

Week 6 Day 5

- Binary Search Tree
- Wrap Up

Binary Search Tree

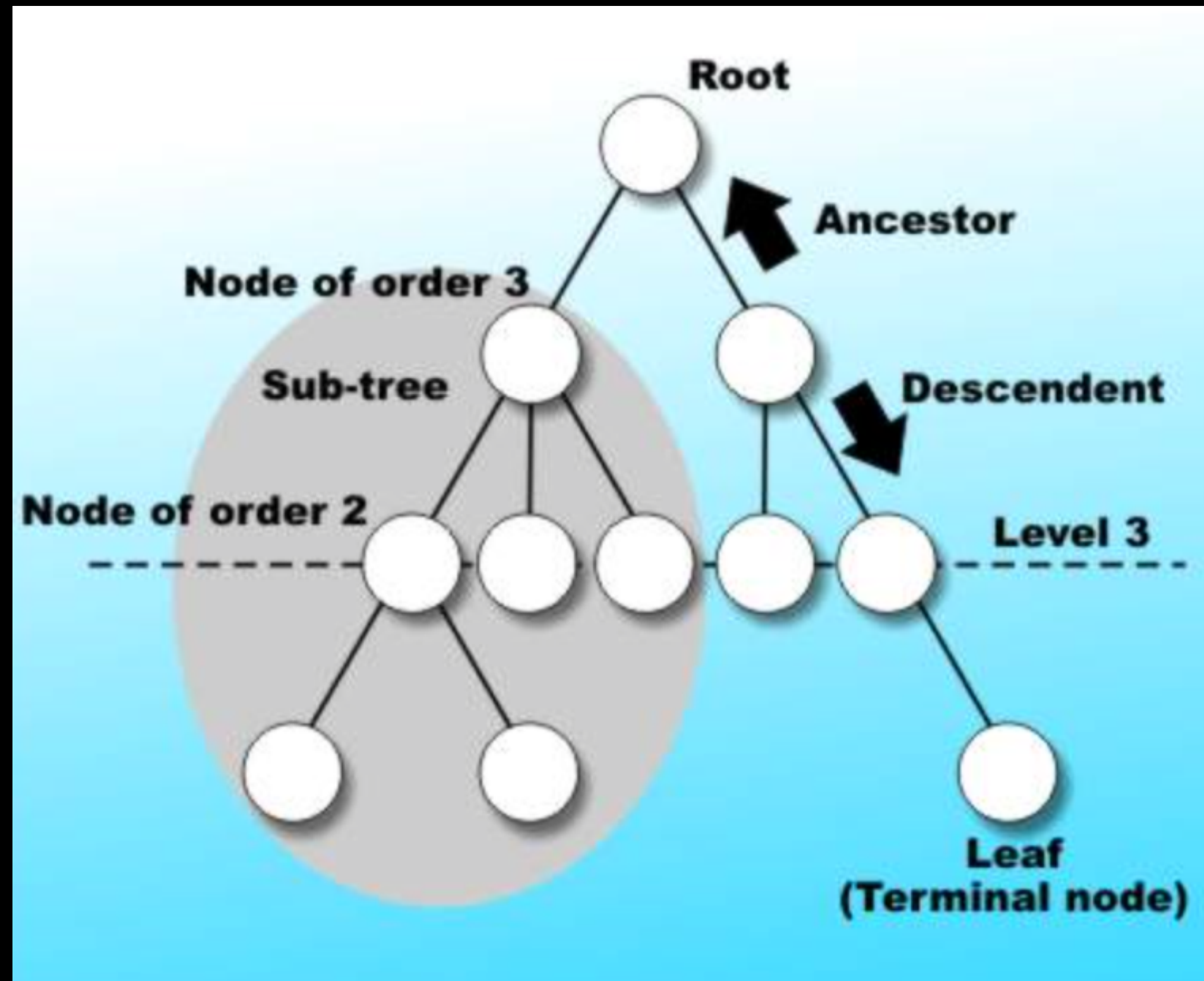
- “A binary search tree is a binary tree where each node has a comparable key and satisfies the restriction that the key in any node is larger than all keys in all nodes in that node’s left subtree, and smaller than the keys in all nodes in that nodes right subtree”

-princeton.edu

Lets dissect that definition a bit

Binary Search Tree

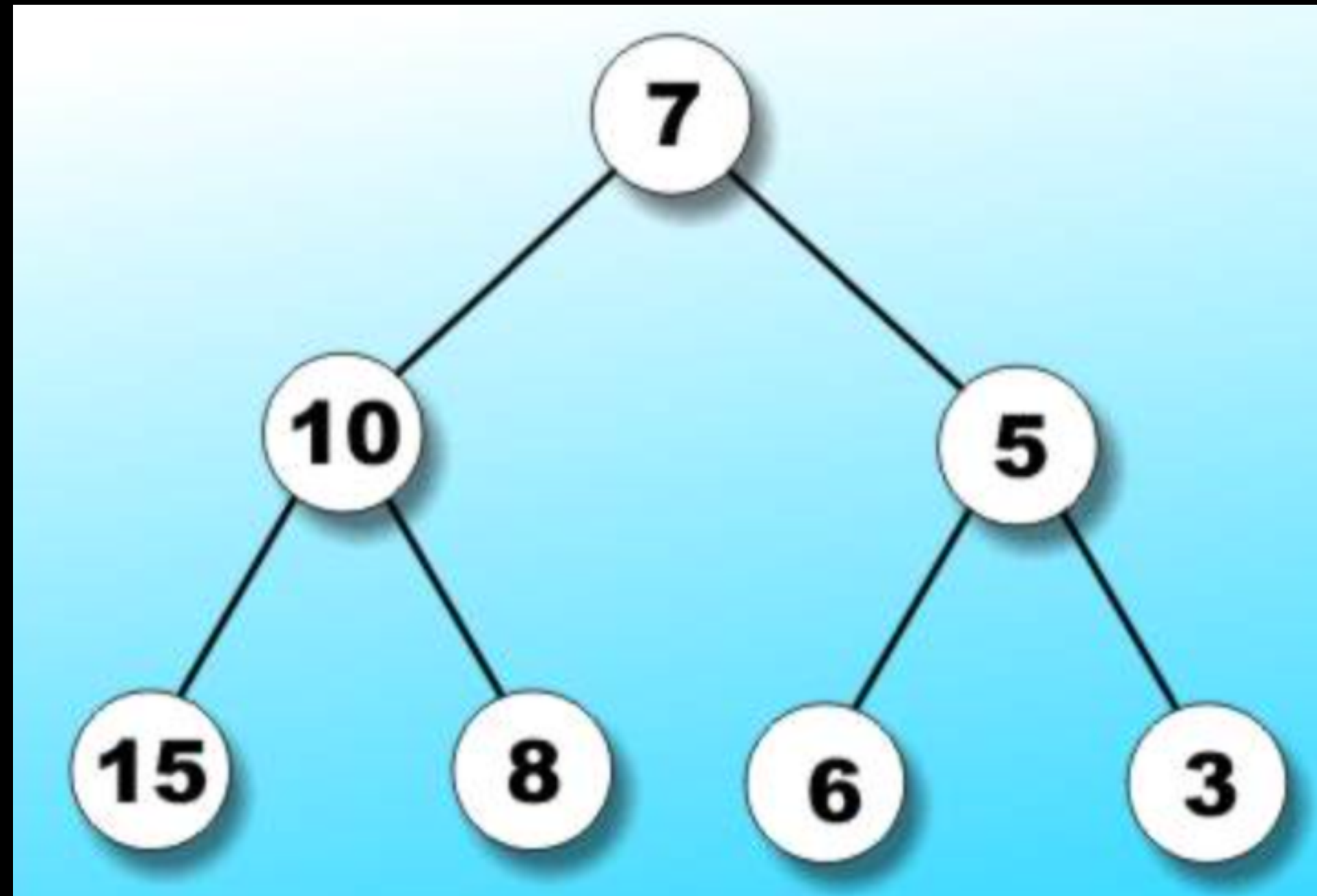
- A binary search tree is a binary **tree**



