Advanced Data Structures

Scapegoat Trees

Recap

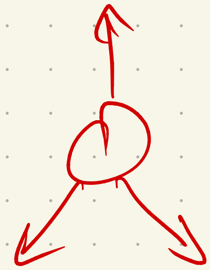
- HW1 up - due Feb. 14
  #3 is essay-type

- Sub on Friday & next Wednesday
  (No class Monday)

- Office hours:
  Monday 10-11am
  Wed 4-5pm
  or by appt - stop in or email

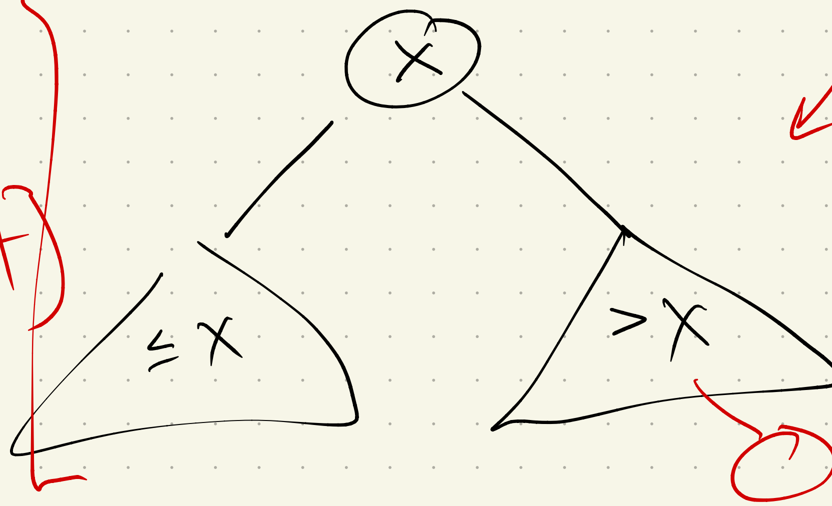
Binary Search Trees:

What is the "best" one?



Recap:

O(height)



Search:

start at root
if v == target
    return yes
else if ≤ target
    recurse left
else
    recurse right

Insert:

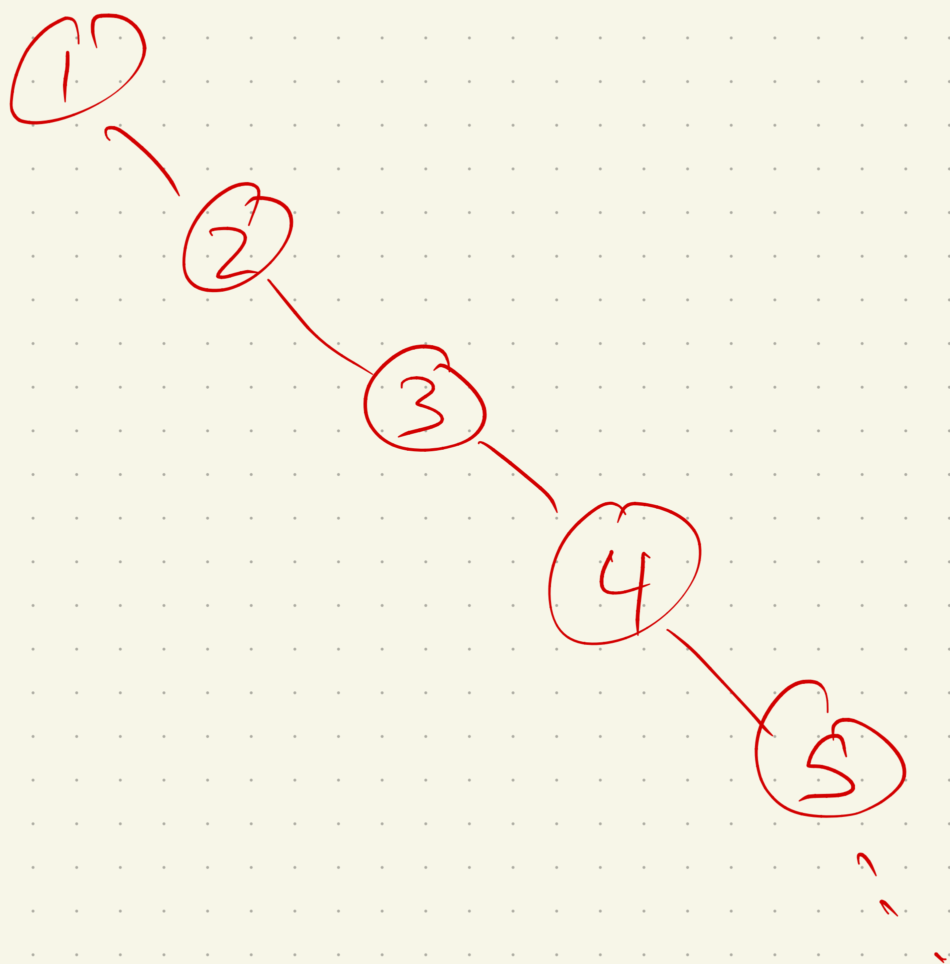
while (v has children)
    if x ≤ v
        go left
    else
        go right

