Trees
- Definition: A tree $T$ is a set of nodes storing elements, and edges. An edge is incident on two nodes (i.e., it connects two nodes).
- Each edge is incident on a parent node and a child node. It is directed, from the parent to the child.
- If $T$ is nonempty, it has a special node, called the root, that has no parent.
- Each node $n$ of $T$ different from the root has a unique parent node $p$; every node with parent $p$ is a child of $p$.
- A tree is ordered if there is a linear ordering defined for the children of each node (i.e., a node has a first child, a second child, etc.)

Other tree definitions
- A path is a sequence of nodes such that any two consecutive nodes form an edge.
- The depth (or level) of a node $n$ is the length of the path from the root to $n$.
- An interior node (or internal node) is a node with at least 1 child.
- A leaf (or external node) is a node with no children.
- The height of node $n$ is the path length from $n$ to the deepest leaf under $n$ (height of a leaf = 0).
  - The height of a tree is the height of its root.
  - Sometimes called the depth of the tree.
  - The height of an empty tree is defined to be -1.
- Siblings: 2 nodes with the same parent.
- Ancestor: any node on the path between a node and the root (sometimes includes the node itself).
- Descendant: opposite of ancestor.
- Subtree: The subtree of $T$ rooted at the node $x$ contains $x$ and its descendants, and all edges in $T$ incident on these nodes.
- A binary tree is an ordered tree in which no node has more than 2 children.
- We refer to the children as the left child and the right child.

Other definitions associated with binary trees
- A binary tree is proper if each node has 0 children or 2 children (aka a full binary tree).
- Example: tree representing an arithmetic calculation is proper (assuming no unary - or +)

\[(3 + 1) \times 3) / ((9 - 5) + 2)\]
A binary tree is **complete** if it contains the maximum number of nodes for a given height:

- What is the height of a complete binary tree with \( N \) nodes? – **Theta(\( \log n \))**

**Binary search trees**

- A Binary Search Tree (BST) \( T \) is a binary tree such that for all nodes \( n \) in \( T \):
  a. If \( n \) has a left child \( l \), then for all nodes \( s \) in the subtree rooted by \( l \), \( \text{Data}(s) \leq \text{Data}(n) \) (as specified by `compareTo`)
  b. If \( n \) has a right child \( r \), then for all nodes \( s \) in the subtree rooted by \( r \), \( \text{Data}(s) \geq \text{Data}(n) \)
- Compared to lists, BSTs will enable us (eventually) to improve the `contains` operation while keeping `add` and `remove` less than linear
- Are these binary search trees?

![Binary Search Tree Diagram](image)

**java.util.Set<E>**

- Collection of data with no duplicates
- Intent: store the collection either in no particular order, or in some "natural" order (ascending, alphabetical, ...)
- Implementations: `TreeSet` (a BST) and `HashSet` (we'll study these later in the quarter)
- Some of their methods:

<table>
<thead>
<tr>
<th>method</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean add(E e)</td>
<td>Adds the specified element to this set if it is not already present.</td>
</tr>
<tr>
<td>boolean contains(Object o)</td>
<td>Returns true if this set contains the specified element.</td>
</tr>
<tr>
<td>boolean equals(Object o)</td>
<td>Compares the specified object with this set for equality.</td>
</tr>
<tr>
<td>boolean isEmpty()</td>
<td>Returns true if this set contains no elements.</td>
</tr>
<tr>
<td>Iterator iterator()</td>
<td>Returns an iterator over the elements in this set.</td>
</tr>
</tbody>
</table>
boolean remove(Object o)
Removes the specified element from this set if it is present.

int size()
Returns the number of elements in this set (its cardinality)

- **Searching a binary search tree**

  Does tree $T$ contain a data item?

  ```
  node = root
  while node != null
    if item equals data(node)
      return true
    else if item less than data(node)
      node = leftChild(node)
    else node = rightChild(node)
  return false
  ```

- **Inserting into a binary search tree**

  - When inserting, always insert a new leaf
  - Example: insert 5 into the first tree above

    ```
    o More examples: insert 1, 3, 11, 10, 12
    ```
Deleting from a binary search tree

- Must preserve the properties listed above
- Easy when removing a leaf node, or an interior node with 1 child
- Removing an interior node with 2 children:
  0. Find leftmost node in right subtree
  1. Will have at most one child
  2. Promote it to replace the deleted node
  3. Promote its child to replace where it was
- Example: delete the 2, 6, 13, 8, and 9, from the tree below in sequence
- **Implementation**: See Homework 2 code
- **Complexity of Binary Search Tree operations**

<table>
<thead>
<tr>
<th>Operation</th>
<th>Worst-case Complexity</th>
<th>Best-case Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>$\Theta(n)$</td>
<td>$\Theta(\log n)$</td>
</tr>
<tr>
<td>contains</td>
<td>$\Theta(n)$</td>
<td>$\Theta(\log n)$</td>
</tr>
<tr>
<td>remove</td>
<td>$\Theta(n)$</td>
<td>$\Theta(\log n)$</td>
</tr>
</tbody>
</table>