- A tree is a data structure consisting of nodes organized as a hierarchy

Binary Search Tree

- A binary search tree is a **binary tree**



- A binary tree is a tree in which each node has at most two descendants
- Each node can have zero, one, or two descendants, but no more than two

Tree terminology

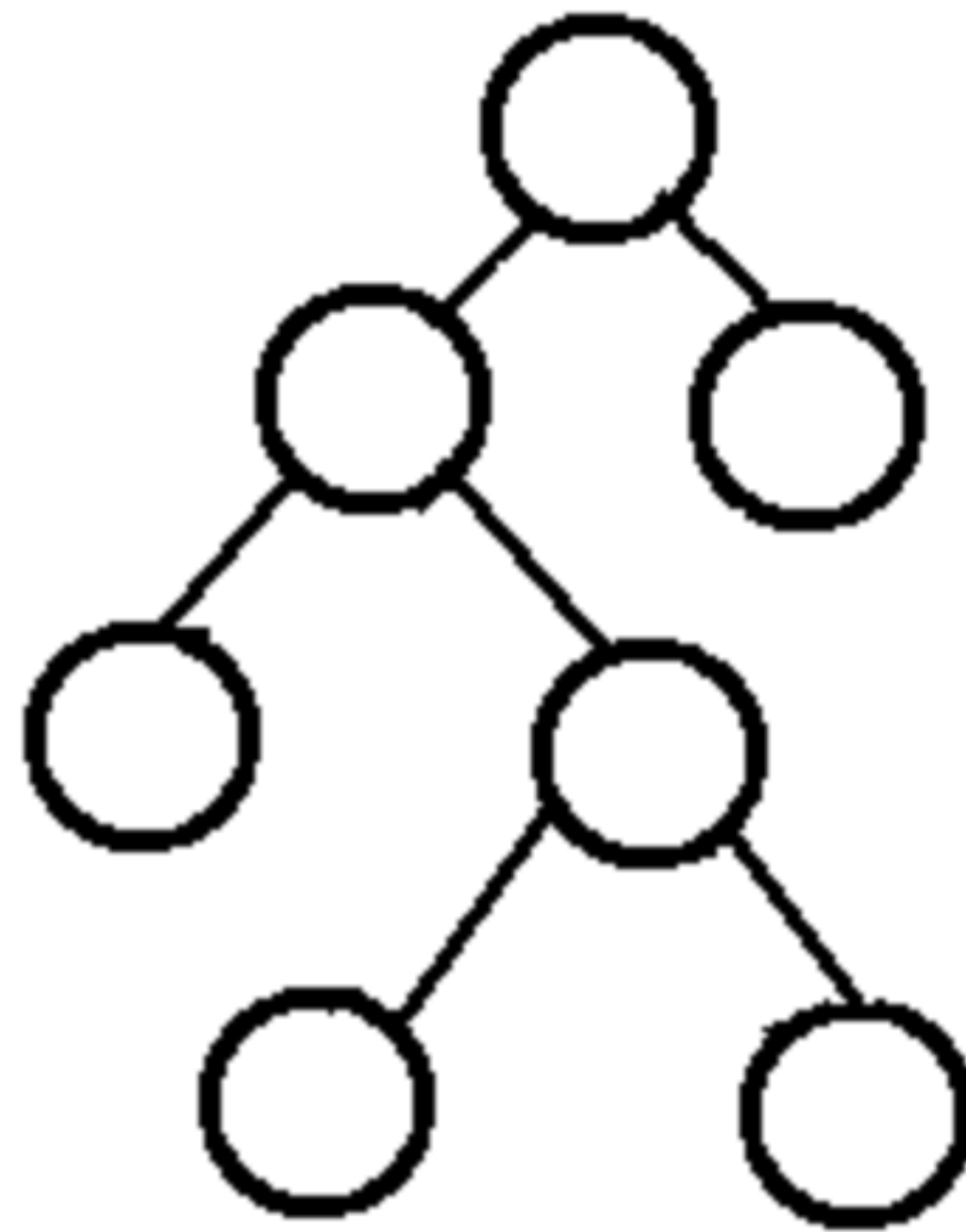
- In a tree data structure, and its different types (like binary), the root node is the topmost node
- In a binary tree, each node contains a left and right reference and a data element
- Every node is connected by a directed edge from exactly one other node. That node is considered the parent.
- The left and right references a node can have are considered that node's children
- Nodes without any children are considered leaves (or sometimes external nodes)
- Nodes with the same parent are considered siblings

Tree terminology cont.

