TREES

Lecture 11

CS2110 – Summer 2019

Announcements

- Confusion about submission due dates/times can be cleared by reading the syllabus (https://courses.cs.cornell.edu/cs2110/2019su/syllabus.html).
- Remember to make groups before submission deadline.
- Grades have been released for Assignment 1 and Discussions 1-3. Please submit (private) questions about either of these on Piazza.

Today's Topics in JavaHyperText

- Search for "trees"
- Read PDFs for points 0 through 5: intro to trees, examples of trees, binary trees, binary search trees, balanced trees

Data Structures

- **Data structure**
  - Organization or format for storing or managing data
  - Concrete realization of an abstract data type
- Operations
  - Always a tradeoff: some operations more efficient, some less, for any data structure
  - Choose efficient data structure for operations of concern

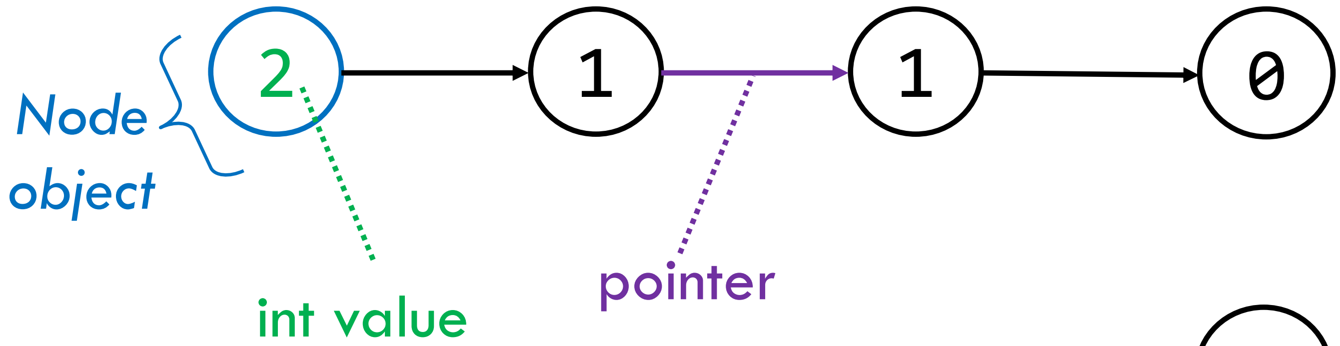
Example Data Structures

Data Structure	add(val v)	get(int i)	contains(val v)
Array 2 1 3 0	$O(n)$	$O(1)$	$O(n)$
Linked List 2 → 1 → 3 → 0	$O(1)$	$O(n)$	$O(n)$

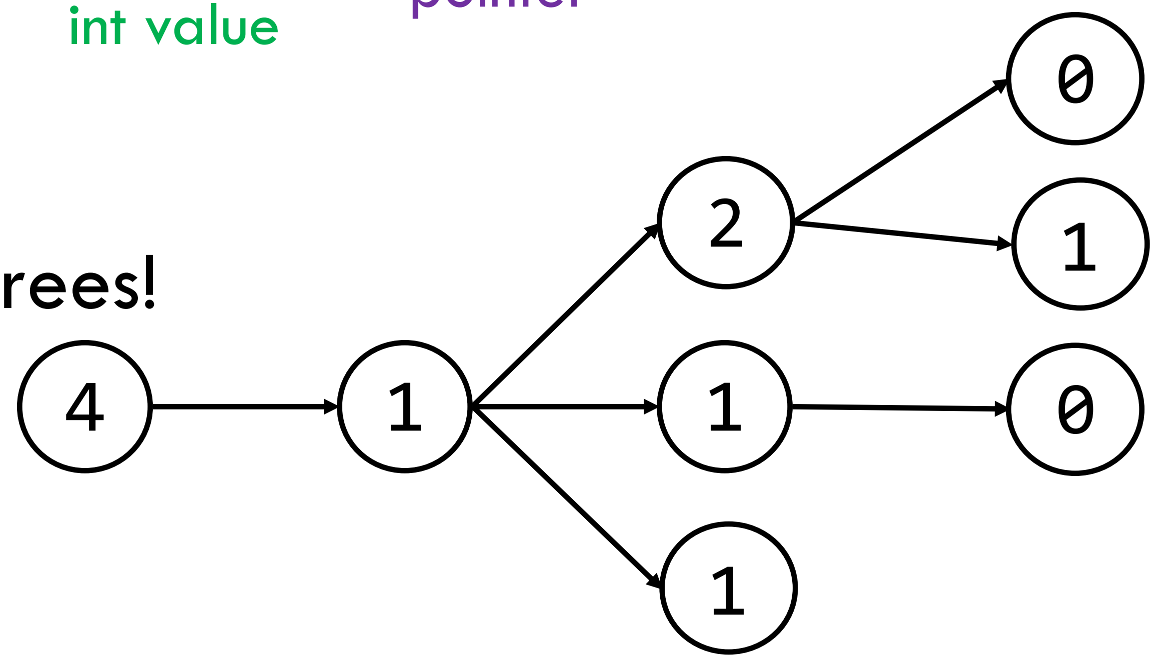
```
add(v): append v
get(i): return element at position i
contains(v): return true if contains v
```

