Introduction to Trees

Trees

- In COMP 2150 you learned about the linked list, which is a **linear** data structure
  - Each list node has only one "previous" node and one "next" node
- A **tree** is a **nonlinear** data structure
  - Each tree node has only one "previous" node, but there is no restriction on the number of "next" nodes
- Many applications of trees in computer science!
  - File system on your computer
  - Parse trees to make sense of grammar (useful in compilers)
  - Decision trees to implement game AI
  - Any application that requires hierarchical organization of data

Tree terminology

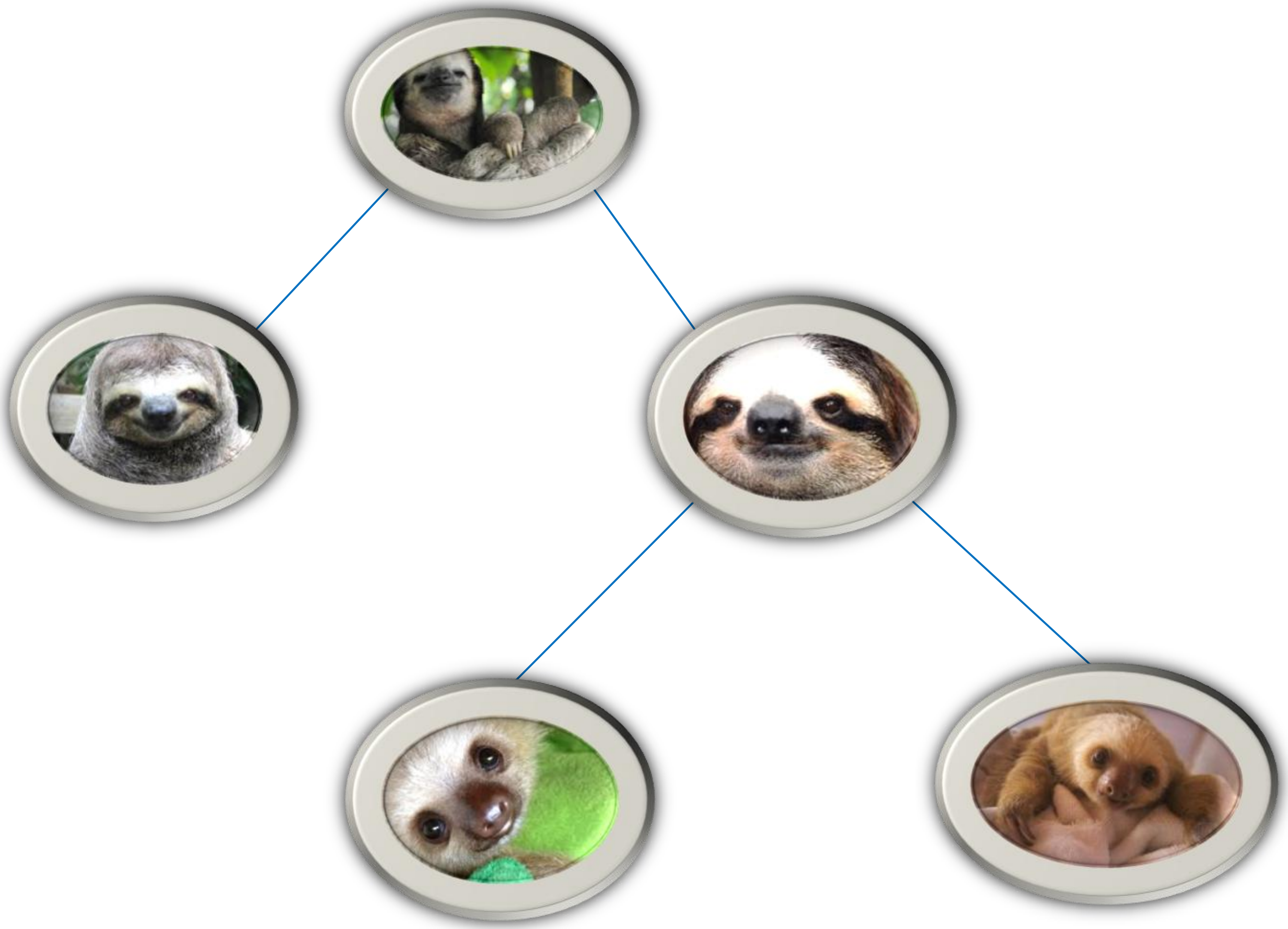
- The **root** of a tree is the highest node in the tree; the root has no "previous" node
- Each node's "next" nodes are known as the **children** of that node (which is the **parent** of said children)
- A **leaf** is a node with no children
- A **subtree** of a node is a tree whose root is a child of that node
- The **level** or **depth** of a node is how far that node is from the root (the root itself has a level of 1)
- The **height** of a tree is the maximum level of its nodes (the tree pictured here has a height of 3)

