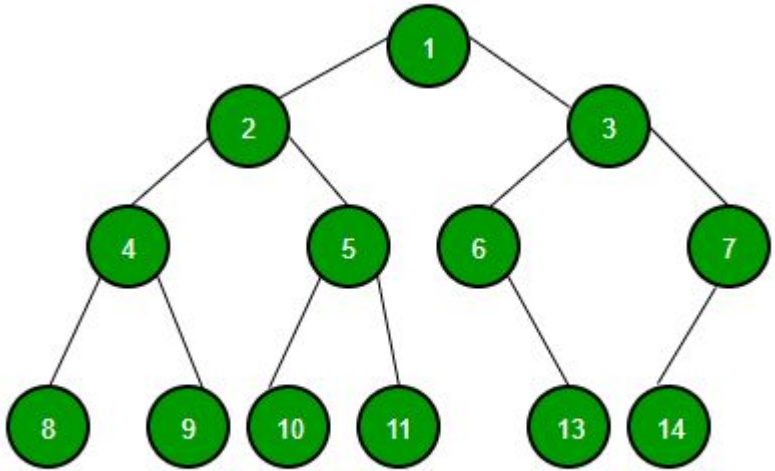
Binary Trees - Searching

Mr. Poole

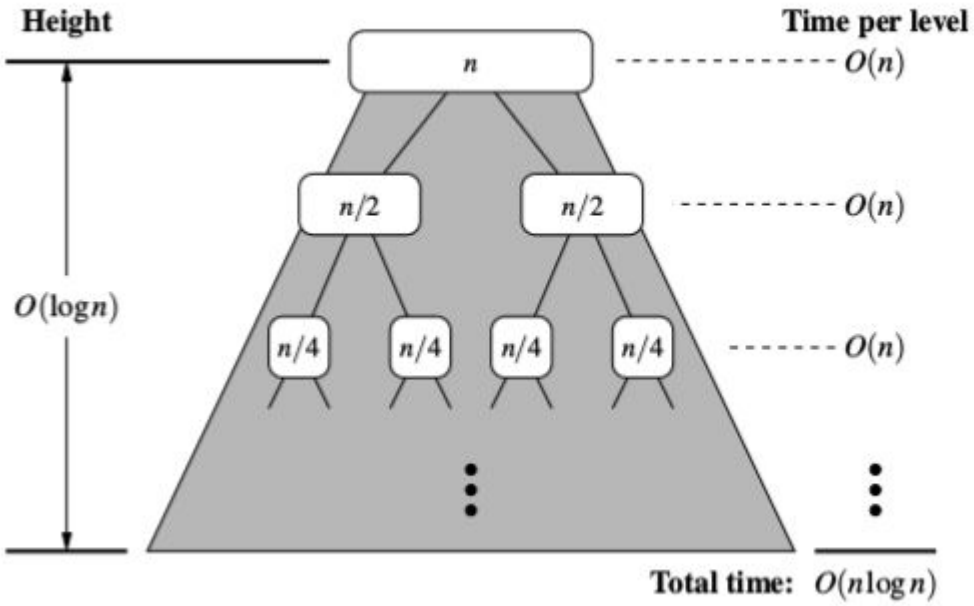
What is a binary tree

Easy example

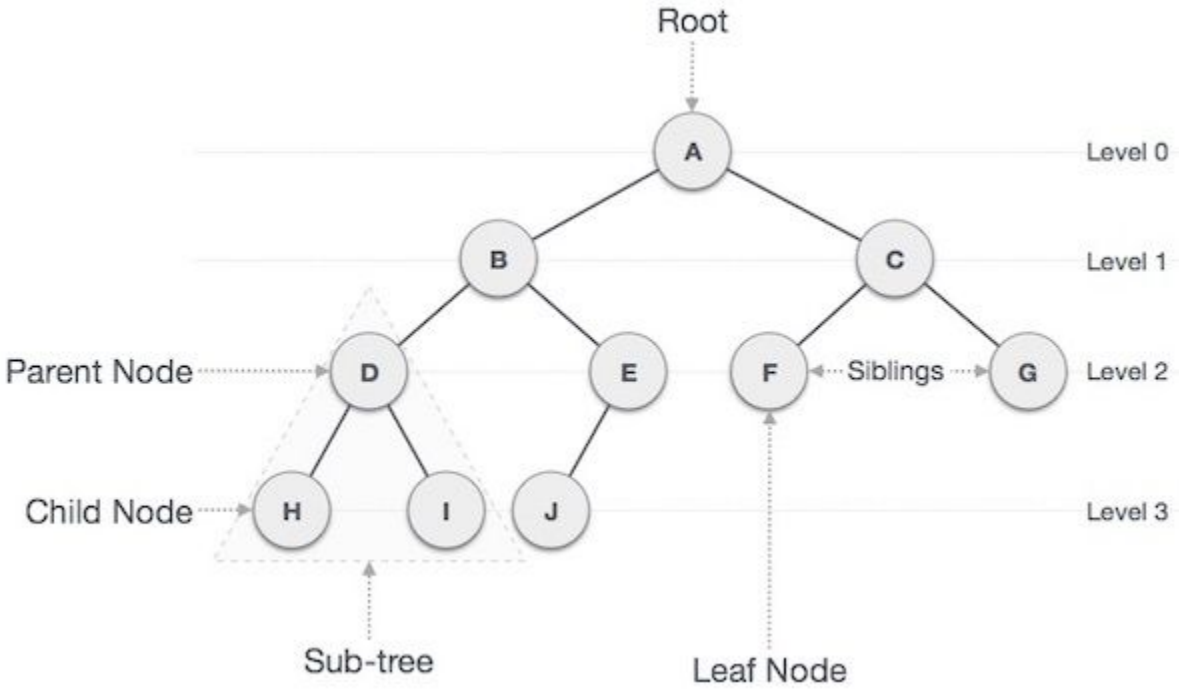
- A phone book
- Family Tree



Runtime of Trees

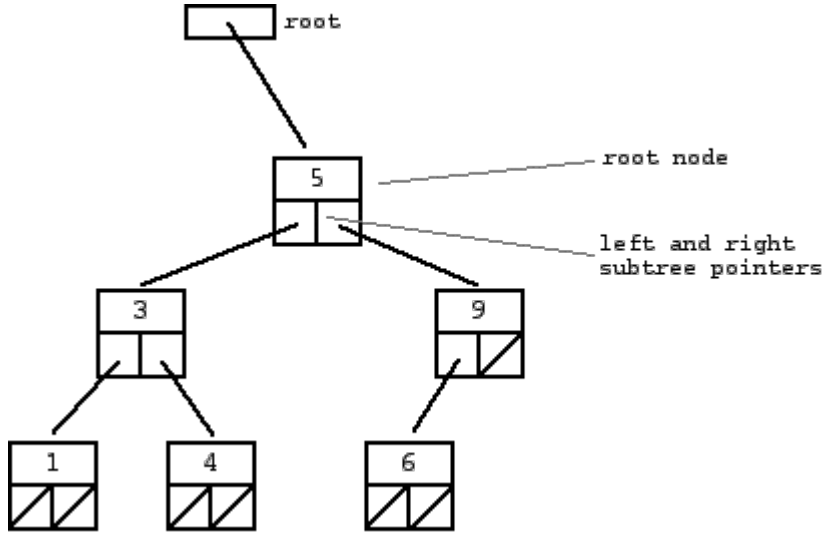


What makes up a Binary Tree?



