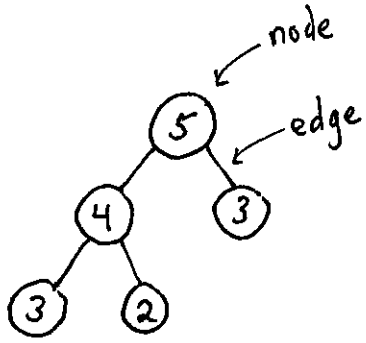
This is a tree:

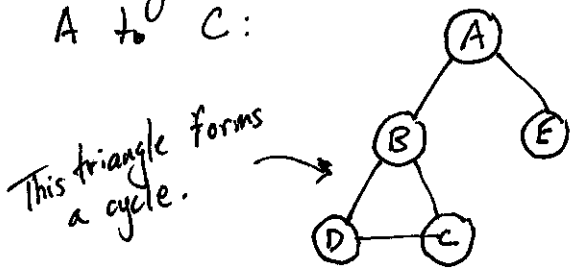


Trees consist of nodes (which contain values) connected by edges. The hierarchical ordering of our diagram conveys parent-child relationships. 5 is the parent of 4 and 3; 4 and 3 are its children. A node can have many children, but only one parent (at most).

Note: 5 is not the parent of 2. It is the grandparent of 2.

Trees cannot contain cycles (loops). That is, there can only be one path from each node to every other node in the tree (assuming we don't allow any repeated edges in our paths).

For example, the following is not a tree because there are multiple paths from A to C:



Path #1: A → B → C

Path #2: A → B → D → C

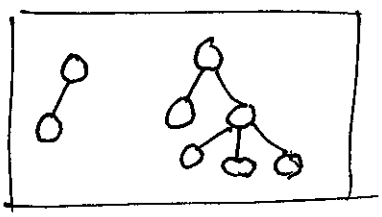
Note: A tree must be connected, meaning we can find a path from any node in the tree to any other node in the tree. Thus, a tree with n nodes necessarily has exactly $n-1$ edges.

A collection of trees is called a forest.

A node with no children is called a leaf.

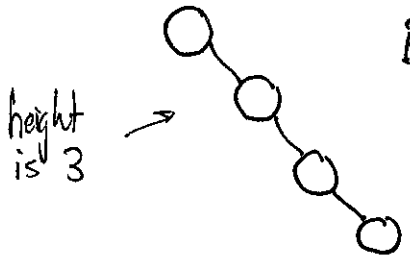
A node with no parent is called the root of a tree.

This is not a tree; it's a forest:



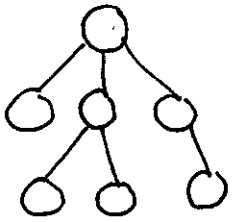
A binary tree is a tree in which every node has 0, 1, or 2 children.

(58) ← Is this a binary tree?
Yes. :)
(height is 0)



Binary tree?
Yes.

← height is -1



Not a binary tree.
(height is 2)

This is the empty tree:
(It has no nodes.)

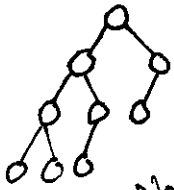
The height of a tree is the greatest distance from the root (in terms of # of edges) to any of the leaf nodes in the tree. (See examples above.)

Note: We define the height of the empty tree to be -1.

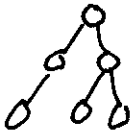
A binary tree is full if every node in the tree has 0 or 2 children.

A binary tree is complete if all levels of the tree are completely filled up, except perhaps for the last level, which must be filled from left to right with no gaps between nodes.

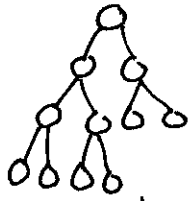
Examples:



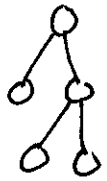
Full? No
Complete? No



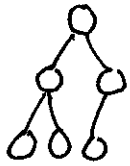
Full? No
Complete? No



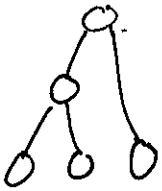
Full? Yes!
Complete? Yes!



Full? Yes!
Complete? No!



