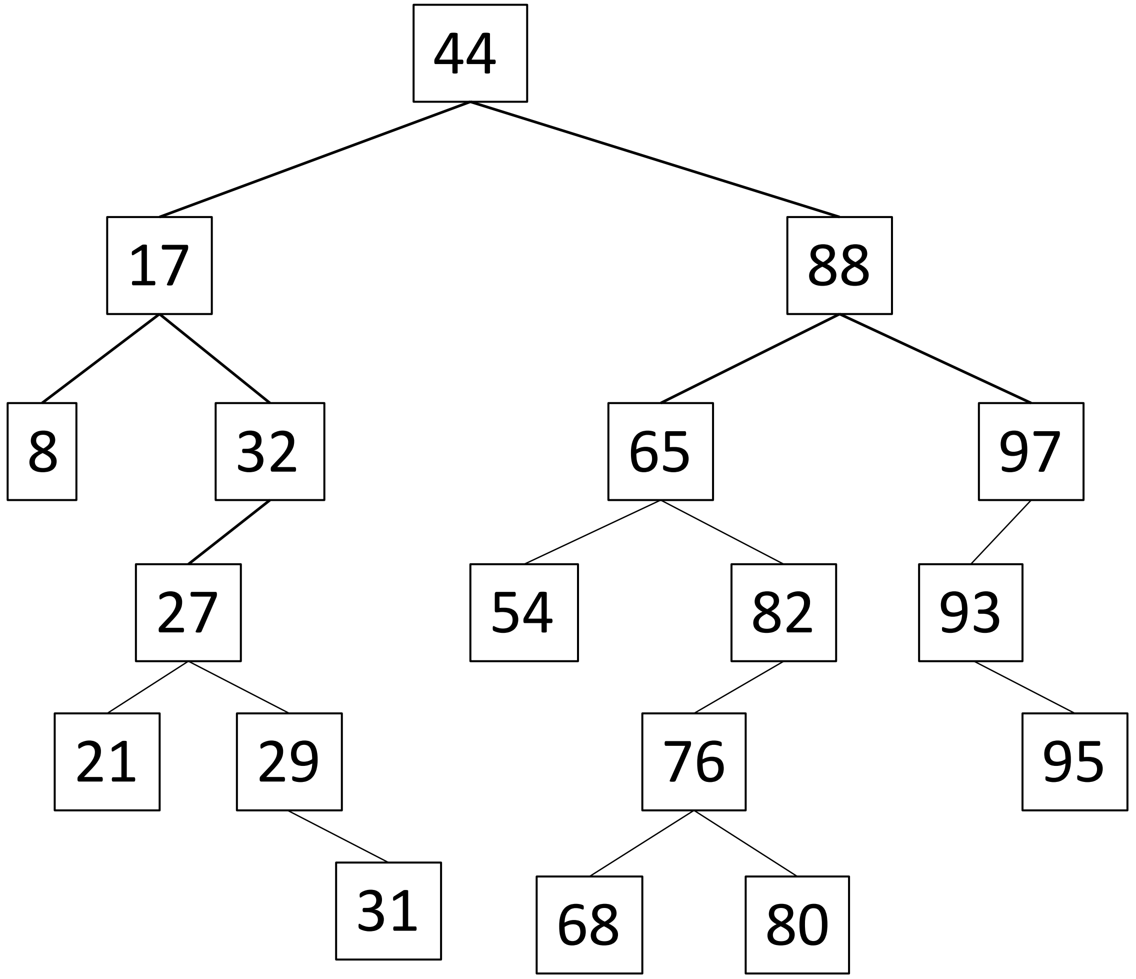
`insert(95);`

Step 1: Find where it should go.

Step 2: Modify pointers.

Any trends??

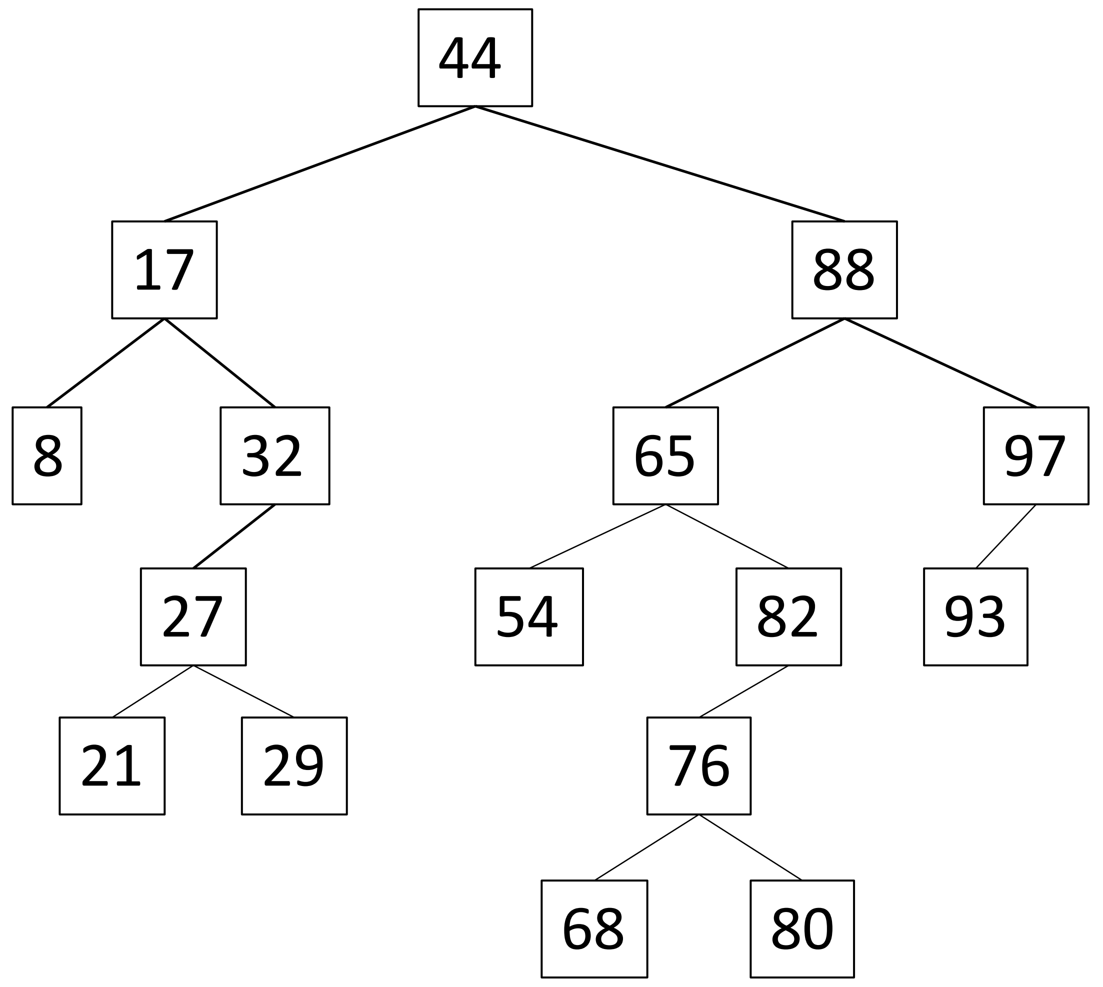
Always insert a new leaf!



Binary Search Tree - Insertion

`insert(28);`

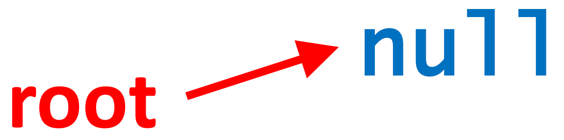
```
public void insert(int newValue) {




}
```



Binary Search Tree - Insertion

insert(28);

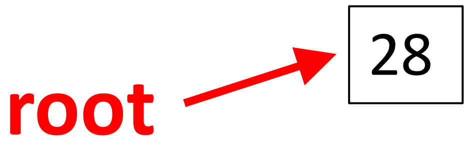
```
public void insert(int newValue) {
  if (root == null) {

  } else {









}
```



Binary Search Tree - Insertion

insert(28);

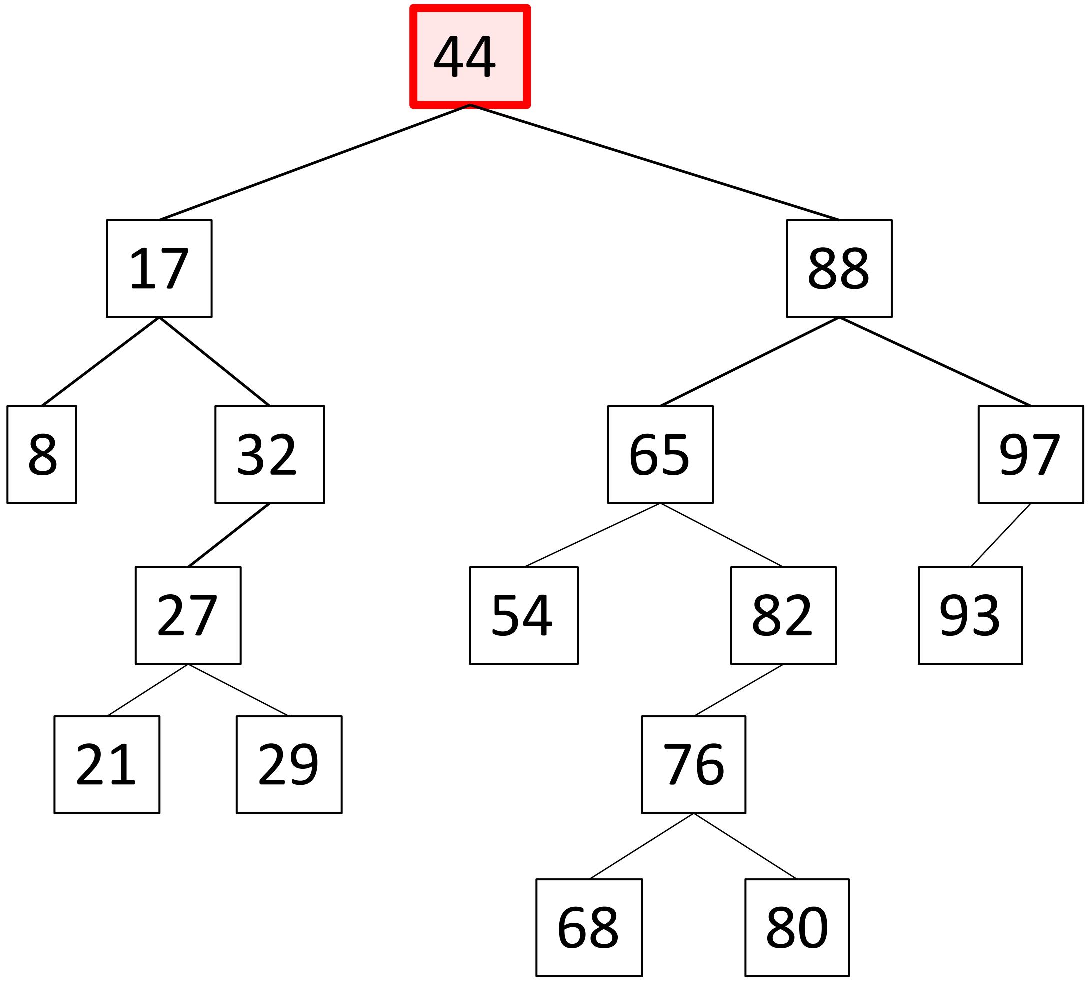
```
public void insert(int newValue) {
  if (root == null) {
    root = new Node(newValue);
  } else {



















}
```



Binary Search Tree - Insertion

insert(28);

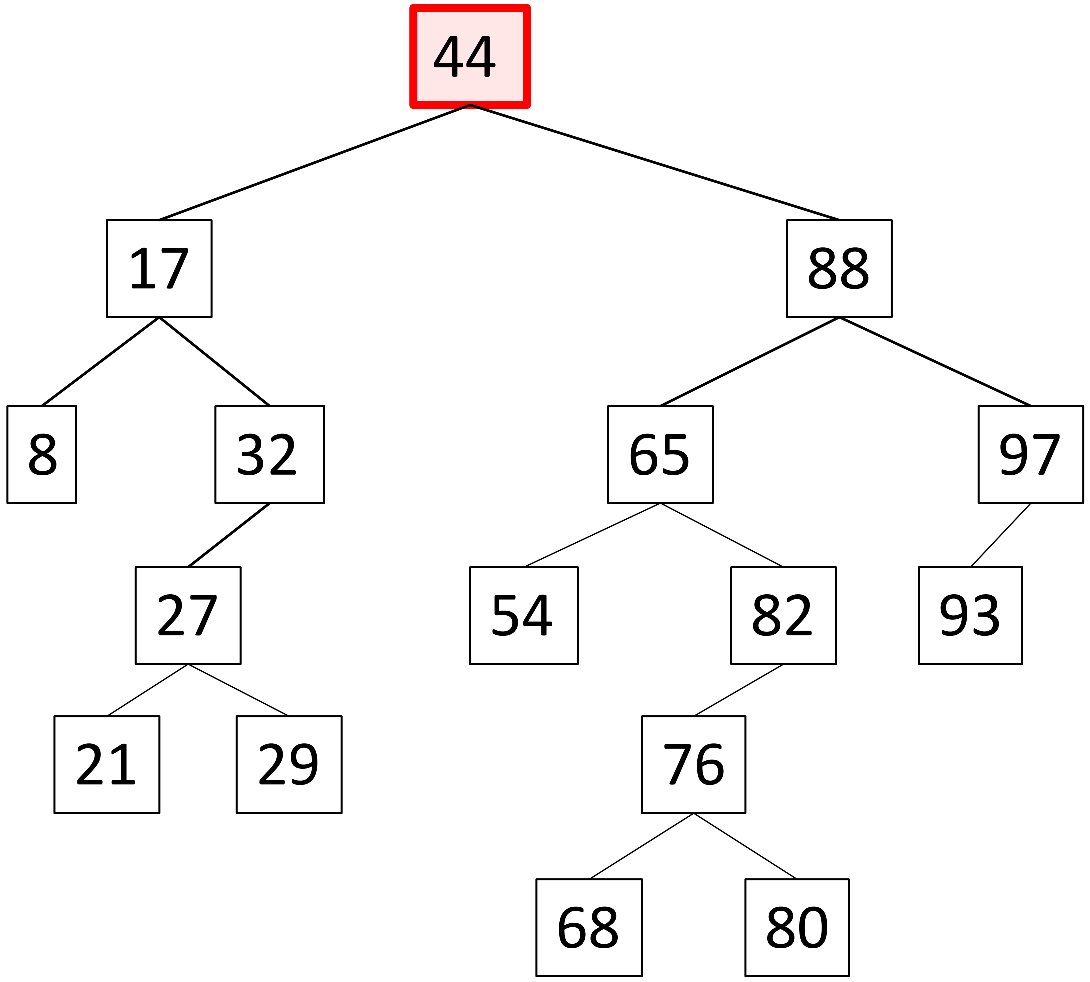
```
public void insert(int newValue) {
  if (root == null) {
    root = new Node(newValue);
  } else {
    Node currentNode = root;




}
```



Binary Search Tree - Insertion

insert(28);

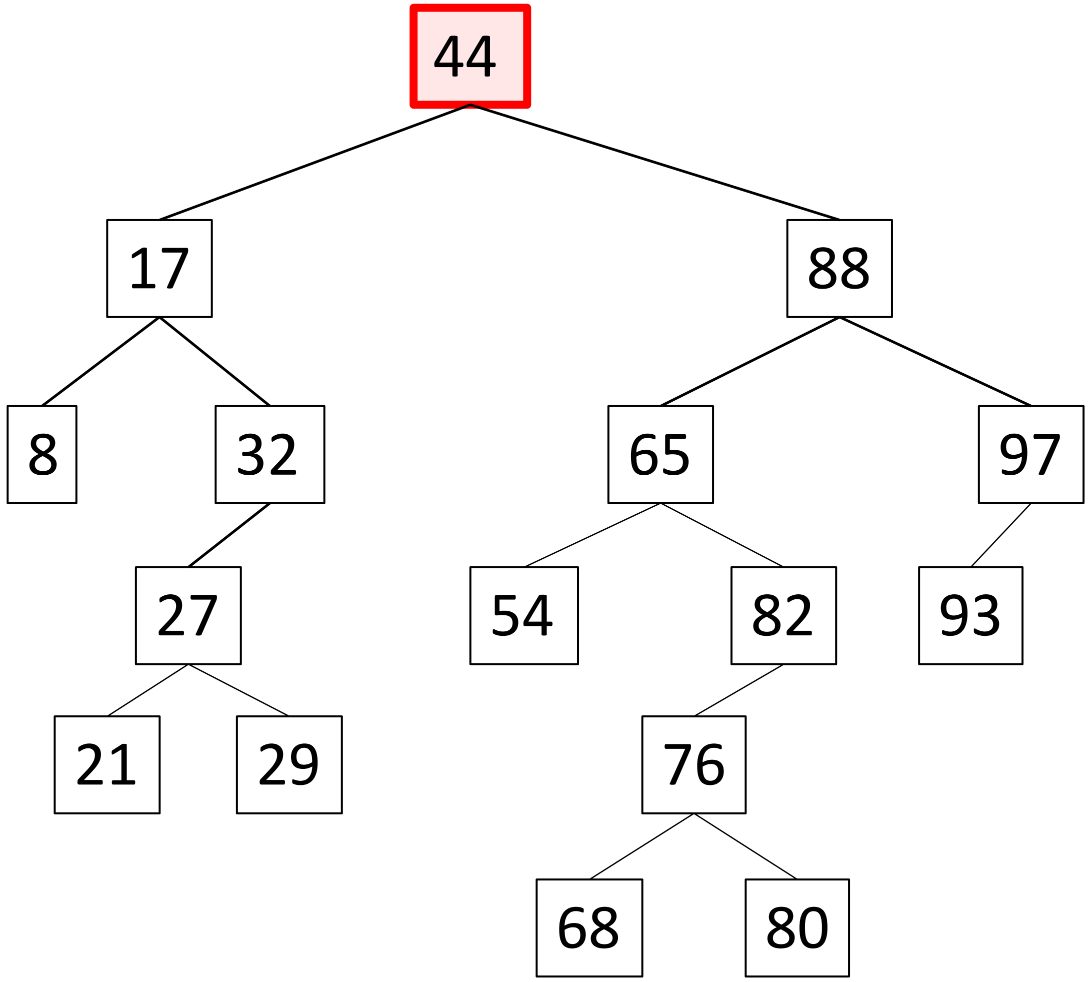
```
public void insert(int newValue) {
  if (root == null) {
    root = new Node(newValue);
  } else {
    Node currentNode = root;
    boolean placed = false;



}
```



Binary Search Tree - Insertion

insert(28);

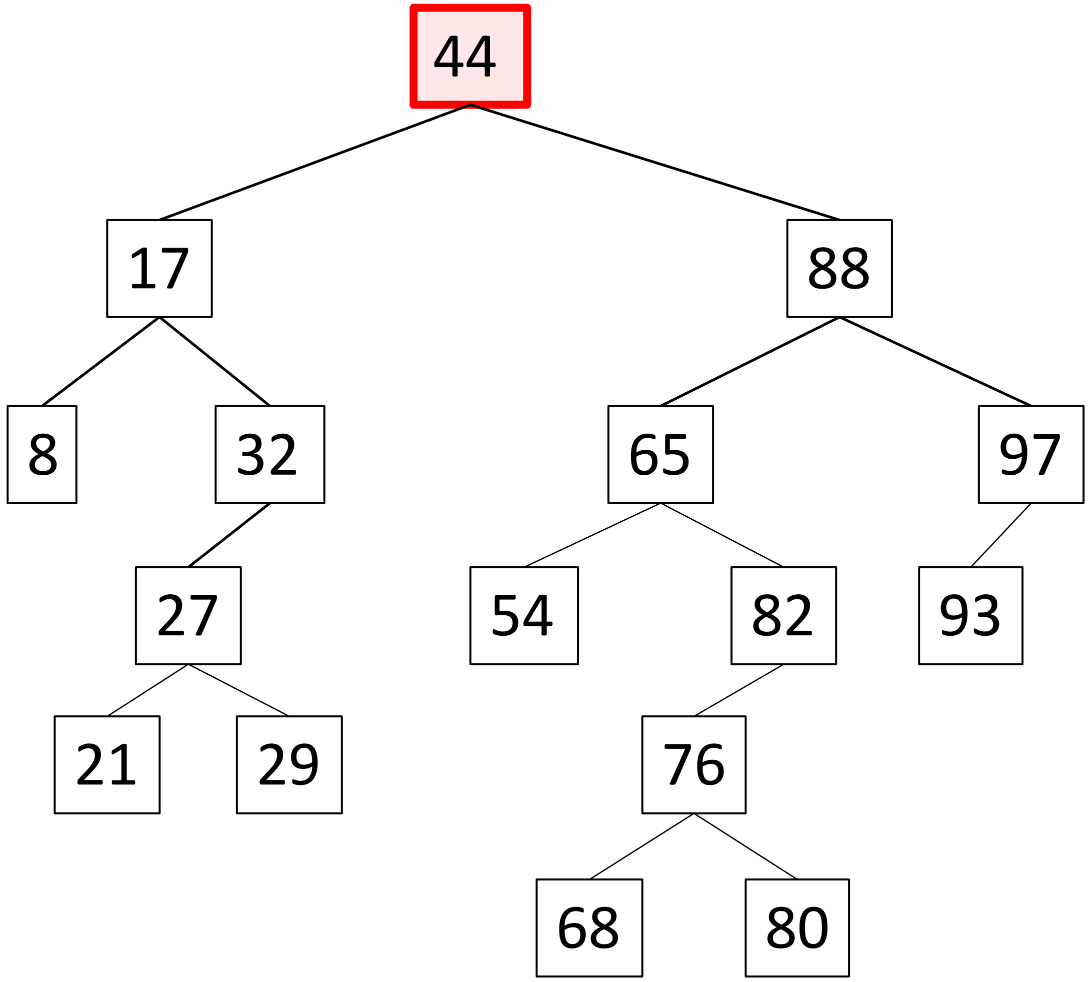
```
public void insert(int newValue) {
  if (root == null) {
    root = new Node(newValue);
  } else {
    Node currentNode = root;
    boolean placed = false;
    while (!placed) {







}
```



Binary Search Tree - Insertion

insert(28);

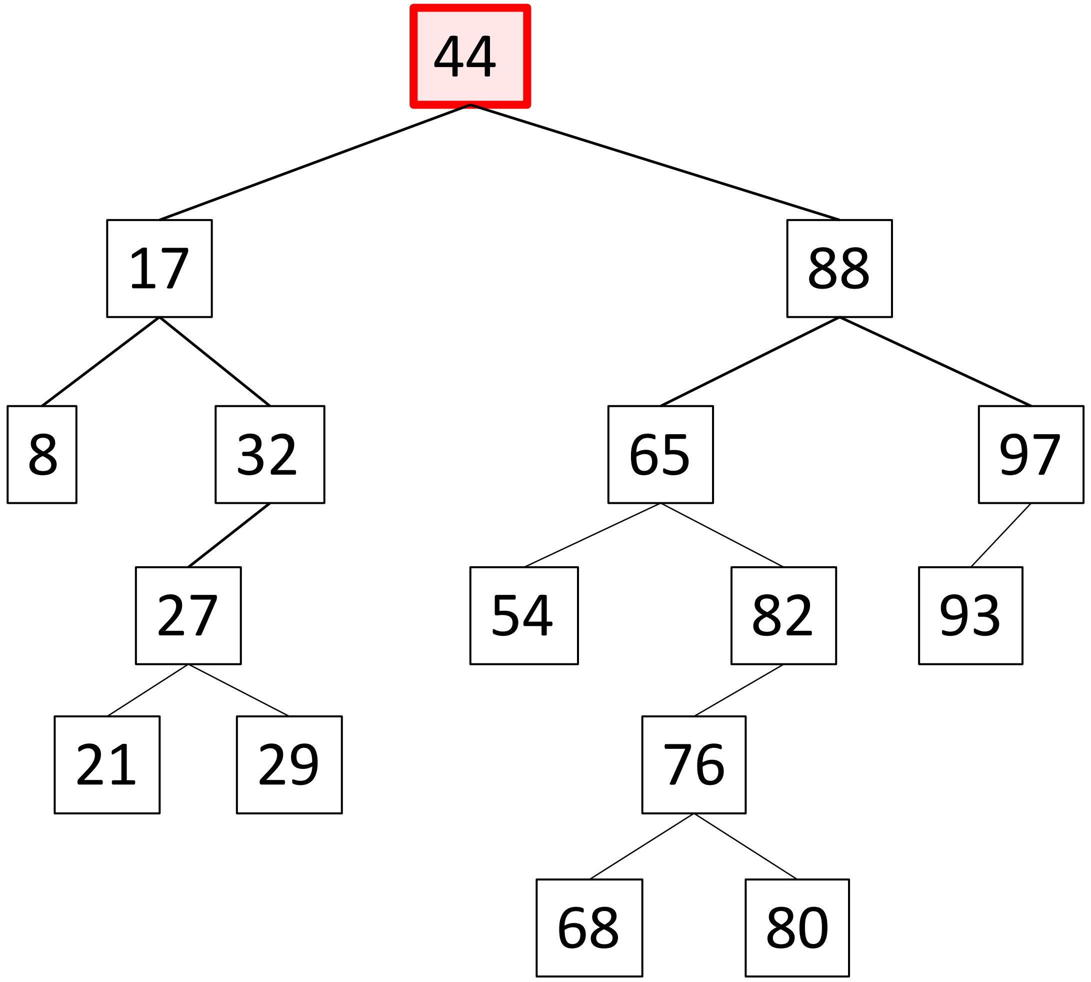
```
public void insert(int newValue) {
  if (root == null) {
    root = new Node(newValue);
  } else {
    Node currentNode = root;
    boolean placed = false;
    while (!placed) {
      if (newValue < currentNode.getValue()) {







      } else {







}
```



Binary Search Tree - Insertion

insert(28);

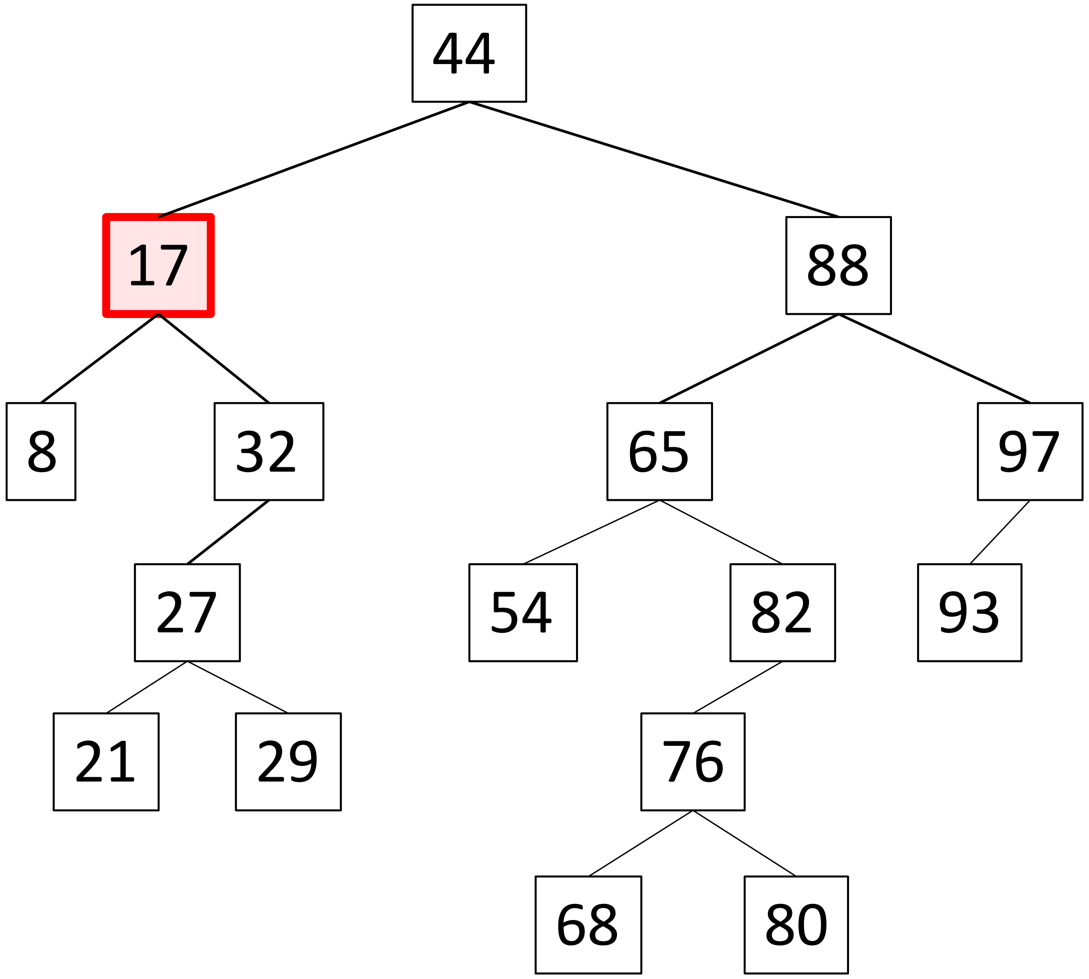
```
public void insert(int newValue) {
  if (root == null) {
    root = new Node(newValue);
  } else {
    Node currentNode = root;
    boolean placed = false;
    while (!placed) {
      if (newValue < currentNode.getValue()) {
        if (currentNode.getLeft() != null) {

        } else {



        }
      } else {






}
```



Binary Search Tree - Insertion

insert(28);

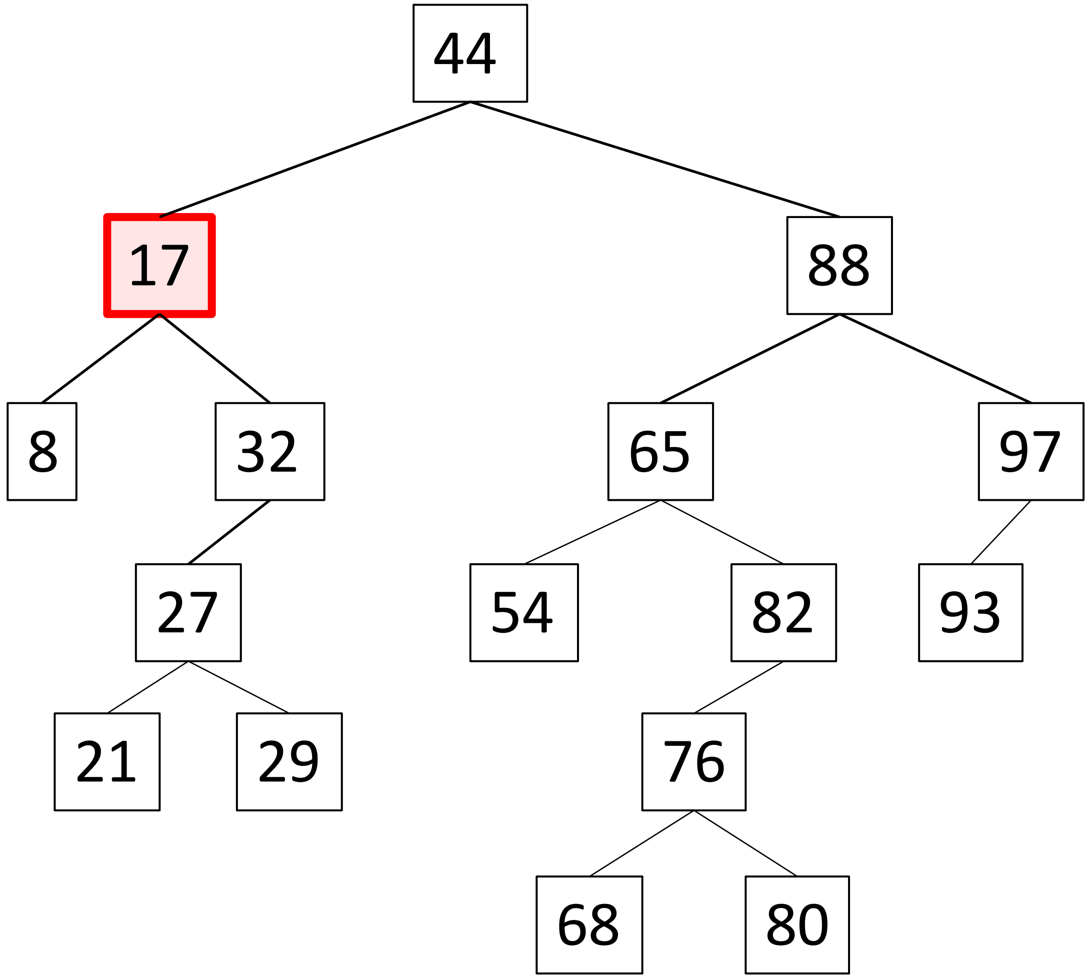
```
public void insert(int newValue) {
  if (root == null) {
    root = new Node(newValue);
  } else {
    Node currentNode = root;
    boolean placed = false;
    while (!placed) {
      if (newValue < currentNode.getValue()) {
        if (currentNode.getLeft() != null) {
          currentNode = currentNode.getLeft();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



        }
      } else {






}
```



Binary Search Tree - Insertion

insert(28);

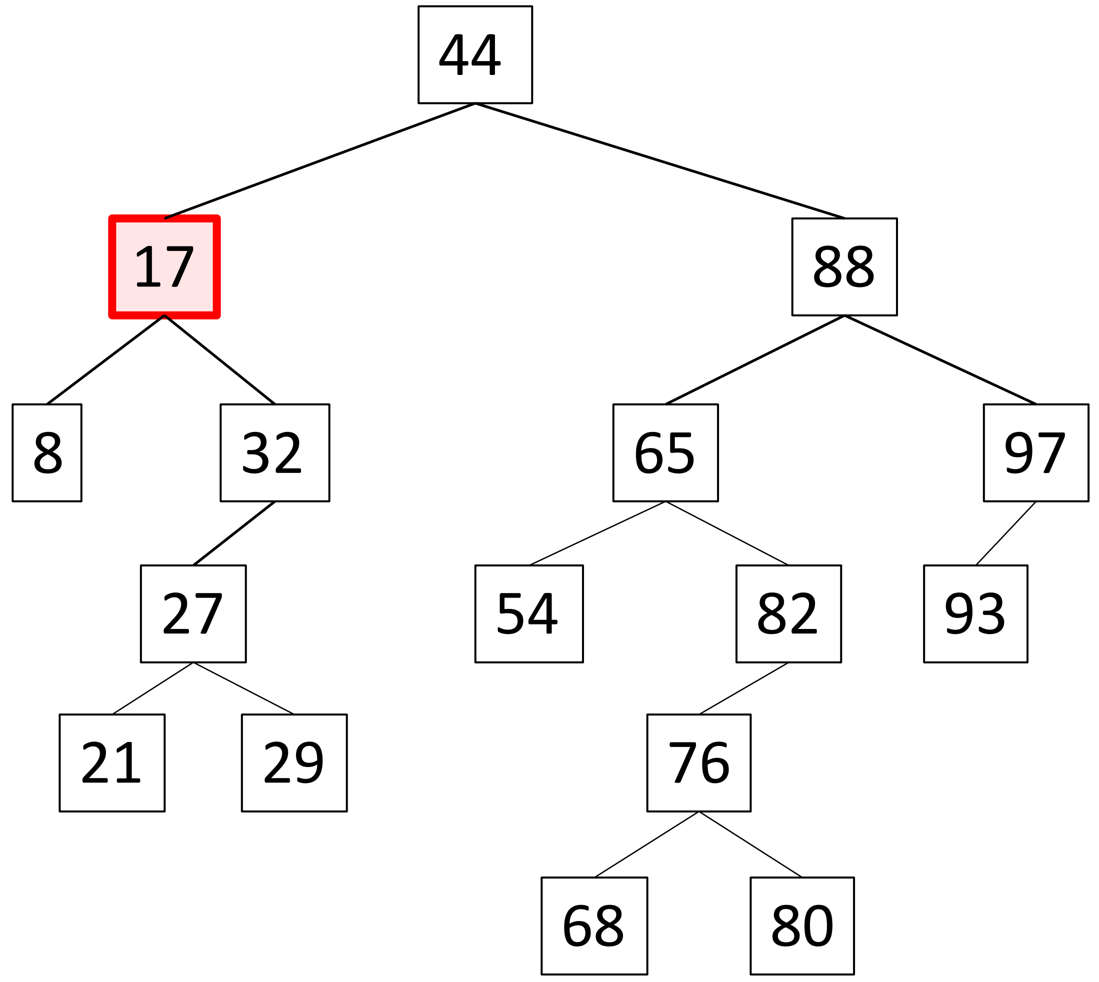
```
public void insert(int newValue) {
  if (root == null) {
    root = new Node(newValue);
  } else {
    Node currentNode = root;
    boolean placed = false;
    while (!placed) {
      if (newValue < currentNode.getValue()) {
        if (currentNode.getLeft() != null) {
          currentNode = currentNode.getLeft();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



        }
      } else {







}
```



Binary Search Tree - Insertion

insert(28);

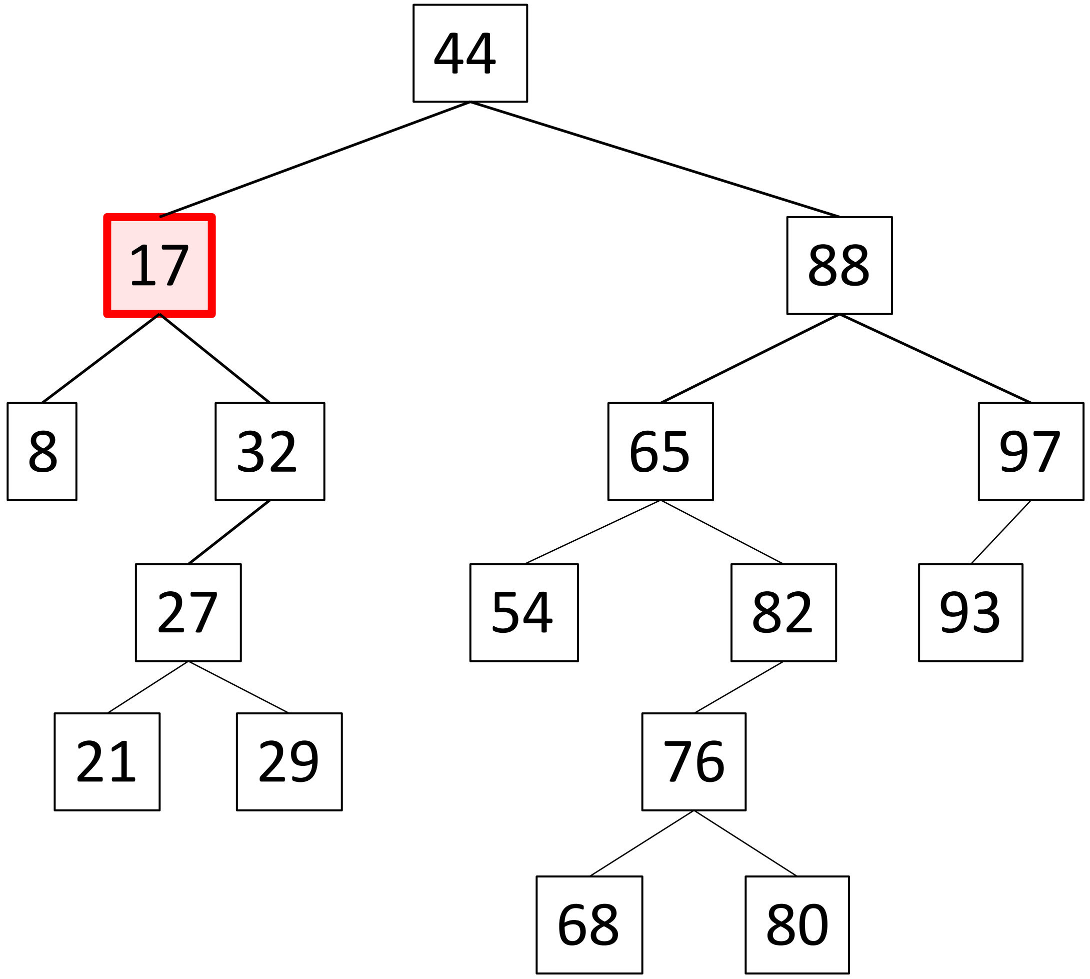
```
public void insert(int newValue) {
  if (root == null) {
    root = new Node(newValue);
  } else {
    Node currentNode = root;
    boolean placed = false;
    while (!placed) {
      if (newValue < currentNode.getValue()) {
        if (currentNode.getLeft() != null) {
          currentNode = currentNode.getLeft();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



        }
      } else {






}
```



