Dictionary:
- *insert* (*Key x, Value v*)
- *delete* (*Key x*)
- *find* (*Key x*)
  - insert (*x, v*) in dict (No duplicates)
  - delete *x* from dict (Error if *x* not there)
  - returns a reference to associated value *v*, or null if not there.

Search:
- Given a set of *n* entries each associated with key *x*:
  - Store for quick access & updates
  - **Ordered**: Assume that keys are totally ordered: <, >, ==

Efficiency:
- **Balanced**: $O(\log n)$
- **Unbalanced**: $O(n)$

Example:
- *find*(5)
- *find*(14)

Can we achieve $O(\log n)$ time for all ops? 

**Binary Search Trees**
- Basic definitions
- Finding keys

Value *find*(Key *x*, BSTNode *p*)
- if (*p* == null) return null
- else if (*x* < *p*.key)
  - *find* (*x*, *p*.left)
- else if (*x* > *p*.key)
  - *find* (*x*, *p*.right)
- else return *p*.value

Idea: Store entries in binary tree sorted (inorder traversal) by key
- Start at root *p* = root
- if (*x* < *p*.key) search left
- if (*x* > *p*.key) search right
- if (*x* == *p*.key) found it!
- if (*p* == null) not there!
Insert (Key x, Value v)
- find x in tree
- if found ⇒ error! duplicate key
- else: create new node where we "fell out"

Replacement Node?

Binary Search Trees II
- insertion
- deletion

Delete (Key x)
- find x
- if not found ⇒ error
- else: remove this node & restore BST structure

3 cases:
1. x is a leaf
2. x has single child
3. x has two children

Find replacement node
copy to x, and then delete x

why did we do:
p.left = insert(x, v, p.left)?
If p == null, error! Key not found.
else if (x < p.key) p.left = delete(x, p.left)
else if (x > p.key) p.right = delete(x, p.right)
else if (either p.left or p.right null) if (p.left == null) p.key = x
return p.right
if (p.right == null) return p.left
else r = findReplacement(p)
copy r's contents to p
p.right = delete(r.key, p.right)
return p

**Find Replacement Node**

```java
BSTNode findReplacement(BSTNode p) { BSTNode r = p.right
while (r.left != null) r = r.left
return r }
```

**Binary Search Trees III**

- Deletion
- Analysis
- Java

**Java Implementation:**
- Parameterize Key + Value types: extends Comparable
  class BinSearchTree<K,V> {
- BSTNode - inner class
- Private data: BSTNode root
- insert, delete, find: local
- provide public fns
  insert, delete, find

But height can vary from O(log n) to O(n)... Expected case is good

Thm: If n keys are inserted in random order, expected height is O(log n).

**Analysis:**
All operations (find, insert, delete) run in O(h) time, where h = tree's height.
Java implementation (see latex notes for details)

```java
public class Bstree<Key extends Comparable, Value> {

    class Node {
        Key key;
        Value value;
        Node left, right;
        // ... constructor, toString...
    }

    private Node root;

    public Value find(Key x) {...}
    public void insert(Key x, Value v) {...}
    public void delete(Key x) {...}
}
```

Inner class for node (protected)

Local helper (private or protected)

Data (private)

Public members (invoke helpers)
Extra: Symmetrical replacement in deletion
- We set the convention of taking replacement node from right subtree
- Why not left? Does it matter?

→ Over a long series of random inserts + deletes, tree becomes unbalanced due to replacement bias
→ Height tends to $O(\sqrt{n})$ [$\sqrt{n} > \log n$]

→ Fix?
  - Ignore - it takes a long time
  - Remove bias - Randomly take replacement from left/right
  - Use a better tree (AVL, red-black)