Tree

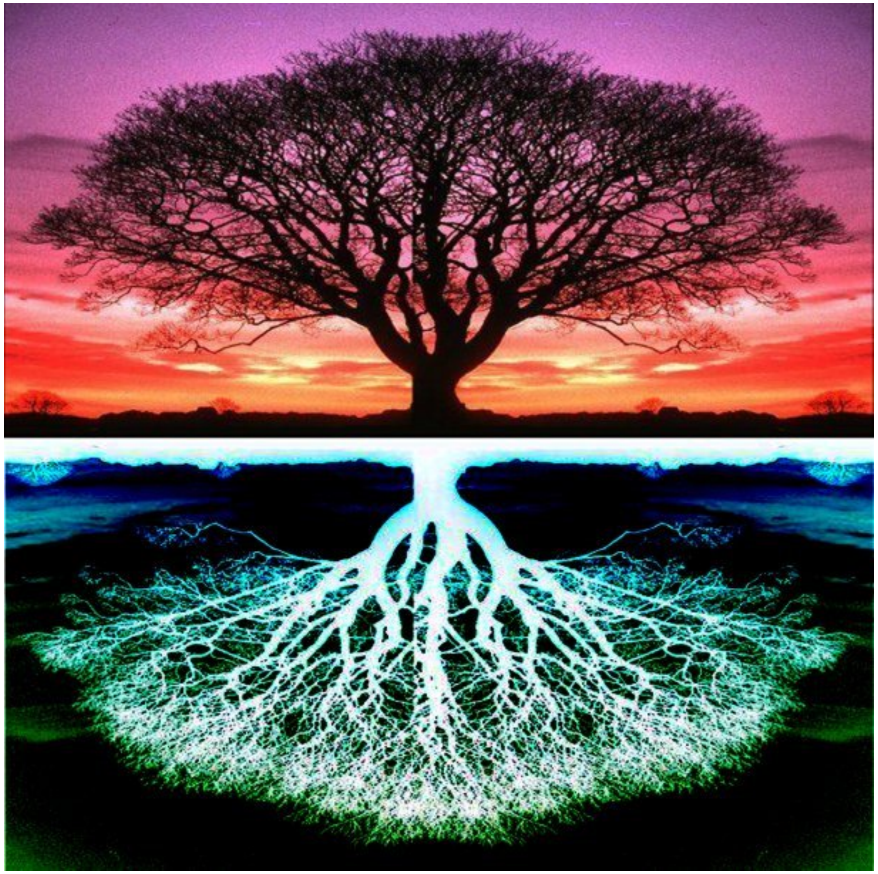
Singly linked list:



Today: trees!



Trees



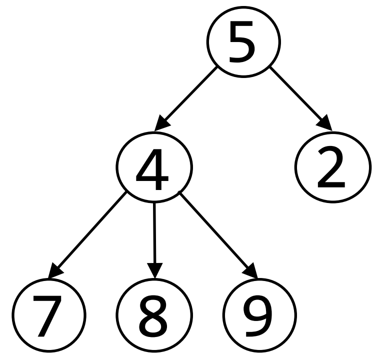
In CS, we draw trees “upside down”

Tree Overview

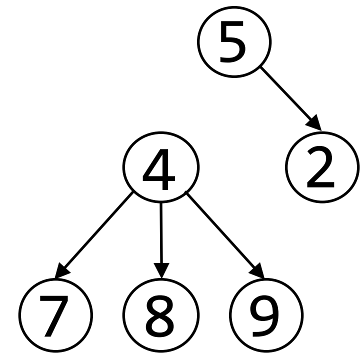
Tree: data structure with nodes, similar to linked list

- Each node may have zero or more *successors* (children)
- Each node has exactly one *predecessor* (parent) except the *root*, which has none
- All nodes are reachable from *root*

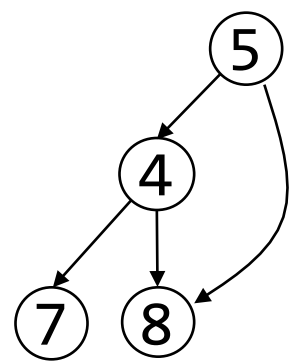
A tree or not a tree?