Binary Search Tree - Insertion

insert(28);

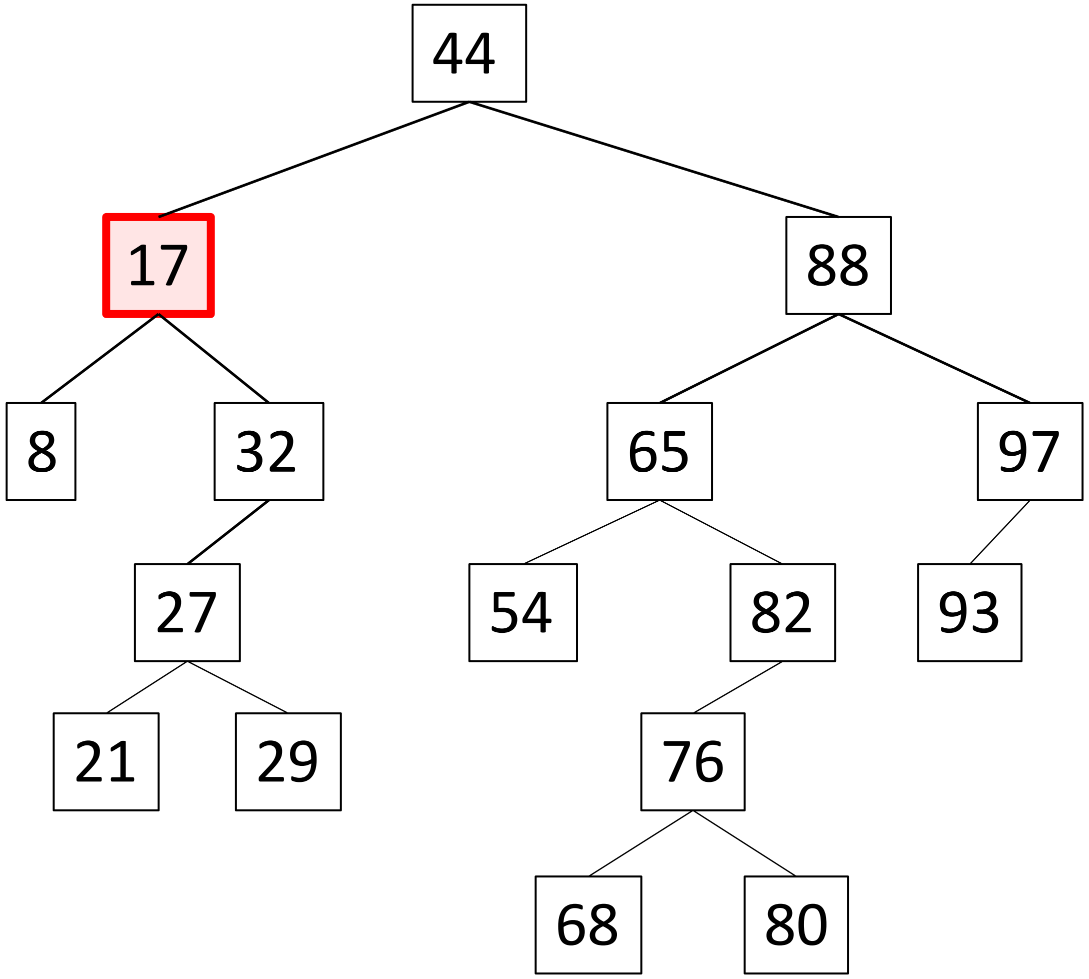
```
public void insert(int newValue) {
  if (root == null) {
    root = new Node(newValue);
  } else {
    Node currentNode = root;
    boolean placed = false;
    while (!placed) {
      if (newValue < currentNode.getValue()) {
        if (currentNode.getLeft() != null) {
          currentNode = currentNode.getLeft();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



        }
      } else {




}
```



Binary Search Tree - Insertion

insert(28);

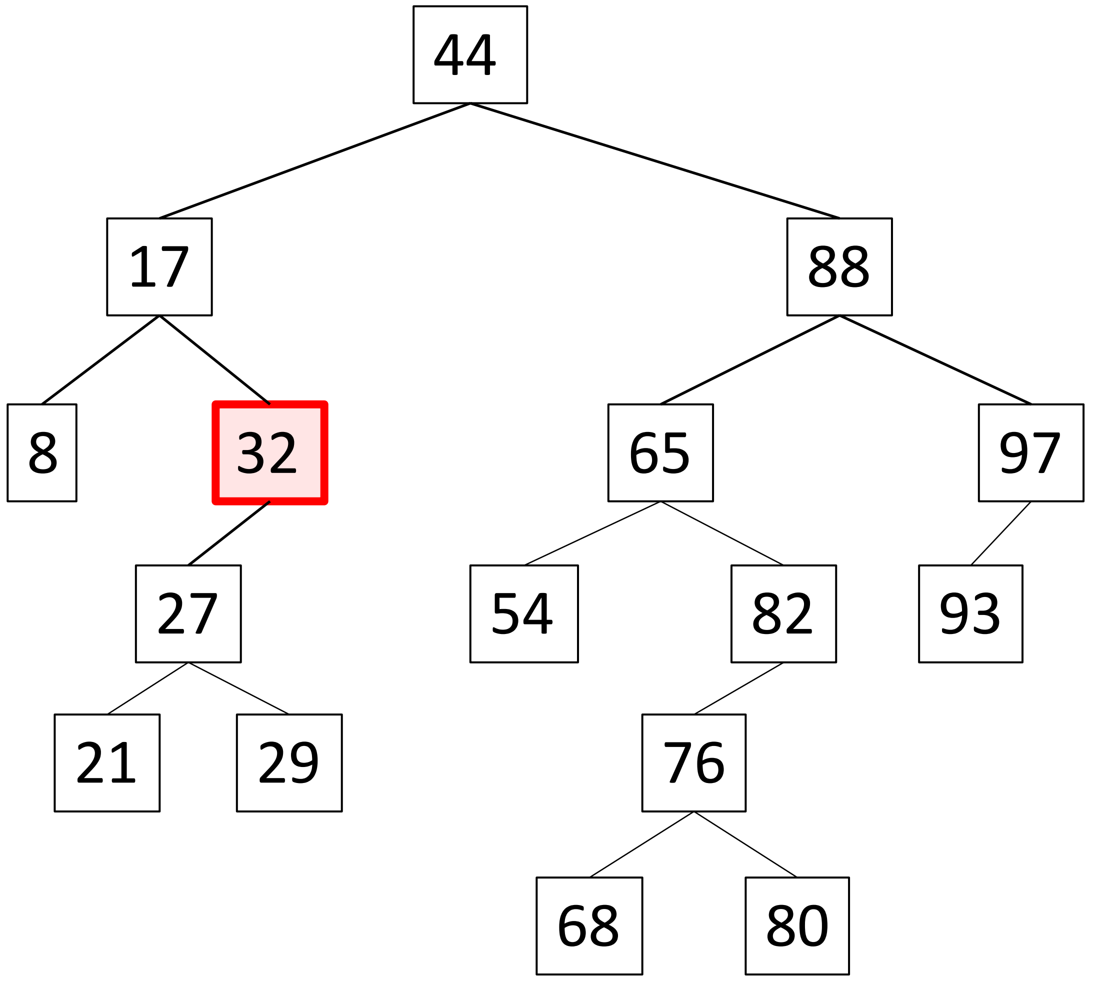
```
public void insert(int newValue) {
  if (root == null) {
    root = new Node(newValue);
  } else {
    Node currentNode = root;
    boolean placed = false;
    while (!placed) {
      if (newValue < currentNode.getValue()) {
        if (currentNode.getLeft() != null) {
          currentNode = currentNode.getLeft();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



        }
      } else {
        if (currentNode.getRight() != null) {

        } else {




}
```



Binary Search Tree - Insertion

`insert(28);`

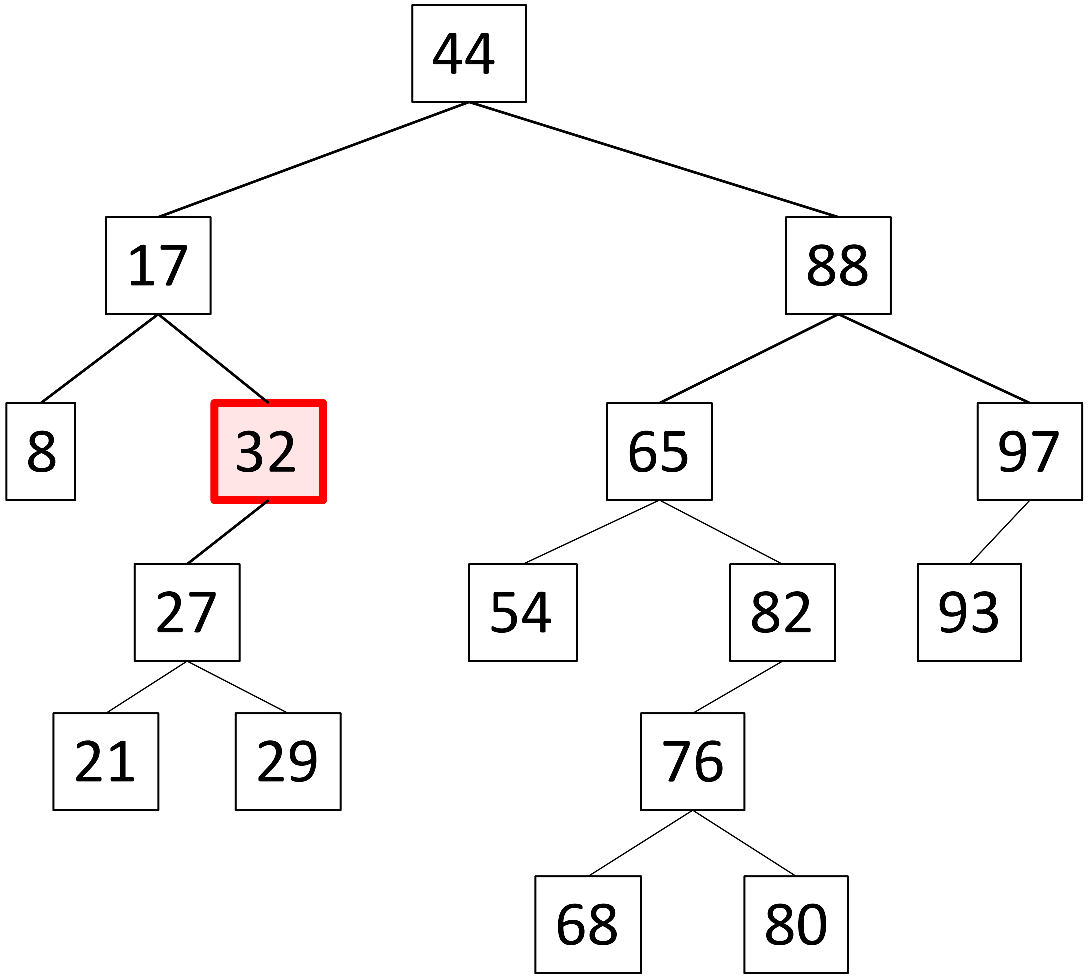
```
public void insert(int newValue) {
  if (root == null) {
    root = new Node(newValue);
  } else {
    Node currentNode = root;
    boolean placed = false;
    while (!placed) {
      if (newValue < currentNode.getValue()) {
        if (currentNode.getLeft() != null) {
          currentNode = currentNode.getLeft();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



        }
      } else {
        if (currentNode.getRight() != null) {
          currentNode = currentNode.getRight();
        } else {




}
```



Binary Search Tree - Insertion

insert(28);

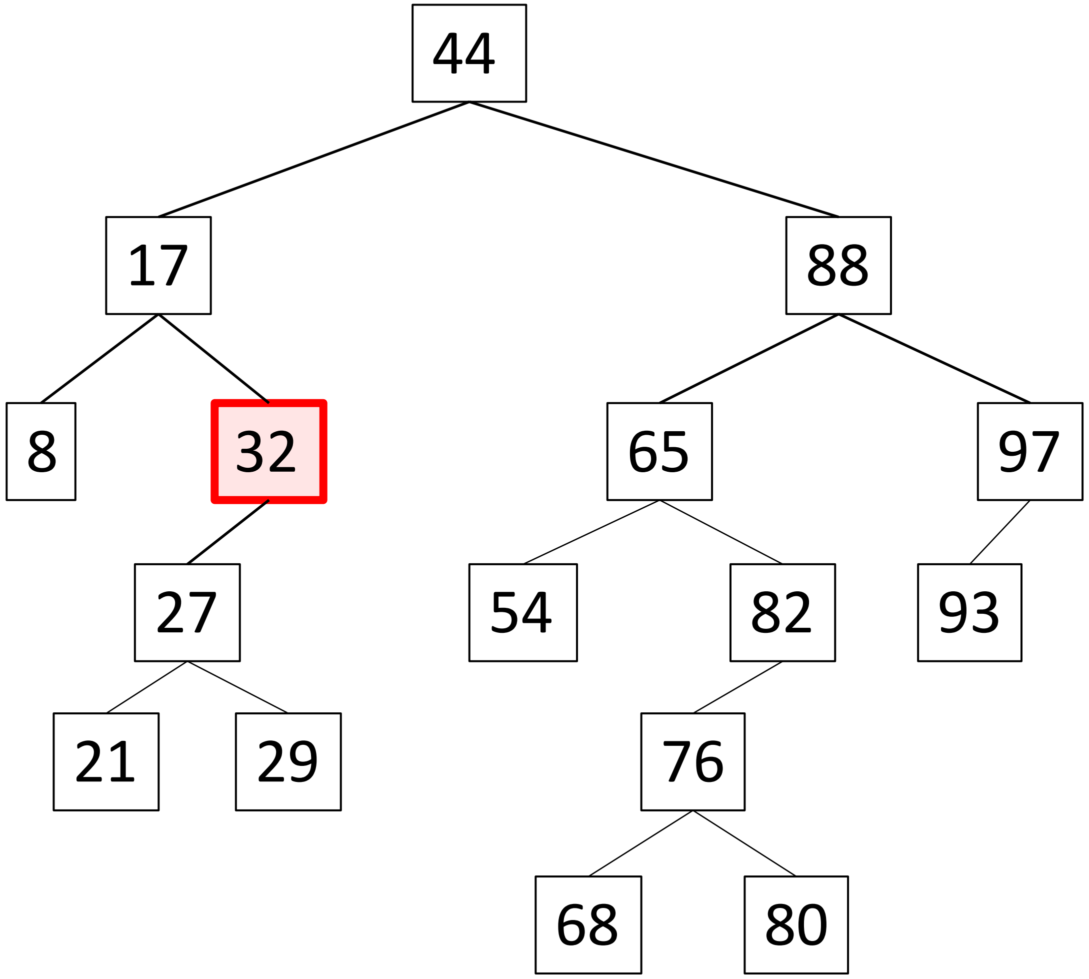
```
public void insert(int newValue) {
  if (root == null) {
    root = new Node(newValue);
  } else {
    Node currentNode = root;
    boolean placed = false;
    while (!placed) {
      if (newValue < currentNode.getValue()) {
        if (currentNode.getLeft() != null) {
          currentNode = currentNode.getLeft();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



        }
      } else {
        if (currentNode.getRight() != null) {
          currentNode = currentNode.getRight();
        } else {




}
```



Binary Search Tree - Insertion

insert(28);

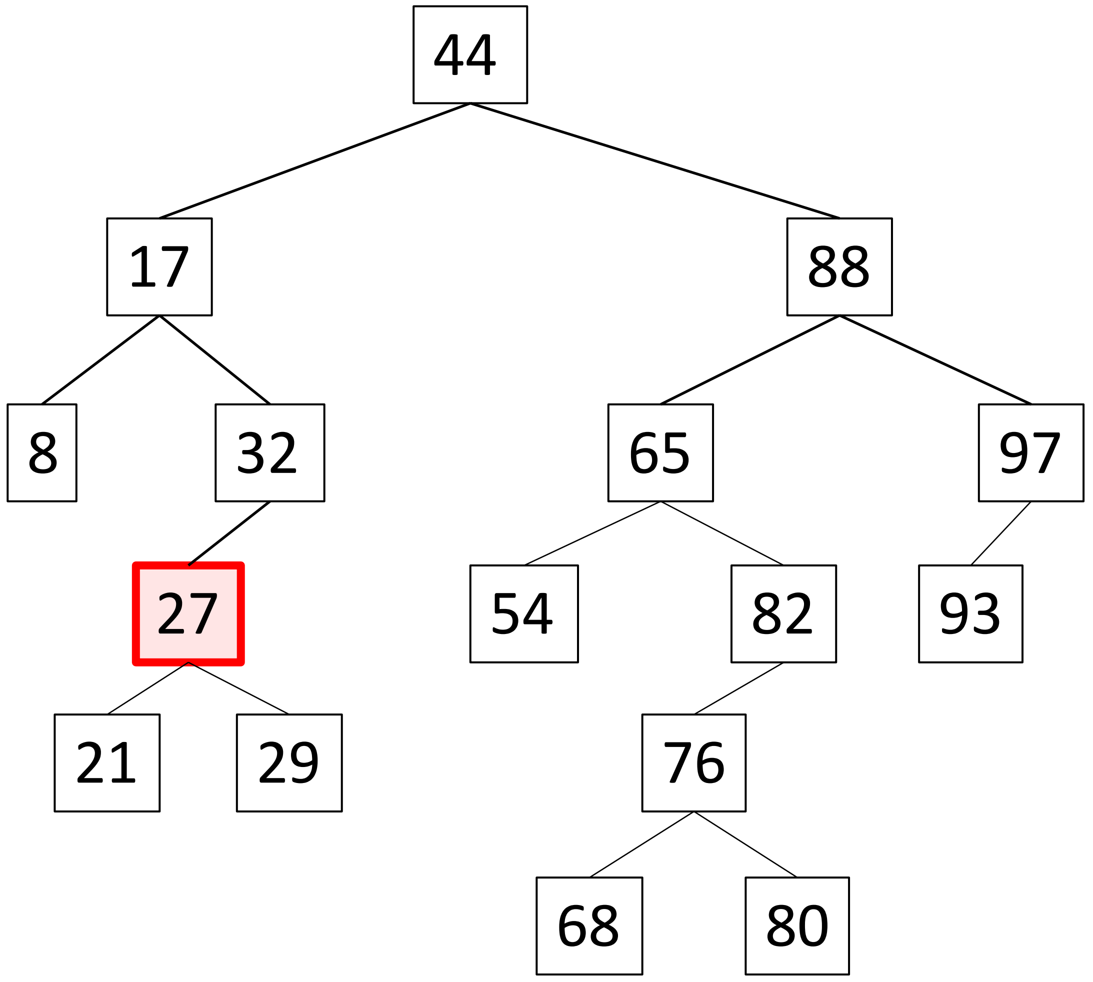
```
public void insert(int newValue) {
  if (root == null) {
    root = new Node(newValue);
  } else {
    Node currentNode = root;
    boolean placed = false;
    while (!placed) {
      if (newValue < currentNode.getValue()) {
        if (currentNode.getLeft() != null) {
          currentNode = currentNode.getLeft();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



        }
      } else {
        if (currentNode.getRight() != null) {
          currentNode = currentNode.getRight();
        } else {




}
```



Binary Search Tree - Insertion

insert(28);

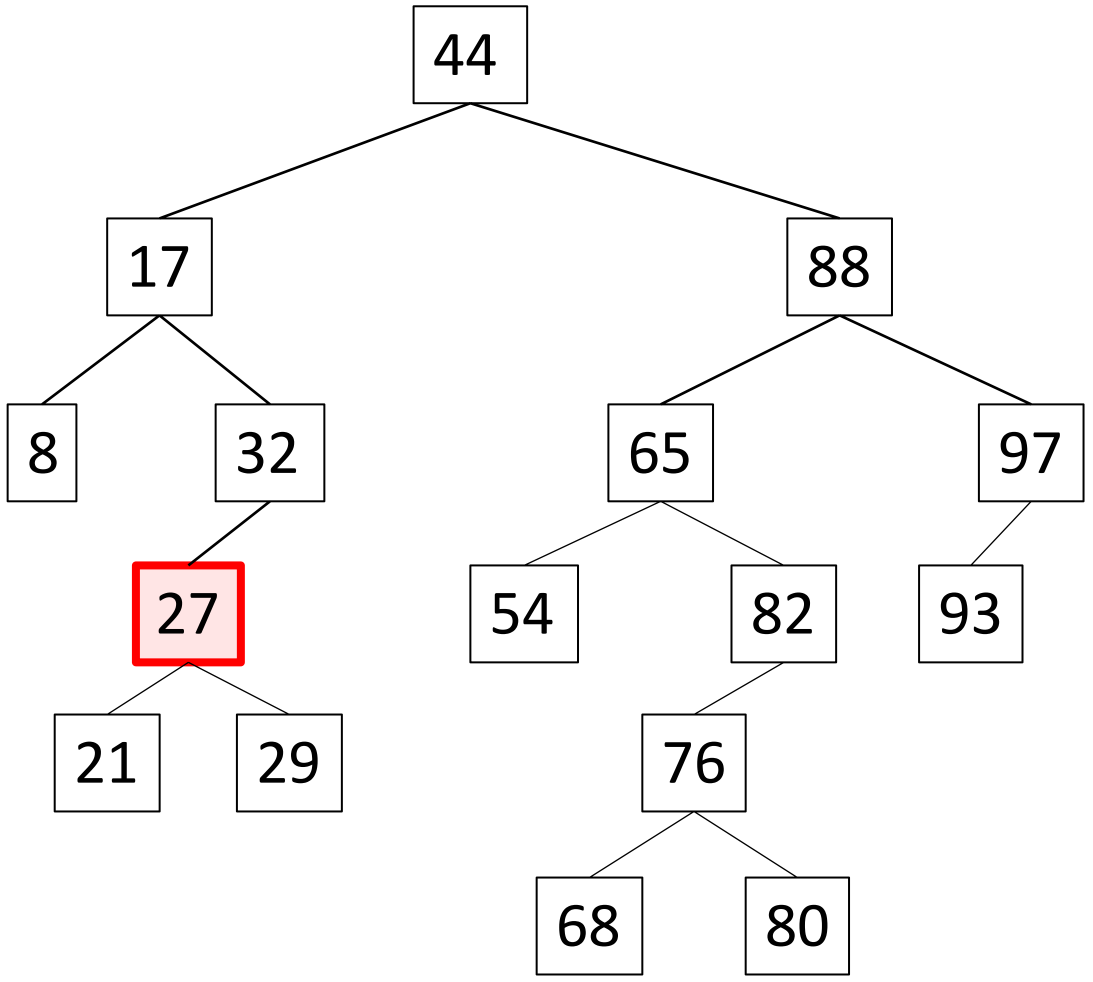
```
public void insert(int newValue) {
  if (root == null) {
    root = new Node(newValue);
  } else {
    Node currentNode = root;
    boolean placed = false;
    while (!placed) {
      if (newValue < currentNode.getValue()) {
        if (currentNode.getLeft() != null) {
          currentNode = currentNode.getLeft();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



        }
      } else {
        if (currentNode.getRight() != null) {
          currentNode = currentNode.getRight();
        } else {




}
```



Binary Search Tree - Insertion

insert(28);

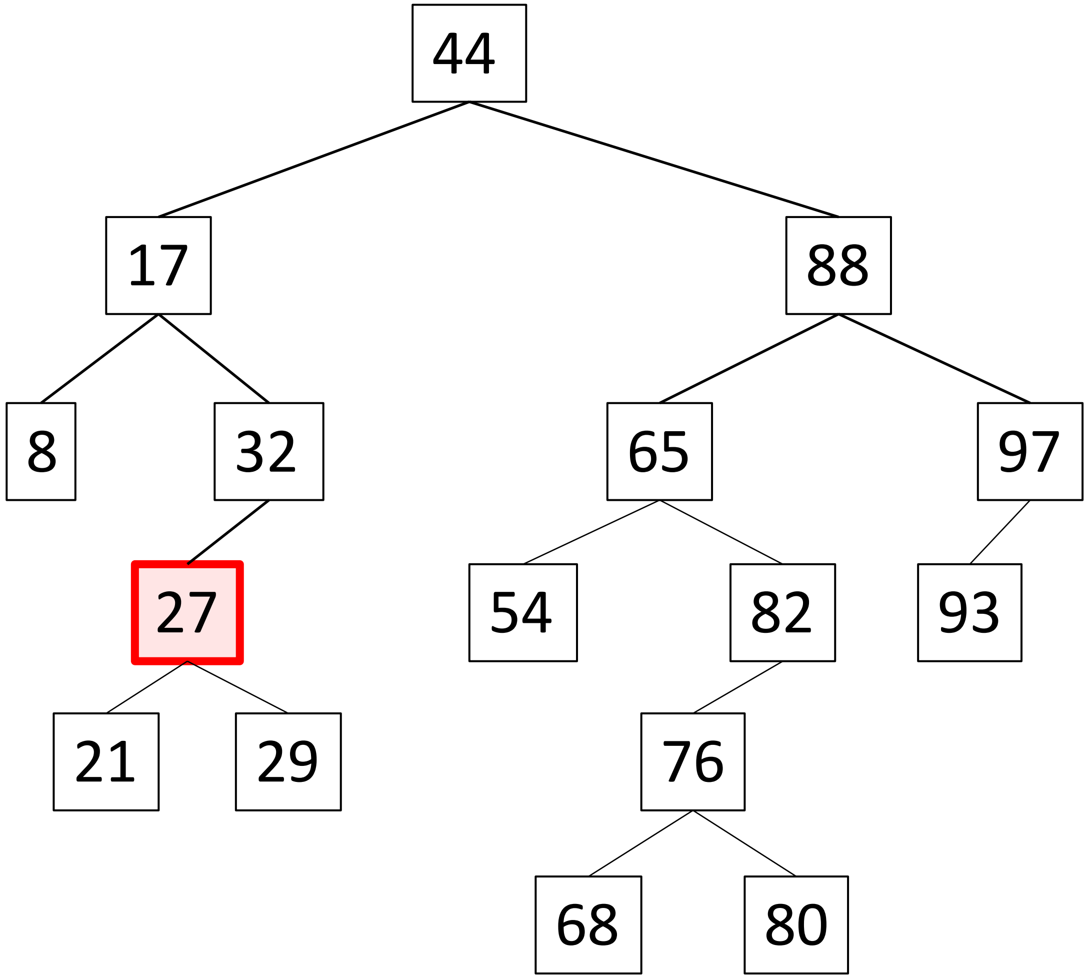
```
public void insert(int newValue) {
  if (root == null) {
    root = new Node(newValue);
  } else {
    Node currentNode = root;
    boolean placed = false;
    while (!placed) {
      if (newValue < currentNode.getValue()) {
        if (currentNode.getLeft() != null) {
          currentNode = currentNode.getLeft();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



        }
      } else {
        if (currentNode.getRight() != null) {
          currentNode = currentNode.getRight();
        } else {




}
```



Binary Search Tree - Insertion

insert(28);

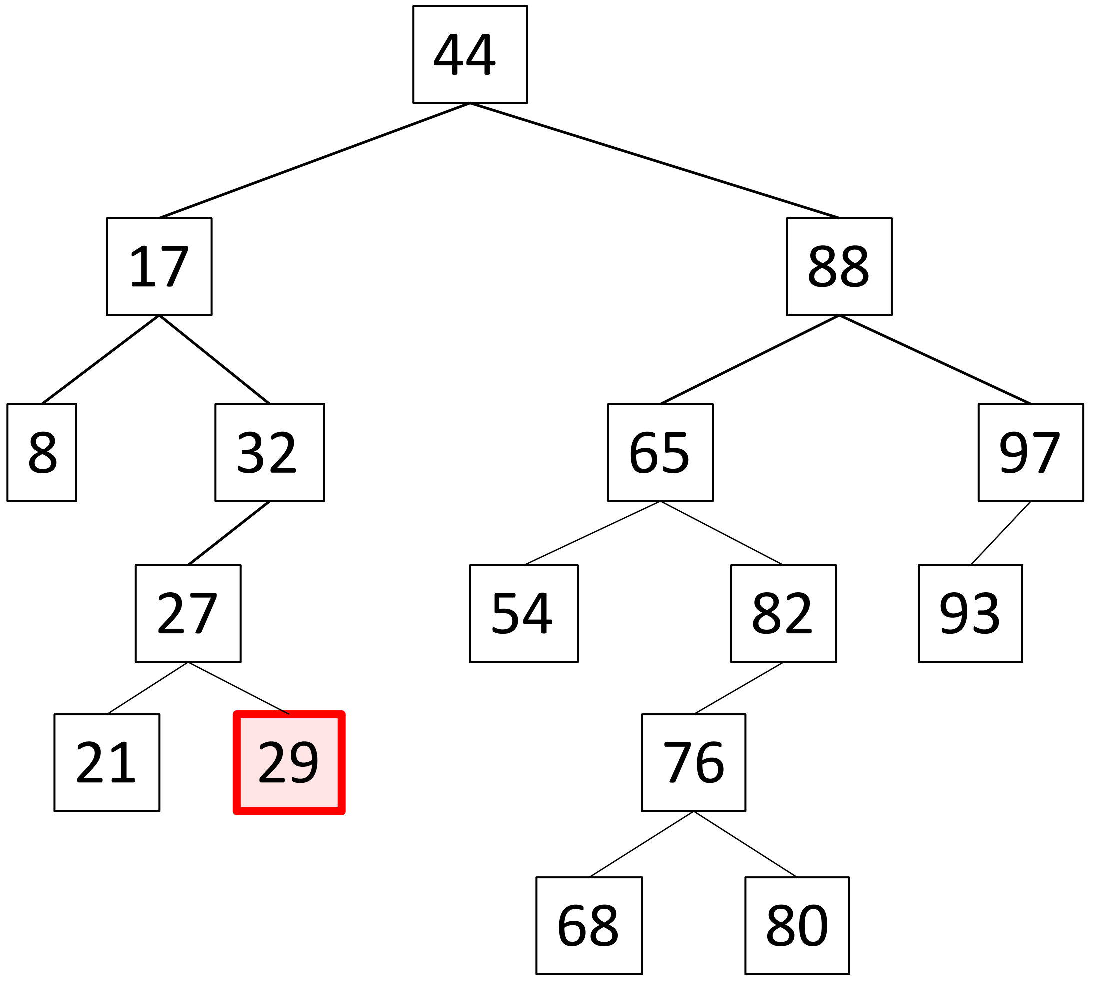
```
public void insert(int newValue) {
  if (root == null) {
    root = new Node(newValue);
  } else {
    Node currentNode = root;
    boolean placed = false;
    while (!placed) {
      if (newValue < currentNode.getValue()) {
        if (currentNode.getLeft() != null) {
          currentNode = currentNode.getLeft();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



        }
      } else {
        if (currentNode.getRight() != null) {
          currentNode = currentNode.getRight();
        } else {




}
```



Binary Search Tree - Insertion

insert(28);

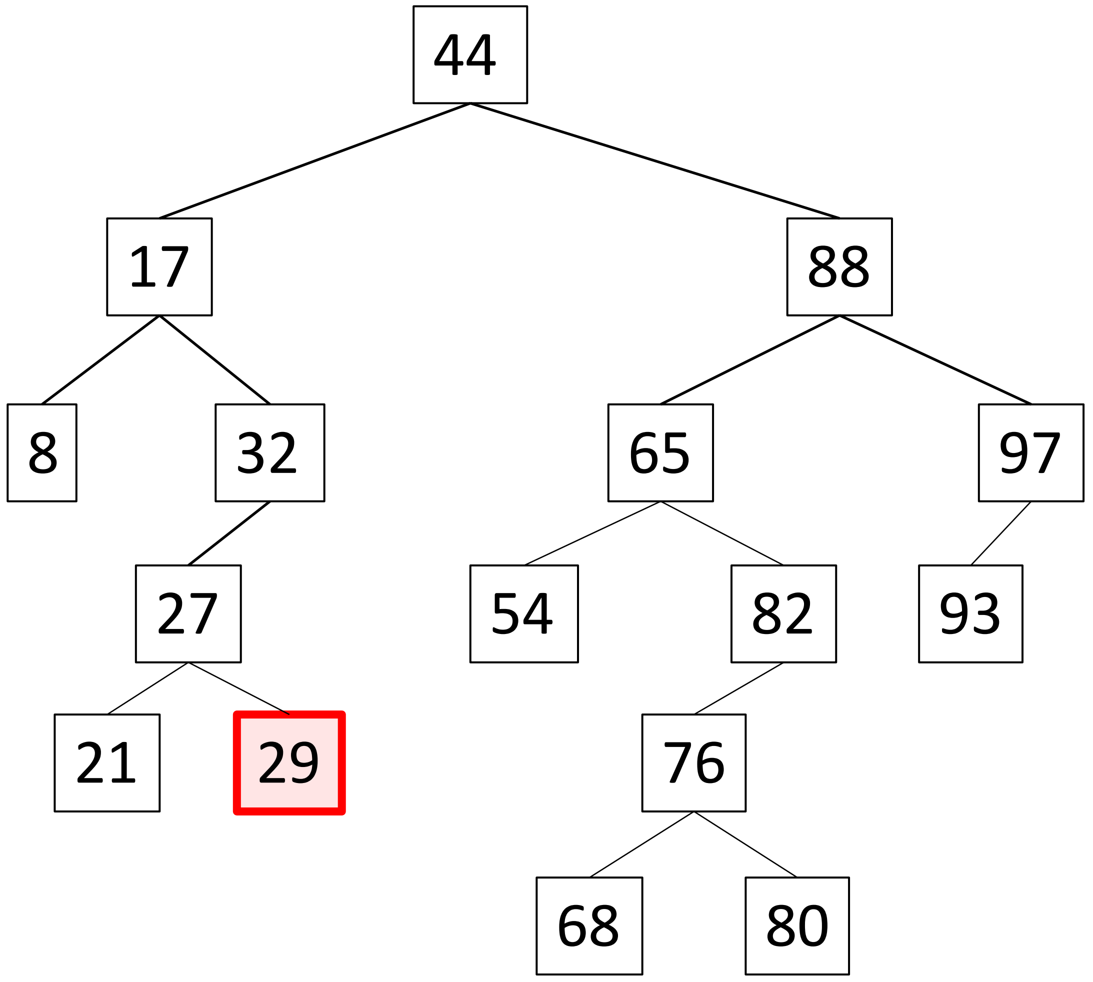
```
public void insert(int newValue) {
  if (root == null) {
    root = new Node(newValue);
  } else {
    Node currentNode = root;
    boolean placed = false;
    while (!placed) {
      if (newValue < currentNode.getValue()) {
        if (currentNode.getLeft() != null) {
          currentNode = currentNode.getLeft();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



        }
      } else {
        if (currentNode.getRight() != null) {
          currentNode = currentNode.getRight();
        } else {



}
```



Binary Search Tree - Insertion

insert(28);

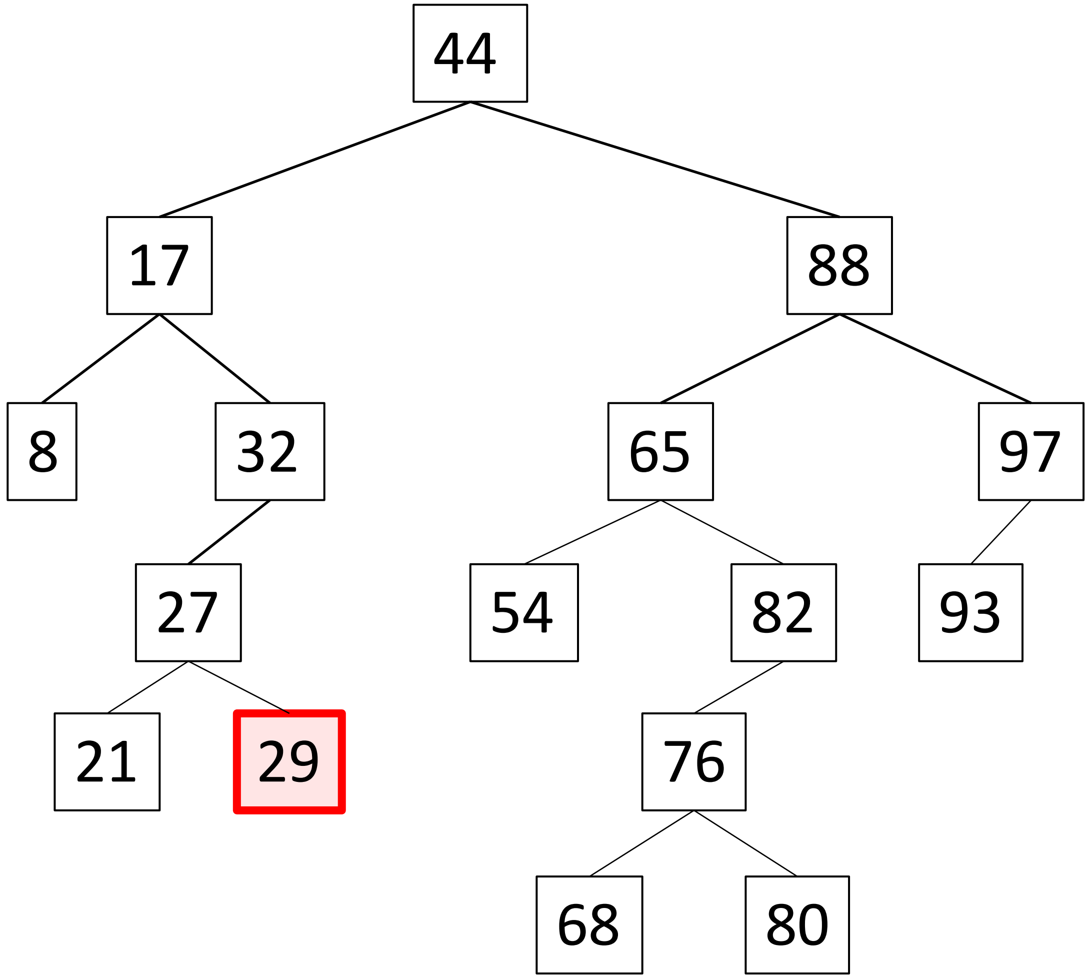
```
public void insert(int newValue) {
  if (root == null) {
    root = new Node(newValue);
  } else {
    Node currentNode = root;
    boolean placed = false;
    while (!placed) {
      if (newValue < currentNode.getValue()) {
        if (currentNode.getLeft() != null) {
          currentNode = currentNode.getLeft();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



        }
      } else {
        if (currentNode.getRight() != null) {
          currentNode = currentNode.getRight();
        } else {




}
```



