Trees

Binary Search Trees

Binary Search Trees

A binary search tree (BST) is a common version of a tree...

For every node in the tree:

- the values of all keys in its left subtree are less than the node's values
- the values of all keys in its right subtree are greater than the node's values

Searching for a specific value is fast

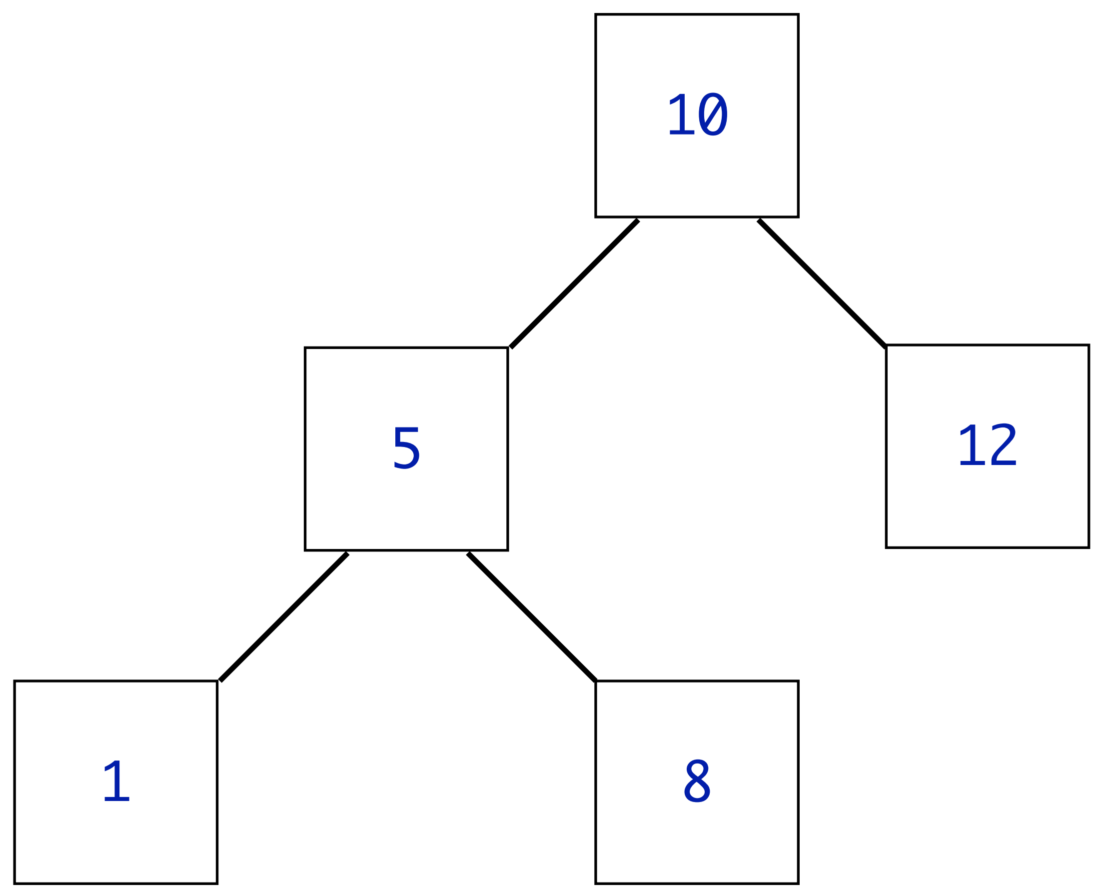
- worst-case complexity of log(h), where h is the maximum height of the tree
- this would be a great candidate for use with our bag class

Complexity of inserts and deletes suffer a bit

- worst-case complexity of log(h) instead of constant-time

Binary Search Trees

Example tree:

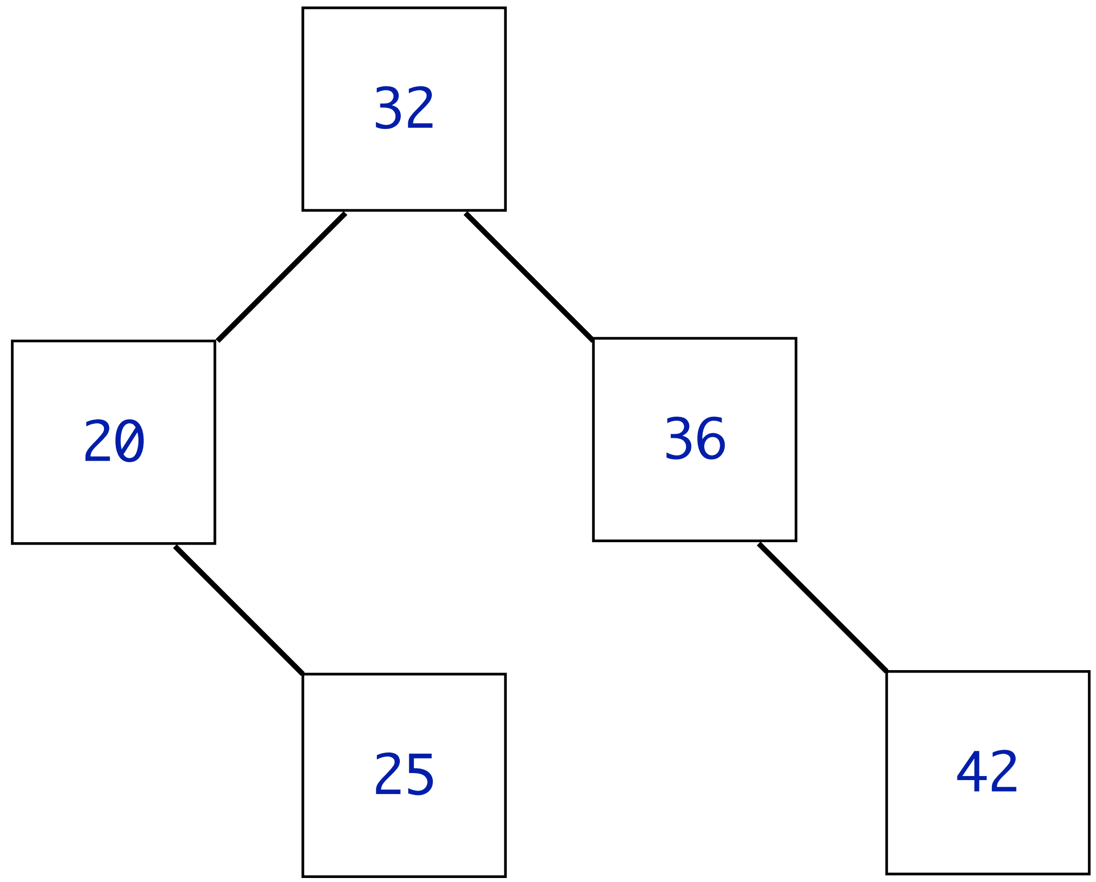


For every node in the tree:

- the values of all keys in its left subtree are less than the node's values
- the values of all keys in its right subtree are greater than the node's values

Binary Search Trees

Example tree:



For every node in the tree:

- the values of all keys in its left subtree are less than the node's values
- the values of all keys in its right subtree are greater than the node's values

Binary Search Trees

Insertion algorithm (recursive):

```
// if root is NULL
    // set root to a node containing the new value
// else if entry < root->data
    // insert value into left subtree
// else if entry > root->data
    // insert value into right subtree
```

What should you do with values that are <u>equal</u>?

