Goals

Know the purpose and operations of the Set Abstract Data Type.

Know the motivation for and the definition of a binary search tree.

Be able to execute on paper, and be prepared to implement the search and add operations on a BST.
The Set ADT

/** A collection that contains no duplicate elements. */
interface Set {
    /** Return true if the set contains ob */
    boolean contains(Object ob);

    /** Add ob to the set; return true iff the collection changed. */
    boolean add(Object ob);

    /** Remove ob from the set; return true iff the collection is changed. */
    boolean remove(Object ob);
    ...
}
# Set ADT: Possible Implementations

<table>
<thead>
<tr>
<th></th>
<th>contains</th>
<th>add</th>
<th>remove</th>
</tr>
</thead>
<tbody>
<tr>
<td>LinkedList</td>
<td>O(n)</td>
<td>O(n)</td>
<td>O(n)</td>
</tr>
<tr>
<td>Array (sorted)</td>
<td>O(log n)</td>
<td>O(n)</td>
<td>O(n)</td>
</tr>
<tr>
<td>Array (unsorted)</td>
<td>O(n)</td>
<td>O(n)</td>
<td>O(n)</td>
</tr>
<tr>
<td>Tree?</td>
<td>O(n)</td>
<td>??</td>
<td>??</td>
</tr>
</tbody>
</table>
Searching a Binary Tree

A binary tree is

- Empty, or
- Three things:
  - value
  - a left binary tree
  - a right binary tree

Find v in a binary tree:
```java
boolean findVal(Tree t, int v):
    if t == null:
        return false
    if t.value == v:
        return true
    return findVal(t.left) || findVal(t.right)
```
(base case - not found!)
if t == null:
    return false
(base case - is this v?)
if t.value == v:
    return true
(recursive call - is v in left?)
return findVal(t.left)
    || findVal(t.right)
(recursive call - is v in right?)
An opportunity

- `contains` is $O(n)$ because we have to search every node.
- Can we somehow avoid that?
public class Tree {
    int value;
    Tree parent;
    Tree left;
    Tree right;
}

aside: sometimes it's helpful to keep a pointer to your parent
/** BST: a binary tree, in which:
 *  - all values in left are < value
 *  - all values in right are > value
 *  - left and right are BSTs */

public class BST {
    int value;
    BST parent;
    BST left;
    BST right;
}
/** BST: a binary tree, in which:
* - all values in left are < value
* - all values in right are > value
* - left and right are BSTs */

public class BST {
    int value;
    BST parent;
    BST left;
    BST right;
}

consequence: no duplicates!
(but not coincidence)
Binary Search Tree: Example

<5
a BST

<5
a BST

>5
a BST
Searching a BST

search(t, 11)

11 > 10

search(right, 11)
Searching a BST

search(t, 11)

\[ t: \quad 10 \]

11 > 10

search(right, 11)

11 < 16

search(left, 11)
Searching a BST

```
search(t, 11)
t: 10

11 > 10
search(right, 11)
11 < 16
search(left, 11)
11 == 11
found it! return.
```
Searching a BST - the nonexistent case

$\text{search}(t, 5)$

$t$: 10

5 < 10

$\text{search(left, 5)}$
Searching a BST - the nonexistent case

\[ \text{search}(t, 5) \]

\[ t: \quad 10 \]

\[ \begin{align*}
5 < 10 \\
\text{search(left, 5)} \\
5 < 8 \\
\text{search(left, 5)}
\end{align*} \]
Searching a BST - the nonexistent case

search(t, 5)

\[
\begin{array}{c}
t: 10 \\
8 \\
4 \\
9 \\
11 \\
16 \\
17
\end{array}
\]

5 < 10
search(left, 5)
5 < 8
search(left, 5)
5 > 4
search(right, 5)
null - not found!
Inserting into a BST

```
  10
  /  
 8   16
 / \
4   9
   / \
  11 17
```
Inserting into a BST

insert(t, 11)

11 > 10

insert(right, 11)
Inserting into a BST

$\text{insert}(t, 11)$

$t$: 10

8 11 16

4 9 17

11 > 10
$\text{insert}(\text{right}, 11)$
11 < 16
$\text{insert}(\text{left}, 11)$
Inserting into a BST

```plaintext
insert(t, 11)

11 > 10
insert(right, 11)

11 < 16
insert(left, 11)

11 == 11
found it! no duplicates, allowed; nothing to do. return.
```
Inserting into a BST - the nonexistent case

\( \text{insert}(t, 5) \)

\( t: \begin{array}{c}
10 \\
8 \\
4 \\
9 \\
11 \\
16 \\
17
\end{array} \)

5 < 10

\( \text{insert}(\text{left}, 5) \)
Inserting into a BST - the nonexistent case

\[
\text{insert}(t, 5)
\]

\[t: \begin{array}{c}
10 \\
8 \\
4 \\
9 \\
11 \\
17 \\
\end{array}\]

5 < 10
\[
\text{insert}(\text{left}, 5)
\]

5 < 8
\[
\text{insert}(\text{left}, 5)
\]
Inserting into a BST - the nonexistent case

```
insert(t, 5)
```

```
t: 10
```

```
8
```
```
4
```
```
9
```
```
11
```
```
16
```
```
17
```
```
5
```

5 < 10
insert(left, 5)
5 < 8
insert(left, 5)
5 > 4
insert(right, 5)
null - not found. insert it here!