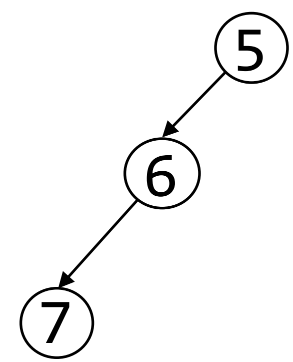
A tree



Not a tree

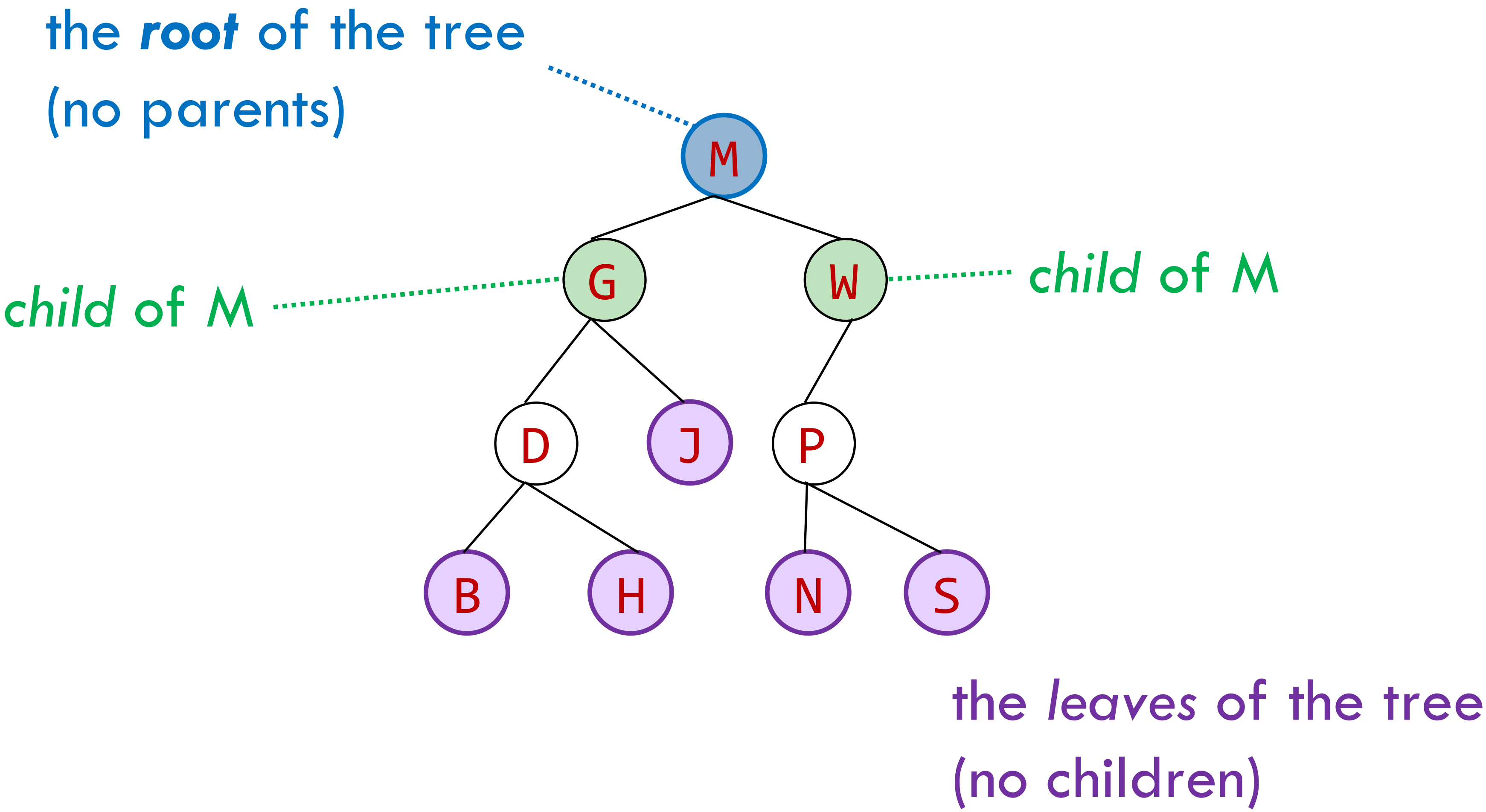


Not a tree

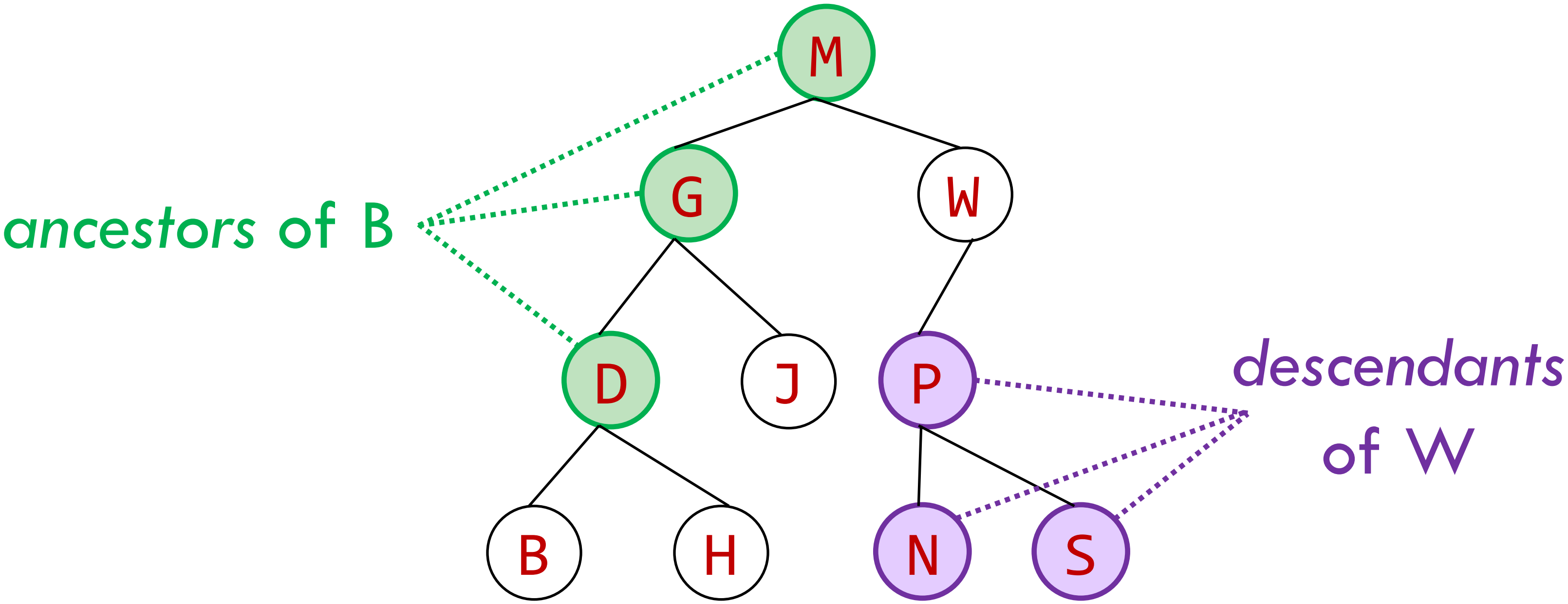


A tree

Tree Terminology (1)

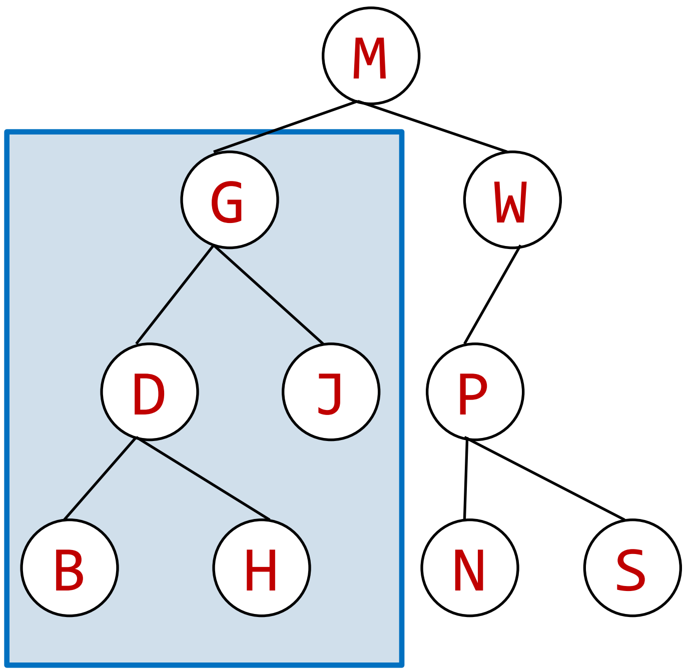


Tree Terminology (2)



Tree Terminology (3)

