CS2100 - Binary Search Trees

Note Title 10/25/2013

Announcements

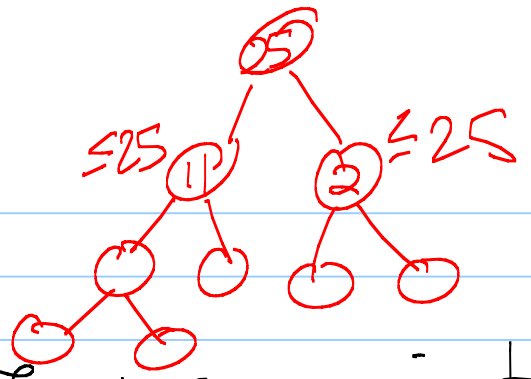
- HW due ~~tomorrow~~ Wednesday
- Next HW - be due in a week (next Tuesday) (on paper)
- No lab tomorrow - normal lecture

Last time: Priority Queues

- insert(e): add e to our data structure
- getMax(): return element with maximum key (its e)
- removeMax(): delete element with maximum key

With vectors or lists: at least 1 function takes O(n) time

Last time: Heaps



A binary tree where we maintain 2 invariants:

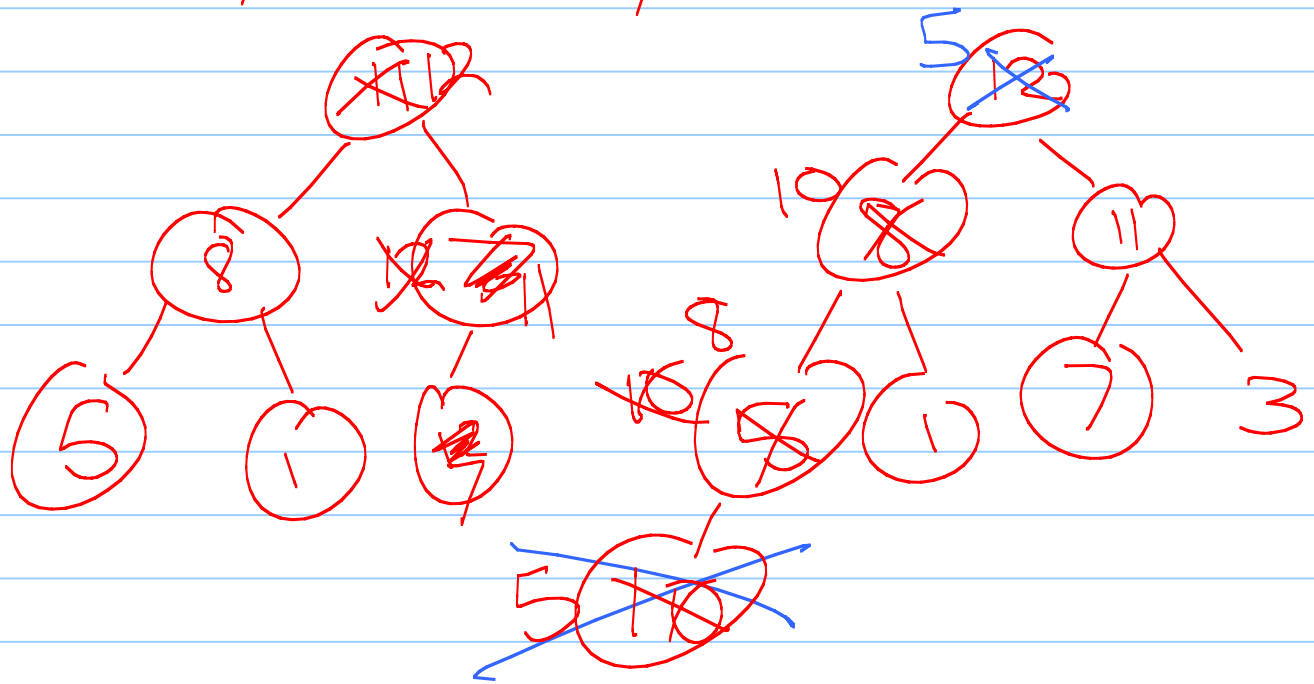
- Tree is complete.
- Any node's value is $\leq$ its parent's value.