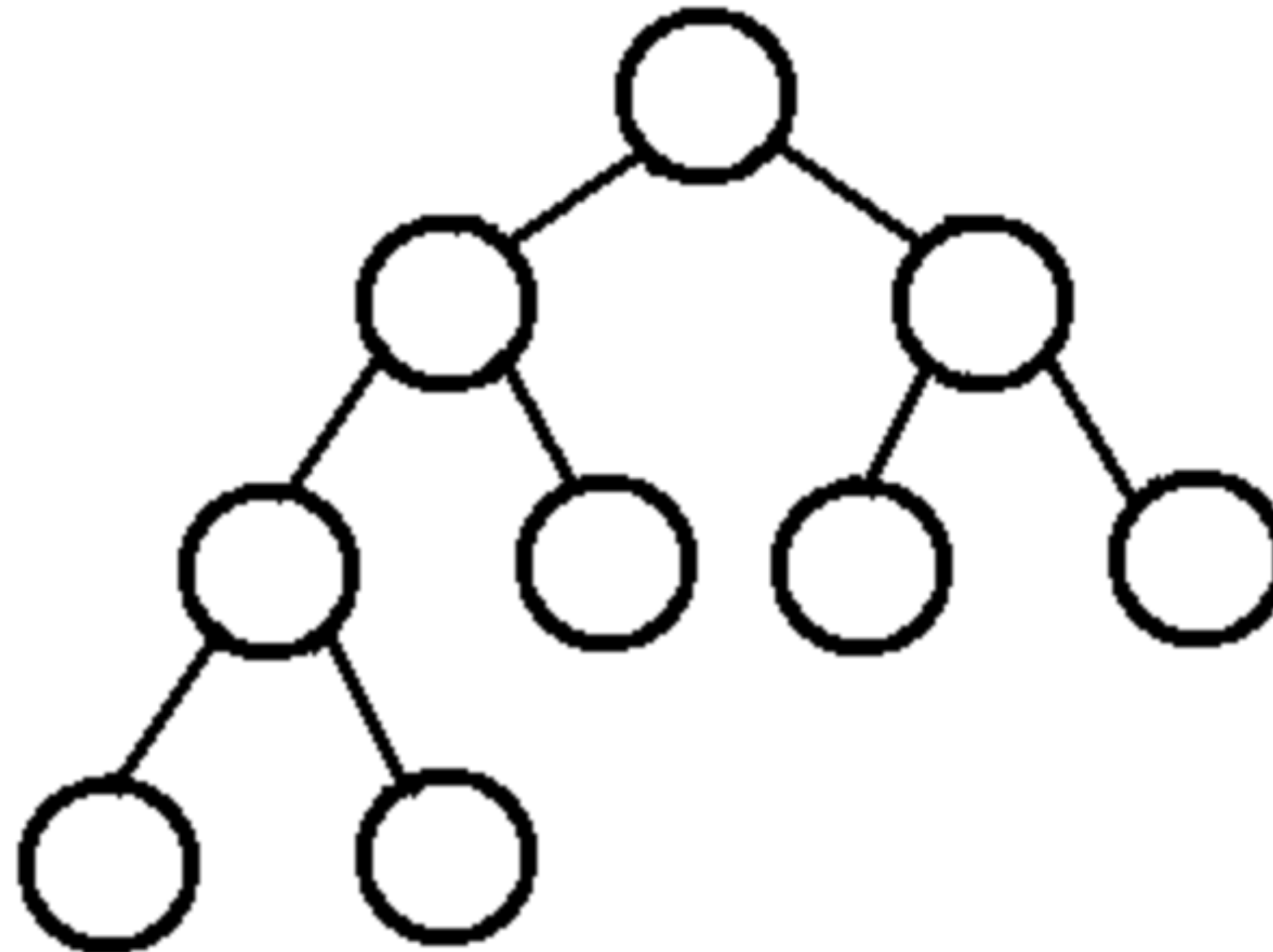
- The depth of a node is the number of edges from the root to the node
- The height of a node is the number of edges from the node to its deepest leaf
- The height of a tree is the height of its root node

Binary Tree types

- A full binary tree is a binary tree in which each node has exactly zero or two children
- A complete binary tree is a binary tree which is completely filled, with the possibly exception of the bottom level, which is filled from left to right



full tree



complete tree

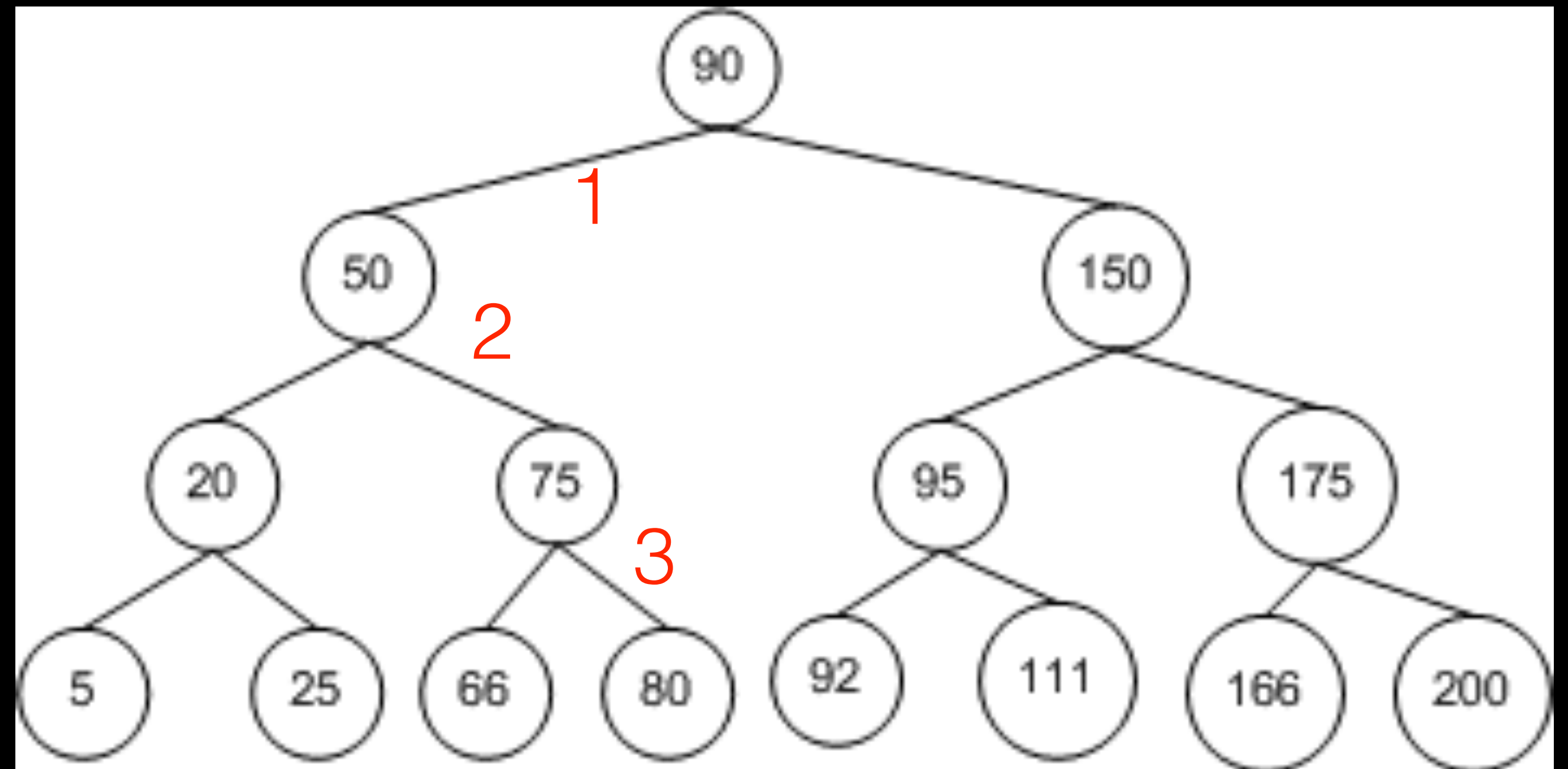
Complete Tree

- A complete tree is a very awesome tree, because it provides the best ratio between number of nodes and height
- The height h of a complete binary tree with N nodes is at most $O(\log n)$
- Thats amazing!

