CSCI 2100

Heap Recap
Binary Trees
Search Trees

Recap

- HW due today
- Next HW - posted after class, due in 1 week (pen + paper)
- Lab tomorrow

Last "time":

- Trees: binary tree (height + depth)

- Priority Queues: ADT supporting:
  - insert
  - getMax
  - removeMax

  - Heaps:

    binary tree implementation of PQ

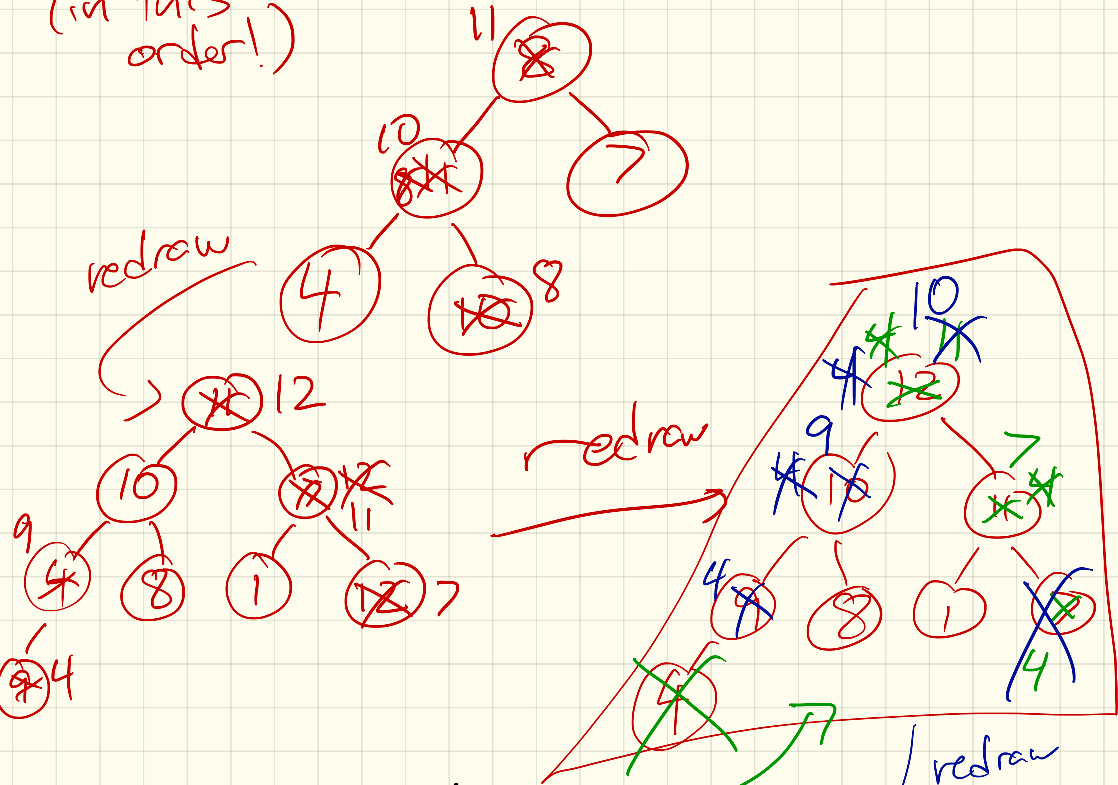
    value at node is $\geq$ children's value

    all operations are $O(\log n)$

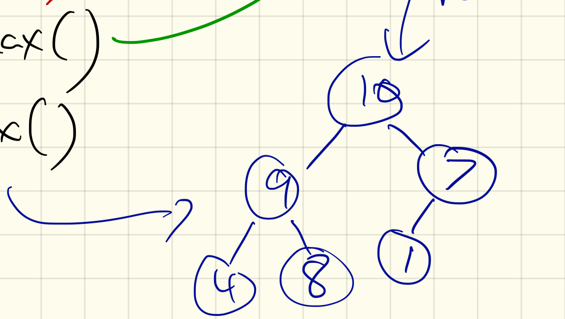
Heap example:

Insert: 8, 11, 7, 4, 10, 1, 12, 9

(in this order!)



