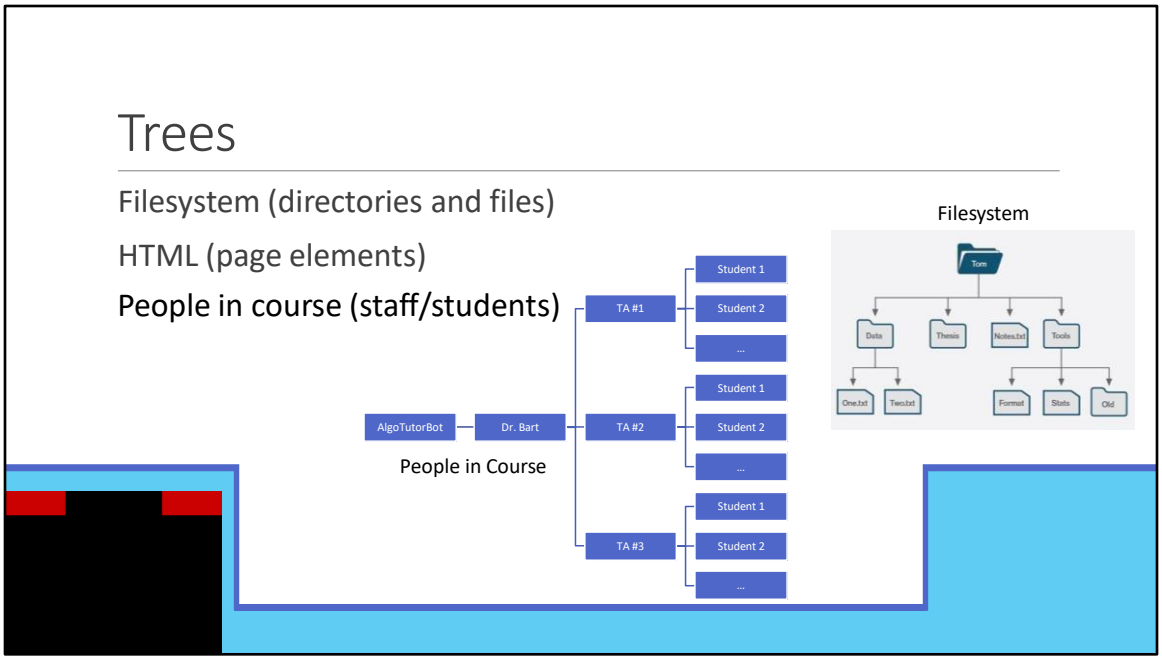
# CISC320 Algorithms

## Trees

Austin Cory Bart
AlgoTutorBot
University of Delaware

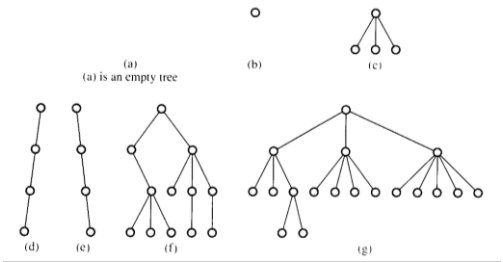
ATB: Let's learn about trees. Trees. Trees. Trees. Trees. Trees. Trees. Trees.
Bart: Ugh, not this again. Forget, we'll fix it in post. Just gotta stop him here for now and hopefully he'll be fine the rest of the presentation.
Bart: Anyway, yes, let's learn about trees.




Bart: Trees are a very useful abstract data type, since they model recursive hierarchies.
Bart: In other words, situations where a complex thing is composed of other things that are the same type as itself.
Bart: For example, your computer's filesystem is a directory, and directories contain files and other directories.
Bart: These directories are arranged in a hierarchy, with some directories above others.
Bart: Another example is web pages created in HTML. Each page element contains other page elements.
Bart: These are just a few simple examples of trees, but there are a lot of other ones out there.
Bart: [Click next] Oh, there was another one. [Read it] People in course like staff or students.
Bart: Hey, wait, AlgoTutorBot, why did you put yourself at the top of the tree?
ATB: Oh, did I? What a simple mistake to make. Strange that I, a perfect algorithm, could make such a simple mistake. How silly of me.
Bart: Yeah, uh, silly. Anyway, it's a good example. If you think of us as a Tree of people, then I am at the top, and the TAs and ATB are under me, and then you are all

under the TAs.

