Revised: 10/29/2019

Trees

Michael C. Hackett

Computer Science Department

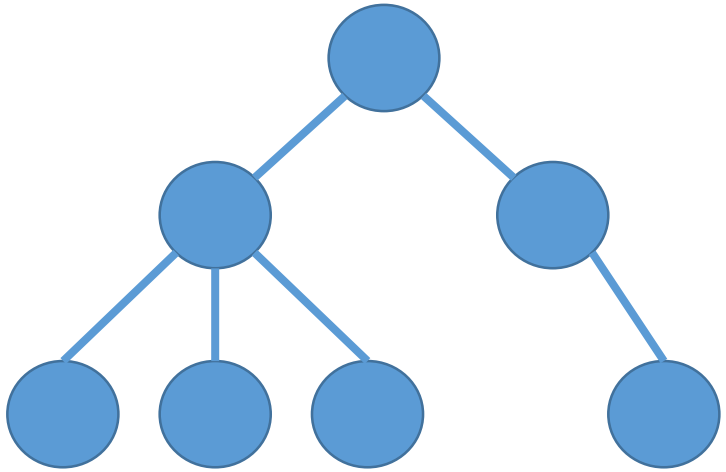
Community College of Philadelphia

Lecture Topics

- Tree Terminology
- Binary Trees
  - Tree Traversals
- Binary Search Trees
- N-ary Trees
- Complexity of Trees
- Other Tree Classifications

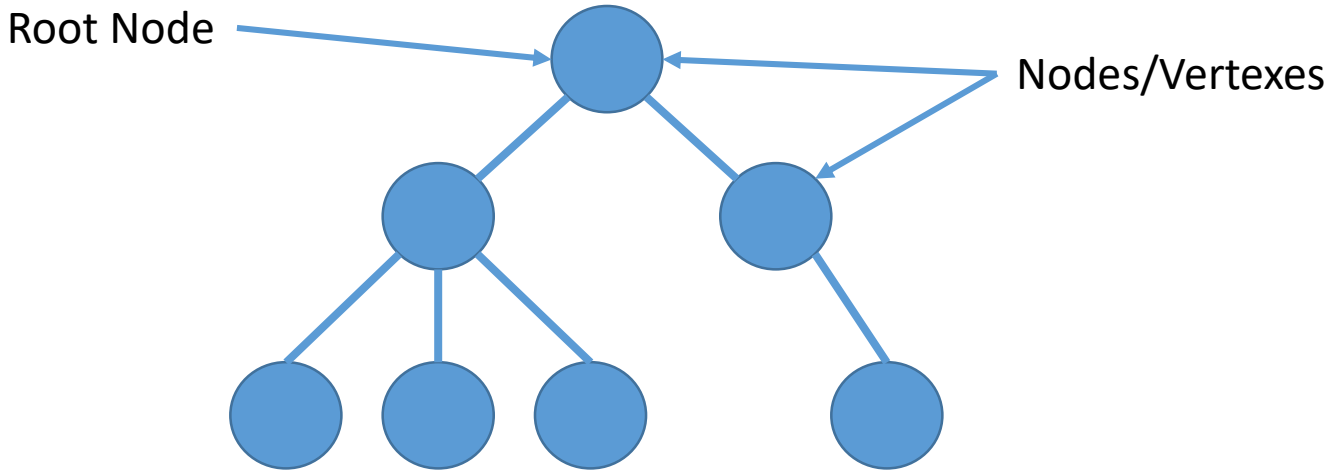
Trees

- A **tree** is a non-linear data structure, where each point in the tree will branch into zero or more points.



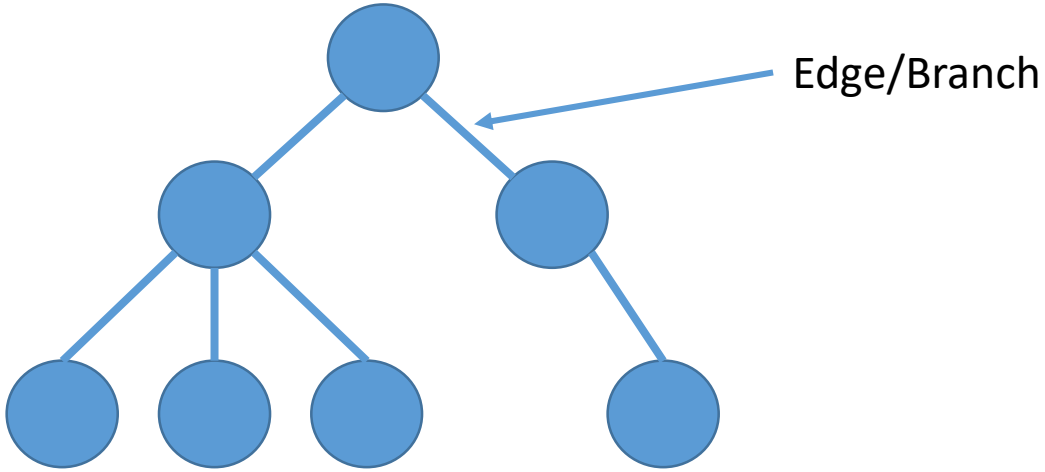
Trees

- Each point in the tree is called a **node** or **vertex**
- The top-most node is called the **root** node



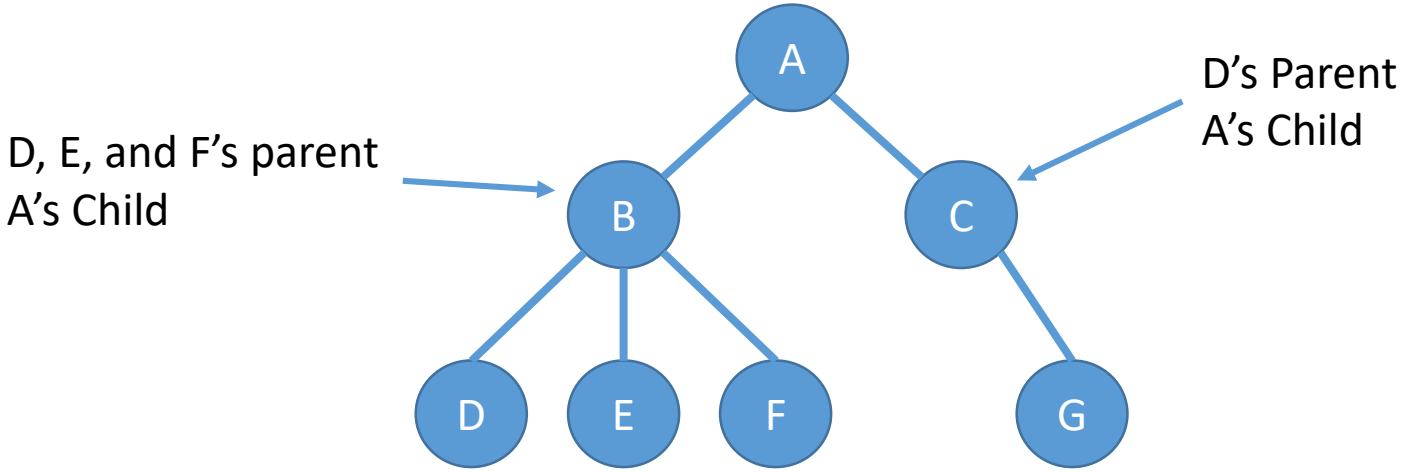
Trees

- The lines connecting the nodes are **edges** or **branches**



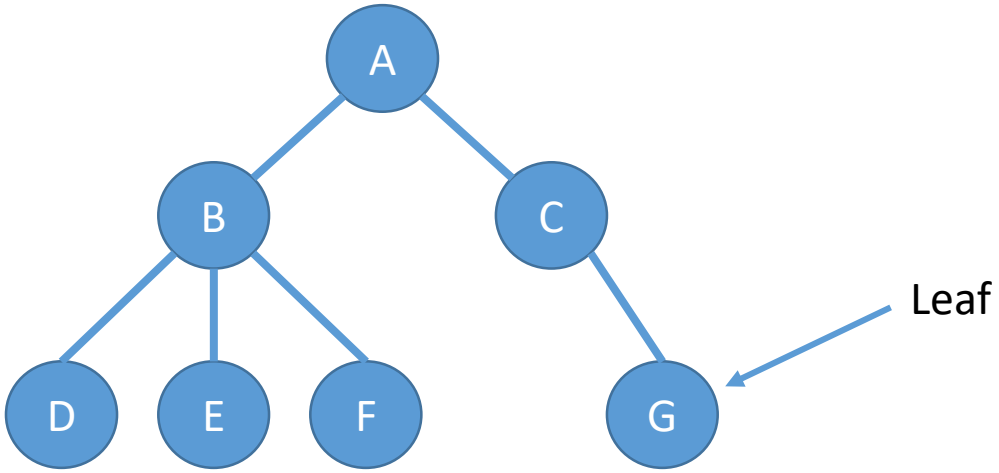
Trees

- **Children** or **child nodes** are nodes that branch from a higher node.
- A node's **parent** is the node it branches from.
  - The root node will not have a parent.



