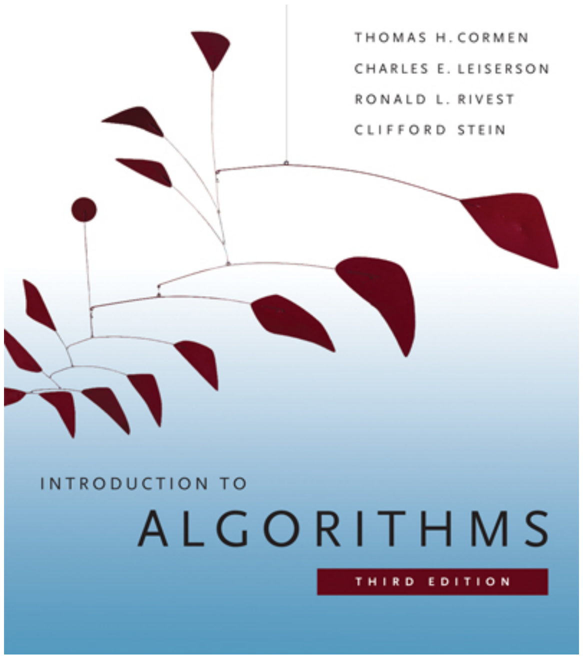
6.006- *Introduction to Algorithms*



Lecture 3

Prof. Piotr Indyk

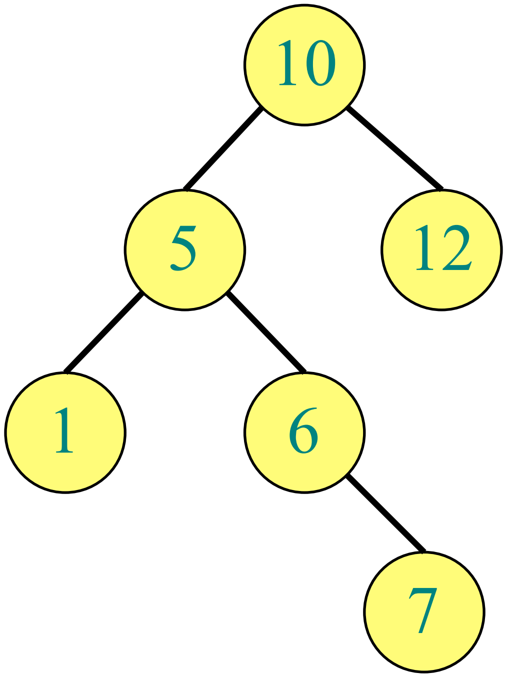
Overview

- Runway reservation system:
  - Definition
  - How to solve with lists

- Binary Search Trees
  - Operations

Readings: CLRS 10, 12.1-3

http://izismile.com/tags/Gibraltar/

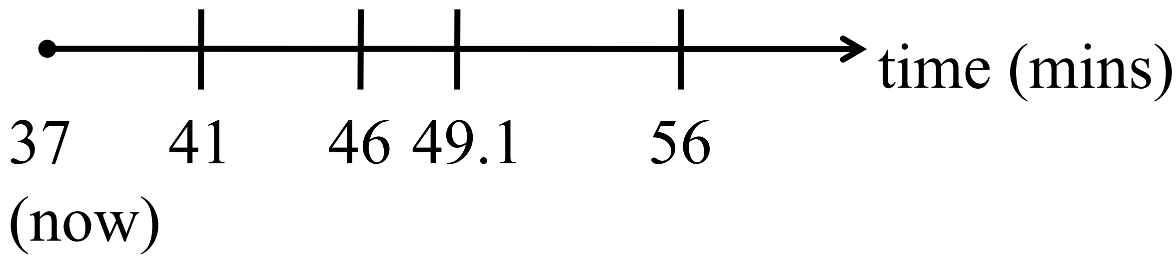


Runway reservation system

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

Runway reservation system

- Example



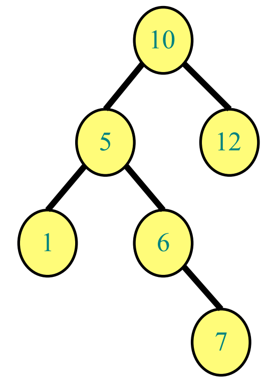
  - R = (41, 46, 49.1, 56)
  - requests for time:
    - 44 => reject (46 in R)
    - 53 => ok
    - 20 => not allowed (already past)
- Ideas for efficient implementation ?

