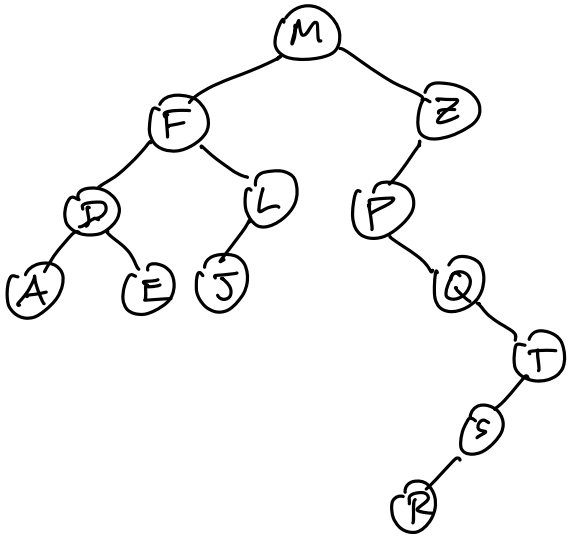
Binary Search Trees



Balanced: $O(\log n)$ search time

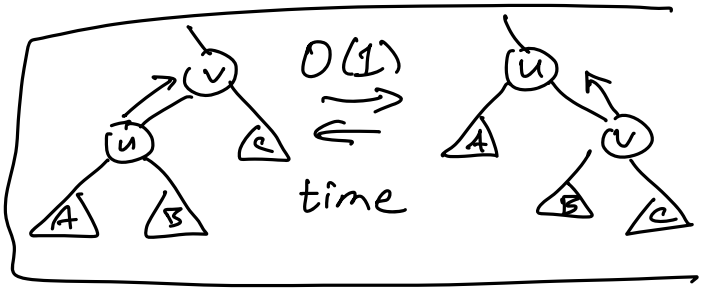
Worst case: $O(n)$

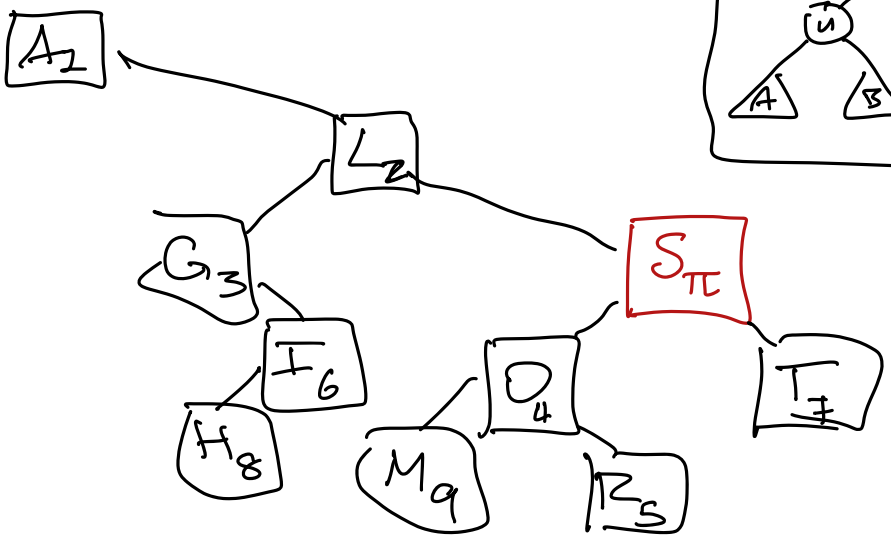
- Find(x) ← Pred(x), Succ(x)
- Insert(x)
- Delete(x)
- $T_<, T_> \leftarrow$ Split(x)
- Join($T_<, T_>$)

ideally: $O(\log n)$ time

Treap — Aragon + Seidel '90 — $O(\log n)$ exp. time

Every node has a search key ← Binary search tree
and a priority ← min-heap





Insert(k, p)

New node (k, p)

Insert looking only at keys

Bubble up new node looking only at priorities

All operations:

time = O(depth of some node)