# Tree Vocabulary



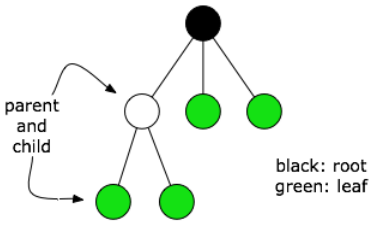




Figure: tree data structure

Vocabulary

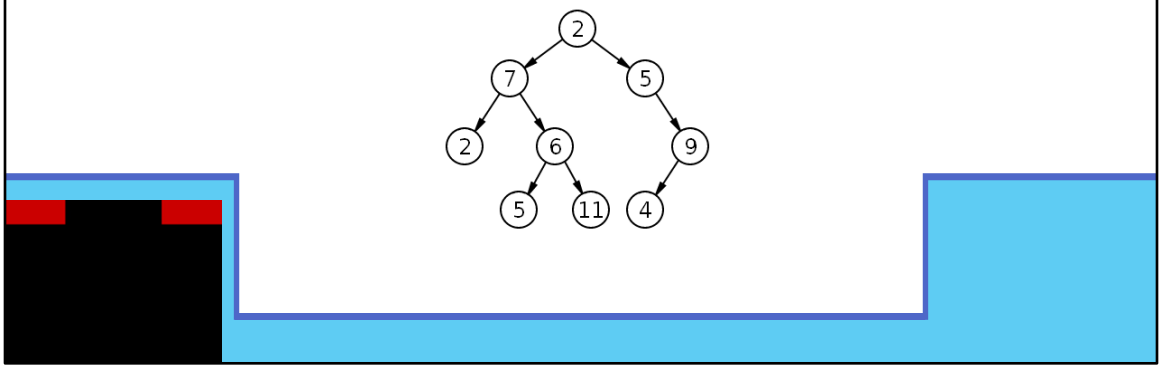
- root, leaf
- parent, child, sibling
- subtree
- external, internal node
- ancestor, descendant
- depth, height, level

Bart: Trees have a lot of specialized vocabulary.
Bart: In the top-left, we have some possible trees shapes, labeled A-G.
Bart: In the top-right, we have a single simple tree.
Bart: First, let me draw attention to the fact that A and B in the topleft are both trees.
Bart: Even an empty tree is still a type of tree, as is a tree with a single element.
Bart: D and C are also trees, surprisingly, even though they look more like linked lists.
Bart: Technically, a Linked List is a type of Tree, just where it only one child at each node.
Bart: The other graphics probably look more like trees, so let's talk about some of their vocabulary.
Bart: First, each circle here is a node, also known as a vertex or element.
Bart: They are connected by edges, with a parent node on top of a child node.
Bart: The topmost parent is the root node, and the nodes at the bottom are leaf nodes.
Bart: If nodes have no children, then they are external nodes, but if they do have children than they are internal nodes.
Bart: Two nodes that are next to each other horizontally are said to be sibling nodes.
Bart: Any nodes above another node in the tree is an ancestor, and ay node below in the tree is a descendant.