Some options:

- Keep R as an unsorted list
  - Bad: takes linear time to search for collisions
  - Good: can insert t in O(1) time
- Keep R as a sorted array (resort after each insertion)
  - Bad: takes "a lot of" time to insert elements
  - Good: 3 minute check can be done in O(log n) time:
    - Using binary search, find* the smallest i such that R[i]>=t (next larger element)
    - Compare t to R[i] and R[i-1]

Need: *fast* insertion into *sorted* list (sort of)

Binary Search Trees

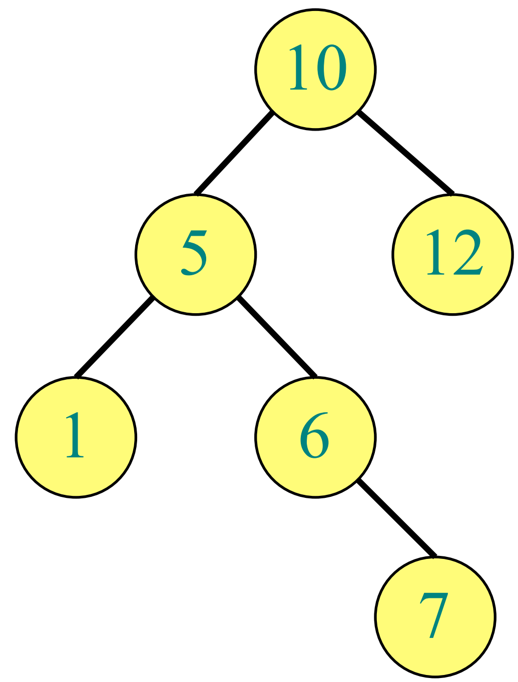


- Simple and natural data structures
- Bulding blocks for

(a,b) tree, 2-3 tree, 2-3-4 tree, AA tree, AVL tree, B

Binary Search Trees (BSTs)

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



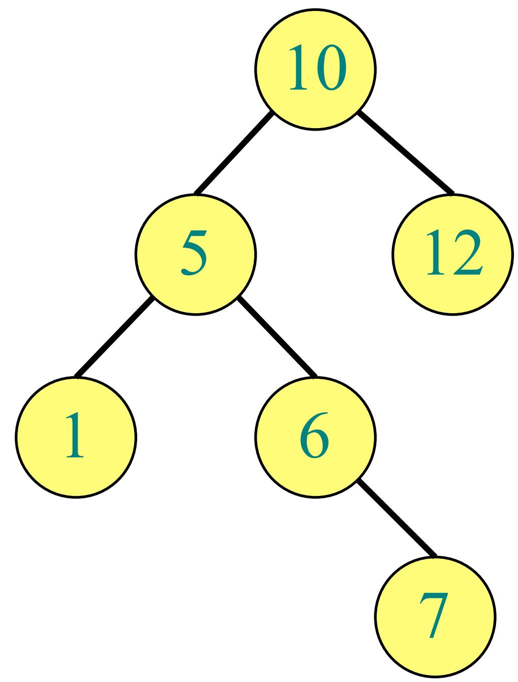
Binary Search Trees (BSTs)

- Property: for any node x:
  - For all nodes y in the left subtree of x:

    $$key[y] \leq key[x]$$

  - For all nodes y in the right subtree of x:

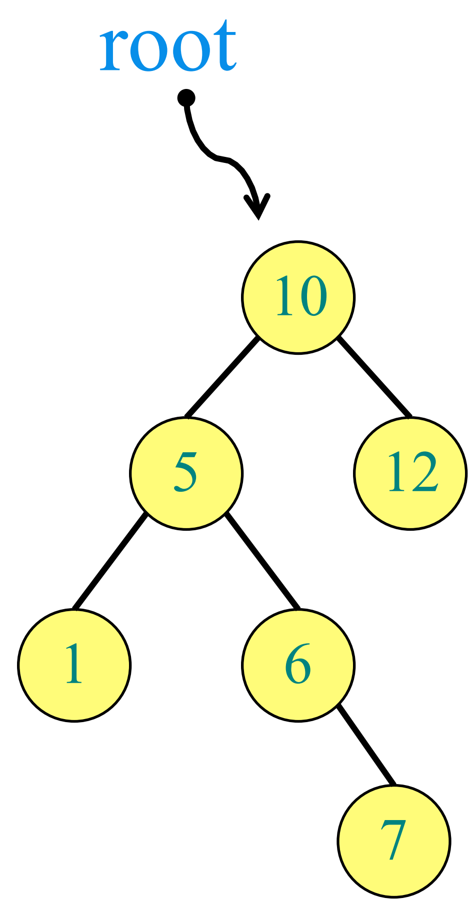
    $$key[y] \geq key[x]$$



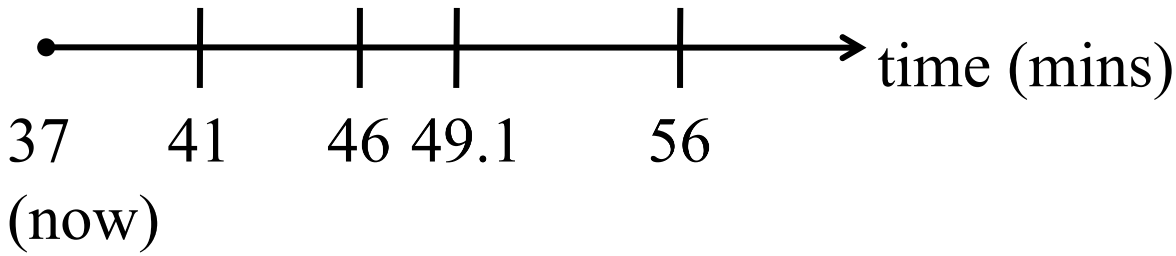
- How are BSTs made ?

Growing BSTs

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



BST as a data structure

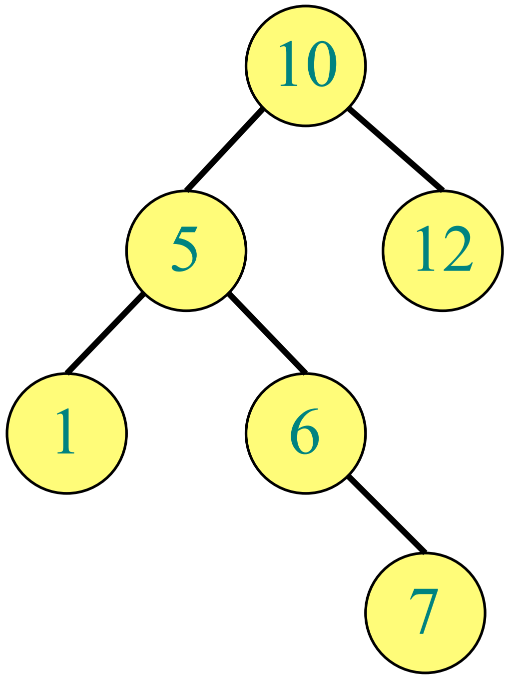


- Operations:
  - insert(k): inserts key k
  - search(k): finds the node containing key k (if it exists)
  - next-larger(x): finds the next element after element x
  - findmin(x): finds the minimum of the tree rooted at x
  - delete(x): deletes node x

Search

Search(k):

