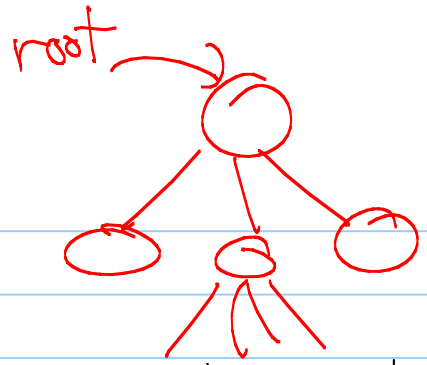
CS2100 - Heaps

Note Title 11/4/2011

Announcements

Trees



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

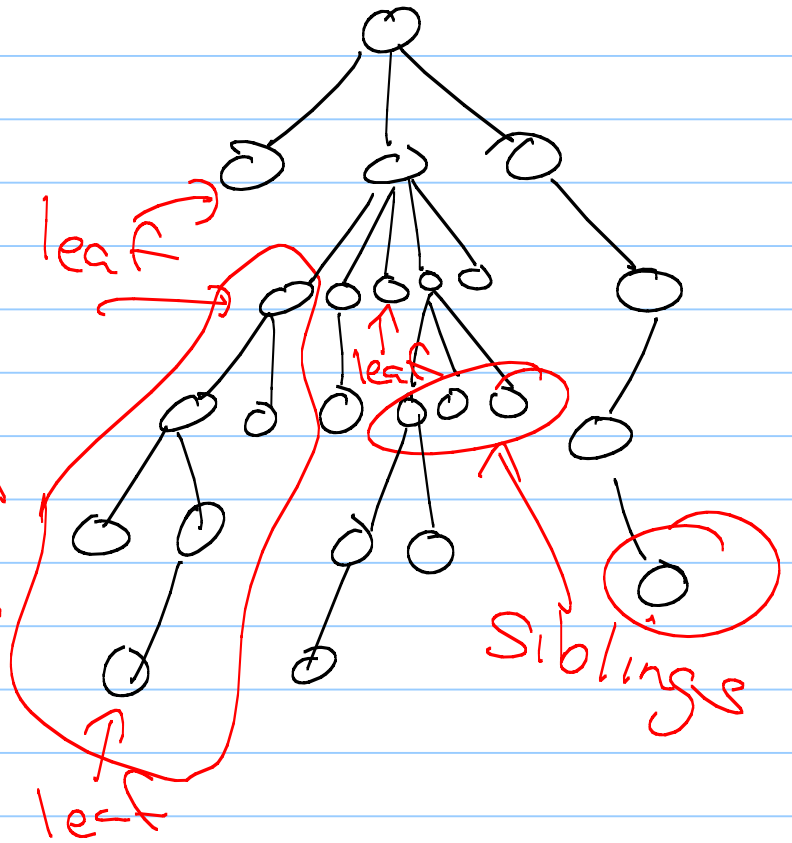
in CS

T has a special node r, called the root.

Each node (except r) has a unique parent.

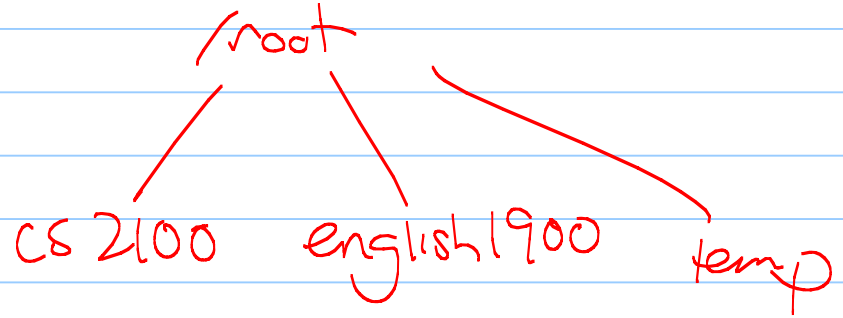
More dfs

- child
- siblings
- leaves — no children
- internal nodes — have children & are not root
- rooted subtree
- desendant / ancestor



Examples

file hierarchies

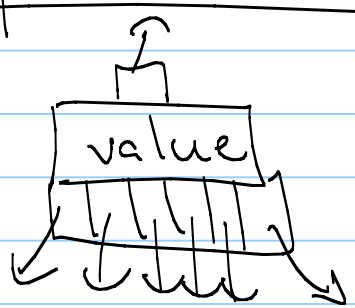


family trees...

