Week 5 Notes

Prof Bill - Apr 2018

First, we'll cover leftovers from week 4… Quicksort.

Then… Week 5 notes are:

A. Midterm Preview

B. Lightning2 Lecture

C. Binary Search Tree (BST) intro

thanks… yow, bill

A. Midterm Preview

Midterm details…

- ❏ When: Thu Apr 26, 2018 @ 2:00 pm
- ❏ Where: Wentz Science Center, Room 104, our regular classroom
- ❏ How much: It’s worth 25 points, 25% of your grade
- ❏ How long: 70 minutes
- ❏ What to bring: Bring 1 side of 1 page of notes to the exam
- ❏ What NOT to bring: Sorry, no calculators or gizmos of any kind

The midterm will cover anything that we have covered in

- ➔ Textbook - almost exclusively Muganda
- ➔ Lecture - weekly notes/outlines are on our class website
- ➔ Homework #1 - #4 - my solution on the k: drive
- ➔ Programs #1, #2 - my solution on the k: drive

In this preview (and for the Midterm), I leaning heavily on my program assignments and class notes/outlines for each week. They’re at the class website:

wtkrieger.faculty.noctrl.edu/csc210-spring2018/

Topics we have covered so far include:

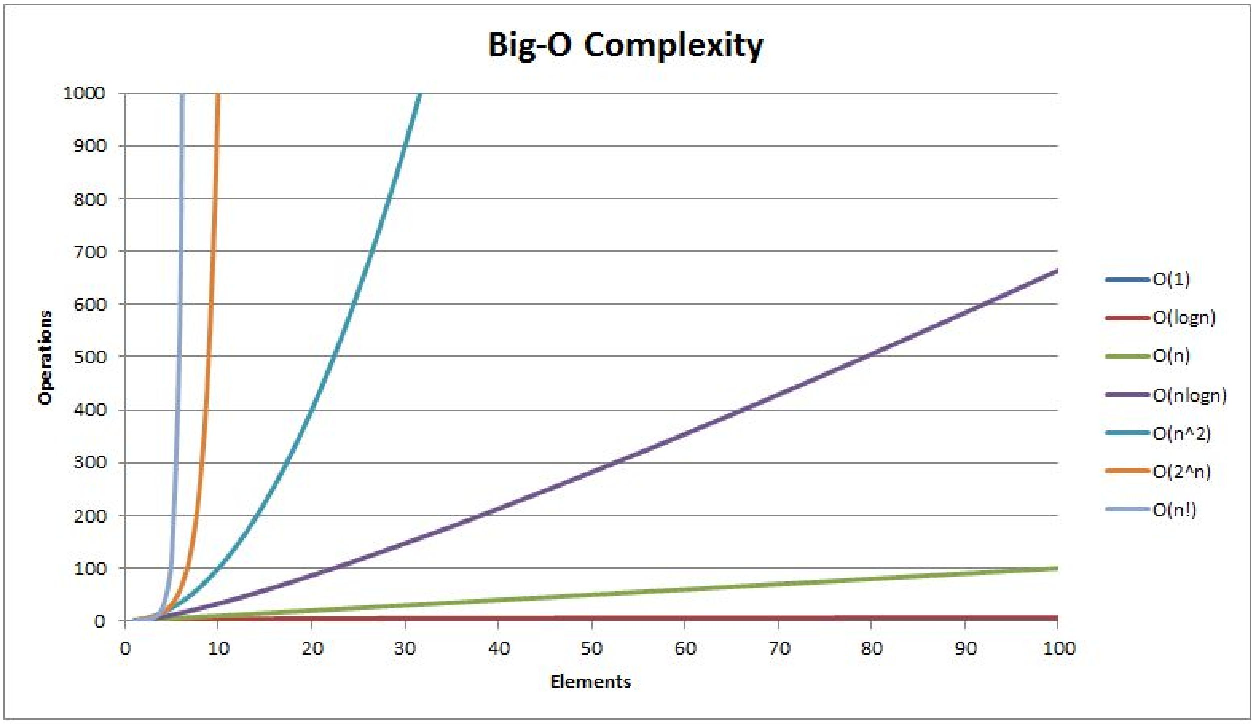
Week	Topic	Book/Etc
1	Java/OOP review	Muganda Ch 1-6, 8, 10
	C programming, linked list	Program #1 Wheel of Decision
2	Arrays and ArrayList	Muganda Ch 7
	Linked Lists	Muganda Ch 20
	Stacks and Queues	Muganda Ch 2
3	Java Collections Framework (JCF)	Muganda Ch 19.1-19.2
	JavaFX	Muganda Ch 15
	JCF: Sets, Maps, etc	Muganda Ch 19.3-19.6
	Hash tables	Muganda Ch 19.3-19.4 + my notes
	JavaFX, JCF, gui design	Program #2 GUI of Decision
4	Recursion	Muganda Ch 16
	Sort + search	Muganda Ch 17.1-17.2
	Algorithm analysis, Big-O	Muganda Ch 17.3 + my notes

B. Lightning2 Lecture

Preview of the 2nd half of 210 in 15 minutes or less. Huzzah!