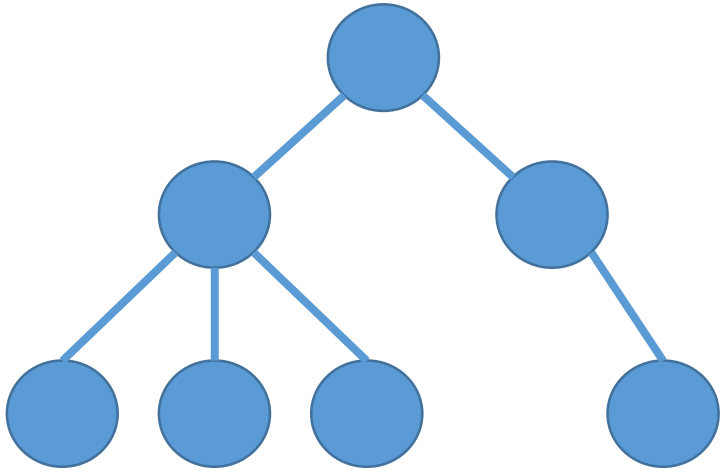
Trees

- A node with no children is called a **leaf** or **leaf node**

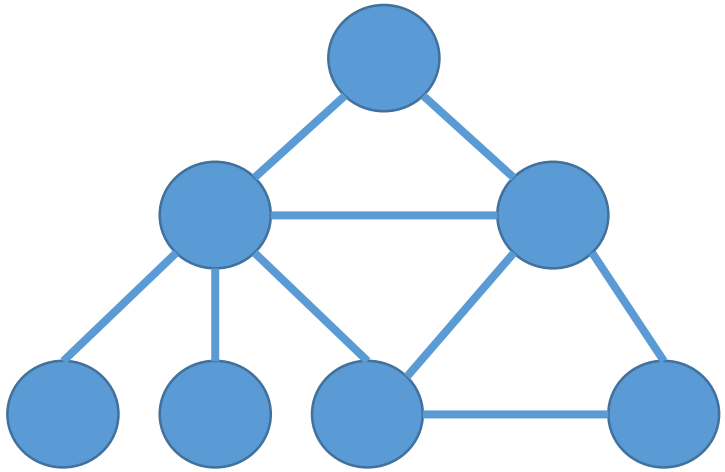


Trees

- Major Characteristic:
  - Only **one** path from the root to any node in the tree



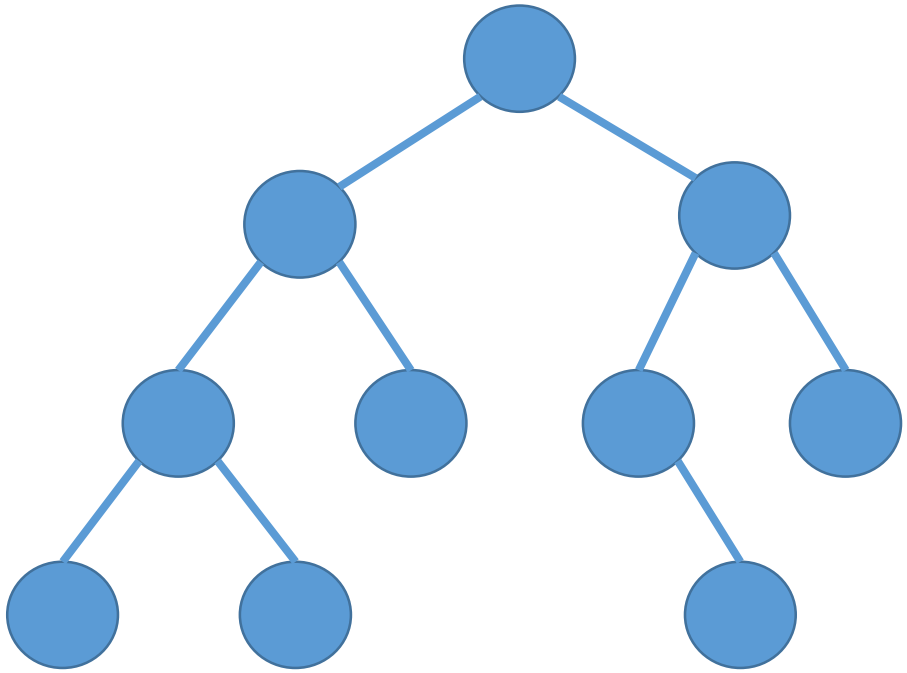
Tree



Not a Tree

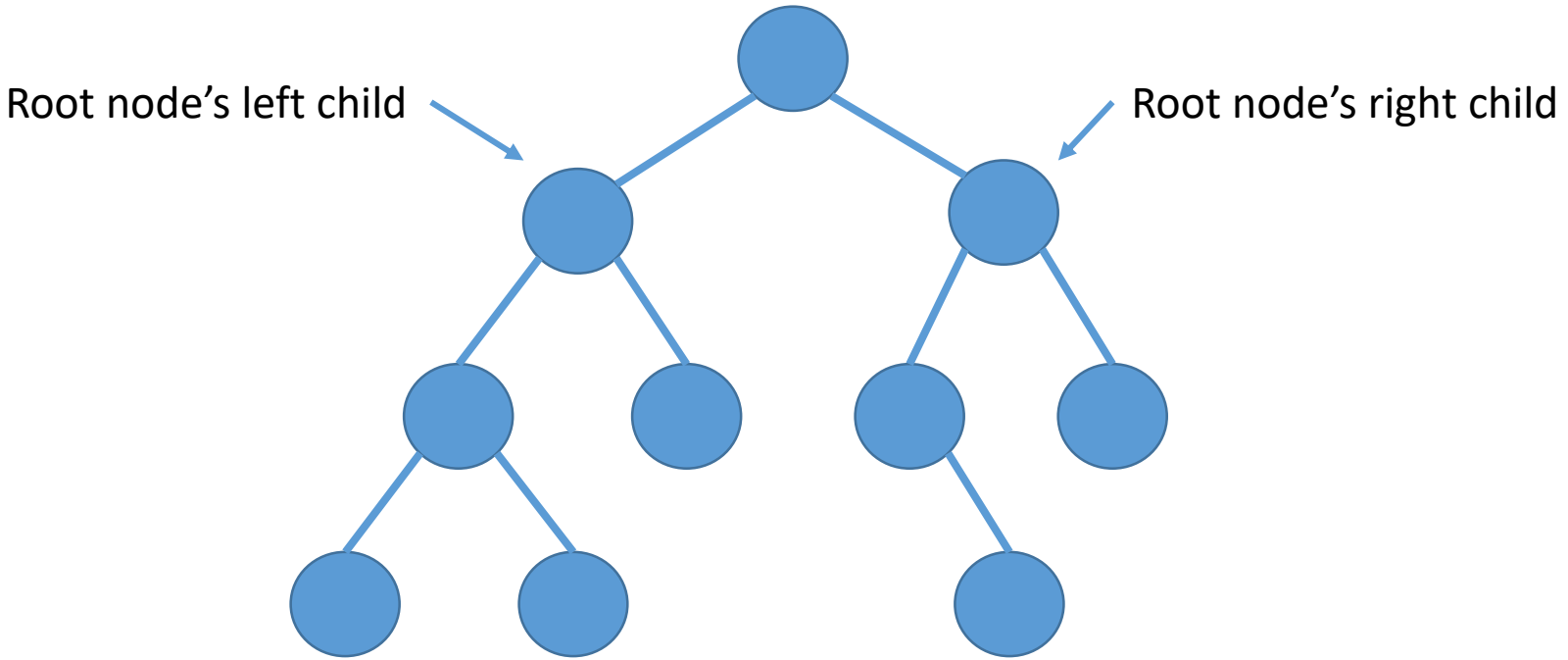
Binary Trees

- While trees can be built without limits for the number of children a node may have, the **binary tree** only allows up to two children for each node.



Binary Trees

- The children of a node are often referred to as the left child and right child

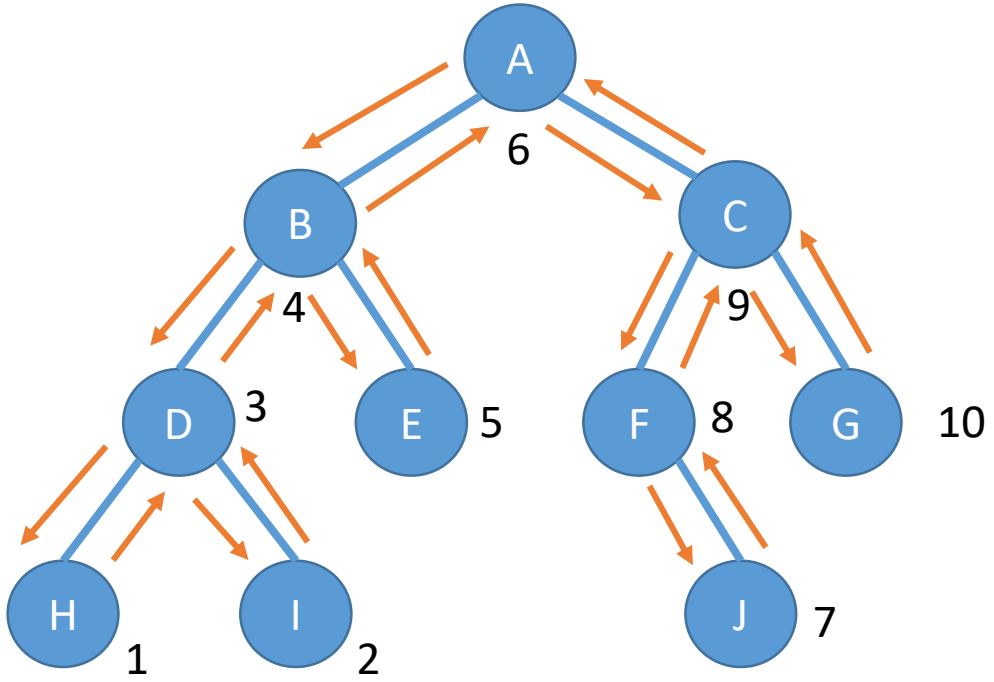


Tree Traversals

- In-Order Traversal
  - Traverse down the left side
  - Use the node's value/data
  - Traverse down the right side
  - In other words, the value of the node is used upon the **second** time it is visited
- Pre-Order Traversal
  - Use the node's value/data
  - Traverse down the left side
  - Traverse down the right side
  - In other words, the value of the node is used upon the **first** time it is visited.
- Post-Order Traversal
  - Traverse down the left side
  - Traverse down the right side
  - Use the node's value/data
  - In other words, the value of the node is used upon the **last** time it is visited.

