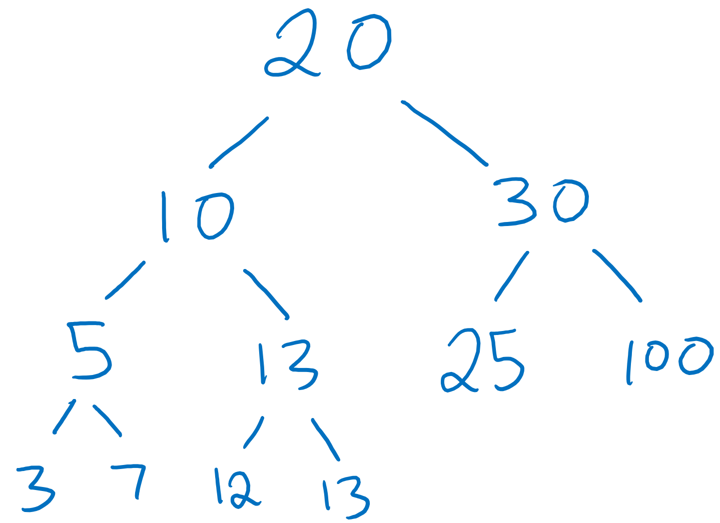
Binary Search Trees

CSC148, INTRODUCTION TO COMPUTER SCIENCE

DIANE HORTON, JONATHAN CALVER, MARYAM MAJEDI & JAISIE SIN

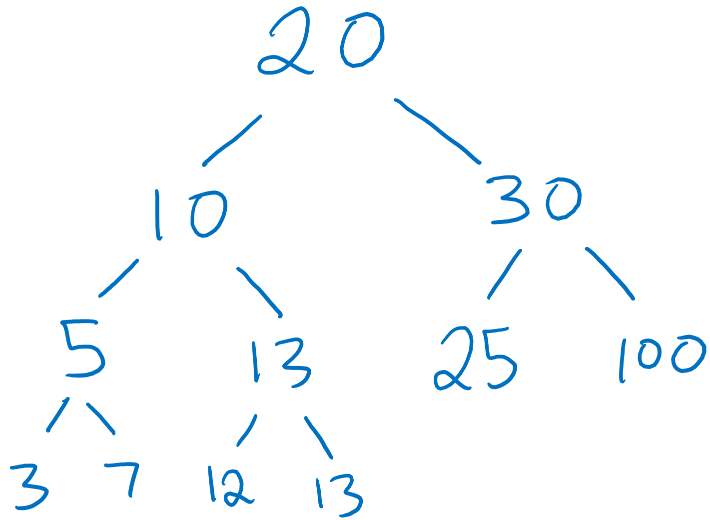
A Binary Search Tree is a “sorted” tree

Every item is
>= all items in its left subtree,
and
<= all items in its right subtree.



More structure → more efficiency

```
def __contains__(self, item) -> bool:
  if self.is_empty():
    return False
  elif item == self._root:
    return True
  elif item < self._root:
    return self._left.__contains__(item)
  else:
    return self._right.__contains__(item)
```



What recursion is required?

`maximum: Return the maximum number in this BST, or None if it’s empty.`

What recursion is required?

`items: Return all of the items in the BST in sorted order.`

More structure → more efficiency? *not always!*

```
def items(self) -> List:
  if self.is_empty():
    return []
  else:
    return (
          self._left.items() +
          [self.root] +
          self._right.items()
    )
```

What recursion is required?

```
count: Return the number of occurrences of <item>
in this BST.
```

What recursion is required?

```
smaller: Return all of the items in this BST
strictly smaller than <item>
```

Representation invariants are key!

`If self._root is not None, then self._left and self._right are BinarySearchTrees.`

If you know that a BST is not empty, you **never** need to check if `self._left` or `self._right` are `None`.

You can call methods on them without an if-statement “guard”.

BST efficiency

WHY SHOULD WE CARE ABOUT BINARY SEARCH TREES?

• base doesn't matter to big-Oh.

a.k.a. "bag"

The "Multiset" ADT (search, insert, delete)

For a **sorted list** with *n* items…

Python list

- search is fast: O(log *n*) worst case, because of binary search
- insert and delete can be slow, if inserting/removing from the *front* of the list – O(*n*) in the worst case

The Multiset ADT (search, insert, delete)

For a **general tree** with *n* items…

O(n)

```
for subtree in self.subtrees:
  if subtree.__contains__(item):
    return True
return False
```

The Multiset ADT (search, insert, delete)

For a **general tree** with *n* items…

- insert can be fast, if you insert as a child of the root – O(1)
- search and delete can be slow, since you might need to check every item in the tree – O(n) in the worst case

Worst case running times so far...

operation	Sorted List	Tree	Binary Search Tree
search	O(log *n*)	O(*n*)	
insert	O(*n*)	O(1)	
delete	O(*n*)	O(*n*)	

BST height vs. size

A binary search tree of size *n*…

- has a maximum height of n: $h \leq n$
- has a minimum height of (approximately) log *n*: $h \geq \log n$

We say that a BST is *balanced* if its left and right subtrees have roughly equal heights, and these subtrees are also balanced.

Balanced BSTs have height ≈ log *n*.