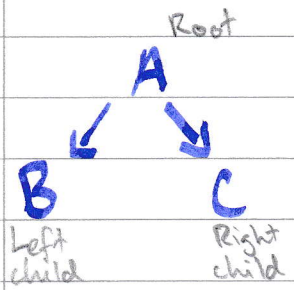
Number of searches = $\log_2$(size of tree)

e.g. size = 7

$\log_2 7 = 2.81$ ($2^{2 \text{ or } 3}$)

Searches = 2 - 3 searches

Trees



Binary tree - a tree that have nodes with up to 2 childs

Binary Search tree - ^, ordered
Everything to the LEFT ≤ root
Everything to the RIGHT > root

Every node has 0, 1 or 2 children

Definitions - Full - a tree with no missing nodes. All leaves on the same level

find max size $2^{(h+1)} - 1$

$\log_2 n = 2^n$

Balanced - height on left and right on every node ≤ 1

Subtree - tree which a child of a node

Complete - No gaps on bottom

Height - length of path from node to longest leaf

Height of tree == Height of root

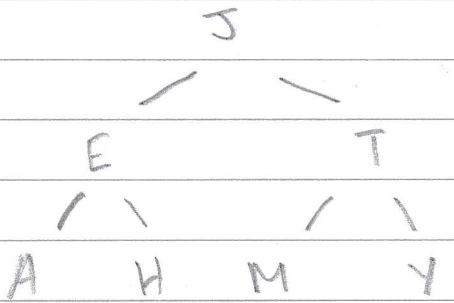
Leaf - node with no childs

Traversal

Pre-Order: Print, Left, Right
root at start
(J), E, A, H, T, M, Y

In-Order: Left, Print, Right
A, E, H, (J), M, T, Y

Post-Order: Left, Right, Print
root at end
A, H, E, M, Y, T, (J)



Writing the code to recursively insert a new item into a binary search tree

Pseudocode Algorithm (called by public helper method)
If tNode == null, return address of new node created
else if new item is smaller (compareTo < 0)
    recursively insert to the left
else if new item is larger (compareTo > 0)
    recursively insert to the right

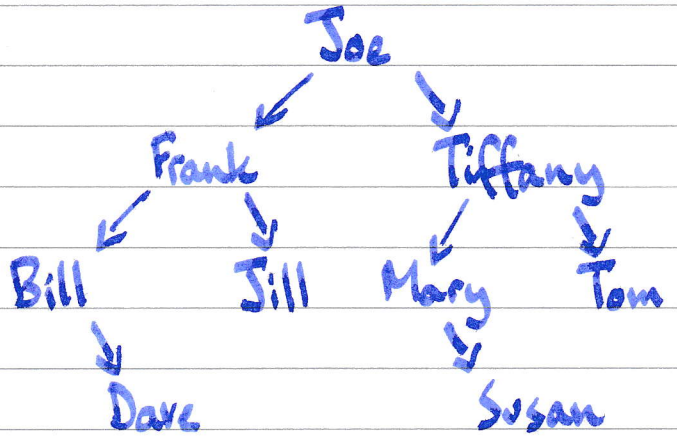
Note - this would never be a public instance method because it uses recursion with the root (which kept private)

Like stacks and queues, trees may also be built as an array

Array		Left	Right
0	Joe	2	1
1	Tiffany	3	6
2	Frank	5	4
3	Mary	-1	7
4	Jim	-1	-1
5	Bill	-1	8
6	Tom	-1	-1
7	Susan	-1	-1
8	Dave	-1	-1

Array of nodes, each slot has 3 items:
1) Data field
2) Left child index array
3) Right child index array

} integer position in array



Root is in Slot 0
Null link is Slot -1

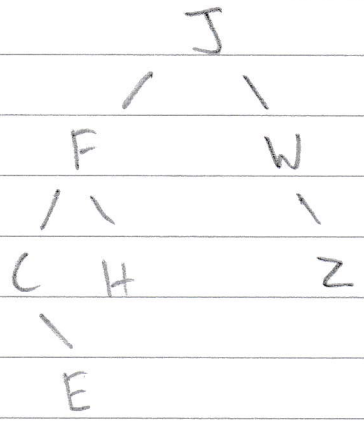
Left, right, print

Post-order traversal from BST

E, C, H, F, Z, W, J root at end (write out the alphabet)

left left left left right right print
left left right print right print
right print print
print

Z is on right of W



Array representation

	Data	LChild	RChild
0	J	2	3
1	E	-1	-1
2	F	4	6
3	W	-1	5
4	C	-1	1
5	Z	-1	-1
6	H	-1	-1

Recursion method

```
protected void traversal(TreeNode tNode) {
        if (tNode != null) {
    preorder - System.get.println(tNode.getItem())
    left-child - traversal(tNode.getLeft());
    right-child - traversal(tNode.getRight());
        }
}
```

If a complete tree has 135 nodes, what is its height?