Full? No
Complete? Yes

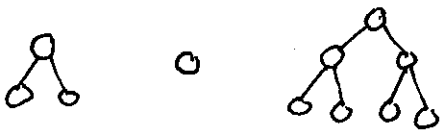


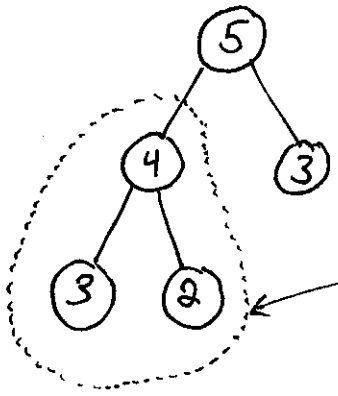
Beware shady diagrams!

Full? Yes
Complete? Yes
Perfect? No

A perfect binary tree is one in which all non-leaf nodes have two children, and all leaf nodes are on the same level within the tree.

These are all perfect binary trees:





This is a subtree of our tree.

This is the subtree rooted at 4.

It's also the left subtree of 5.

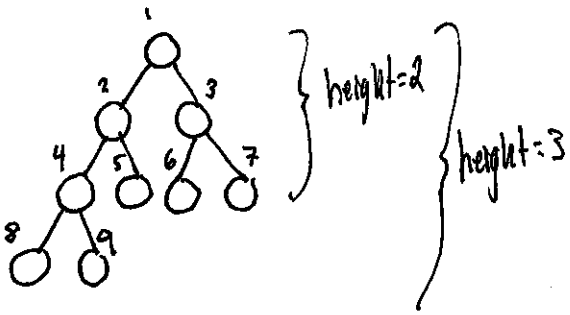
The right subtree of node 5 has height 0.

The left subtree of node 2 is the empty tree.

The empty tree does not have left or right subtrees, Attempting to discuss those subtrees might create a segfault in the universe. Please be careful!

Let's examine the height of a complete binary tree. As we fill each level from left to right, notice that the tree's height increases when we insert the 1st, 2nd, 4th, 8th, and 16th nodes (and so on).

height = 0 height = 1 height = 2 height = 3 ...



Aha! The height of a complete binary tree with exactly n nodes is exactly:

$$\lfloor \log_2 n \rfloor$$

More knowledge on the next page. Keep going. ↓

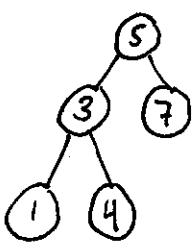
:)

A binary search tree (BST) is a binary tree with the following property:

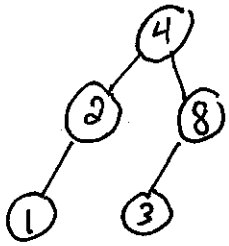
For any given node, x, the values in the left subtree of x must be less than the value at x, and the values in the right subtree of x must be greater than the value at node x.

(If we allow duplicate values in a BST, then the values to the left of x must be less than <u>or equal to</u> the value at node x.)

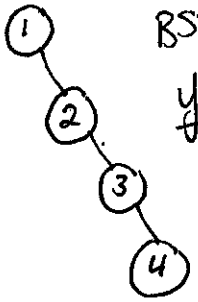
<u>Examples:</u>



BST?
Yes!

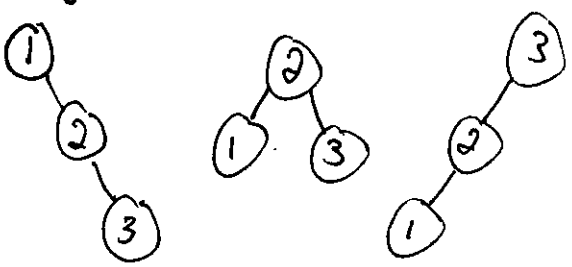


BST?
No



BST?
Yes!

How many BSTs can you make with the values 1, 2, and 3?



Most people overlook these two at first.

*<u>Note</u>: If you shuffle n distinct elements and then place them into a BST, the average height will be $O(\log n)$.

Thus, the runtimes for searching for an element in a BST are:

Best Case: $O(1)$ — the search key is at the root!

Worst Case: $O(n)$ — the BST has devolved into a linked list of height $n-1$.