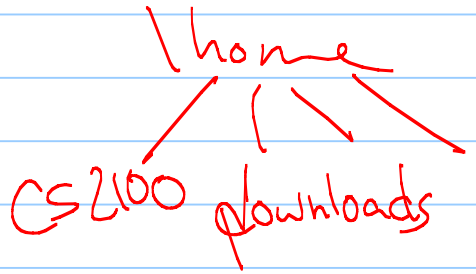
(we won't do)

General Tree Implementation

Pointer based:



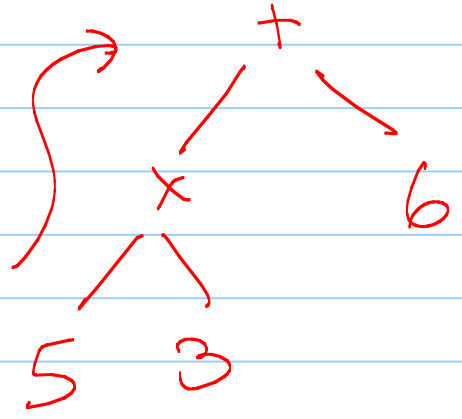
Need a list of children in each node.



Applications

Anything where relationships are more complex than linear orderings!

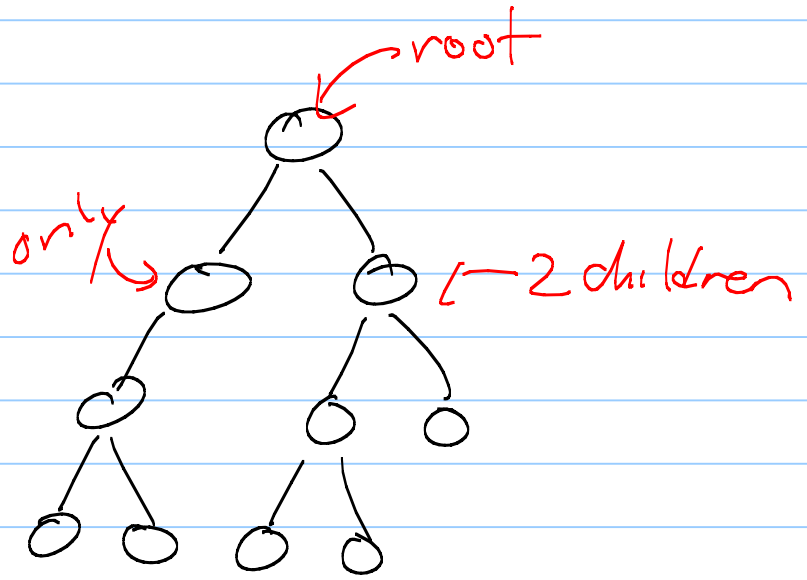
Ex:
- family tree
- file systems
- Numeric expressions
- ⋮



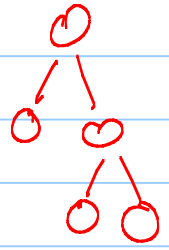
(5 x 3) + 6

Binary Tree

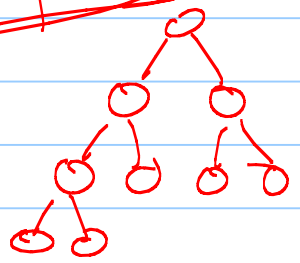
- Every node has ≤ 2 children.