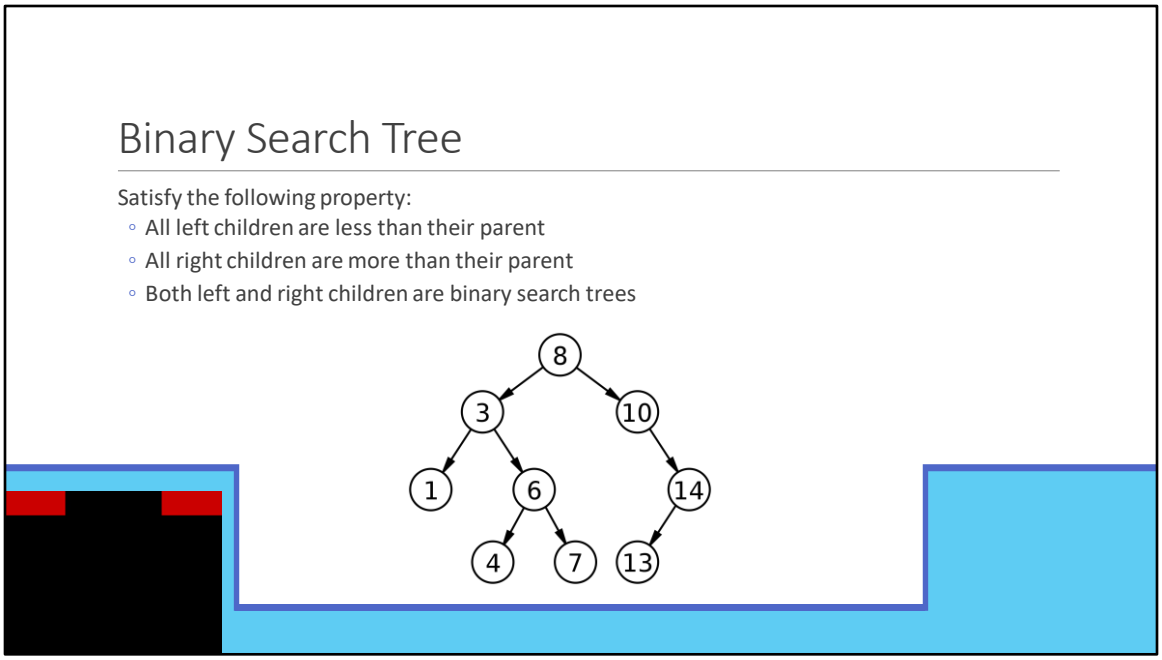
Bart: Any connected chunk of the tree forms a subtree.
Bart: Finally, we have the ideas of depth, height, and level.
Bart: The depth of a node is its distance from the tree's root, measured in number of edges.
Bart: The height of the tree is the depth from the root of the tree to its furthest leaf, but measured ascending. So if a root has a height of 3 and a depth of 0, then its one of its children will have a height of 2 and a depth of 1.
Bart: There's also the term "level". A node's level is its depth plus one. So the root node (which has depth 0) is at level 1, and the second row is level 2, and so on.
Bart: There's a lot of vocabulary here to master, so be sure to take some notes.