- Recurse left or right until you find k, or get NIL



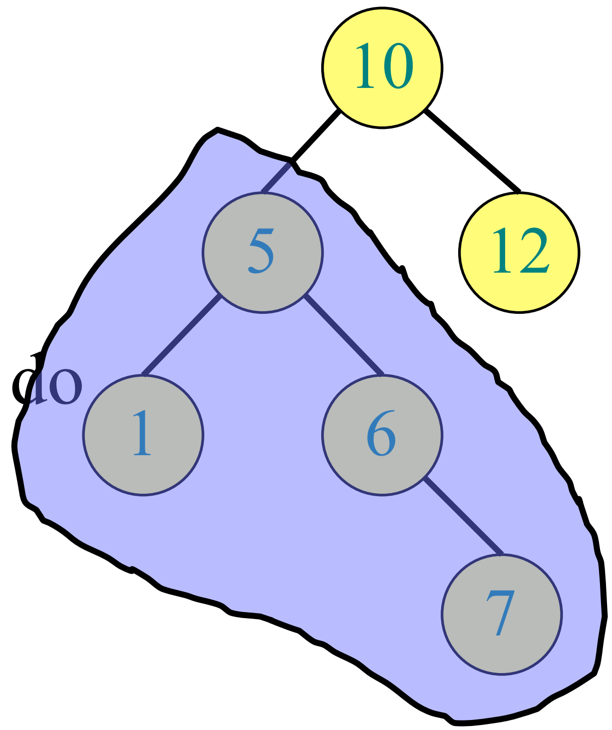
Search(7)

Search(8)

Next-larger

next-larger(x):

- If right[x] ≠ NIL then
  return minimum(right[x])
- Otherwise
  y ← p[x]
  While y≠NIL and x=right[y] do
    - x ← y
    - y ← p[y]
  Return y



next-larger(5)

next-larger(7)

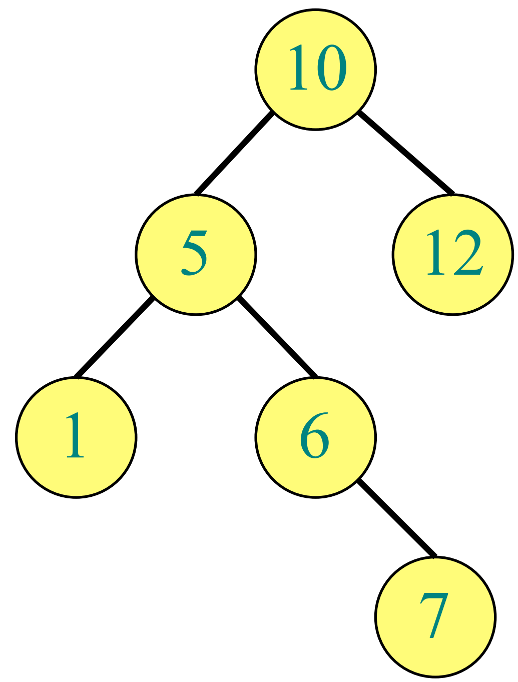
Minimum

Minimum(x)

- While left[x]≠NIL do

  x ← left[x]

- Return x

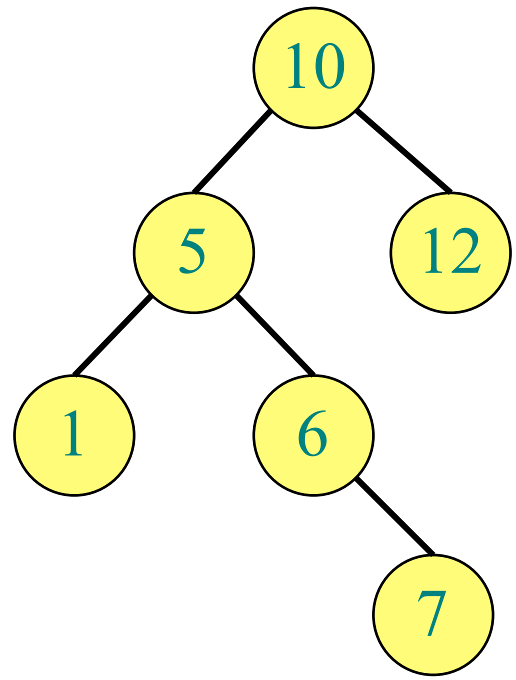


minimum(5)

Analysis

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

  => height 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
- Next lecture: **balanced** BSTs
- **Readings: CLRS 13.1-2**