$$n = 15$$

$$\log(2) * 15 = 3.9$$

**Which means you only
need
to traverse 3 edges to get
to the most bottom
leaf**

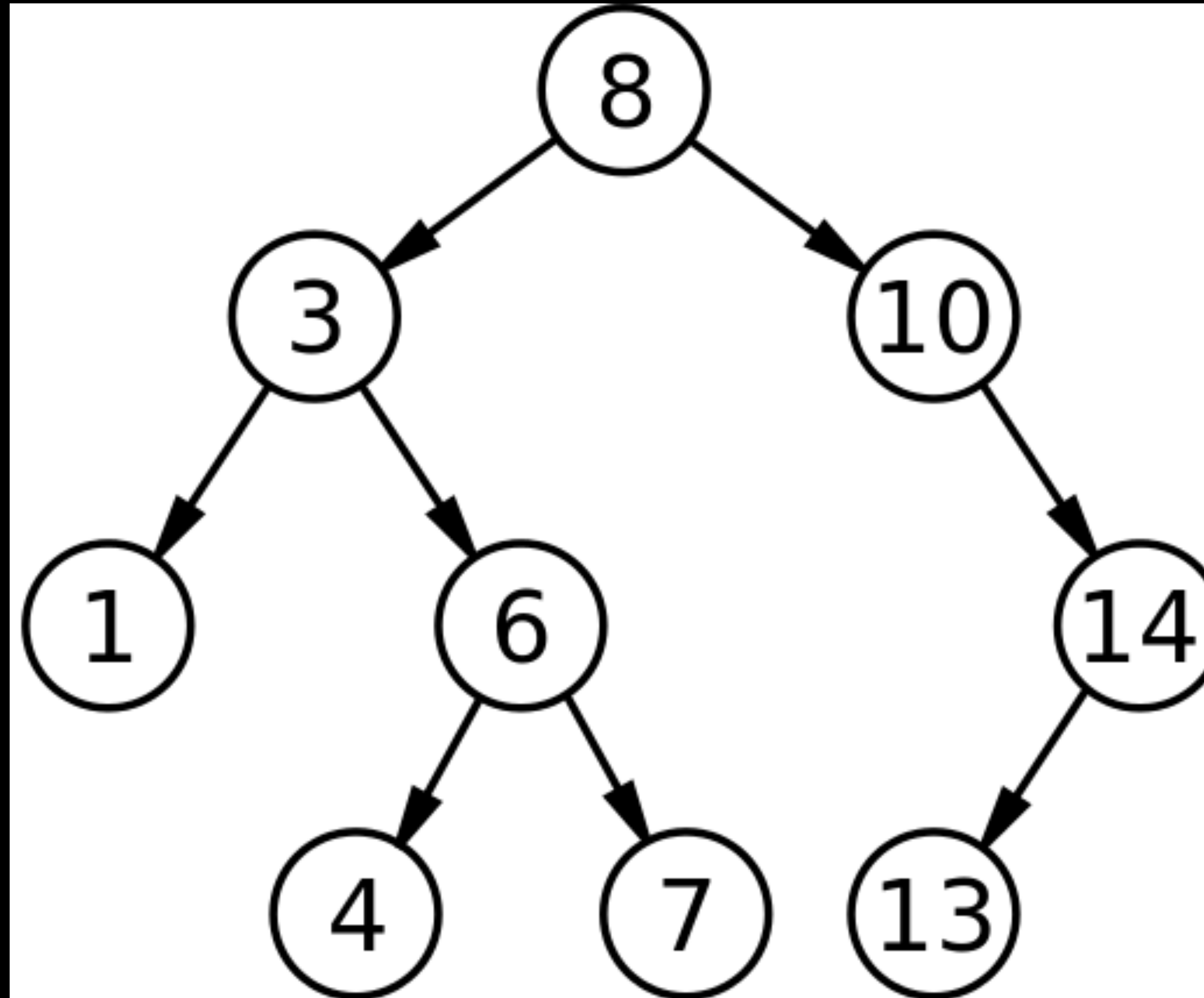


Binary Search Tree

Back to that definition:

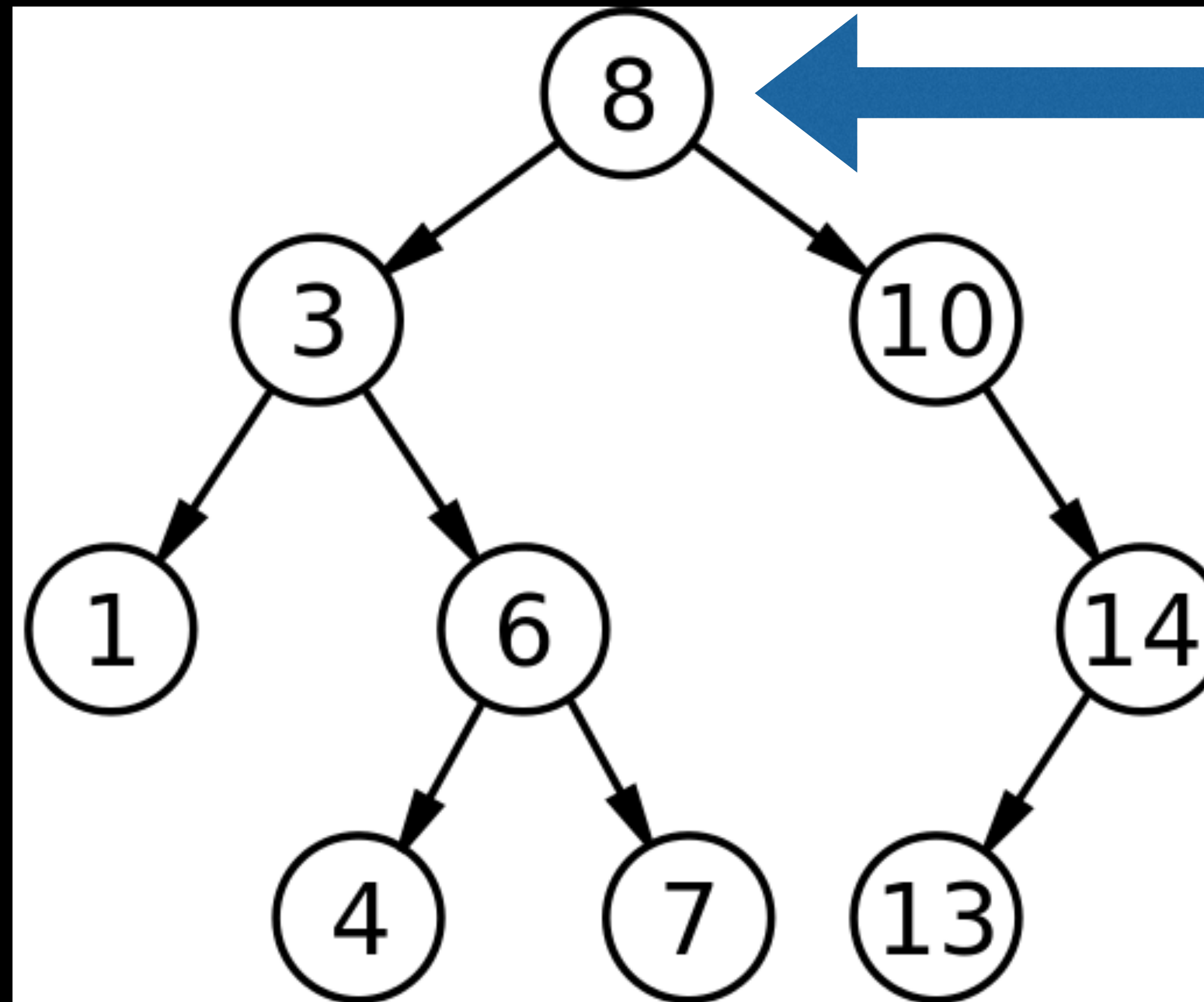
- “A binary search tree is a binary tree where each node has a comparable key and satisfies the restriction that **the key in any node is larger than all keys in all nodes in that node’s left subtree, and smaller than the keys in all nodes in that nodes right subtree**”

