L04.01
Binary Search Trees (BST)

50.004 Introduction to Algorithm

Gemma Roig
(slides adapted from Dr. Simon LUI)
ISTD, SUTD

L04.01
Binary Search Trees (BST)

Chapter 12 and 14 of CLRS book

Overview

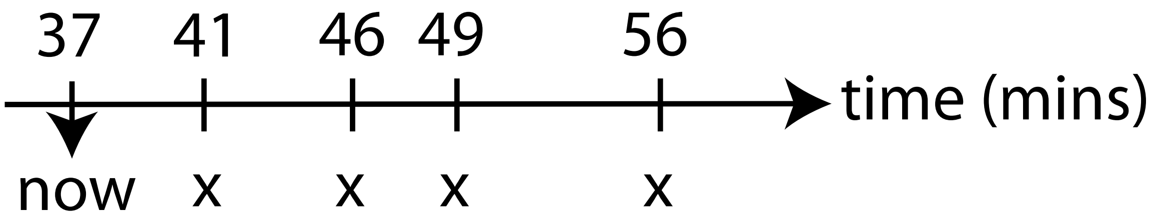
- We will talk about a runway reservation system
- Use an array/list to solve it
  - In O(n) time
- Use Binary Search Tree (BST) to solve it
  - In O(height) time
    - usually height = logn,
    - but in the worst case height = n-1

Runway reservation system

- Problem definition:
  - Single (busy) runway
  - Reservations for landings
    - maintain a set of future landing times (R)
    - a new request to land at time t
    - add t to R if no other landings are scheduled within < 3 minutes from t
    - when a plane lands, remove its reservation from the set

Runway reservation system

- Example



  - R = (41, 46, 49, 56) - reserved landing times
  - requests for time:
    - 40 => reject (crash with 41)
    - 42 => reject (crash with 41)
    - 53 => ok
    - 20 => not allowed (already past)

Proposed algorithm

```
init: R = [ ]
req(t): if t < now: return "error"
for i in range (len(R)):
     if abs(t-R[i]) < 3: return "error"
R.append(t)
R = sorted(R)
land: t = R[0]
if (t != now) return;
R = R[1: ] (drop R[0] from R)
```

- Complexity? O(n) + sorting complexity
- Can we do better?

Some options:

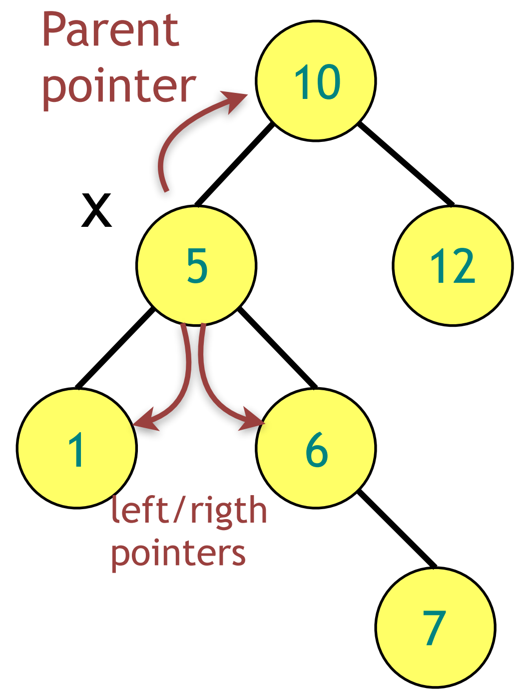
- Keep R as a sorted list:
  - insert new element in proper place: O(n)
  - <3 minute check: O(1)
- Keep R as a sorted array:
  - Find the place to insert new element: O(logn)
  - <3 minute check: O(1)
  - Insert (shifting of elements): O(n)
- Keep R in unsorted order (min-heap):
  - search for collisions: O(n)
  - insert new element: O(logn)

To improve it, we need fast insertion

Binary Search Trees (BSTs)

Binary ~~Search~~ tree:

- Each node x has:
  - key[x]
  - Pointers:
    - left[x]
    - right[x]
    - parent[x]