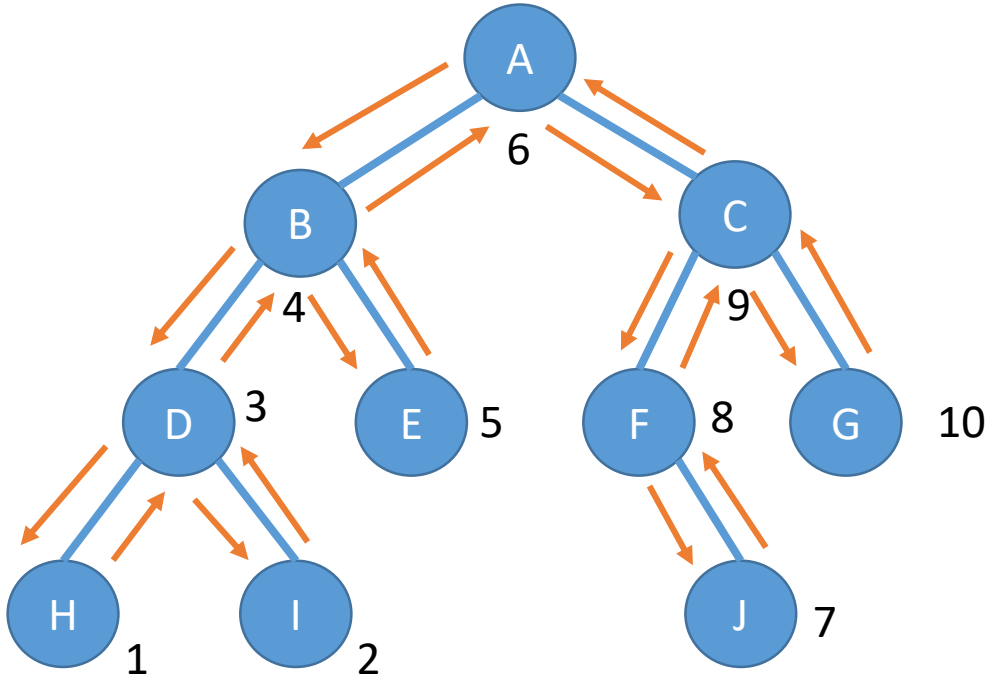
In-Order Traversal

- Arrows show the direction of the traversal.
- Numbers indicate when the values of the nodes are used in the traversal.



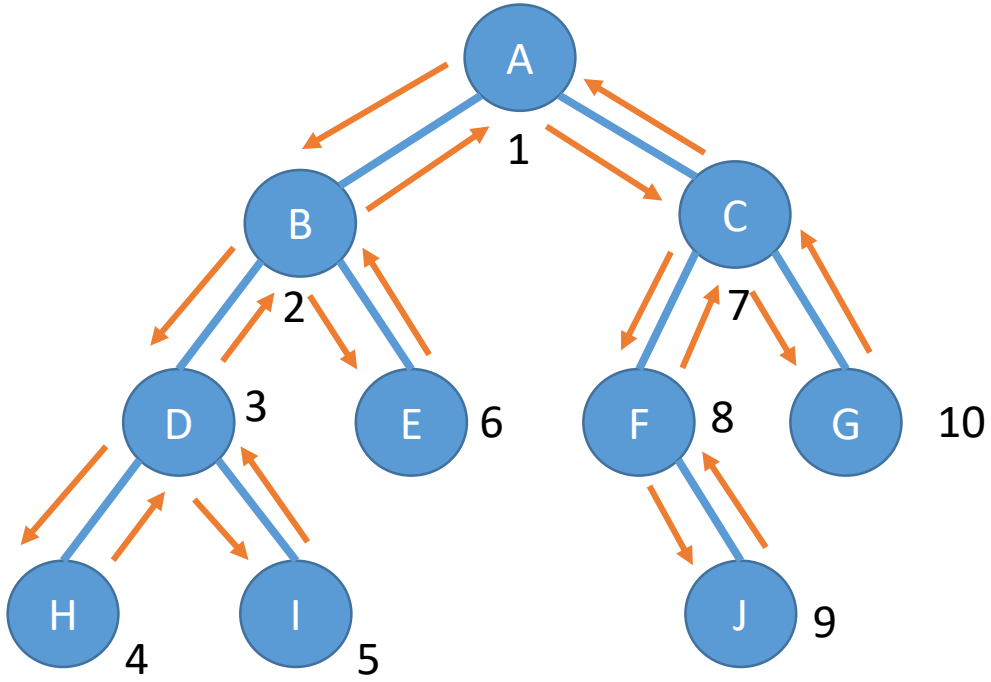
In-Order Traversal

- **Infix Format**: H I D B E A J F C G



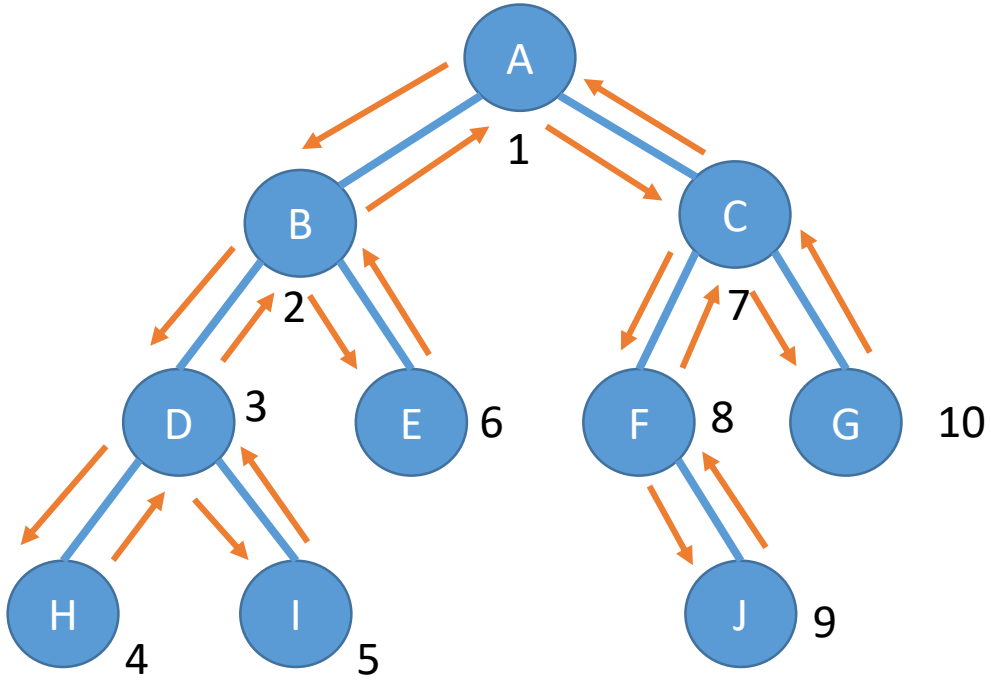
Pre-Order Traversal

- Arrows show the direction of the traversal.
- Numbers indicate when the values of the nodes are used in the traversal.