Full tree: every node has 0 or 2



Complete bin tree:



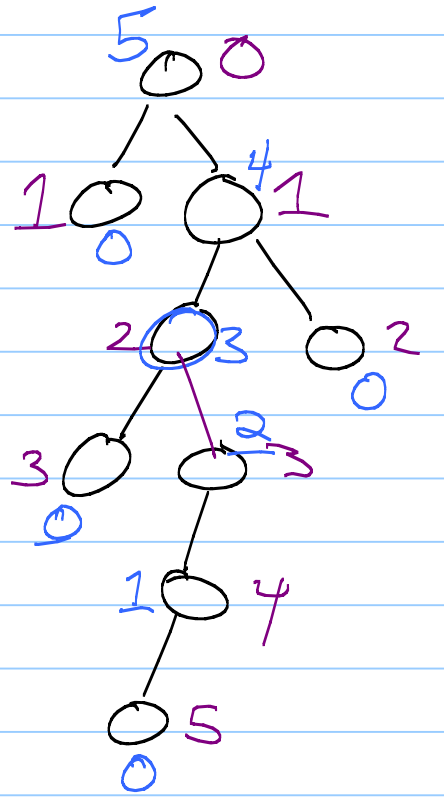
Depth & Height - defined recursively

depth: depth(r) = 0

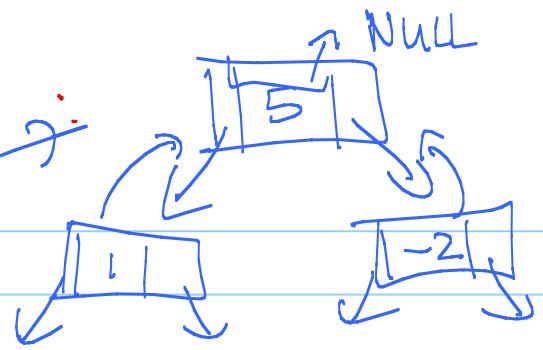
depth(v) = depth(parent(v)) + 1

height: height(leaf) = 0

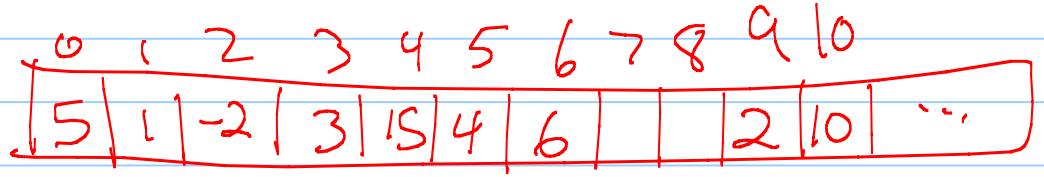
height(v) = max(height of children) + 1



Nice trick

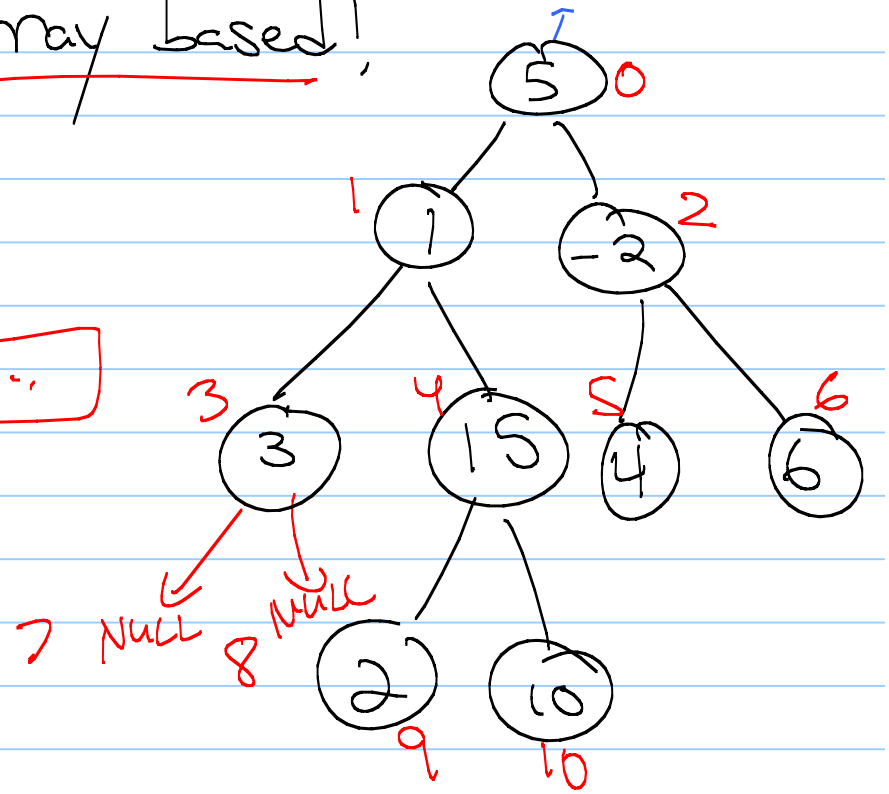


Can be pointers or array based!



