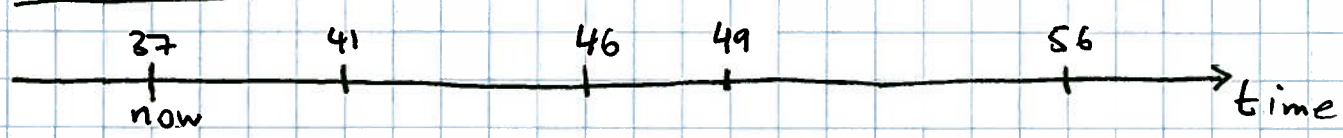
6.006 Lecture 3: Binary Search Trees

- Runway reservation problem
- Binary Search Trees (BSTs), operations
- CLRS Ch 10 + 12.1-3

Runway Reservation System

- Maintains a set R of future landing times
- When a plane lands, remove from the set
- For a request to land at time t:
  - reject request if there are reservations within <3 minutes of t,
  - otherwise add t to set

Example



$R = \{41, 46, 49, 56\}$, $n = |R| = 4$

request for time:

44 reject (too close to $46 \in R$)

53 add reservation (just right)

20 already past, not allowed.

Goal: handle requests & landings in $O(\lg n)$ time.

- Keeping R as an (arbitrarily orded) list
  - Checking a request takes $O(n)$ time
  - Deleting upon landing takes $O(n)$ time

- Keeping R as a <u>sorted</u> list

```
init : R = []
req(t): if t < now return 'error'
        for i in range(len(R)):
            if abs(t-R[i]) < 3 : return 'error'   } O(n
        R.append(t)
        R.sort()                                  ← O(n lg n)
land : t = R[0]
        if (t ≠ now) return 'error'
        R = R[1:]                                 ← O(n)
```

- Can we do better with a sorted list?

37	41	46	49	56			

can use binary search to determine conflicts in $O(\lg n)$ but inserting t into R and deleting R[0] still takes $O(n)$

→ better (no sorting) but not enough

- <u>Indicator representations</u>
  - Represent a set R such that

    A[t] == True iff $t \in R$

    - Python dictionaries
    - Python sets
    - A Python list of all possible landing times (all possible elements of R)

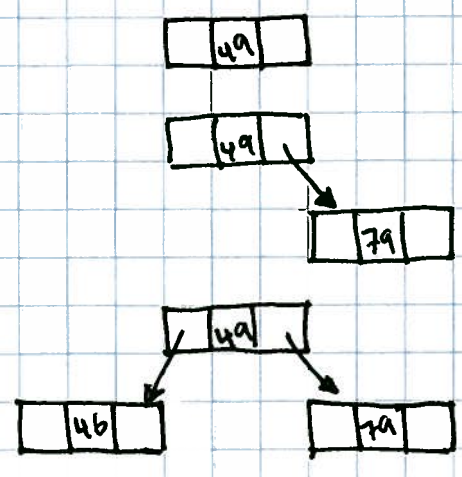
  - Fast insertion, deletion
  - Checking a request is fast if landing times are whole minutes, but expensive for hi-res
  - Not a flexible data structure:

    How many planes land on or before t?

  - [illegible]

- Binary Search Trees (BSTs)

1. insert(49)
2. insert(79)
~~3. insert~~
3. insert(46)

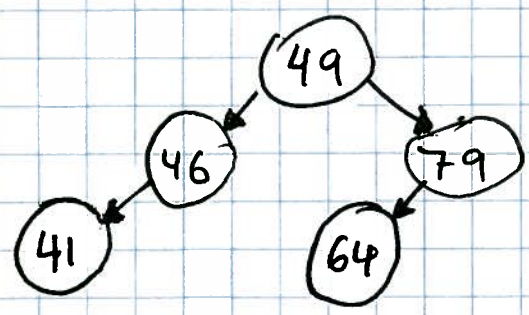


this is the root

to find elements > 49 go right

elements < 49 are in the left subtree

A more compact graphical representation:



Same representation in the program: every node has a value and left/right pointers

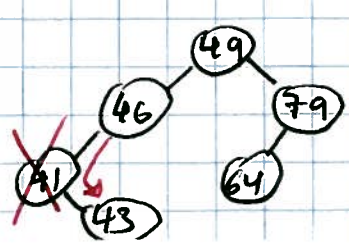
find(64): follow left/right pointers until done

find(48): how do you tell that 48 is not in the set?

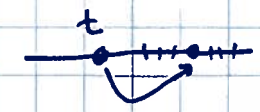
findmax(): just go right until you can't go right

findmin(): similar, go left

deletemin(): min & max nodes will have zero or one child ⇒ find, eliminate, & patch



next-larger(t): v = find (t)



if right(v) ≠ None : return findmin (right(v))

else: y = parent (v)

while $y \neq$ None and v = right(y):

$v = y$

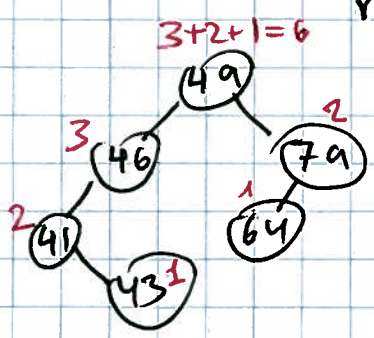
y = parent (v)

return y

or

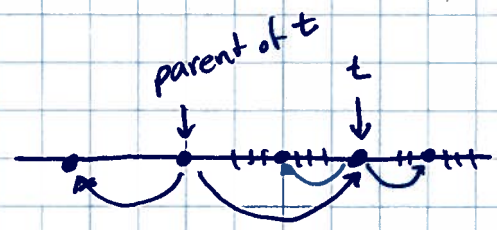
- All of these operations on the BST take $O(h)$ time, where h is the height of the tree

- Other operations that we can do in $O(h)$ time:



rank(t) how many planes land ≤ t?
must augment each node with size of subtree, update during insertions & deletions.

delet(t) more tricky if t has 2 children, but still $O(h)$.



(delete next-larger(t) than put it in the node that stored t; see textbook).