Binary Search Tree



Binary Search Tree

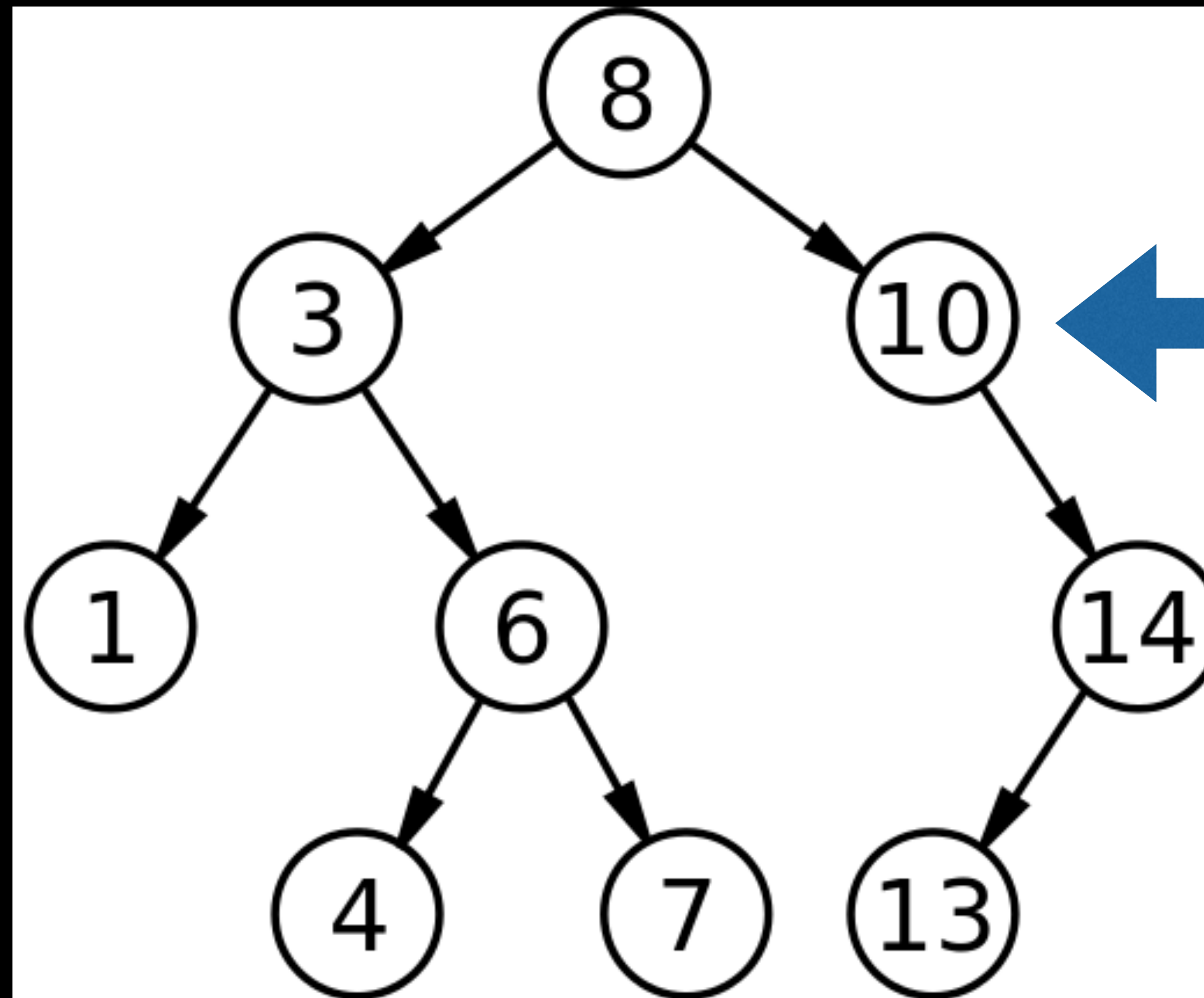
What if wanted to insert 15 into this tree?



We first try to insert it at the root, 15 is larger than 8, and 8 already has a right child, so...

Binary Search Tree

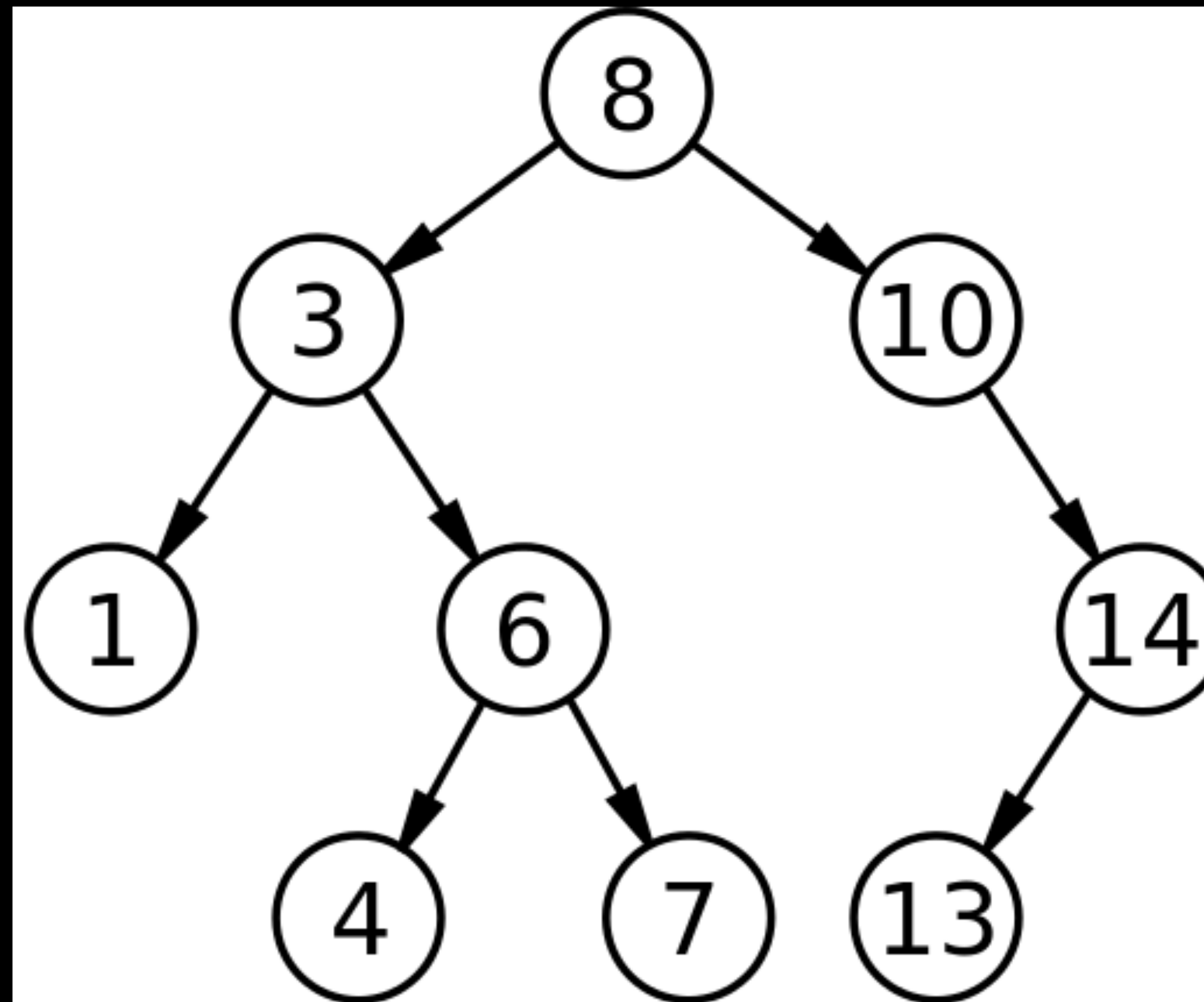
What if wanted to insert 15 into this tree?