- depends... can insert into left subtree consistently
- alternatively, can have each node maintain a count of the number of copies of their value
- inserting an existing item increments the count; removing a value with more than one copy simply decrements the count

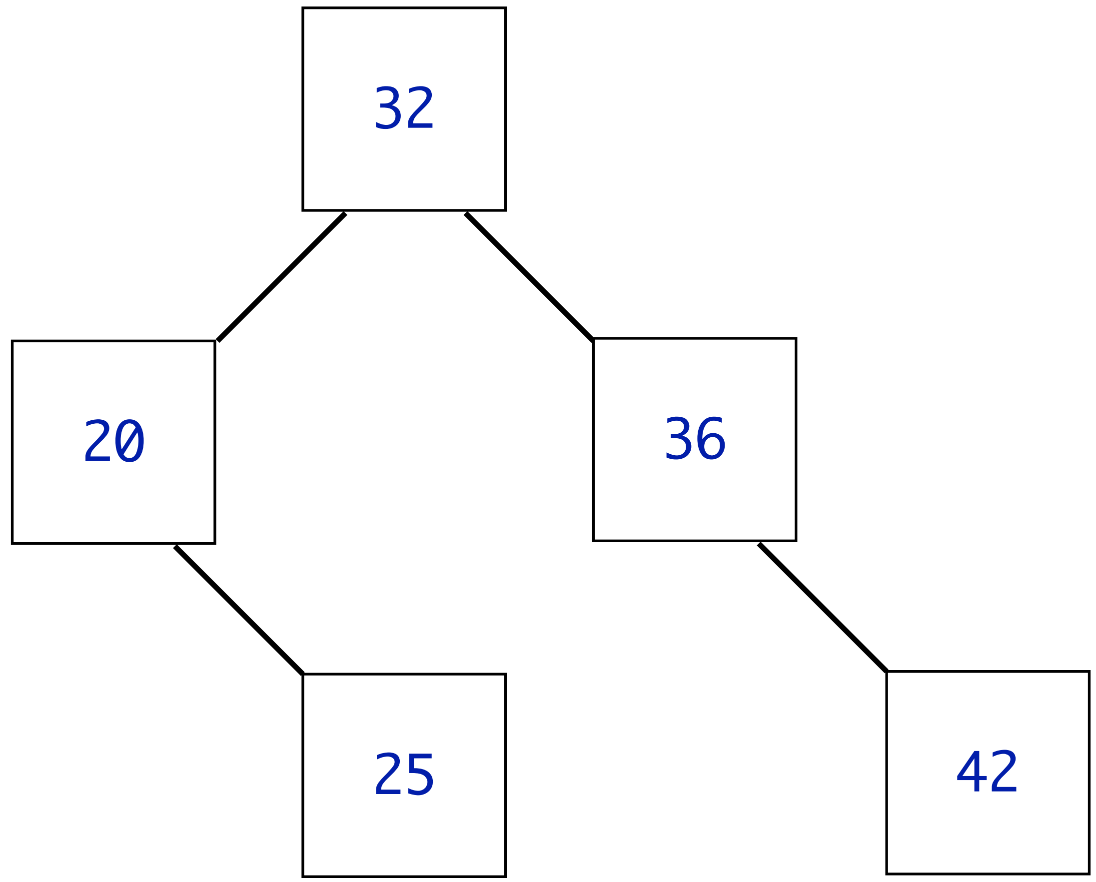
Binary Search Trees

Removal algorithm (recursive):

```
// if root is NULL
    // all done
// else if target < root->data
    // call delete with left pointer
// else if target > root->data
    // call delete with right pointer
// else when target == root->data
    // if no left child
        // delete root; make right child new root
    // else
        replace root with largest from left subtree
```

Binary Search Trees

Example tree:

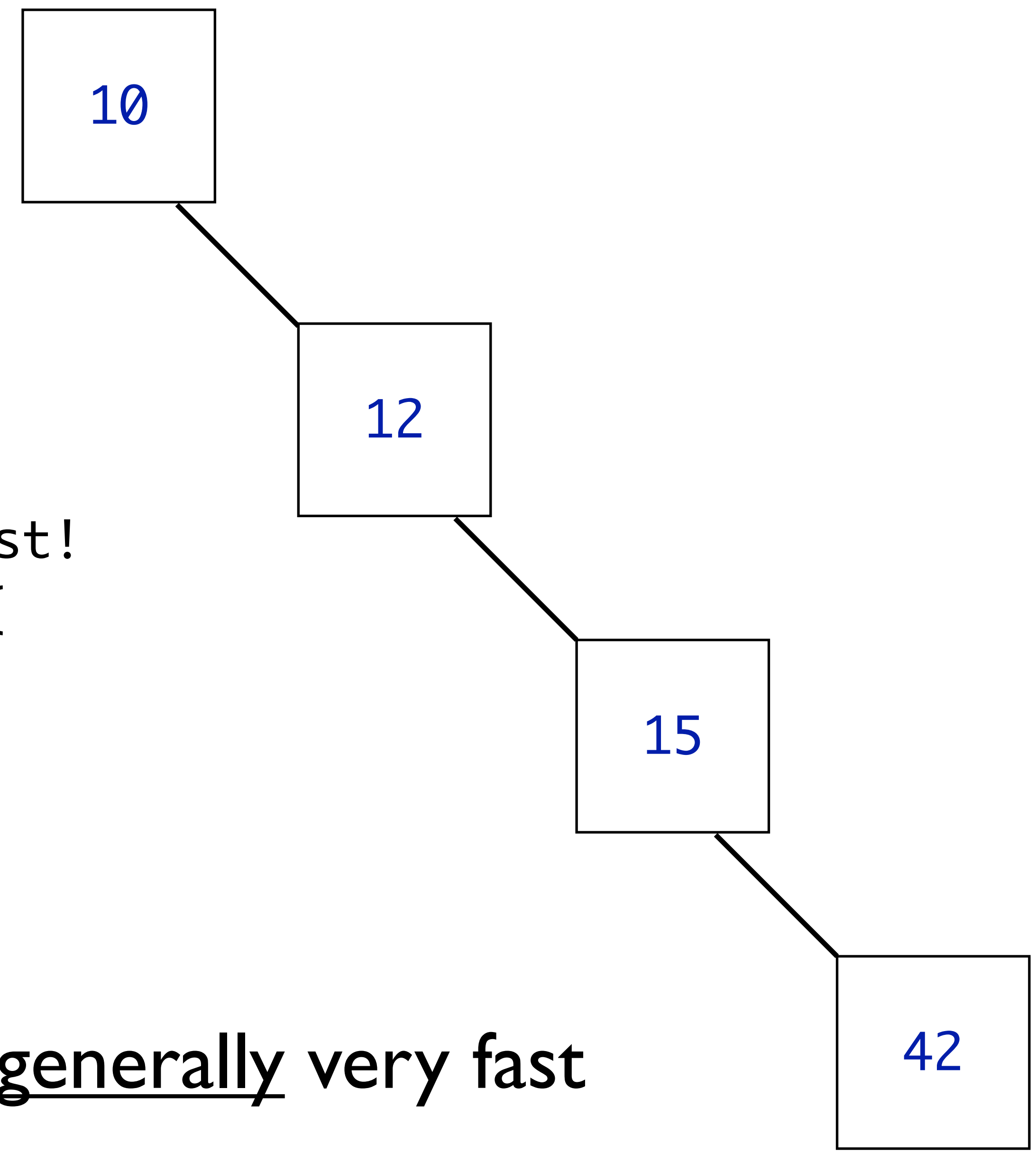


Searching for a specific value is generally very fast

- worst-case complexity of log(h), where h is the maximum height of the tree

Binary Search Trees

Example tree:




```
This is essentially a linked list!
  (searching in linear time) =(
```

Searching for a specific value is <u>generally</u> very fast

- worst-case complexity of log(h), where h is the maximum height of the tree
- but not always! common cases (inserting a sorted list) can yield trees where `h == n`

Binary Search Trees

Searching for a specific value is <u>generally</u> very fast

- we need a way to "balance" the tree on insertions and deletions, to guarantee that h ≈ log(n)
- this guarantees that insertion, deletion, and searching are O(log(h)) operations instead of O(n) operations

There are a number of ways this can be done

- result is called a "self-balancing" BST
- common to use "rotations" of nodes

Example implementations:

- AVL tree
- red-black tree
- yours (for extra credit on lab 10)   =)