Announcements
Last time: Trees

Def: A tree $T$ is a set of nodes storing elements in a parent-child relationship.

$T$ has a special node $r$, called the root.
Each node (except $r$) has a unique parent.
More dfs

- child
- siblings
- leaves
- internal nodes
- rooted subtree
- descendant/ancestor
Binary Tree

- Every node has ≤ 2 children.
Depth + Height - defined recursively

\[ \text{depth} : \quad \text{depth}(r) = 0 \]
\[ \text{depth}(v) = \text{depth}(\text{parent}(v)) + 1 \]

\[ \text{height} : \quad \text{height}(\text{leaf}) = 0 \]
\[ \text{height}(v) = \max(\text{height of children}) + 1 \]
Nice trick

Can be pointers or array based!
Potential downside (of array)

Array:

How big?
Data Structure:

Priority Queue: supports the following operations

insert(e): adds element e to the data structure
removeMax(): removes maximum element
getMax(): returns maximum element

How to build?
Why?

Good if you need limited sorting.

Ex:

How?

Maintaining with list or vector:
Heaps

A binary tree where:

- for every node \( v \) (other than root), the key stored at \( v \) is \( \leq \) key stored at \( v \)’s parent

- the tree is complete: levels 0 to \( h-1 \) are full, and level \( h \) is filled in left to right order
Max Heap

```
     21
    /  \
  15   13
  /  \   /  \
14   6  2  5
 / \ / \  / \ \
2  6 10
```
**Insert**

- Insert (2)
- Insert (52)
- Insert (7)
Remove
Running times

How many comparisons / swaps?
Code for this class

- Array-Based. Why?