We now ask 10, 15 is greater than 10, and 10 node already has a right child, so....

Binary Search Tree

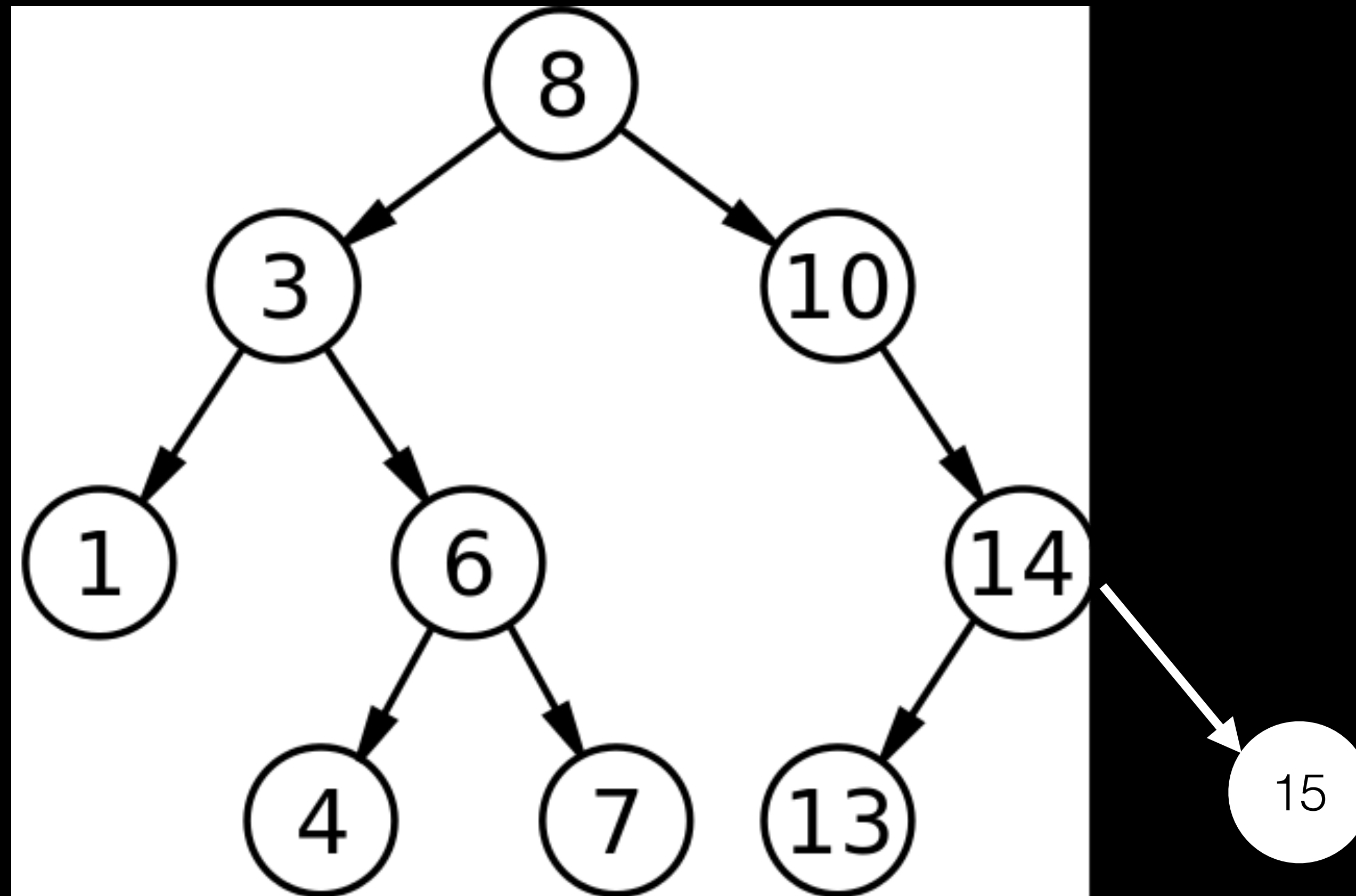
What if wanted to insert 15 into this tree?



We now ask the 14 node. 15 is greater than 14, and 14 has no right child node!

Binary Search Tree

What if wanted to insert 15 into this tree?



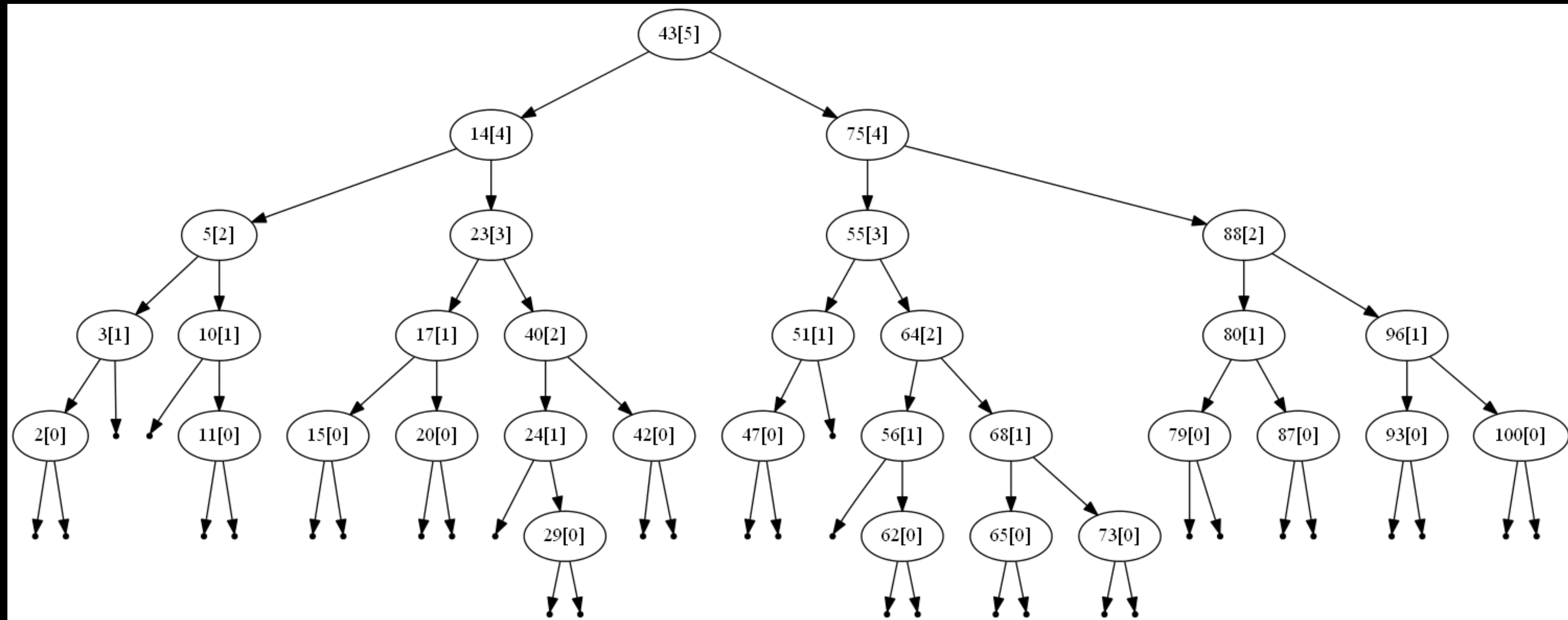
15 now takes it right
rightful place in the BST

So why is a binary search tree such a commonly used data structure?

