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
  - `get Max()`: return element with maximum key (its e)
  - `remove Max()`: 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.
Aside: Traversals of trees

**Preorder**
\[ \text{preorder}(v) = \text{print } v \text{ \( \rightarrow \)} \text{ recurse left} \text{ \( \rightarrow \)} \text{ recurse right} \]

**Inorder**
\[ \text{inorder}(v) = \text{recurse left} \text{ \( \rightarrow \)} \text{ print } v \text{ \( \rightarrow \)} \text{ recurse right} \]

**Postorder**
\[ \text{postorder}(v) = \text{recurse left} \text{ \( \rightarrow \)} \text{ recurse right} \text{ \( \rightarrow \)} \text{ print } v \]
Example:
Back to BST:

Insert:

- insert (5)
- insert (20)
- insert (93)
Find

Check root

if ≤ root

Recurse left

if > root

Recurse right
Delete:
More complex!
remove (16)
remove (27)
Note: BSTs are not unique!

Can you make another BST with these elements?
Runtimes:

- Find: $O(\text{height})$
- Insert: $O(\text{height})$
- Delete: (crossed out)

$\text{height} = O(n)$

(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.)