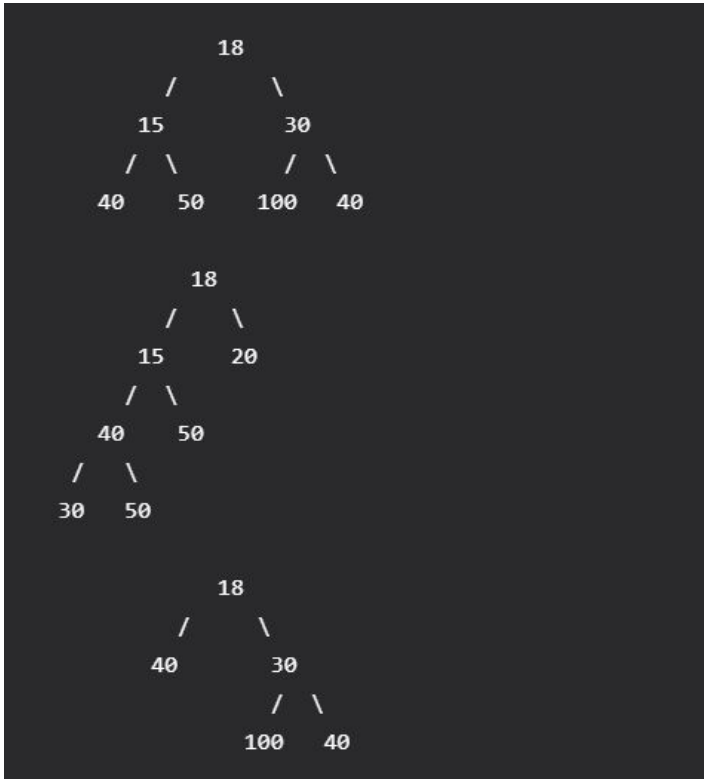
Node Class

```
9   // we will need a node to create binary tree
10  class Node {
11          // it has data
12          int data;
13          // left node
14          Node left;
15          // right node
16          Node right;
17
18          // and a constructor
19          public Node(int data) {
20                  this.data = data;
21          }
22  }
```



Example of Binary Tree - Full Binary Tree

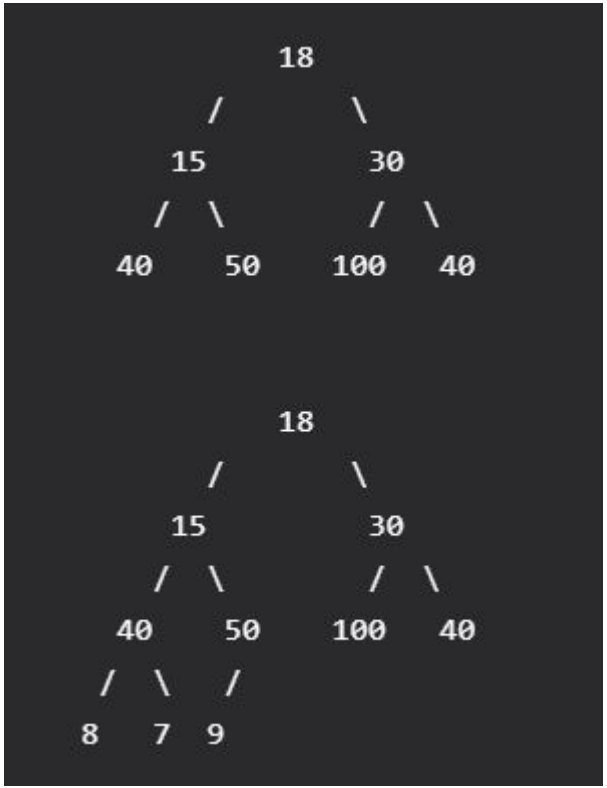
Every node has
0 or 2 nodes as children.



Example of Binary Tree - Complete Binary Tree

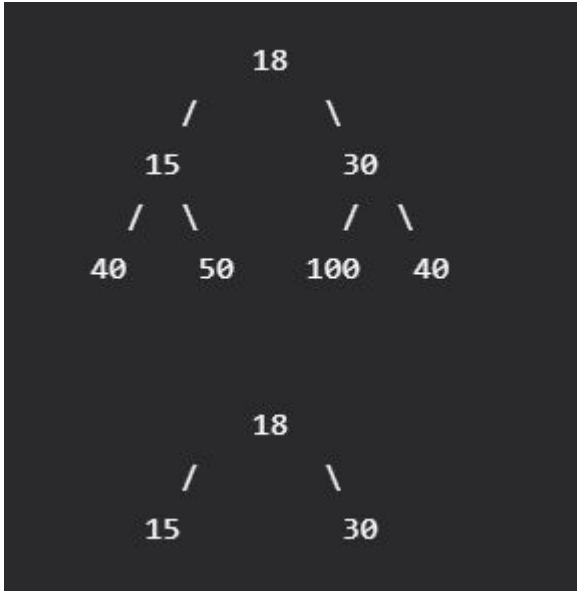
All levels are filled of nodes with the exception of the last level.

The last level must have all nodes to the left most positions.



Example of Binary Tree - Perfect Binary Tree

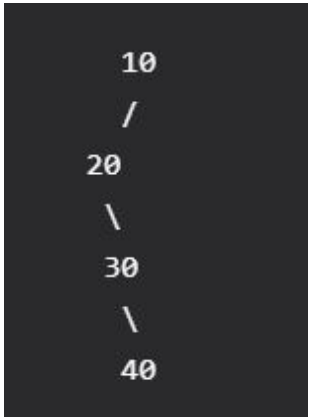
All nodes have two children and all leaf nodes are at the same level.



Example of Binary Tree - Degenerate Binary Tree

Every internal nodes has one child.

Performance is similar to a LinkedList.



Now let’s Search!

Two Types of Search Methods

Breadth First Search vs **Depth** First Search

Breadth starts at the highest nodes and finishes highest first before going deeper.

Depth goes down each branch of the tree until it’s reached the bottom most node, then traverses back to do another branch.

Breadth First Search

Search through the breadth of the highest elements before working your way down.

```
A, B, C, D, E, F
```

Depth First Search

Search through the full DEPTH of the tree till you reach the bottom. Then go back to the top and start again.


```
A, B, D, C, E, F
```

Two Types of Search Methods

Breadth First Search

1. Uses Queue
2. Time Complexity - O(V + E)
3. Siblings visited before children
4. Suitable for searching vertices closer to source

Depth First Search

1. Uses Stack
2. Time Complexity - O(V + E)
3. Children visited before siblings
4. Suitable for vertices further away from source

Both have strengths and weaknesses that compliment each other. Very equal.

Breadth First Search - Using a queue

When a node is visited, add the children to the queue. Pop that visited node.

The Queue looks like the following:

[**A**] A is visited, push B and C

[**B**,C] B is visited, push D

[**C**,D] C is visited, push E and F

[**D**, E, F]

A, B, C, D, E, F

Depth First Search

Push node to stack, visit left node. If end reached, pop from stack until a right node is visible. Only pop if left and right side have been visited.

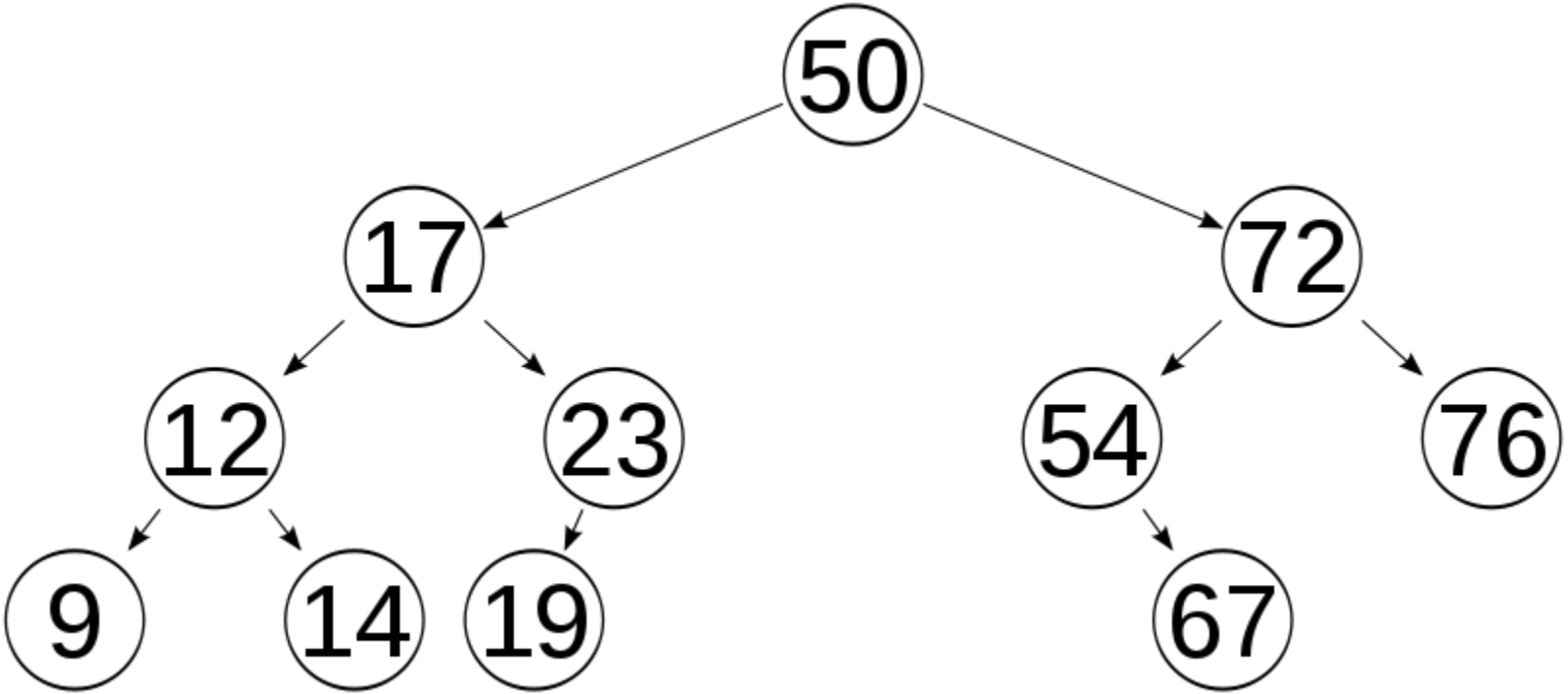
[A, B, D]	Visited left side, pop D & B
[A, C, E]	Visited A, C, E. pop E.
[A, C, F]	Visited F, pop F.
[A]	Visited C then A. Pop C & A

A, B, D, C, E, F

Sorting makes searching easier!