Here is a nice tree of sloths!

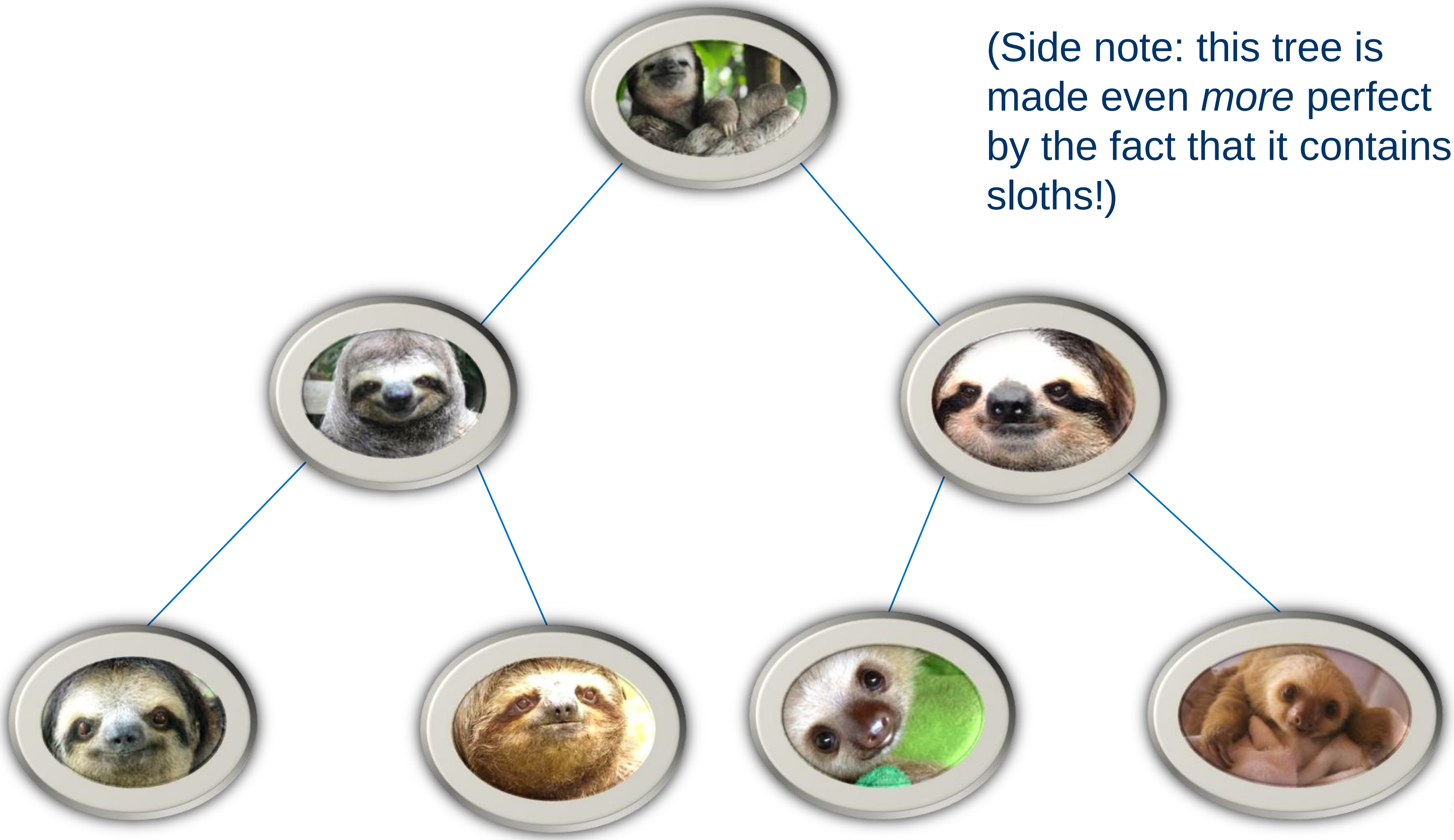
Binary trees

- In a **binary tree**, each node can have no more than 2 children (known as the **left child** and the **right child**)
- Some special types of binary trees:
  - A **full binary tree** means that all nodes have 0 or 2 children
  - A **perfect binary tree** is a binary tree containing *n* levels that has exactly $2^n - 1$ nodes (informally, all the tree's "rows" are completely filled)
  - A **complete binary tree** is a binary tree containing *n* levels that is a perfect tree up to level $n - 1$, with the nodes in level *n* all placed toward the left (informally, all the tree's "rows" are completely filled except the last row)