Avg. Case: $O(\log n)$ — expected height of a BST

Traversal Algorithms

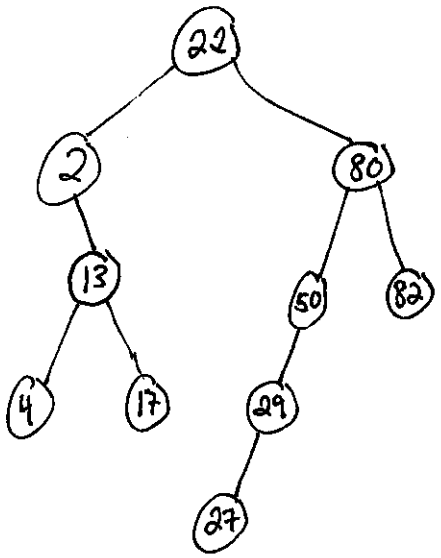
These allow us to visit each node in a binary tree in a certain order.

inorder traversal:
1. Traverse the left subtree using inorder traversal.
2. Visit the root node of the (sub)tree for this particular function call.
3. Traverse the right subtree using inorder traversal.

preorder traversal: (visit root before subtrees)
1. Visit the root node of this (sub)tree.
2. Traverse the left subtree using preorder traversal.
3. Traverse the right subtree using preorder traversal.

postorder traversal: (visit root after subtrees)
1. Traverse the left subtree using postorder traversal.
2. Traverse the right subtree using postorder traversal.
3. Visit the root node of this (sub)tree.

For example, let's traverse the following BST, which we constructed by inserting values into the BST in the following order: 22, 2, 80, 82, 50, 29, 13, 27, 17, 4



Inorder Traversal:
4 17 13 2 27 29 50 82 80 22

Preorder Traversal:
22 2 13 4 17 80 50 29 27 82

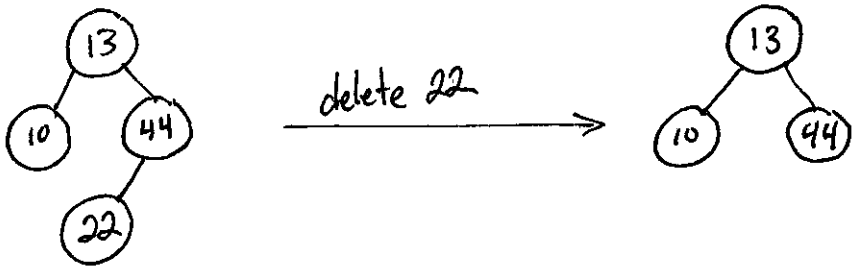
Postorder Traversal:
4 17 13 2 27 29 50 82 80 22

*Note: The only integer values we could insert as the right child of 50 in the BST above (if we don't allow insertion of duplicate values) would be 51 through 79.

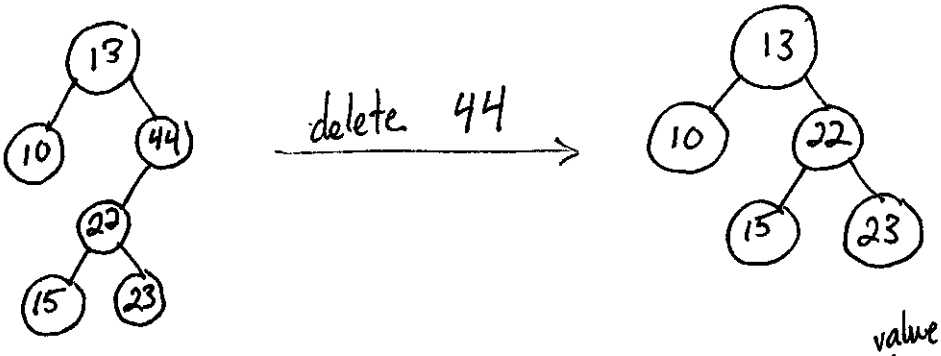
Deletion Examples

There are three cases we must account for when deleting from a BST:

[1] The node to be deleted is a leaf. In this case, just remove it.