Lightning1

Remember the Lightning Lecture from our first class?
Topics were: array, Big-O, linked list, hash table



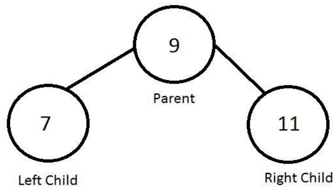
/ Note - after my favorite chart, it's all 2nd half… nothing on the midterm exam */*

The 2nd half will feature: trees, graphs, heaps, and some cool algorithms.
Read on!

Trees

Trees have nodes, like a linked list. So, tree structures are recursive/self-referential.

Binary Search Tree (BST) is popular and simple.
A doubly-linked list node has: 1) data, 2) next and 3) prev node pointers.
BST Tree nodes have: 1) data, 2) left and 3) right node pointers.



The BST trick (so you can sort and search quickly)... for each node:

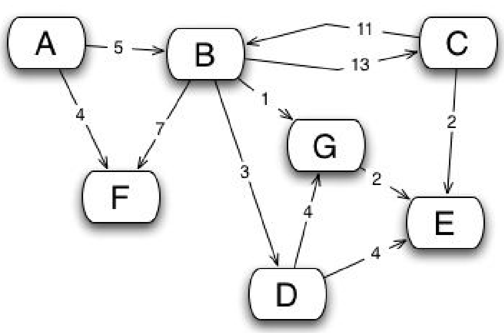
- Left child (key) < parent node (key)
- Right child > parent node

Other trees:

- ➔ **AVL tree**, **Red-black tree** - two tree schemes to try and keep BST balanced
- ➔ **B-tree** - trees where nodes have more than two children

Graphs

Graphs have nodes as well. It's a little more complicated because these nodes are used to model connectivity in a graph. Graph nodes are connected by edges.



Graphs are used to model many things: networks, circuits, maps, mazes, etc.
Graph algorithms allow us to search graphs, find shortest paths, etc.

The Princeton text rocks (but it goes deep fast):

algs4.cs.princeton.edu/40graphs/

A better intro may be this (cool) lecture from Rutgers CS:

www.cs.rutgers.edu/~mlittman/courses/cs105-06b/lectures/10graphs.pdf

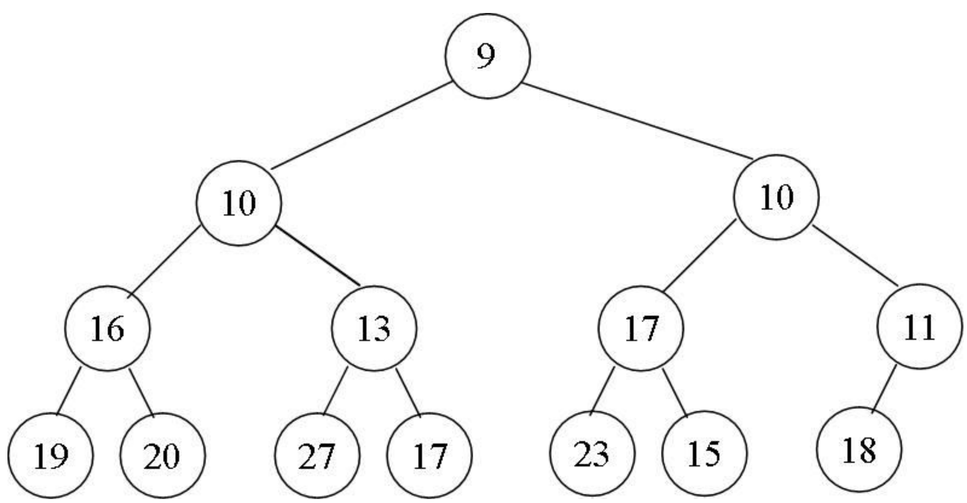
Heap, Priority Queue

A Priority Queue is a queue that isn't FIFO. It's a queue that's ordered by an object's priority.

en.wikipedia.org/wiki/Priority_queue

There are numerous important priority queue applications: CPU job scheduling, printer job scheduling, search algorithms, etc.

A heap is a (very cool) data structure that efficiently implements a priority queue.
It looks like a tree, and it is. But it's usually implemented using an array. (yow!)