- Because it rocks.
- Specifically, it rocks for searching.
- BST's have a BigO of $O(\log n)$ for search, thats great!
- Lets take a look

Search

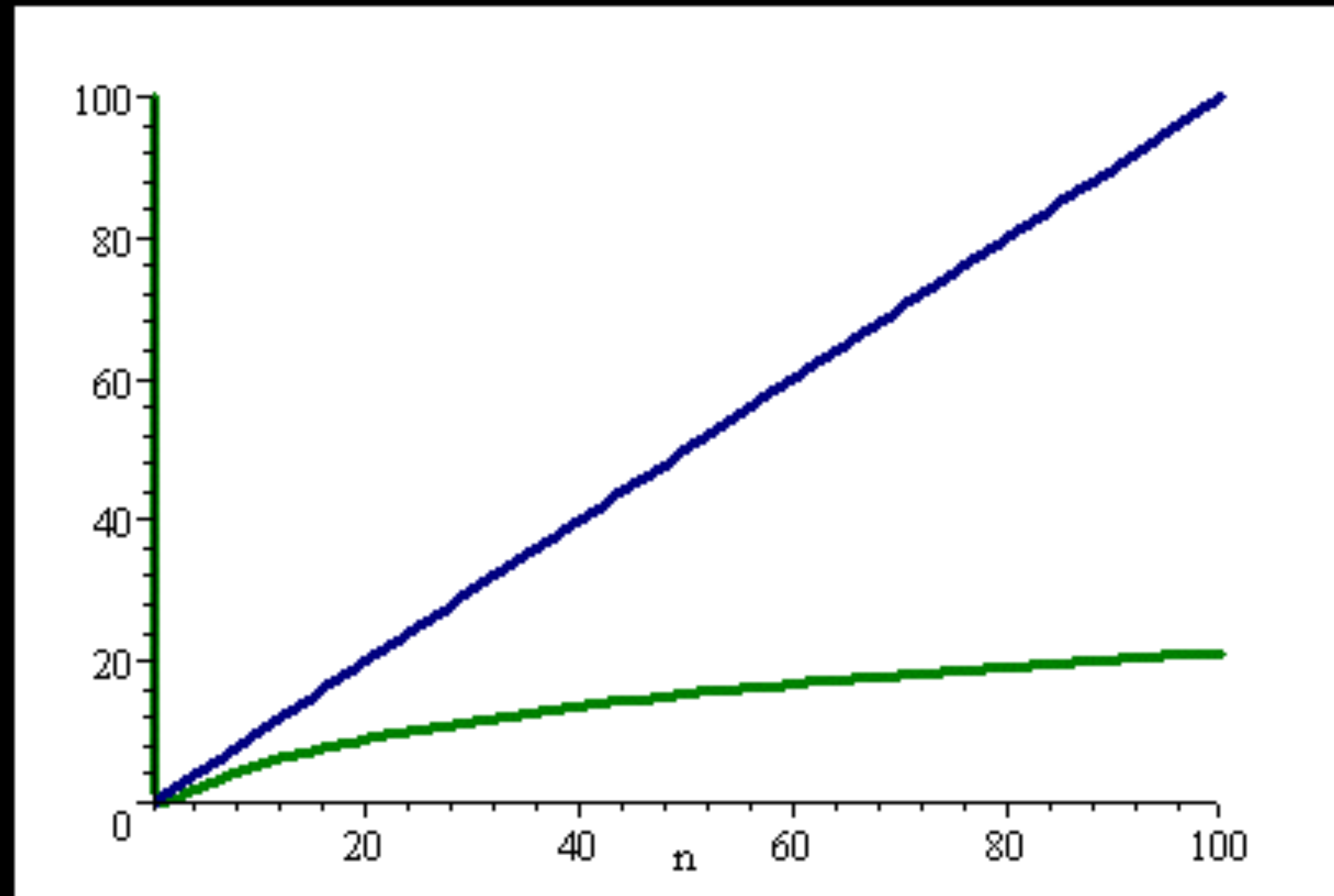
Heres a large looking BST. It has 32 nodes.



$\log(2) * 32 = 5$. At most it would take us 5 traversals, or steps, through this BST to find what we are looking for.

$O(\log(n))$ vs $O(n)$

Look how slowly $\log(n)$ rises compared to the blue (n)



- If your input size n is one million, the big o of n is 1000000, while $\log(n)$ is 19.93

BST and $\log(n)$

- So a Binary search tree with 1 million nodes takes at most 20 steps to find what you are looking for
- Doubling the node count to 2 million, only increases the necessary steps by 1. Thats good.

Implementing a BST

- When implementing a BST, you can do it a number of different ways.
- I prefer to do it by creating both a binary search tree class, and then a node class to go along with it.

BinarySearchTree Class

- Your Binary search tree class can be pretty simple
- The only property it needs is a node property called root. This is your root or topmost node.
- Just like a linked list, as long as you have a strong reference to the root node, the rest of the tree stays alive
- And like a linked list, you will need to come up with ways to traverse the tree.
- The minimum functionality your BST needs is to add a value and find a value. Deleting a value can be good to learn as well.