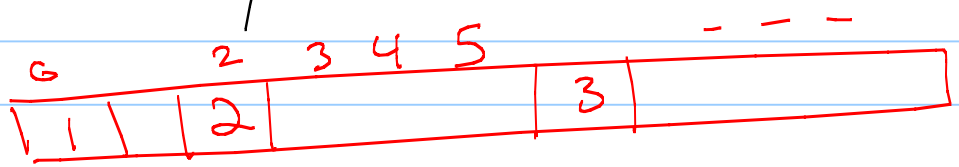
$left(i) = 2i + 1$

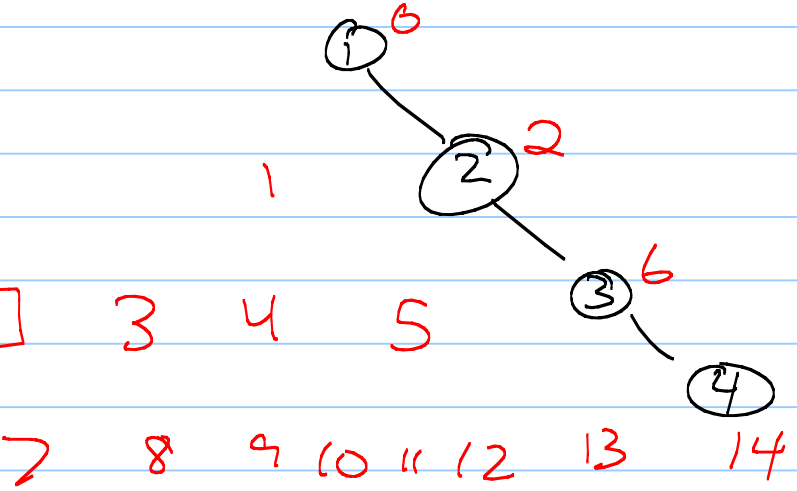
$right(i) = 2i + 2$



Potential downside (of array)

Array:





How big?

n pieces of data → 2^n size array

Uses of trees in D.S:

- Binary Search Trees
  - Balanced

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.

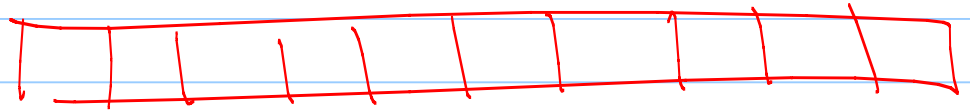
Ex:

How to implement?

Many options: Array | Vector, Linked list, tree-based

Vector implementation:

if unsorted vector:



get or find Max: linear search O(n)

insert: O(1) amortized (just push_back)

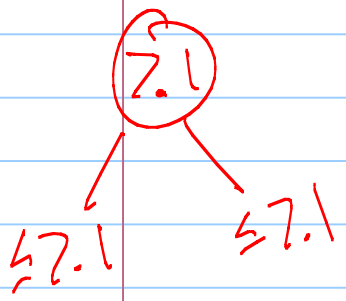
Sorted vector:

get or find Max: O(1) (look at end!)

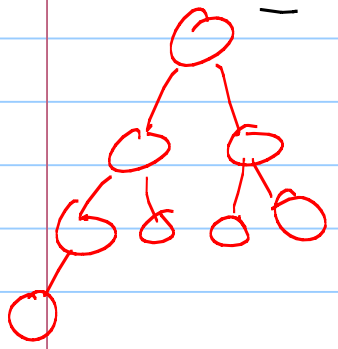
insert: find (binary search) + then insert OK!