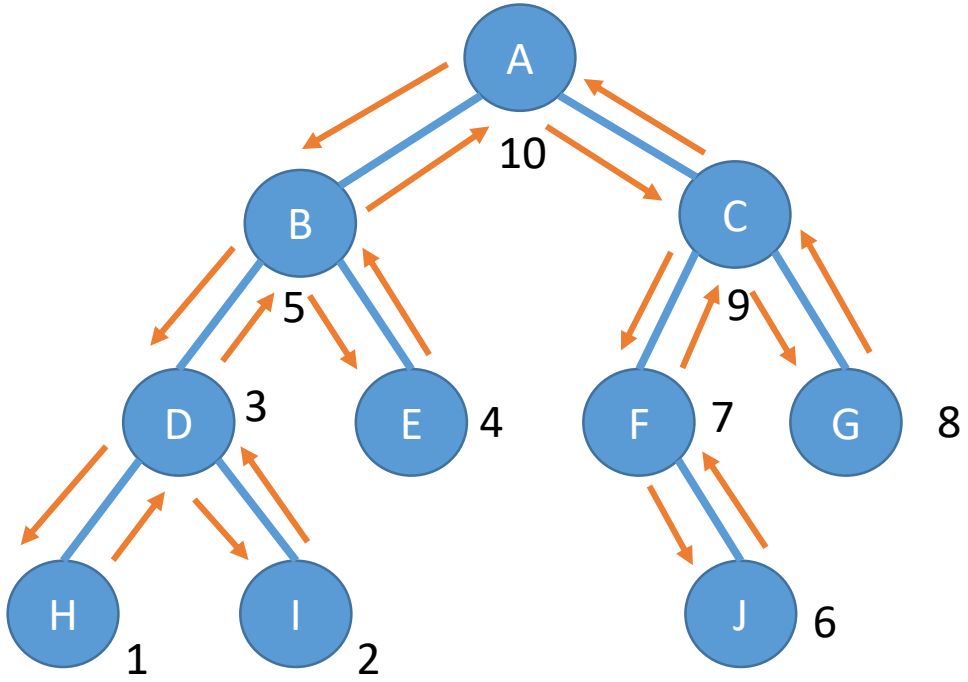
Pre-Order Traversal

- **Prefix Format**: A B D H I E C F J G



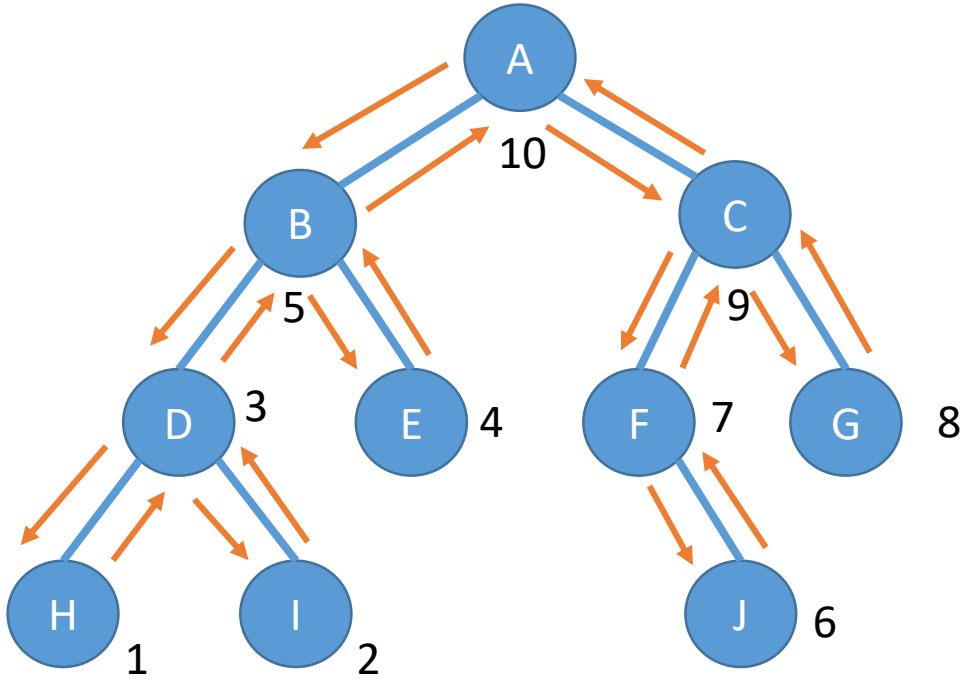
Post-Order Traversal

- Arrows show the direction of the traversal.
- Numbers indicate when the values of the nodes are used in the traversal.



Post-Order Traversal

- **Postfix Format**: H I D E B J F G C A



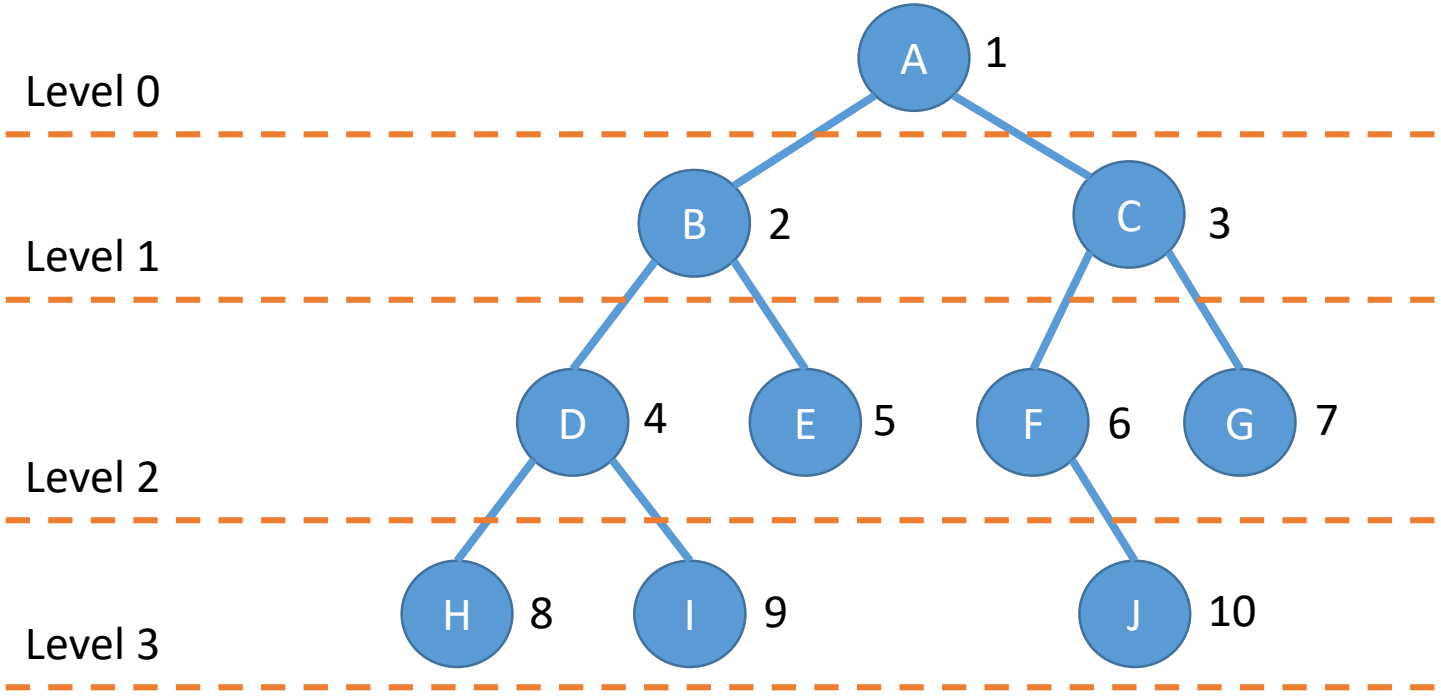
Depth-First Traversal

- In-Order, Pre-Order, and Post-Order are all examples of a **depth-first traversal**.

- The traversal algorithms always start by going to the lowest point on the left side.
  - Regardless of when each node's data/value is used.
  - Looking at the previous examples, the path taken is always the same.

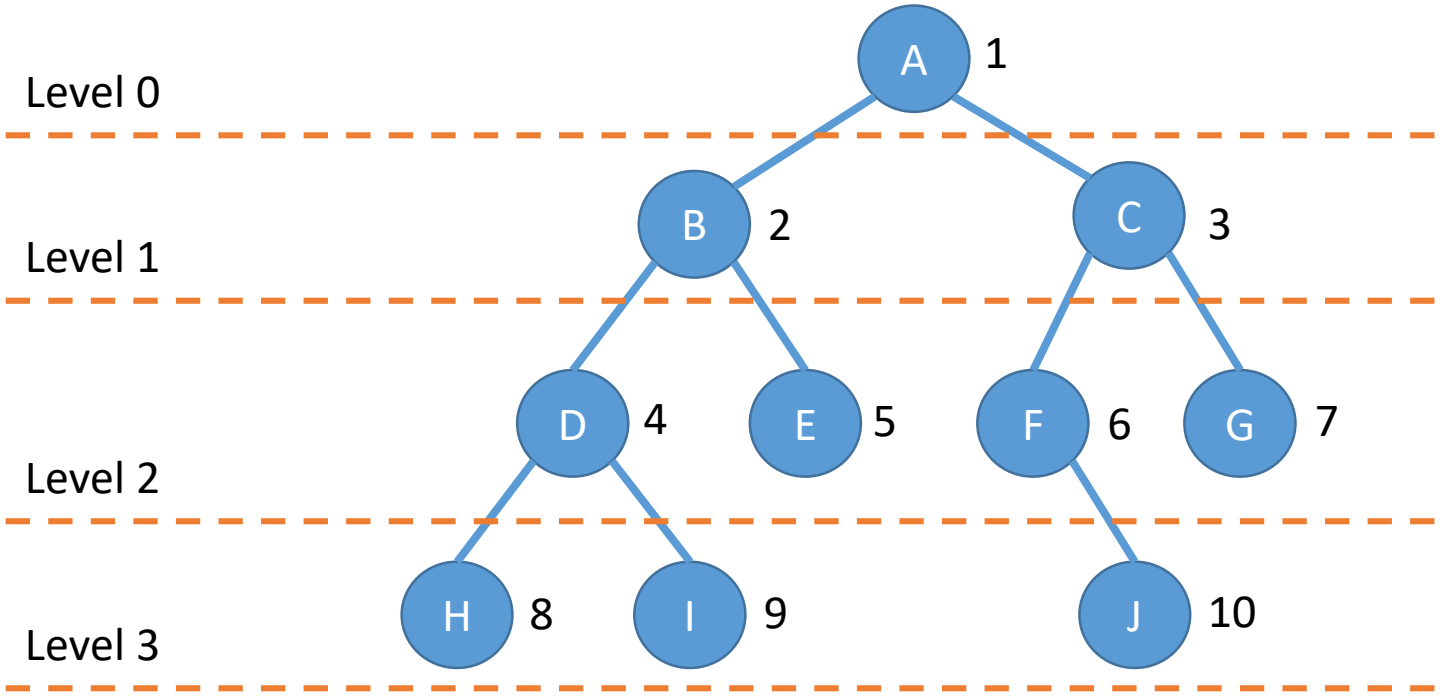
Breadth-First Traversal

- Using a **breath-first** or **level-order traversal**, the tree is traversed by visiting all nodes at each level of the tree, working its way to the bottom.