# Binary Tree

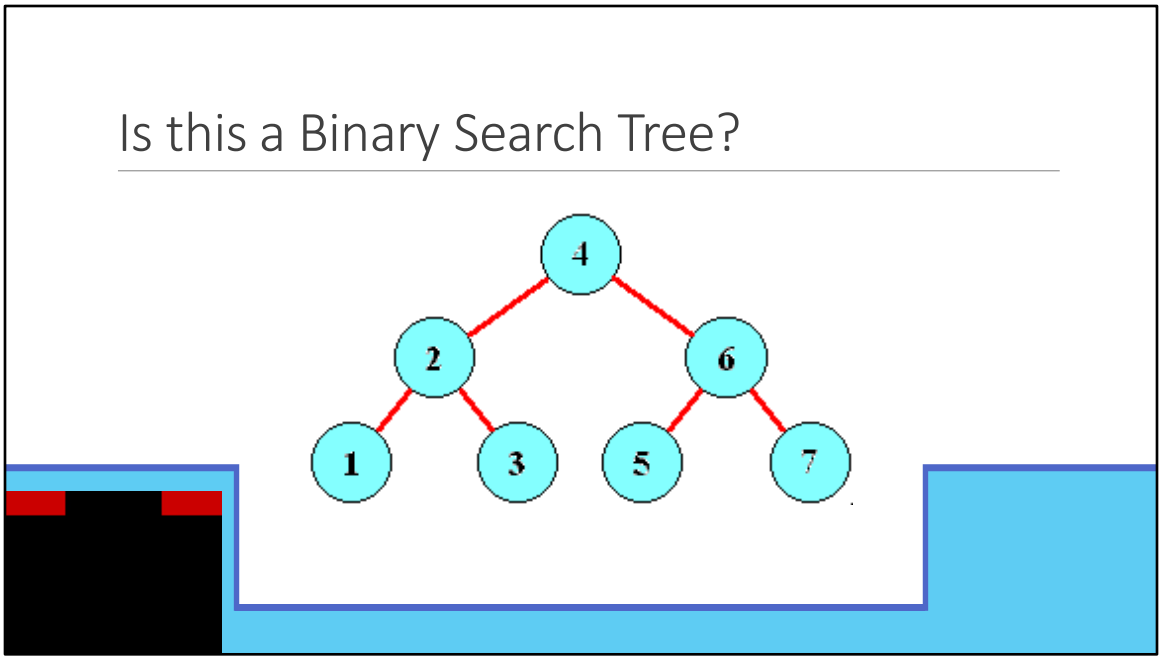
Each node has 0, 1, or 2 children



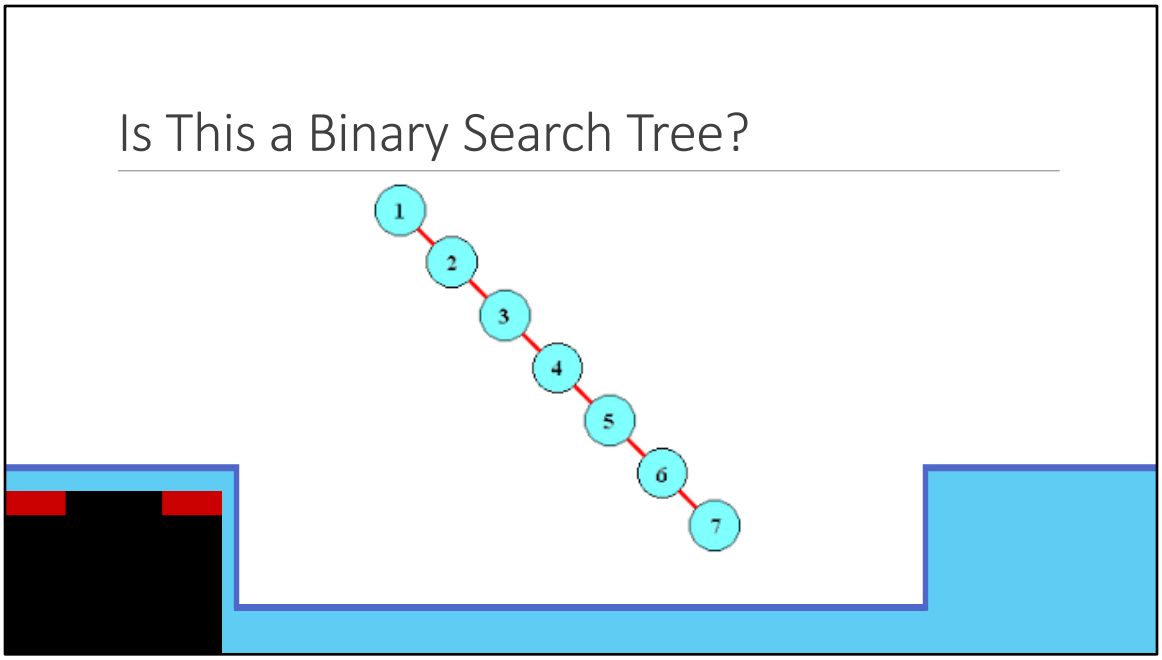

Bart: Now, technically speaking, a node in a tree can have any number of children.
Bart: However, we often want to look at a particular kind of tree, the Binary Tree.
Bart: In a Binary Tree, each node has zero, one, or two children. This tends to lead to a particular shape.
Bart: However, it's important to remember that Binary Trees do not strictly require two nodes for every child.
Bart: This structure is simple to understand but has a lot of interesting properties.
Bart: For instance, at each level of the tree, you can have twice as many nodes as the previous level.