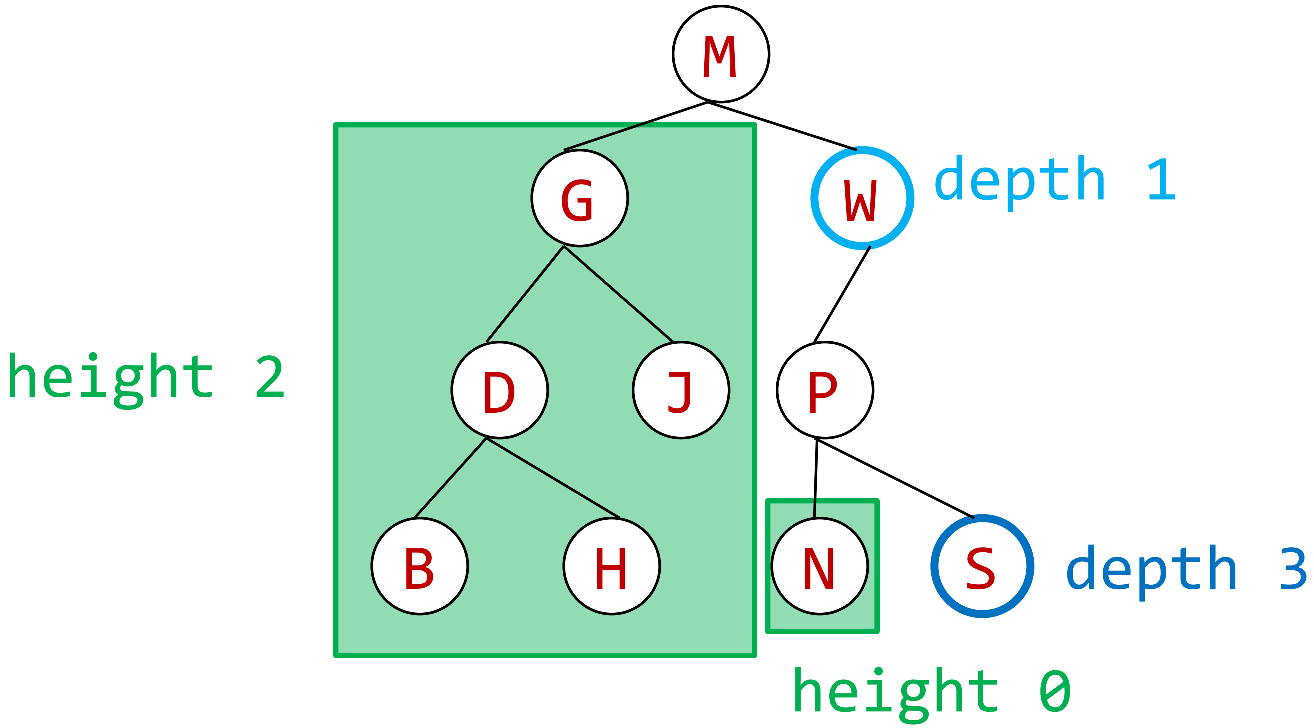
subtree of *M*



Tree Terminology (4)

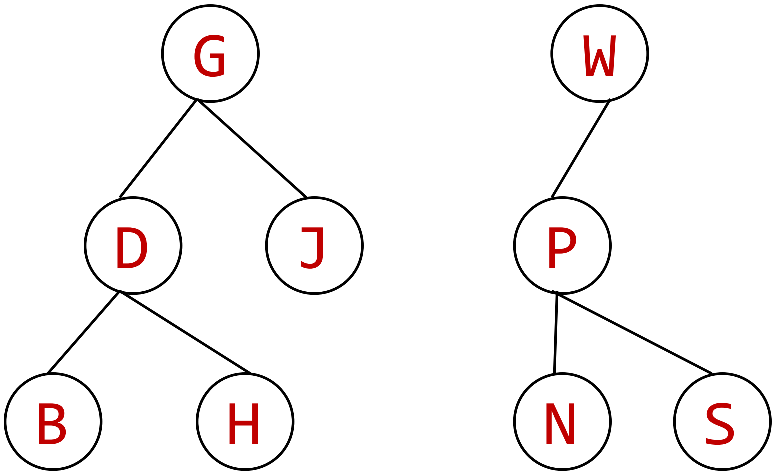
A node's ***depth*** is the length of the path to the root.

A tree's (or subtree's) ***height*** is the length of the longest path from the root to a leaf.



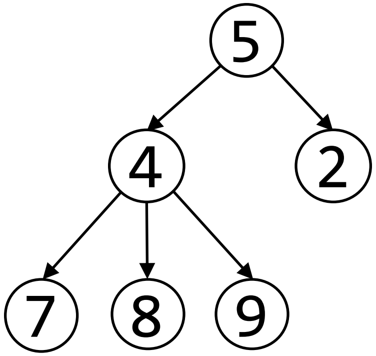
Tree Terminology (5)