Then [removeMax()
[removeMax()



Aside: Tree traversals

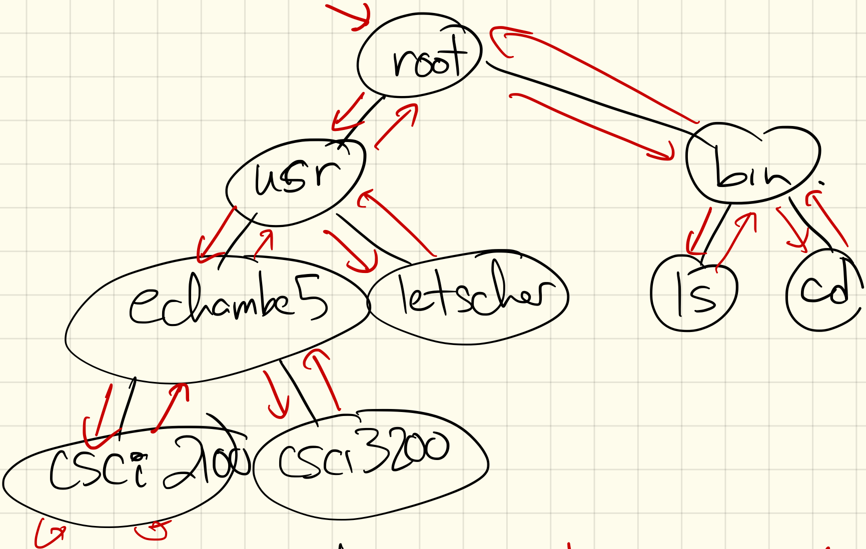
3 ways to traverse a tree:

Starting at root:

Preorder(v) =

```
    visit v
    Preorder(v → left)
    Preorder(v → right)
```

print

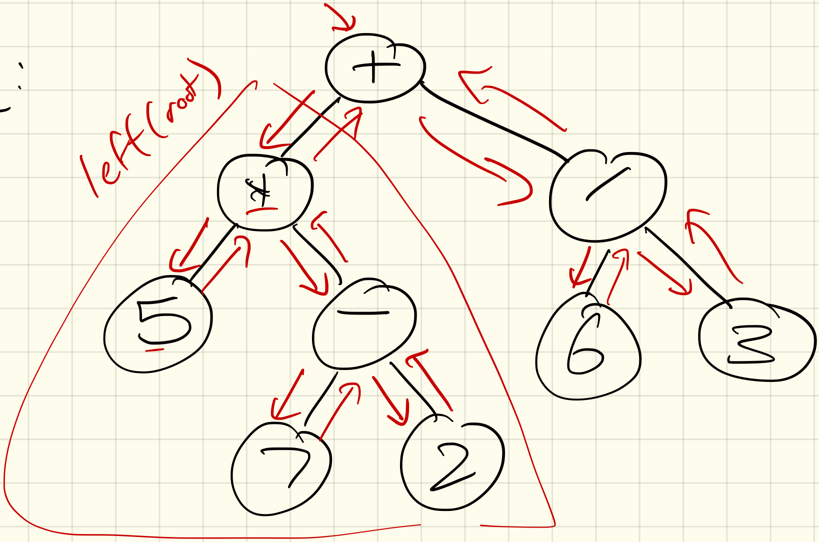


traversal order: root, usr, echambe5, csci2100, csci3200, letscher, bin, ls, cd,

(start at root)

```
Inorder(v) =
    Inorder(v → left)
    visit v
    Inorder(v → right)
```

Ex:



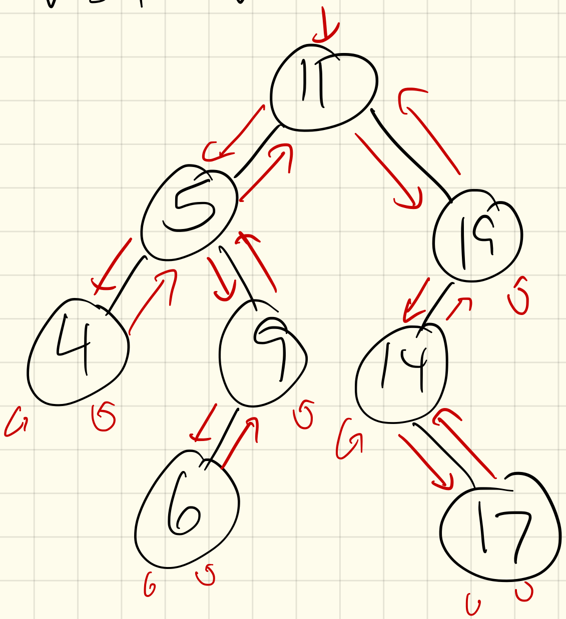
traversal: 5, *, 7, -, 2, +, 6, /, 3

↑

"middle"

```
Postorder(v) =
    post order (v -> left)
    post order ( v -> right)
    visit v
```

