6.006
Spring 2008
L3

Runway reservation system
- Definition
- How to solve with lists

Binary Search Trees.
- Operations

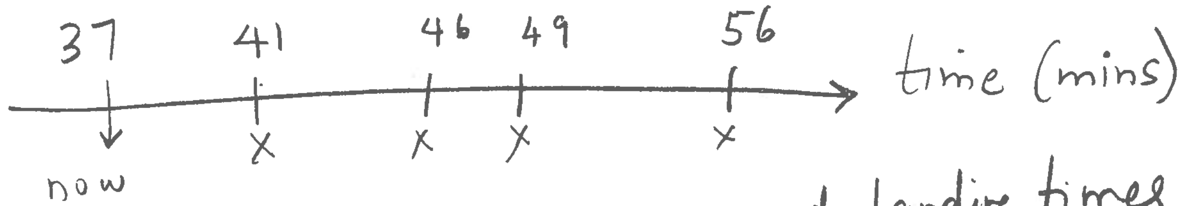
Readings: CLRS
Chapter 10, 12.1-3

Runway Reservation System

- Airport with single (very busy) runway (Boston $6 \to 1$)
- "Reservations" for future landings
- When plane lands, it is removed from set of pending events

Reserv req specify "requested landing time" t
Add t to the set if no other landings are scheduled within < 3 minutes either way.
- else error, don't schedule

Example



$R = \{41, 46, 49, 56\}$ reserved landing times

Request for time : 44 not allowed ($46 \in R$)

53 OK

20 not allowed (already past)

$|R| = n$ Goal: run this system efficiently $O(\lg n)$ time

Keep R as <u>sorted</u> list.

Init : R = [ ]

req(t) : if t < now: return "error"

$\theta(n)$ $\left\{ \begin{array}{l} \text{for i in range(len(R)):} \\ \quad \text{if abs (t - R[i]) < 3: return error} \end{array} \right.$

$\left. \begin{array}{l} \text{R.append(t)} \\ \text{R = sorted (R)} \end{array} \right\}$ $\theta(n \lg n)$

land : t = R[0]

if (t $\neq$ now) return error

R = R[1:]. (drop R[0] from R)

Can we do better?

Sorted list: 3 min check can be done in $O(1)$
Can insert new time/plane rather than append & sort, but insertion takes $O(n)$ time.

Sorted array? Can do binary search to find place to insert in $O(\log n)$ time
Actual Insertion requires shifting elements $O(n)$ time!

Unsorted list/array: Search takes $O(n)$ time

Dictionary or Python Set: Insertion is $O(1)$ time
3-min check takes $\Omega(n)$ time

What if times are in <u>whole</u> minutes?
Large array indexed by time does the trick.
Doesn't work for arbitrary precision time or varying width slots for landing.

$\boxed{\text{Need fast insertion into sorted list!}}$

<u>New requirement:</u> Rank(t): how many planes scheduled to land at times $\leq t$?

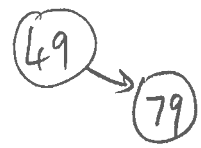
design amendment?

BST

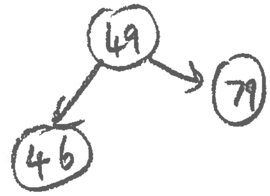
BST = nil

Insert(49) BST → 

"root"

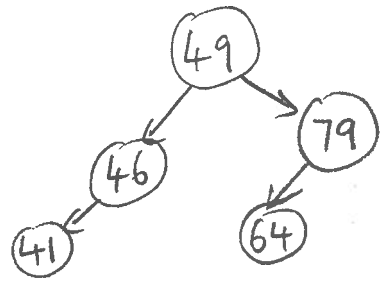
" 79 " 

all elements > 49 off to the right, in right subtree

46 " 

all elements < 49, go into left subtree

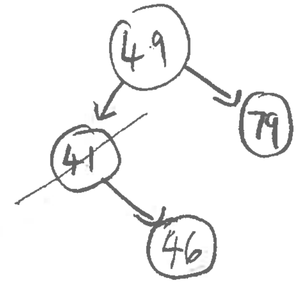
41,
64,

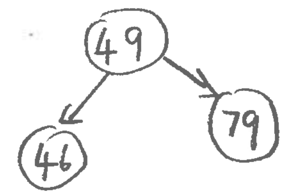


Finding the min element in a BST.

Just go left! (till you can't anymore)

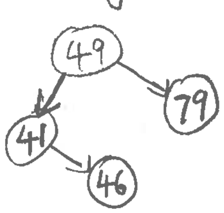
Delete-min:
find min & eliminate





$\boxed{\text{All are } O(h) \text{ where } h \text{ is height of BST}}$

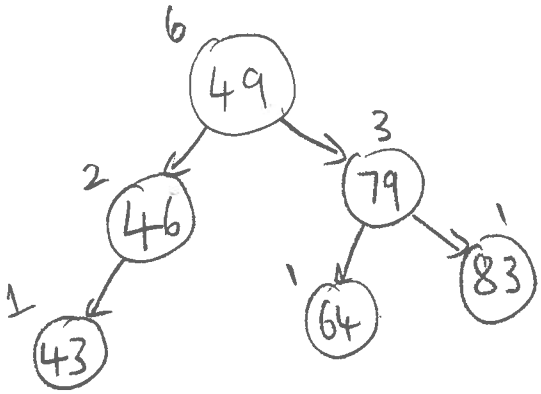
next-larger(x)



if right child ≠ NIL, return minimum(right)
else y = parent(x)
while y ≠ NIL and x = right(y)
x = y ; y = parent(y)
return y;

What about rank(t)?

Can't solve it efficiently with what we have, but can augment the BST structure.



what lands before 79?

Keep track of size of subtrees, during insert & delete.

Walk down tree to find desired time
Add in nodes that are smaller, and add in subtree sizes to the left.

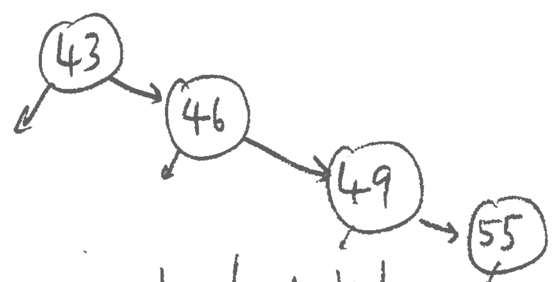
O(h) time

49 + 46 subtree + 79 + 64 subtree
1 + 2 + 1 + 1 = 5

(Have we accomplished anything?)

Height h of the tree should be O(log n).

Insert into BST in sorted order.



looks like a linked list.

O(n) not O(log n)

Balanced BSTs to the rescue!