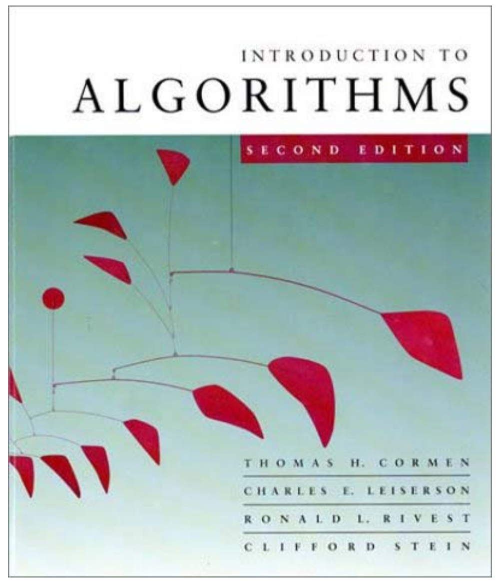
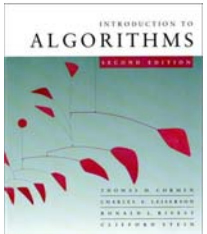
Introduction to Algorithms

6.046J/18.401J



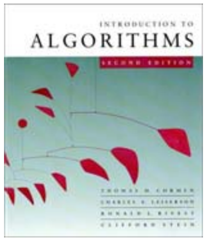
Lecture 8

Prof. Piotr Indyk



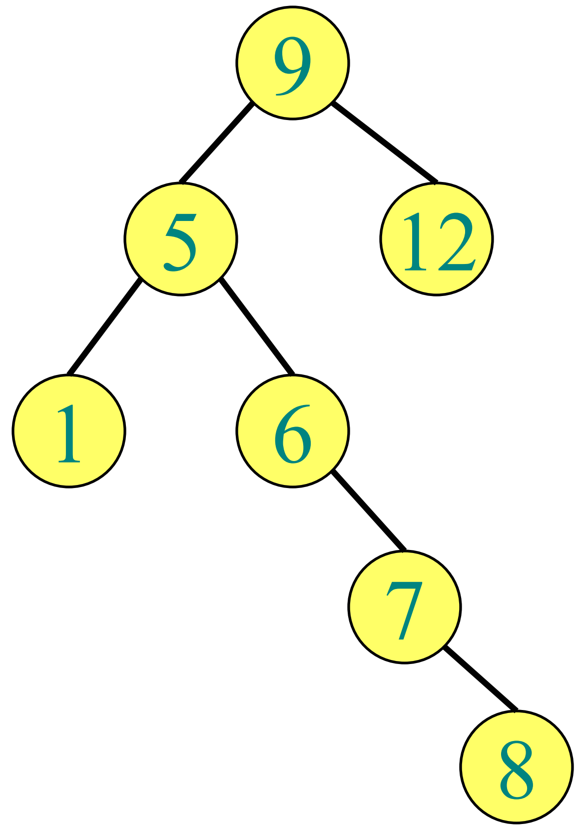
Data structures

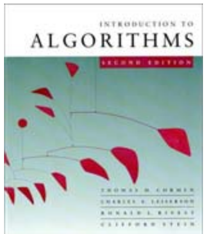
- Previous lecture: hash tables
  - Insert, Delete, Search in (expected) constant time
  - Works for integers from $\{0 \ldots m^r\text{-}1\}$
- This lecture: Binary Search Trees
  - Insert, Delete, Search (Successor)
  - Works in comparison model