Multiple trees: a ***forest***



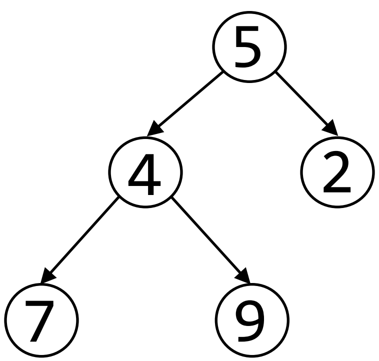
General vs. Binary Trees

General tree: every node can have an arbitrary number of children



General tree

Binary tree: at most two children, called *left* and *right*



Binary tree

…often "tree" means binary tree

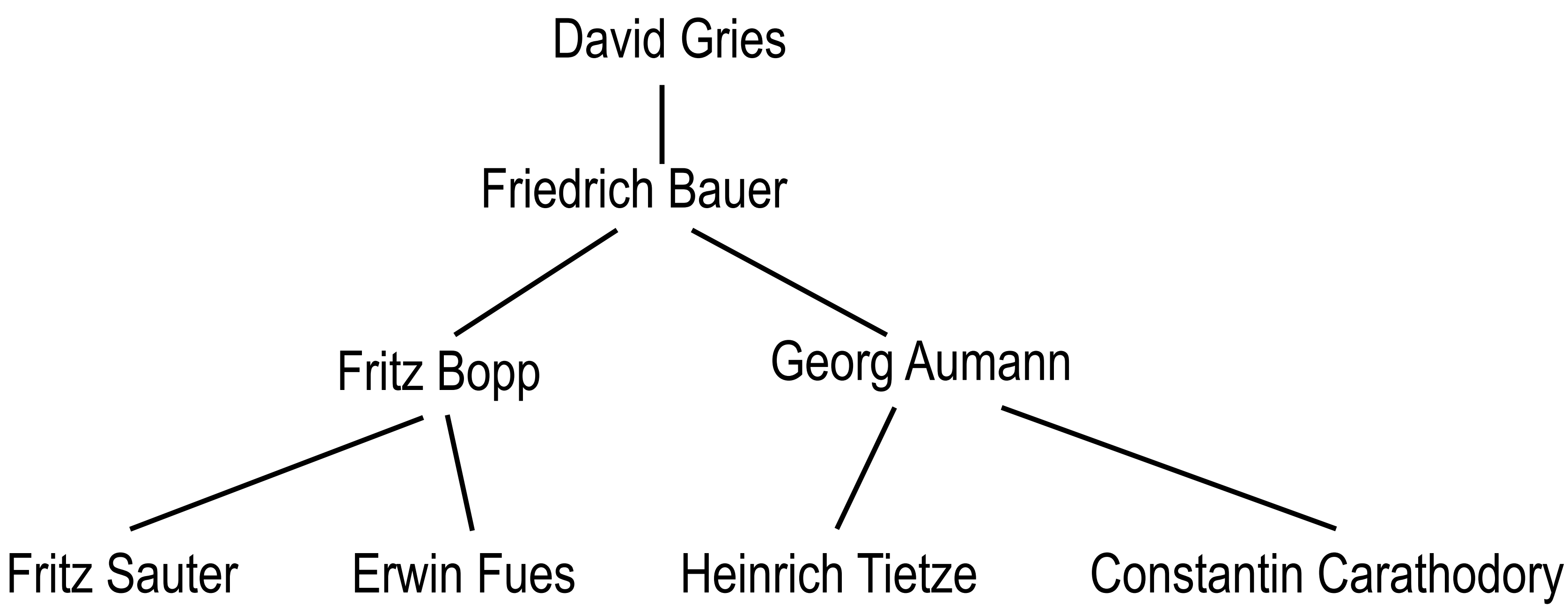
Demo

Binary trees were in A1!

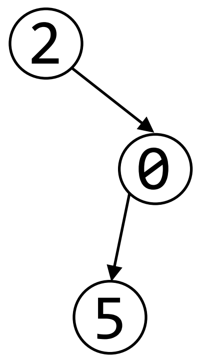
You have seen a binary tree in A1.

A PhD object has one or two advisors.

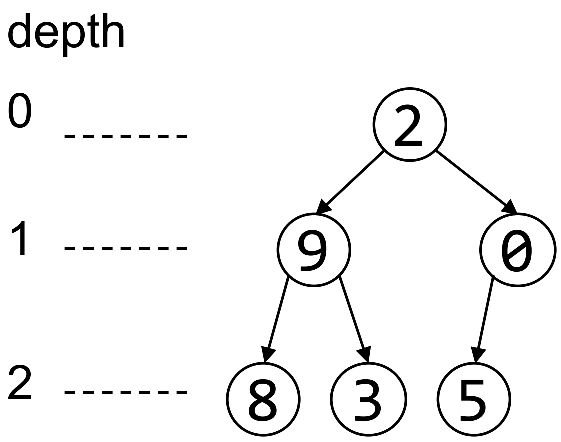
(Note: the advisors are the "children".)



Special kinds of binary trees



Height 2,
minimum number of nodes



Height 2,
maximum number of nodes

Max # of nodes at depth d: 2^d

If height of tree is h:

min # of nodes: $h + 1$

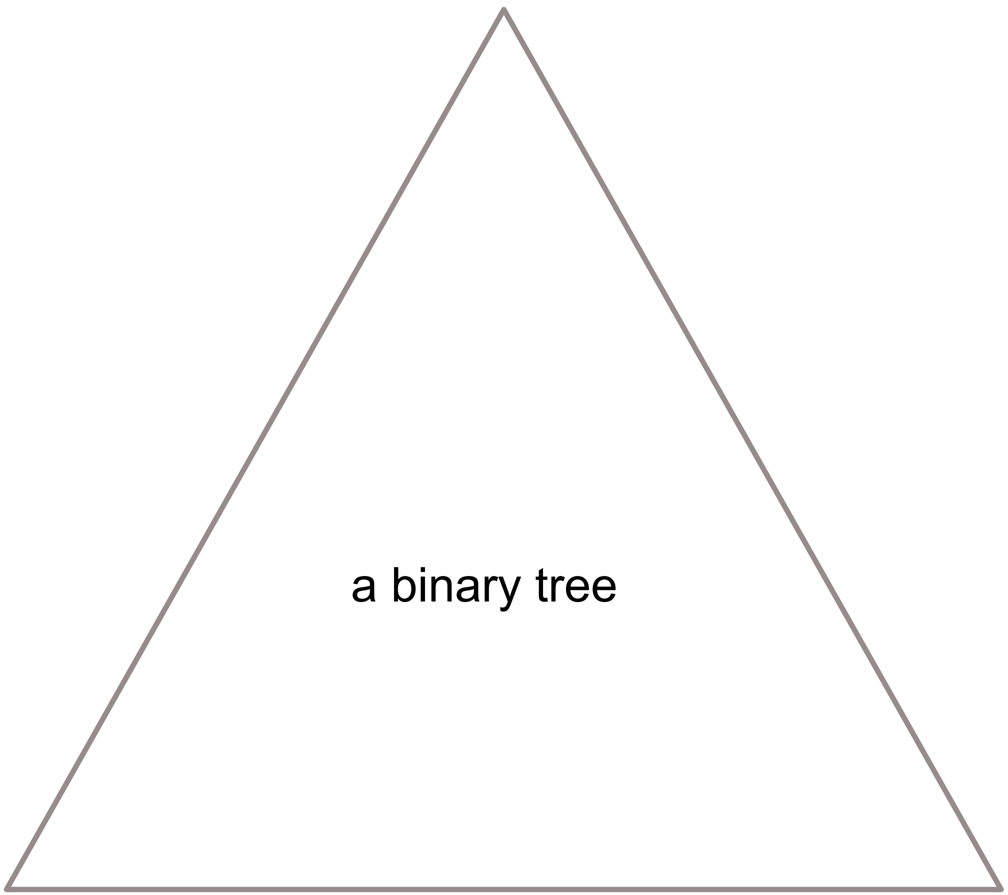
max #of nodes: (Perfect tree)