Runtimes: $O(\log n)$

(Code is on webpage.)

Example:

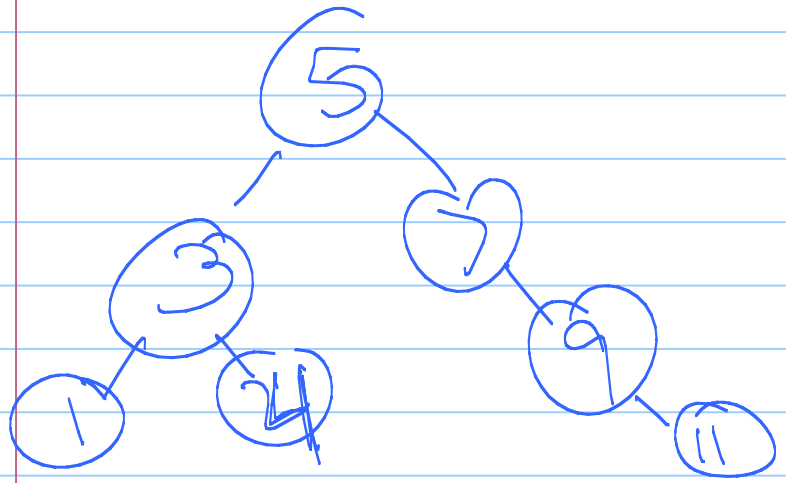
Insert: 8, 11, 7, 5, 1, 12, 3, 10

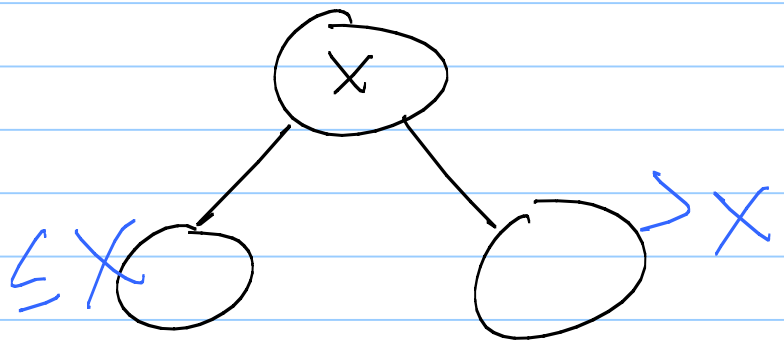


Binary Search Trees

A binary tree where we maintain the following:

The value at any node is $\geq$ its left child and $<$ its right child.