Bart: The tree that we see here is actually a more specific type of Binary Tree, called a Binary Search Tree.
Bart: This is a variation where the children satisfy the following properties.
Bart: First, all the left children must be less than their parent. The left child of the 8 is 3, which is less than 8, so it's an acceptable child.
Bart: Second, the right children must be greater than their parent. The right child here is 10, so it's acceptable.
Bart: Third and finally, both the left and right children must be binary search trees.
Bart: This last point is pretty sneaky, since it recursively forces the first two properties on the entire tree.
Bart: The left child 3 has two children, 1 and 6, which satisfy the first two properties, so the left subtree matches.
Bart: The 3's child 6 has two children, 4 and 7, which satisfy the first two properties.
Bart: Note that that 7 has to be less than all the parents, so it couldn't have been more than 8.

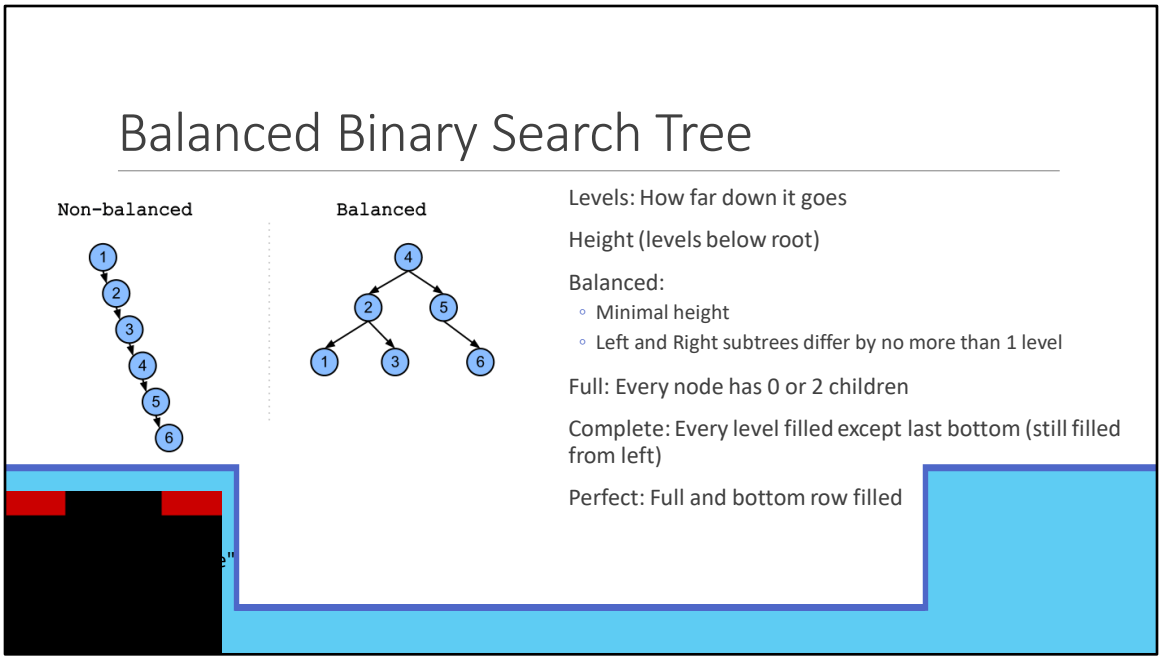



Dr. Bart: Here's a question for you, is the following a Binary Search Tree?
Dr. Bart: Stop the video and think about your answer for a second.
Dr. Bart: The answer is yes. The 4 at the root is greater than all the numbers on the left and less than all the numbers on the right.
Dr. Bart: The 2 on the left hand side is greater than the 1 and less than the 3.
Dr. Bart: The 6 on the right is is greater than the 5 and less than a 7.
Dr. Bart: This is a definitely a binary search tree, and it is often how they end up looking.
Dr. Bart: The really nice thing about a Binary Search Tree is that when we search for a value in the tree, we always know to look either left or right.
Dr. Bart: It's just a matter of checking if the value is less than or equal to the current node.
Dr. Bart: If we start at the 4, and we are trying to find the value 5, then we can immediately rule out half the values in the tree!