Ex:

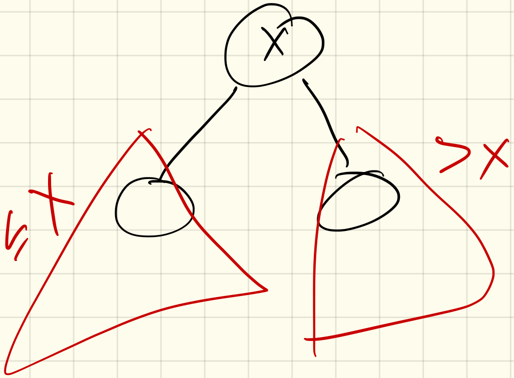


traversal : 4, 6, 9, 5, 17, 14, 19, 11

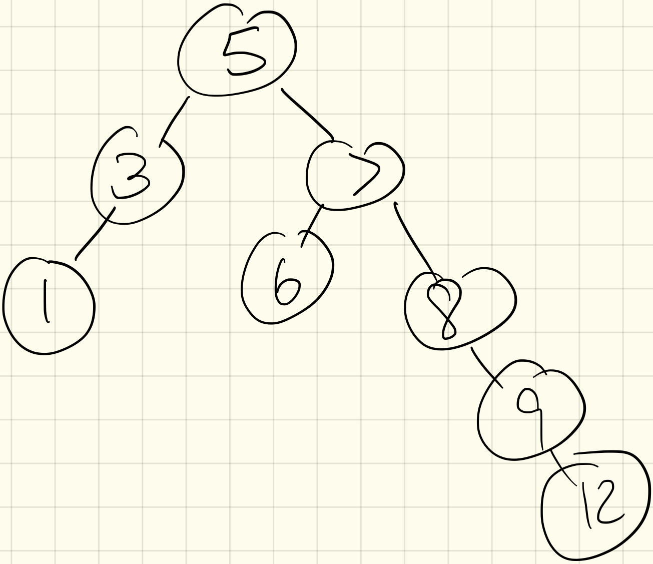
Next: Binary Search Trees

A binary tree where we maintain the following:

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



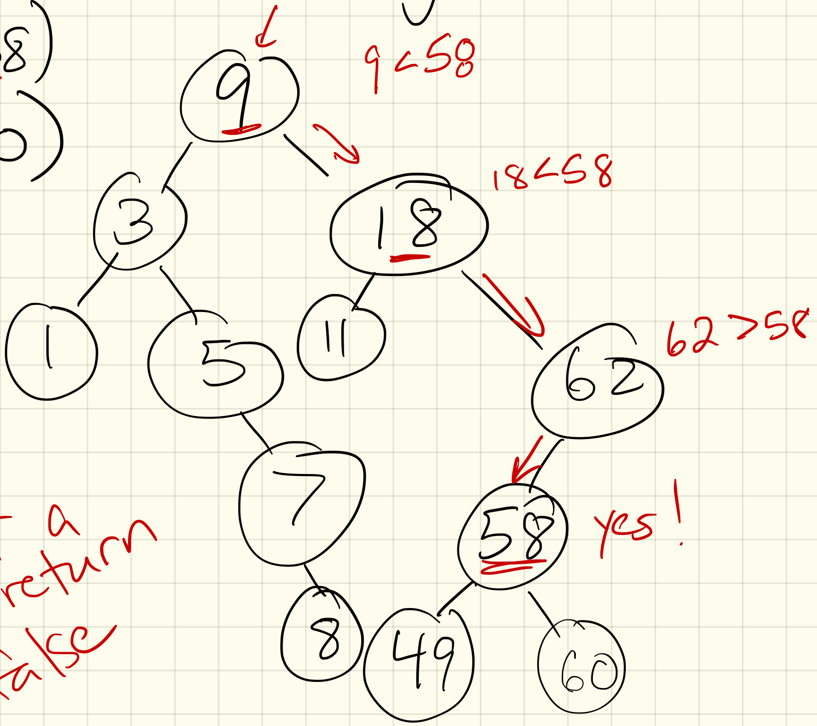
Ex:



Finding in a BST:

- Check if root = target value
  return true
- If root > target
  if left != NULL
  recurse on left child
  else return false
- If root < target
  recurse on right child

Ex:
find(58)
find(50)



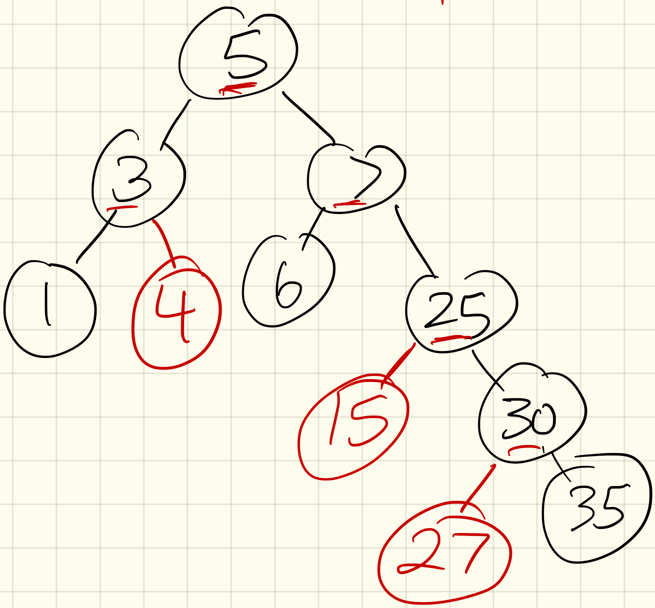
If hit a leaf return false

Inserting:

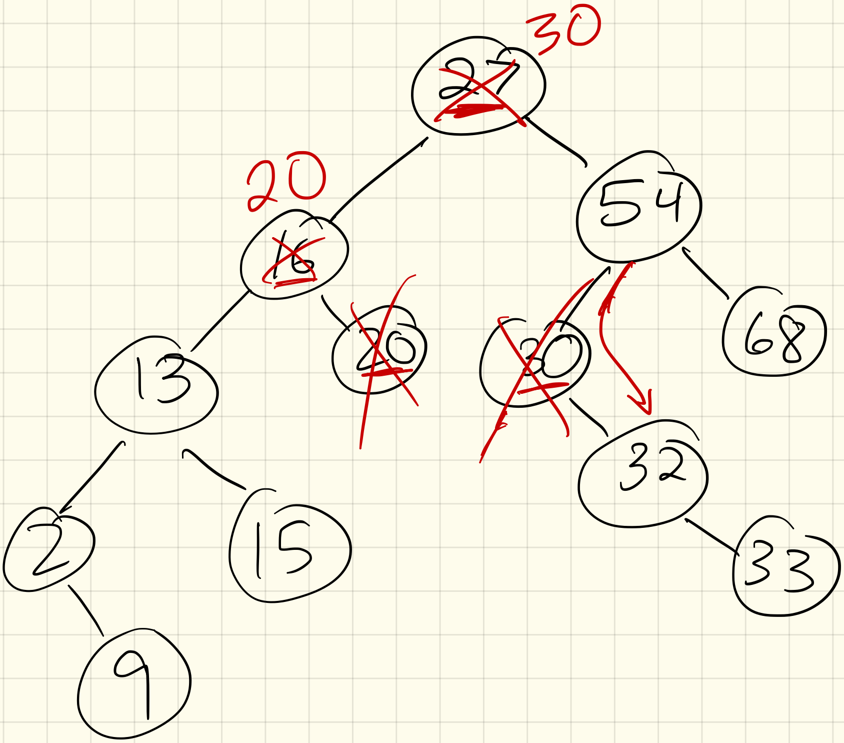
Given a BST, insert is done by finding the (unique) leaf location where the value fits.