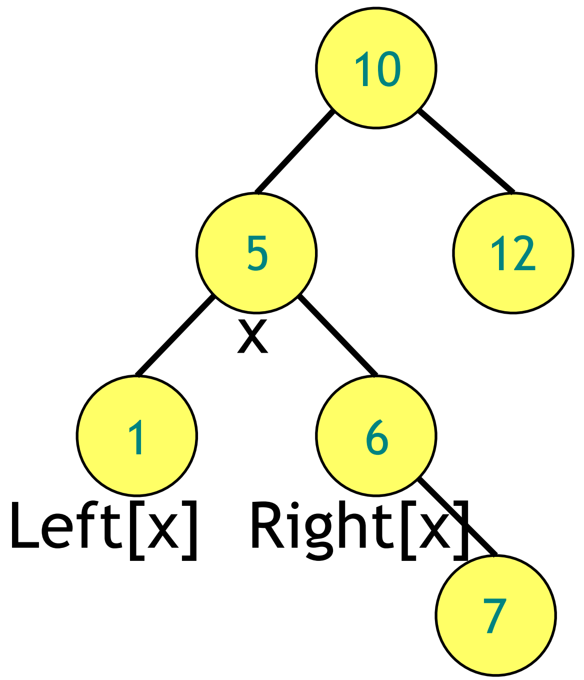


Binary Search Trees (BSTs)

Binary Search tree:

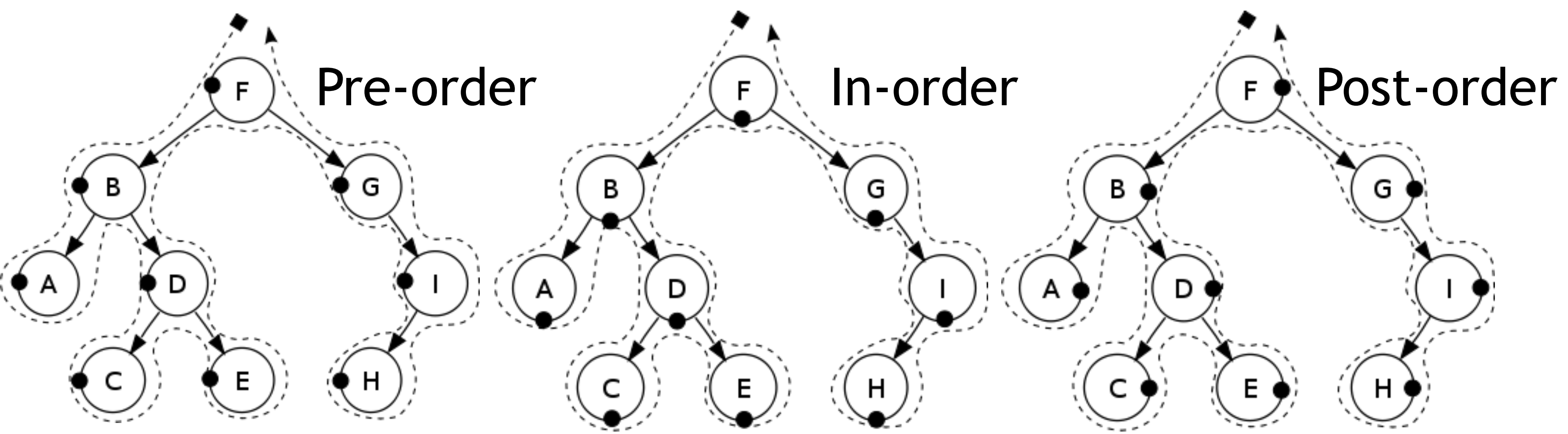
Property:

- Key[left[x]] < key[x]
- Key[x] < key[right[x]]



