Today: Binary Trees

* Each node has _at most_ two children.

```
  A
 / \
B   C
 |   |
D   E
```

pre-order: A B C D E F G  
(recursively: node first, then children)
post-order: B E G F D C A  
(recursively: children first, then node last)

for binary trees only: in-order  
recursively: first left child, then node, then right
B A E D F G C

```
  data
/     |
left   right
?      ?
```

Efficiency of `__contains__`? Running time, as a function of the size of the tree: \( n = \text{number of nodes in the entire tree} \)

\( O(n) \): `__contains__` gets called once for each node.

Better than linked list? no...


Using a built-in list, can we do better?  
IF the list is sorted, we can use binary search.
Exercise: write code for this!

Efficiency? how many elements must be examined, in worst-case?
step 0: n elements
step 1: n/2 elements (roughly)
step 2: n/4 elements (roughly)
step 3: n/8 elements (roughly)
...
step k: n/(2^k) elements (roughly)

When does this stop? When \( n/2^k \leq 1 \)

\[
\frac{n}{2^k} \leq 1 \quad \text{solve for } k: \quad n \leq 2^k \\
\log_2 n \leq \log_2 (2^k): \quad k \geq \log_2 n
\]

binary search takes time \( O(\log n) \), in worst-case
possible only because the list is _sorted_

For trees: what if we kept values in the tree "sorted"?
Binary Search Trees (BST):

* A binary tree (each node has at most two children)
* Restrictions:
  - All values are unique (no repeated value allowed) [not strictly necessary]
  - Values must be comparable to each other
  - Values must be ordered following the "BST ordering":

for every node in the BST:
- every value in the left subtree is strictly smaller than value in the node;
- every value in the right subtree is strictly greater than value in the node.

inorder traversal:
(everything in left subtree), root, (everything in right subtree)