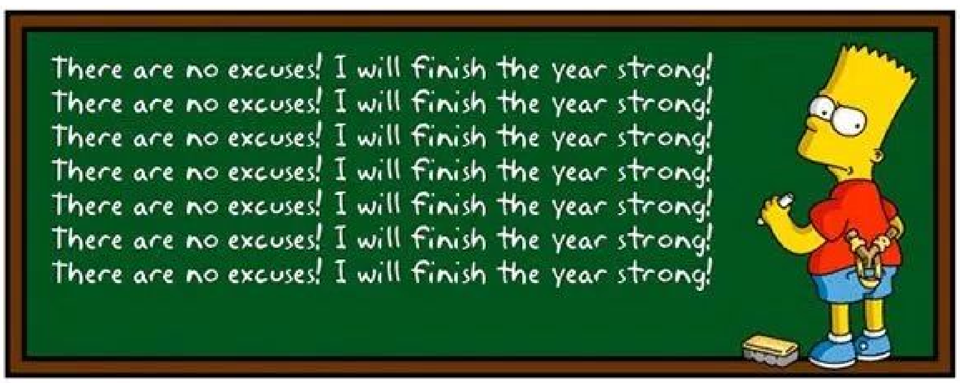


Cool algorithms

The more sophisticated data structures in the 2nd half being with them, some cool algorithms:

- ➔ Shortest path in a graph
- ➔ Breadth-first and depth-first traversal of a graph
- ➔ Inorder, preorder, postorder traversal of graph
- ➔ Finding cycles in graph
- ➔ Min spanning tree in a graph
- ➔ BST search
- ➔ Balanced tree rotations
- ➔ Priority queue removeMin()

And more!
thanks… yow, bill

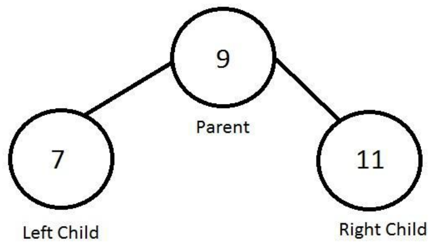


Source: fromcaterpillarstobutterflies.com/inspiration/top-10-quotes-finishing-strong/

C. Binary Search Tree (BST)

** Book: **Muganda 22.1-22.2**
** Online: **Princeton**, algs4.cs.princeton.edu/32bst
** Online: This **animation** is great: www.cs.usfca.edu/~galles/visualization/BST.html

Binary Search Tree (BST) is built out of nodes, ala linked lists.
Each node has data (key, value) and two node pointers: left and right.

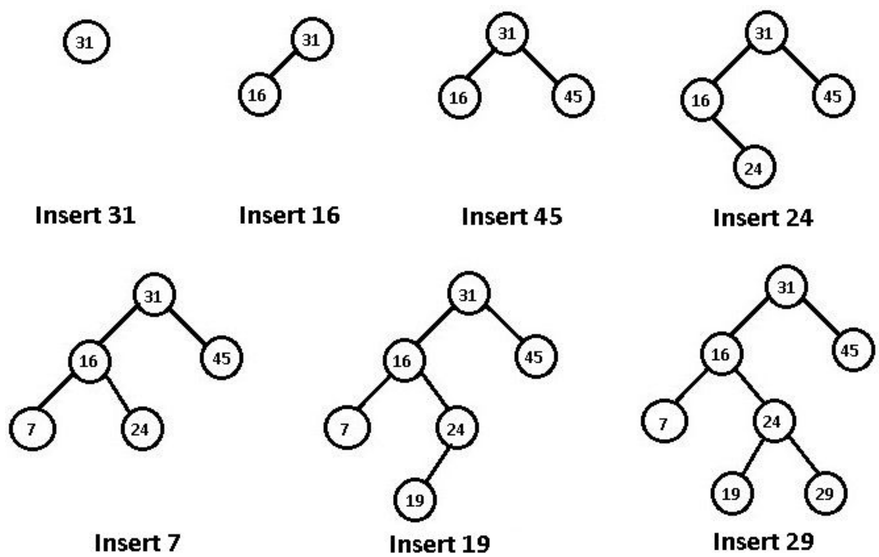


Source: cppbetterexplained.com/binary-search-trees/

Here's **the magic**… for every node:

Its left child is less than (<) and the right child is great (>).

That's it. Let's build one.
An empty BST is root = null (not shown below).
Below: the root is the first node added; in this case 31.



.

Source: csegeek.com/csegeek/view/tutorials/algorithms/trees/tree_part2.php

Notice in our example:

- The root doesn't change when adding to the tree
- Every new node is added as a leaf

There are 3 important methods in the BST ADT:

1. put(K key, V value) - we just did this
2. V get(K key)
3. V remove(K key)

With **put()** - Often, we just show the keys. The value is there or it's just keys (like a set)

Here's **get()** pseudocode… it's a recursive search:

```
get( K key) {
   return getNode( root, key)              // start at root
}

V getNode( Node n, K key) {
   if n == null then return null            // NOT found

   if key == node.key return node.value    // FOUND

   if key < node.key
        return getNode( node.left, key) // look LEFT
   else
        return getNode( node.right, key)    // look RIGHT
}
```

We'll do **remove()** later. It's a little more involved.

Questions:

- What is the average Big-O for put and get methods? Worst case?

- Is it possible for BST to get VERY unbalanced? Example, please.

- TreeMap and TreeSet in JCF are (supposedly) red-black (balanced) trees.

- Ready for Homework #5?