Full binary tree

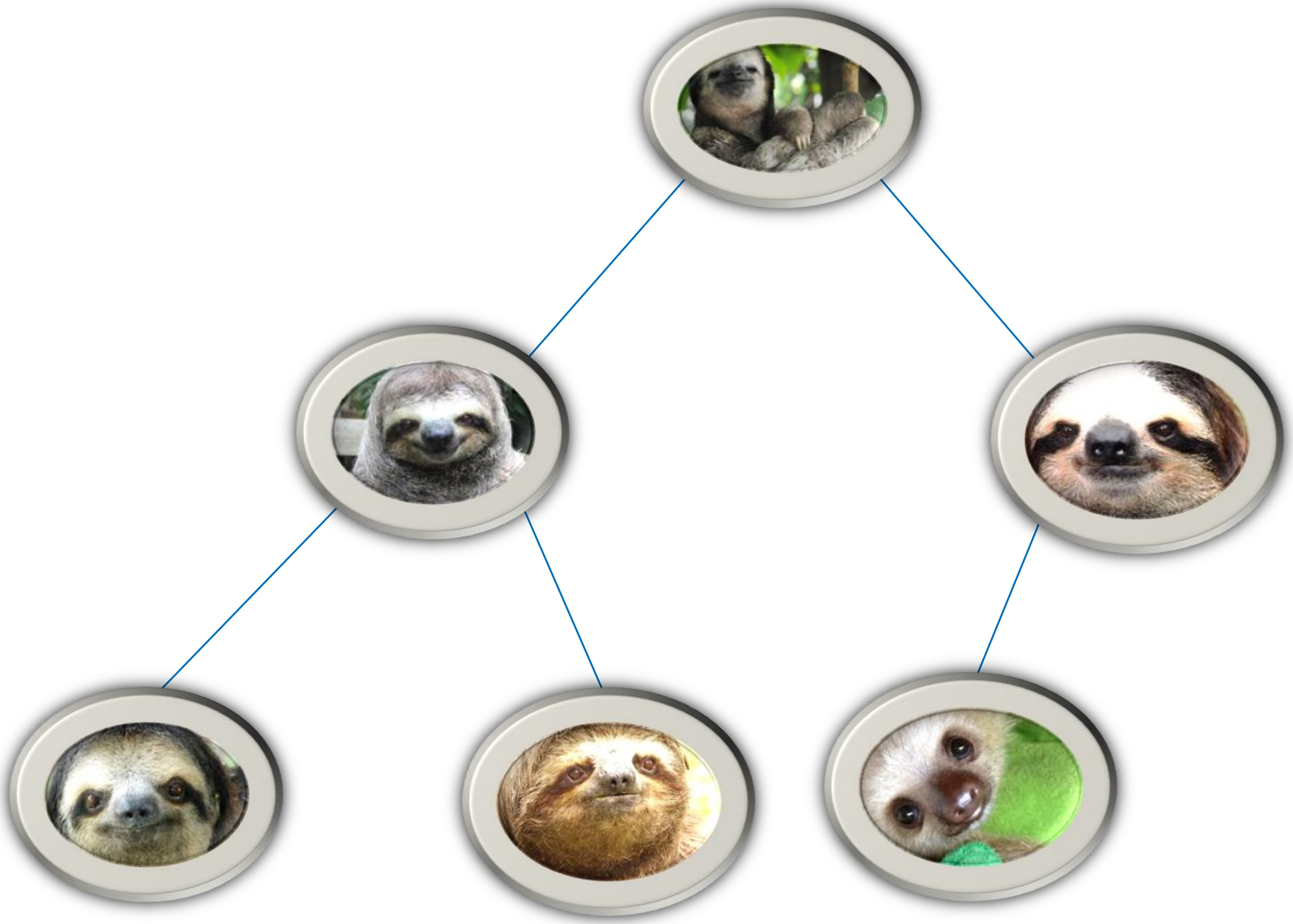


Perfect binary tree



(Side note: this tree is made even *more* perfect by the fact that it contains sloths!)

Complete binary tree

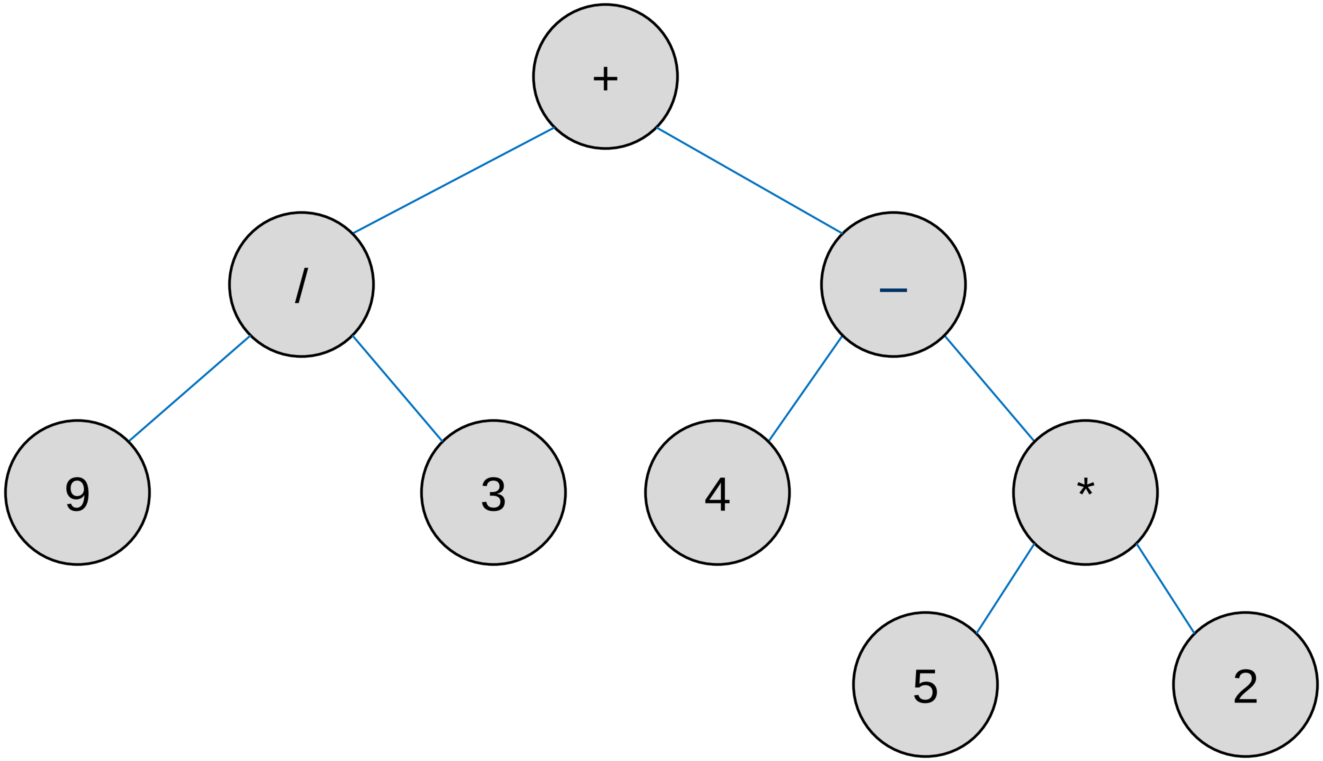


Expression trees

- One nice application of binary trees is to organize arithmetic expressions
  - The leaf nodes in an expression tree are **operands** (numbers/quantities)
  - All other nodes are **operators** indicating how its child nodes should be combined