Let’s sort!

Binary Trees are ideally sorted when created



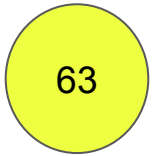
This tree's array may have been: 50, 17, 12, 72, 54, 14, 23, 9, 76, 67, 19

Let’s make the following Sorted Binary Tree:

63, 97, 102, 44, 75, 12, 55, 49, 2, 113, 111, 99, 51, 46

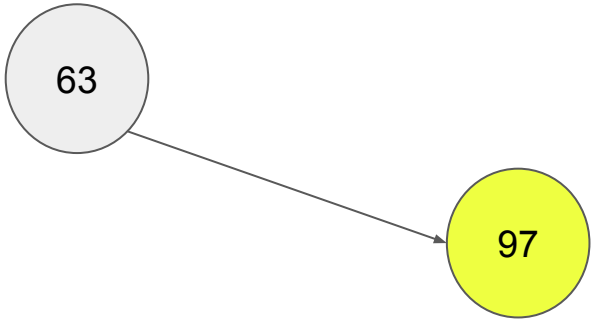
Let’s make the following Sorted Binary Tree:

63, 97, 102, 44, 75, 12, 55, 49, 2, 113, 111, 99, 51, 46



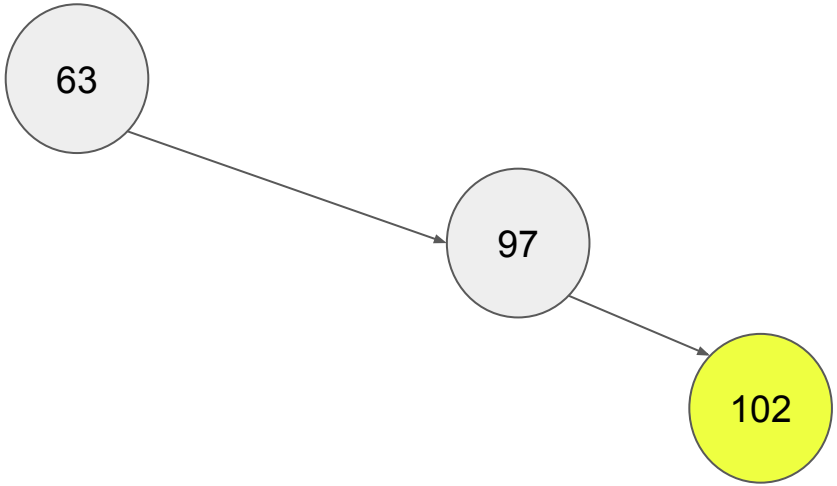
Let’s make the following Sorted Binary Tree:

63, **97**, 102, 44, 75, 12, 55, 49, 2, 113, 111, 99, 51, 46



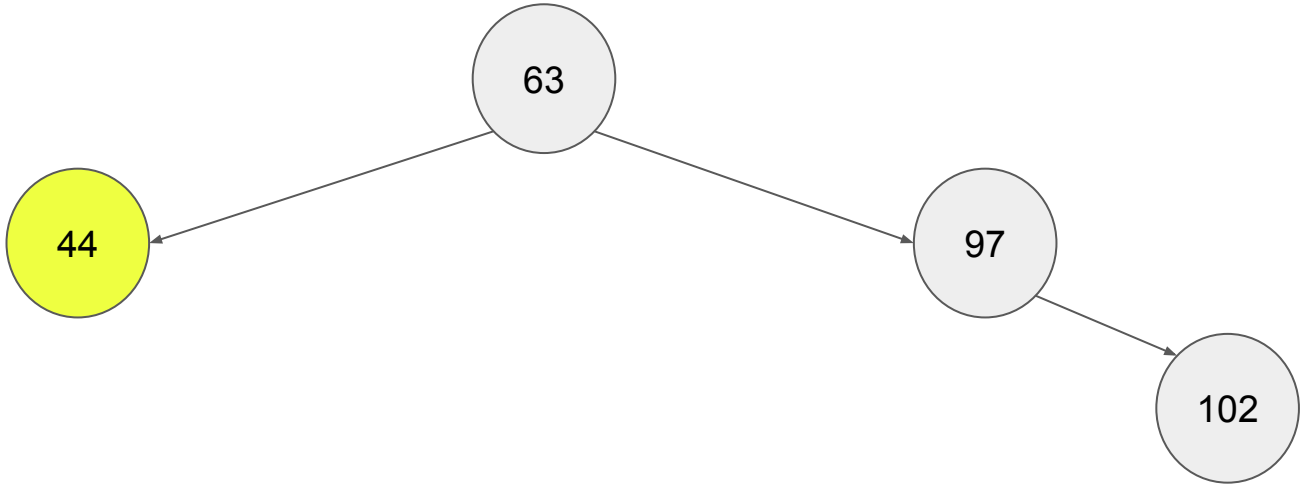
Let’s make the following Sorted Binary Tree:

63, 97, **102**, 44, 75, 12, 55, 49, 2, 113, 111, 99, 51, 46



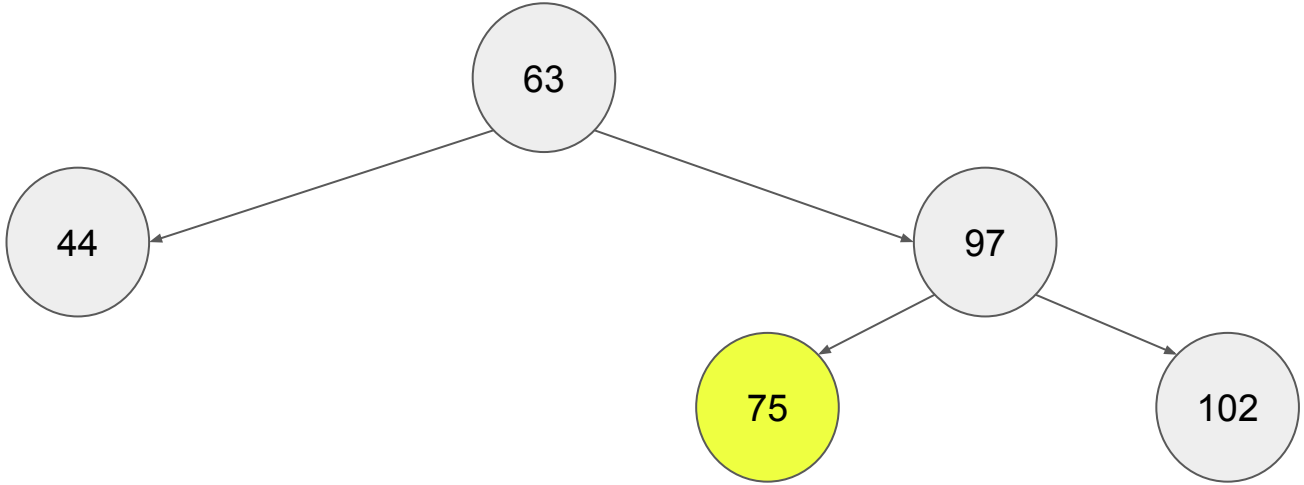
Let’s make the following Sorted Binary Tree:

63, 97, 102, **44**, 75, 12, 55, 49, 2, 113, 111, 99, 51, 46



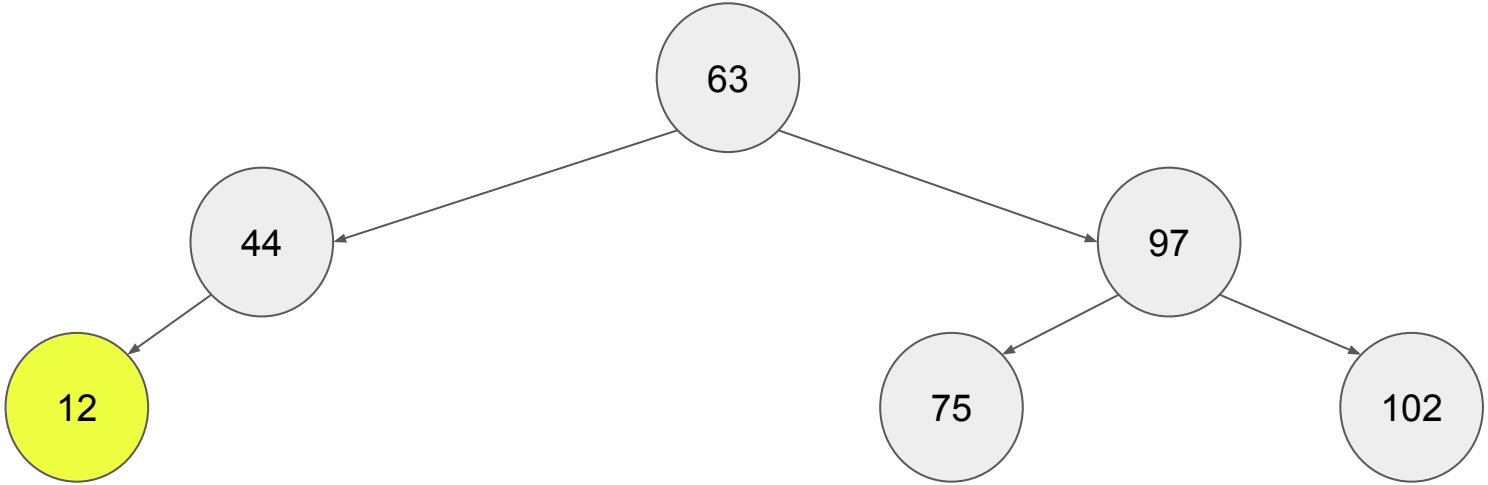
Let’s make the following Sorted Binary Tree:

63, 97, 102, 44, **75**, 12, 55, 49, 2, 113, 111, 99, 51, 46



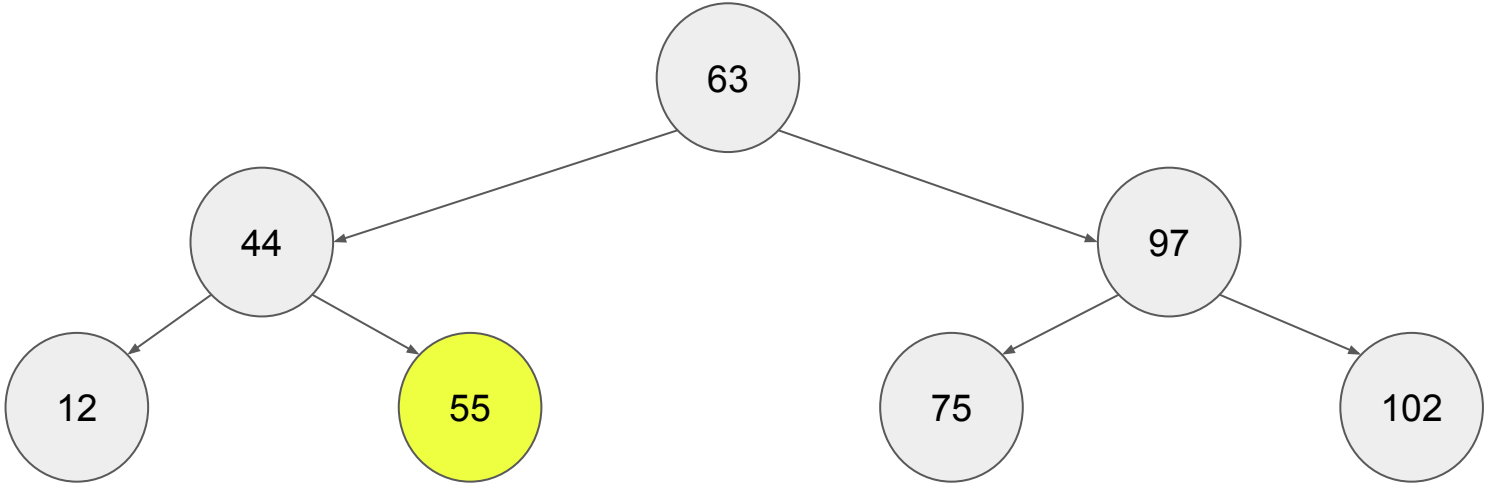
Let’s make the following Sorted Binary Tree:

63, 97, 102, 44, 75, **12**, 55, 49, 2, 113, 111, 99, 51, 46



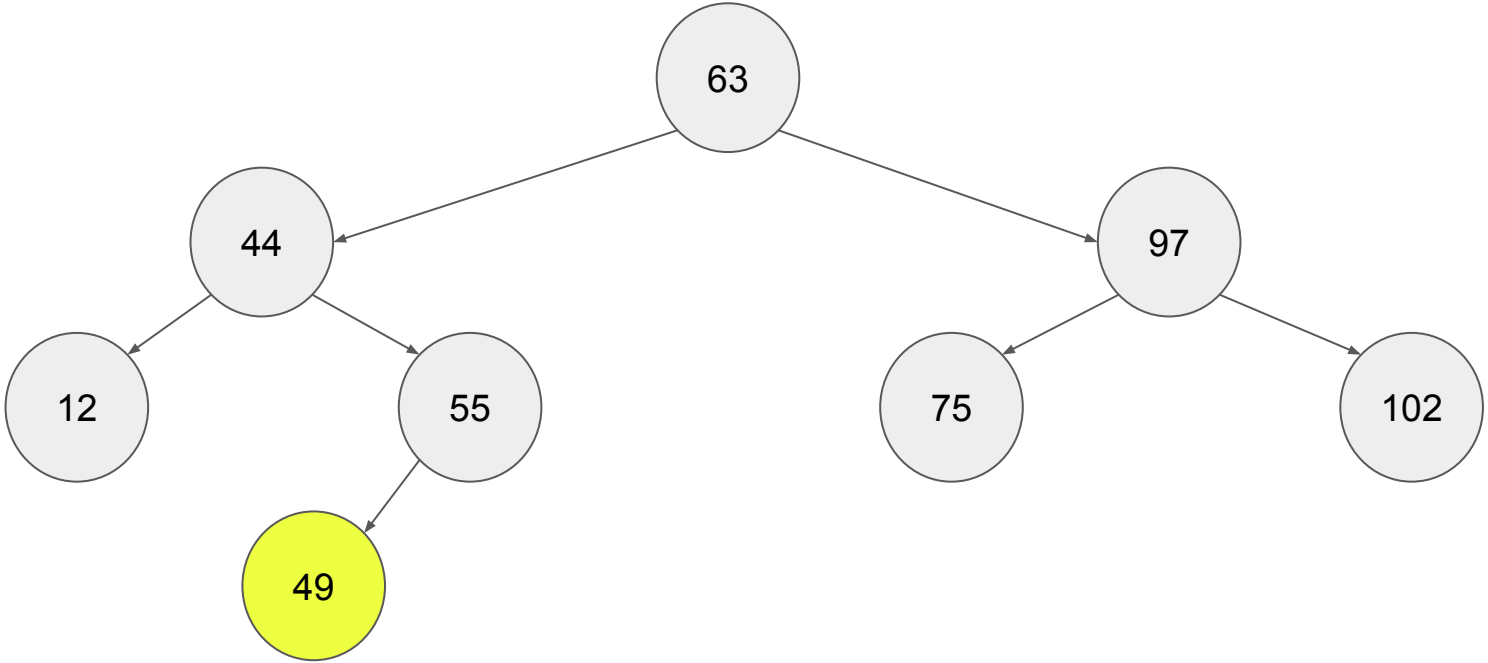
Let’s make the following Sorted Binary Tree:

63, 97, 102, 44, 75, 12, **55**, 49, 2, 113, 111, 99, 51, 46



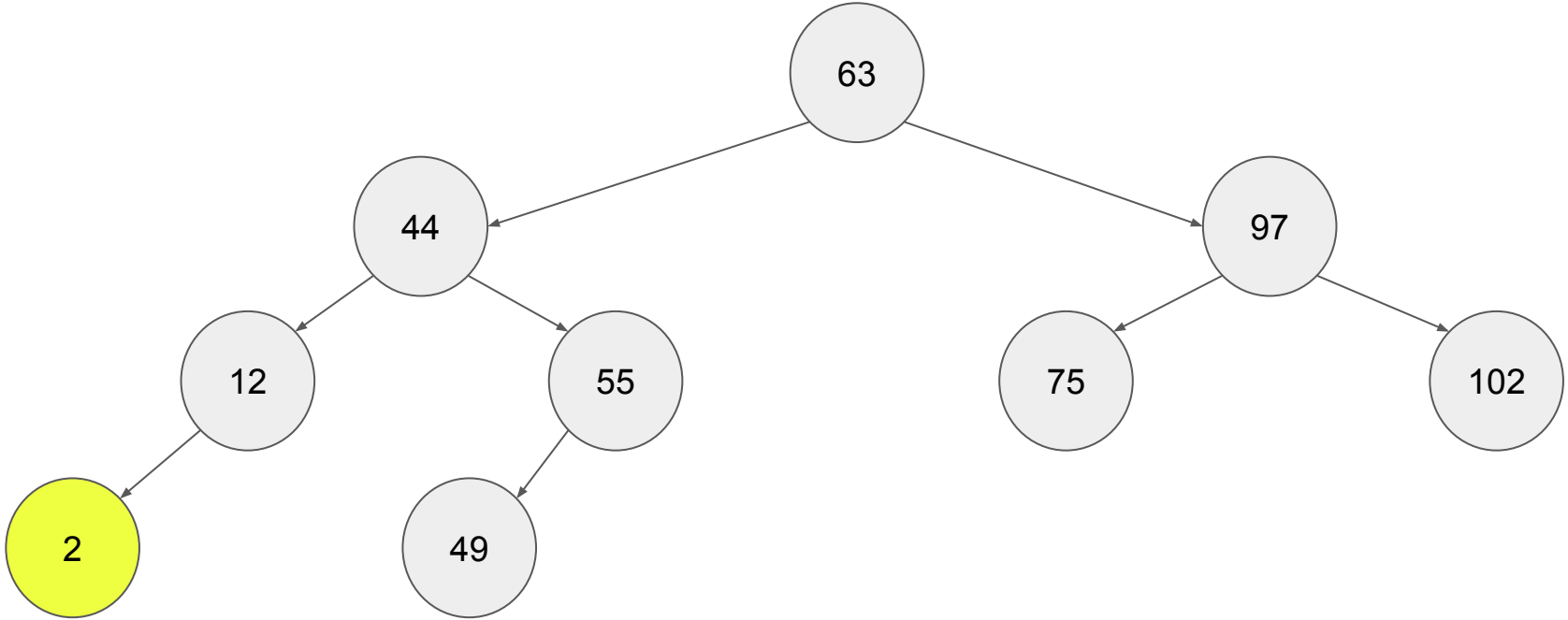
Let’s make the following Sorted Binary Tree:

63, 97, 102, 44, 75, 12, 55, **49**, 2, 113, 111, 99, 51, 46



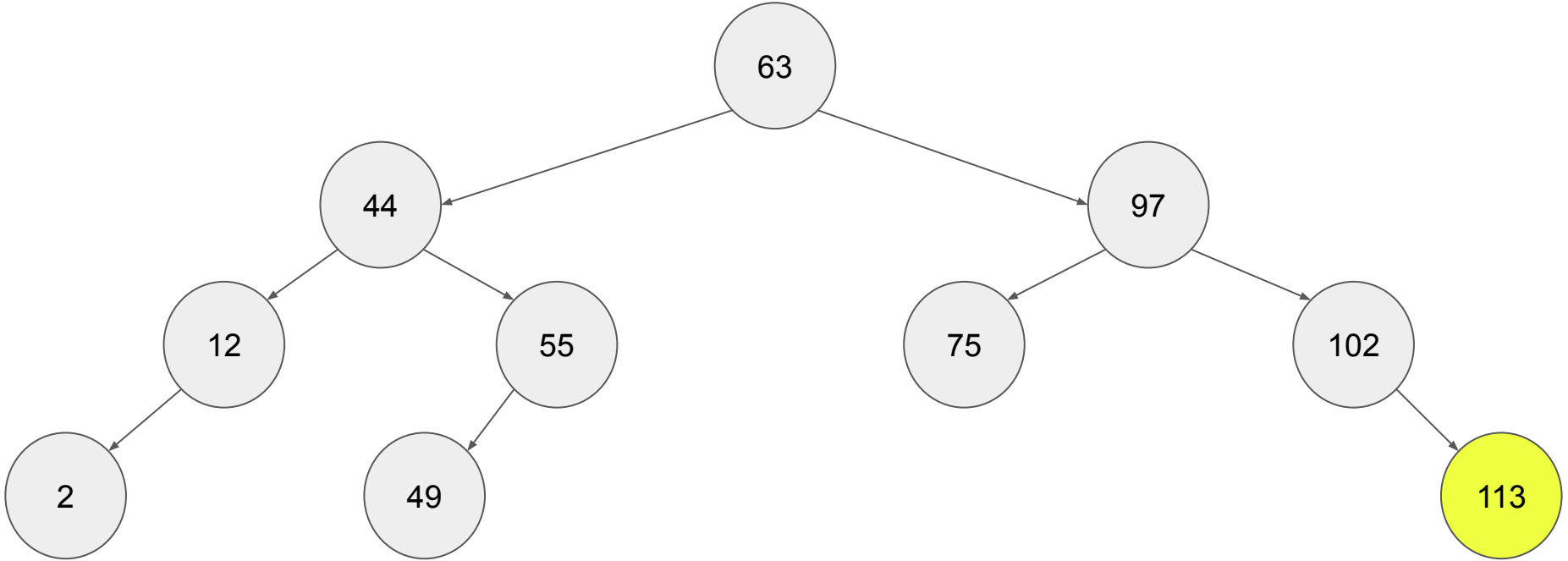
Let’s make the following Sorted Binary Tree:

63, 97, 102, 44, 75, 12, 55, 49, **2**, 113, 111, 99, 51, 46



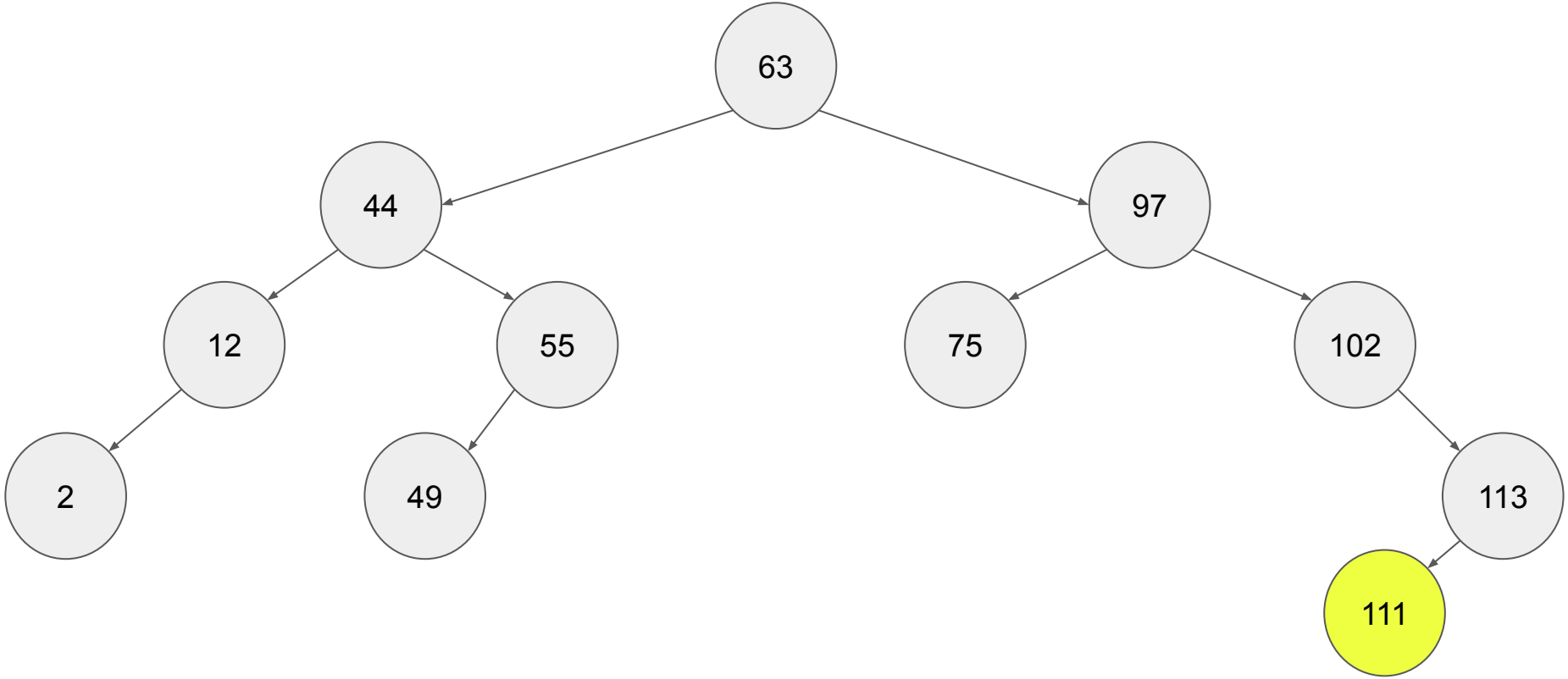
Let’s make the following Sorted Binary Tree:

63, 97, 102, 44, 75, 12, 55, 49, 2, **113**, 111, 99, 51, 46



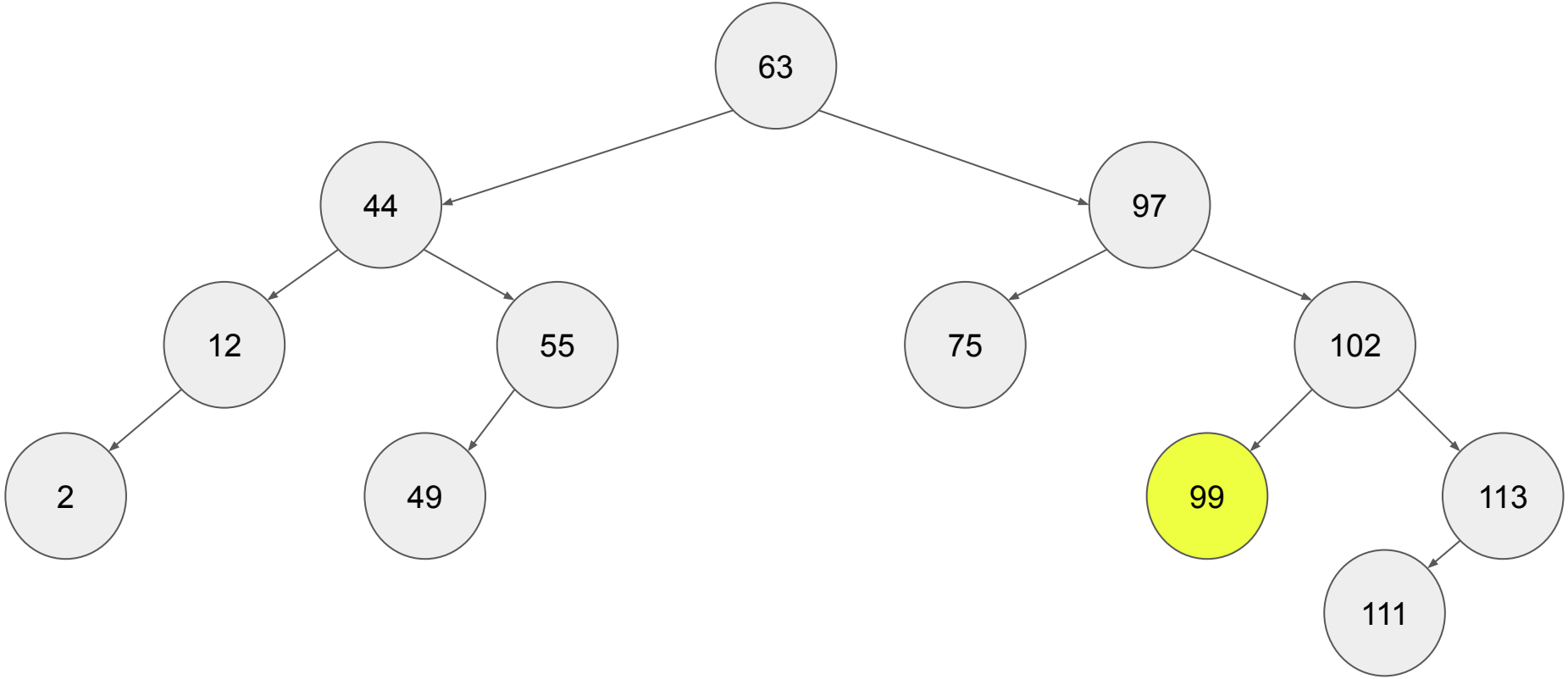
Let’s make the following Sorted Binary Tree:

63, 97, 102, 44, 75, 12, 55, 49, 2, 113, **111**, 99, 51, 46



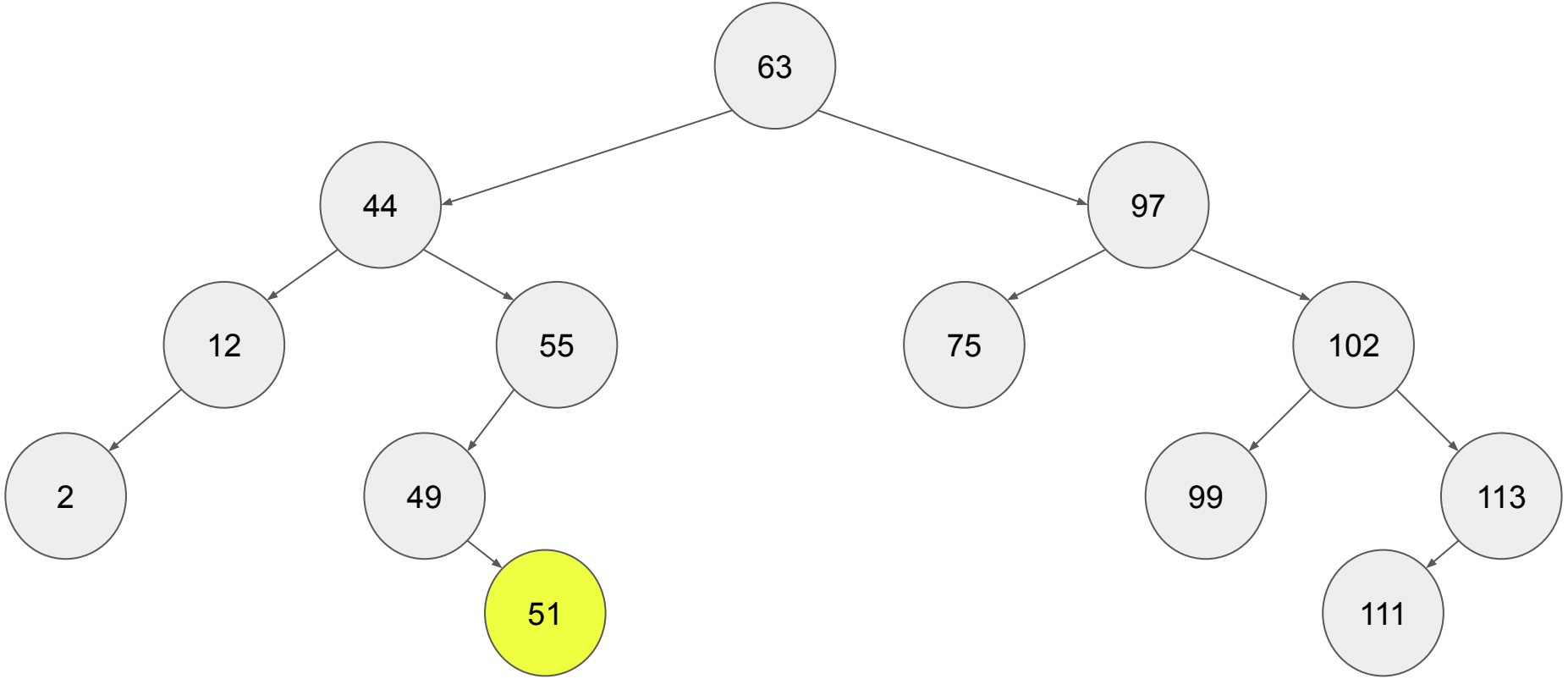
Let’s make the following Sorted Binary Tree:

63, 97, 102, 44, 75, 12, 55, 49, 2, 113, 111, **99**, 51, 46



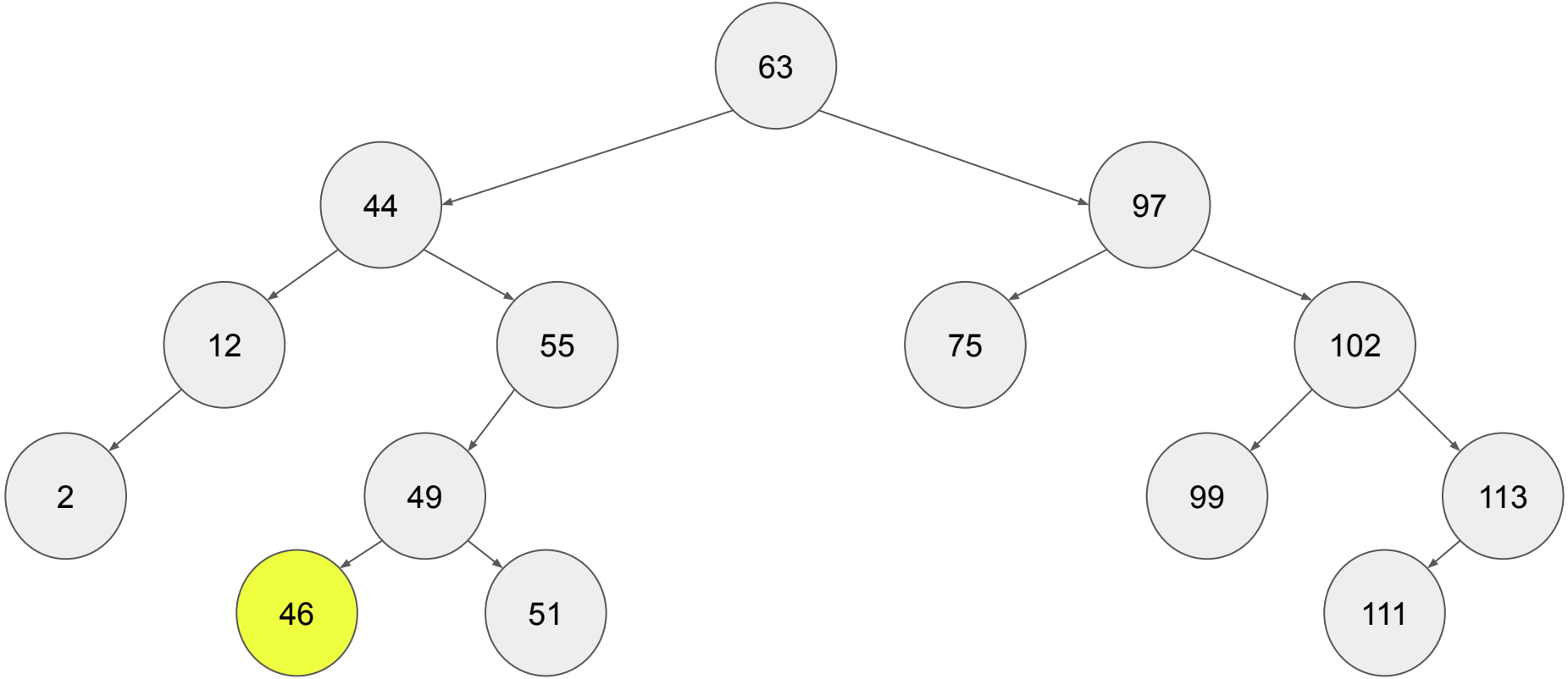
Let’s make the following Sorted Binary Tree:

63, 97, 102, 44, 75, 12, 55, 49, 2, 113, 111, 99, **51**, 46



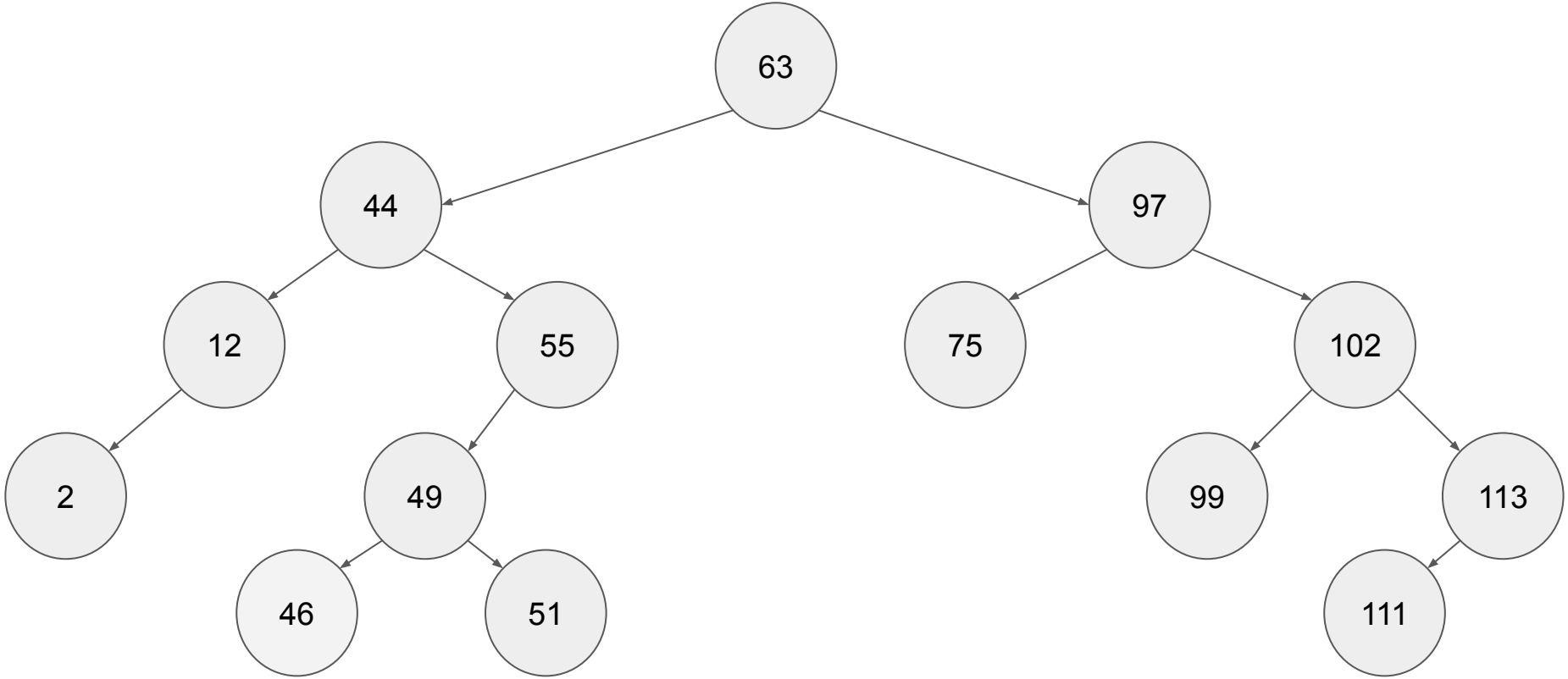
Let’s make the following Sorted Binary Tree:

63, 97, 102, 44, 75, 12, 55, 49, 2, 113, 111, 99, 51, **46**



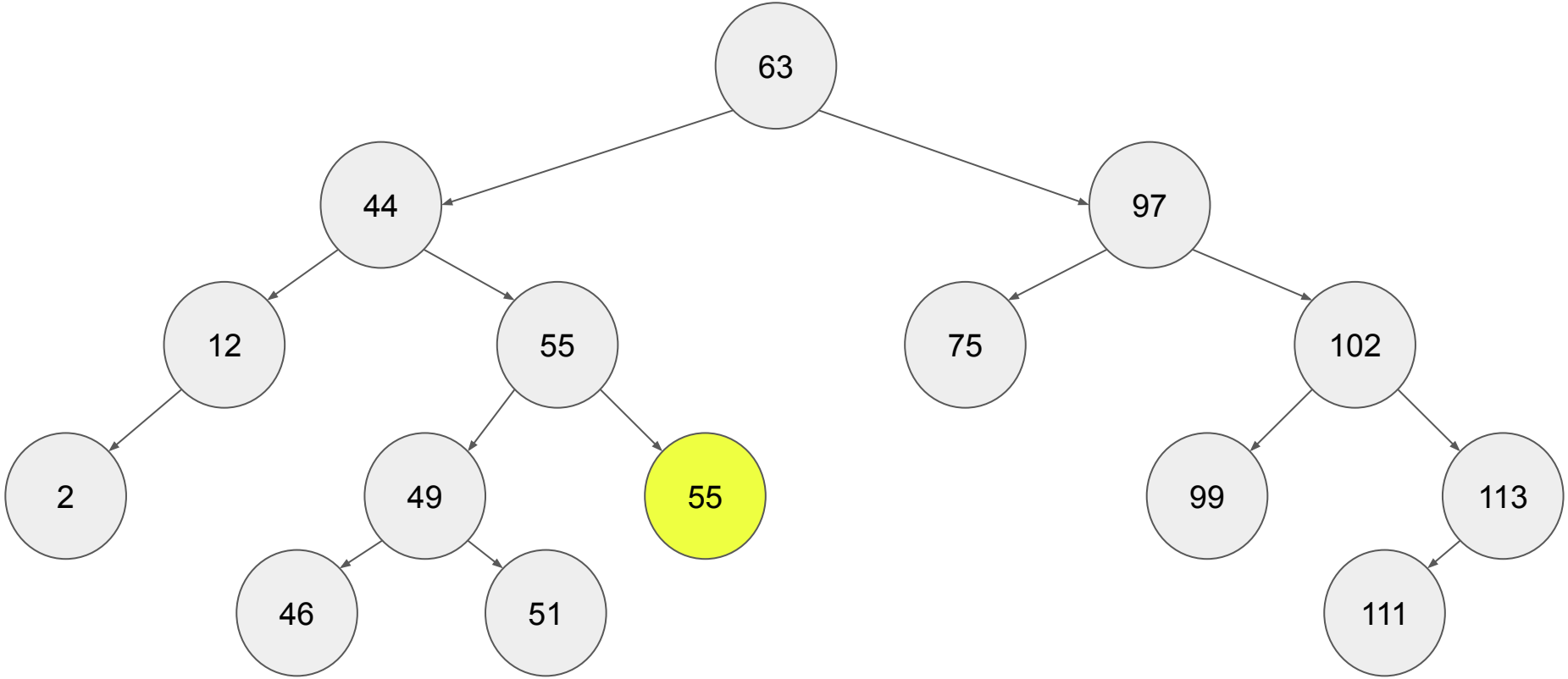
Completed Tree!!!