Data Structures Class

- "Vanilla" BSTs (no rotations or balancing)

Runtime: $O(n)$

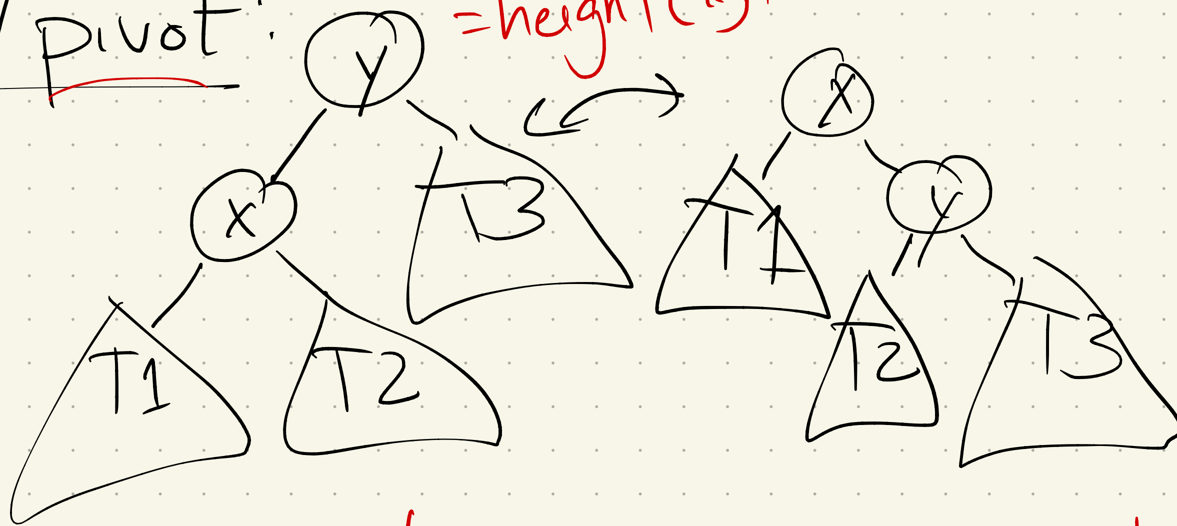
How can it get this bad?



BSTrees : balancing

Rotation / pivot :

$\text{height}(y) = \text{height}(x)+1$



unbalanced: left (or right) is too big

- AVL trees ← want $|h(\ell(v)) - h(r(v))| \leq 1$
- Red-Black trees

↳ $O(\log_2 n)$

~~Today~~ This week :

- Scapegoat Trees
- Splay Trees

Terminology I'll assume:

- search key
- node
- left/right child, parent
- internal/leaf node
- root
- ancestor/descendant
- preorder, inorder, postorder

Recap:

- Height(v): distance to furthest leaf in v's subtree
- Depth(v): distance from v to the root
- Size(v): # of nodes in v's subtree

Scapegoat Trees:

[Anderson '89, Galperin-Rivest '93]

Supports amortized $O(\log n)$.