Binary Search Tree - Insertion

insert(28);

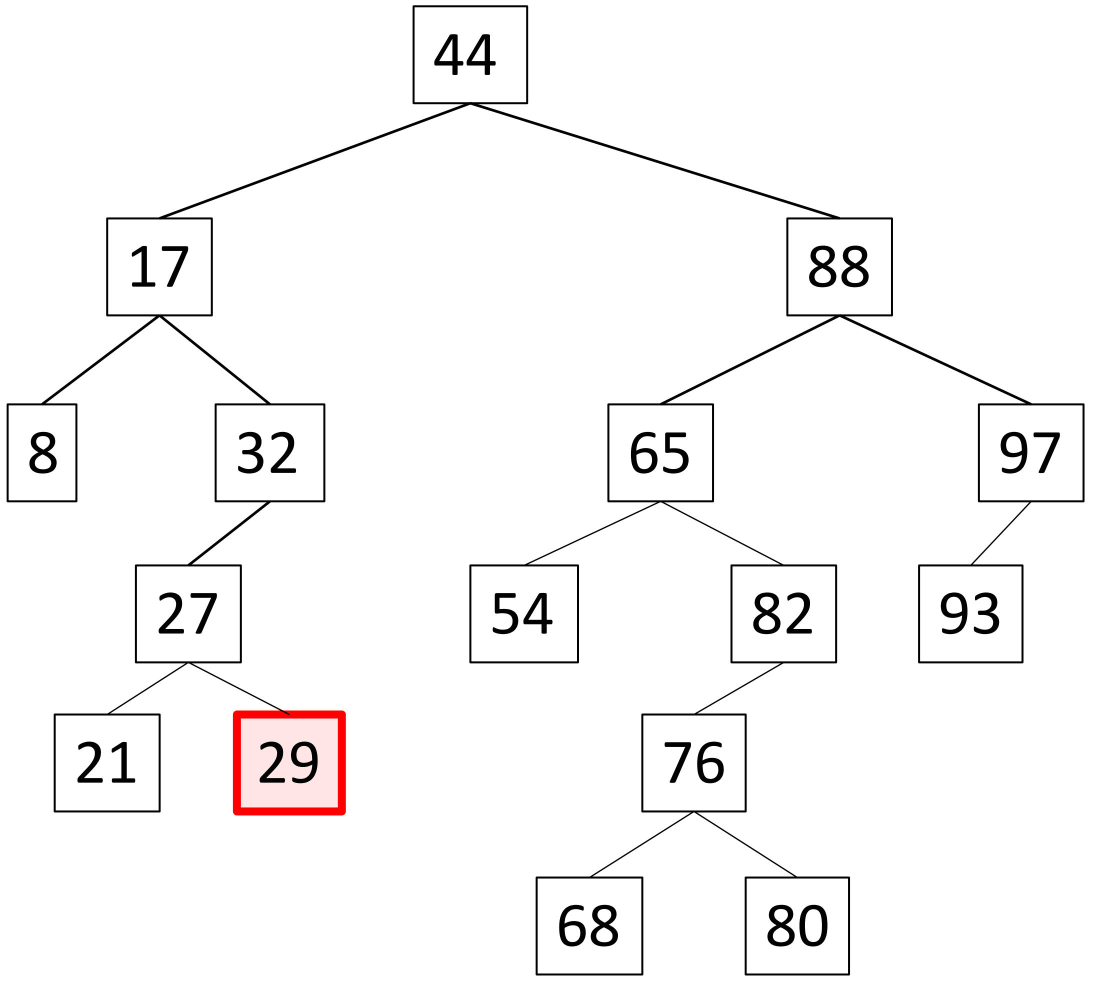
```
public void insert(int newValue) {
  if (root == null) {
    root = new Node(newValue);
  } else {
    Node currentNode = root;
    boolean placed = false;
    while (!placed) {
      if (newValue < currentNode.getValue()) {
        if (currentNode.getLeft() != null) {
          currentNode = currentNode.getLeft();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



        }
      } else {
        if (currentNode.getRight() != null) {
          currentNode = currentNode.getRight();
        } else {




}
```



Binary Search Tree - Insertion

insert(28);

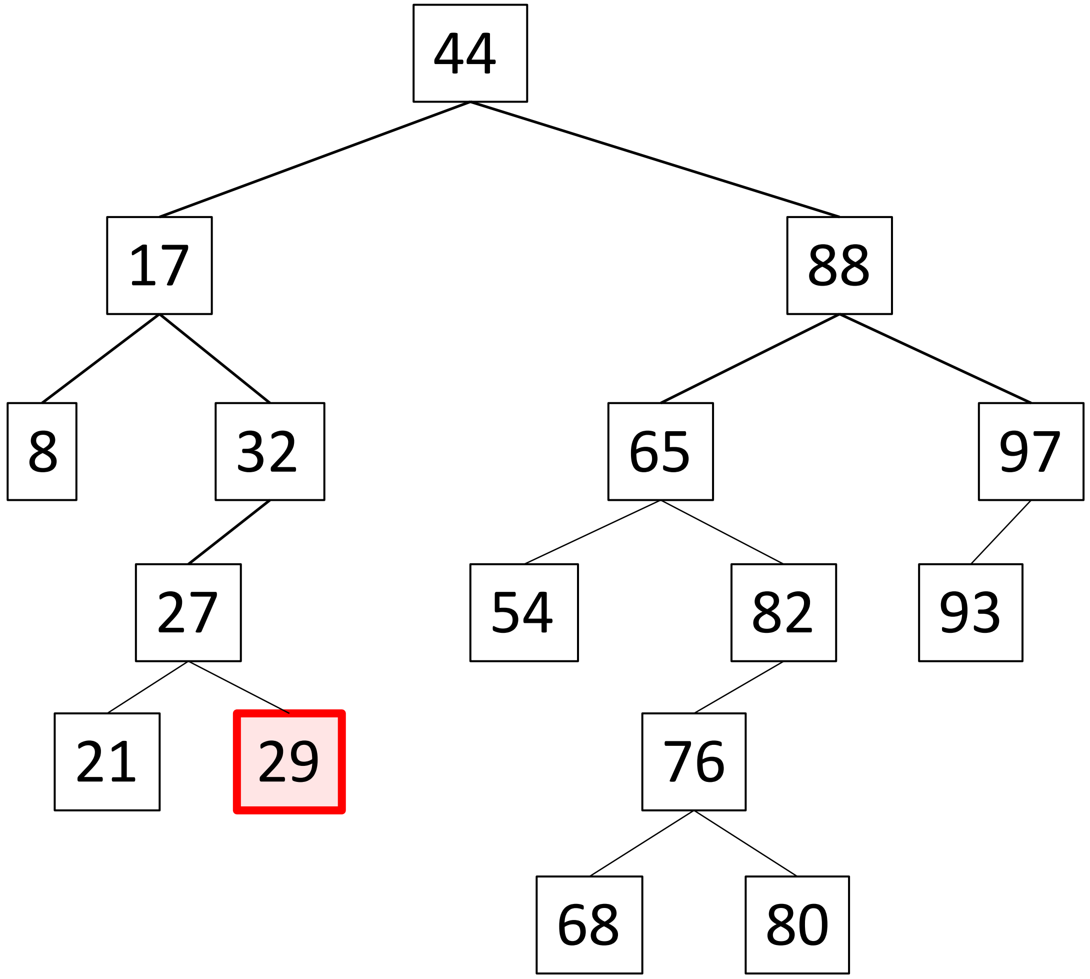
```
public void insert(int newValue) {
  if (root == null) {
    root = new Node(newValue);
  } else {
    Node currentNode = root;
    boolean placed = false;
    while (!placed) {
      if (newValue < currentNode.getValue()) {
        if (currentNode.getLeft() != null) {
          currentNode = currentNode.getLeft();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



        }
      } else {
        if (currentNode.getRight() != null) {
          currentNode = currentNode.getRight();
        } else {




}
```



Binary Search Tree - Insertion

insert(28);

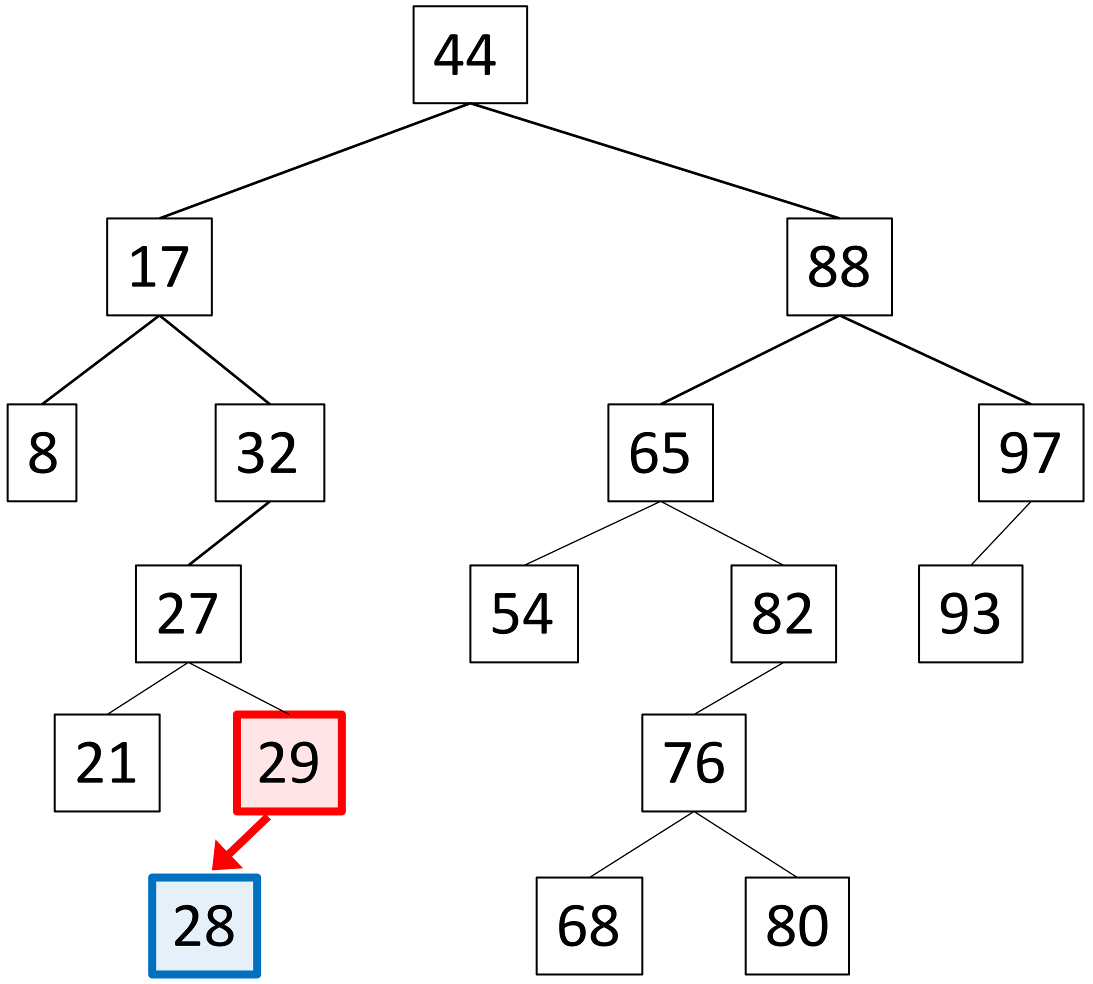
```
public void insert(int newValue) {
  if (root == null) {
    root = new Node(newValue);
  } else {
    Node currentNode = root;
    boolean placed = false;
    while (!placed) {
      if (newValue < currentNode.getValue()) {
        if (currentNode.getLeft() != null) {
          currentNode = currentNode.getLeft();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



        }
      } else {
        if (currentNode.getRight() != null) {
          currentNode = currentNode.getRight();
        } else {



}
```



Binary Search Tree - Insertion

insert(28);

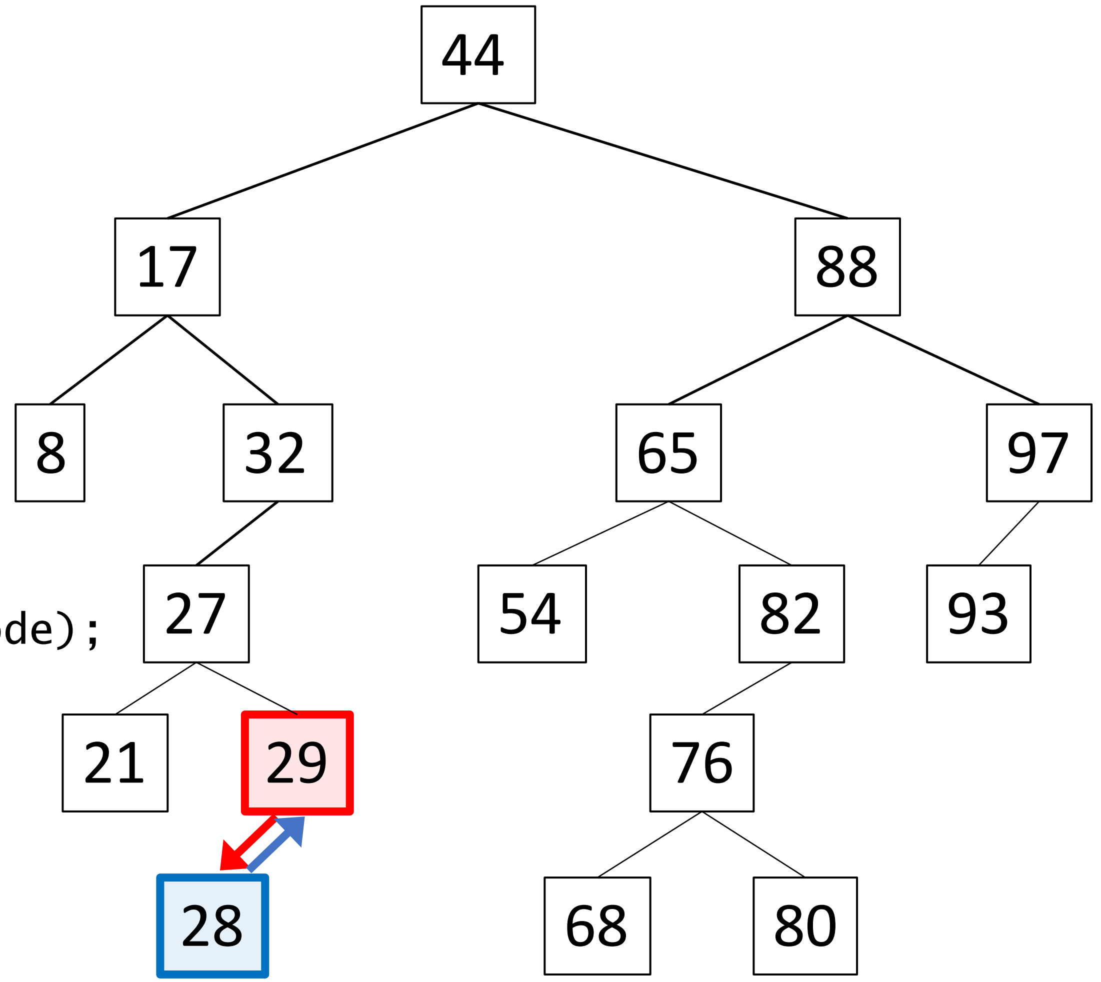
```
public void insert(int newValue) {
  if (root == null) {
    root = new Node(newValue);
  } else {
    Node currentNode = root;
    boolean placed = false;
    while (!placed) {
      if (newValue < currentNode.getValue()) {
        if (currentNode.getLeft() != null) {
          currentNode = currentNode.getLeft();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


        }
      } else {
        if (currentNode.getRight() != null) {
          currentNode = currentNode.getRight();
        } else {



}
```



Binary Search Tree - Insertion

insert(28);

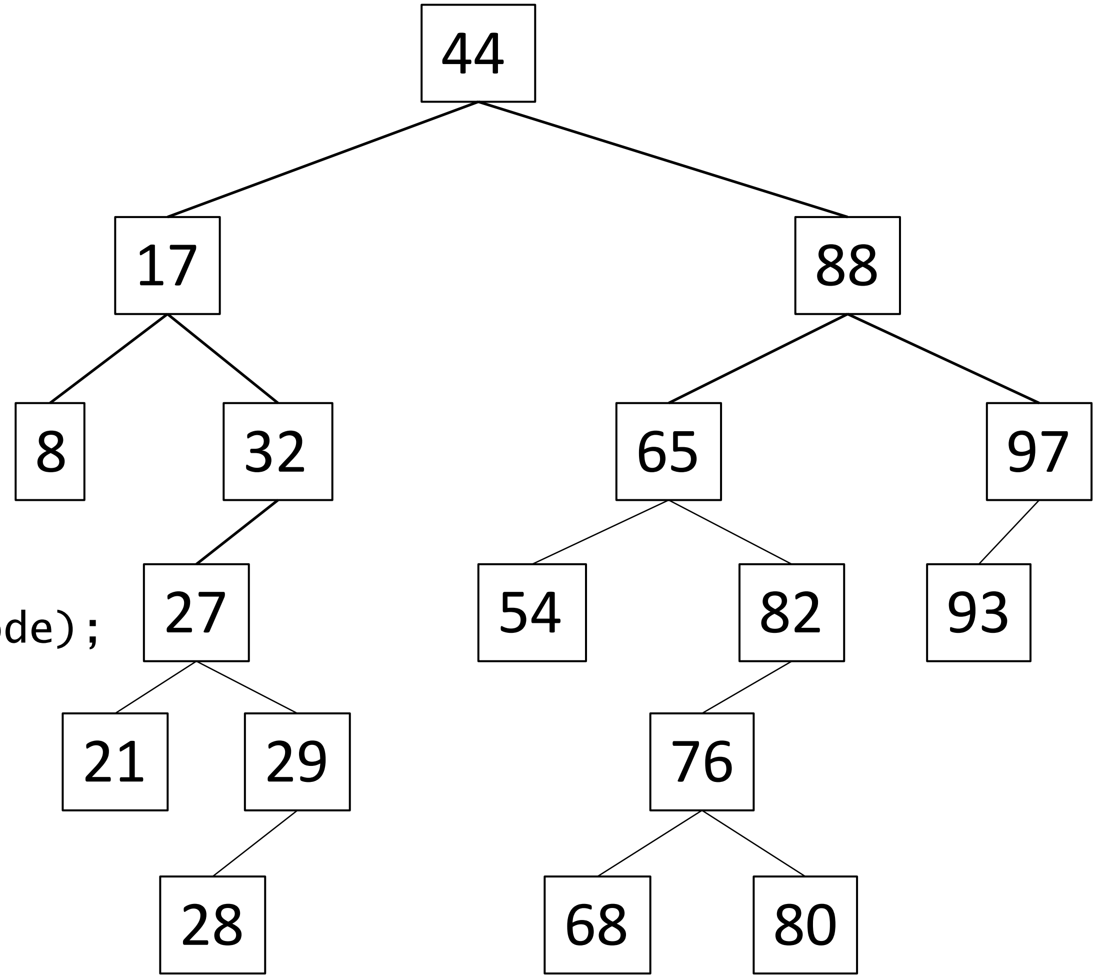
```
public void insert(int newValue) {
  if (root == null) {
    root = new Node(newValue);
  } else {
    Node currentNode = root;
    boolean placed = false;
    while (!placed) {
      if (newValue < currentNode.getValue()) {
        if (currentNode.getLeft() != null) {
          currentNode = currentNode.getLeft();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

        }
      } else {
        if (currentNode.getRight() != null) {
          currentNode = currentNode.getRight();
        } else {



}
```



Binary Search Tree - Insertion

insert(28);

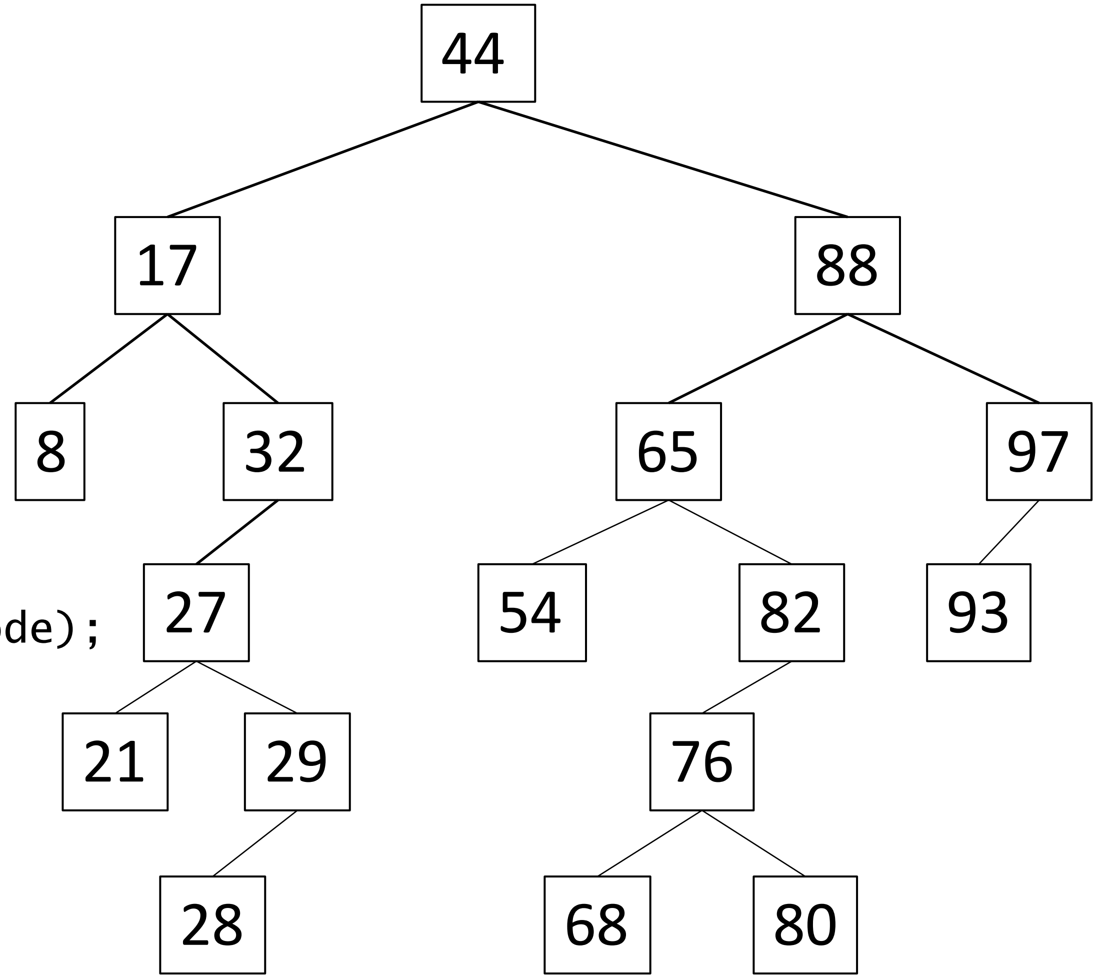
```
public void insert(int newValue) {
  if (root == null) {
    root = new Node(newValue);
  } else {
    Node currentNode = root;
    boolean placed = false;
    while (!placed) {
      if (newValue < currentNode.getValue()) {
        if (currentNode.getLeft() != null) {
          currentNode = currentNode.getLeft();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



}
```



Binary Search Tree - Insertion

insert(28);

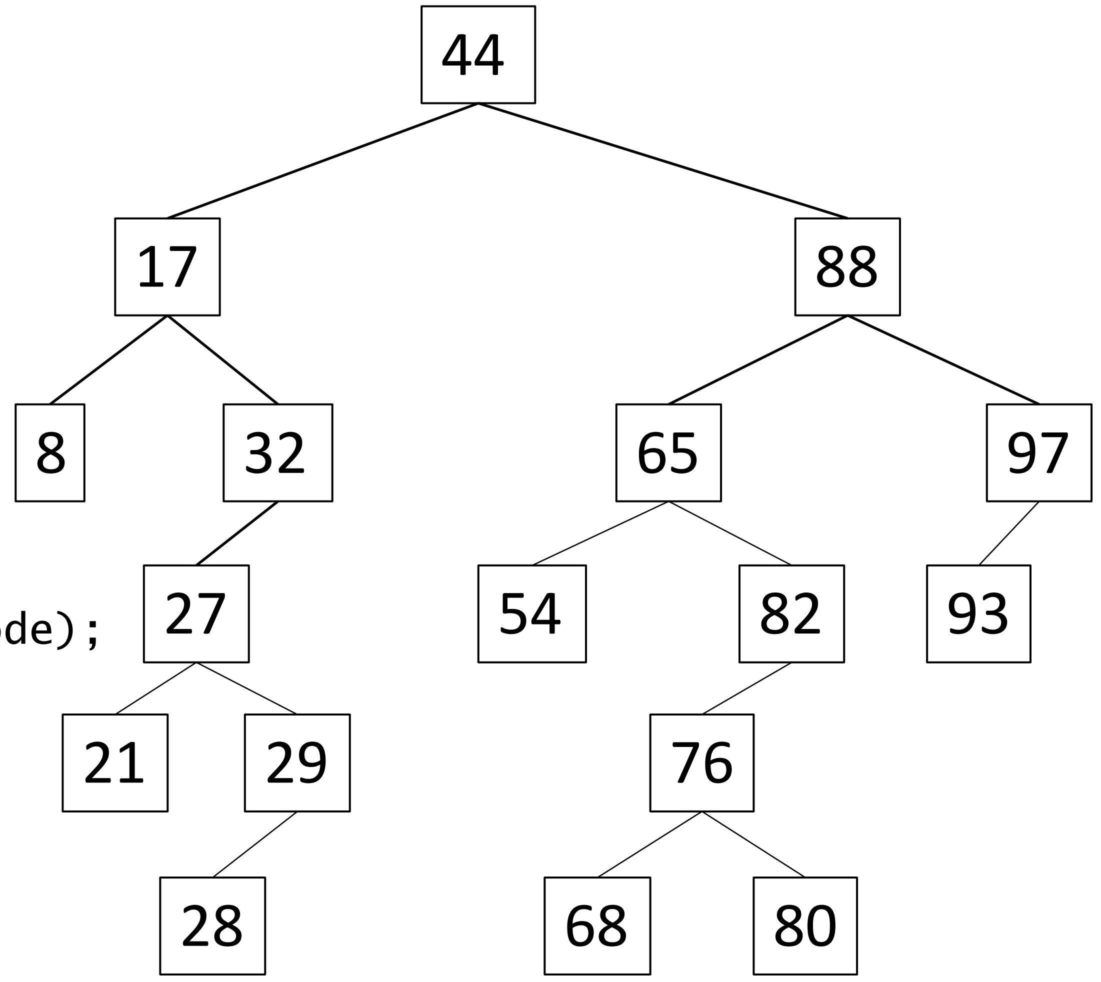
```
public void insert(int newValue) {
  if (root == null) {
    root = new Node(newValue);
  } else {
    Node currentNode = root;
    boolean placed = false;
    while (!placed) {
      if (newValue < currentNode.getValue()) {
        if (currentNode.getLeft() != null) {
          currentNode = currentNode.getLeft();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




}
```



Binary Search Tree - Insertion

`insert(28);`

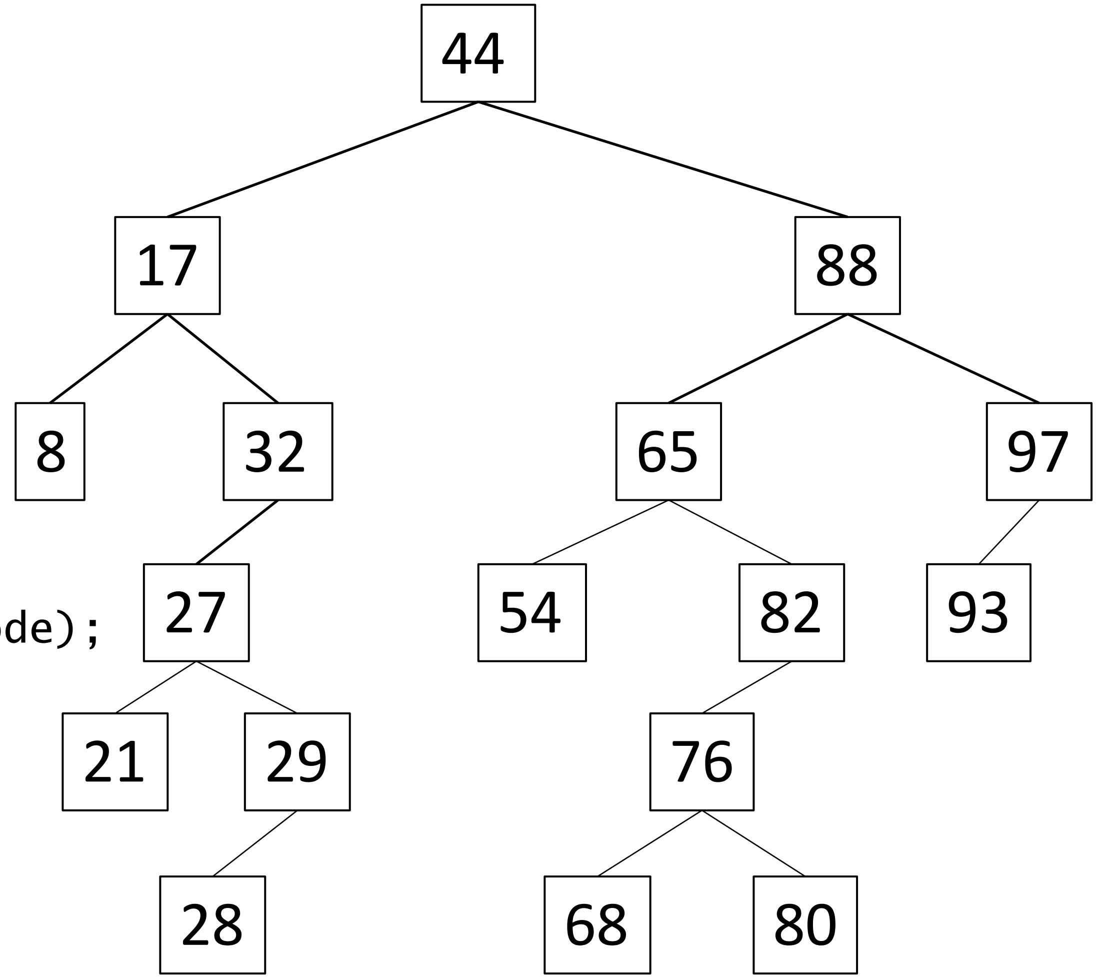
```
public void insert(int newValue) {
  if (root == null) {
    root = new Node(newValue);
  } else {
    Node currentNode = root;
    boolean placed = false;
    while (!placed) {
      if (newValue < currentNode.getValue()) {
        if (currentNode.getLeft() != null) {
          currentNode = currentNode.getLeft();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

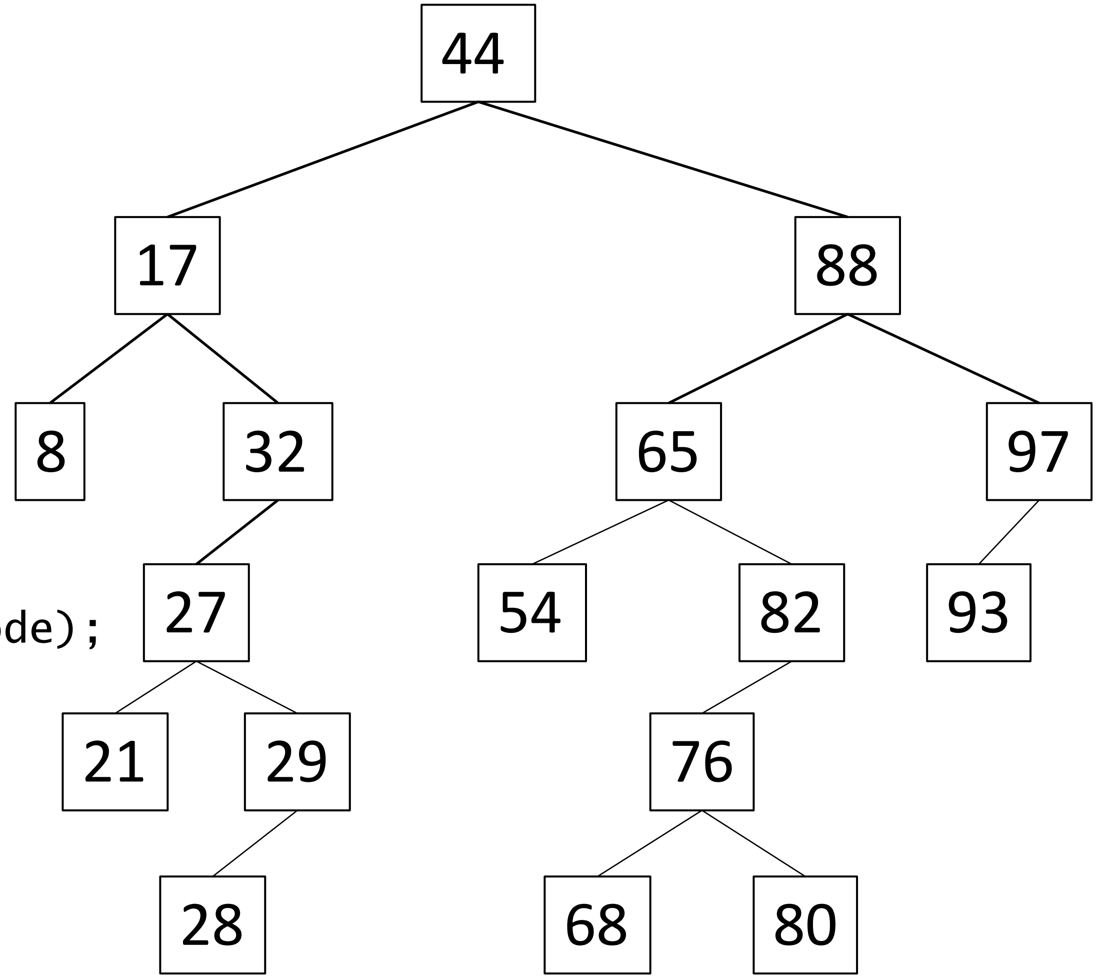
```
public void insert(int newValue) {
  if (root == null) {
    root = new Node(newValue);
  } else {
    Node currentNode = root;
    boolean placed = false;
    while (!placed) {
      if (newValue < currentNode.getValue()) {
        if (currentNode.getLeft() != null) {
          currentNode = currentNode.getLeft();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

}
```



Binary Search Tree - Insertion

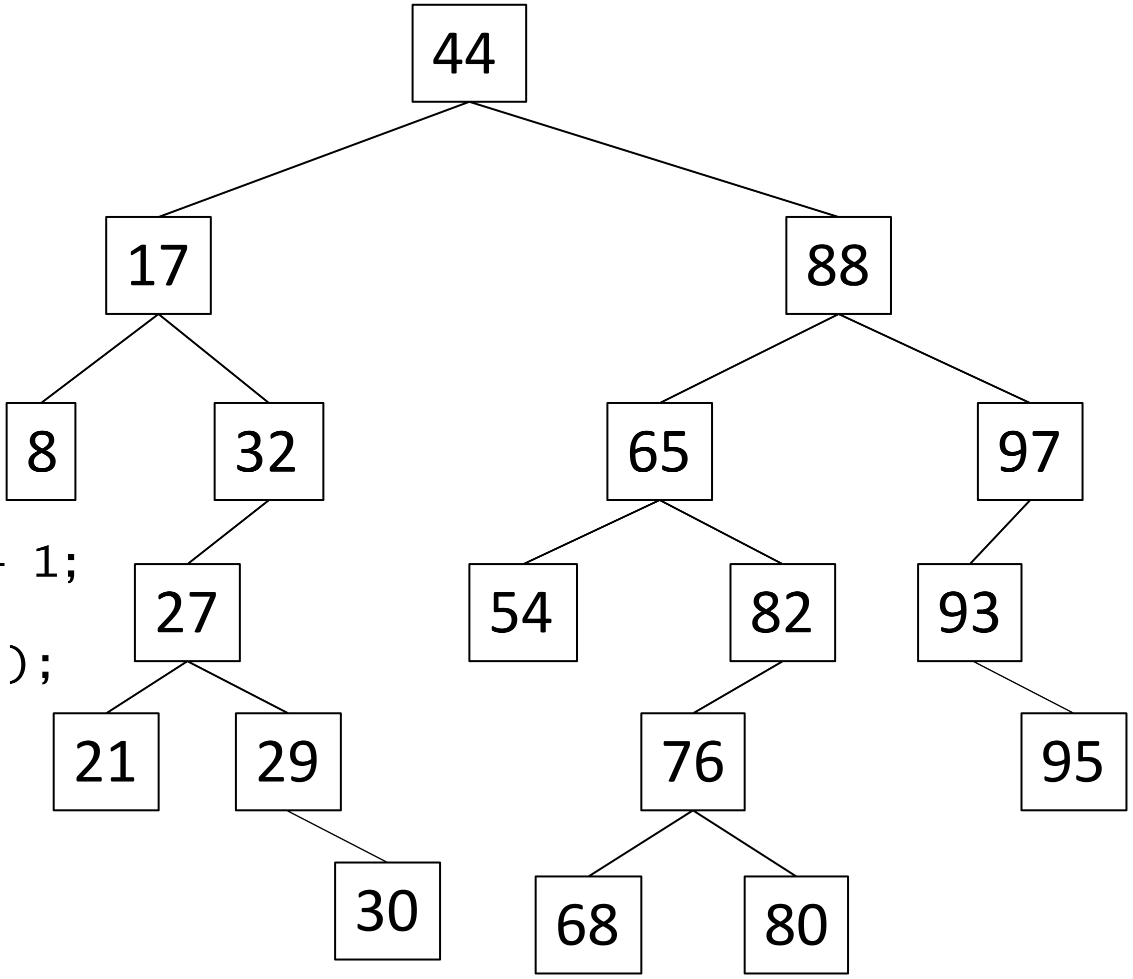
`insert(28);`