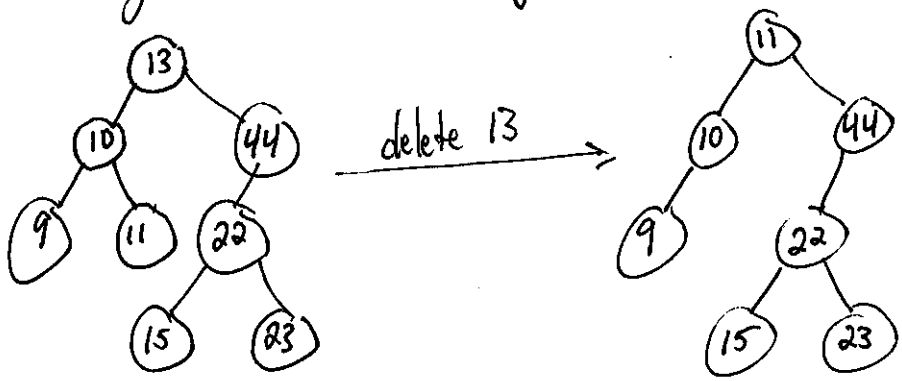
[2] The node to be deleted has a single child. Just move that child up. Notice that the BST properties are always preserved when we do this.



[3] The node to be deleted has two children. Replace the node to be deleted with the max value of the left subtree. Then delete that value from the left subtree. (It's guaranteed to have only 0 or 1 children. Do you see why?)

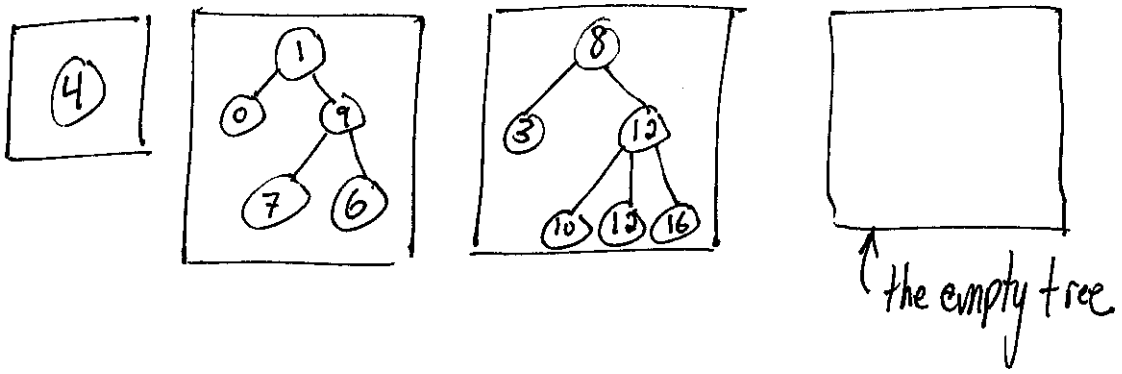
value at the

We could also have used the min value in the right subtree, but let's stick to the policy above for this semester, for the sake of consistency.

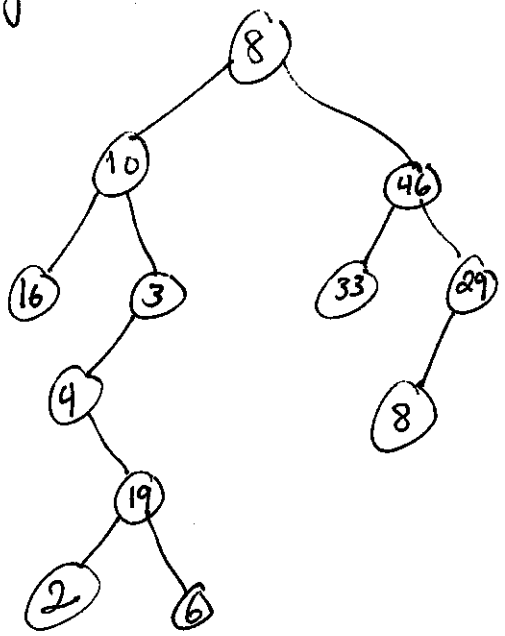


Practice Problems

(1) Indicate whether each of the following is a tree, a binary tree, and/or a BST. If it is a tree, give its height. If it is a binary tree, tell whether or not it is full, whether or not it is complete, and whether or not it is perfect.



(2) Give the pre-order, post-order, and in-order traversals for the following binary tree:



(3) Is this a BST? What is the tree's height?

(4) How many distinct BSTs can be constructed with node values 1 through n?
integer

(5) Draw a single BST that has both the following traversal orderings:

pre-order: 1 15 3 2 7 29 18

post-order: 2 7 3 18 29 15 1