Basic idea:

- Standard BST search
- Delete: mark "deleted" node.
  When tree is half dirty, rebuild into perfect tree.

Runtime:

Claim: rebuild a perfect tree in linear time

$\Rightarrow O(\log n)$ amortized time

And insert:

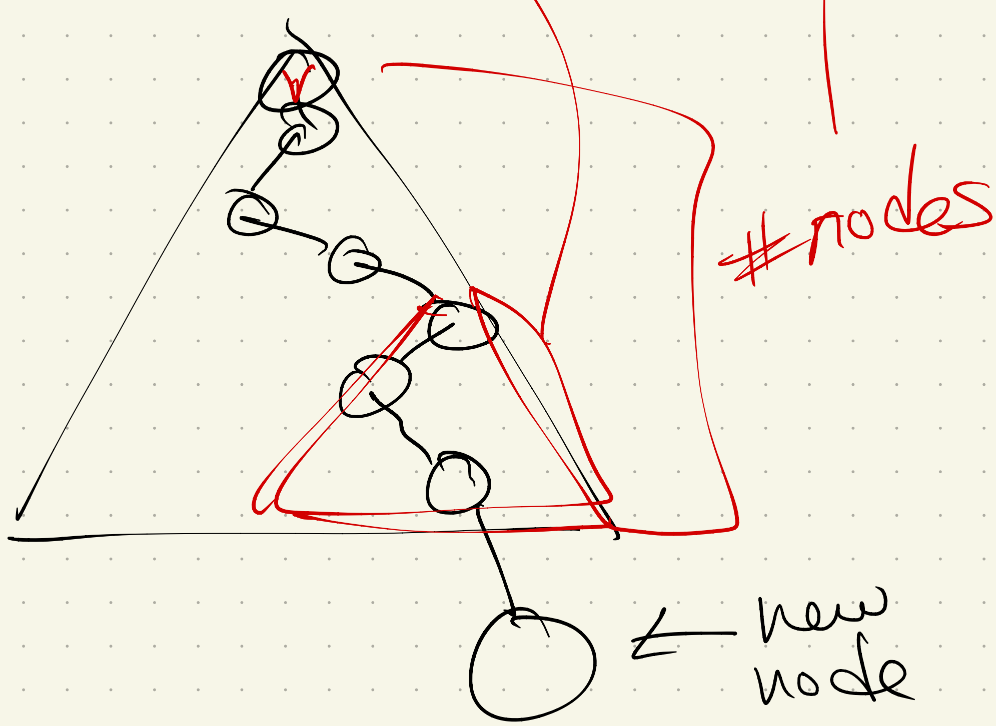
Standard insert

But: if imbalanced, rebuild a subtree containing new leaf

Dfn: Fix any $\alpha > 2$.

