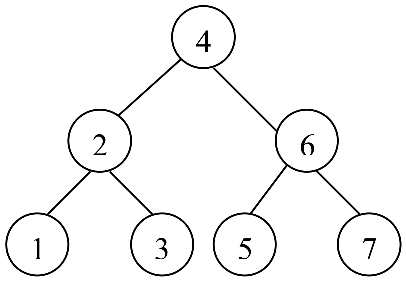
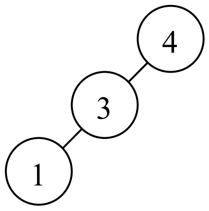
AP Computer Science Name ______________________________
Binary Search Trees

What is the best way to store and retrieve ordered (sorted) items? Using an array allows O(N) access and takes O(N) to keep them ordered. Using a linked list also allows O(N) access and O(N) to keep them ordered. Be able to explain why.

A Binary Search Tree allows O(log n) access. Recall that in a binary search tree, every node is greater than its left child and less than its right child. Every comparison thus throws away half the remaining data. We have seen that behavior before when we performed a binary search in an array. That property means BSTs can access any item in O(log n). Keeping the tree in order is also fast, because finding the right place is O(log n), followed by linking and unlinking some nodes.



One problem to keep in mind is that the BST's performance depends on the shape of the tree. Obviously, finding an item in a linear tree is O(N). If the data in a BST changes often, programmers may need to pause and to rebuild the tree so that it is balanced. Or they may use a self-balancing tree, such as an AVL tree, or a red-black tree.



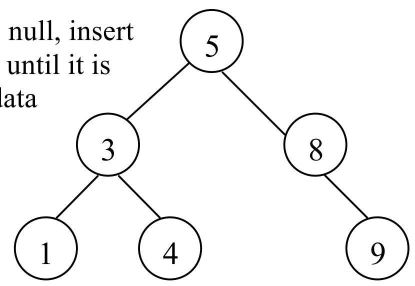
1. General algorithm for building a BST: Compare the root to the new item. If the item is less, go left, else go right. When you reach the end, insert the item, either left or right. The shape of each tree depends on the order that the data is given to you. Draw the BST formed by the letters in this string: COMPUTER

2. Build a BST from this data: CEMOPRTU

3. Build a BST from this data: OERCMPTU

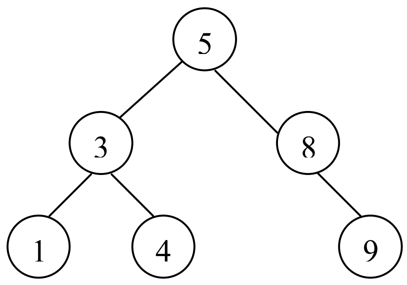
4. Looking at the trees above, and generalizing, what is the best case scenario for building a balanced binary tree? When the data is ______________________. In the best case, each of the n elements will require about log n comparisons, resulting in O(_________) time for building a BST. In the worst case, the case of building a linear tree, the resulting Big-O performance is O(______) time, because there are n elements, but each requires n comparisons.

5. Now let's write some code. Iterative algorithm to build a BST: if the root is null, insert the new node. Else, create two pointers, p and q. Advance the "front" pointer q until it is null, at which point the "second" pointer p points to the node at which the new data will be attached. A simple comparison tells whether the new node goes left or right. For example, insert a "2" in this tree. Then insert a "7".



```
public TreeNode insert(TreeNode t, Comparable item)
{









}
```

6. Recursive algorithm to build a BST: if the node is null, insert the new node. Else, if the item is less, set the left node and recur to the left. Else, if the item is greater, set the right node and recur to the right. For example, insert a "2" in this tree. Then insert a "7".



```
public TreeNode insert(TreeNode t, Comparable Item)
{









}
```

7. Find the target. Iterative algorithm: create a temporary pointer p at the root. While p is not null, if the p's value equals the target, return *true*. If the target is less than the p's value, go left, otherwise go right. If the target is not found, return *false*.

```
public boolean find(TreeNode t, Comparable item)
{









}
```

8. Find the target.   Recursive algorithm: If the tree is empty, return *false*. If the target is less than the current node value, return the left subtree. If the target is greater, return the right subtree. Otherwise, return *true*.

```
public boolean find(TreeNode t, Comparable target)
{









}
```

9. Starting at the root, return the min value in the BST.   Use iteration.   Hint: look at several BSTs. Where are the min values always located?

```
public Object min(TreeNode t)
{









}
```

10. Starting at the root, return the max value in the BST. Use recursion!

```
public Object max(TreeNode t)
{









}
```

Assignment

Write a menu-driven program that will perform the above operations on a binary search tree of any size using Characters. Use the class TreeNode as defined by the College Board (available in class folder as TreeNode.java). Prompt the user for an input string. Build a Binary Search Tree, using Comparables. Display it as a sideways tree (use the given method below), then let the menu take over.

Sample Run

```
Input string: OERCMPTU
        U
     T
  R
     P
O
     M
  E
     C

Binary Search Tree
A) Create a tree
B) Add a node
C) Display the tree sideways
D) Display a level
E) Display Preorder Traversal
F) Display Postorder Traversal
G) Display Inorder Traversal
H) Display number of nodes
I) Display number of leaves
J) Display tree height
K) Display minimum value
L) Display maximum value
M) Search for an element in the tree
N) Quit

Choice: G

In Order: C E M O P R T U
```

Challenges

- Balance a BST. That is, for the entire tree and each subtree, find the ideal root such that the number of nodes in each subtree is as close to equal as possible.
- Look on the Internet for red-black trees, AVL trees, AA Trees. Implement one or all.

Code

```
public static void displaySideways(TreeNode t, int level)
{
   if(t == null)
      return;
   displaySideways(t.getRight(), level + 1); //recurse right
   for(int k = 0; k < level; k++)
      System.out.print("\t");
   System.out.println(t.getValue());
   displaySideways(t.getLeft(), level + 1);  //recurse left
}
```