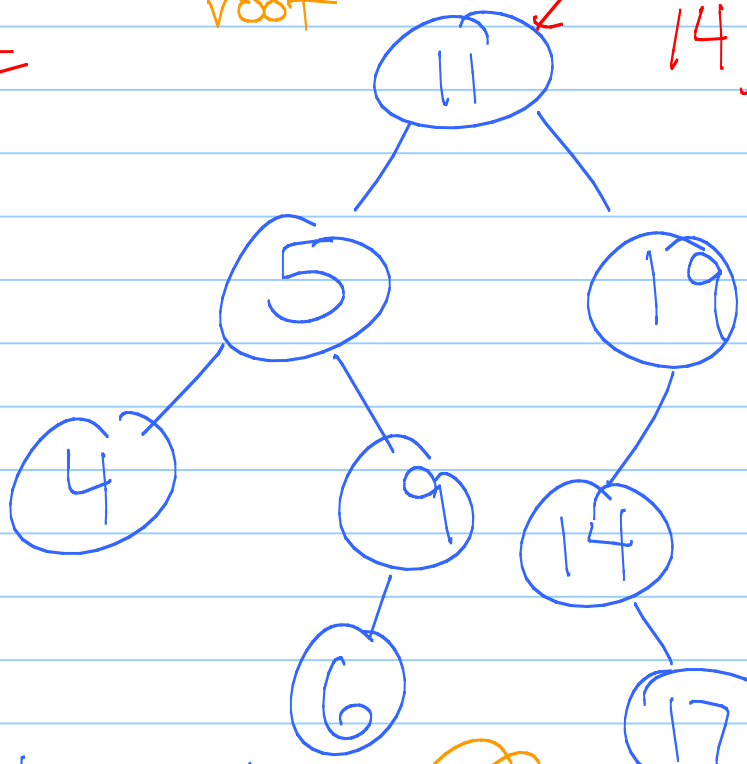
4, 6, 9, 5, 17, 14, 19, 11

root

Aside: Traversals of trees

11, 5, 4, 9, 6, 19, 14, 17

root



4, 5, 6, 9, 11, 14, 17, 19

root

Preorder

preorder(v) =
print v
recurse left
recurse right