Expression trees

- <u>Example:</u> Draw an expression tree for
  9 / 3 + (4 – 5 * 2)



Tree traversals

- **Traversing** a tree means to visit every node in the tree
- Unlike a linked list, there is more than one way to traverse a binary tree! (since each node has up to two children)
- Three common ways to traverse a binary tree:
  - **Pre-order traversal**
  - **In-order traversal**
  - **Post-order traversal**

Pre-order traversal

- To pre-order traverse a binary tree starting from node *n*:
  - Visit node *n*
  - Pre-order traverse *n*'s left subtree
  - Pre-order traverse *n*'s right subtree
- Looks awfully recursive, doesn't it?
  - I told you we weren't done with recursion for the semester ☺

In-order traversal

- To in-order traverse a binary tree starting from node *n*:
  - In-order traverse *n*'s left subtree
  - Visit node *n*
  - In-order traverse *n*'s right subtree

Post-order traversal

- To post-order traverse a binary tree starting from node *n*:
  - Post-order traverse *n*'s left subtree
  - Post-order traverse *n*'s right subtree
  - Visit node *n*

Visualizing traversals

- Draw an **Euler tour** of the tree (this is hard to do in PowerPoint, so I'll just refer you to p. 305 of your textbook ☺)
- As you perform the Euler tour, keep track of the first, second, and third time that you encounter each node
  - Pre-order traversal: visit each node the first time you encounter it on the Euler tour
  - In-order traversal: visit each node the second time you encounter it on the Euler tour
  - Post-order traversal: visit each node the third time you encounter it on the Euler tour

