CS280 – Trees
Febuary 10, 2016

http://azrael.digipen.edu/~mmead/www/Courses/CS280/Trees.html

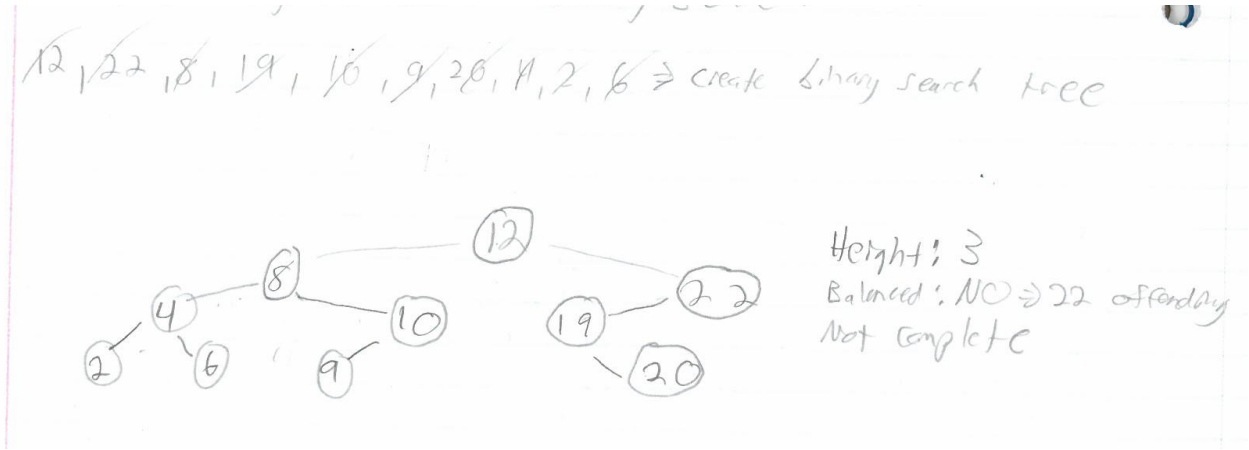
Assignment Notes

- Make sure and compile w/ -g for Dr. Memory so that you can see call stack
- If memory allocation fails, just abort
- Do NOT use alloca!
- **Midterm Wednesday. Feb. 24th**

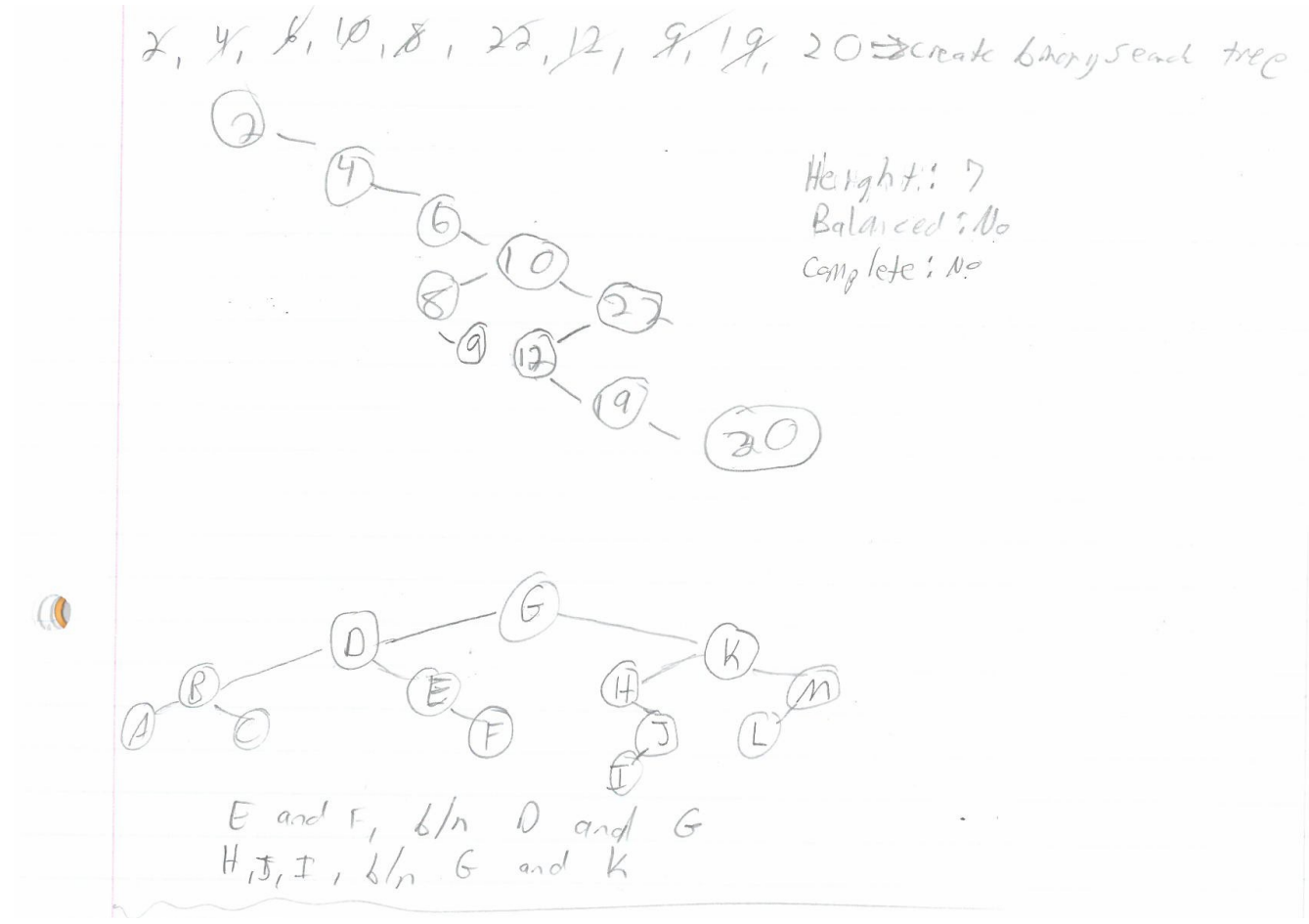
Trees

- Balancing the tree is similar to sorting a linked list
- Build a tree recursively
  - Build the node
  - Build left
  - Build right
- Tree with one node has a height of 0
- For almost all tree functions, we will be using recursion to traverse the tree
- Don't really gain anything by passing notes by reference instead of by value
- O(n) to find the height
- Breadth-first (aka level-order traversal)
  - Traverse row by row instead of following the links
- Depth-first
  - Go along the links (columns)
- Recursion isn't necessarily the best way to traverse the tree breadth-first

    - Use a queu
- For level-order traversal algorithm using queu
    - Popping from the front
    - Pushing on the back
- Directory explorers are usually trees
    - Drag and drop a node into its own tree
    - Detect that case, don't do it
- Binary search a tree
    - Sorted, so you can do a binary search



Creating a binary search tree



Creating another binary search tree with same numbers. Note how it ends up much different simply because the numbers were given in another order. This is an example where randomness actually makes the data structure BETTER!

Replacing in a Tree

- 4 Cases
  - No children (handled by left child case)
  - Left child
  - Right child
  - Two children