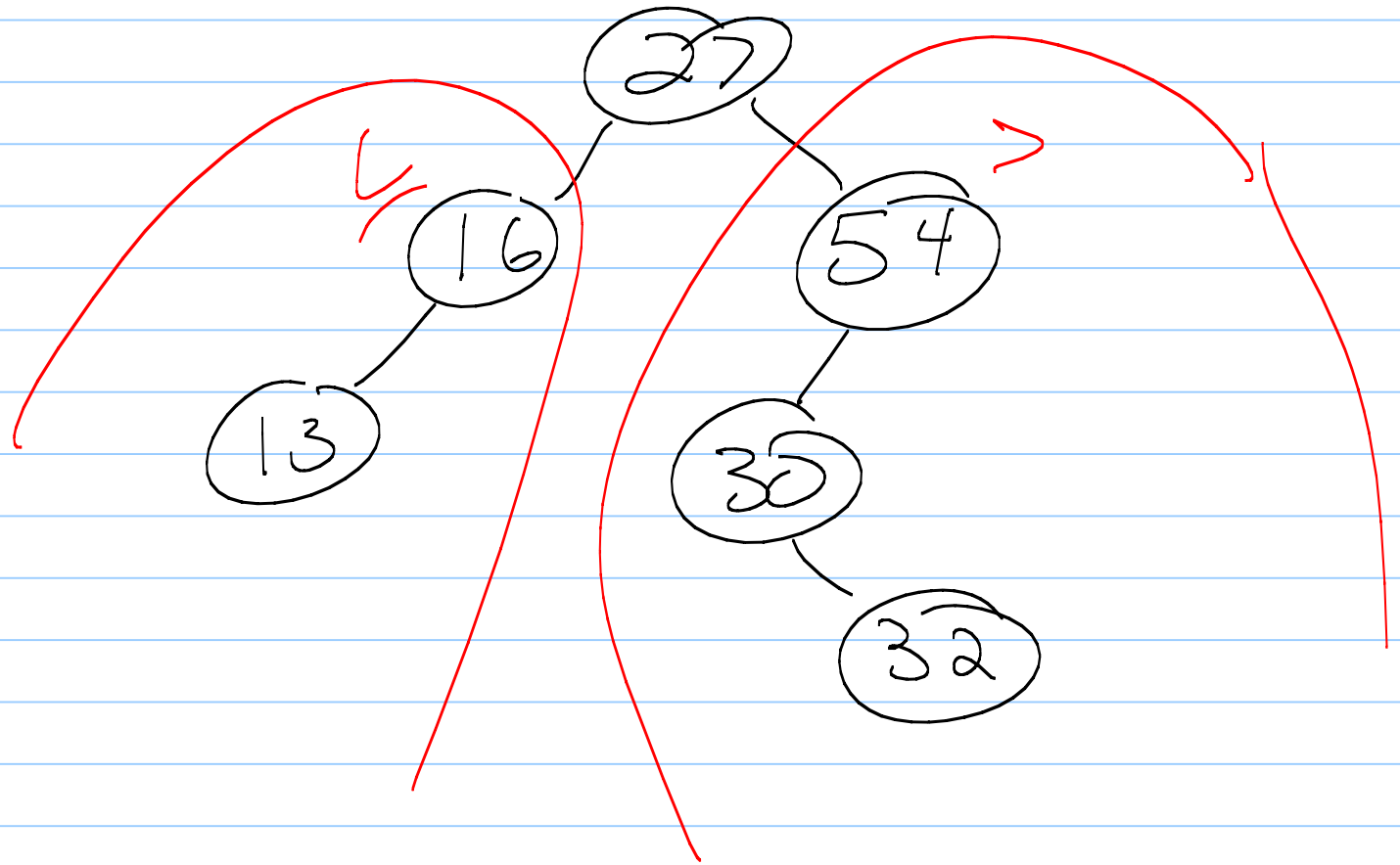
Inorder

inorder(v) = recurse left
print v
recurse right

Postorder

postorder(r) =
recurse left
recurse right
print v

Example:



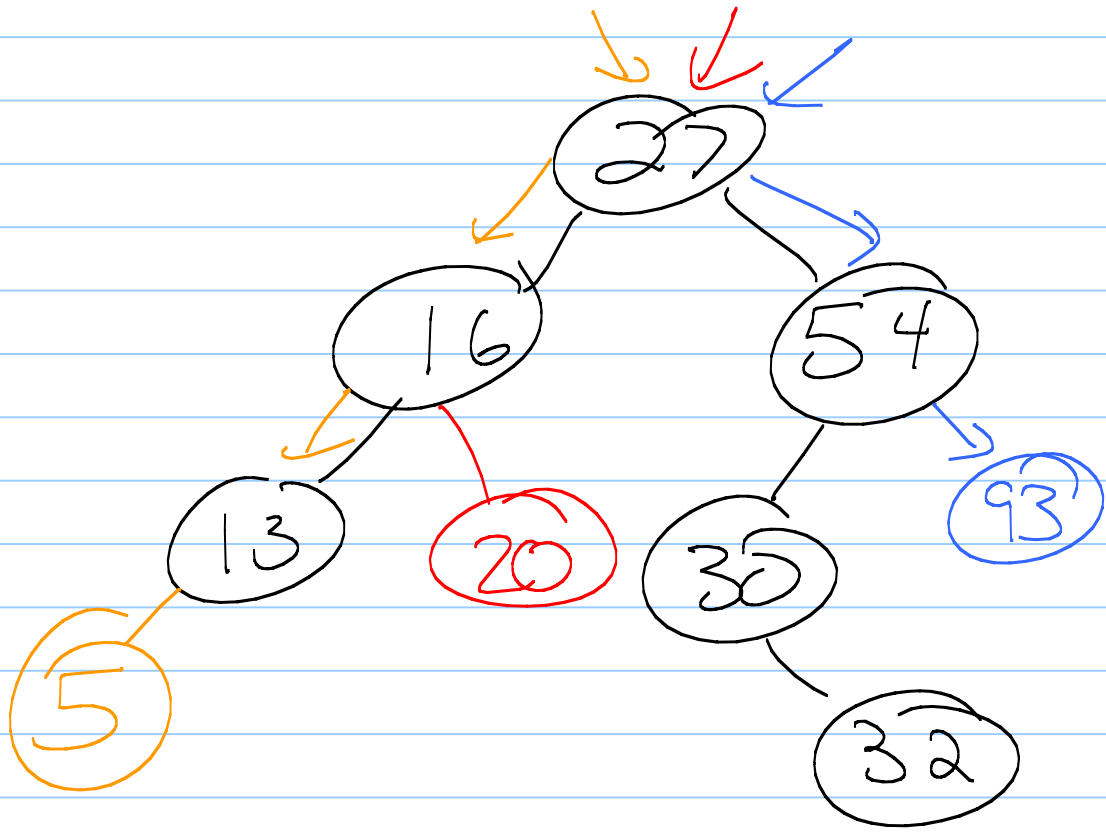
Back to BST:

Insert:

insert (20)

insert (5)

insert (93)

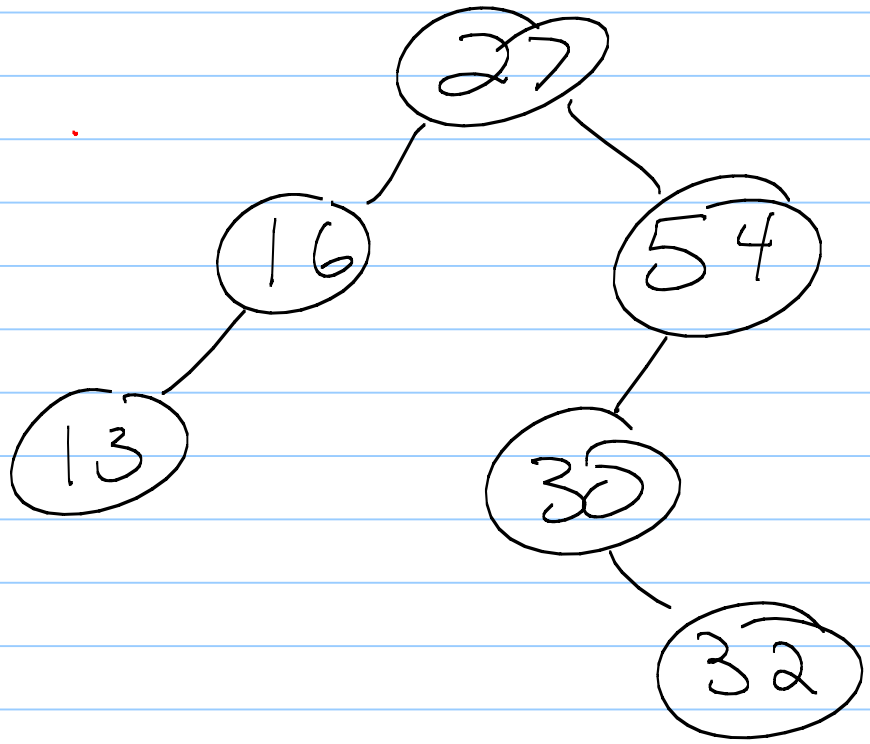


Find

check root

recurse left
if ≤ root