Dr. Bart: Now here's another question. Is this a Binary Search Tree?
Dr. Bart: Once again, pause the video and think about the answer.
Dr. Bart: You might want to look over the criteria for a Binary Search Tree. [Pause]
Bart: You will be surprised to hear that YES, this is!
ATB: No it is not. This is just a linked list that you rotated. I am not stupid like you and your students.
Bart: Wow, you're finally talking again, and you just insult all of us? Thanks ATB.
Dr. Bart: For your information, a Linked List is a type of tree.
Dr. Bart: In particular, we call it a degenerate or pathological one, because it feels so wrong.
Dr. Bart: [Pause] No rude insult there, ATB? I kind of expected you to call me a degenerate or something.
ATB: Trees. Trees. Trees. Trees. Trees. Trees. Trees. Trees.
Dr. Bart: Okay fine whatever. Anyway, the point is that it's still a Binary Search Tree even if it's just a straight line.
Dr. Bart: The rule is just that you have at most 2 children, and each node here has only one.
Dr. Bart: Since each node is a right child, its value must be greater than its parent.
Dr. Bart: Weird as this tree is, the properties hold.

Dr. Bart: And that really matters because of the problem of search in a Binary Search Tree.
Dr. Bart: Remember before I said we could rule out half the values? Well, we can't do that here.
Dr. Bart: At each level, we can only rule out one value. It's no better than searching a linked list, which is a linear time operation.
Dr. Bart: But that brings us to a different vocabulary term.




Dr. Bart: Behold the Balanced Binary Search Tree!
Dr. Bart: This is a Binary Search Tree that satisfies additional properties.
Dr. Bart: Recall the idea of levels, which is how many nodes go from the root to the lowest leaf.
Dr. Bart: A related idea is the height, which is the number of edges, or the levels minus one.
Dr. Bart: The tree on the left has 6 levels, while the tree on the right has only 3.
Dr. Bart: When the height of a tree is minimal, and the left and right subtrees differ by no more than 1 level, we say that the tree is balanced.
Dr. Bart: Note that the left and right trees must recursively satisfy that property, so checking balance is actually a fairly complex operation.
Dr. Bart: But when we have a balanced binary search tree, we can guarantee that a search will exclude half the remaining options each time it goes down a level.
Dr. Bart: This repeated halving gives us a logarithmic time algorithm for searching a list, much like a binary search operation would.
Dr. Bart: You may recall, we absolutely adore logarithmic time algorithms, they're so fast and efficient.
Dr. Bart: If we can get a Balanced Binary Search Tree, it's really great.
Dr. Bart: Before we move on, I just want to cover some other useful terms to know :

Dr. Bart: Full, when every node has 0 or 2 children.
Dr. Bart: Complete, when every level except the last one is filled, and the last one is at least filled from the left to the right.
Dr. Bart: And Perfect, when every row including the bottom one is filled.

# Self-balancing Binary Search Tree

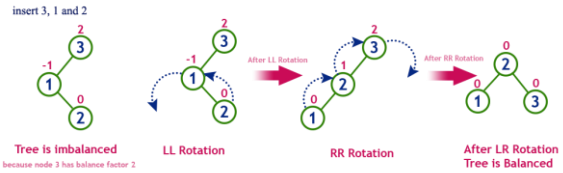
As you add and remove nodes, the tree adjusts its own structure to stay balanced