Breadth-First Traversal

- On a related note, this tree's **height** is 4



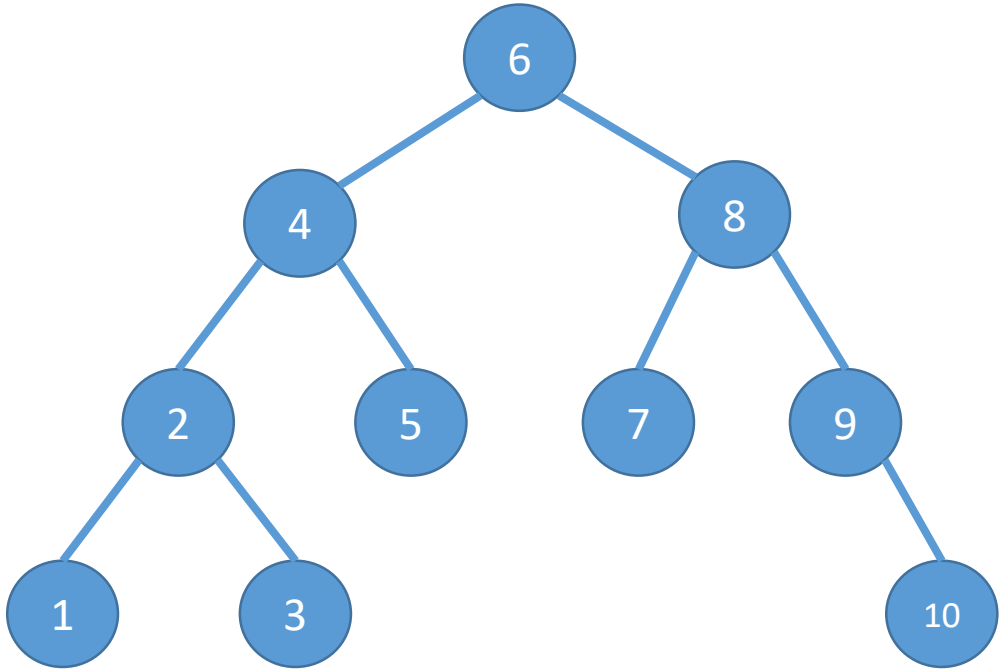
Binary Search Trees

- A **binary search tree** (or **BST**) is a binary tree where the node are comparatively added to preserve the natural ordering of the values stored in the tree's nodes.

- For each node:
  - Its left child node's value will be less than the node's data
    - As will all of its children
  - Its right child node's value will be greater than the node's data
    - As will all of its children

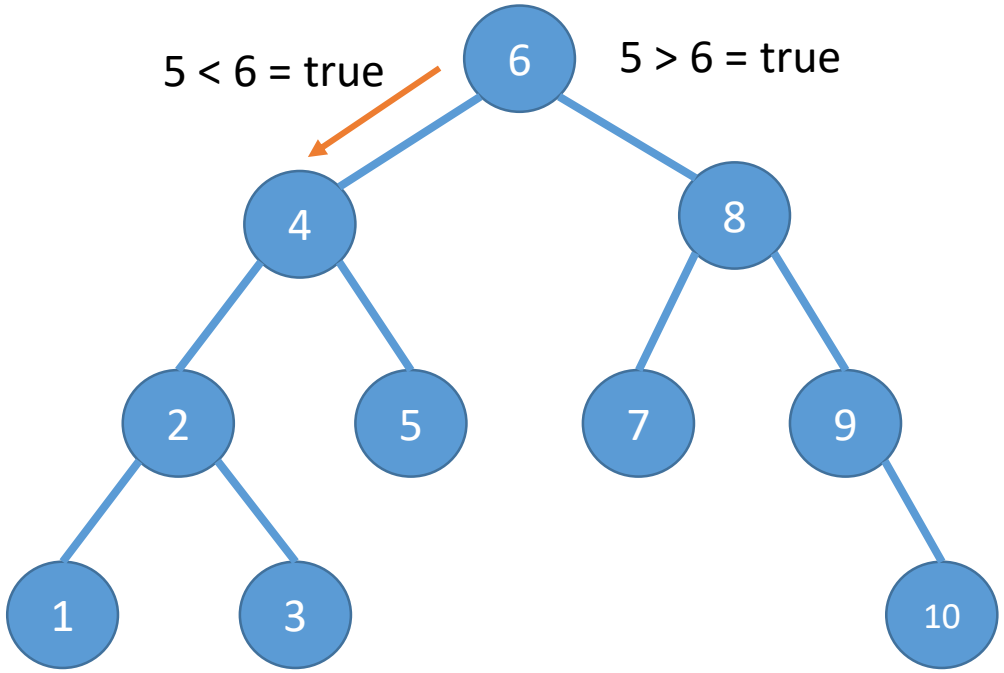
Binary Search Trees

- Left child's value is less than the parent's value
- Right child's value is greater than the parent's value



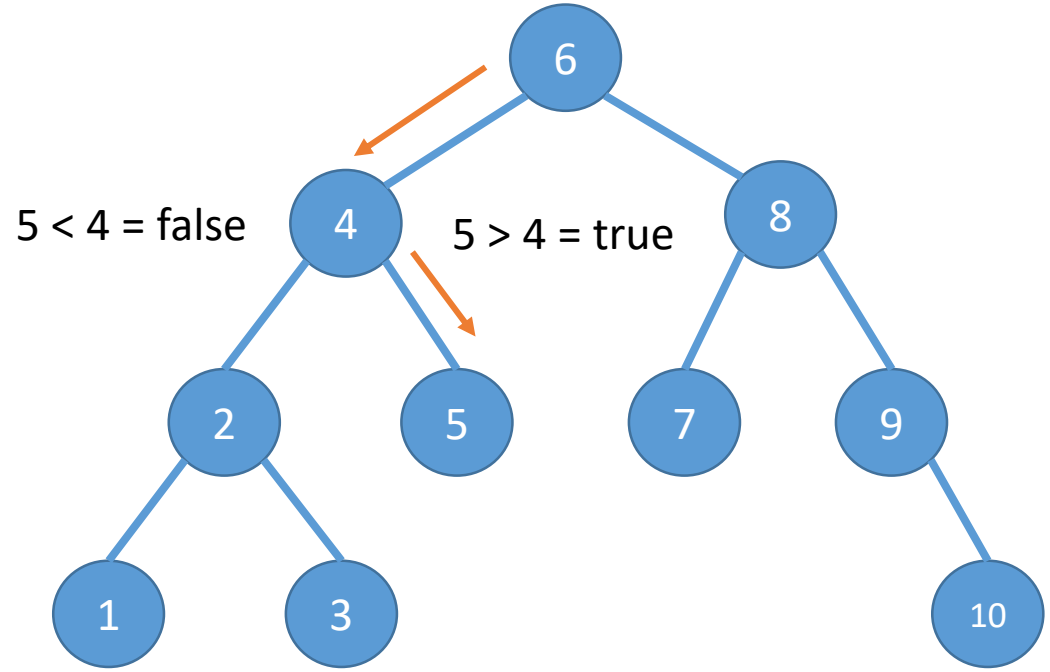
Binary Search Trees

- To search the tree for a value, simple > or < comparisons are used
  - Searching for 5



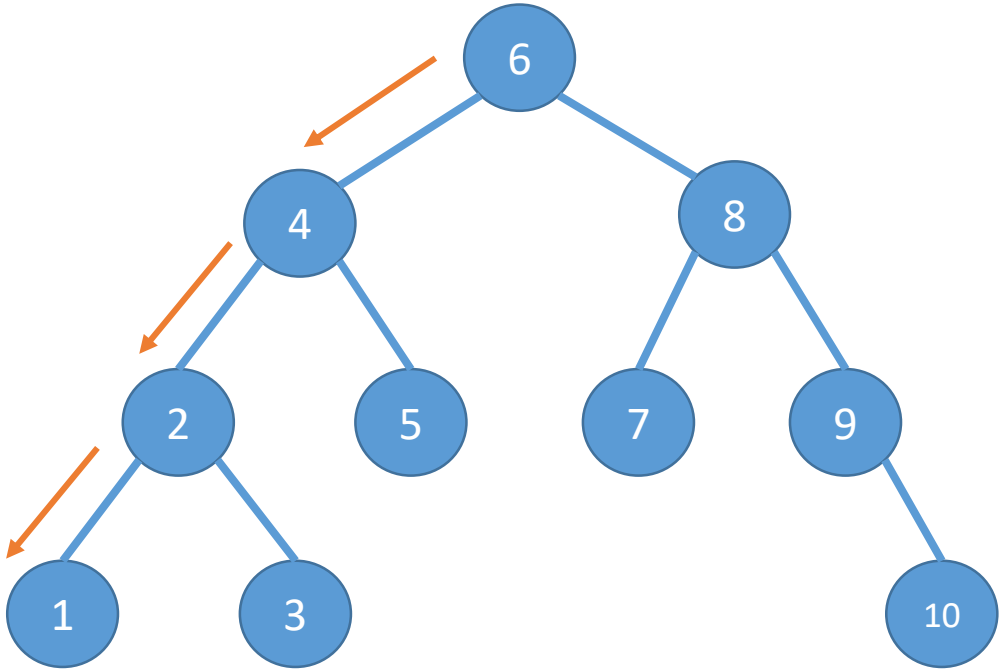
Binary Search Trees

- To search the tree for a value, simple > or < comparisons are used
  - Searching for 5
- A similar process is used for adding new nodes to a BST



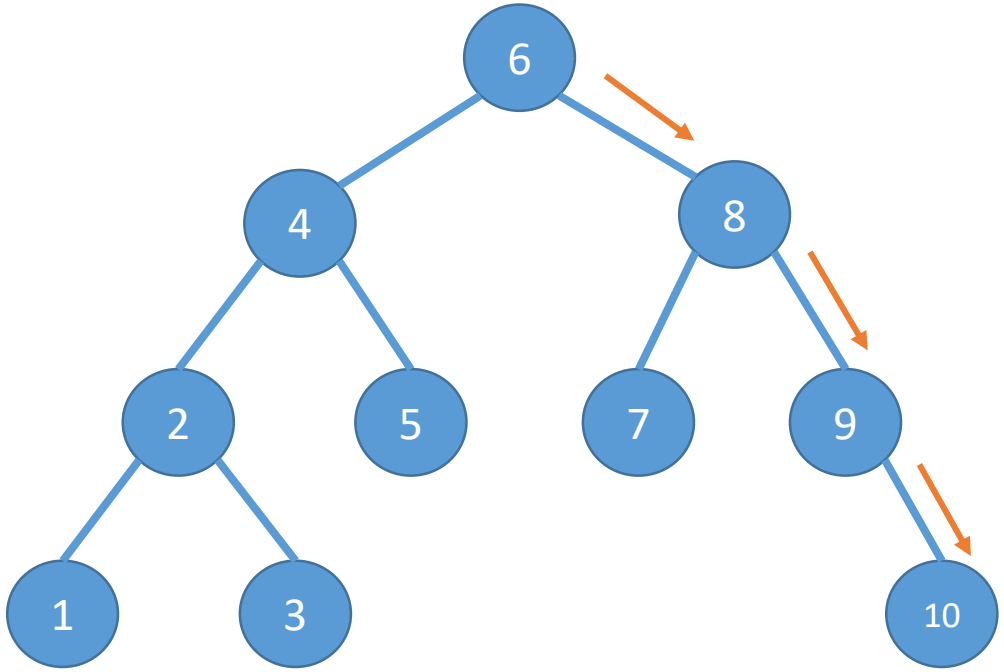
Binary Search Trees

- The smallest (min) value in a BST is always the left-most leaf.