recurse right
if > root

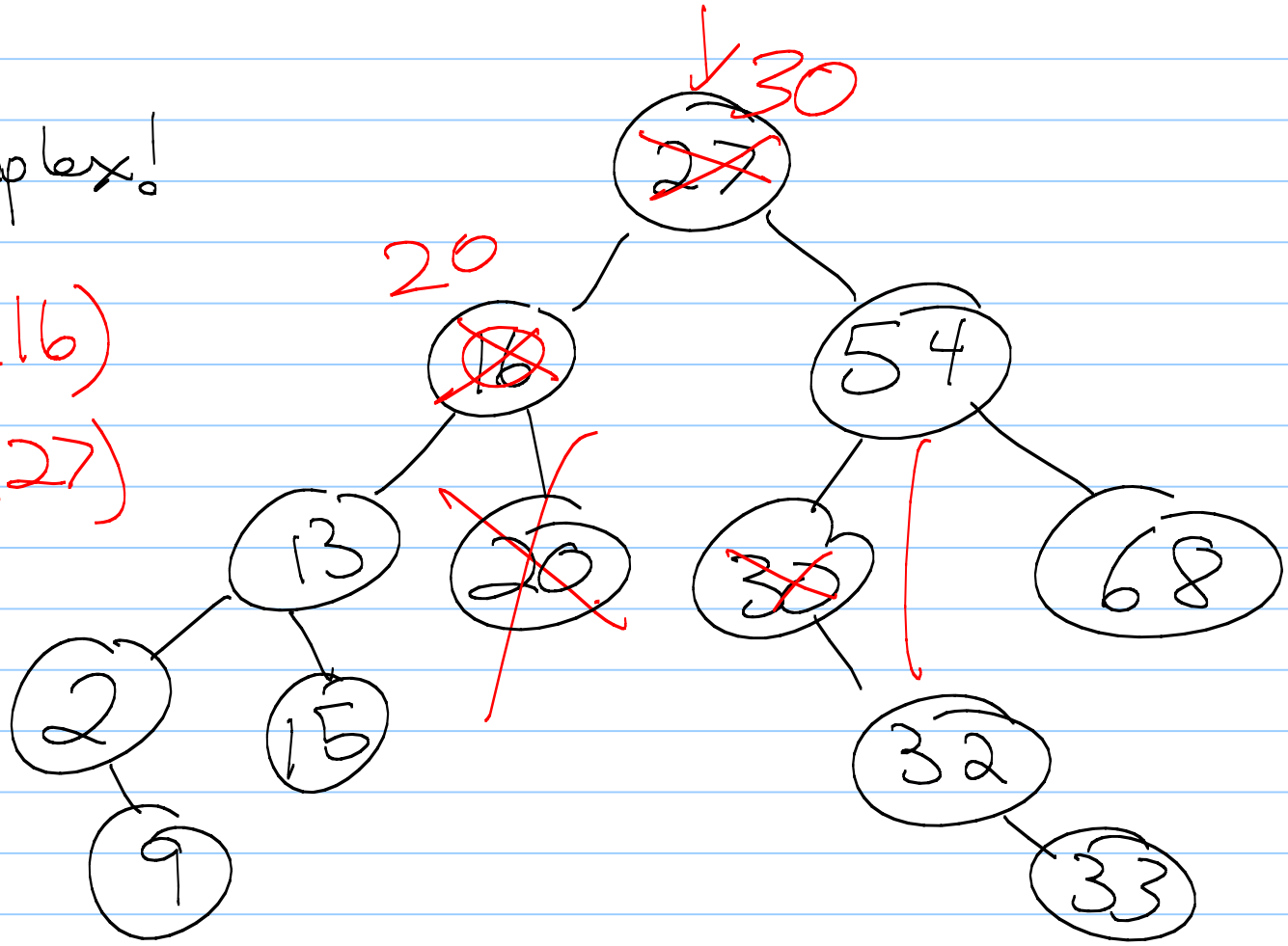


Delete:

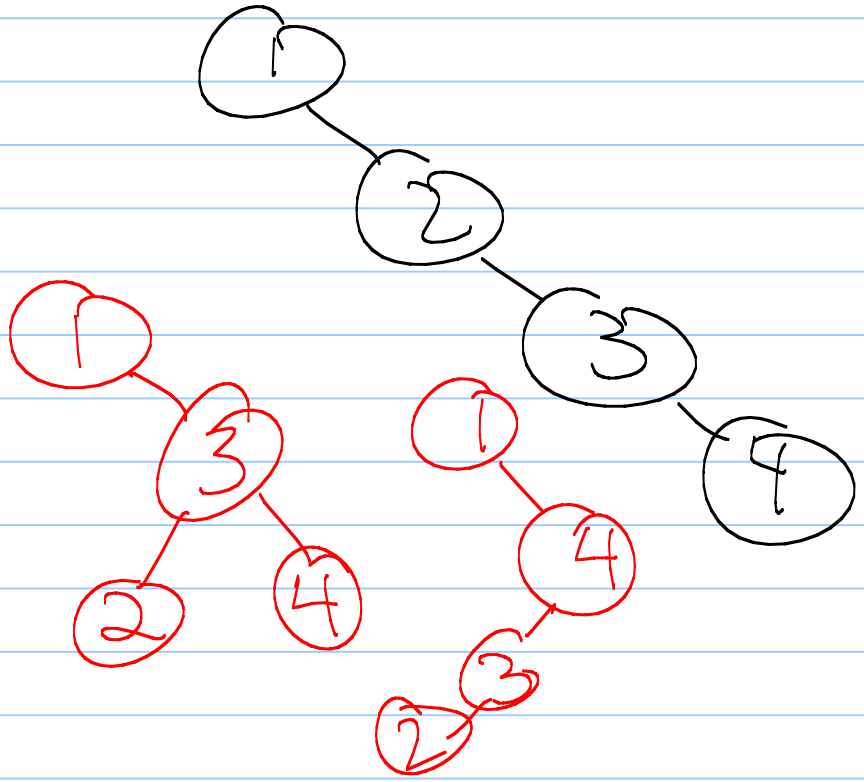
More complex!