Full tree size $= 2^{(h+1)} - 1$

$2^5 = 32$
$2^6 = 64$
$2^7 = 128$
$2^8 = 256$

At $h=7$, $2^{(7+1)} - 1 = 255$ nodes max
So bottom row has 8 nodes, 255 - 135 = 120
120 gaps

How many leaves does a tree with 135 nodes have? 68 leaves
Max height = 134, Min leaf == 1 } Degenerate tree, linked list

A tree of any size, will always have "n+1" links
Number of leaves = $\frac{\text{height} + 1}{2} = \frac{135 + 1}{2} = 68$ Always round down

The TreeMap Class
(from the Java Collections Framework)

```
import java.util.*;

public class JCFTreeMap
{
        public static void main(String[] args)
        {
            TreeMap persons = new TreeMap();

            persons.put(new Integer(888888888), "Taylor, Fritz");
            persons.put(new Integer(222222222), "Johnson, Alan");
            persons.put(new Integer(123456789), "Bleaux, Joe");
            persons.put(new Integer(999999999), "Johnson, Alan");
            persons.put(new Integer(123456789), "Rogers, Tiffany");
            persons.put(new Integer(555555555), "Nguyen, Viet");
            persons.put(new Integer(666666666), "Jones, Adam");

            System.out.println(persons.containsKey(new Integer(888888888)));
            System.out.println(persons.containsValue("Rogers, Tiffany"));
            System.out.println("************************");

            Iterator itr = persons.entrySet().iterator();
            while (itr.hasNext())
                   System.out.println(itr.next());

            System.out.println("************************");

            itr = persons.values().iterator();
                while (itr.hasNext())
                        System.out.println(itr.next());
        }
}
```

built-in binary search tree that uses 2 keys for data elements. Orders by 1st key = primary key. 2nd key is secondary key that is also contained in node.