63, 97, 102, 44, 75, 12, 55, 49, 2, 113, 111, 99, 51, 46

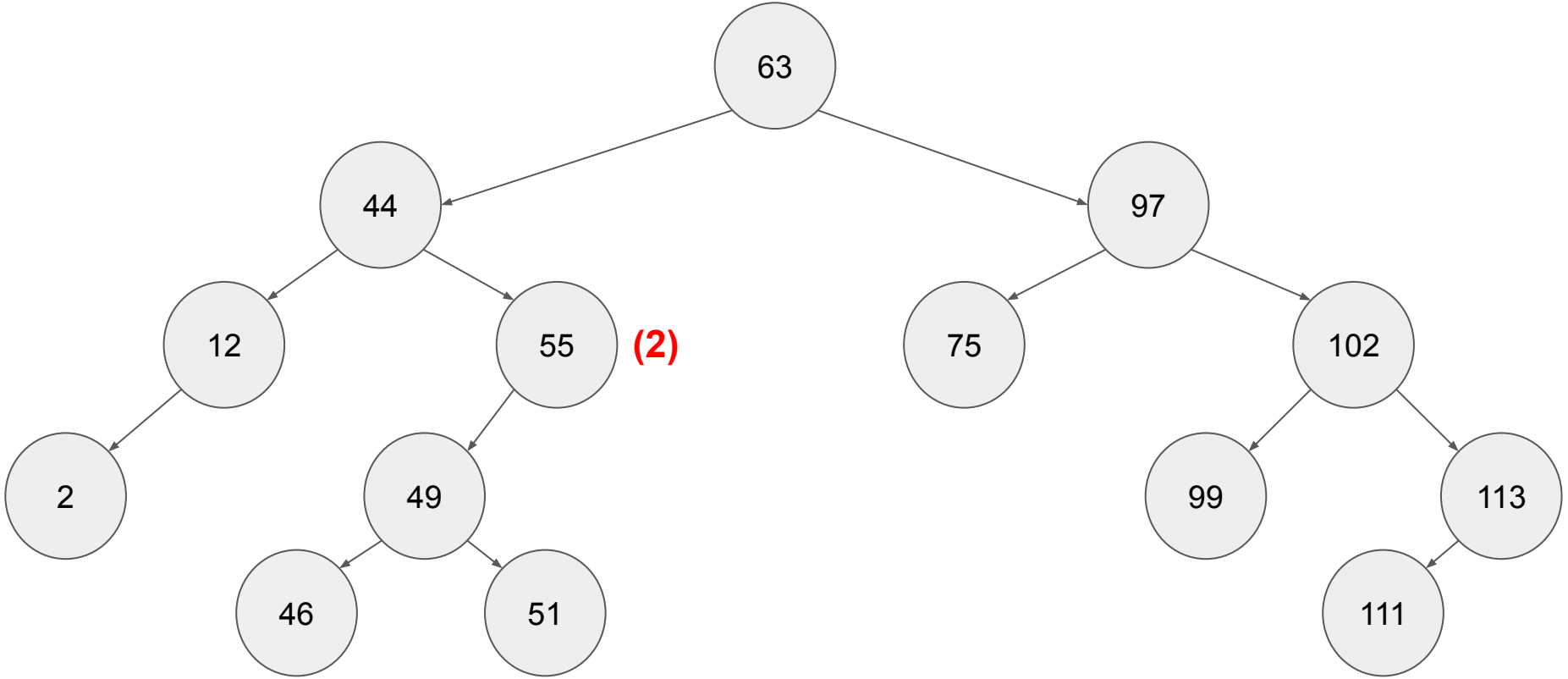


Duplicates can be handled: First by always using one side.

Ex: Always place duplicates on the right



Duplicates can be handled: Second, by weight
Add weights to nodes to show how many of that number



That’s how to make a tree. Woo :)

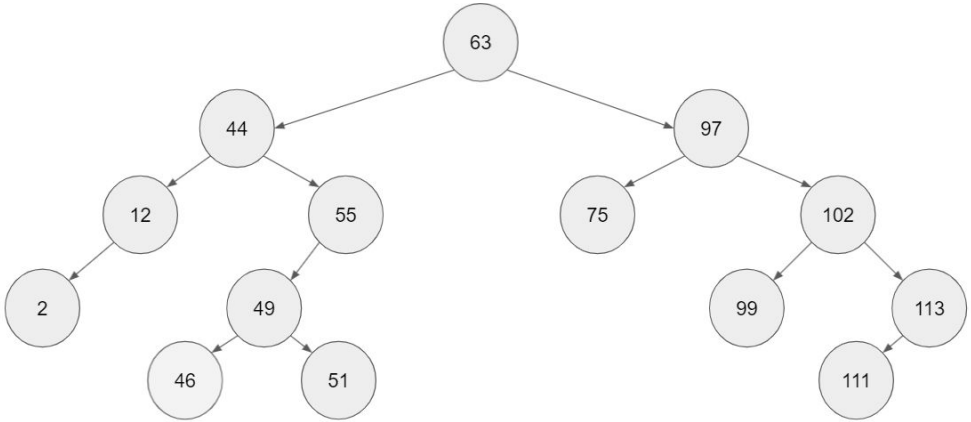
Practice: Try the following tree

55, 33, 95, 17, 25, 66, 57, 99, 97, 100, 3

Trees love Recursion!

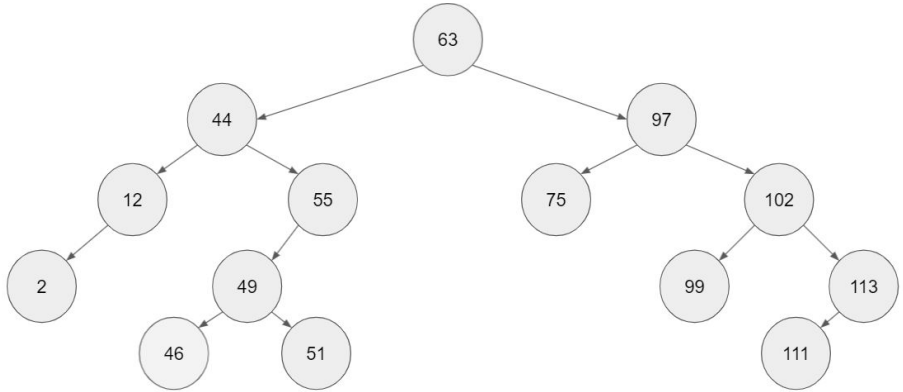
Tree sorting is Recursive!

```
/* A recursive function to
insert a new key in BST */
Node insertRec(Node root, int key)
{

    /* If the tree is empty,
    return a new node */
    if (root == null)
    {
        root = new Node(key);
        return root;
    }

    /* Otherwise, recur
    down the tree */
    if (key < root.key)
        root.left = insertRec(root.left, key);
    else if (key > root.key)
        root.right = insertRec(root.right, key);

    /* return the root */
    return root;
}
```



Tree printing is Recursive!

```
// A function to do
// inorder traversal of BST
void inorderRec(Node root)
{
    if (root != null)
    {
        inorderRec(root.left);
        System.out.print(root.key + " ");
        inorderRec(root.right);
    }
}
```



Next up: GRAPHS!

Graph theory is super cool