remove (16)

remove (27)



Note: BSTs are not unique!



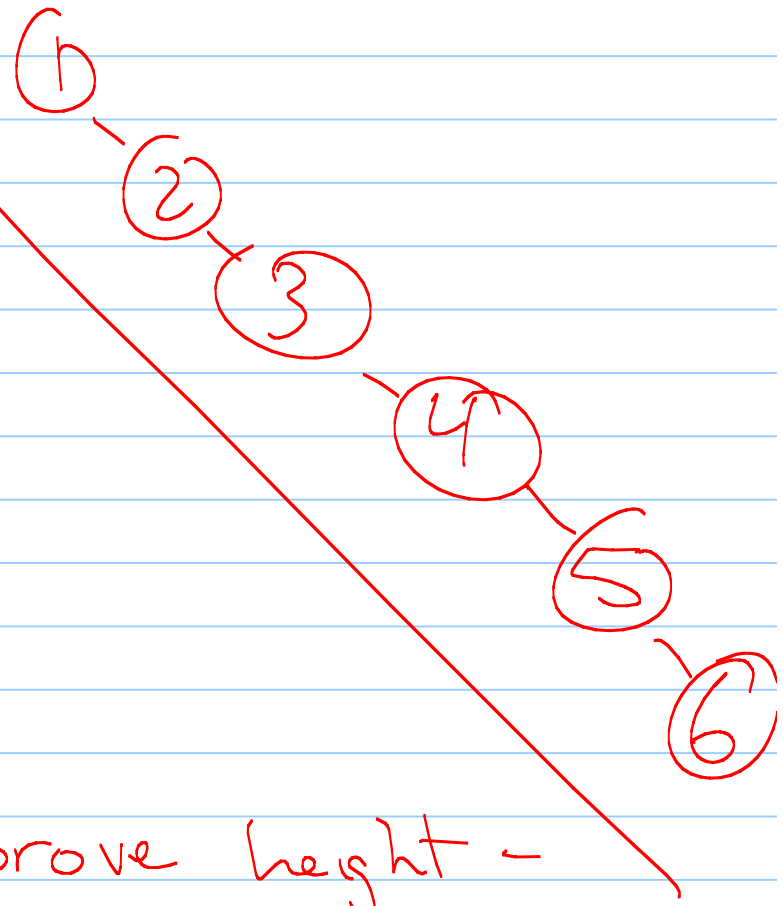
Can you make another BST with these elements?

Runtimes:

Find: O(height)

Insert: O(height)

~~Delete:~~



→ height = O(n)

(AVL trees will improve height - next week)

Code

- Will be pointer based. Why?
  - Pointers will make moving subtrees around much easier.
  - not complete tree, so array wastes space

(Need nodes, iterators, etc.)

~~Today~~ Tomorrow:

Code for generic binary trees.

BinaryTree.h will be generic — not BSTs.

BST.h will inherit from BinaryTree.h (but so will other classes.)