```
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;
}
```



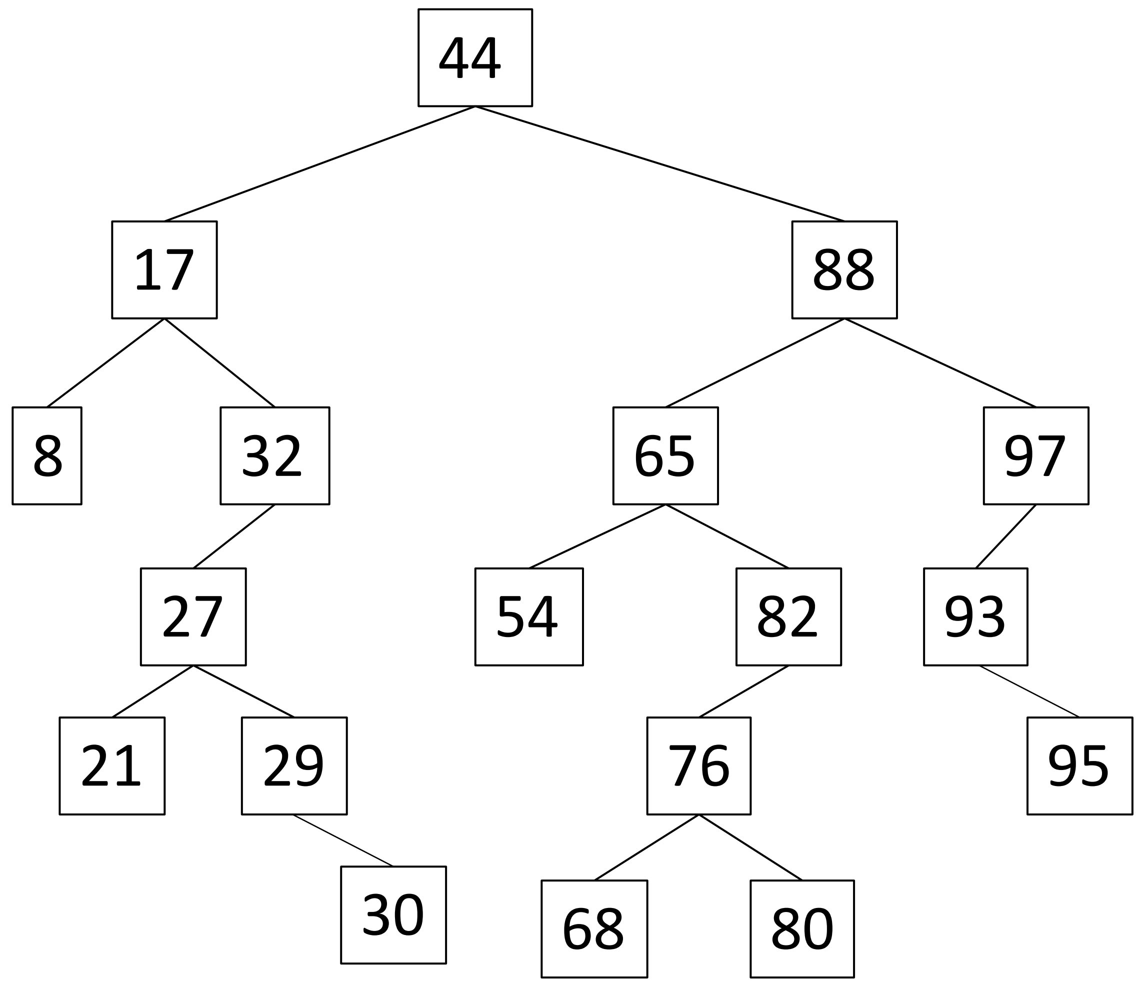
Binary Search Tree - Traversal

```
public void depthFirst() {
    Stack<Node> stack = new Stack<>();
    if (root != null) {
        stack.add(root);
        while(!stack.isEmpty()) {
            Node node = stack.pop();
            System.out.println(node.getName());
            for (int i = node.getChildren().size() – 1;
                 i >= 0; i--) {
                stack.push(node.getChildren().get(i));
            }
        }
    }
}
```



Binary Search Tree - Traversal

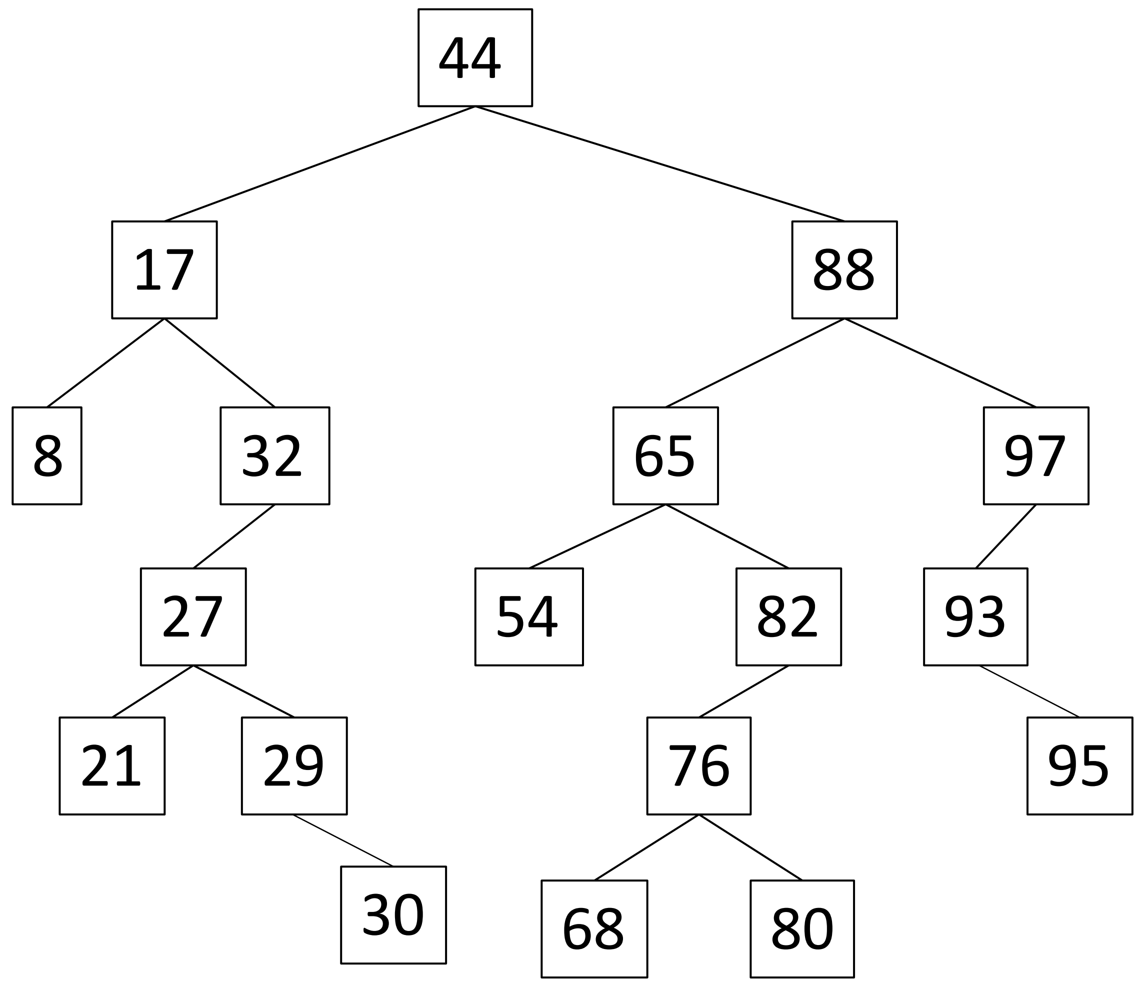
Recursion:



Binary Search Tree - Traversal

Recursion:

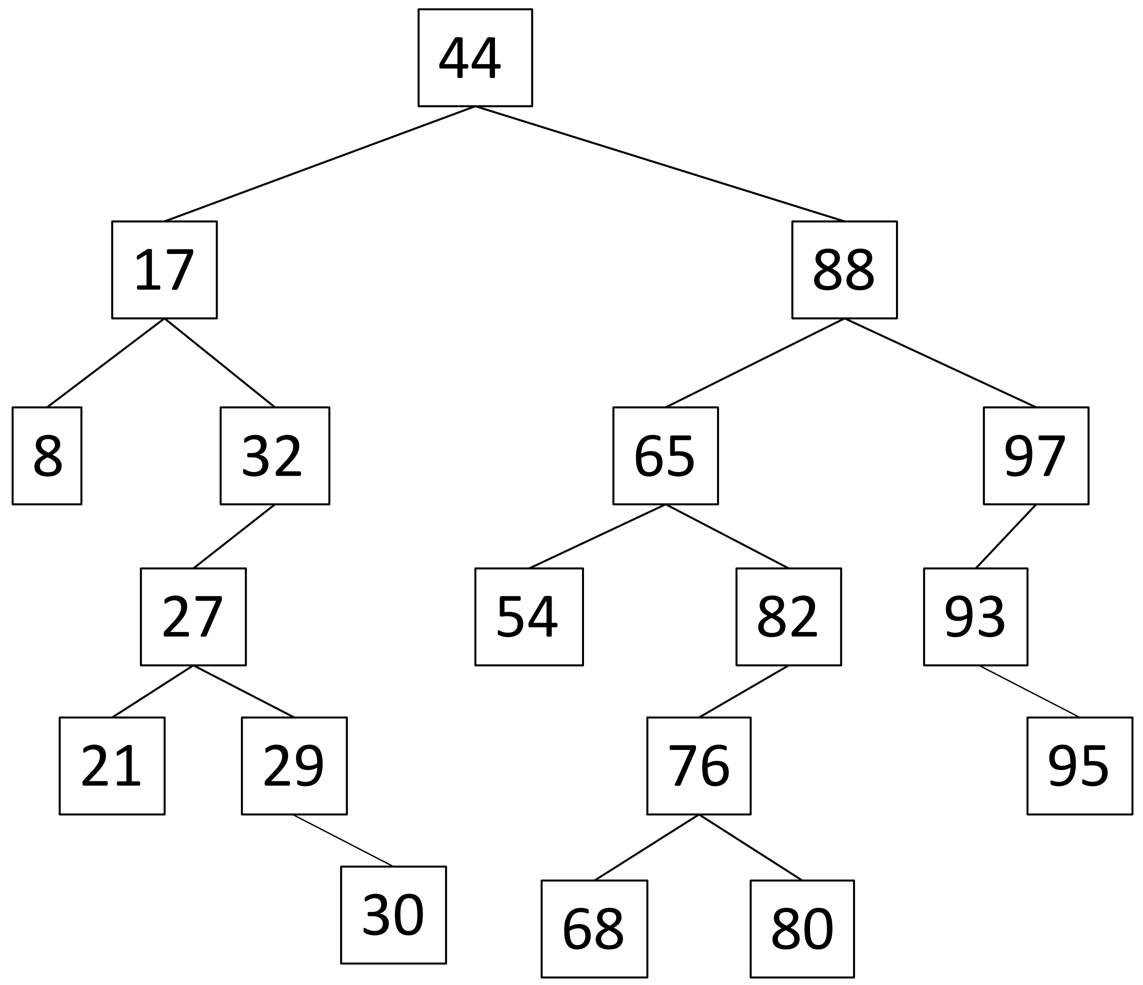
- **Calling a method from inside itself.**



Binary Search Tree - Traversal

Recursion:

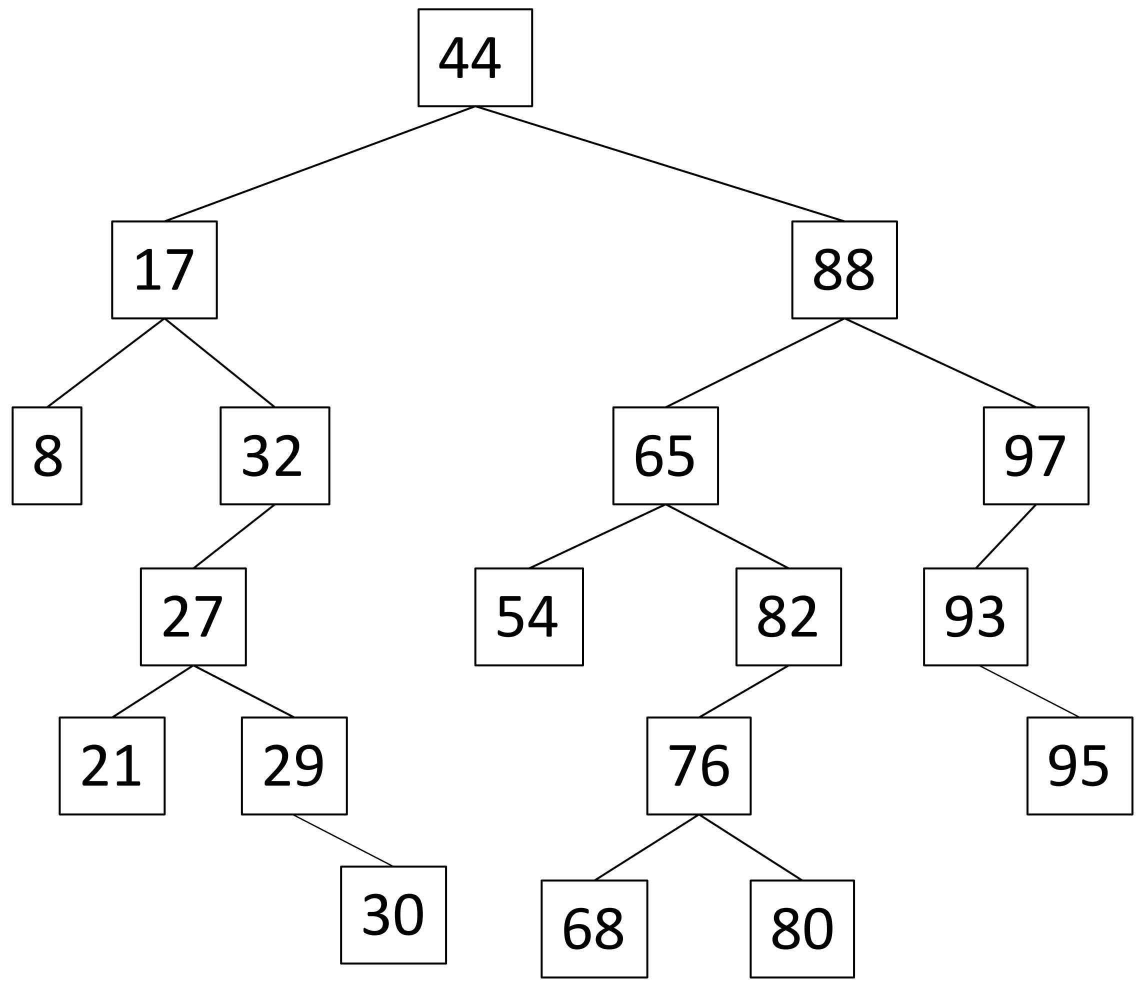
- **Calling a method from inside itself.**
- **Solve the problem by solving identical smaller problems.**



Binary Search Tree - Traversal

Recursion:

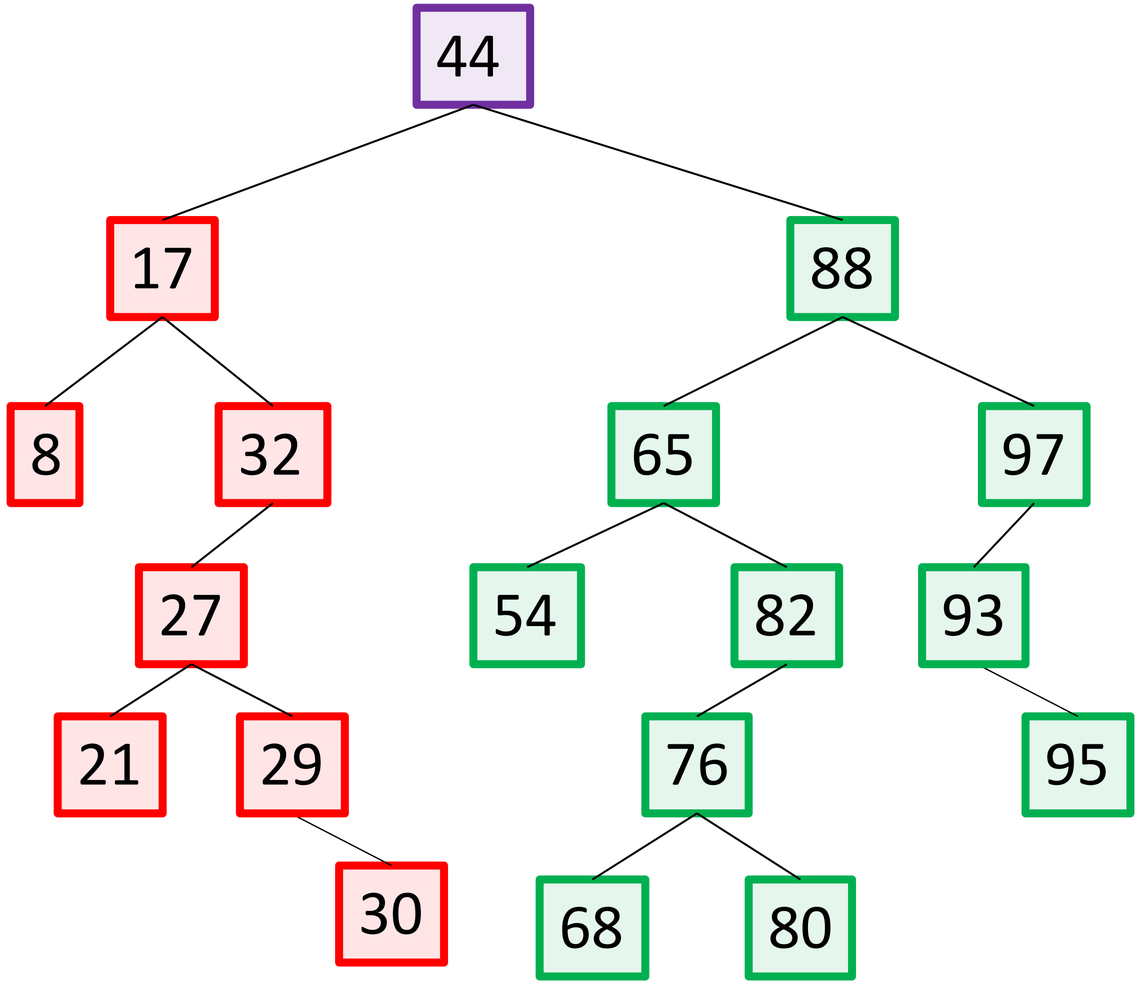
- **Calling a method from inside itself.**
- **Solve the problem by solving identical smaller problems.**
- **What is the "smaller problem"?**



Binary Search Tree - Traversal

Recursion:

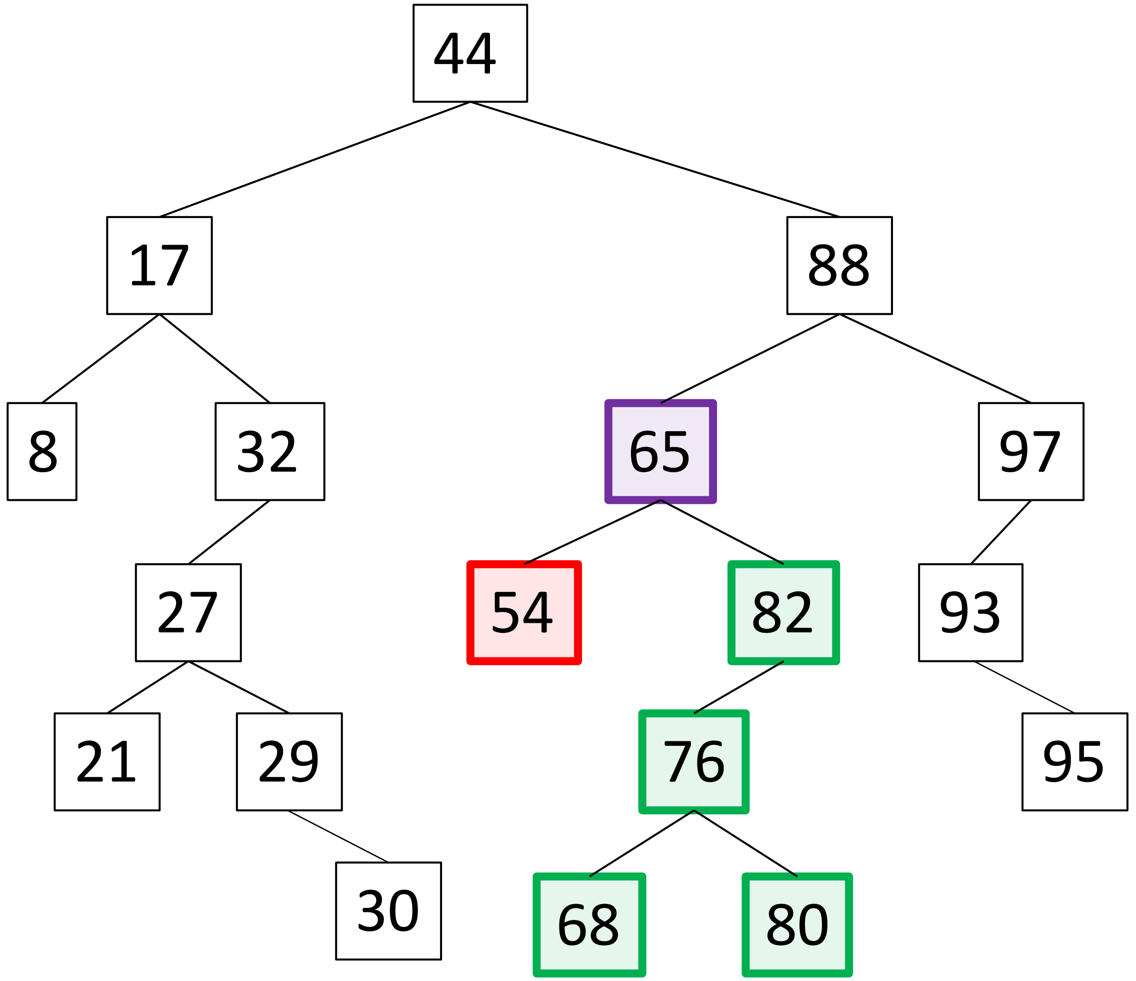
- **Calling a method from inside itself.**
- **Solve the problem by solving identical smaller problems.**
- **What is the "smaller problem"?**
  - **Process the left side, then process the right side.**



Binary Search Tree - Traversal

Recursion:

- **Calling a method from inside itself.**
- **Solve the problem by solving identical smaller problems.**
- **What is the "smaller problem"?**
    - **Process the left side, then process the right side.**

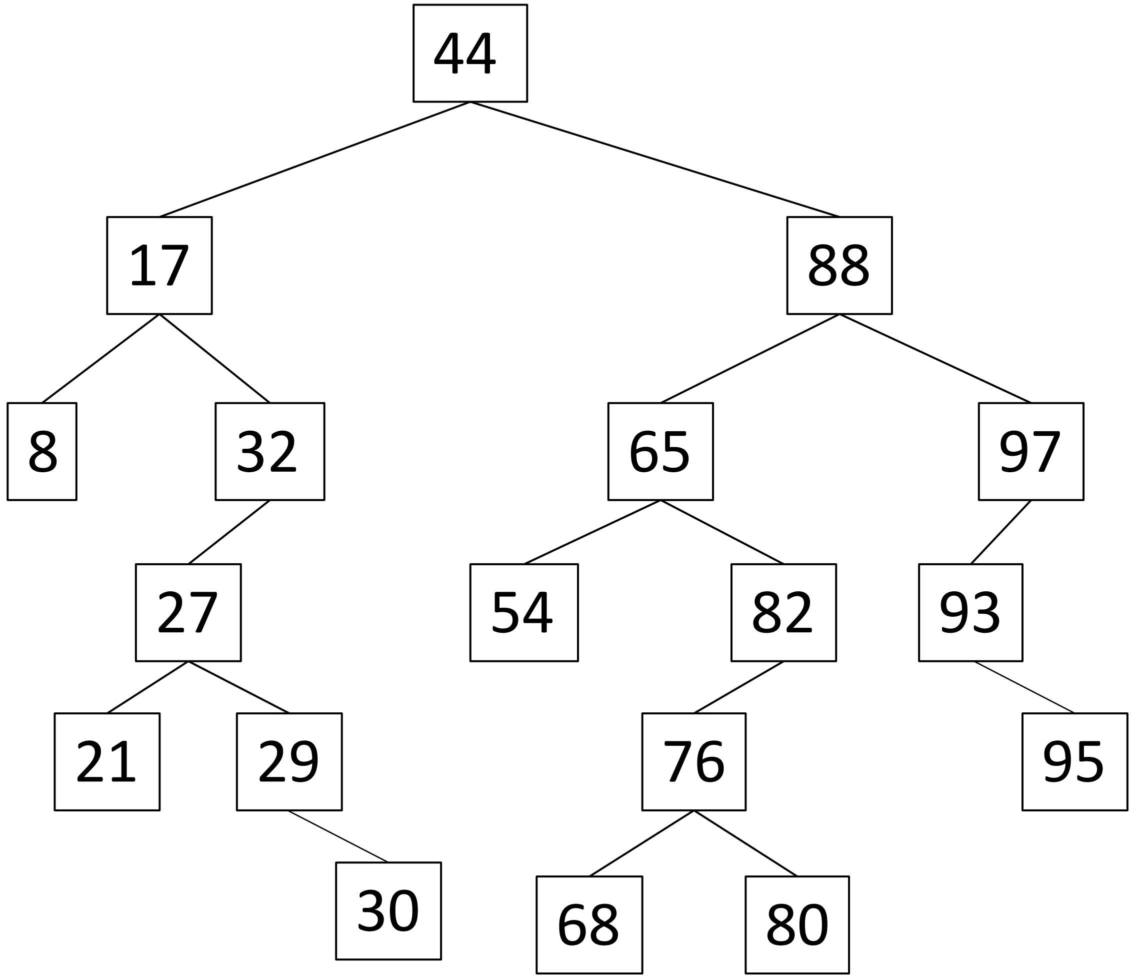


Binary Search Tree - Traversal

```
public void depthFirst(Node n) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```

Recursion:

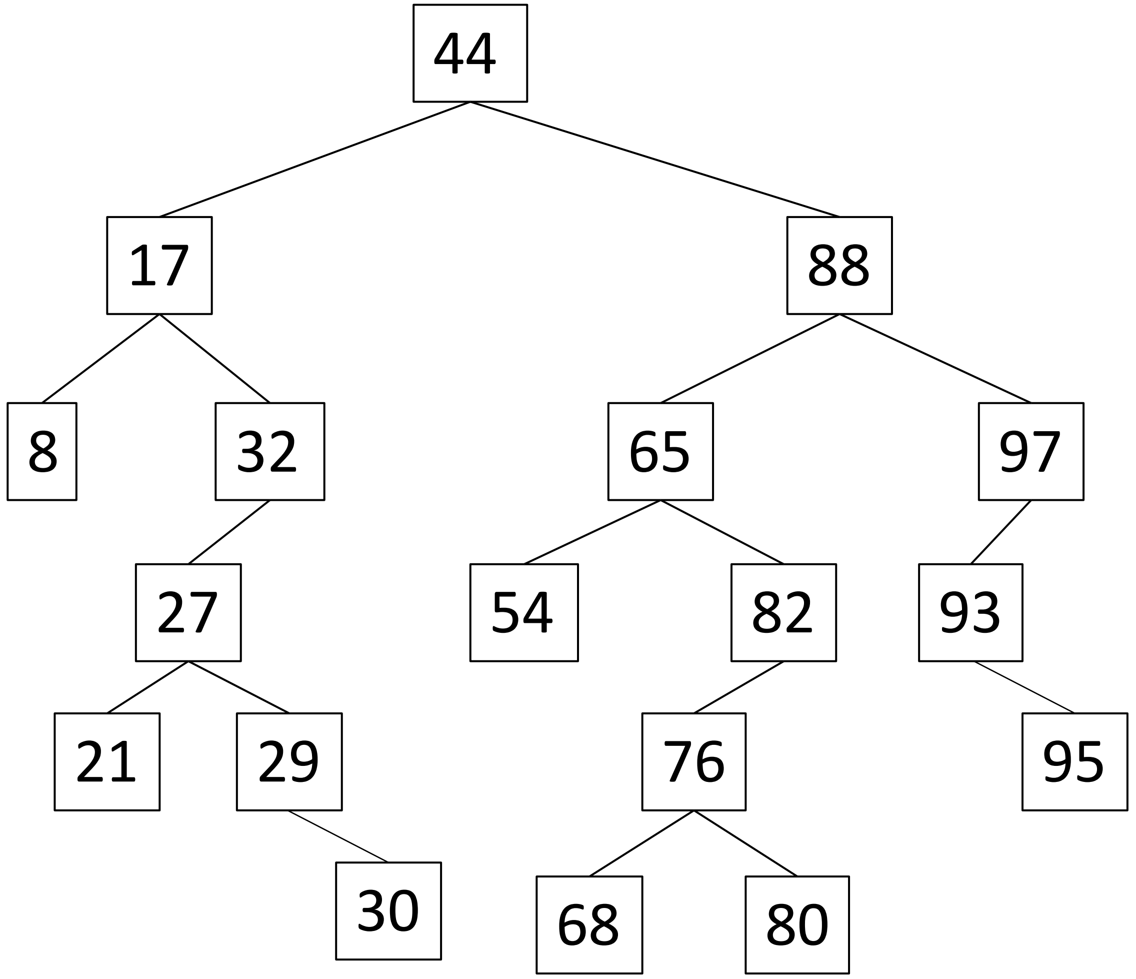
- **Calling a method from inside itself.**
- **Solve the problem by solving identical smaller problems.**
- **What is the "smaller problem"?**
  - **Process the left side, then process the right side.**



Output:

Binary Search Tree - Traversal

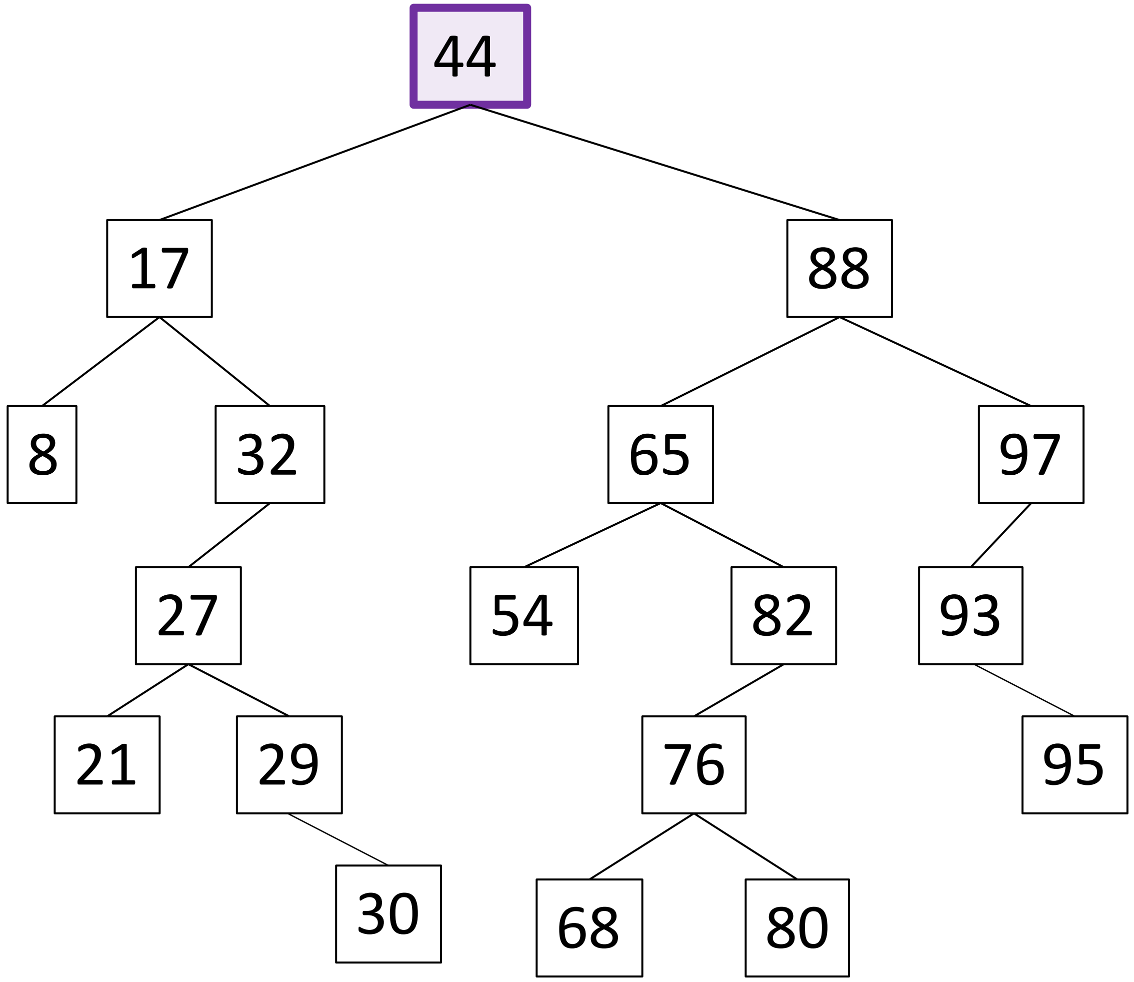
```
public void depthFirst(Node n) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```



Output:

Binary Search Tree - Traversal

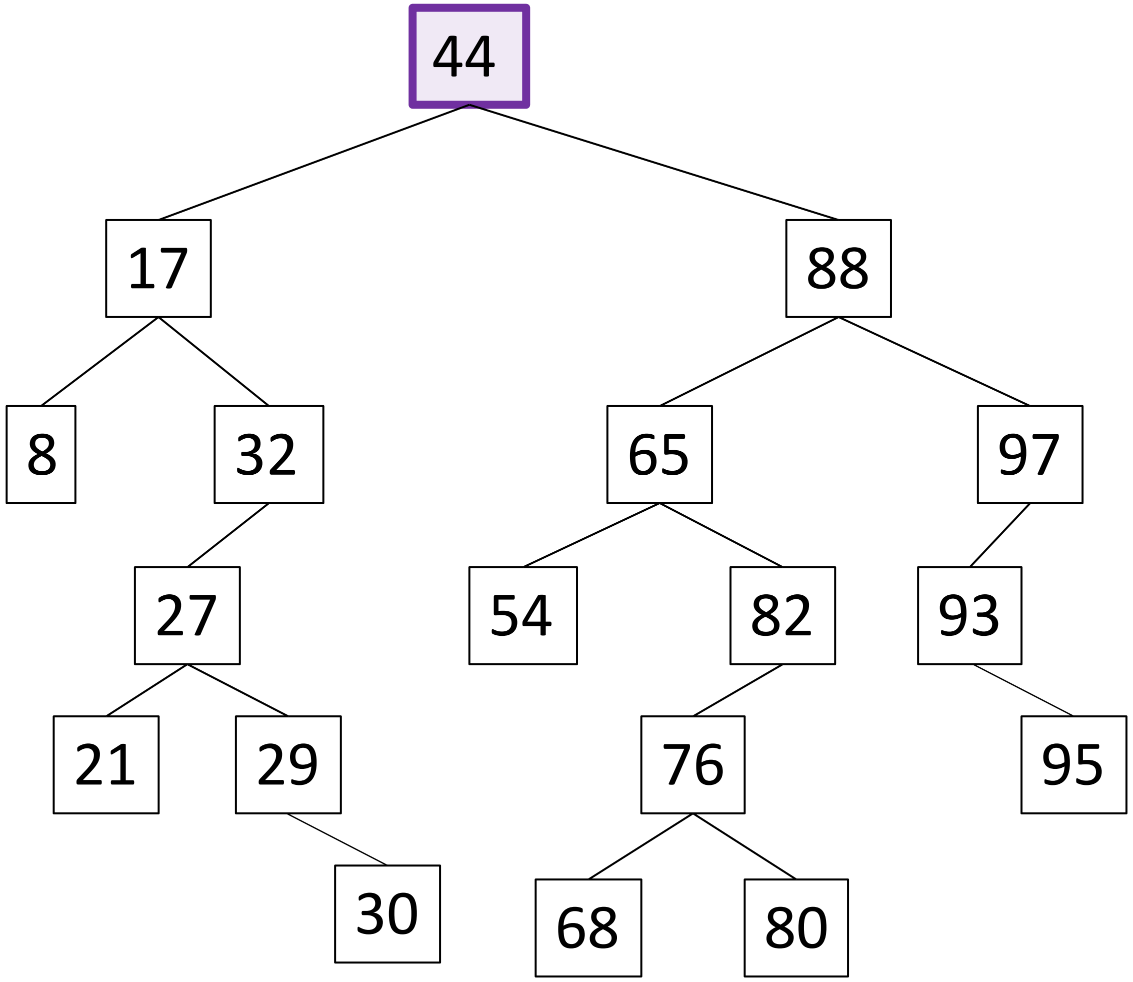
```
public void depthFirst(44) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```



Output:
44

Binary Search Tree - Traversal

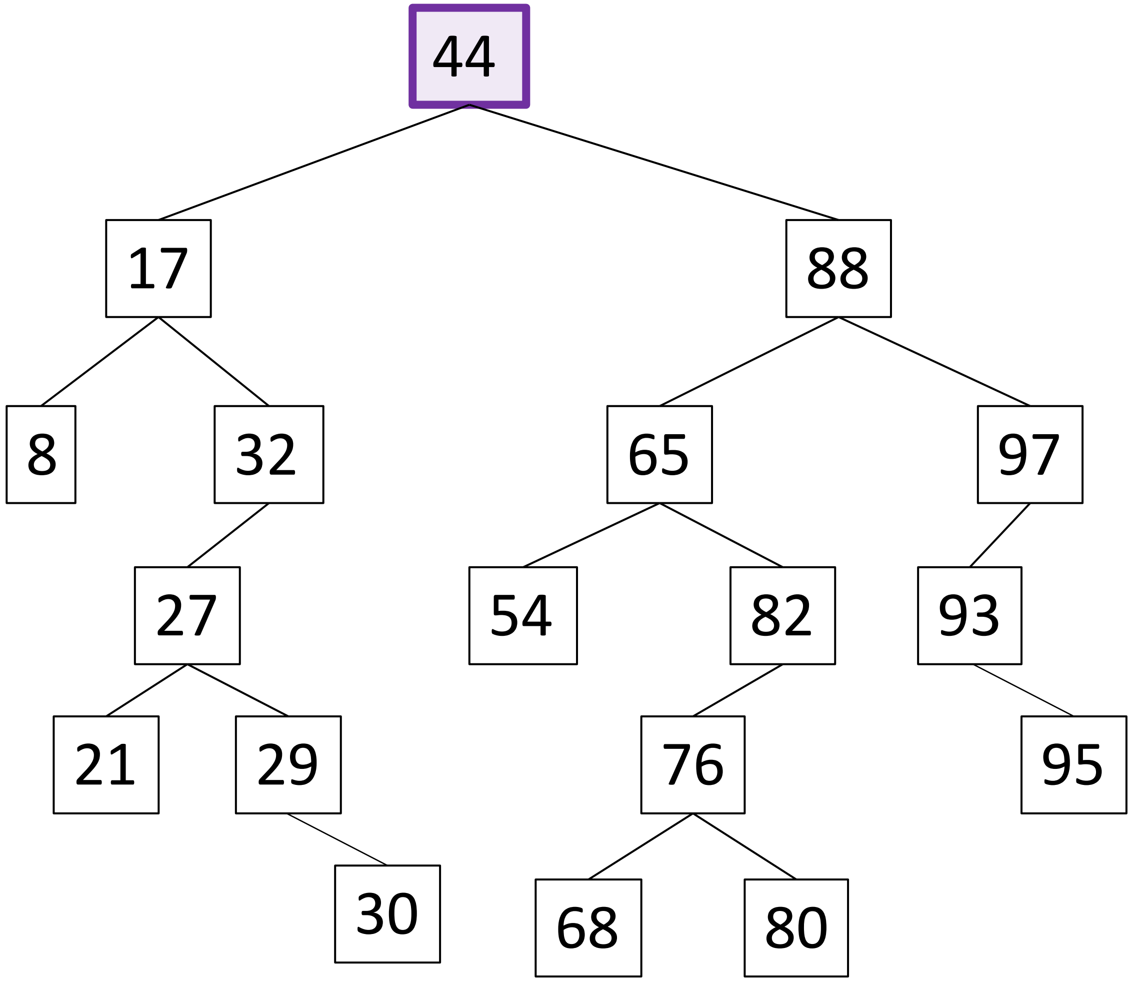
```
public void depthFirst(44) {
    if (n != null) {
        System.out.println(n.getValue());

        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```



Output:
44

Binary Search Tree - Traversal