$2^0 + \ldots + 2^h = 2^{h+1} - 1$

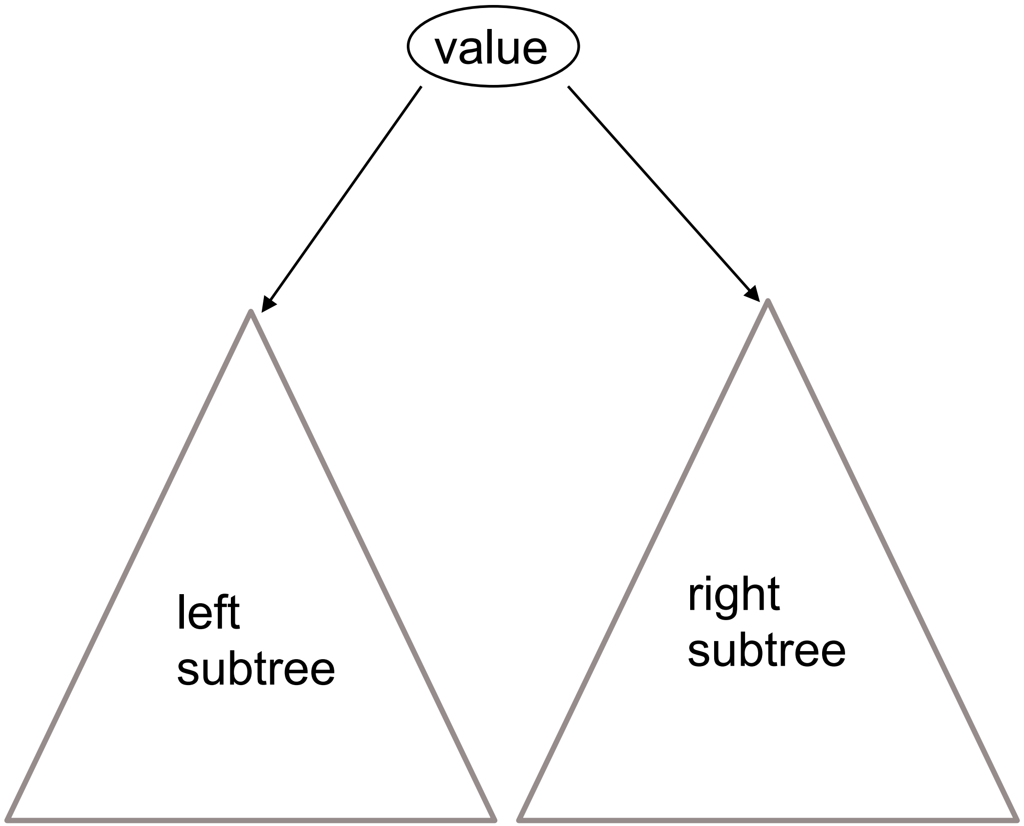
Complete binary tree

Every level, except last, is completely filled, nodes on bottom level as far left as possible. No holes.

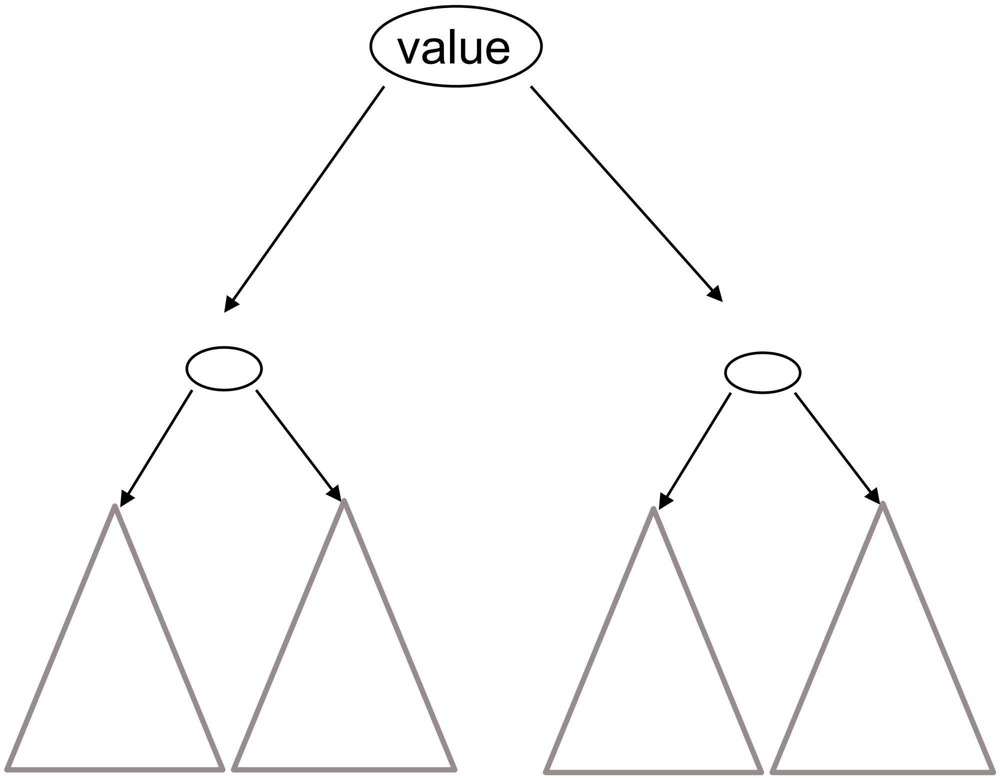
Trees are recursive



Trees are recursive



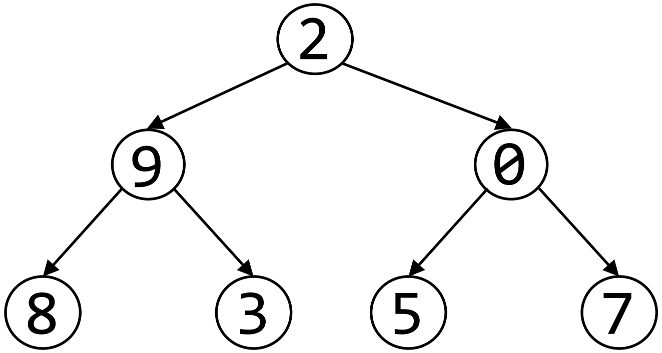
Trees are recursive



Trees are recursive

Binary Tree

Left subtree, which is also a binary tree



Right subtree (also a binary tree)

Trees are recursive

A binary tree is either `null`

or an object consisting of a value, a left binary tree, and a right binary tree.