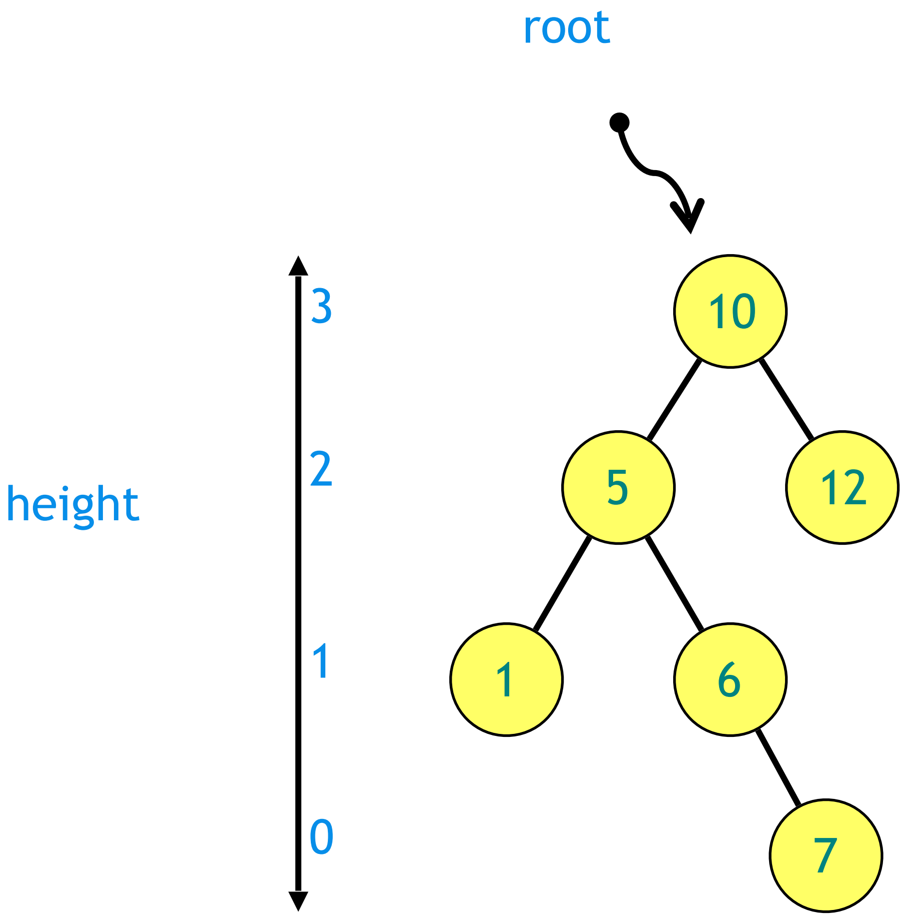
Tree traversals

- Algorithms to output the tree nodes
- Pre-order: FBADCEGIH
- In-order: ABCDEFGHI(In-order = sorted list of BST)
- Post-order: ACEDBHIGF