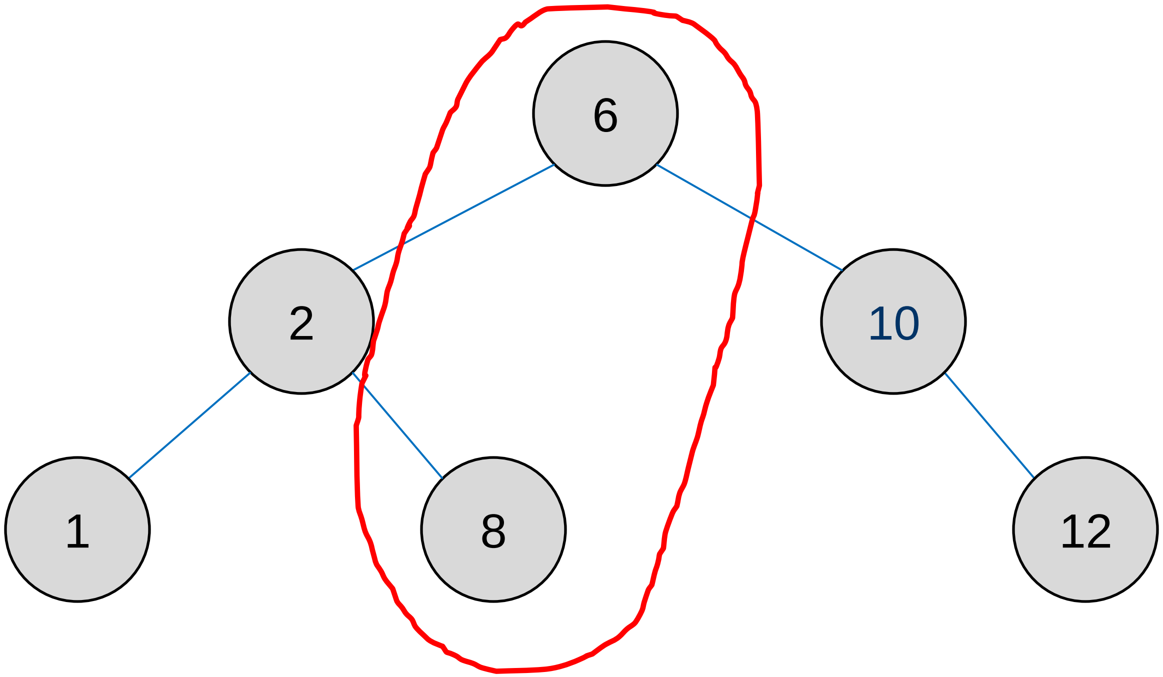
Traversing expression trees

- A pre-order traversal results in the expression in **prefix notation** (operators come before operands)
- An in-order traversal results in the expression in **infix notation** (our usual way of writing arithmetic)
- A post-order traversal results in the expression in **postfix notation** (operators come after operands)
- For the expression tree shown earlier in these notes:
  - Pre-order traversal: **+ / 9 3 – 4 * 5 2**
  - In-order traversal: **9 / 3 + 4 – 5 * 2**
  - Post-order traversal: **9 3 / 4 5 2 * – +**

Binary search trees

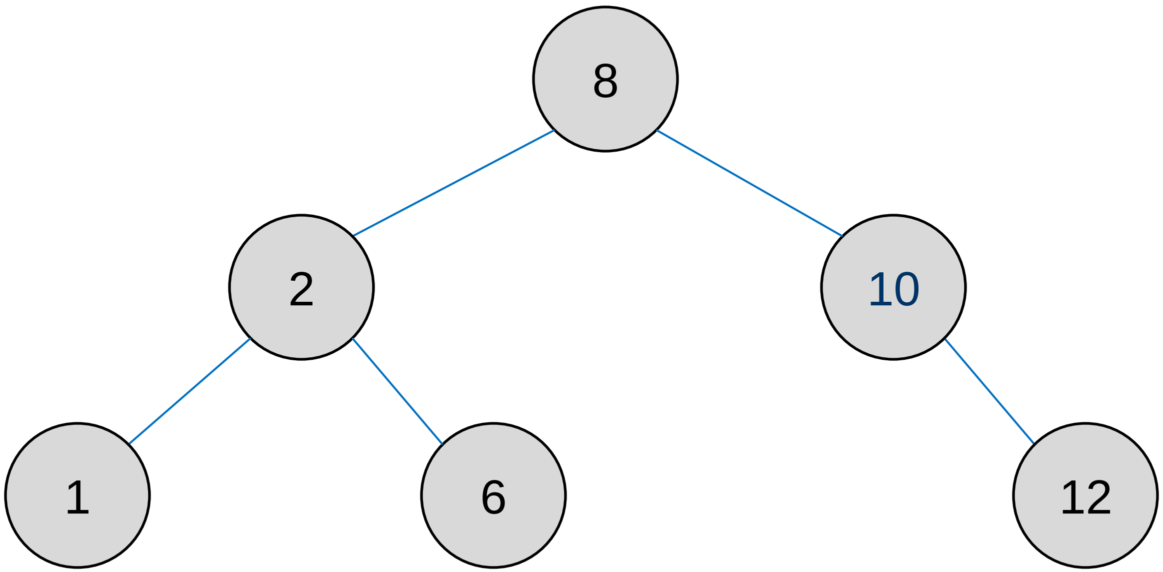
- A **binary search tree** (**BST**) is a special case of a binary tree
- A BST does not have to be full, complete, or perfect. It just has to satisfy this property: Each node in a BST must be greater than all nodes in its left subtree, and less than all nodes in its right subtree
- Note that this is NOT the same as saying that each node must be greater than its left child and less than its right child (that's a much weaker statement)

Binary search trees



All nodes satisfy the property that the node is greater than its left child and less than its right child. However, this is NOT a BST because the 8 is part of 6's left subtree, and 6 is not greater than 8.

Binary search trees



This IS a BST – each node is greater than all elements in its left subtree, and less than all elements in its right subtree.

Traversals with BSTs

- Work the same way as traversals of any binary tree
- For the BST on the previous slide:
  - Pre-order traversal: **8 2 1 6 10 12**
  - In-order traversal: **1 2 6 8 10 12**
  - Post-order traversal: **1 6 2 12 10 8**
- Note that an in-order traversal of a BST will produce the elements of the tree ordered from least to greatest