A Recipe for Recursive Functions

Base case:

If the input is “easy,” just solve the problem directly.

Recursive case:

Get a smaller part of the input (or several parts).
Call the function on the smaller value(s).
Use the recursive result to build a solution for the full input.

A Recipe for Recursive Functions on Binary Trees

Base case:

an empty tree (null), or possibly a leaf

If the input is "~~easy~~," just solve the problem directly.

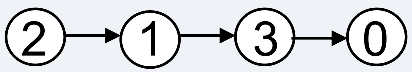
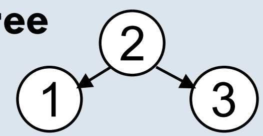
Recursive case:

~~Get a smaller part of the input (or several parts).~~

Call the function on ~~the smaller value(s).~~ each subtree

Use the recursive result to build a solution for the full input.

Comparing Searches

Data Structure	**add**(val v)	**get**(int i)	**contains**(val v)
Array 	$O(n)$	$O(1)$	$O(n)$
Linked List 	$O(1)$	$O(n)$	$O(n)$
Binary Tree 			$O(n)$

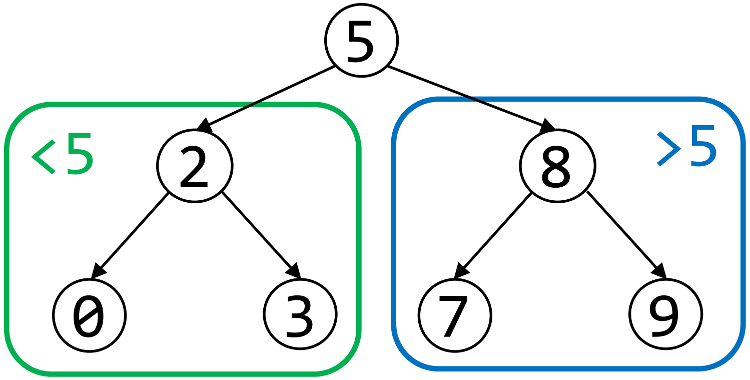
Node could be *anywhere* in tree

Binary Search Tree (BST)

A *binary search tree* is a binary tree with a **class invariant**:

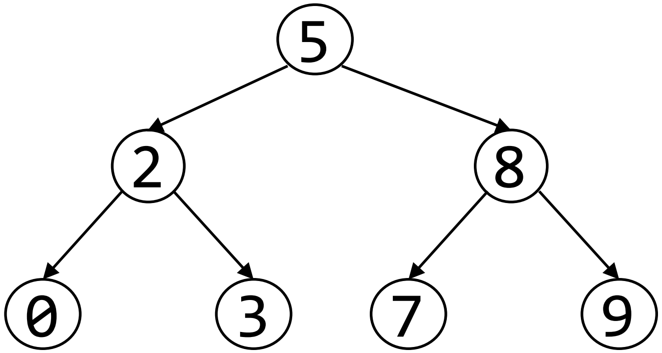
- All nodes in the left subtree have values that are less than the value in that node, and
- All values in the right subtree are greater.

(assume no duplicates)



Demo

Binary Search Tree (BST)



Contains:

- Binary tree: two recursive calls: O(n)
- BST: one recursive call: O(height)

BST Insert

To insert a value:

- Search for value
- If not found, put in tree where search ends

Example: Insert month names in chronological order as Strings, (Jan, Feb…). BST orders Strings alphabetically (Feb comes *before* Jan, etc.)

