* Project Part I: marks to be released this week
* Project Part II: to be posted (partly) this week

Binary Trees
* tree where each node has at most two children

```
B
  D
G
  A
  E
  F
C
```

preorder (recursive, node before children):  
B D G A E F C

postorder (recursive, node after children):  
D E C F A G B

inorder (recursive, node in between children;  
makes sense only for binary trees, where children  
have a "side" -- left or right):  
D B E A F C G

```
recursive code working with  
BTNodes:  
    item:  
    self:  
        data  
        left  
        right  

?  
?  
```

Running time of __contains__? As a function of n = number of nodes in  
the tree? In the worst-case?  
O(n^2)? O(n)? O(log n)?

Proportional to the size of the tree.  
How does this compare to storing values in a list?
* Search for a value in a list?
  Efficiency depends on one factor: is the list sorted or not.
* Unsorted list: searching also takes $O(n)$.  
* Sorted list: we can do better!

**binary search:** eliminate half the list by comparing with middle element until the list is down to 1 element
step 0: $n$ elements
step 1: $n/2$ elements (roughly)  \[
\frac{n}{2^k} \leq 1 \iff n \leq 2^k
\]
step 2: $n/4$ elements (roughly)  \[
\iff \log_2 n \leq k
\]
step 3: $n/8$ elements ...
step $k$: $n/2^k$ elements

What's $k$? When does $n / 2^k \leq 1$?

So running time of binary search = $O(\log n)$.

EXERCISE: write binary search!
Binary Search Trees (BST):
* binary tree with additional restrictions:
  - no repeated value [not strictly necessary but makes things easier]
  - values can all be compared to one another
  - BST ordering for values:

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