Heaps

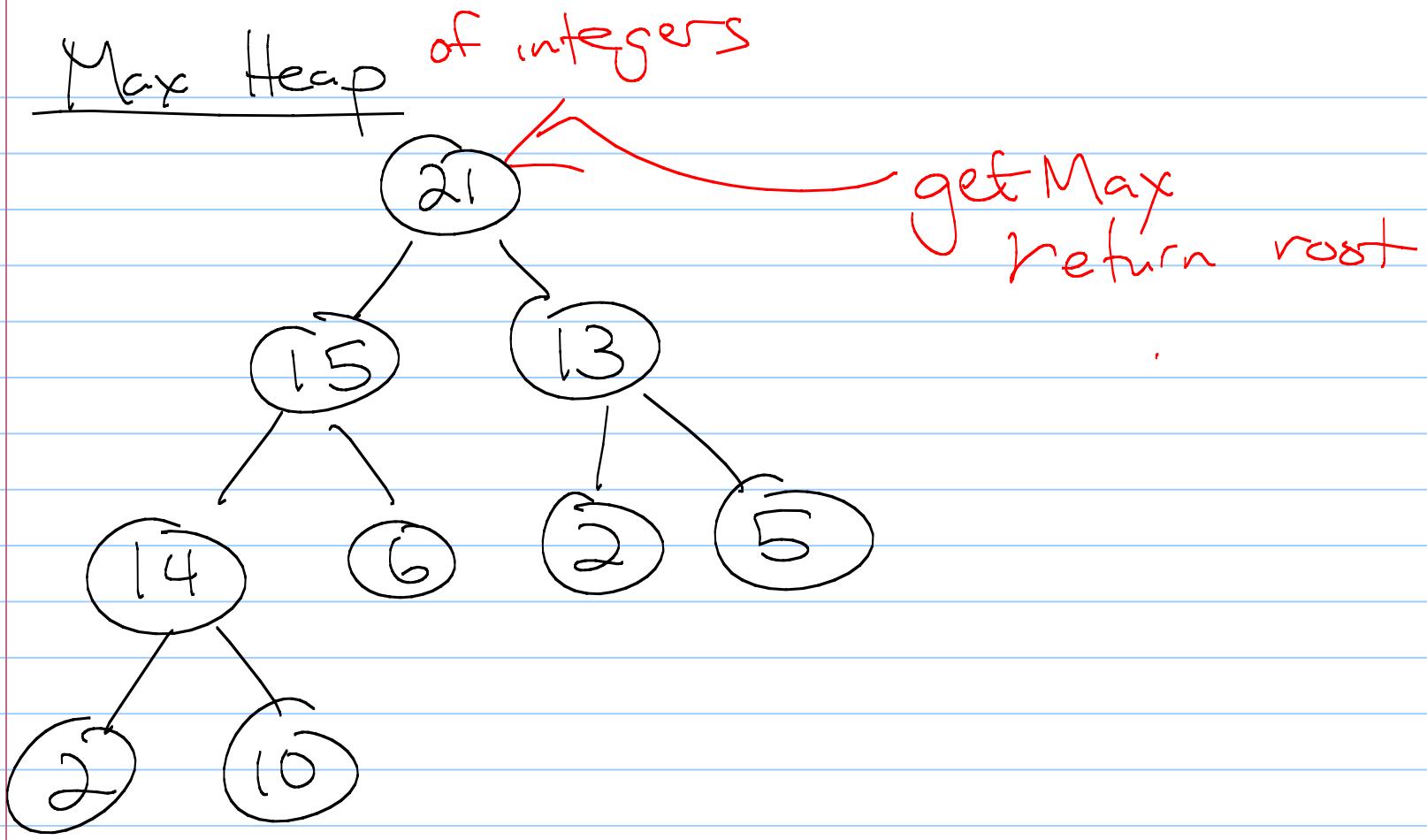
A binary tree where:



- 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

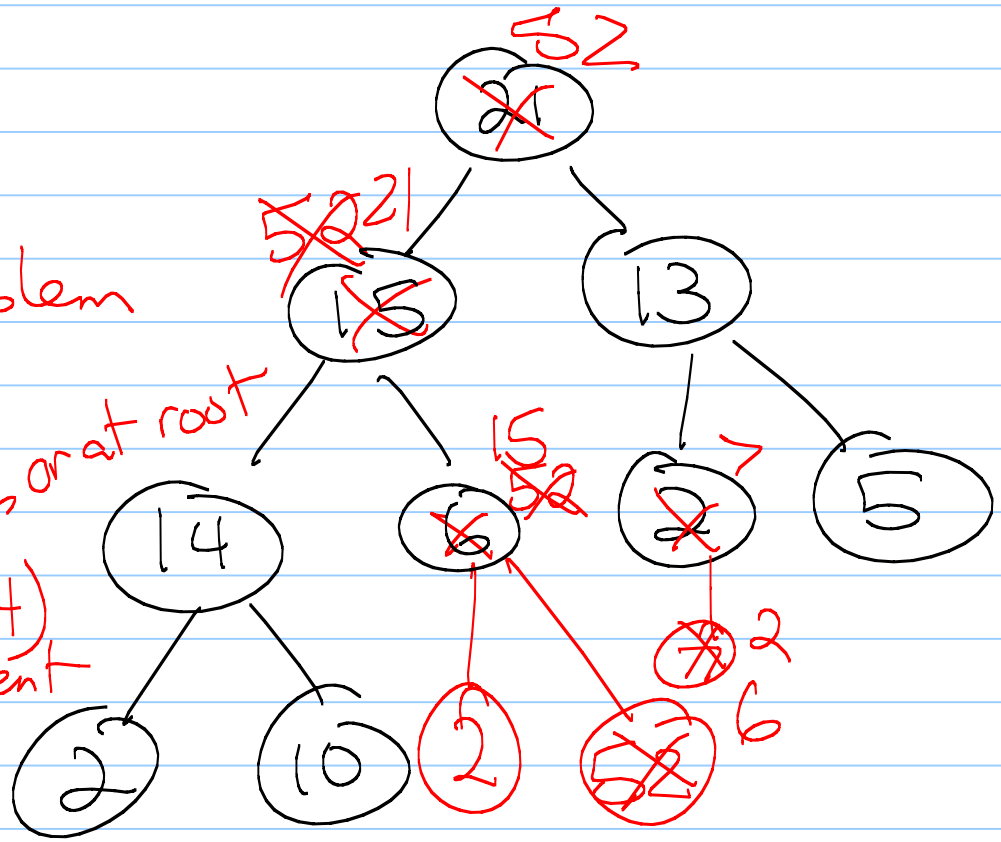

Max Heap
of integers
21
15
13
14
6
2
5
2
10
getMax
return root

Insert

insert (2)