BST Insert

insert: January

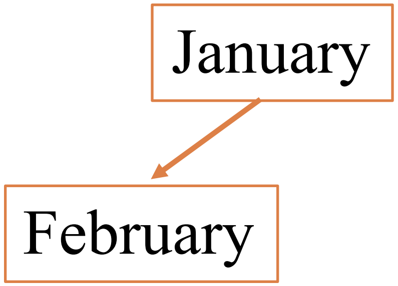
BST Insert

insert: February

January

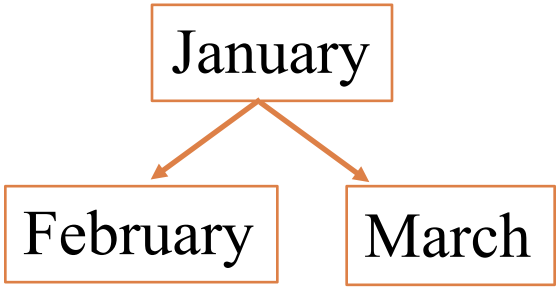
BST Insert

insert: March

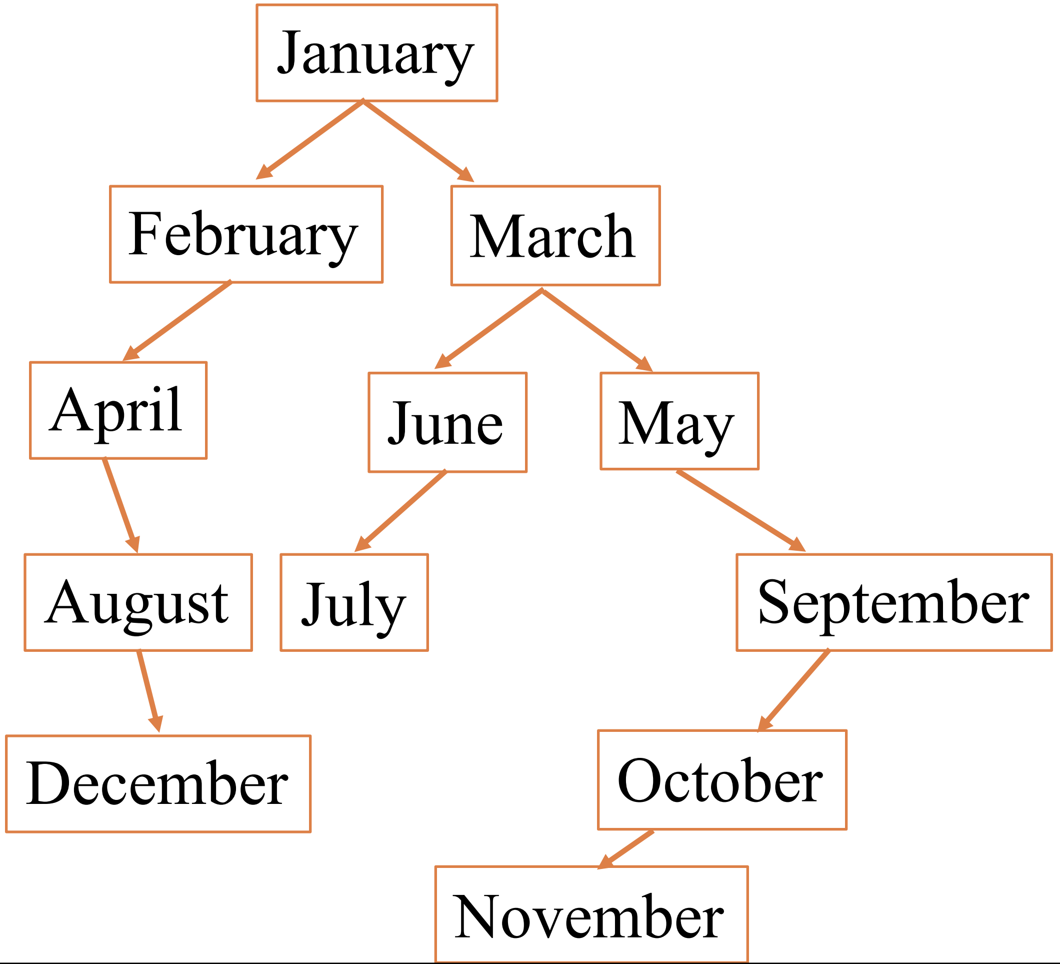


BST Insert

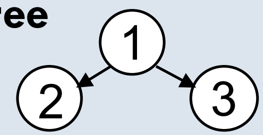
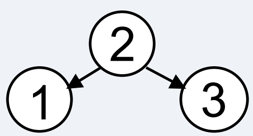
insert: April…



BST Insert


January
February
March
April
June
May
August
July
September
December
October
November

Comparing Data Structures

Data Structure	add(val x)	get(int i)	contains(val x)
Array [2, 1, 3, 0]	$O(n)$	$O(1)$	$O(n)$
Linked List 2 → 1 → 3 → 0	$O(1)$	$O(n)$	$O(n)$
Binary Tree 			$O(n)$
BST 	$O(height)$		$O(height)$

How big could height be?

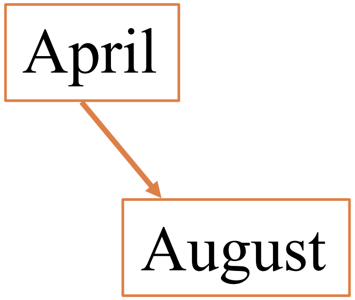
Worst case height

Insert in alphabetical order…

April

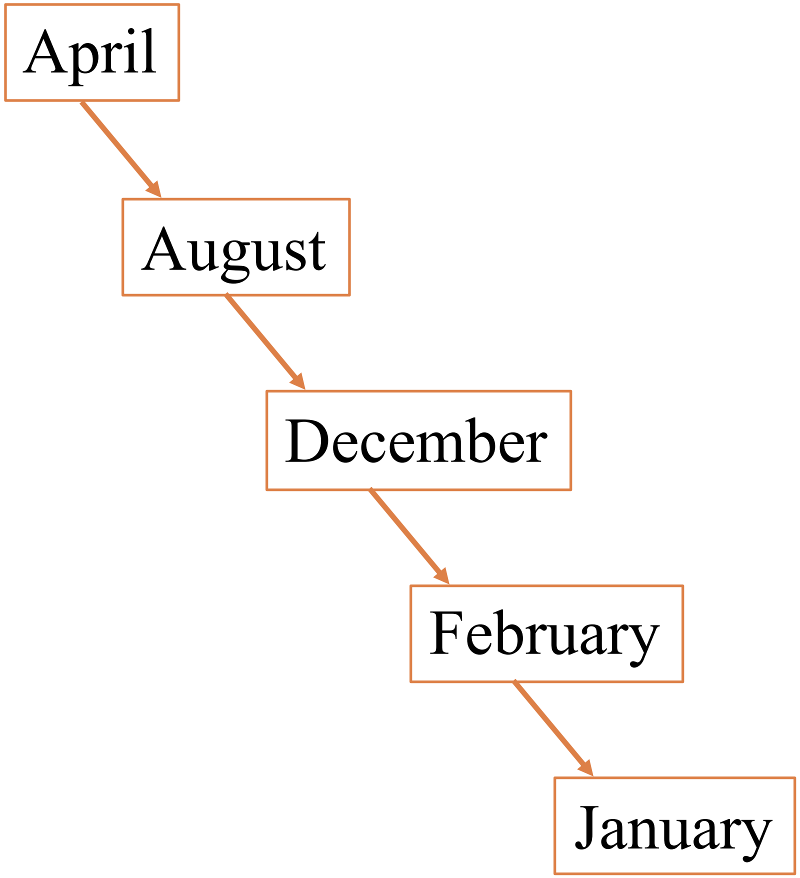
Worst case height

Insert in alphabetical order…



Worst case height

Insert in alphabetical order…



Tree degenerates to list!

Need Balance

- Takeaway: BST search is O(n) time
  - Recall, big O notation is for worst case running time
  - Worst case for BST is data inserted in sorted order

- **Balanced binary tree:** subtrees of any node are about the same height
  - In balanced BST, search is O(log n)
  - Deletion: tricky! Have to maintain balance
  - [Optional] See JavaHyperText "Extensions to BSTs"
  - Also see CS 3110