Binary Search Trees

- The largest (max) value in a BST is always the right-most leaf.



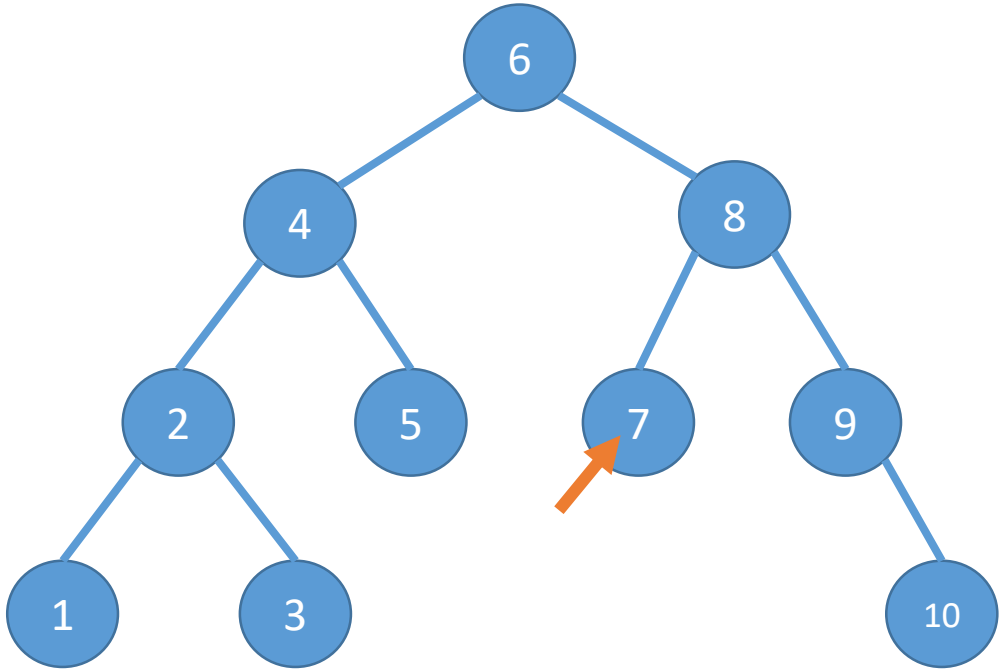
Binary Search Trees

- To remove a node, we need to determine its **successor**- the node that will replace it.

- If the node to remove..
  - Has no children – Safe to remove
  - Has only a right child – The right child is the successor
  - Has only a left child – The left child is the successor
  - Has both a right and left child – The smallest value down the right side of the node is the successor.