```
public void depthFirst(44) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```

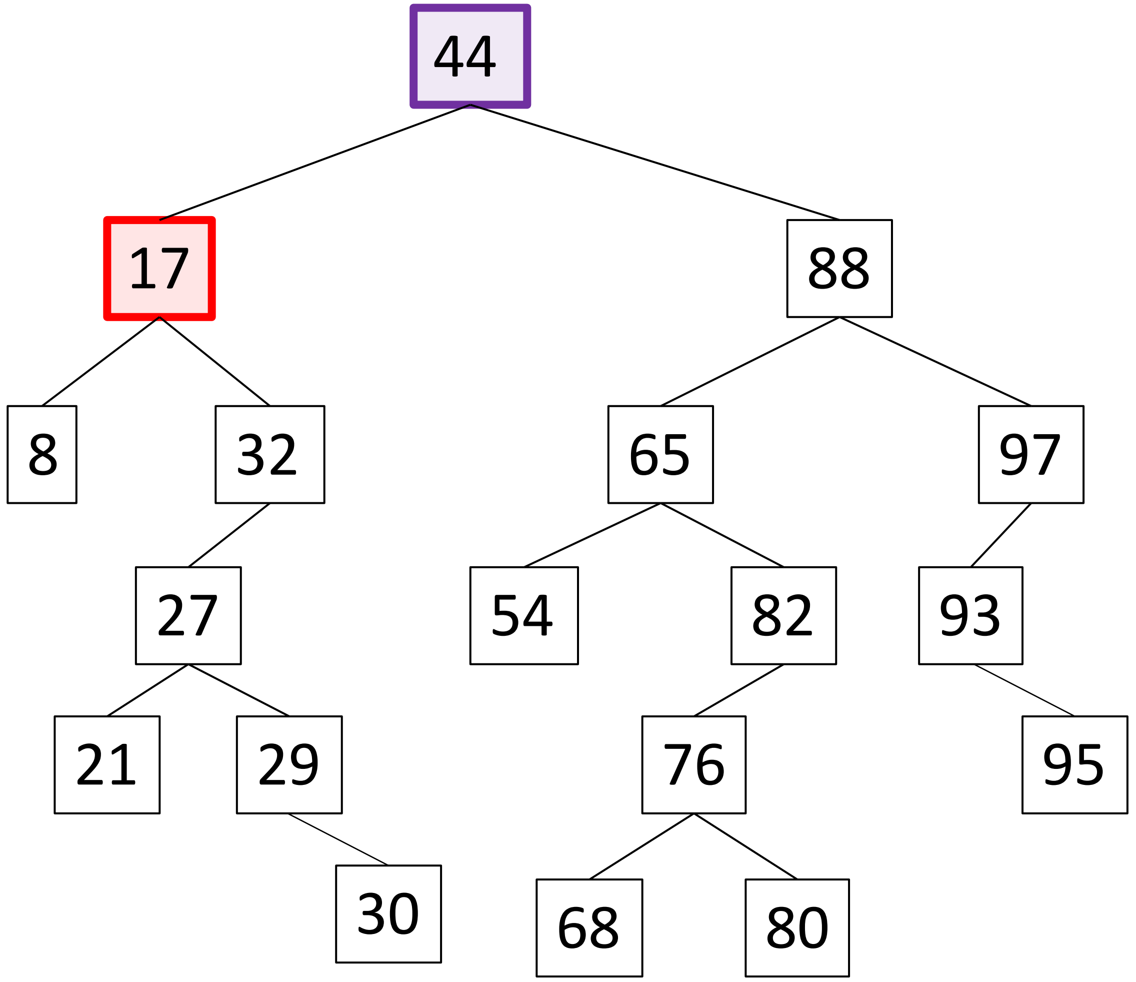


Output:
44

Binary Search Tree - Traversal

```
public void depthFirst(44) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```

```
public void depthFirst(17) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```

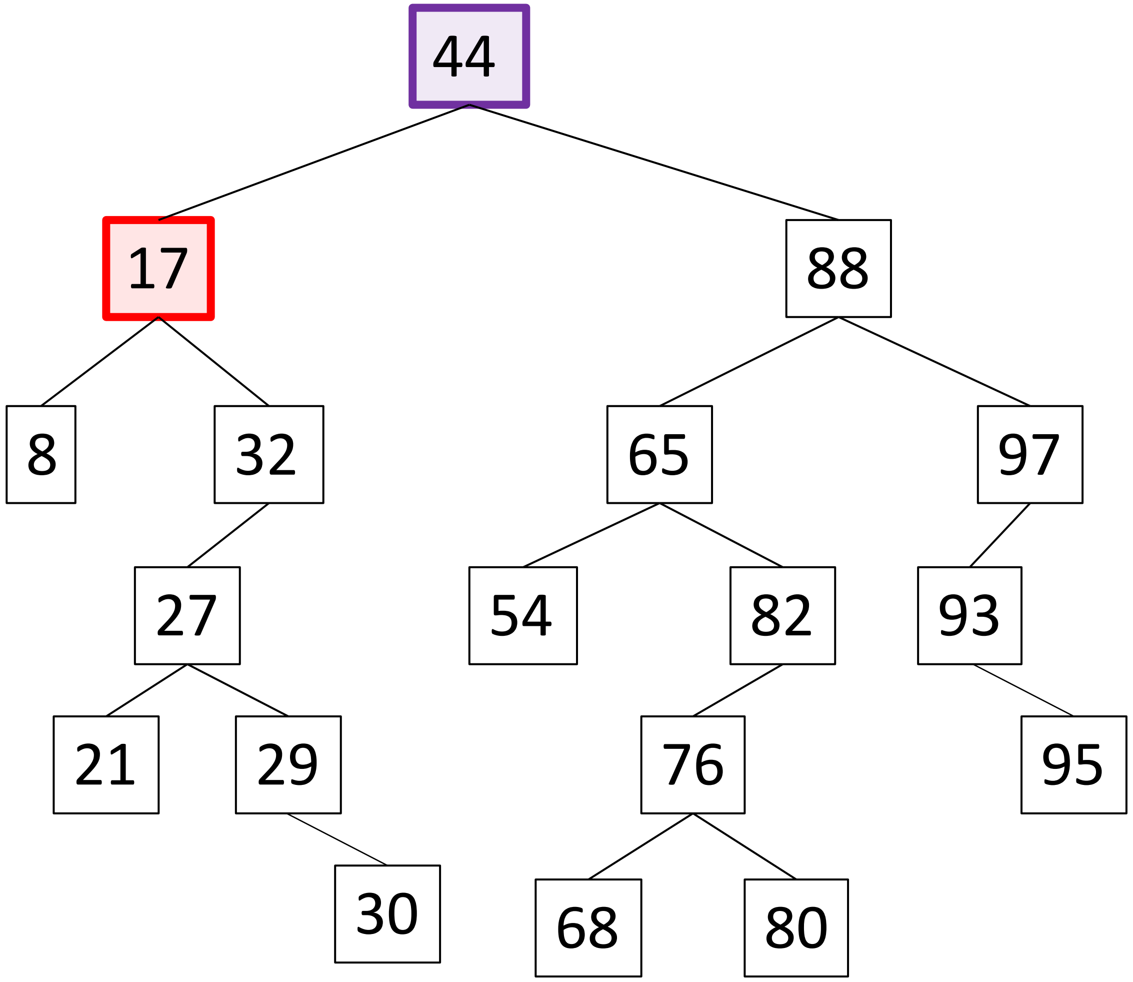


Output:
44
17

Binary Search Tree - Traversal

```
public void depthFirst(44) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```

```
public void depthFirst(17) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```

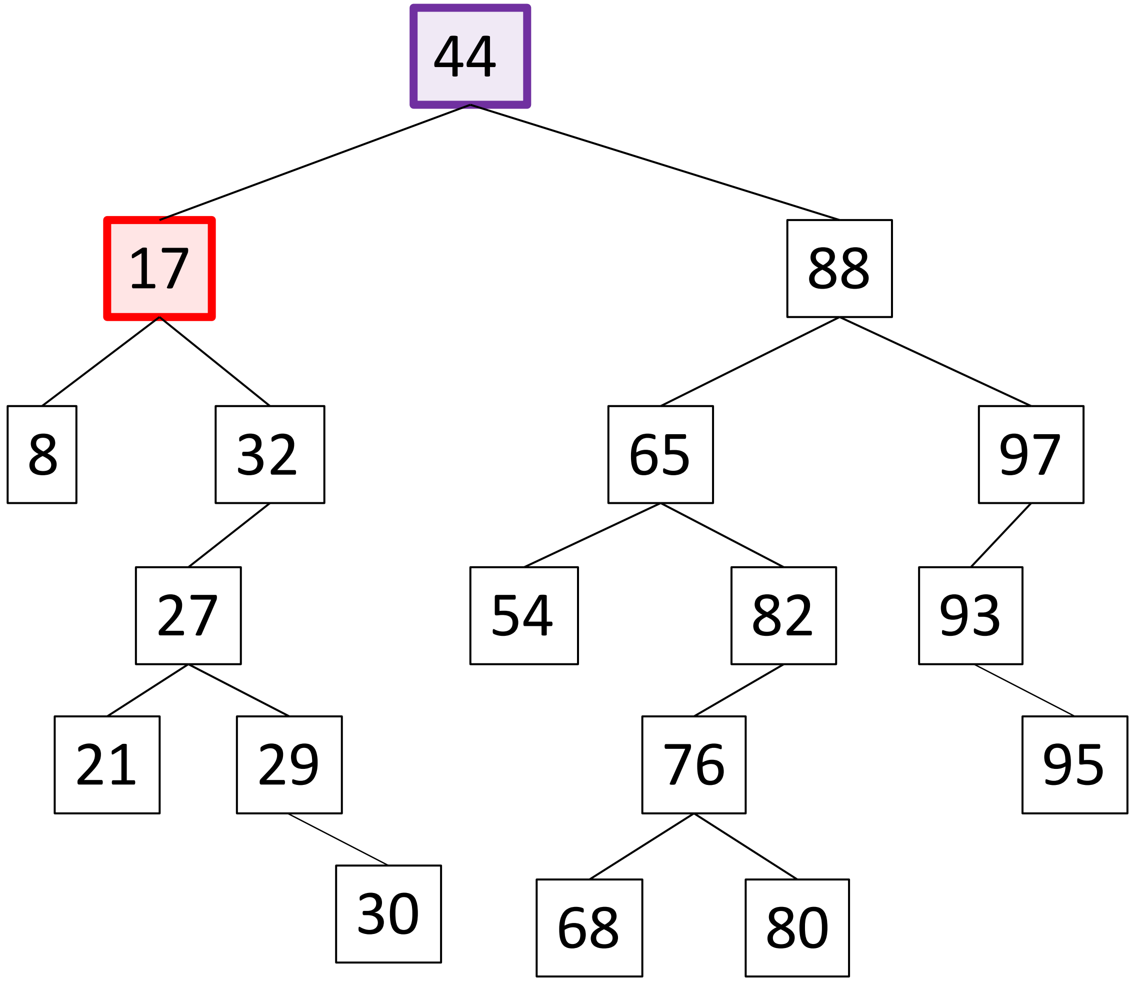


Output:
44
17

Binary Search Tree - Traversal

```
public void depthFirst(44) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```

```
public void depthFirst(17) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```



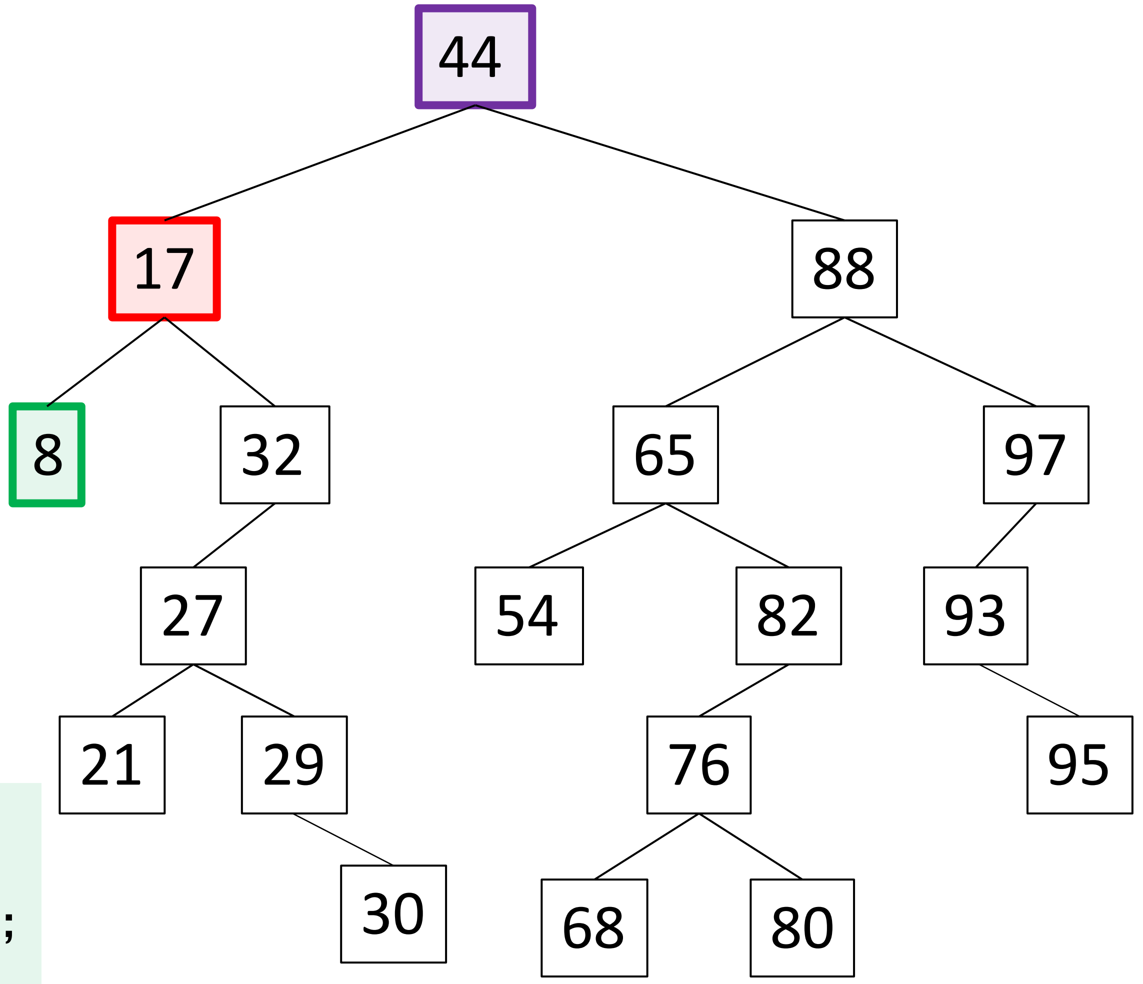
Output:
44
17

Binary Search Tree - Traversal

```
public void depthFirst(44) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```

```
public void depthFirst(17) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```

```
public void depthFirst(8) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```



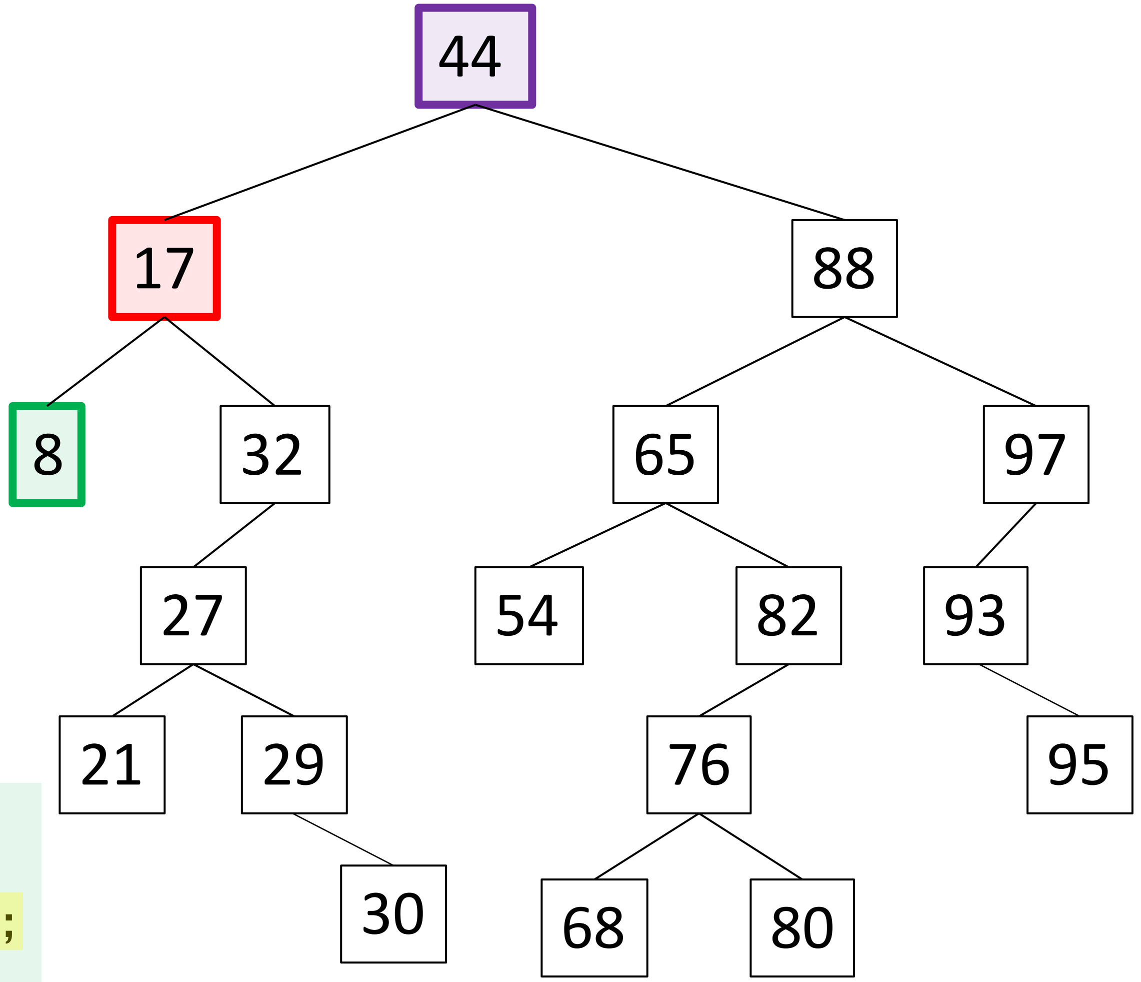
Output:
44
17
8

Binary Search Tree - Traversal

```
public void depthFirst(44) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```

```
public void depthFirst(17) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```

```
public void depthFirst(8) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```



Output:
44
17
8

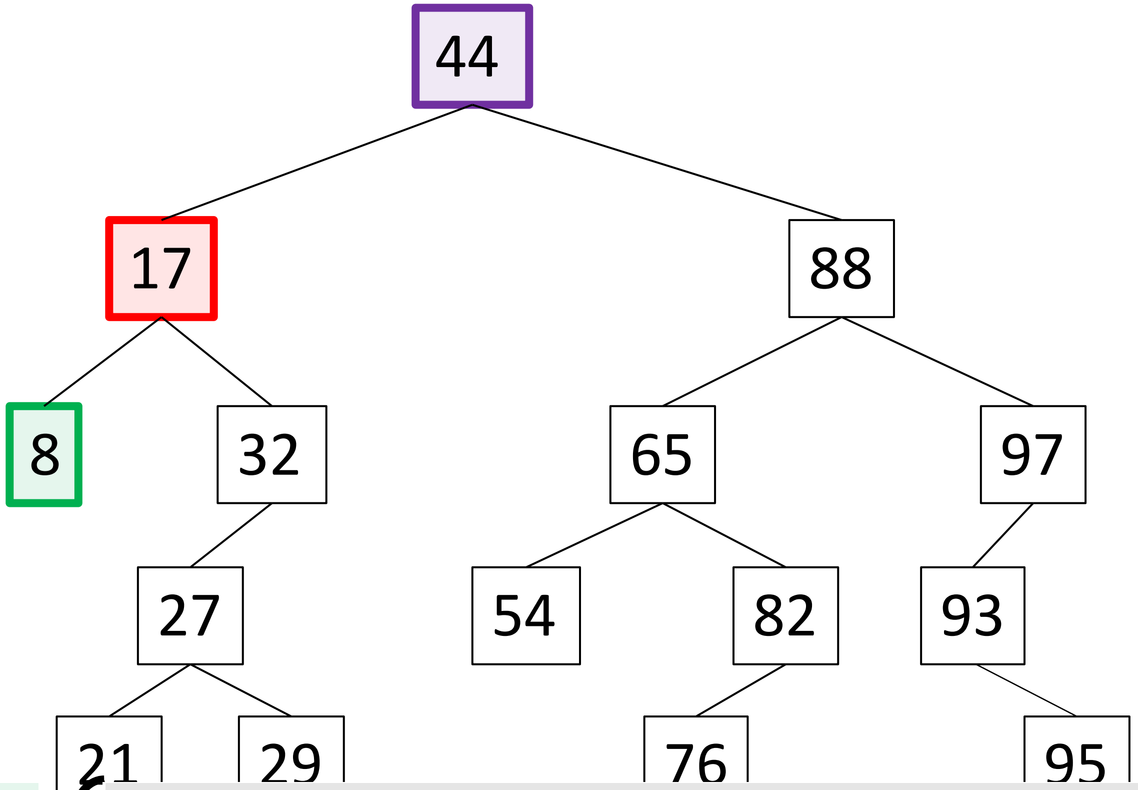
Binary Search Tree - Traversal

```
public void depthFirst(44) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```

```
public void depthFirst(17) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```

```
public void depthFirst(8) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```

```
public void depthFirst(null) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```



Output:

44

17

8

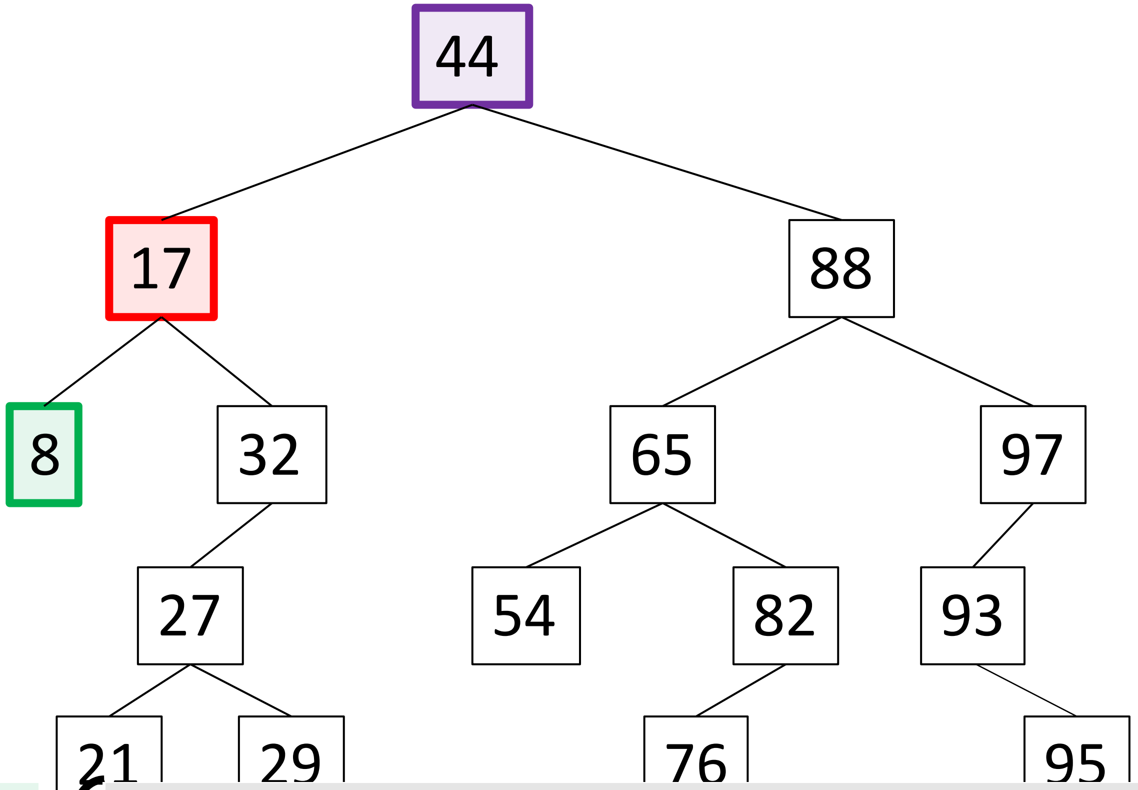
Binary Search Tree - Traversal

```
public void depthFirst(44) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```

```
public void depthFirst(17) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```

```
public void depthFirst(8) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```

```
public void depthFirst(null) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```



Output:
44
17
8

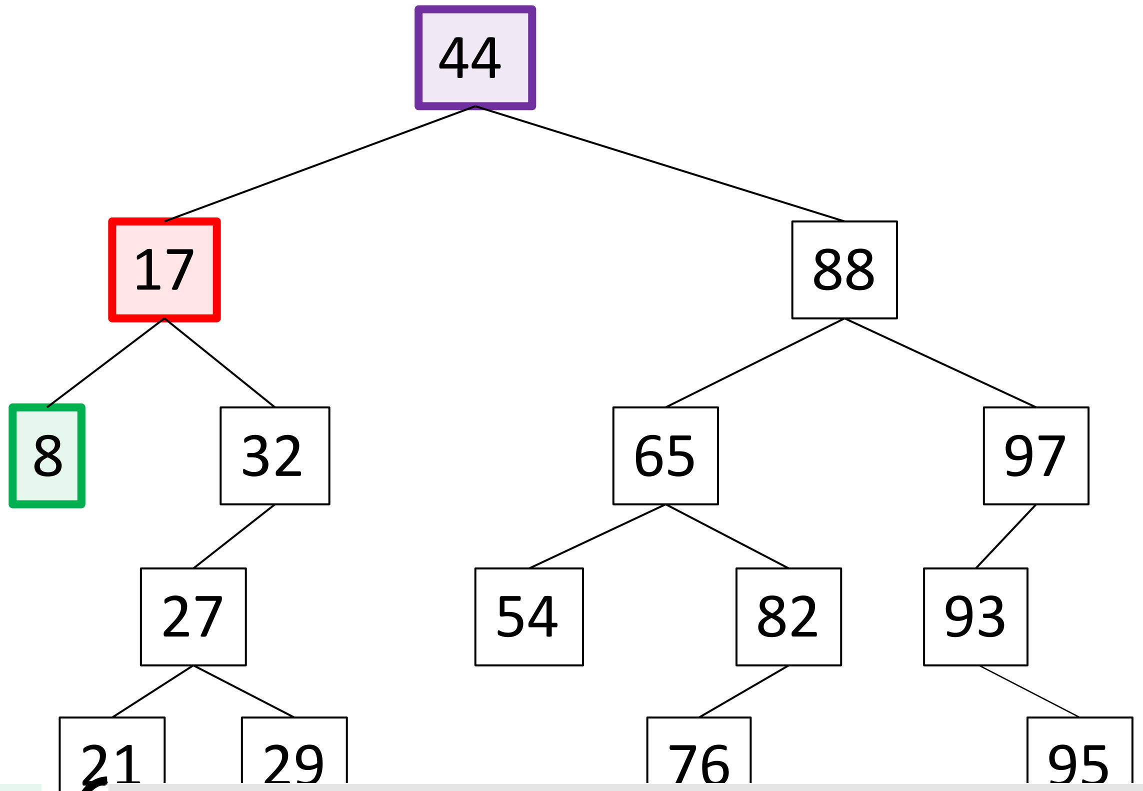
Binary Search Tree - Traversal

```
public void depthFirst(44) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```

```
public void depthFirst(17) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```

```
public void depthFirst(8) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```

```
public void depthFirst(null) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```



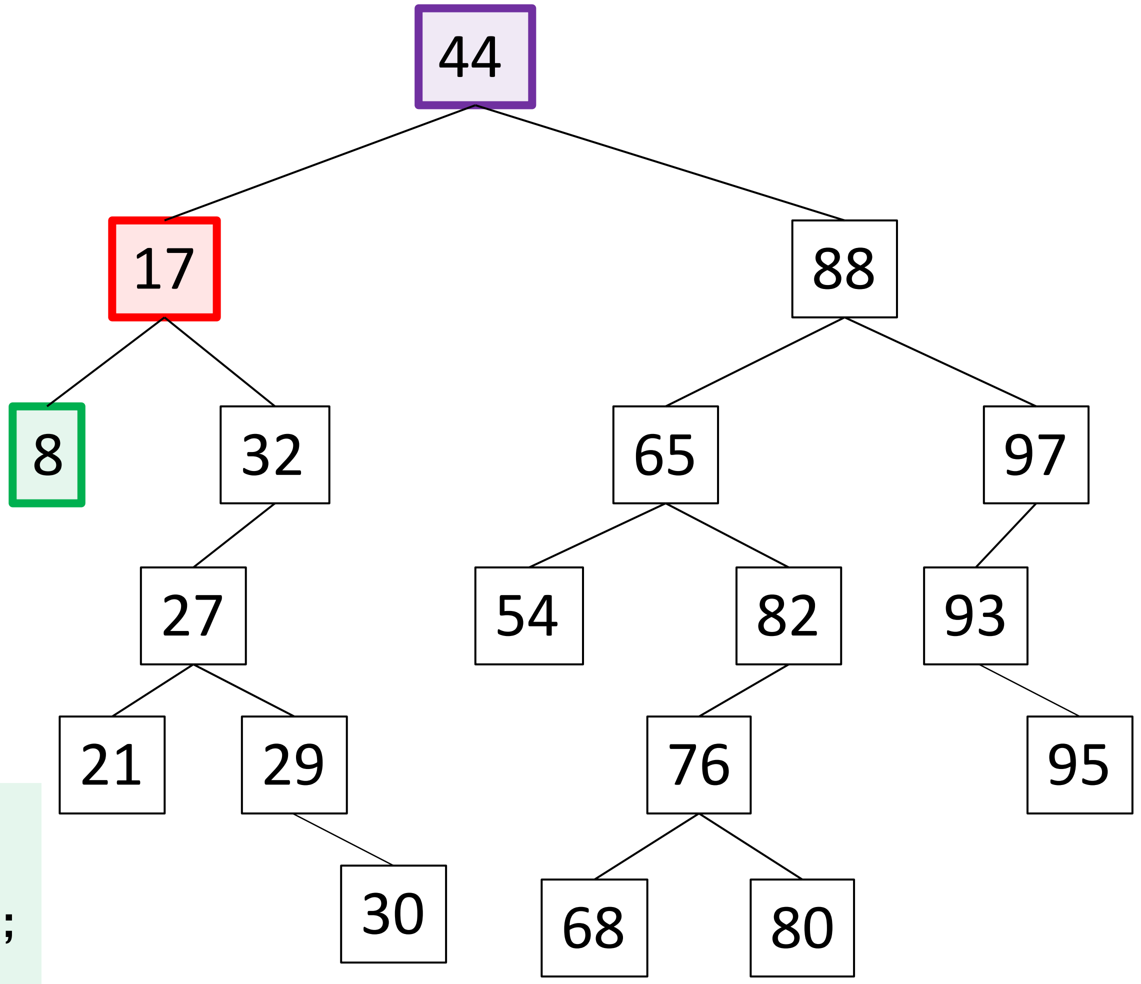
Output:
44
17
8

Binary Search Tree - Traversal

```
public void depthFirst(44) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```

```
public void depthFirst(17) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```

```
public void depthFirst(8) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```



Output:

44

17

8

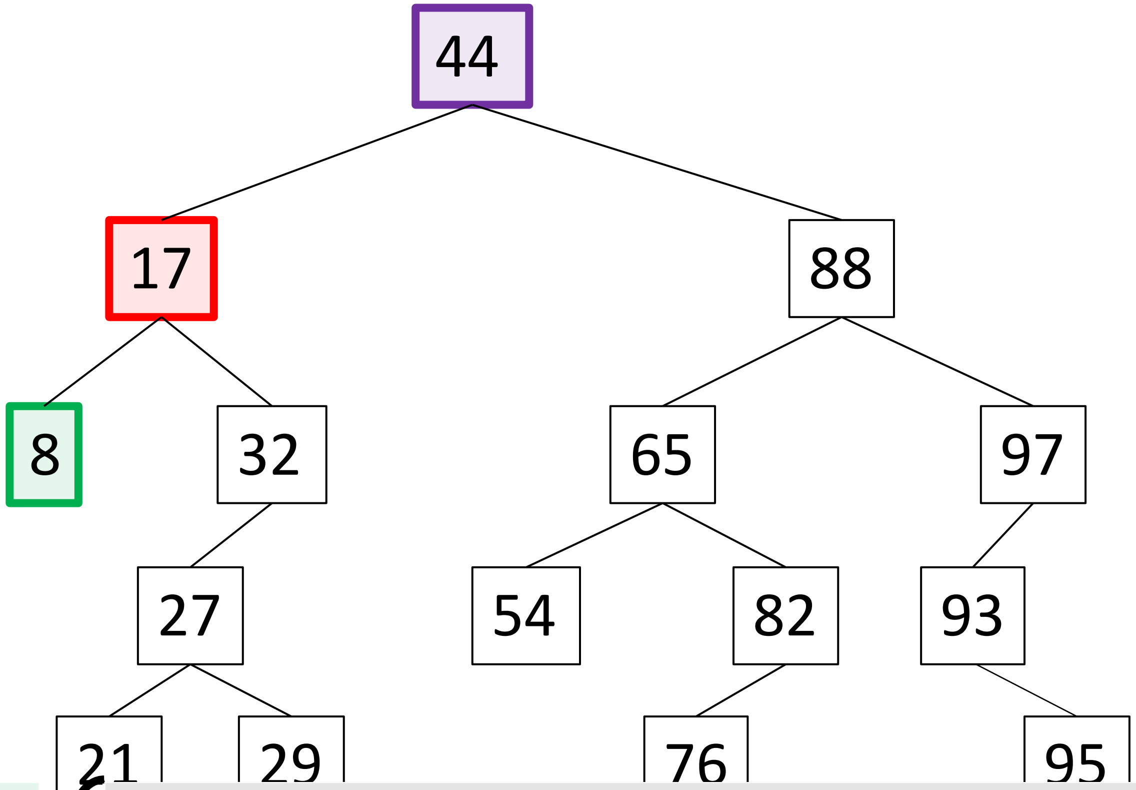
Binary Search Tree - Traversal

```
public void depthFirst(44) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```

```
public void depthFirst(17) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```

```
public void depthFirst(8) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```

```
public void depthFirst(null) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```



Output:
44
17
8

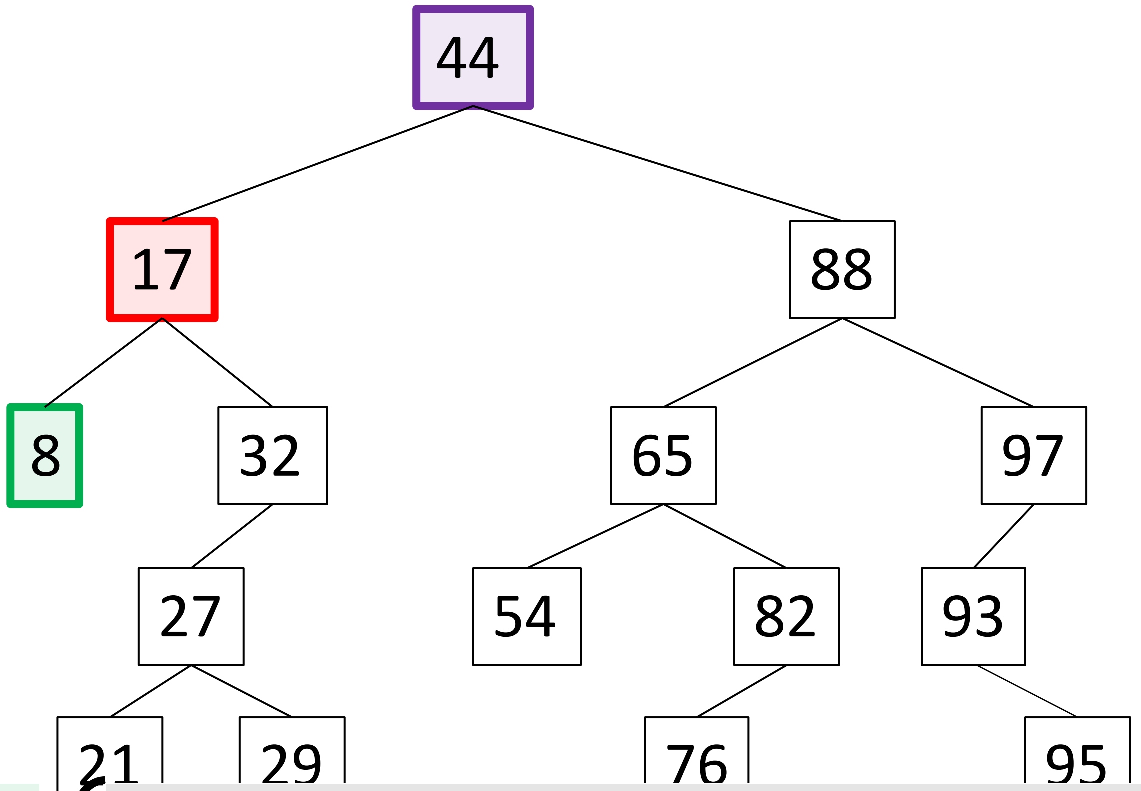
Binary Search Tree - Traversal

```
public void depthFirst(44) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```

```
public void depthFirst(17) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```