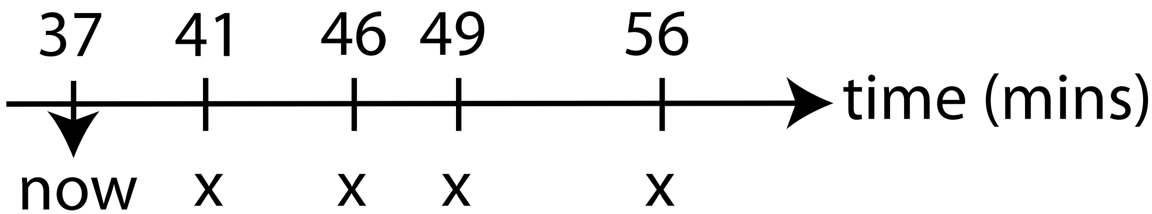


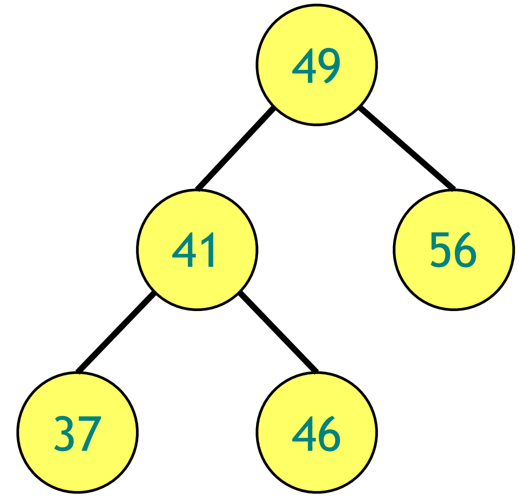
Growing BSTs

- Insert 10
- Insert 12
- Insert 5
- Insert 1
- Insert 6
- Insert 7



Use BST for the runway problem





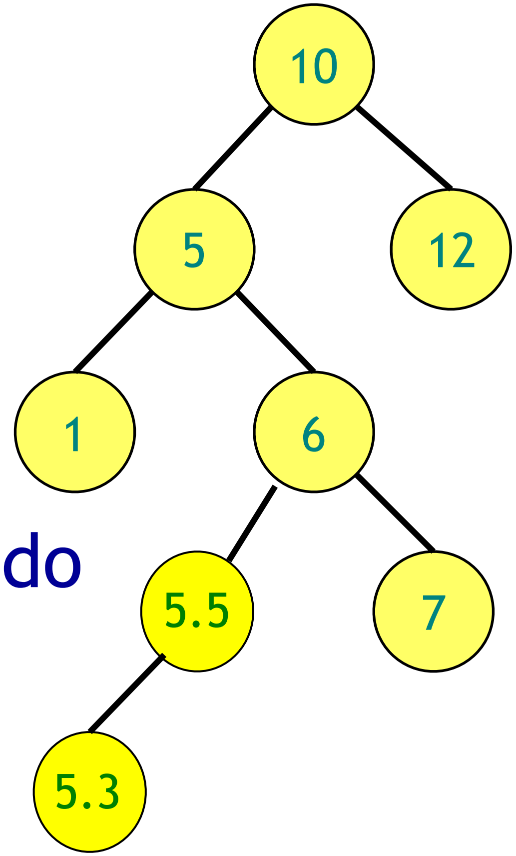
Operations:

- insert(k) (with "< 3 minute check")
- find(k): finds the node containing key k (if it exists)
- findmin(x): finds the minimum of the tree rooted at x
- deletemin(): finds the minimum of the tree and delete it
- next-larger(x): finds the next element after element x

Next-larger

3 cases to consider!

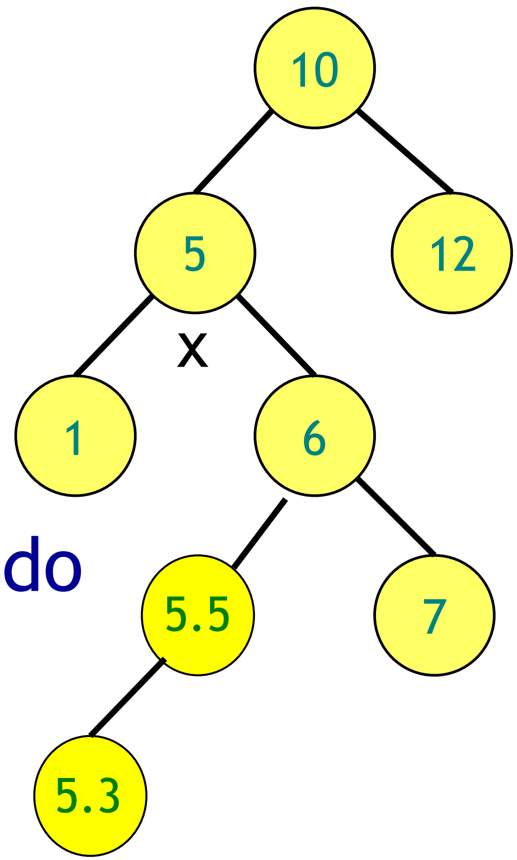
```
next-larger(x):
if right[x] ≠ NIL then
   return findmin(right[x])
else
    y = parent[x]
    while y≠NIL and x==right[y] do
        x = y
        y = parent[y]
    return y
```



Next-larger

3 cases to consider!

```
next-larger(x):
if right[x] ≠ NIL then
   return findmin(right[x])
else
     y = parent[x]
     while y≠NIL and x==right[y] do
          x = y
          y = parent[y]
     return y
```

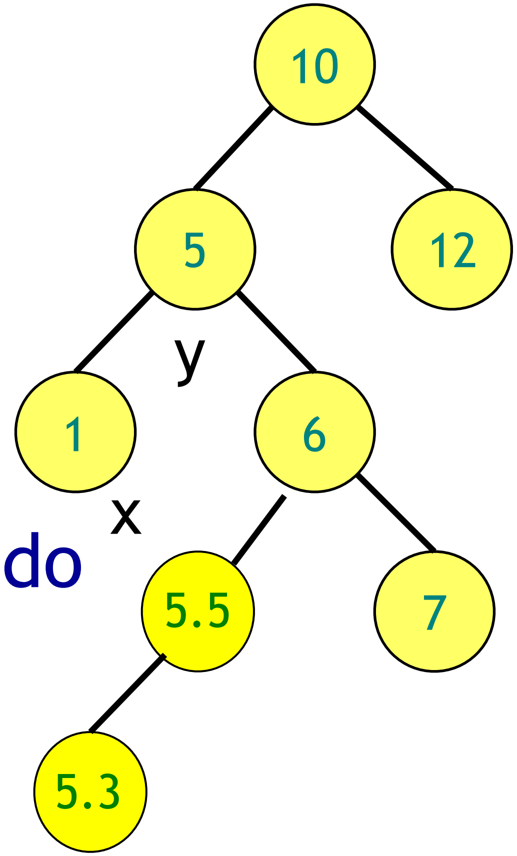


Example 1:
next-larger(5) = 5.3

Next-larger

3 cases to consider!

```
next-larger(x):
if right[x] ≠ NIL then
   return findmin(right[x])
else
    y = parent[x]
    while y≠NIL and x==right[y] do
        x = y
        y = parent[y]
    return y
```