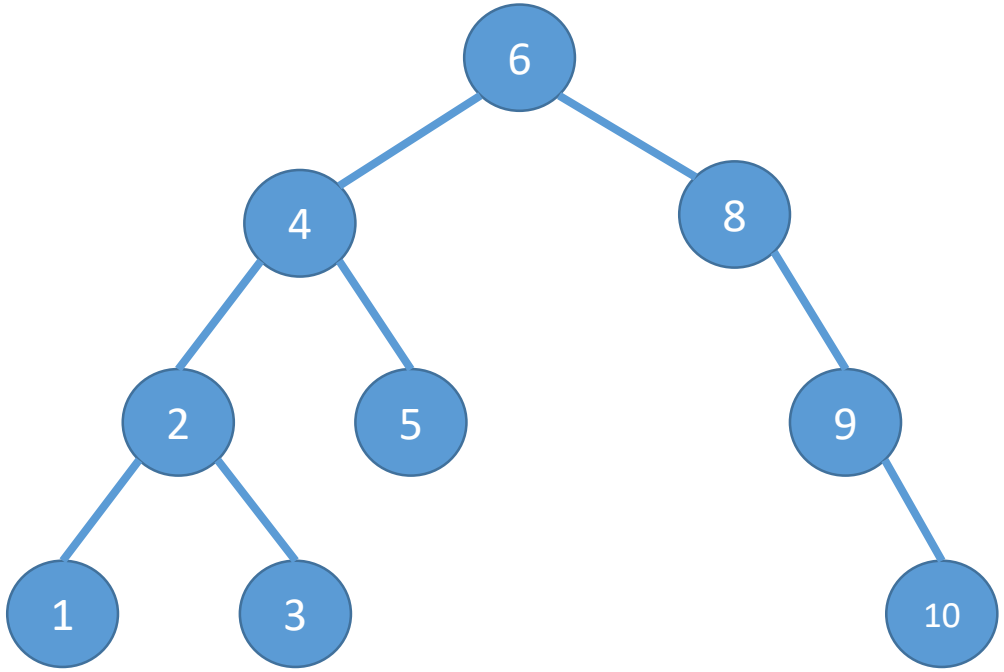
Binary Search Trees

- Removing the node containing 7…



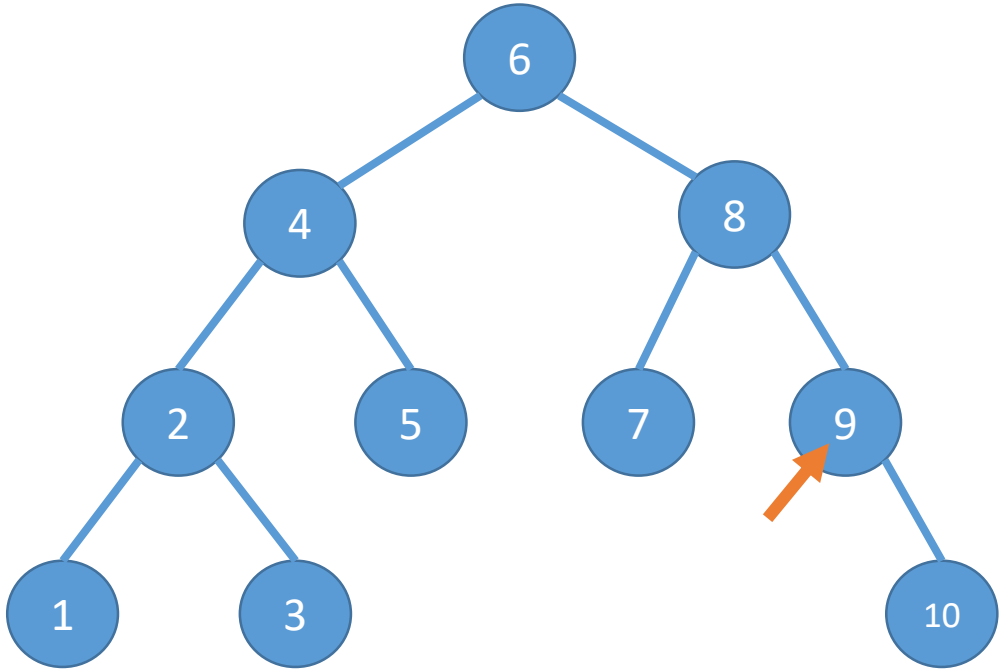
Binary Search Trees

- Removing the node containing 7… No successor



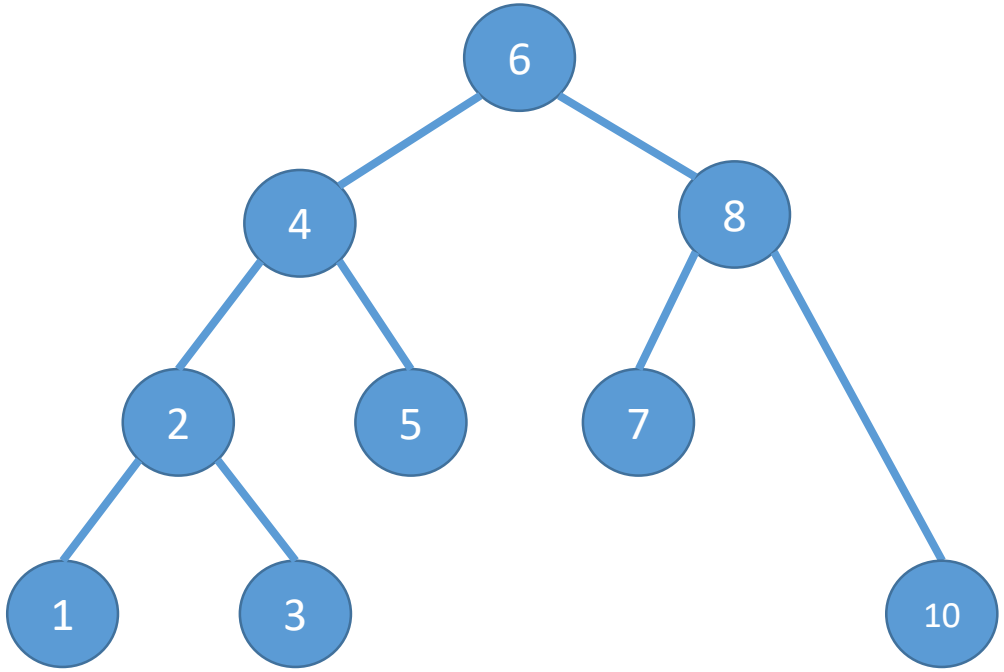
Binary Search Trees

- Removing the node containing 9…



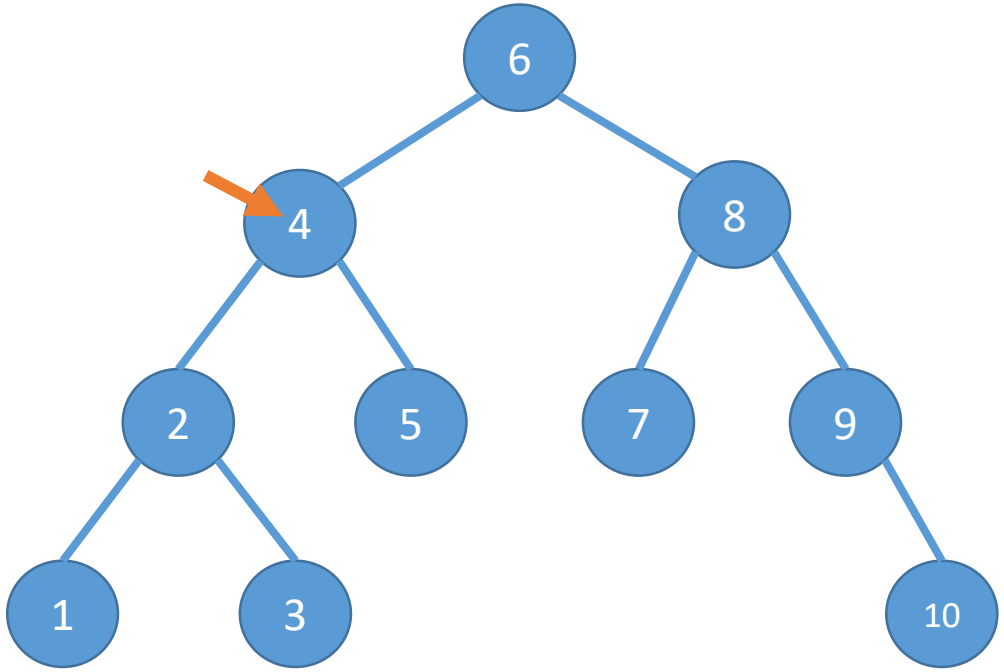
Binary Search Trees

- Removing the node containing 9… (didn't have a left child) the node containing 10 is its successor.



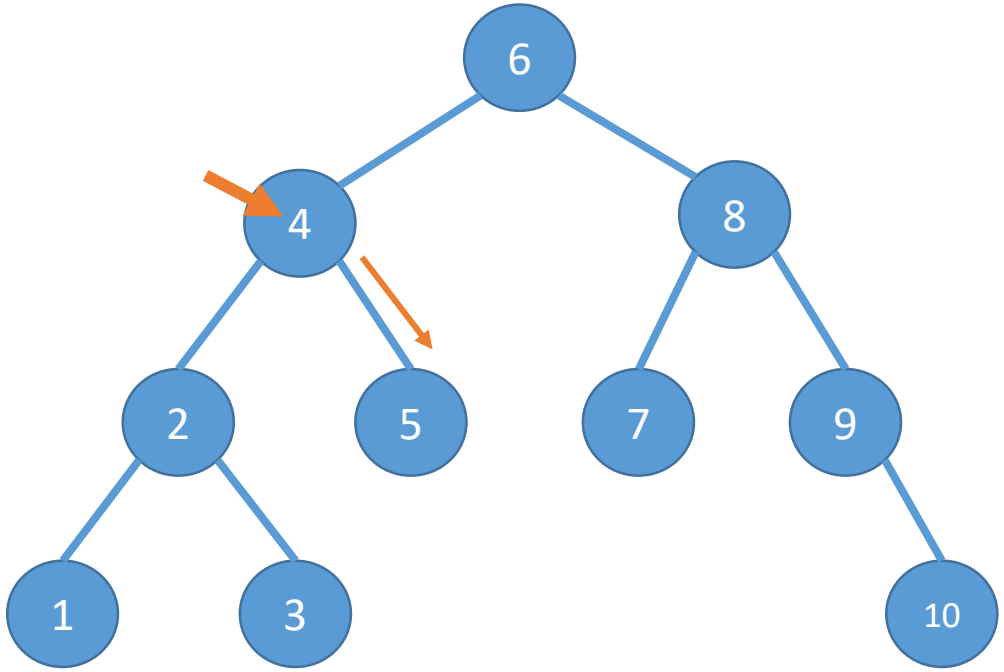
Binary Search Trees

- Removing the node containing 4…



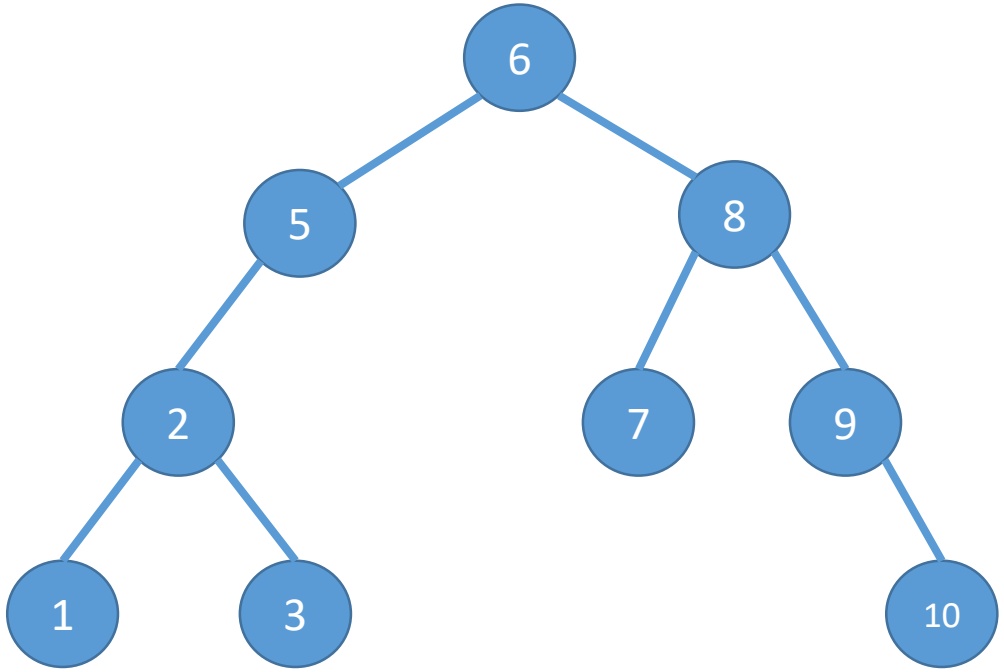
Binary Search Trees

- Removing the node containing 4… Goes down its right side looking for the smallest value (only one node to check)…



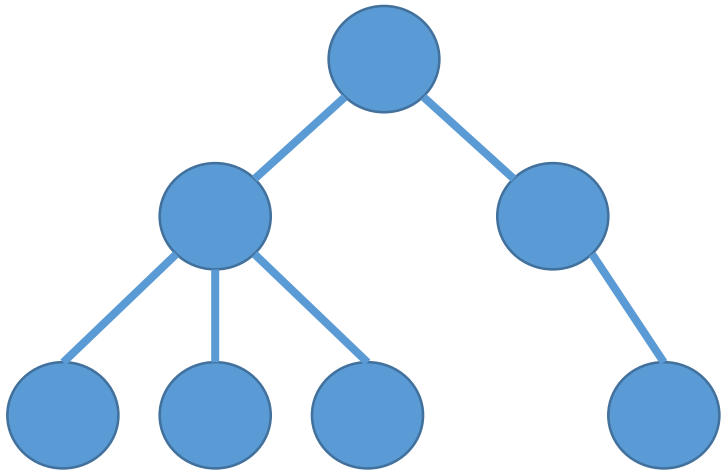
Binary Search Trees

- Removing the node containing 4… Goes down its right side looking for the smallest value (only one node to check)… 5 is the smallest, so that is its successor.