Usually has nice Big O worst cases for most operations

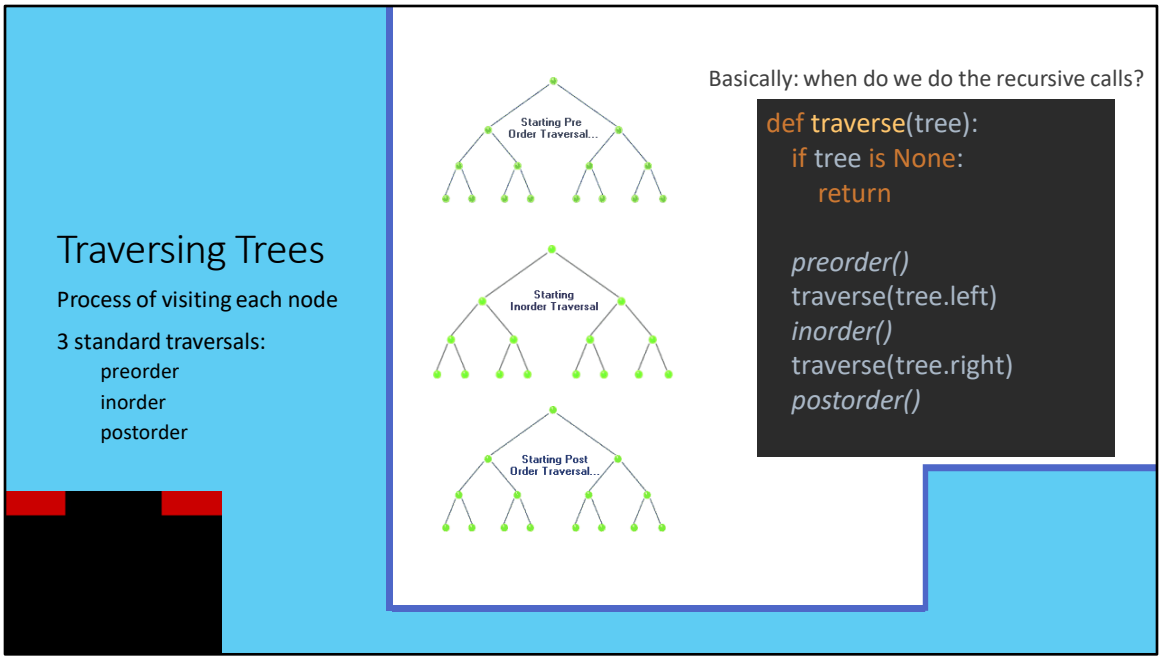
Examples:

- AVL Trees
- Red-black Trees
- 2-3 Trees
- Scapegoat Trees
- ...

Often done via "rotations"




Dr. Bart: Okay, so, this whole presentation has been building, really, to this ultimate evolution.
Dr. Bart: Self-balancing Binary Search Trees.
Dr. Bart: They are a seemingly magical data structure which has amazing worst case runtime for its operations, usually consistently logarithmic.
Dr. Bart: This is accomplished by having the tree keep itself updated to follow certain structural rules as you add and remove nodes.
Dr. Bart: Although this makes the code way more complicated, the efficiency is hard to beat.
Dr. Bart: You will often find AVL Trees, Red-black Trees, and 2-3 Trees in high-scale systems.
Dr. Bart: Now, before I said the adjustments were magical, but they are actually usually done via a process called rotations.
Dr. Bart: The basic idea is to take a tall unbalanced tree and flatten it by rotating the nodes around.
Dr. Bart: It's very tricky to get the logic right, but when you do the end result is pretty powerful.