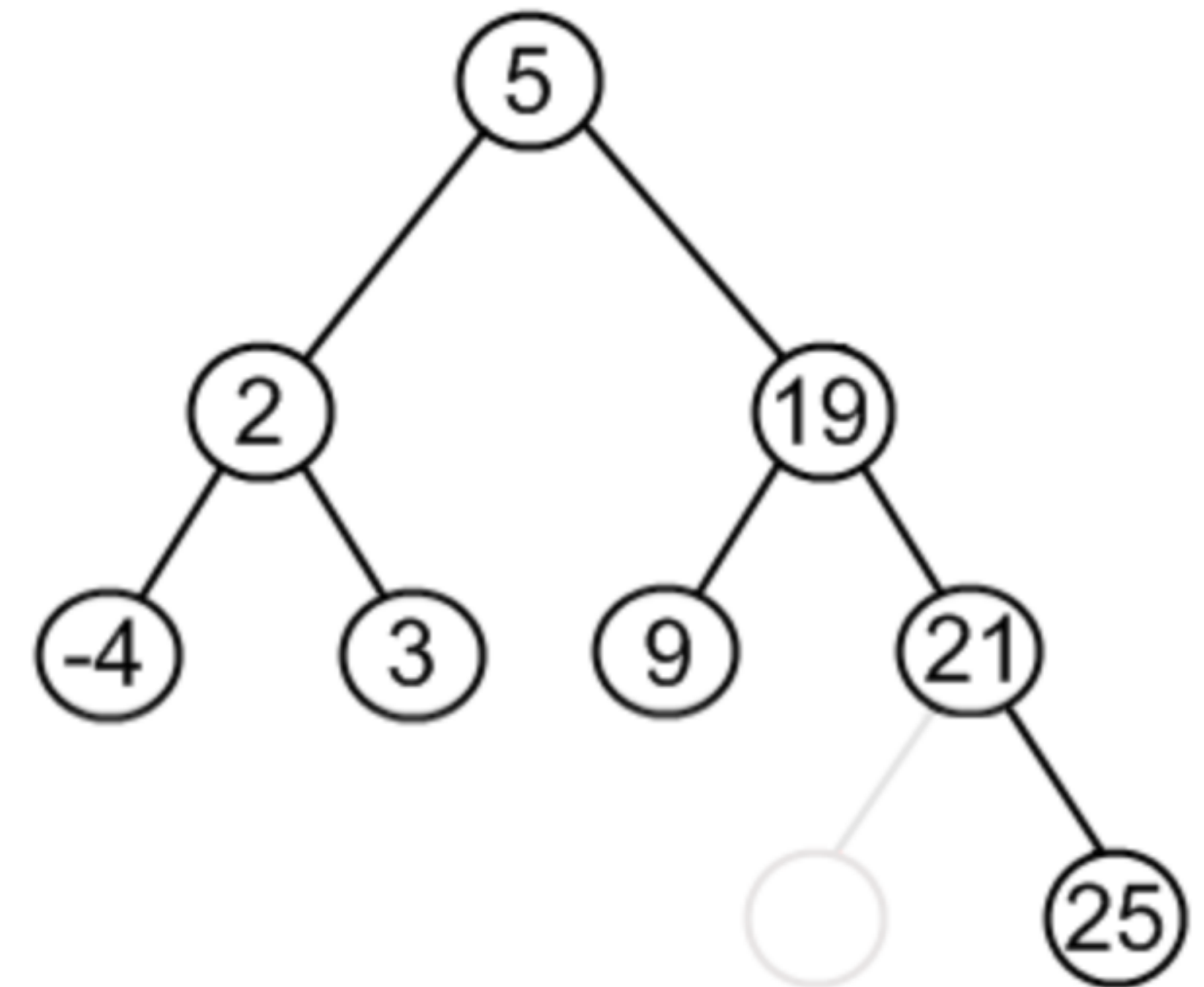
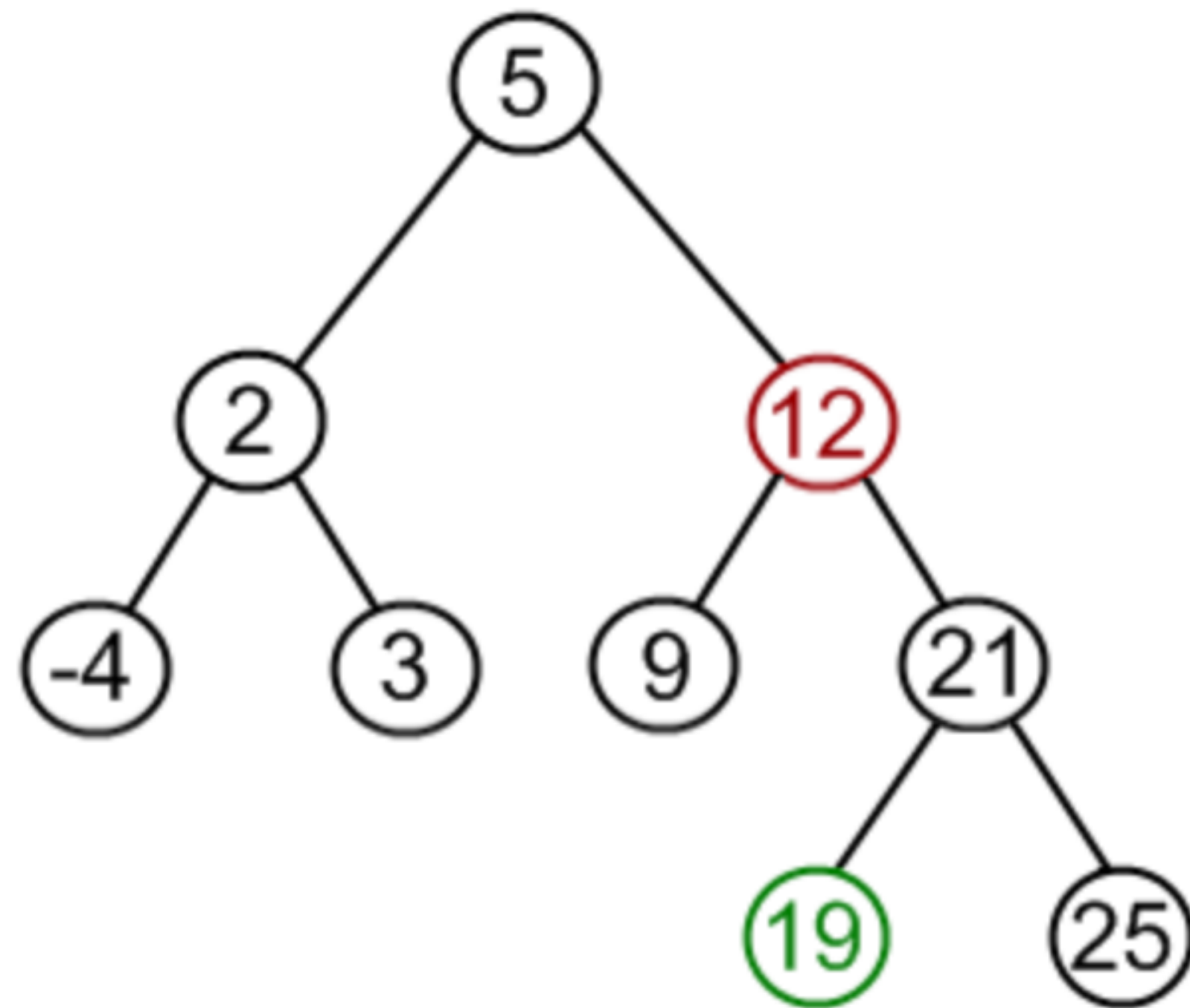
Demo

Deleting a value

- There are basically 3 separate cases when you need to remove a node from the tree:
 - Node to be removed has no children. Super easy, just nil out the reference that the parent node has to this node.
 - Node to be removed as one child. Super easy, just make the single child take the place of its parent node that is being removed
 - Node to be removed as two children. Not easy.

Removing a node with 2 children

- A rule you can follow when removing a node: replace it with the minimum element from the node's right sub tree:



Demo

Wrap Up

Big Topics

- Core Data
- Date Handling
- No IB interface
- TDD
- Core Data advanced (Multiple Contexts, Migrations, iCloud)

Extra Credit

- Store Images for each hotel in core data, have some sort of interface allowing the user to swipe through them.
- Add WatchKit support to the app, showing you any reservations that start today