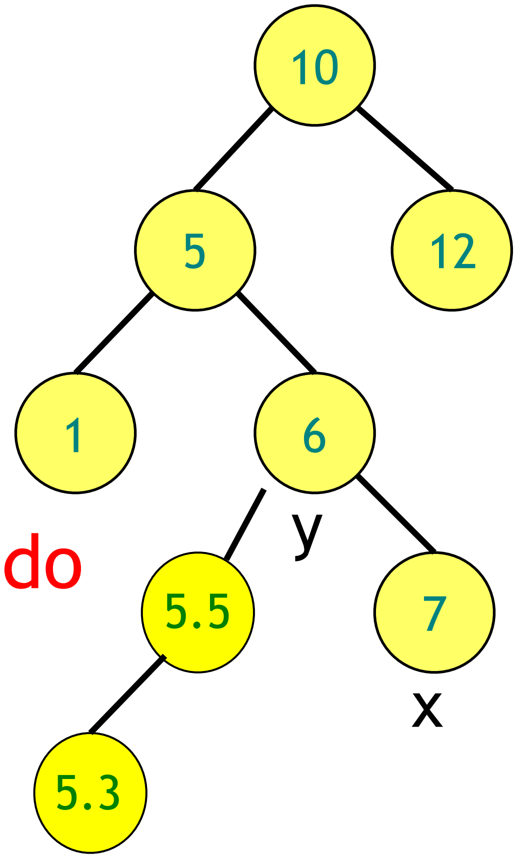


Example 2:
next-larger(1) = 5

Next-larger

3 cases to consider!

```
next-larger(x):
if right[x] ≠ NIL then
   return findmin(right[x])
else
     y = parent[x]
     while y≠NIL and x==right[y] do
          x = y
          y = parent[y]
     return y
```

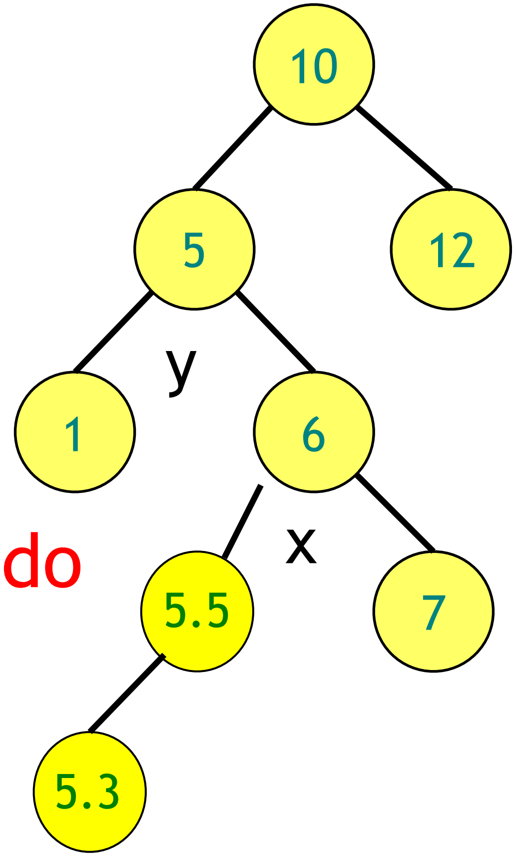


Example 3:
next-larger(7) =

Next-larger

3 cases to consider!

```
next-larger(x):
if right[x] ≠ NIL then
   return findmin(right[x])
else
     y = parent[x]
     while y≠NIL and x==right[y] do
          x = y
          y = parent[y]
     return y
```