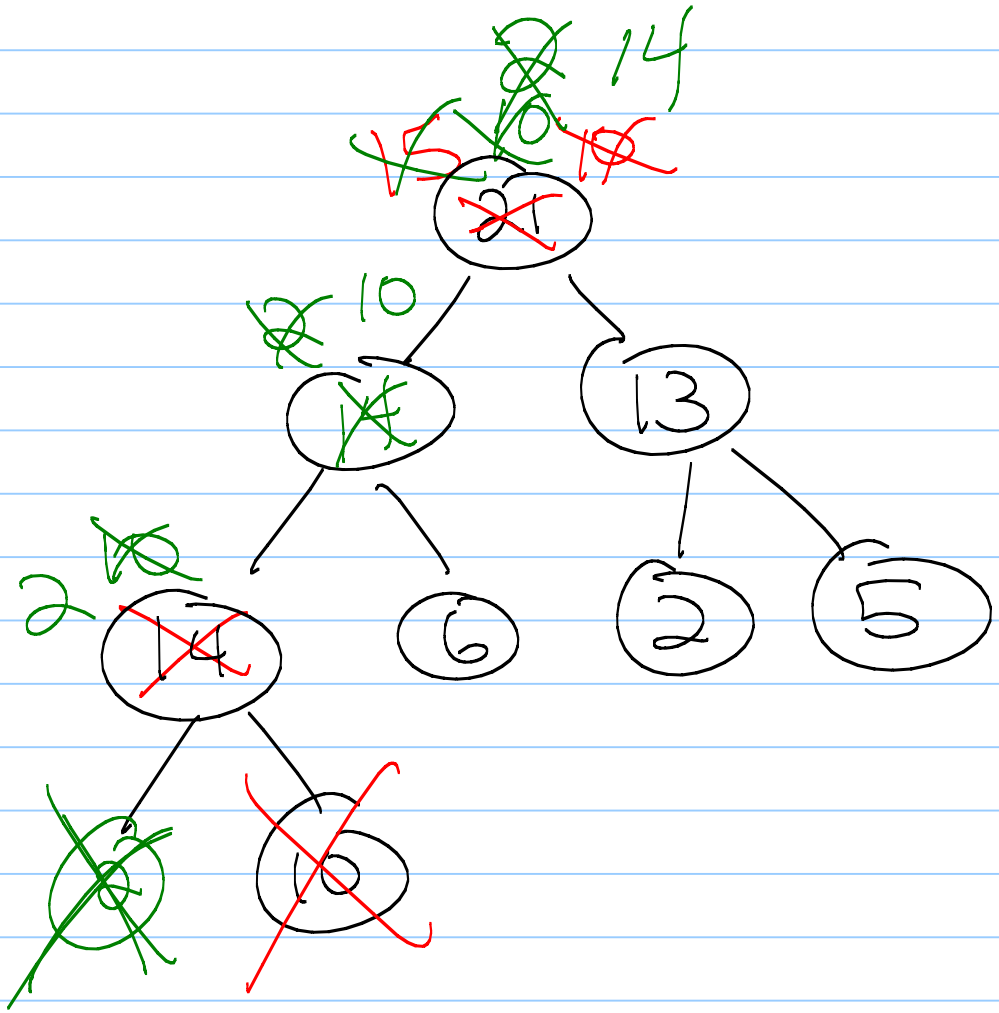
insert (52) ← problem

insert (7)



insert at bottom
while (new one ≥ parent)
swap new one ↓ parent
move up

Remove

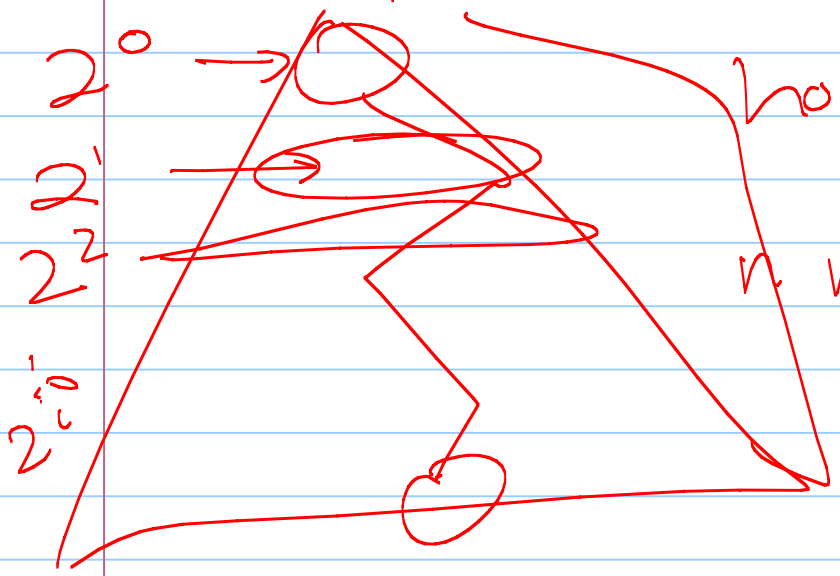


Running times

How many comparisons / swaps?

$O(\lg n)$

Each insert or removeMax travels + swaps along entire root to leaf path.



n nodes

How long are these?

$$\sum_{i=0}^{d} 2^i = n$$

$$2^0 + 2^1 + 2^2 + \cdots 2^d = n$$

$$= 2^{d+1} - 1$$

$$2^{d+1} - 1 = n$$

$$\log_2\left(2^{d+1}\right) = \log_2(n-1)$$

$$\log_2 2^{(d+1)} = (d+1)\log_2 2 = d+1$$

So: $d = \log_2(n-1) + 1$

$$d = O(\log_2 n)$$

Code for this class

- Array - Based. Why?

these are neatly complete trees, so saves space

To do : Code