Ex: insert (4)
insert (27)
insert (15)

find, but when hit a null, insert the new node



Delete: More Complex



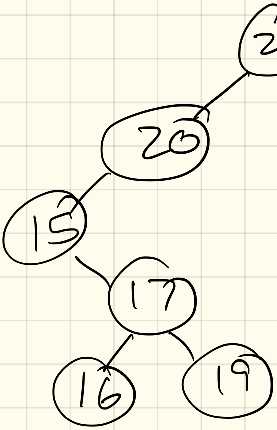
remove(16) : delete 20's node
copy 20 into 16's node

remove(27) Copy 30 up to root

Cases:

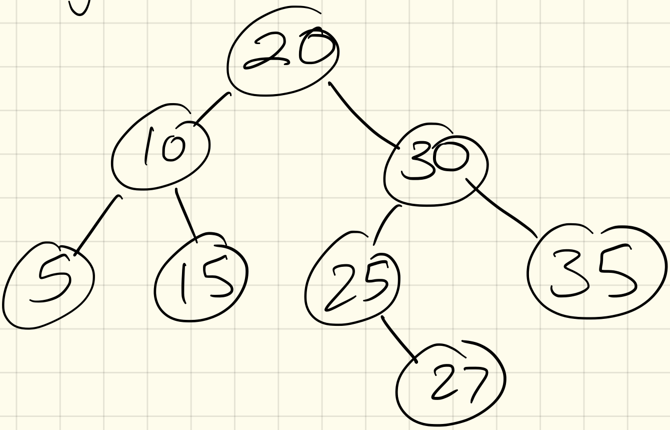
- Target could be a leaf

- Target could have only 1 child: remove(15)



Cases (cont):

- Target could have 2 children



remove(20):

```
Remove(x):
  Find (x)
  If x is a leaf
     delete x's node
  else if x has one child
       can remove x's node
       and "promote" it's
       only child to its spot

  Else:
     Find smallest value > x
        (Note: this is next value
         in an inorder traversal!)
     Copy that value to x's spot
       & remove that node,
         promoting it's right child
         in its spot (if it has one)
       Note: it will have no
             left child
```

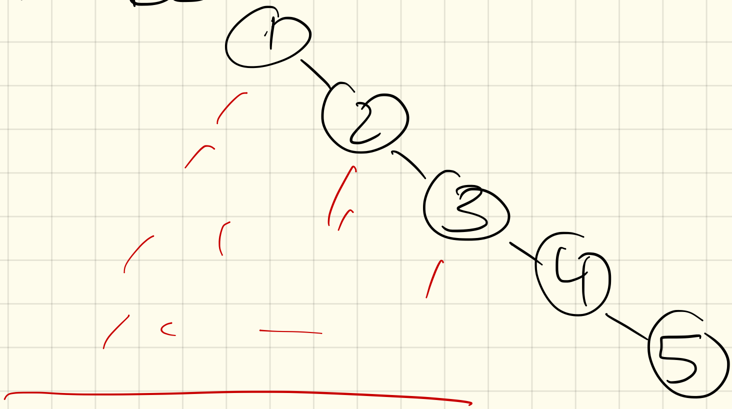
Code:

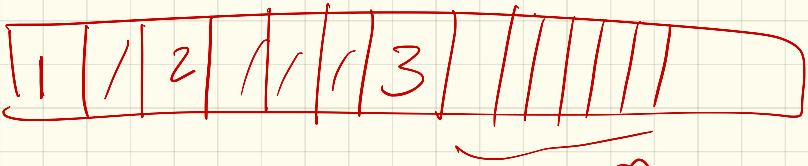
- Pointer based.

  Reason: we need to move around entire subtrees.

  Also, tree is not complete!

Valid BST:

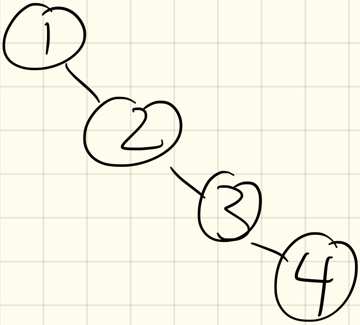




not space efficient

Note: BSTs are not unique!

Consider



Can you make another BST with these values?