```
public void depthFirst(8) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```

```
public void depthFirst(null) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```



Output:
44
17
8

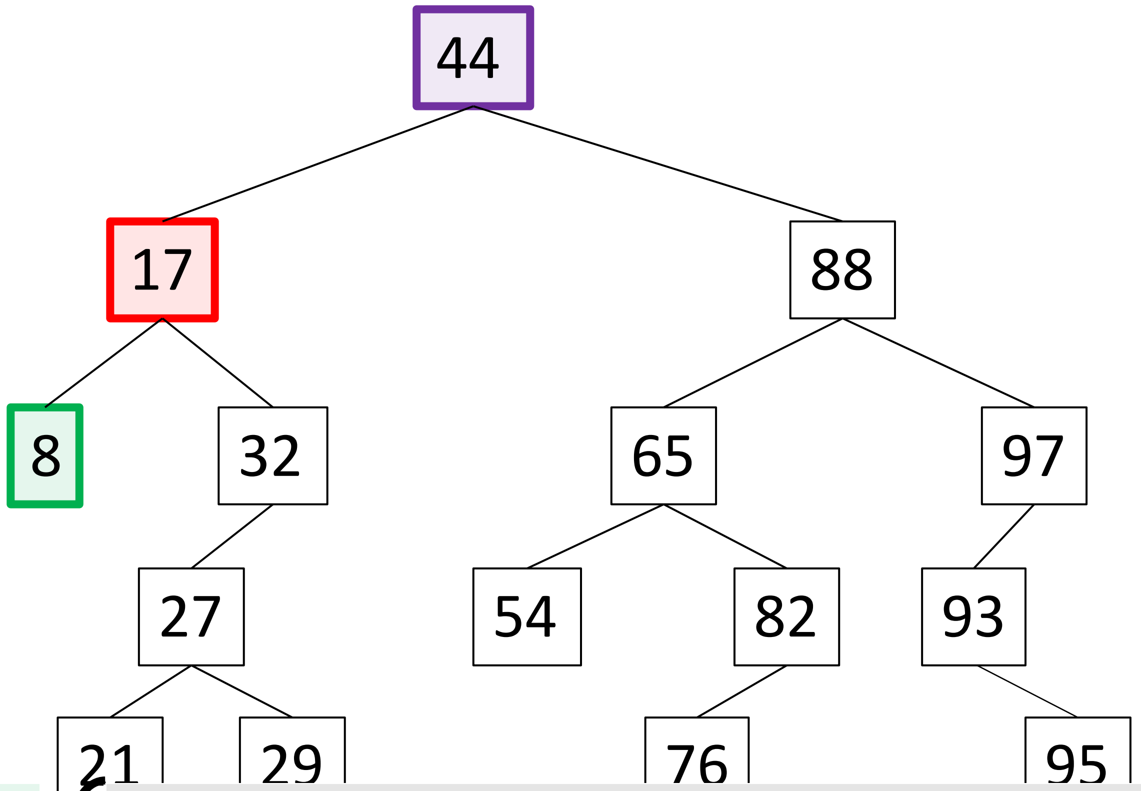
Binary Search Tree - Traversal

```
public void depthFirst(44) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```

```
public void depthFirst(17) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```

```
public void depthFirst(8) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```

```
public void depthFirst(null) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```



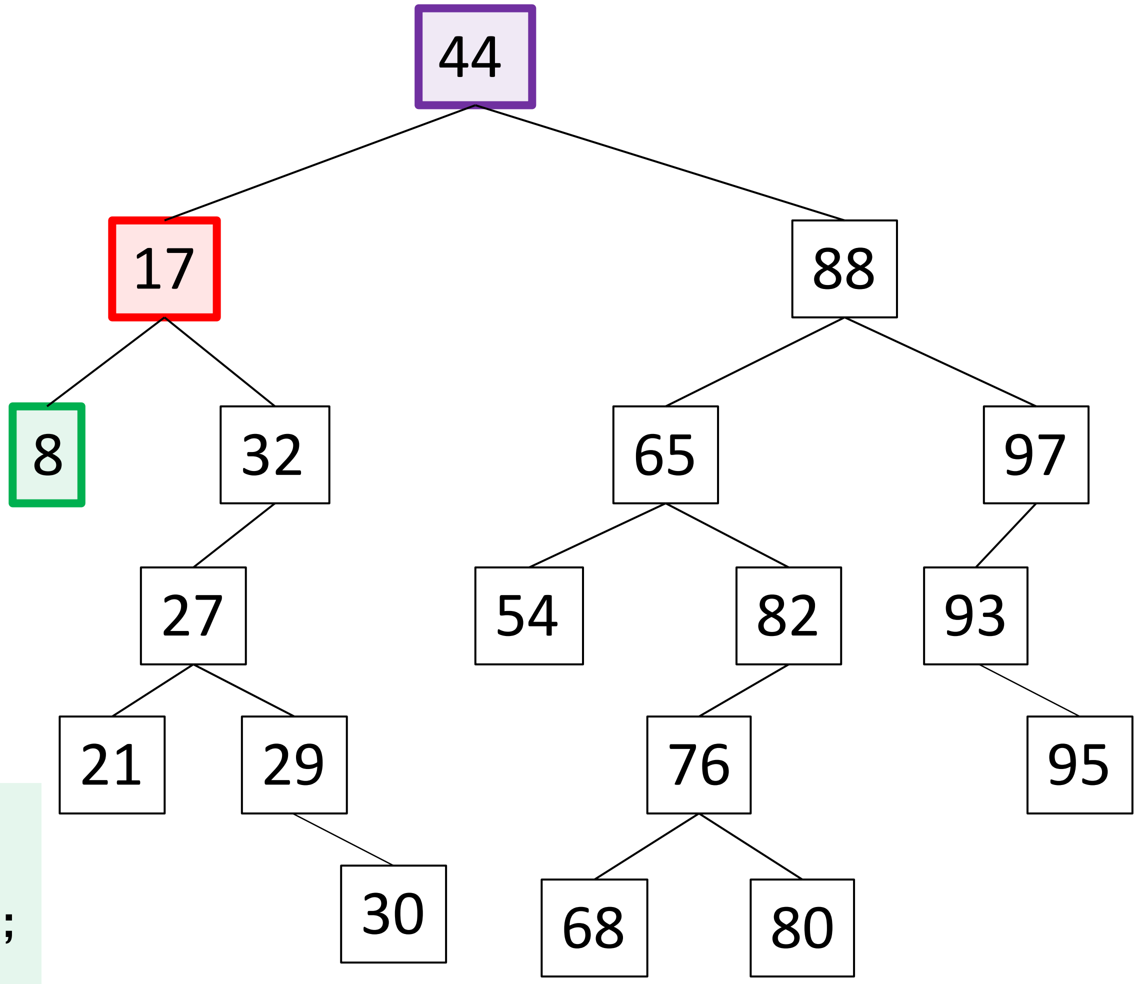
Output:
44
17
8

Binary Search Tree - Traversal

```
public void depthFirst(44) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```

```
public void depthFirst(17) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```

```
public void depthFirst(8) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```



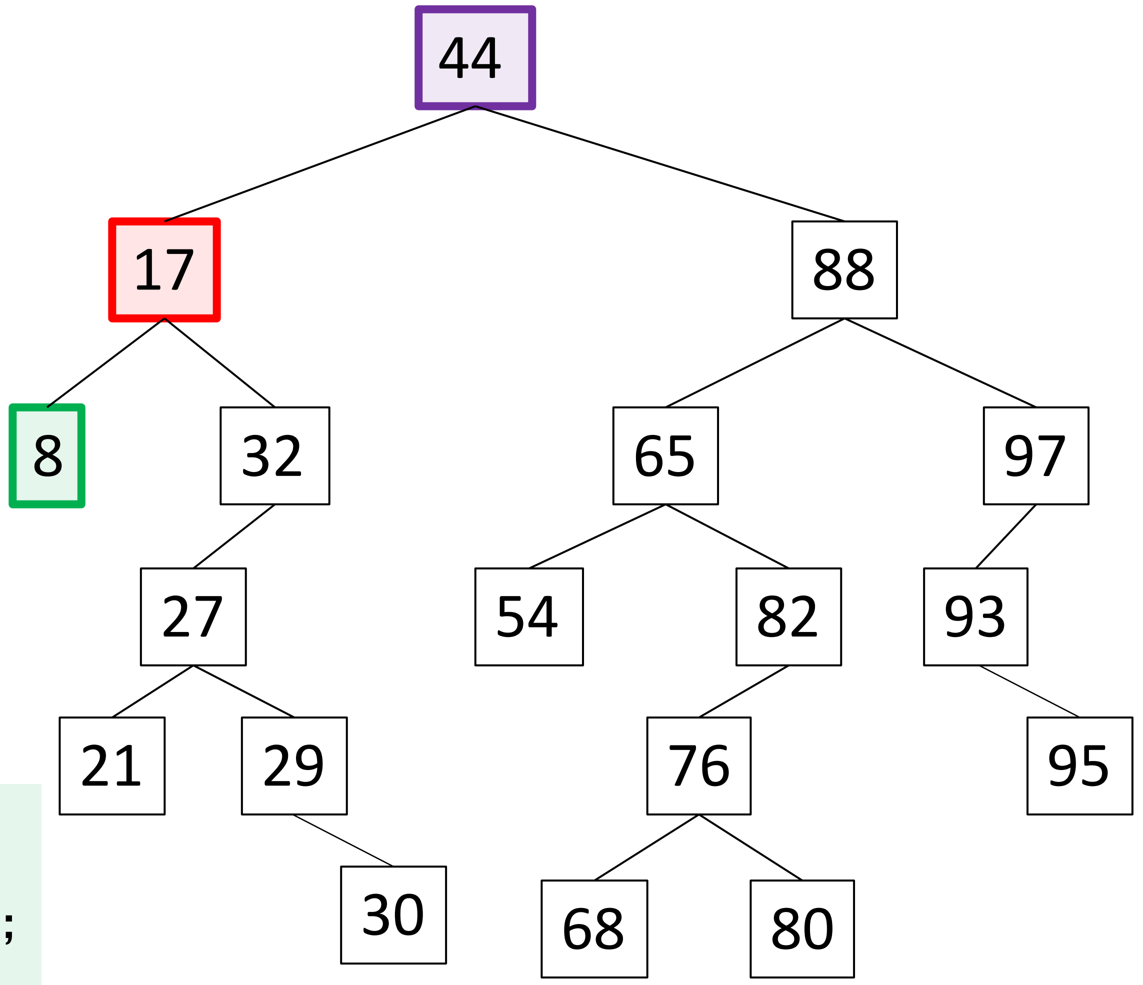
Binary Search Tree - Traversal

Output:
44
17
8

```
public void depthFirst(44) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```

```
public void depthFirst(17) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```

```
public void depthFirst(8) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```



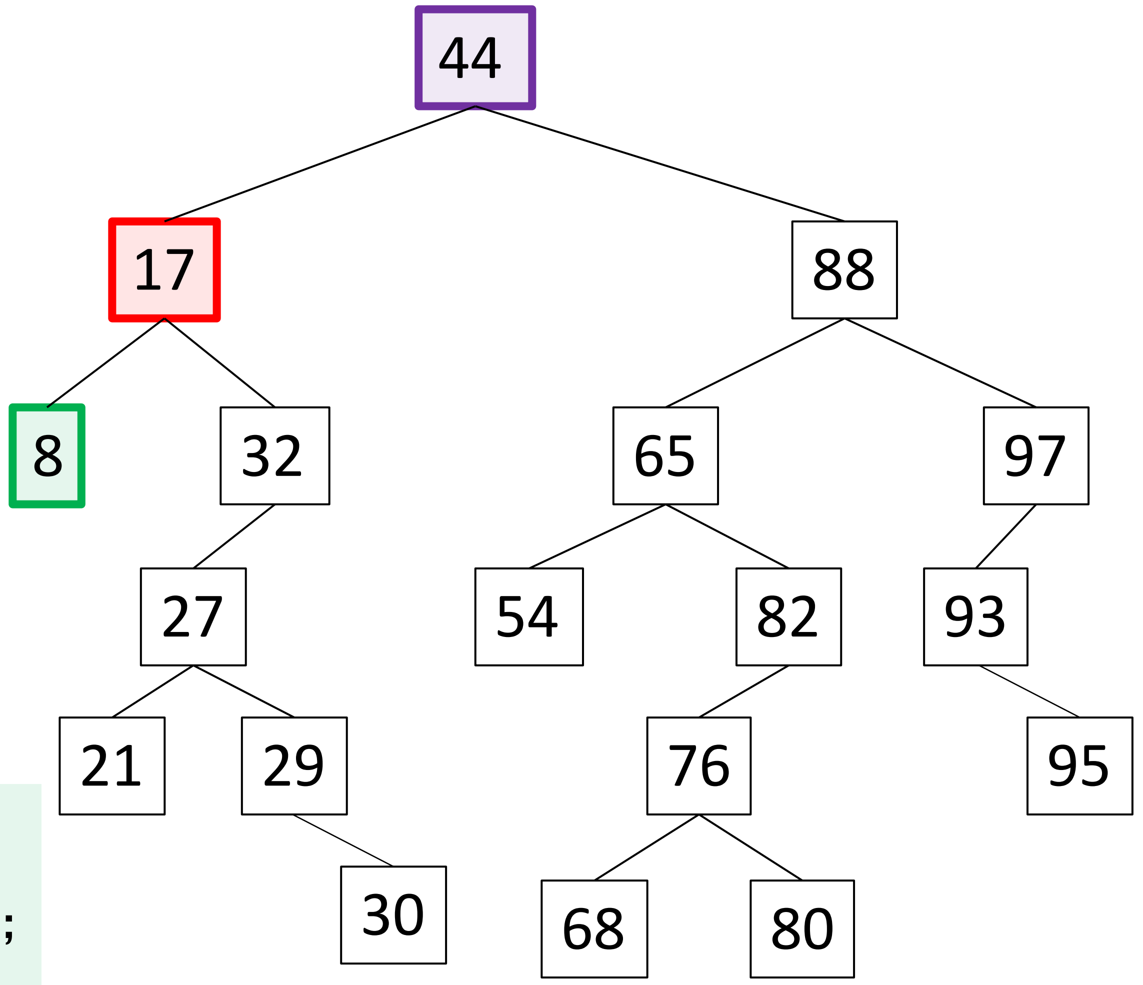
Output:
44
17
8

Binary Search Tree - Traversal

```
public void depthFirst(44) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```

```
public void depthFirst(17) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```

```
public void depthFirst(8) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```

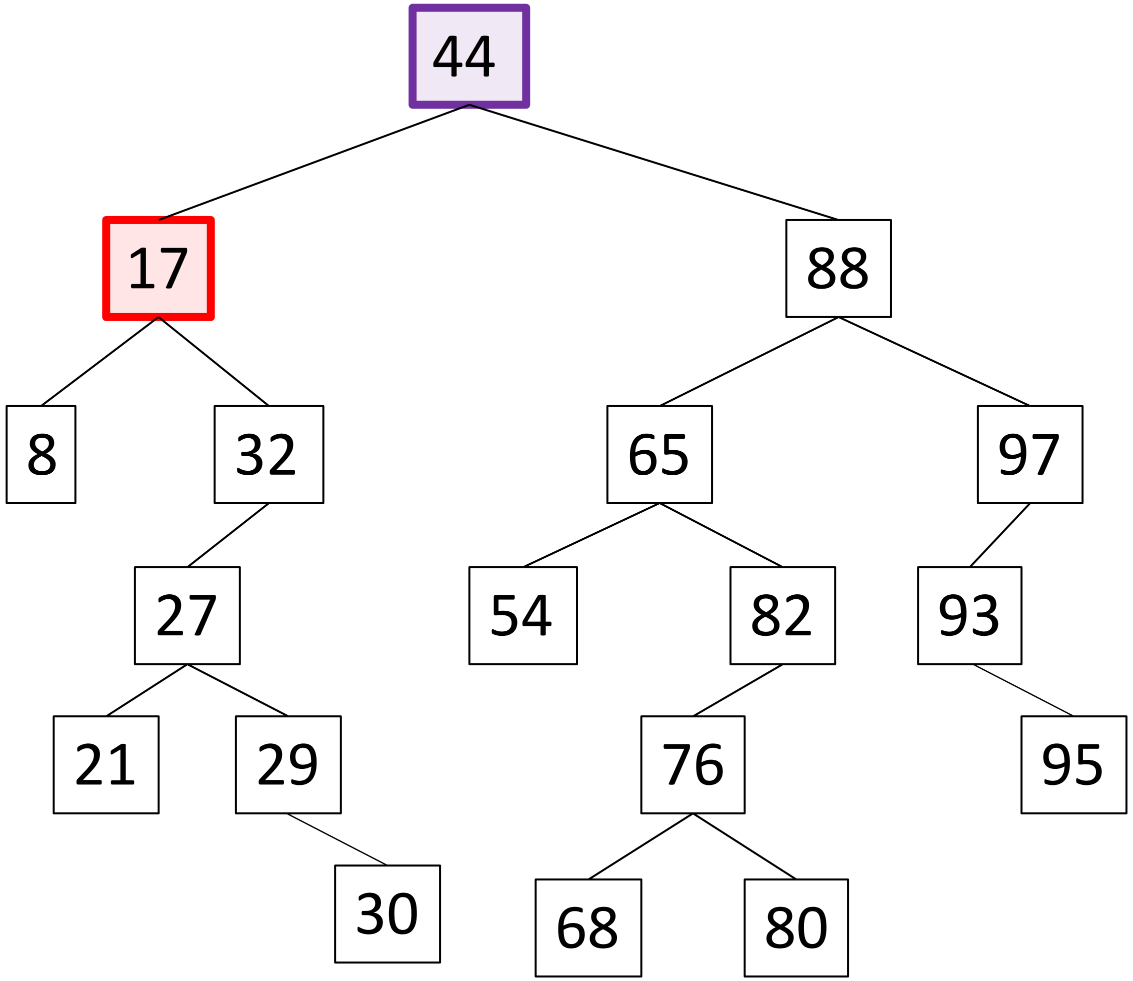


Output:
44
17
8

Binary Search Tree - Traversal

```
public void depthFirst(44) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```

```
public void depthFirst(17) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```



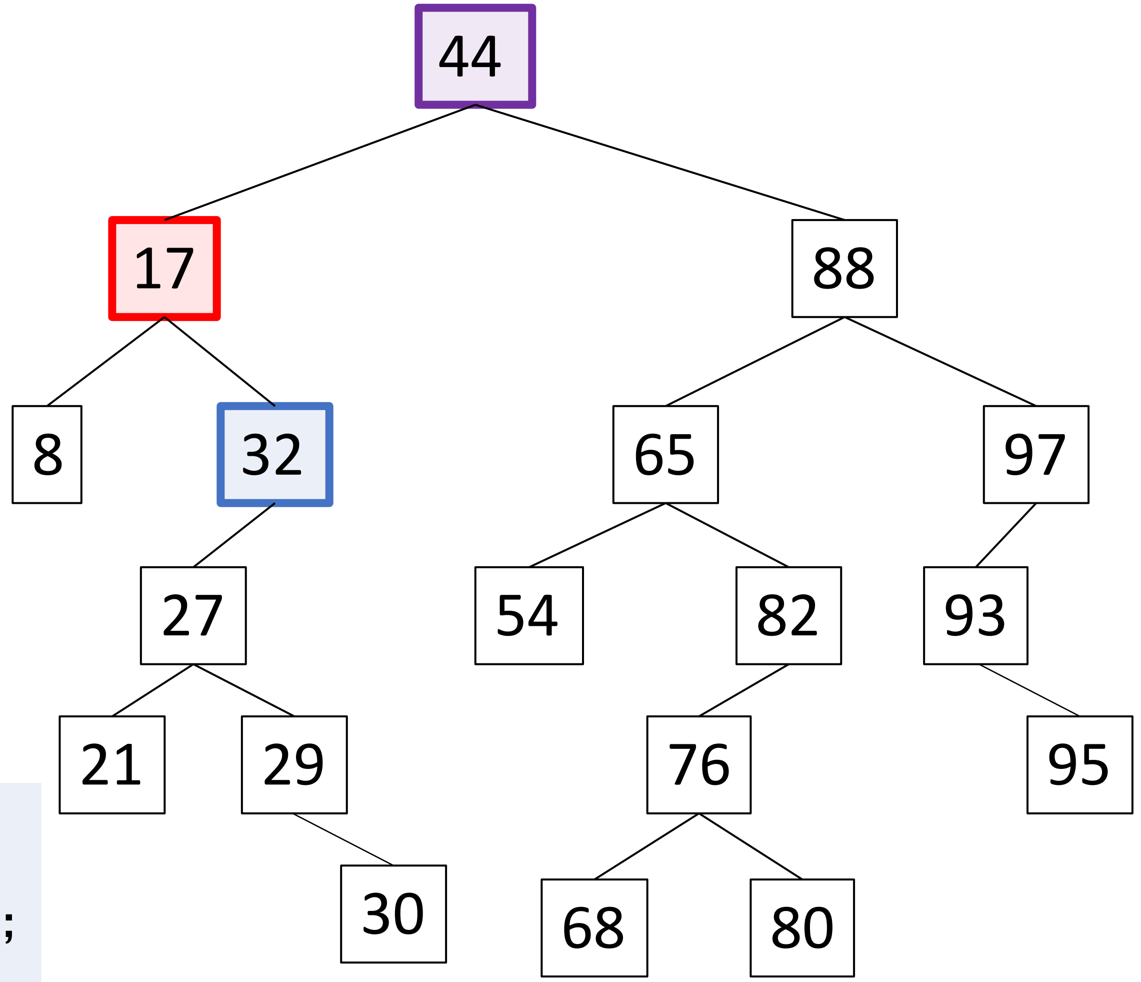
Output:
44
17
8

Binary Search Tree - Traversal

```
public void depthFirst(44) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```

```
public void depthFirst(17) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```

```
public void depthFirst(32) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```



Output:

44

17

8

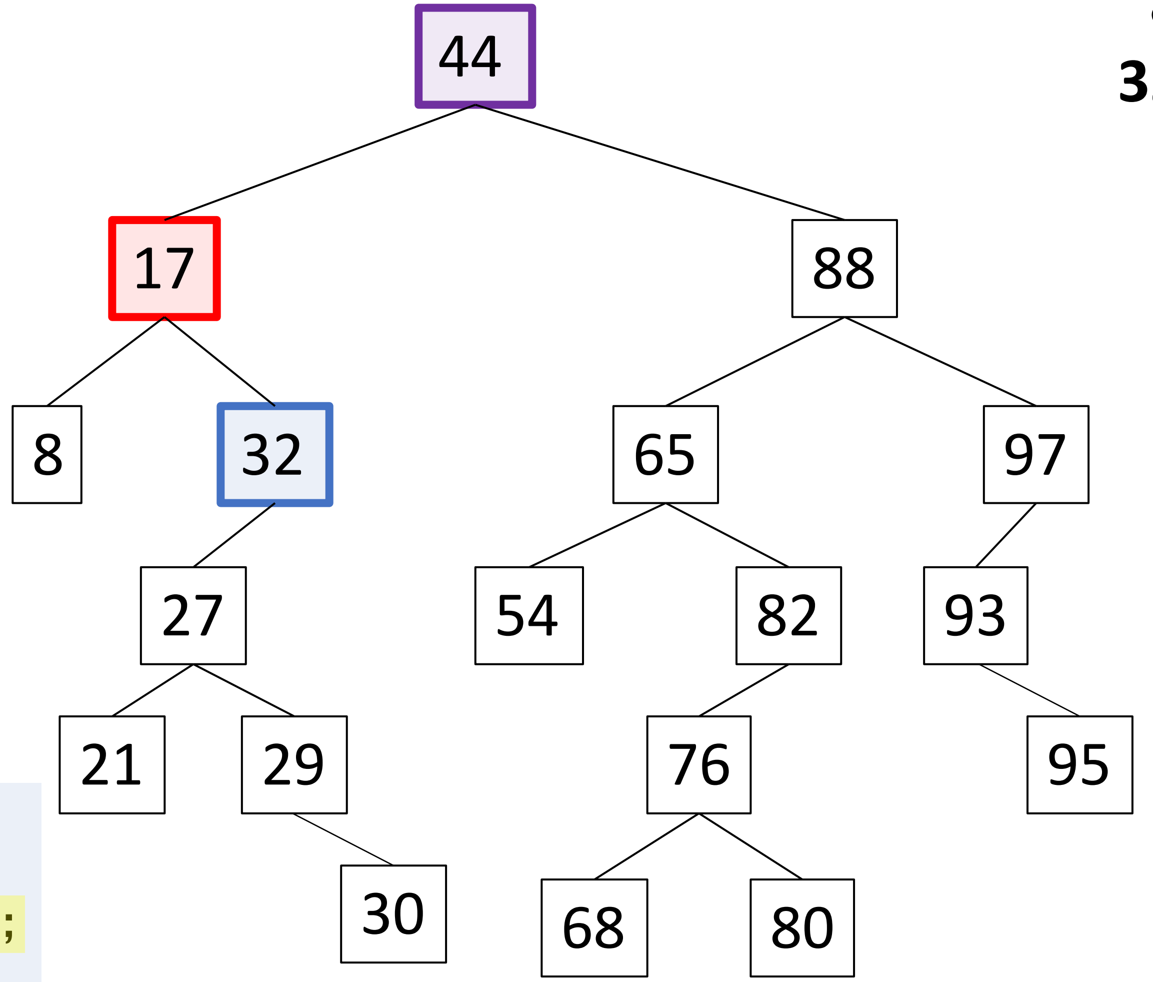
32

Binary Search Tree - Traversal

```
public void depthFirst(44) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```

```
public void depthFirst(17) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```

```
public void depthFirst(32) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```



Output:

44

17

8

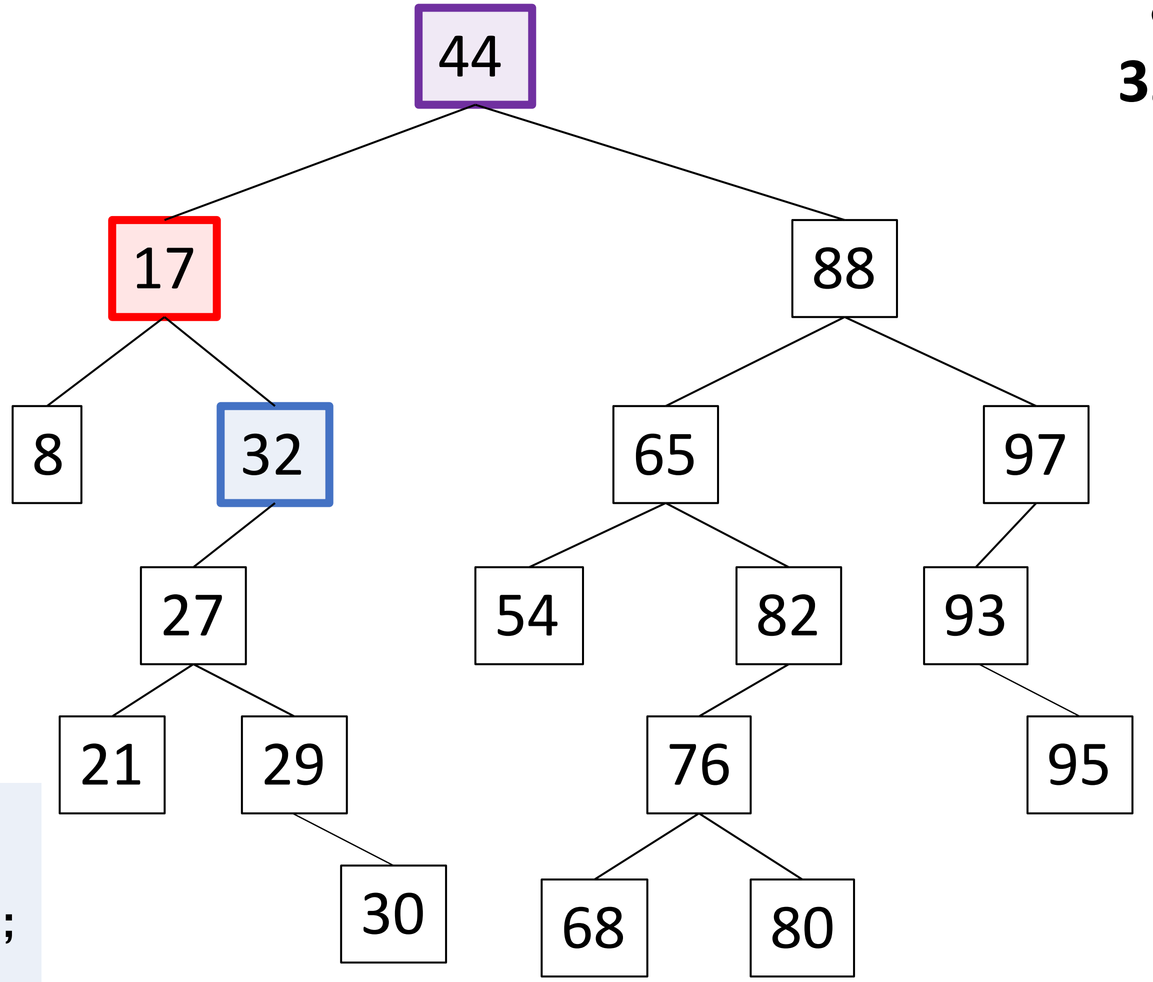
32

Binary Search Tree - Traversal

```
public void depthFirst(44) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```

```
public void depthFirst(17) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```

```
public void depthFirst(32) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```



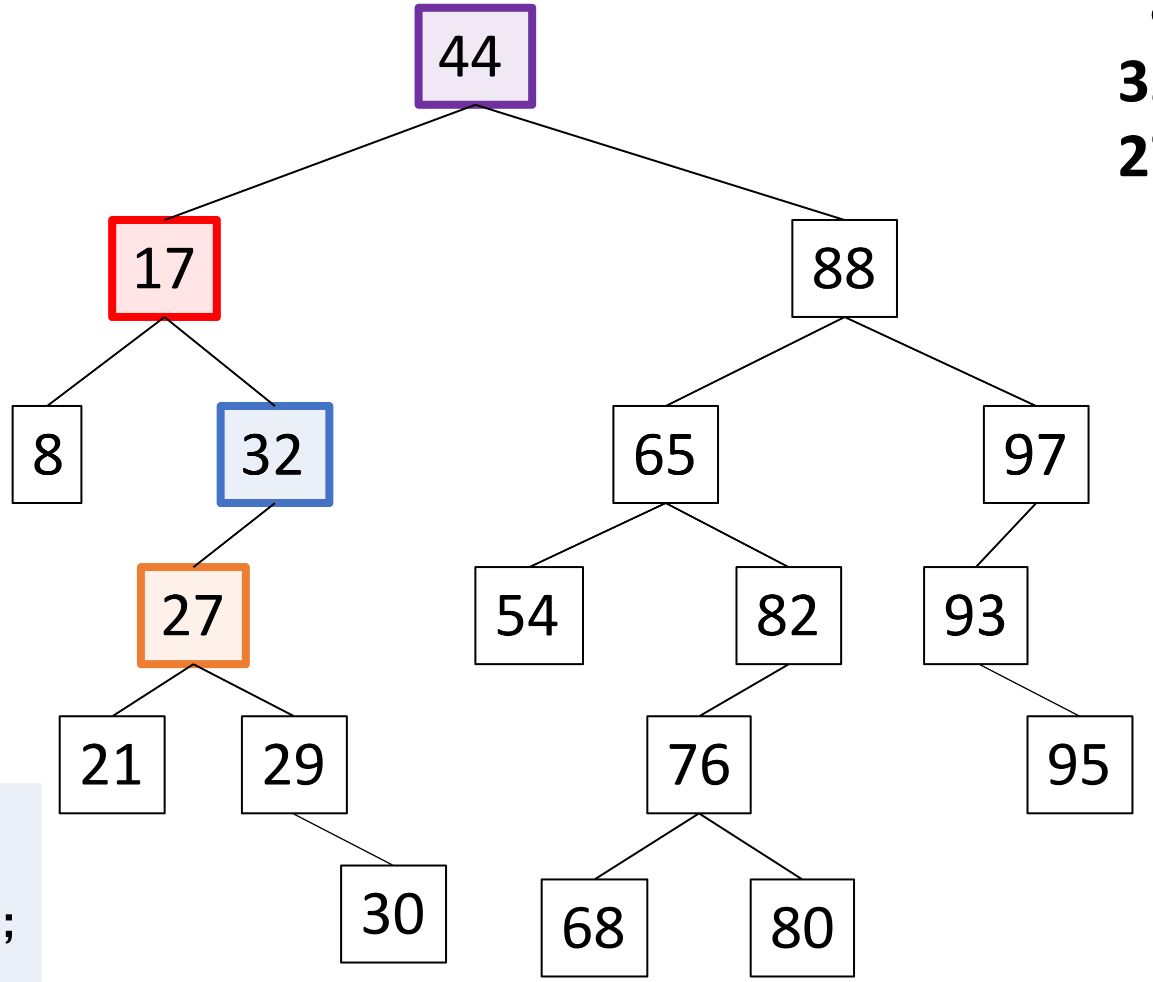
Output:
44
17
8
32
27

Binary Search Tree - Traversal

```
public void depthFirst(44) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```

```
public void depthFirst(17) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```

```
public void depthFirst(32) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```



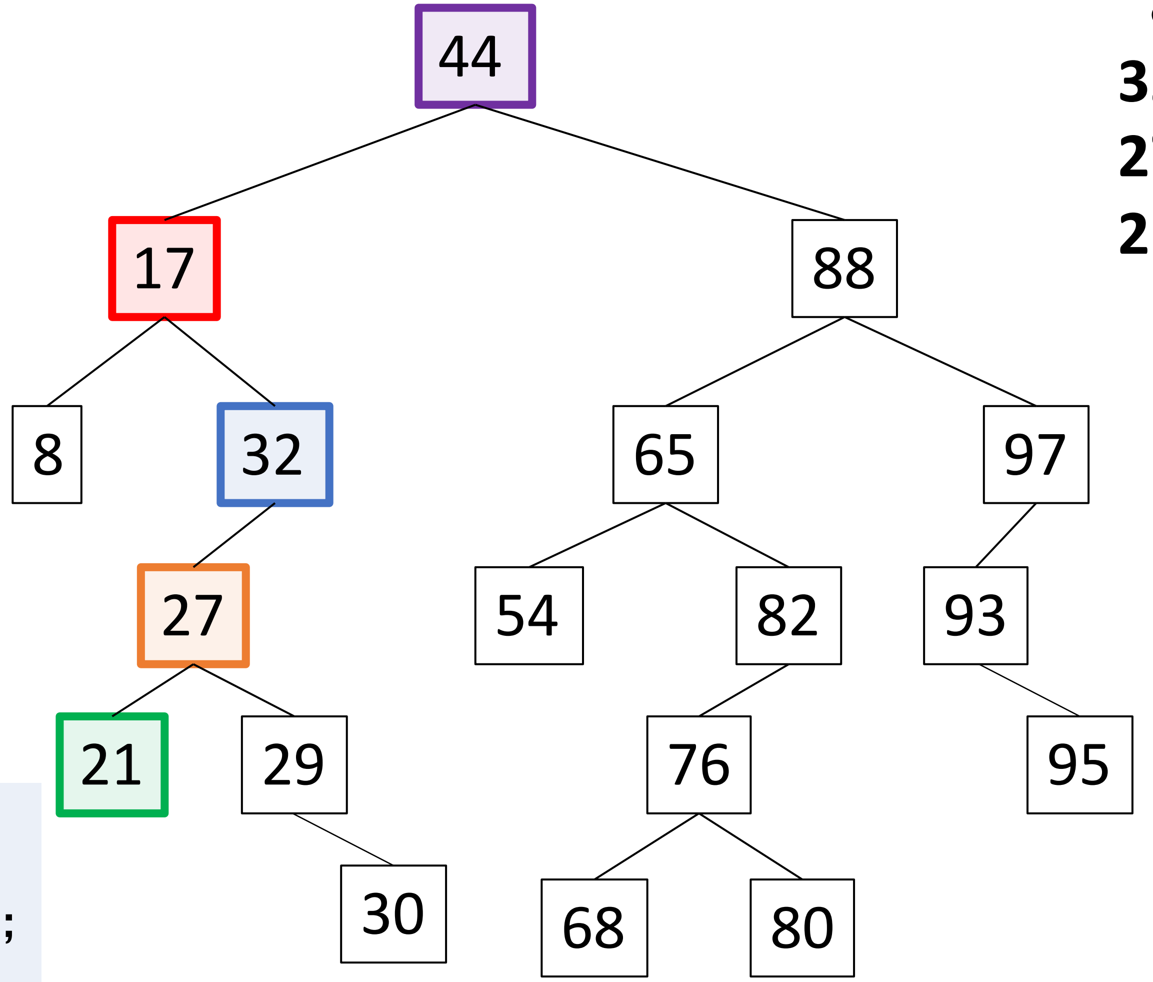
Binary Search Tree - Traversal

Output:
44
17
8
32
27
21

```
public void depthFirst(44) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```

```
public void depthFirst(17) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```

```
public void depthFirst(32) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```



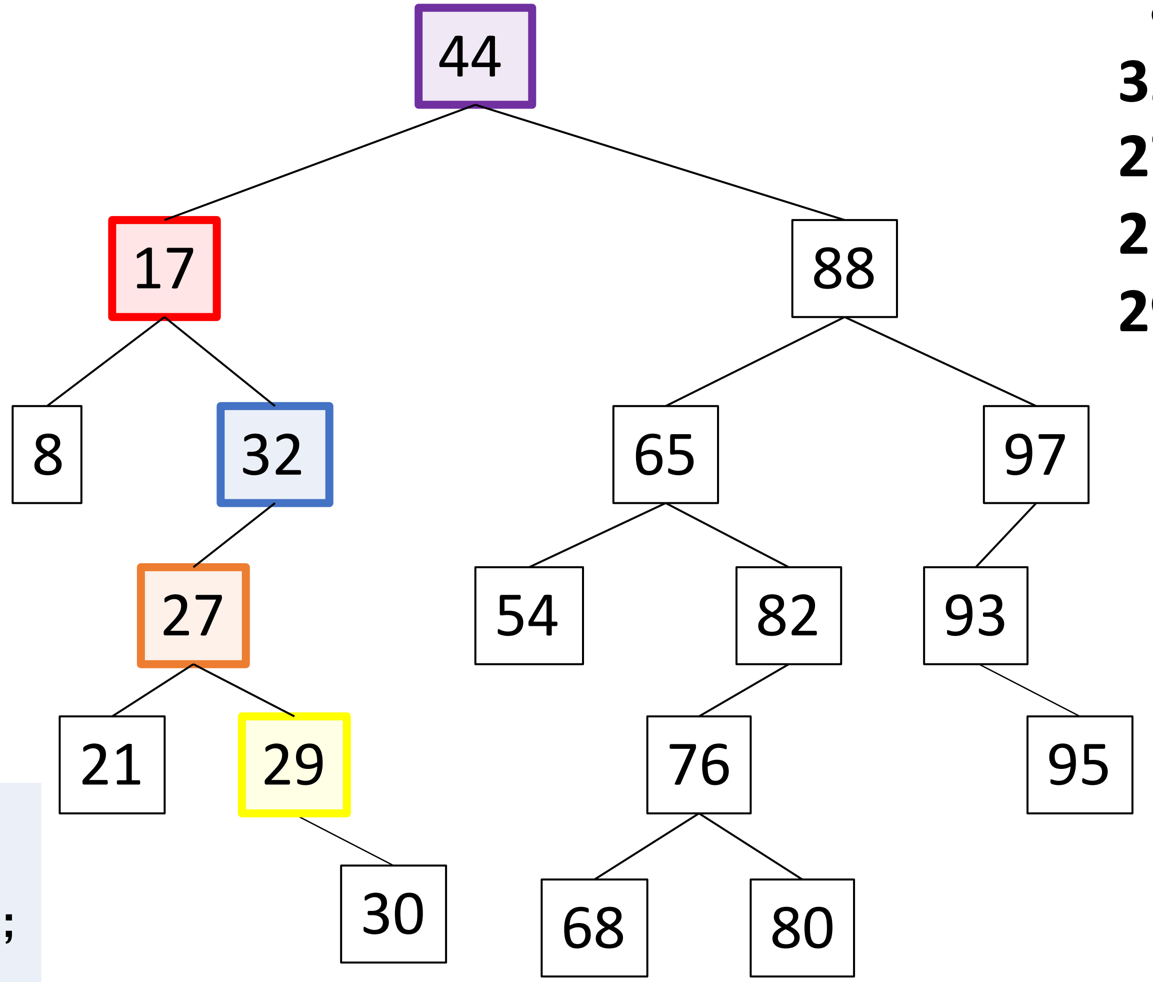
Output:
44
17
8
32
27
21
29

Binary Search Tree - Traversal