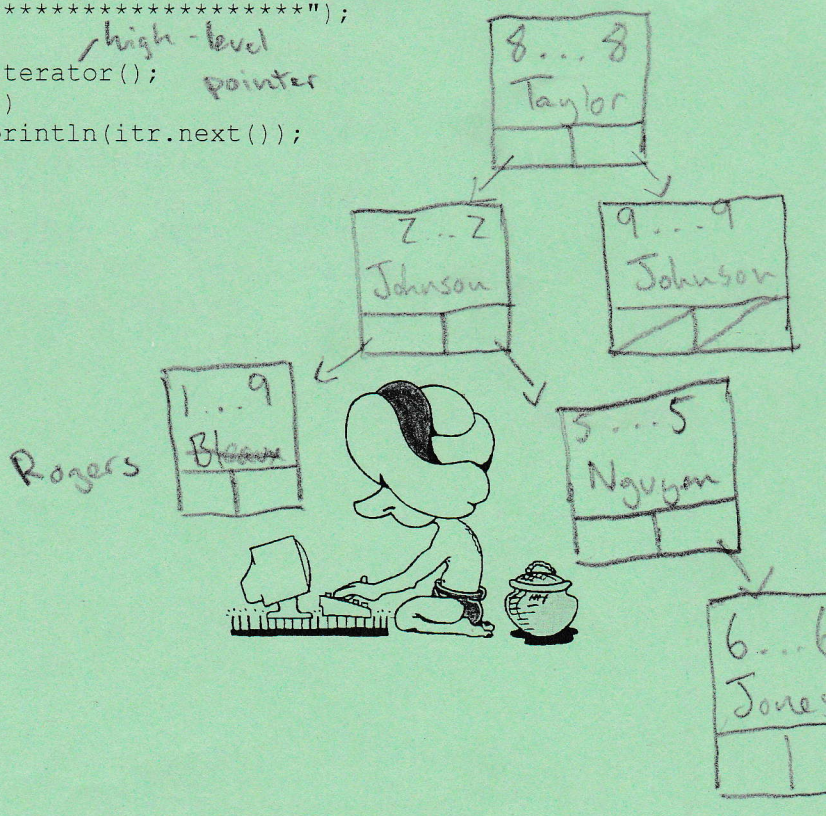
e.g. mercer id = primary key

name = secondary key

default inorder

Duplicate primary keys get overwritten

high-level pointer



Sample Run

```
true
true
************************
123456789=Rogers, Tiffany
222222222=Johnson, Alan
555555555=Nguyen, Viet
666666666=Jones, Adam
888888888=Taylor, Fritz
999999999=Johnson, Alan
************************
Rogers, Tiffany
Johnson, Alan
Nguyen, Viet
Jones, Adam
Taylor, Fritz
Johnson, Alan
```

The TreeSet Class
(from the Java Collections Framework)

built-in binary search tree to quickly store a collection of values. It is for single keys and it removes duplicates.

```
TreeSet myTree = new TreeSet();
myTree.add ("C");
myTree.add ("O");
myTree.add ("N");
myTree.add ("G");
myTree.add ("R");
myTree.add ("A");
myTree.add ("T");
myTree.add ("U");
myTree.add ("L");
myTree.add ("A");
myTree.add ("T");
myTree.add ("I");
myTree.add ("O");
myTree.add ("N");
myTree.add ("S");
myTree.remove ("C");
Iterator itr1 = myTree.iterator( );
itr1.next( );
itr1.next( );
itr1.next( );
System.out.println (itr1.next( ));
itr1.remove( );
itr1.next( );
System.out.println (itr1.next( ));
System.out.println("****************");
Iterator itr2 = myTree.iterator( );
while (itr2.hasNext())
        System.out.println(itr2.next());
```

OOB: out of balance

ignore duplicates

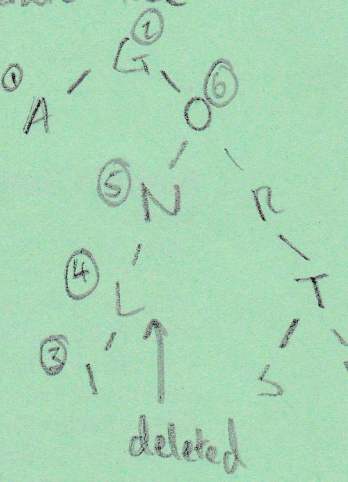
before 1st sorted item

Delete "C", bring up "A" or "G" Choose "G" because more balanced

remove

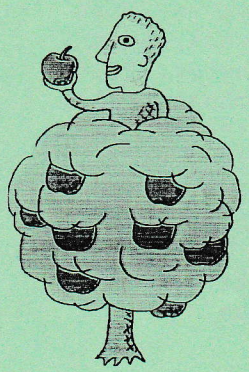
If not balanced, becomes degenerate tree

itr.next()



Sample Run

```
L
O
****************
A
G
I
N
O
R
S
T
U
```



Java has support for multi-tasking

An Intro to Multithreading in Java

```
public class ThreadTester
{
        public static void main(String args[])
        {
                PrintThread thread1, thread2, thread3, thread4;

                // creating four PrintThread objects
                thread1 = new PrintThread("thread1");
                thread2 = new PrintThread("thread2");
                thread3 = new PrintThread("thread3");
                thread4 = new PrintThread("thread4");

                System.err.println("\nStarting threads");
                thread1.start(); // triggers run method
                thread2.start();
                thread3.start();
                thread4.start();
                System.err.println("Threads started\n");
        }
}

class PrintThread extends Thread   -> built-in
{
        private int sleepTime;

        public PrintThread(String name)
        {
                super(name);
                sleepTime = (int) (Math.random() * 5000);   -> 0-5 seconds
                System.err.println("Name:  " + getName() +
                                   ";  sleep:  " + sleepTime);
        }                  use when inside exception

        // control thread's execution
        public void run()
        {
                try                    high priority unbuffered
                {                      (error)
                        System.err.println(getName() + " going to sleep" );

                        Thread.sleep(sleepTime);
                }                          pause in milliseconds

                catch (InterruptedException interruptedException)
                {
                        System.err.println(interruptedException.toString());
                }

                finally   -> always runs
                {
                        System.err.println(getName() + " done sleeping" );
                }
        }
}
```