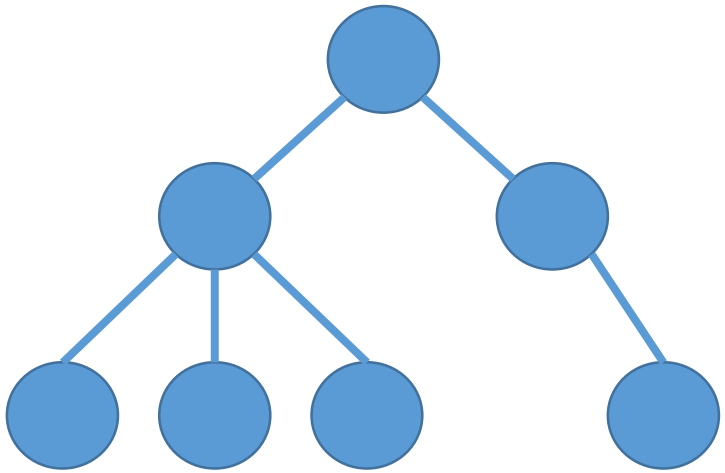
N-ary Trees

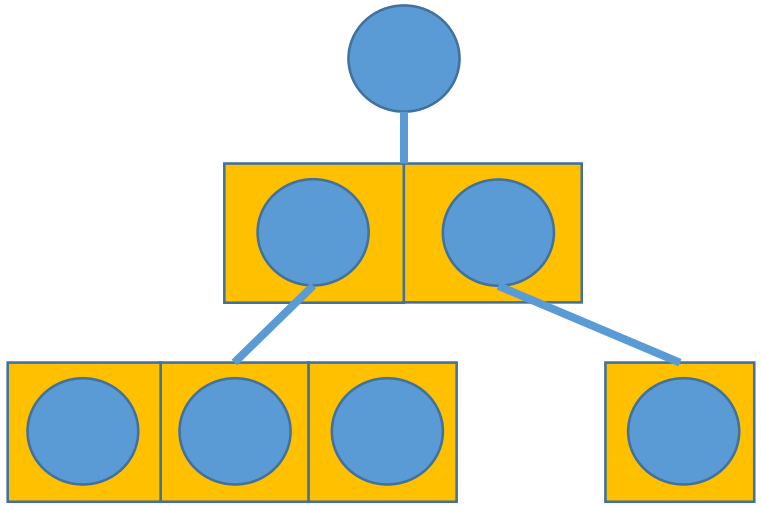
- An **n-ary tree** (or **general tree**) is a tree where each node may have any number of children.
  - The first tree shown at the beginning of the lecture was such a tree.



N-ary Trees

- Since we don’t know how many children each node has, it won’t have left or right children like a binary tree.
- Instead, each node maintains a list structure of its children.



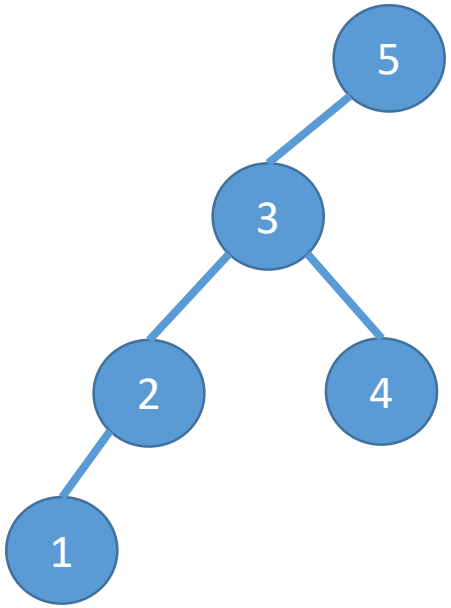


Tree Complexity

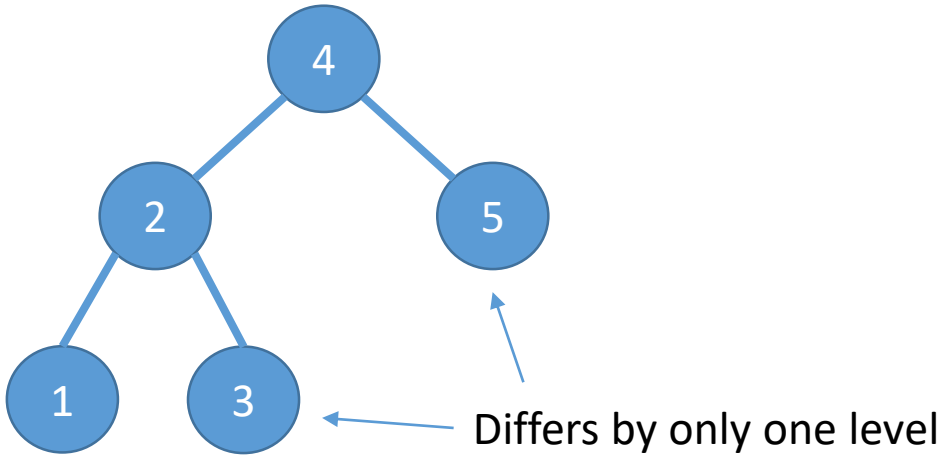
- Most tree operations such as adding/inserting, searching, and removing will have O(h) time complexity (where h is the height of the tree)

- How the tree is structured will have an impact.
  - A binary tree is **balanced** when the left side and right side differs by, at most, one level

Tree Complexity



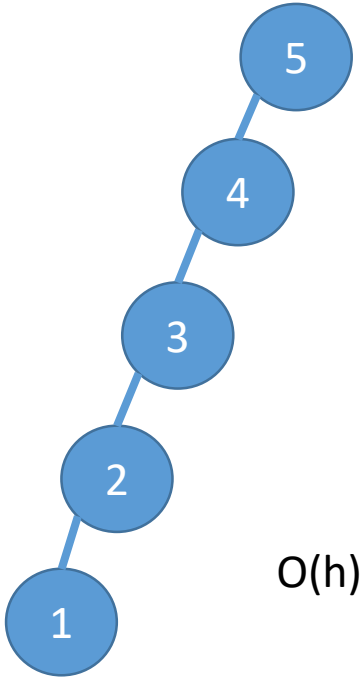
Unbalanced BST



Balanced BST

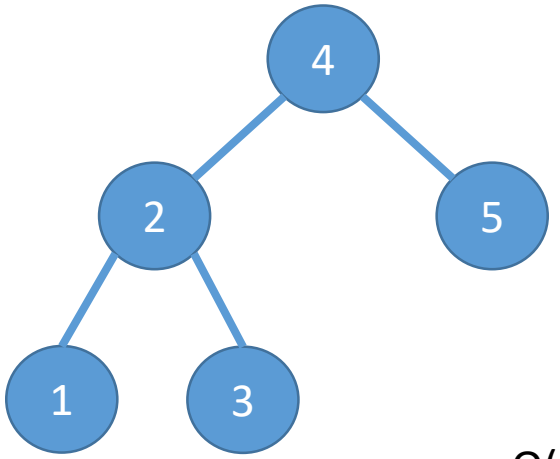
Tree Complexity

- Finding the minimum…



O(h) == O(n)

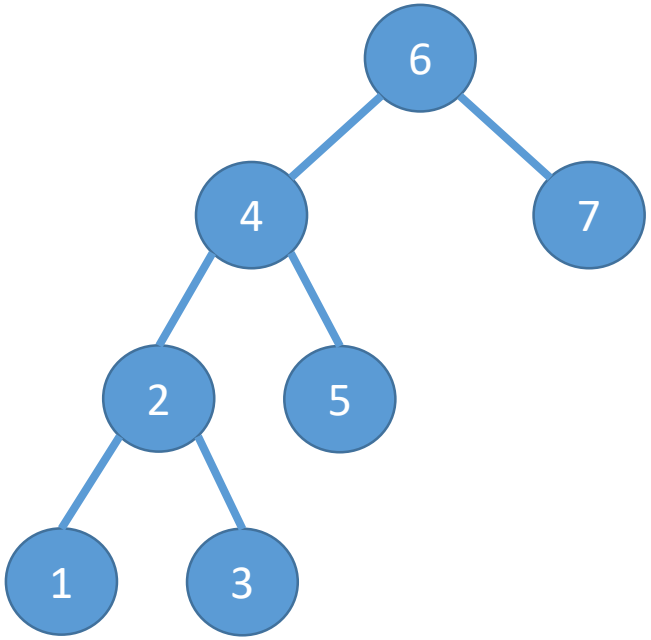
“Pathological Tree”



O(h)

Other Tree Classifications

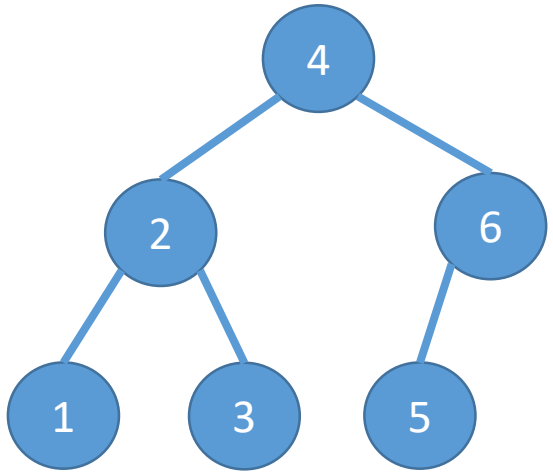
- A **full binary tree** is when every node has either 0 or 2 children



Full but not Balanced

Other Tree Classifications

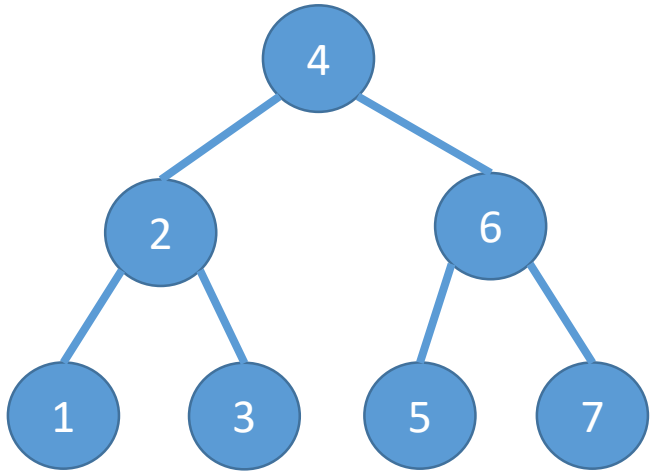
- A **complete binary tree** is when every level is filled (except for the last level) and the leaves are as far left as possible.



Balanced and Complete, but not Full

Other Tree Classifications

- A **perfect binary tree** is when every node has two children and the leaves are all at the same level.



Balanced, Complete, Full, and Perfect