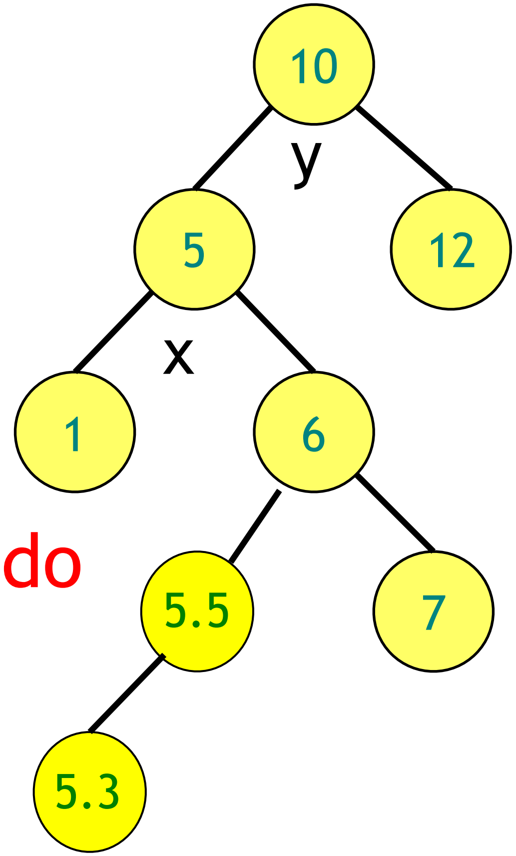


Example 3:
next-larger(7) =

Next-larger

3 cases to consider!

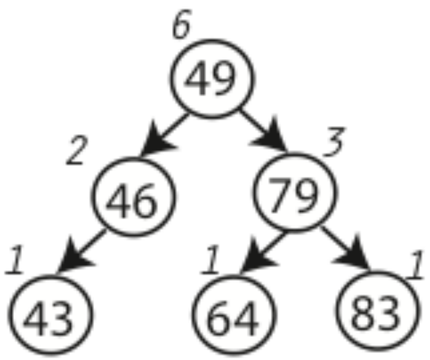
```
next-larger(x):
if right[x] ≠ NIL then
   return findmin(right[x])
else
     y = parent[x]
     while y≠NIL and x==right[y] do
          x = y
          y = parent[y]
     return y
```



Example 3:
next-larger(7) = 10

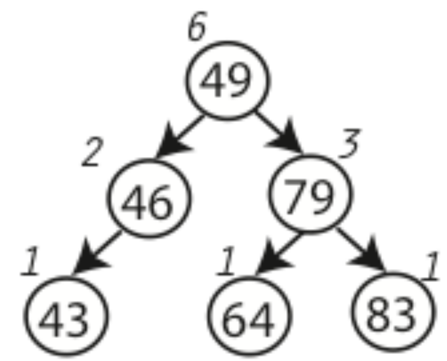
Augmenting a BST

1. Keep the total size of sub-trees at each node



Back to runway reservation system

- This can help us to answer “How many planes will land before t=79?”



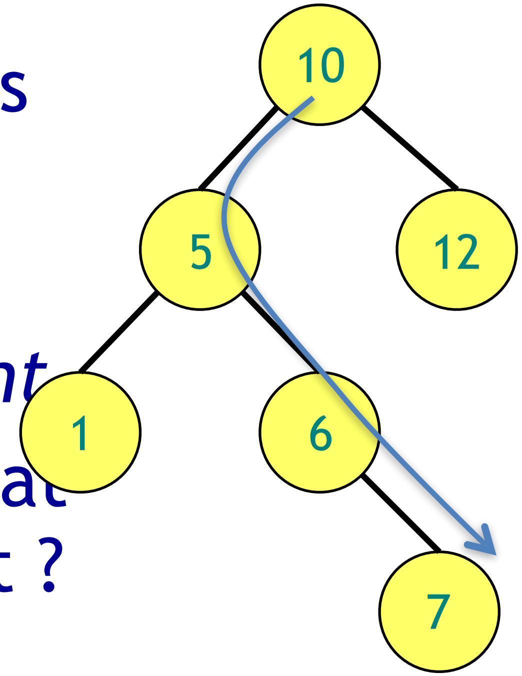
Answer:
leftsubtree(79) + parentnode(79) + leftsubtree(parent of 79)

= (64) + (49) + left(49)
= 1 + 1 + 2
= 4

Analysis

- We have seen insertion, search, findmin…
- How much time does any of this take ?
- Worst case: O(height)

  => *height is really important*
- After we insert n elements, what is the worst possible BST height ?



Analysis

- n-1
- so, still O(*n*) for the runway reservation system operations in the worst case, if the height is bad