```
public void depthFirst(44) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```

```
public void depthFirst(17) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```

```
public void depthFirst(32) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```

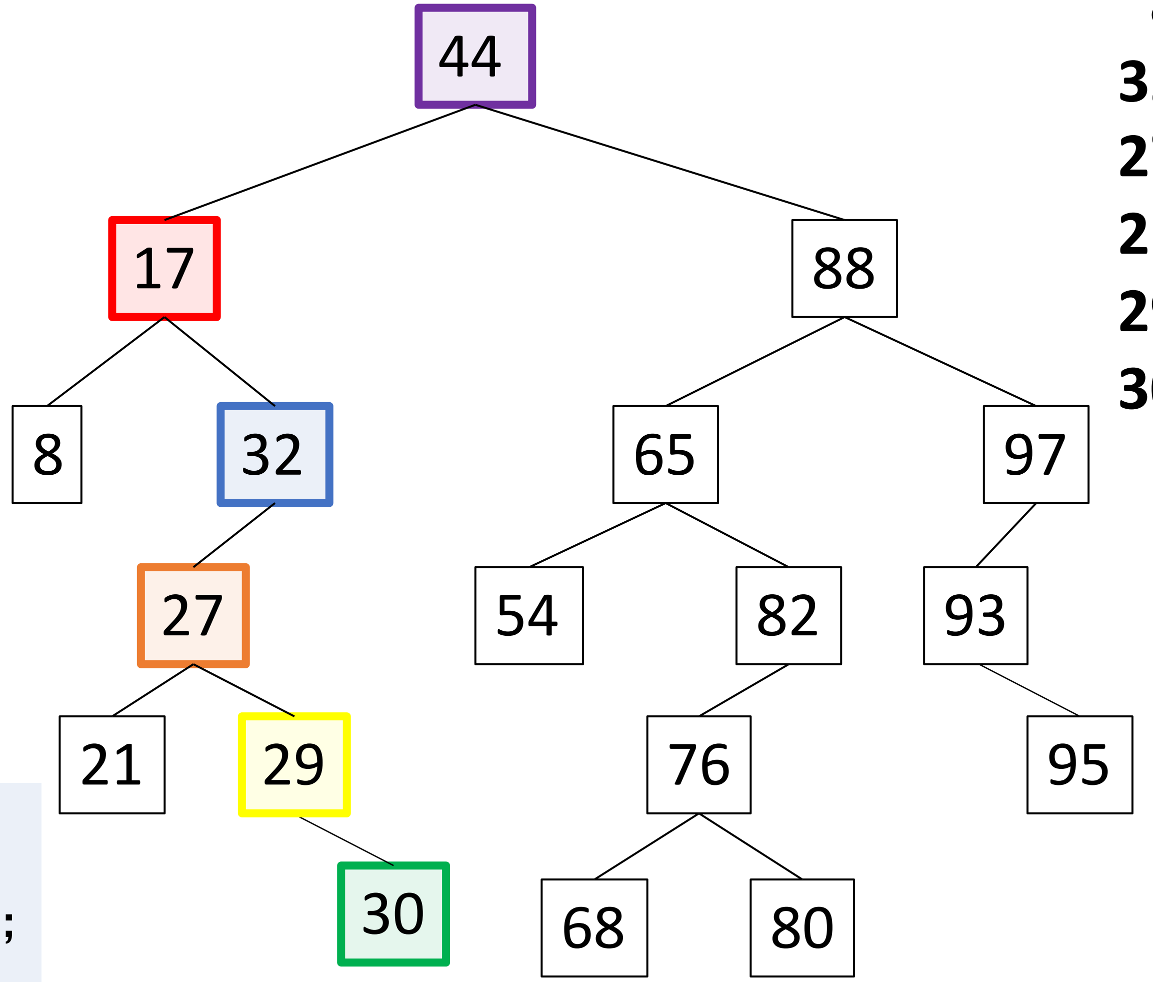


Binary Search Tree - Traversal

```
public void depthFirst(44) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```

```
public void depthFirst(17) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```

```
public void depthFirst(32) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```



Output:

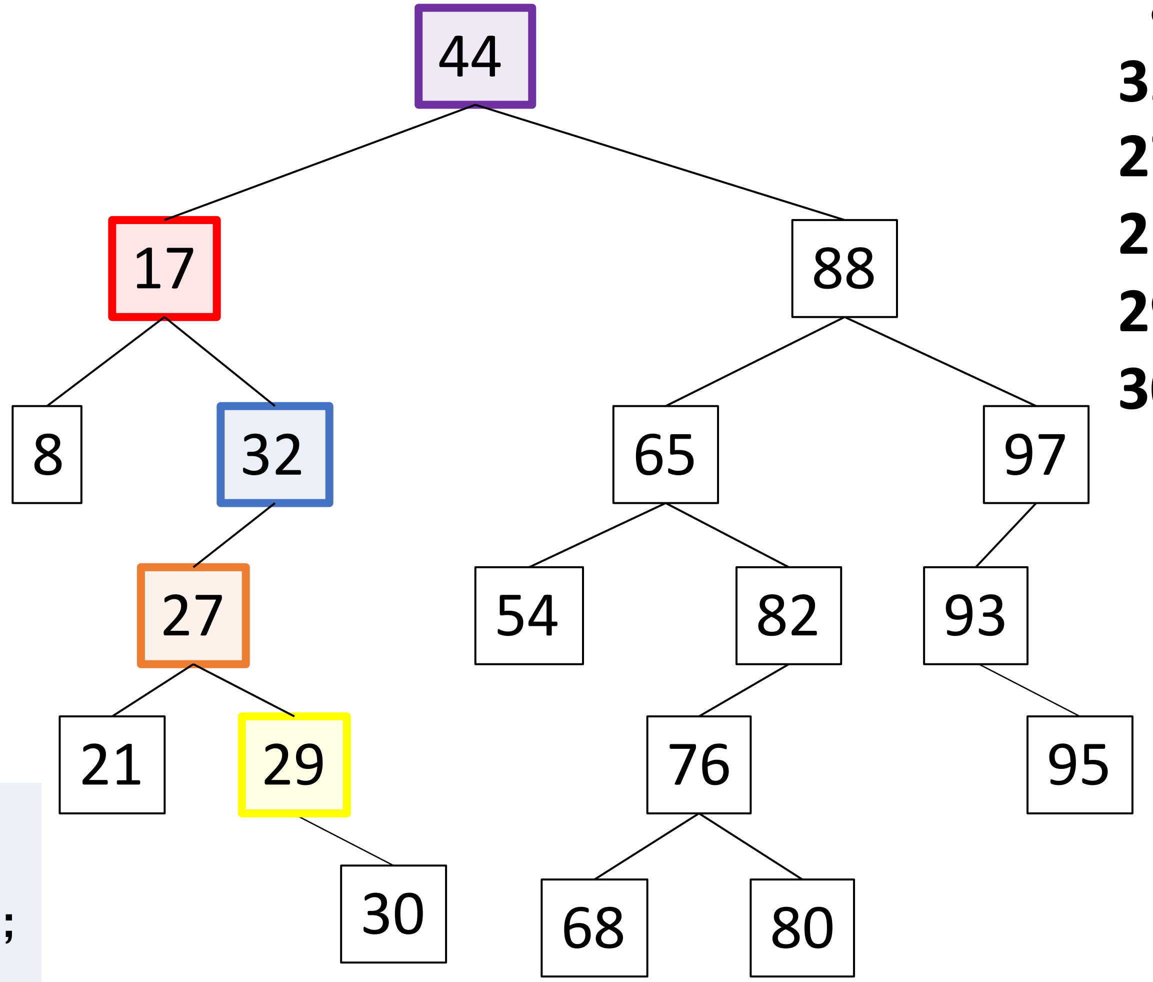
```
44
17
8
32
27
21
29
30
```

Binary Search Tree - Traversal

```
public void depthFirst(44) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```

```
public void depthFirst(17) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```

```
public void depthFirst(32) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```



Output:

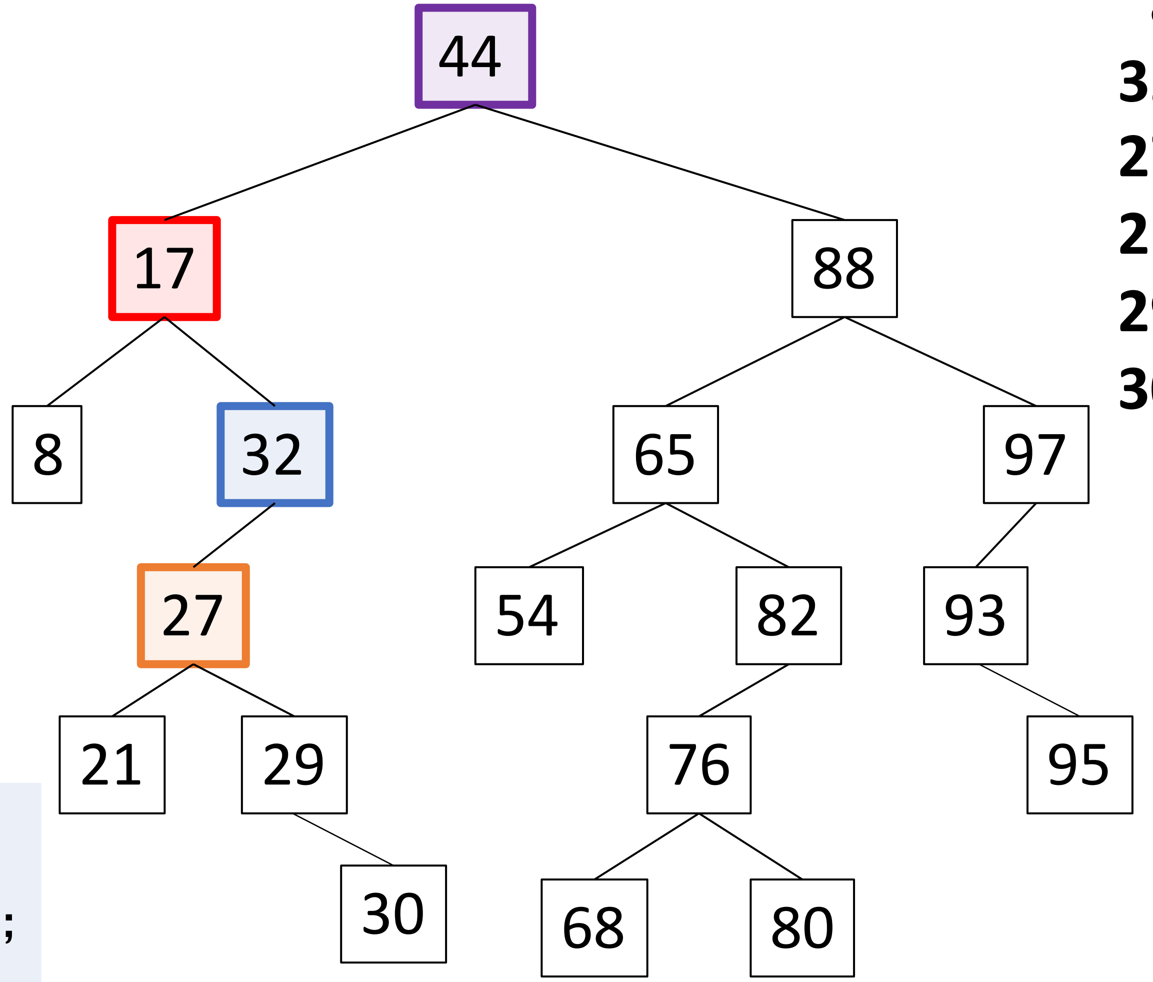
```
44
17
8
32
27
21
29
30
```

Binary Search Tree - Traversal

```
public void depthFirst(44) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```

```
public void depthFirst(17) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```

```
public void depthFirst(32) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```



Output:

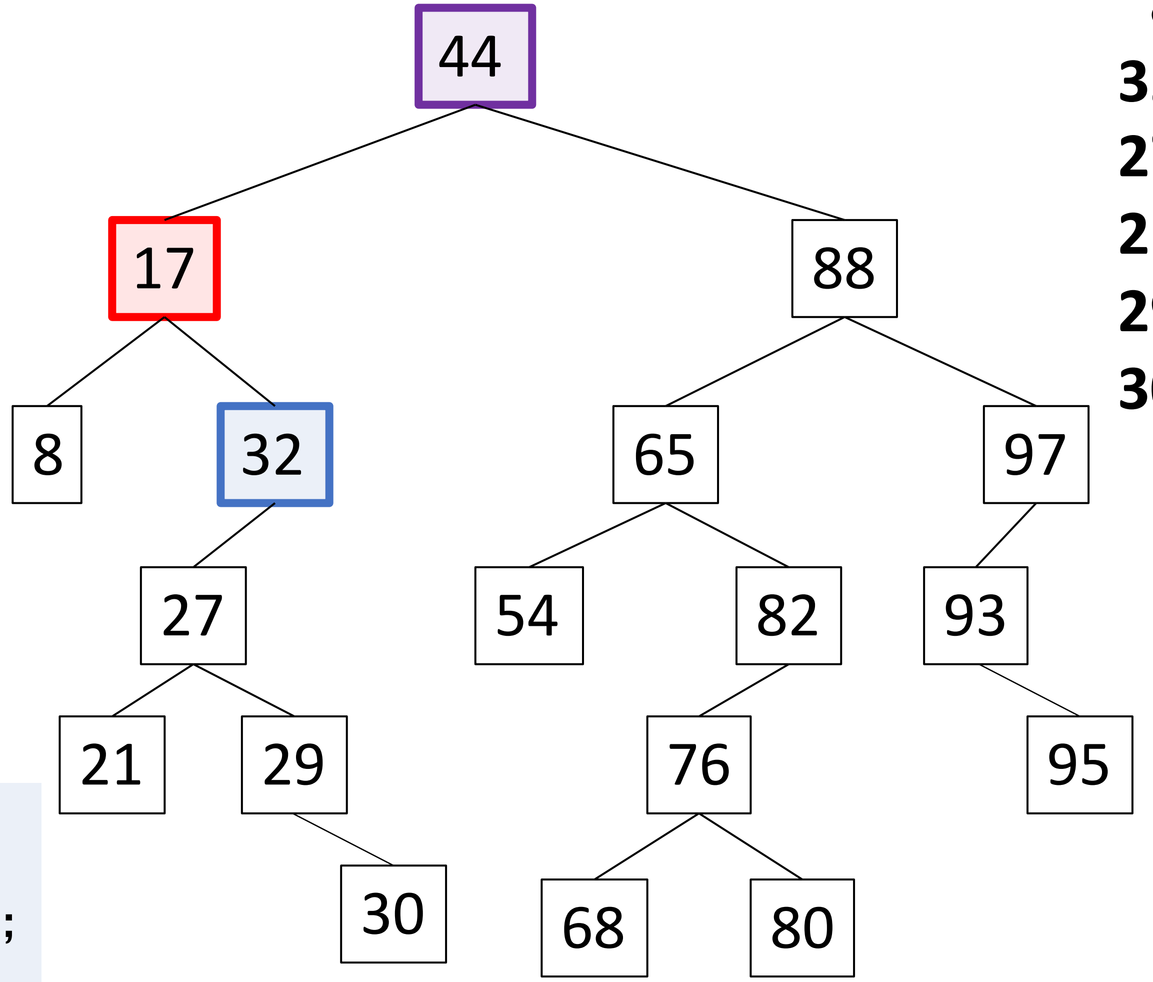
```
44
17
 8
32
27
21
29
30
```

Binary Search Tree - Traversal

```
public void depthFirst(44) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```

```
public void depthFirst(17) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```

```
public void depthFirst(32) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```



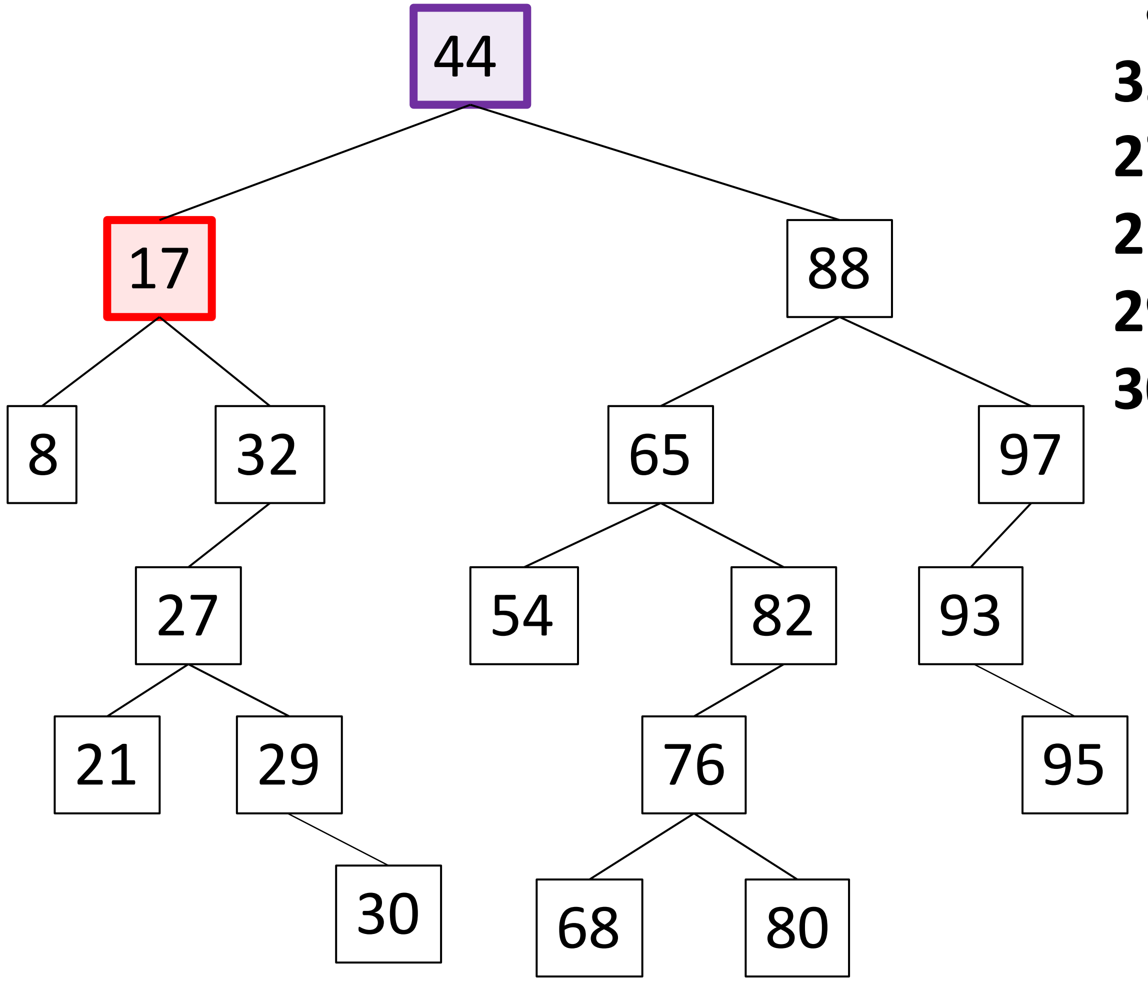
Output:

```
44
17
 8
32
27
21
29
30
```

Binary Search Tree - Traversal

```
public void depthFirst(44) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```

```
public void depthFirst(17) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```



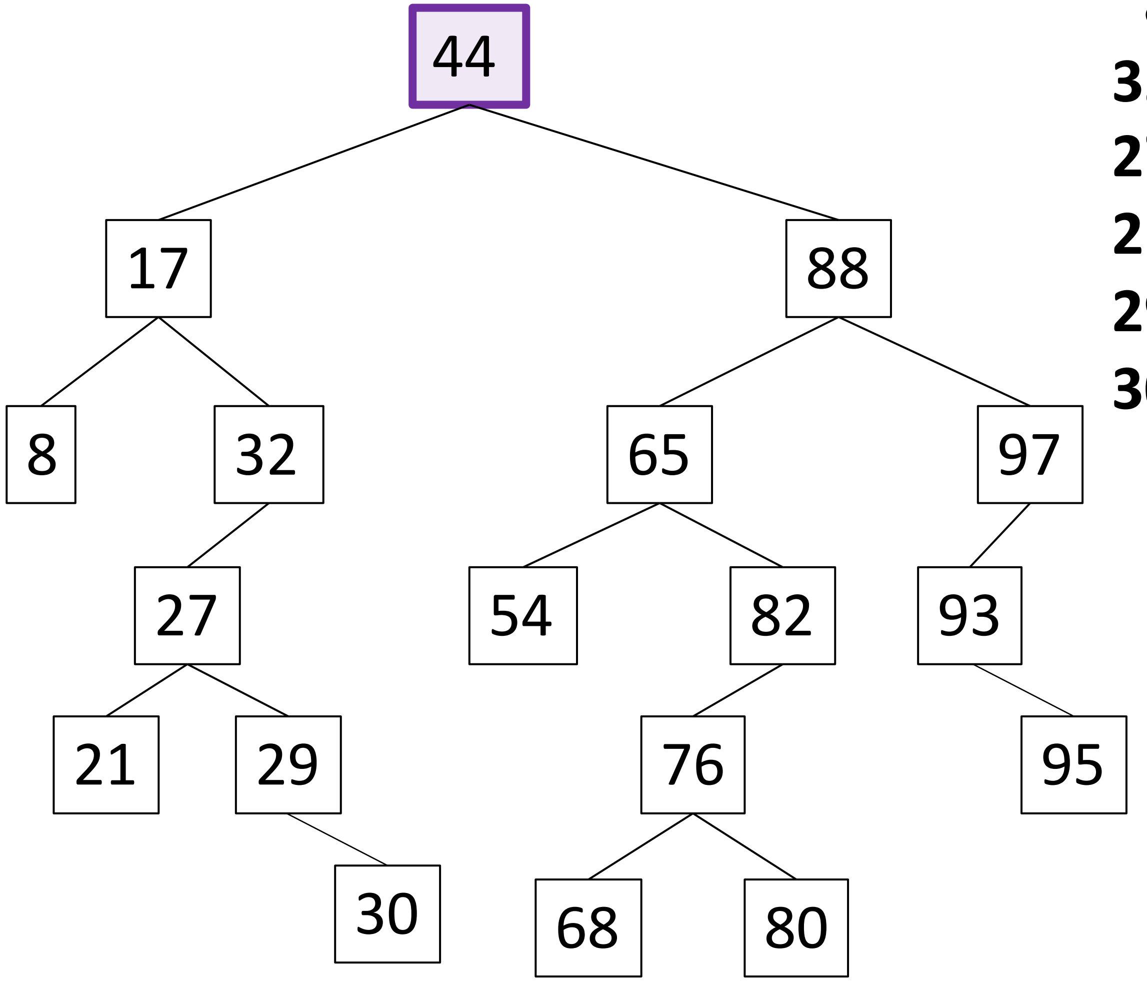
Output:

```
44
17
 8
32
27
21
29
30
```

Output:
44
17
8
32
27
21
29
30

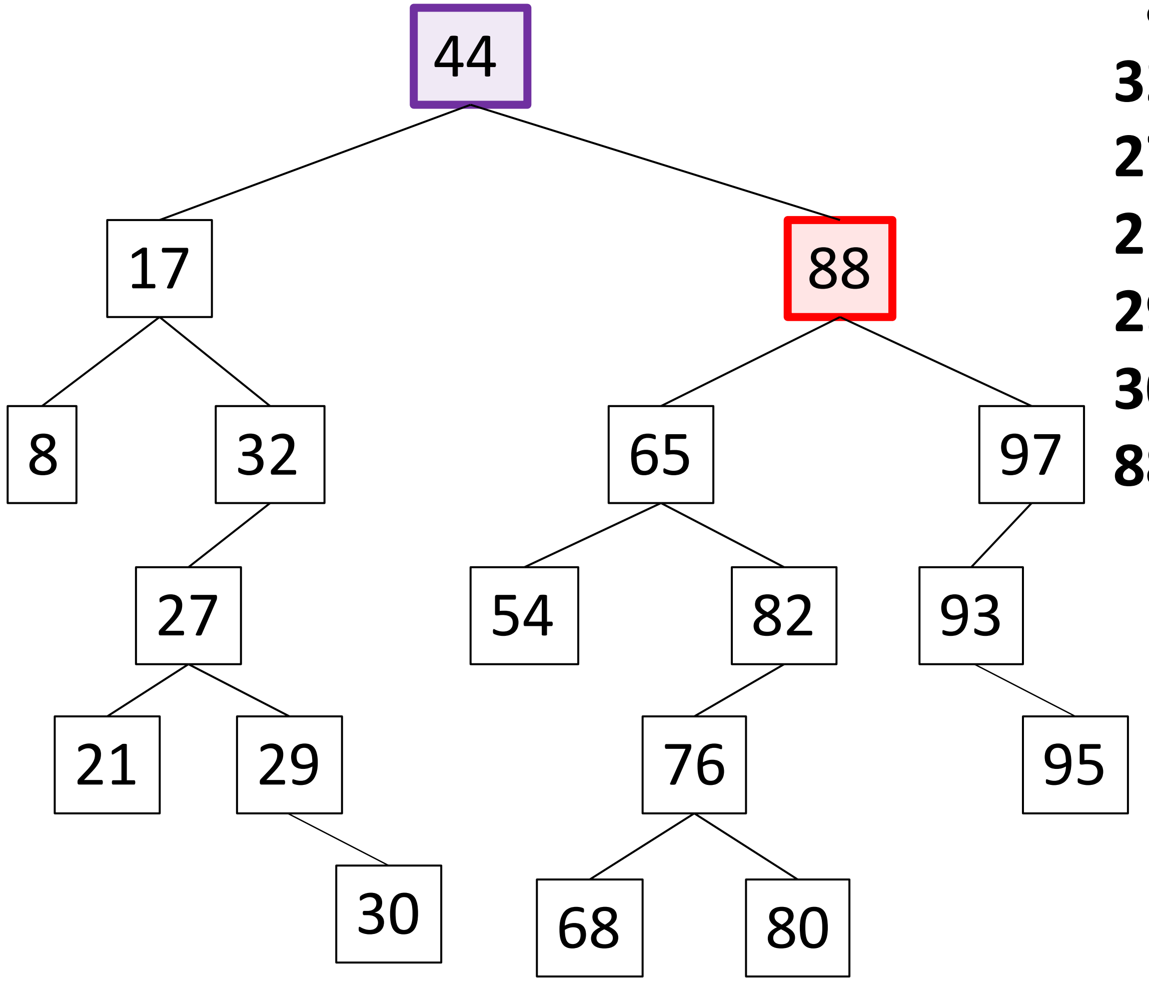
Binary Search Tree - Traversal

```
public void depthFirst(44) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```



Binary Search Tree - Traversal

```
public void depthFirst(44) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```



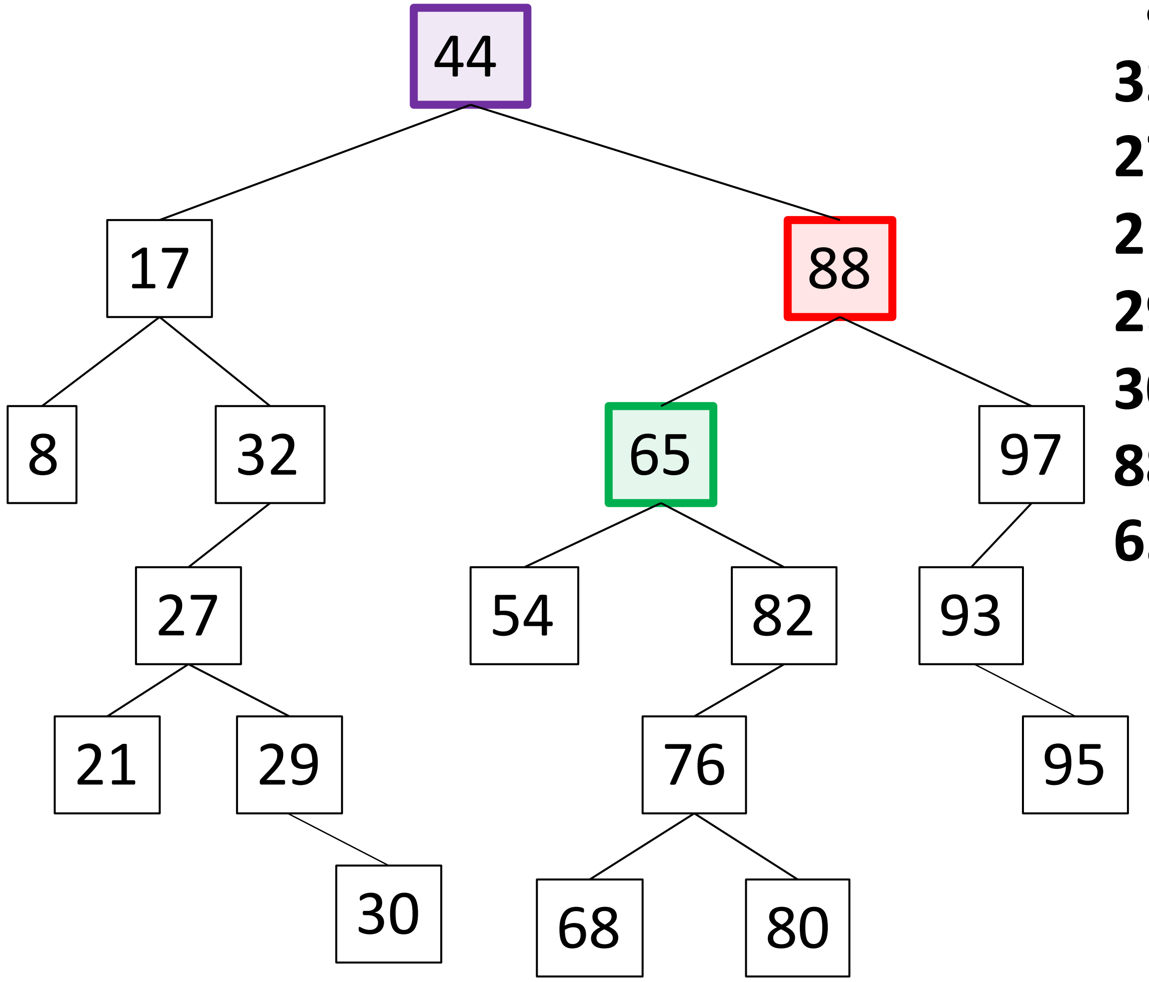
Output:

44
17
8
32
27
21
29
30
88

Output:
44
17
8
32
27
21
29
30
88
65

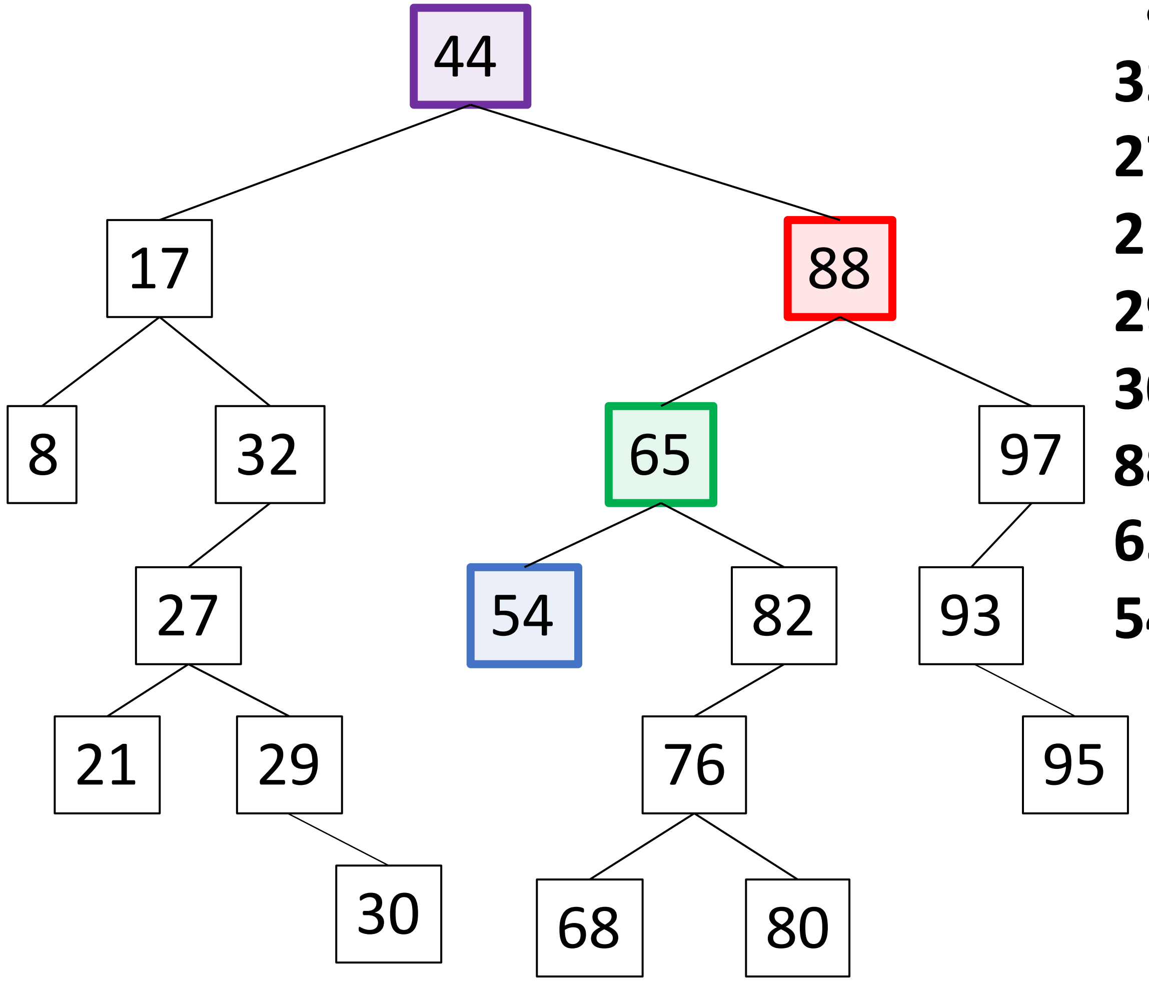
Binary Search Tree - Traversal

```
public void depthFirst(44) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```



Binary Search Tree - Traversal

```
public void depthFirst(44) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```



Output:

44
17
8
32
27
21
29
30
88
65
54

Output:
44
17
8

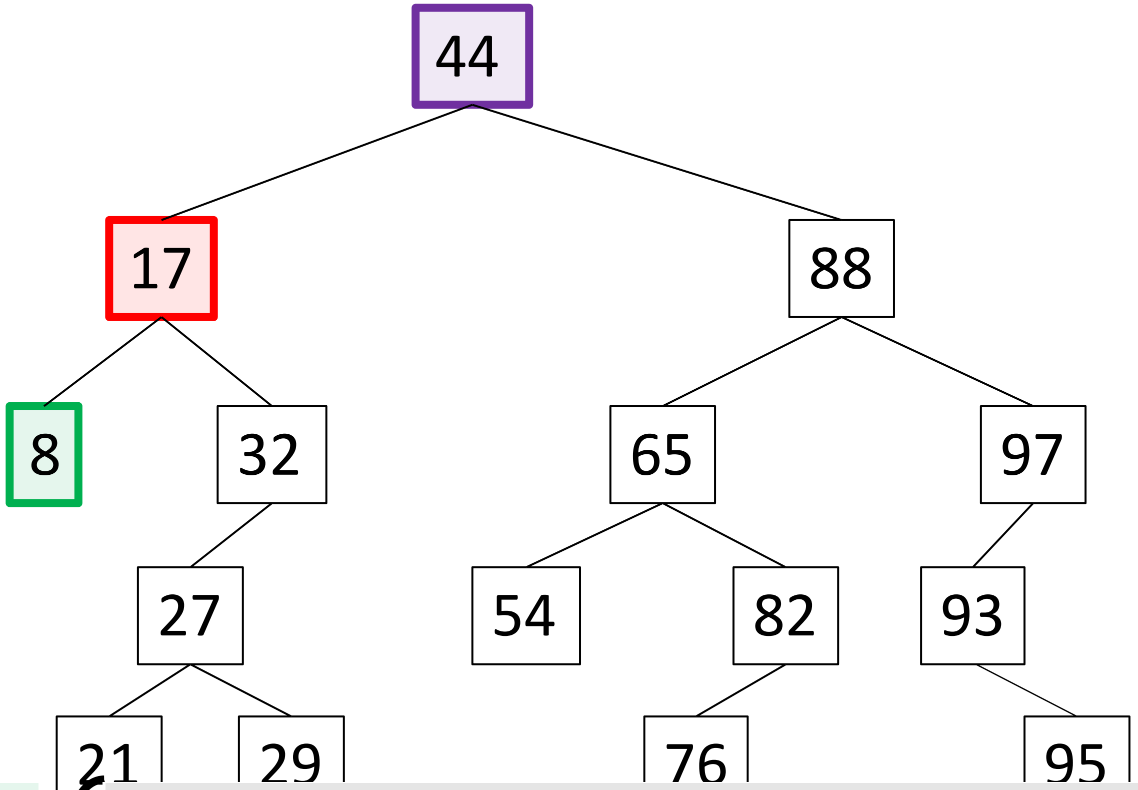
Binary Search Tree - Traversal

```
public void depthFirst(44) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```

```
public void depthFirst(17) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```

```
public void depthFirst(8) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```

```
public void depthFirst(null) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```



Output:

44

17

8

32

27

21

29

30

88

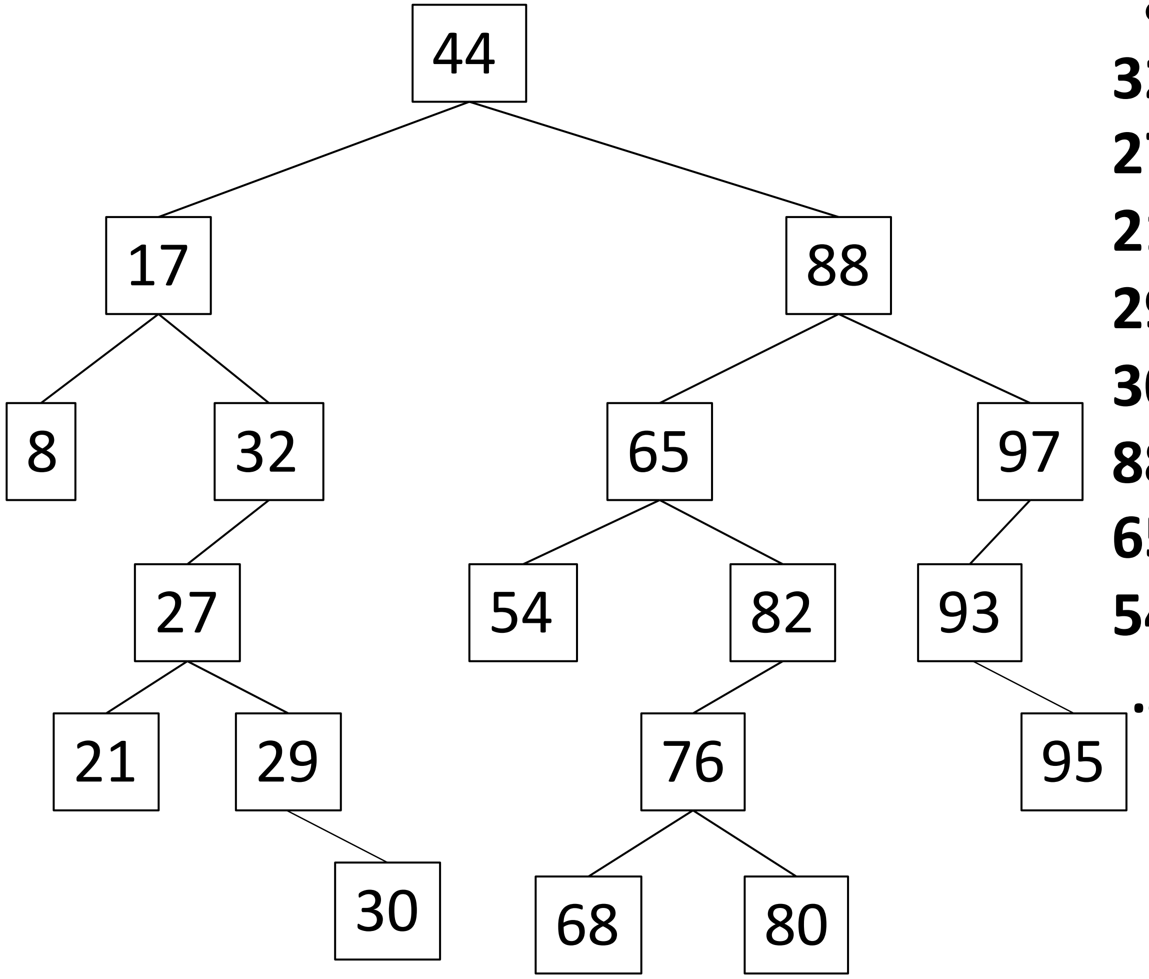
65

54

…

Binary Search Tree - Traversal

```
public void depthFirst(44) {
    if (n != null) {
        System.out.println(n.getValue());
        depthFirst(n.getLeft());
        depthFirst(n.getRight());
    }
}
```



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(currentNode.getValue() == newValue) {
              placed = true;
              System.out.println("No duplicate values allowed");
         }
         else if(newValue < currentNode.getValue()) {
              if(currentNode.getLeft() == null) {
                   currentNode.setLeft(new Node(newValue));
                   currentNode.getLeft().setParent(currentNode);
                   placed = true;
              }
              else {
                   currentNode = currentNode.getLeft();
              }
        }
        else {
             if(currentNode.getRight() == null) {
                   currentNode.setRight(new Node(newValue));
                   currentNode.getRight().setParent(currentNode);
                   placed = true;
             }
             else {
                   currentNode = currentNode.getRight();
             }
      }
   }
```

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(currentNode.getValue() == newValue) {
              placed = true;
              System.out.println("No duplicate values allowed");
         }
         else if(newValue < currentNode.getValue()) {
              if(currentNode.getLeft() == null) {
                   currentNode.setLeft(new Node(newValue));
                   currentNode.getLeft().setParent(currentNode);
                   placed = true;
              }
              else {
                   currentNode = currentNode.getLeft();
              }
        }
        else {
             if(currentNode.getRight() == null) {
                   currentNode.setRight(new Node(newValue));
                   currentNode.getRight().setParent(currentNode);
                   placed = true;
             }
             else {
                   currentNode = currentNode.getRight();
             }
      }
   }
```

We repeatedly move left or right until we find the correct spot for our new node

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(currentNode.getValue() == newValue) {
              placed = true;
              System.out.println("No duplicate values allowed");
         }
         else if(newValue < currentNode.getValue()) {
              if(currentNode.getLeft() == null) {
                   currentNode.setLeft(new Node(newValue));
                   currentNode.getLeft().setParent(currentNode);
                   placed = true;
              }
              else {
                   currentNode = currentNode.getLeft();
              }
        }
        else {
             if(currentNode.getRight() == null) {
                   currentNode.setRight(new Node(newValue));
                   currentNode.getRight().setParent(currentNode);
                   placed = true;
             }
             else {
                   currentNode = currentNode.getRight();
             }
      }
   }
}
```

We repeatedly move left or right until we find the correct spot for our new node

Once we find the correct spot, we update some pointers

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(currentNode.getValue() == newValue) {
              placed = true;
              System.out.println("No duplicate values allowed");
         }
         else if(newValue < currentNode.getValue()) {
              if(currentNode.getLeft() == null) {
                   currentNode.setLeft(new Node(newValue));
                   currentNode.getLeft().setParent(currentNode);
                   placed = true;
              }
              else {
                   currentNode = currentNode.getLeft();
              }
        }
        else {
             if(currentNode.getRight() == null) {
                   currentNode.setRight(new Node(newValue));
                   currentNode.getRight().setParent(currentNode);
                   placed = true;
             }
             else {
                   currentNode = currentNode.getRight();
             }
      }
   }
```

Running time?