A node in *imbalanced* if $\text{height}(v) > \alpha \lg(\text{size}(v))$

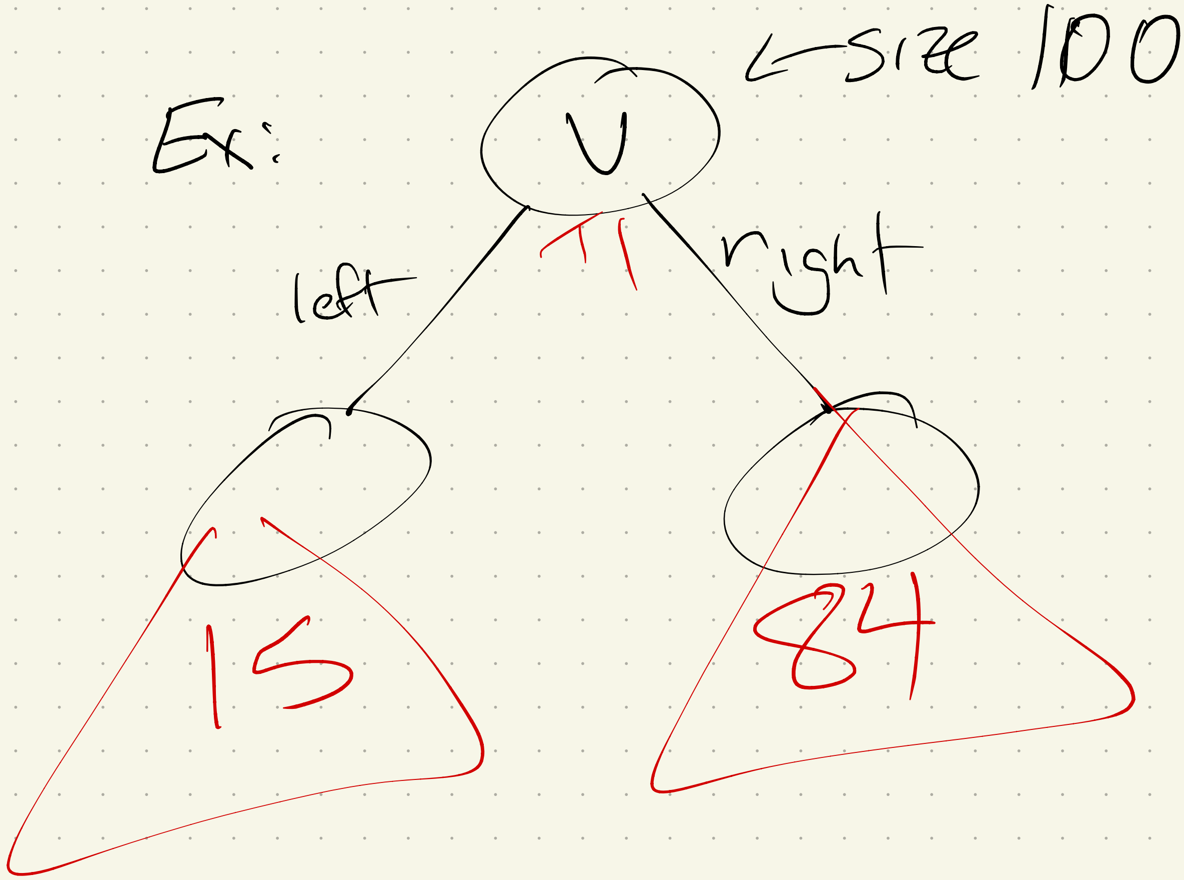
So here:



Let: $I(v) =$

$\max\{0, |size(left(v)) - size(right(v)) - 1\}$

Ex:



Lemma:

Just before rebuilding at v,

$$I(v) = \Omega(n)$$

$\text{size}(\ell(v)) - \text{size}(r(v)) \geq cn$

proof:

If imbalanced, $h(v) > \alpha(\lg \text{size}(v))$

(by dfn of imbalanced)

but left(v) + right(v) were not imbalanced:

$$h(\text{left}(v)) \leq \alpha(\lg(\text{size}(\text{left}(v))))$$

$$h(\text{right}(v)) \leq \alpha \lg(\text{size } r(v))$$

Wlog: Assume insert on left; so:

$$h(v) \overset{?}{=} h(\text{left}(v)) + 1$$

$$\leq \alpha \lg \text{size}(\ell(v)) + 1$$

Some intense math:

$$\alpha \lg(size(\ell(v))) + 1$$

$$> \alpha \lg(size(v))$$

raise both sides to power of 2

$$2 \cdot 2^{\alpha \lg(size(\ell(v)))} > 2^{\alpha \lg(size(v))}$$

rule: $2^{ab} = (2^a)^b = (2^b)^a$

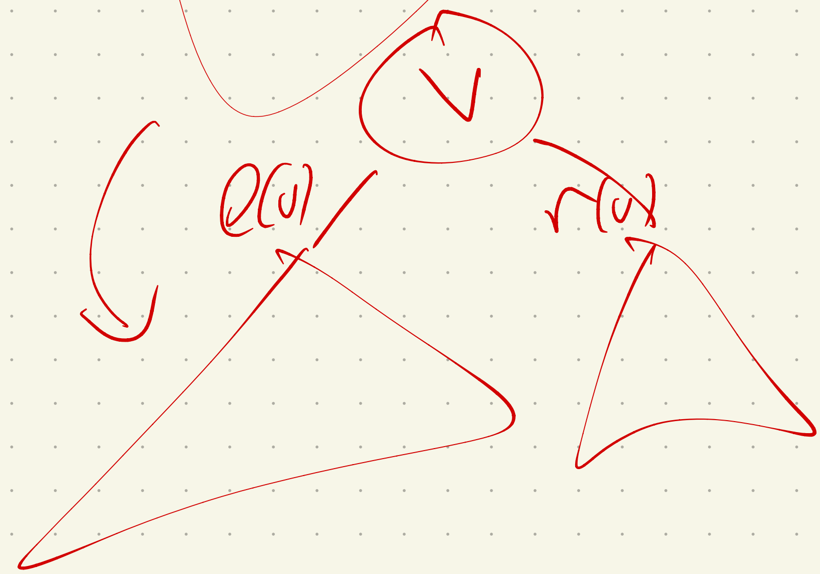
$$2 \cdot (size(\ell(v)))^\alpha > (size(v))^\alpha$$

take $\sqrt[\alpha]{}$:

$$\sqrt[\alpha]{2} \cdot size(\ell(v)) > size(v)$$

$$\Rightarrow size(\ell(v)) > \frac{size(v)}{2^{1/\alpha}}$$

$$\rightarrow \quad size(\ell(v)) > \frac{size(v)}{2^{1/\alpha}}$$



$$size(r(v)) = size(v) - size(\ell(v)) + 1$$

$$< \underbrace{\left(1 - \frac{1}{2^{1/\alpha}}\right) size(v)} + 1$$

so

$$I(v) \geq \underbrace{\left(\frac{2}{2^{1/\alpha}} - 1\right)}_{\text{constant}} size(v) \qquad (\text{goal: } \Omega(n))$$

$$= \Omega(n)$$

So: takeaway

$$I(v) = \Omega(\text{size}(v))$$

This means $\sim$size(v) insertions since the last rebuilding.

So: rebuild! How?

Several ways to do this in $O(\text{size}(v))$ time.

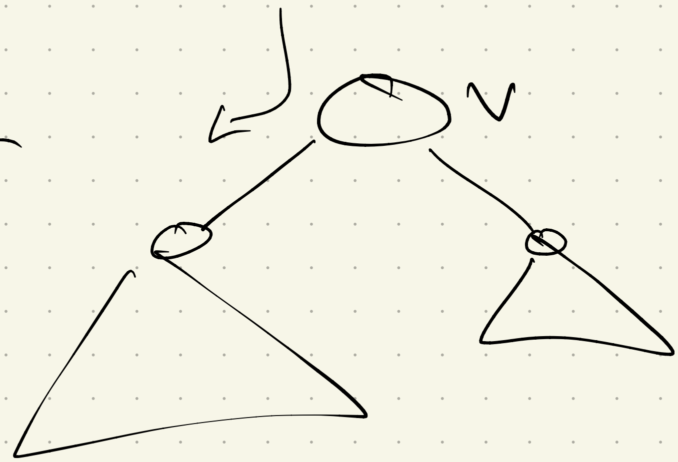
(HW question!)

k inserts to trigger $O(k)$ rebuild

$\Rightarrow$ amortizes to $O(1)$

Claim: ≤ 1 tree rebuild for each insertion

pf



when rebuild to "perfect" tree in $\ell(v)$, height goes down

Final runtime:

Find: no worse than $h = \alpha \log n$ } $O(\log n)$

Delete: $O(\log n)$ to find & mark

rebuild when $\geq \frac{1}{2}$ dirty

$\Rightarrow$ amortized $O(\log n)$

Insert: find, so $O(\log n)$ amortized

Next Topics

- Fractional Cascading
- Splay Trees
- (a,b)-Trees