Delete:

Set priority ← ∞

Rotate down (promoting child with smaller priority)

Chop off leaf

Random priorities ⇒

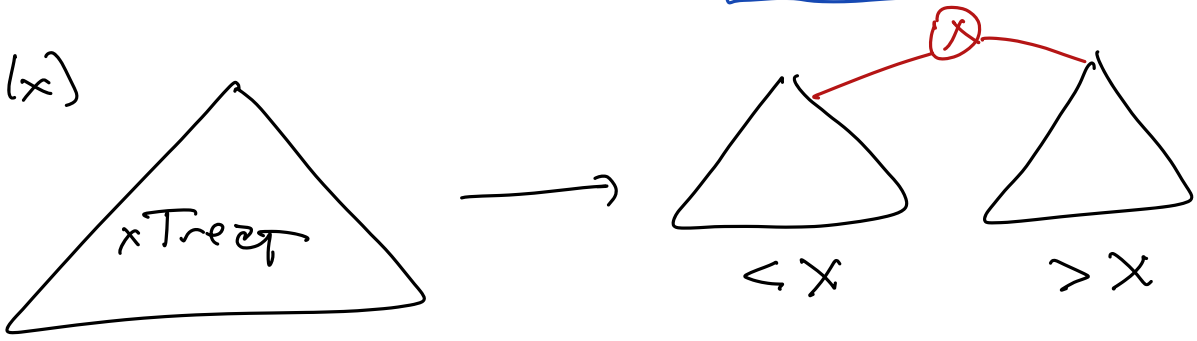
For any node v

$E[\text{depth}(v)] = O(\log n)$

Stronger:

$E[\max_v \text{depth}(v)] = O(\log n)$

Split(x)



set priority(x) ← -∞
bubble up to root

WLOG: keys are integers 1...n

$$E[\text{depth}(k)] = E[\#\text{proper ancestors of } k]$$

$$= \sum_{i=1}^{n} \Pr[i \Uparrow k]$$

$i \Uparrow k$ = i is proper ancestor of k

Lemma: For all $i < k$,

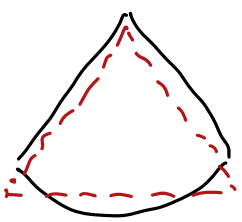
$$i \Uparrow k \iff \text{priority}(i) = \min\{\text{priority}(j) \mid i \le j \le k\}$$

Lemma ⇒ $\Pr[1 \Uparrow n] = 1/n$

$$\Pr[i \Uparrow k] = \frac{1}{k-i+1}$$

$$E[\text{depth}(k)] = \sum_{i=1}^{n} \Pr[i \Uparrow k]$$

$$= \sum_{i=1}^{k-1} \Pr[i \Uparrow k] + \sum_{i=k+1}^{n} \Pr[i \Uparrow k]$$



$$= \underbrace{\sum_{i=1}^{k-1} \frac{1}{k-i+1}}_{j = k-i+1} + \underbrace{\sum_{i=k+1}^{n} \frac{1}{i-k+1}}_{j = i-k+1}$$

$$= \sum_{j=2}^{k} \frac{1}{j} + \sum_{j=2}^{n-k+1} \frac{1}{j}$$

$$= H_k - 1 + H_{n-k+1} - 1$$

$$< 2 \ln n - 2$$

Lemma: For all $i<k$,

$$i \Uparrow k \iff \text{priority}(i) = \min\{\text{priority}(j) \mid i \le j \le k\}$$

Proof:

Five cases:

- root $< i$ → by IH
- root $= i$ → $i \Uparrow k$ priority(i) is min
- $i <$ root $< k$ → $i \not\Uparrow k$ priority(i) is not min
- root $= k$
- root $> k$ → by IH □

Treap = BST defined by random priorities

Treap = BST obtained by inserting keys in random order

Treap = recursion tree for randomized quicksort

RANDOMQUICKSORT(A):

randomly permute A

for each x in A
  Insert x into BST

inorder(BST)