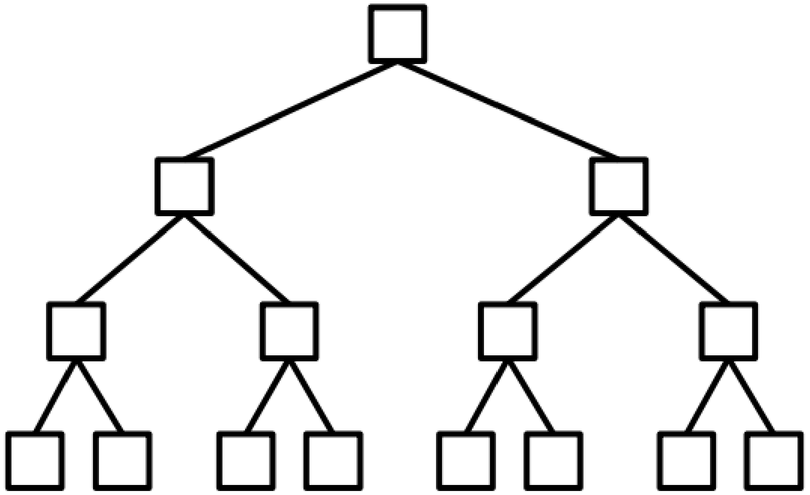
Binary Search Tree - Insertion

```
public void insert(int newValue) {
    if(root == null) {
        root = new Node(newValue);
    }
    else {
    Node currentNode = root;
    boolean placed = false;
    while(!placed) {
         if(currentNode.getValue() == newValue) {
              placed = true;
              System.out.println("No duplicate values allowed");
         }
         else if(newValue < currentNode.getValue()) {
              if(currentNode.getLeft() == null) {
                   currentNode.setLeft(new Node(newValue));
                   currentNode.getLeft().setParent(currentNode);
                   placed = true;
              }
              else {
                   currentNode = currentNode.getLeft();
              }
        }
        else {
             if(currentNode.getRight() == null) {
                   currentNode.setRight(new Node(newValue));
                   currentNode.getRight().setParent(currentNode);
                   placed = true;
             }
             else {
                   currentNode = currentNode.getRight();
             }
       }
    }
}
```

Running time?

We will always be inserting a leaf node, so worst cast scenario we will need to travel the **height** of the tree

If we have a "balanced tree" the height of the tree, is log(n)   n = # of nodes



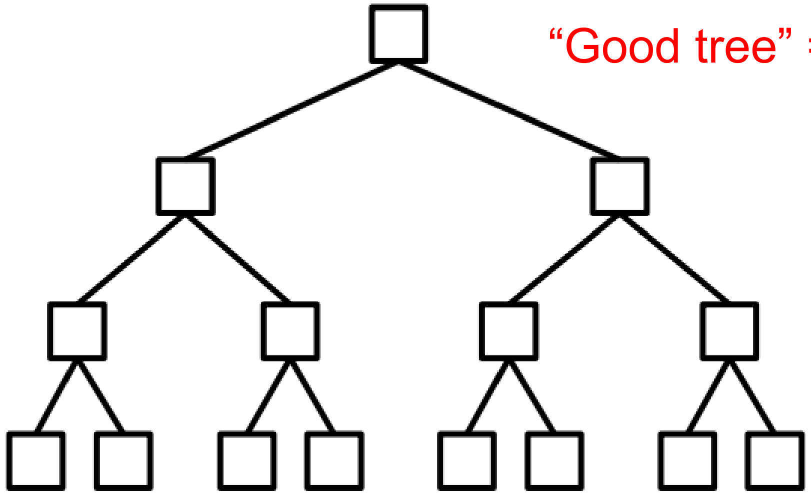
Binary Search Tree - Insertion

```
public void insert(int newValue) {
    if(root == null) {
        root = new Node(newValue);
    }
    else {
    Node currentNode = root;
    boolean placed = false;
    while(!placed) {
         if(currentNode.getValue() == newValue) {
              placed = true;
              System.out.println("No duplicate values allowed");
         }
         else if(newValue < currentNode.getValue()) {
              if(currentNode.getLeft() == null) {
                   currentNode.setLeft(new Node(newValue));
                   currentNode.getLeft().setParent(currentNode);
                   placed = true;
              }
              else {
                   currentNode = currentNode.getLeft();
              }
        }
        else {
             if(currentNode.getRight() == null) {
                   currentNode.setRight(new Node(newValue));
                   currentNode.getRight().setParent(currentNode);
                   placed = true;
             }
             else {
                   currentNode = currentNode.getRight();
             }
       }
   }
}
```

Running time?

We will always be inserting a leaf node, so worst cast scenario we will need to travel the **height** of the tree

If we have a "balanced tree" the height of the tree, is log(n)   n = # of nodes



"Good tree" = **O(logn)**

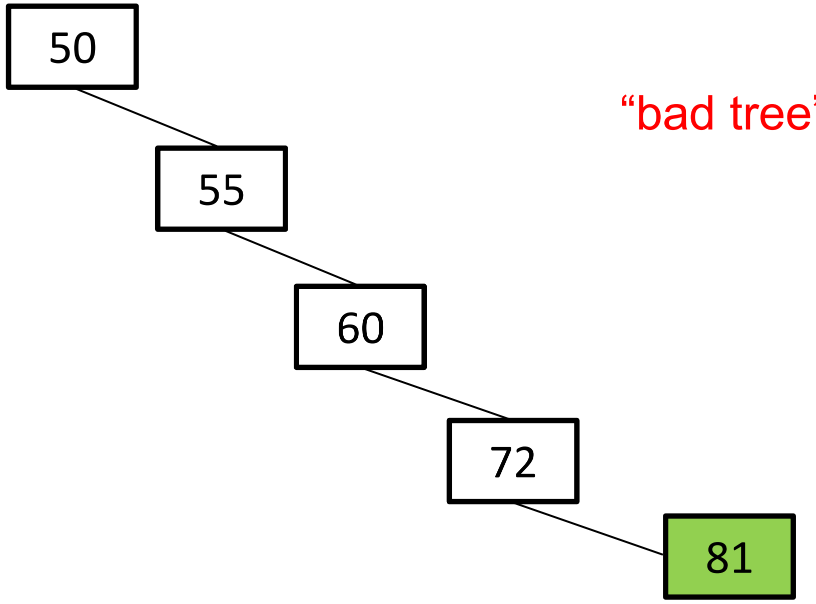
Binary Search Tree - Insertion

```java
public void insert(int newValue) {
    if(root == null) {
        root = new Node(newValue);
    }
    else {
    Node currentNode = root;
    boolean placed = false;
    while(!placed) {
         if(currentNode.getValue() == newValue) {
              placed = true;
              System.out.println("No duplicate values allowed");
         }
         else if(newValue < currentNode.getValue()) {
              if(currentNode.getLeft() == null) {
                   currentNode.setLeft(new Node(newValue));
                   currentNode.getLeft().setParent(currentNode);
                   placed = true;
              }
              else {
                   currentNode = currentNode.getLeft();
              }
        }
        else {
             if(currentNode.getRight() == null) {
                   currentNode.setRight(new Node(newValue));
                   currentNode.getRight().setParent(currentNode);
                   placed = true;
             }
             else {
                   currentNode = currentNode.getRight();
             }
       }
   }
}
```

Running time?

We will always be inserting a leaf node, so worst cast scenario we will need to travel the **height** of the tree

If we have a "bad tree" the height of the tree, is O(n-1)   n = # of nodes



"bad tree" = **O(n)**

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(currentNode.getValue() == newValue) {
              placed = true;
              System.out.println("No duplicate values allowed");
         }
         else if(newValue < currentNode.getValue()) {
              if(currentNode.getLeft() == null) {
                   currentNode.setLeft(new Node(newValue));
                   currentNode.getLeft().setParent(currentNode);
                   placed = true;
              }
              else {
                   currentNode = currentNode.getLeft();
              }
        }
        else {
             if(currentNode.getRight() == null) {
                   currentNode.setRight(new Node(newValue));
                   currentNode.getRight().setParent(currentNode);
                   placed = true;
             }
             else {
                   currentNode = currentNode.getRight();
             }
      }
   }
```

Running time?

We will always be inserting a leaf node, so worst cast scenario we will need to travel the **height** of the tree

"Bad" tree → O(n)
"Good" tree → O(logn)

O(h) → h = height of tree

Running time for adding to an array?

O(n)

Binary Search Tree - Insertion

```
public void insert(int newValue) {
    if(ro
        r
    }
    else
    Node
    boole
    while




            }
            e




        }
    }
```

If we can find a way to keep a tree “balanced”, we can achieve **O(logn)** insertion time, and **O(logn)** searching time

Running time?

We will always be inserting a leaf node, so worst cast scenario we will need to travel the **height** of the tree

“Bad” tree → O(n)
“Good” tree → O(logn)

O(h) → h = height of tree

Running time for adding to an array?

O(n) 🤔

Binary Search Tree - Insertion

```
public void insert(int newValue) {
    if(ro
        r
    }
    else
    Node
    boole
    while



        }
        e



      }
  }
```

If we can find a way to keep a tree “balanced”, we can achieve **O(logn)** insertion time, and **O(logn)** searching time

There is a way! Coming soon

Running time?

We will always be inserting a leaf node, so worst cast scenario we will need to travel the **height** of the tree

“Bad” tree → O(n)
“Good” tree → O(logn)

O(h) → h = height of tree

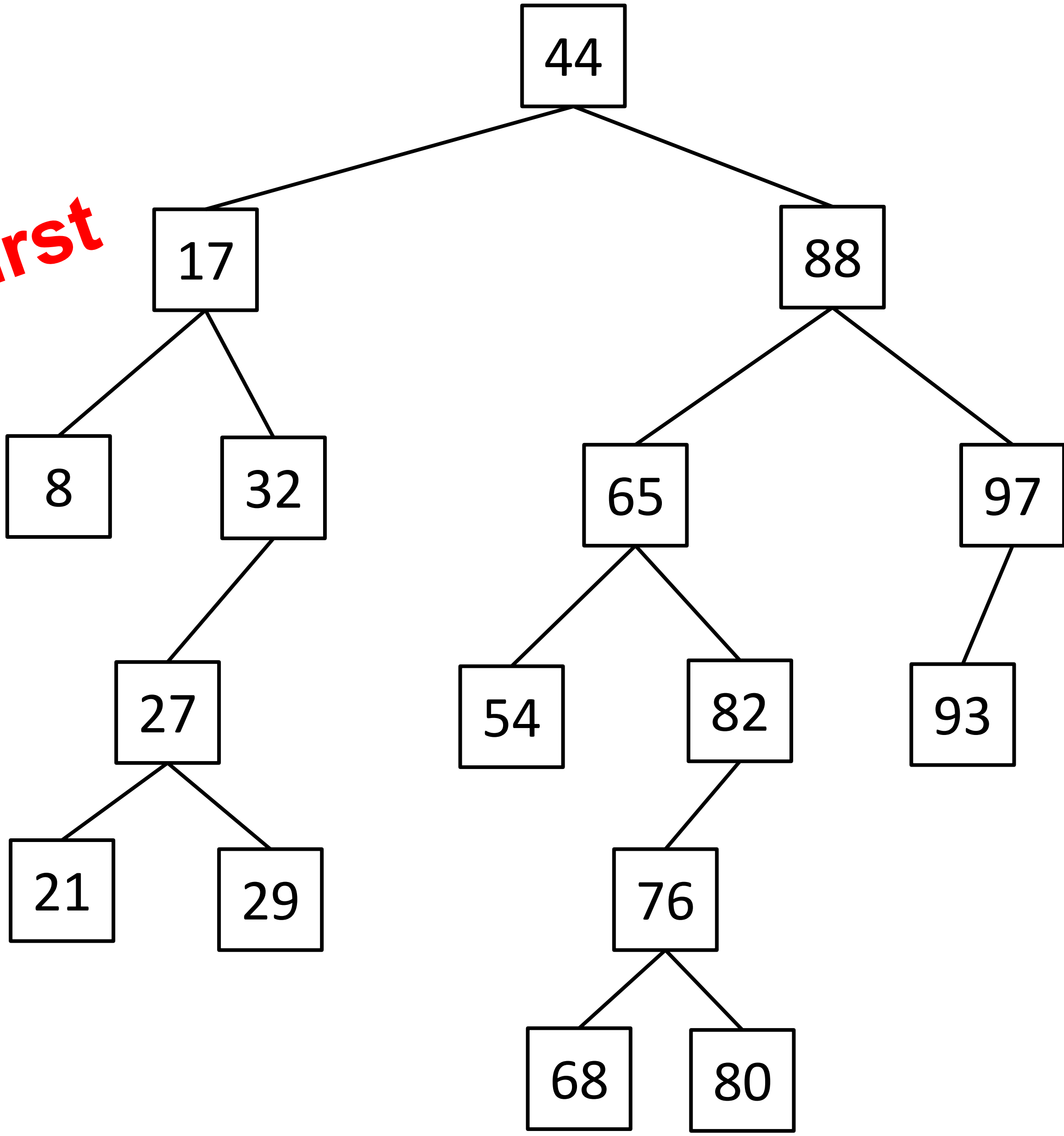
Running time for adding to an array?

O(n)

Binary Search Tree- Traversal

```java
public void depthFirst(Node n) {
  if(n != null) {
     System.out.println(n.getValue());
     depthFirst(n.getLeft());
     depthFirst(n.getRight());
  }
}
```

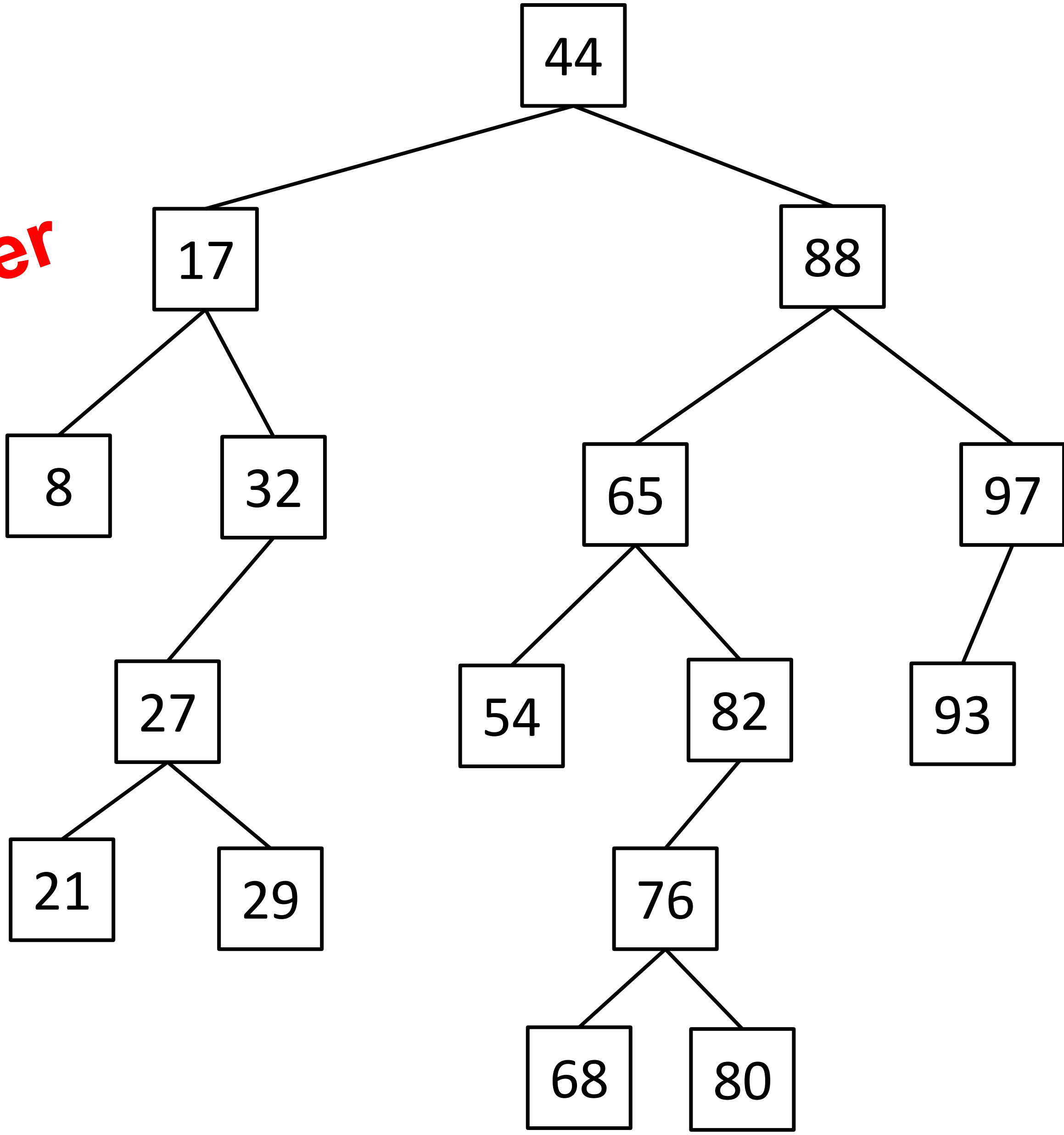
Depth First



Binary Search Tree- Traversal

```
public void preorder(Node n) {
  if(n != null) {
    System.out.println(n.getValue());
    preorder(n.getLeft());
    preorder(n.getRight());
  }
}
```

Preorder

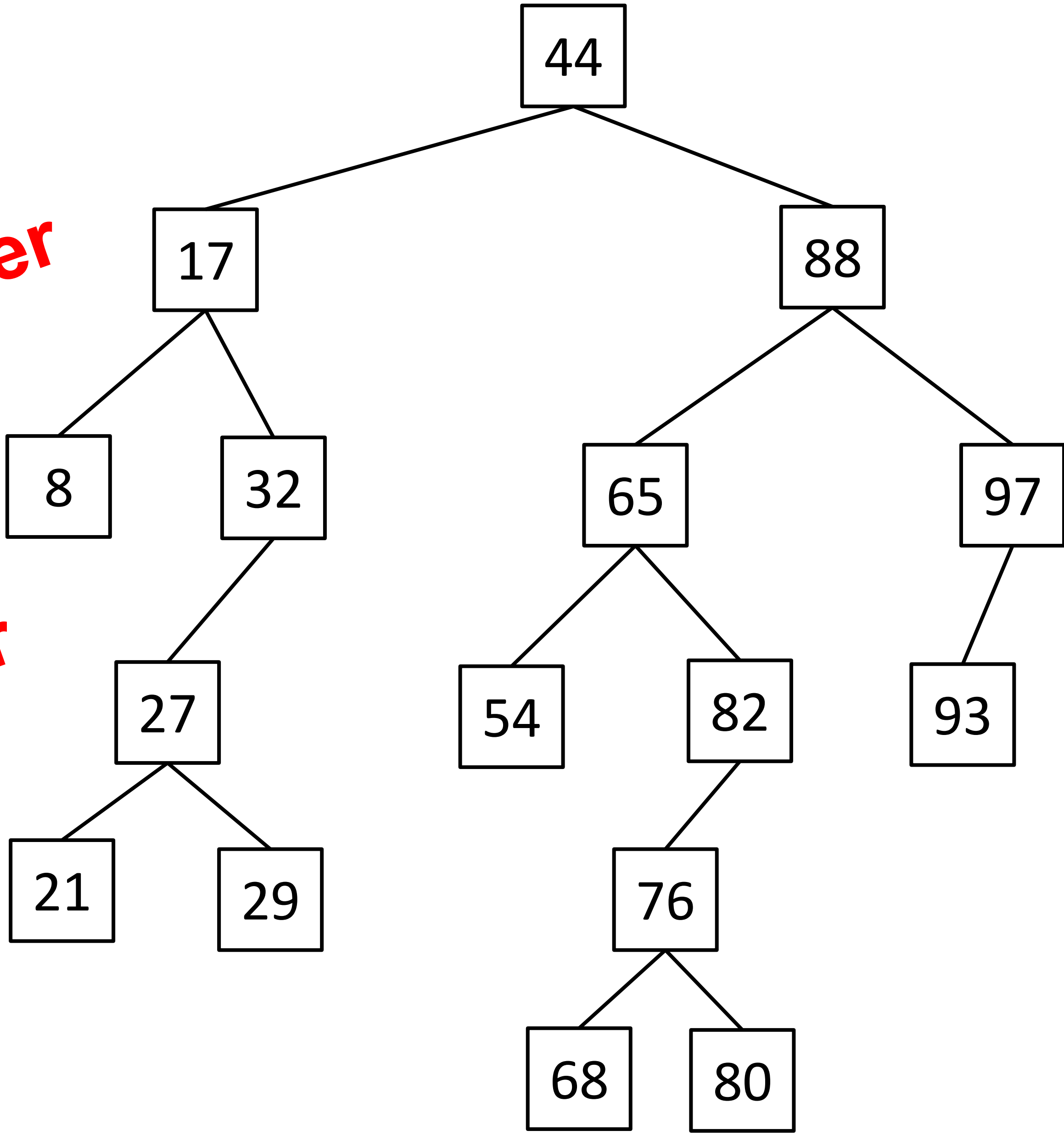


Binary Search Tree- Traversal

```
public void preorder(Node n) {
  if(n != null) {
     System.out.println(n.getValue());
     preorder(n.getLeft());
     preorder(n.getRight());
  }
}
public void inorder(Node n) {
  if(n != null) {
     inorder(n.getLeft());
     System.out.println(n.getValue());
     inorder(n.getRight());
  }
}
```

Preorder

Inorder



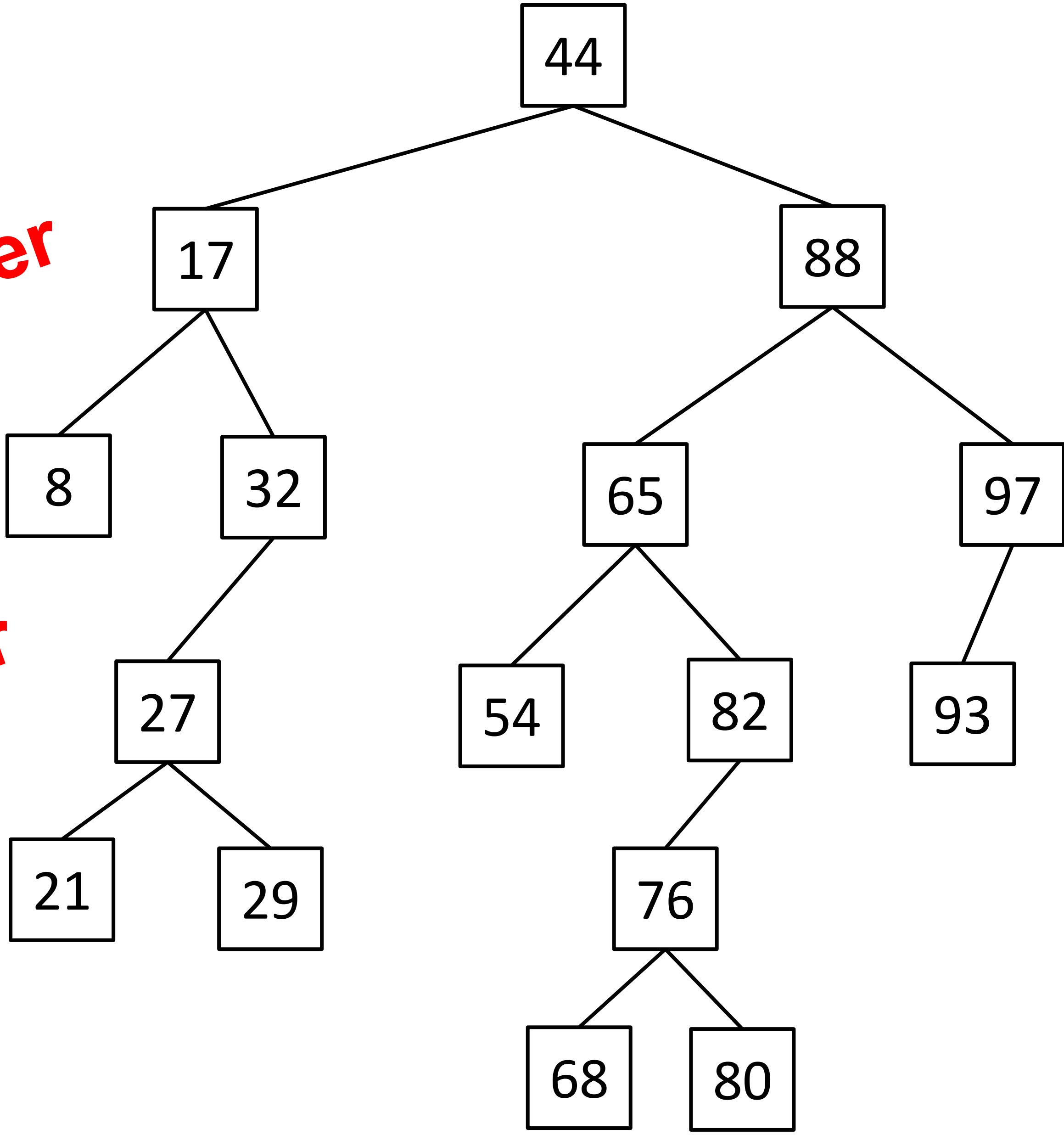
Binary Search Tree- Traversal

```
public void preorder(Node n) {
  if(n != null) {
    System.out.println(n.getValue());
    preorder(n.getLeft());
    preorder(n.getRight());
  }
}
public void inorder(Node n) {
  if(n != null) {
    inorder(n.getLeft());
    System.out.println(n.getValue());
    inorder(n.getRight());
  }
}
public void postorder(Node n) {
  if(n != null) {
    postorder(n.getLeft());
    postorder(n.getRight());
    System.out.println(n.getValue());
  }
}
```

Preorder

Inorder

Postorder



Binary Search Tree- Traversal

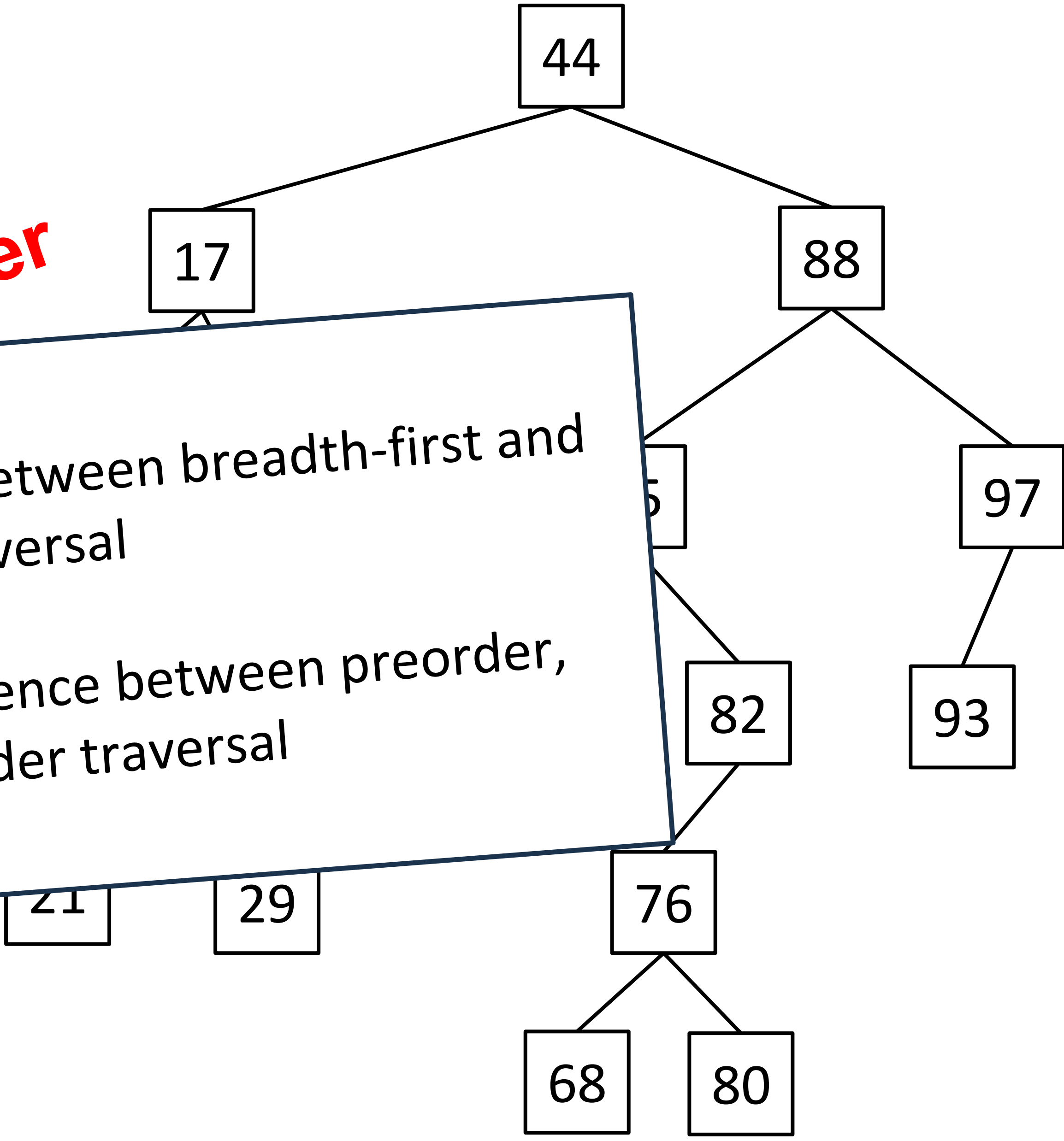
```
public void preorder(Node n) {
    if(n != null) {
        System.out.println(n.getValue());
        preorder(n.getLeft());
        preorder(
    }
}
public void in
    if(n != null
        inorder(n
        System.out
        inorder(n
    }
}
public void pos
    if(n != null) {
        postorder(n.getLeft());
        postorder(n.getRight());
        System.out.println(n.getValue());
    }
}
```

Preorder

You should know the difference between breadth-first and depth-first traversal

You should also know the difference between preorder, inorder, and postorder traversal

Postorder

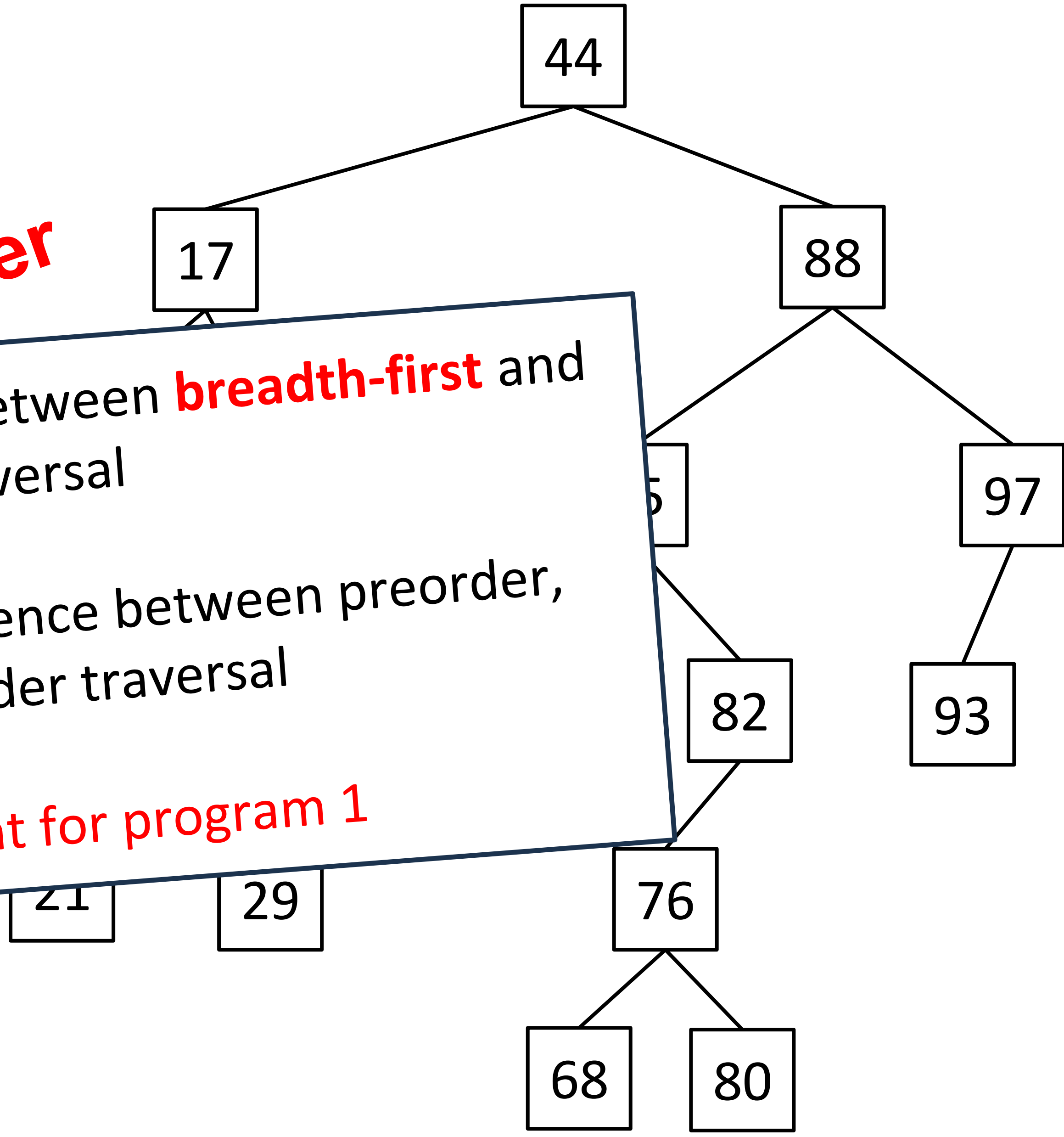


Binary Search Tree- Traversal

```
public void preorder(Node n) {
  if(n != null) {
     System.out.println(n.getValue());
     preorder(n.getLeft());
     preorder(n.getRi
  }
}
public void in
  if(n != null
     inorder(n
     System.out
     inorder(n.
  }
}
public void pos
  if(n != null) {
     postorder(n.getLeft());
     postorder(n.getRight());
     System.out.println(n.getValue());
  }
}
```

Preorder

Postorder



You should know the difference between **breadth-first** and depth-first traversal

You should also know the difference between preorder, **inorder**, and postorder traversal

These will be important for program 1