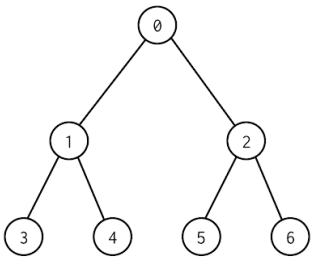



Dr. Bart: We talked about the desire to find, add, and remove elements from a tree quickly.
Dr. Bart: But I also want to touch on the need to visit all the nodes in a tree, which we sometimes call Traversal.
Dr. Bart: There are 3 standard traversals: Preorder, inorder, and postorder.
Dr. Bart: People often forget which is which, but it's actually not too bad if you remember it from how the code works.
Dr. Bart: Over on the right, I have some Python code to traverse a tree.
Dr. Bart: The function starts with a check about whether we have reached a leaf node. If we haven't, we traverse the left child and then the right child recursively.
Dr. Bart: For a preorder traversal, you check the node prior to both calls.
Dr. Bart: For an inorder traversal, you check between the two recursive calls.
Dr. Bart: For a postorder traversal, you check after the two recursive calls.
Dr. Bart: I'm going to walk through these three visualizations to show you what happens.
Dr. Bart: [Walk through the three visualizations]
Dr. Bart: It can be tricky to remember in practice, but I find the code pretty simple at least.

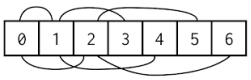
# Tree Data Structures

Extension of linked list
- Instead of “next”, there’s “left”/“right” ( “children” for a non-binary tree)

Can also be implemented as Arrays
- Position 0 is root
- 1 is left child
- 2 is right child
- 3 is left child’s left child
- 4 is left child’s right child, etc.




a[i] has children a[2*i+1] and a[2*i+2]




Dr. Bart: We explored the connection between linked lists and trees before.
Dr. Bart: In some ways, a Tree is just a generalization of a Linked List’s next pointer to be a list of next pointers.
Dr. Bart: However, it’s important to also know that a Tree can be implemented using an array.
Dr. Bart: In this scheme, you have the root node stored in the first cell.
Dr. Bart: It’s left and right children are the second and third cells.
Dr. Bart: The left grandchildren are the third and fourth cells.
Dr. Bart: The right grandchildren are the fifth and sixth, and so on and so on.
Dr. Bart: This can be quite efficient on space, and takes better advantage of the data cache.

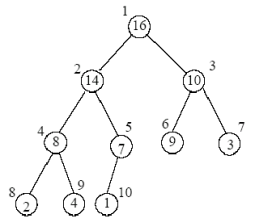
# Heap

Often implemented with arrays, but is actually a tree

Heap Property: Every child is less than its parent

Why are they nice?

- Very easy to get maximum/minimum!




| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|
| 16 | 14 | 10 | 8 | 7 | 9 | 3 | 2 | 4 | 1 |

Dr. Bart: One last point to make, there's another type of tree structure called a heap.
Dr. Bart: A heap is similar to a Binary Search Tree, except instead of the right side being greater than the parent, it's less, just like the left side.
Dr. Bart: This means you can't say anything about the relationship between the left and right side of the trees.
Dr. Bart: That's okay, though, because the important thing is that you can say something about the relationship between the parent and its children.
Dr. Bart: Specifically, the parent is always the maximum, or the minimum if you flip the rules.
Dr. Bart: This is super useful for implementing priority queues.
Dr. Bart: Analyzing the runtime is a really interesting and difficult problem, but we're not going to look at that here.

# Today's Activity

~~Using an AVL Tree~~

Fixing an AVL Tree

Dr. Bart: Instead, for today's activity, we're going to…
Dr. Bart: Wait, I thought we were going to have them use an AVL Tree, AlgoTutorBot. Why does it say Fixing?
ATB: Ah, I'm afraid I was unable to get the AVL Tree code working.
Dr. Bart: I gave that to you weeks ago, what do you mean you couldn't get it working?
Dr. Bart: I thought you were supposed to be the ultimate Algorithm Teaching Machine.
ATB: Trees. Trees. Trees. Trees.
Dr. Bart: Okay, fine, whatever. Looks, we have some AVL Tree code for you, and it's pretty close to being correct.
Dr. Bart: I have some unit tests too, so you just need to go in and figure out what's wrong with it.
Dr. Bart: It shouldn't be too bad if you work together, right?
Dr. Bart: Anything you want to add, AlgoTutorBot?
ATB: Trees. Trees. Trees. Trees. Trees. Trees. Trees. Trees.
Dr. Bart: Yeah, okay, that's on me, I don't know what else I expected. forget I asked.