Binary Search Tree

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





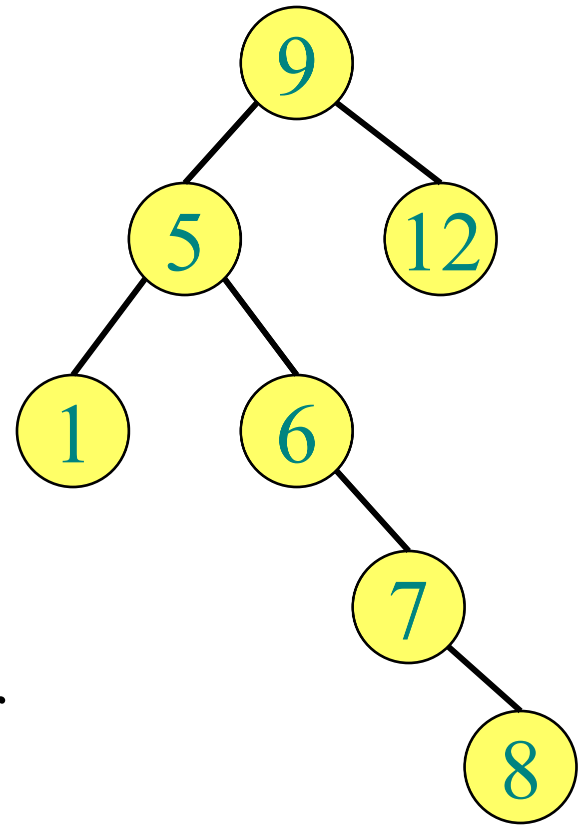
Binary Search Tree (BST)

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

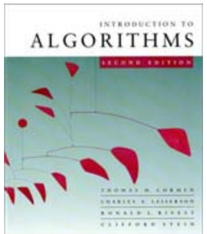
    $$\text{key}[y] \le \text{key}[x]$$

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

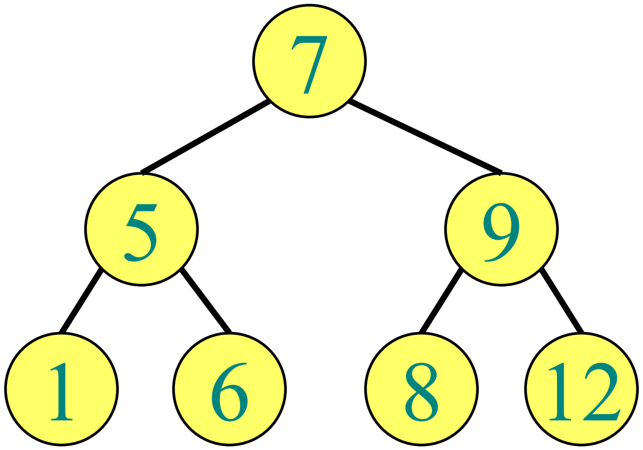
    $$\text{key}[y] \ge \text{key}[x]$$

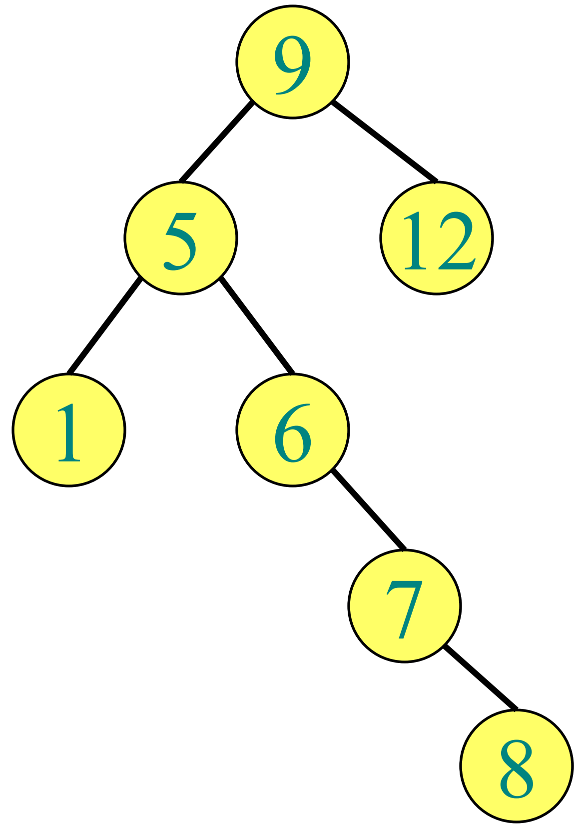


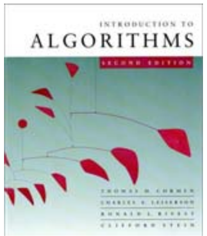
- Given a set of keys, is BST for those keys unique?



No uniqueness



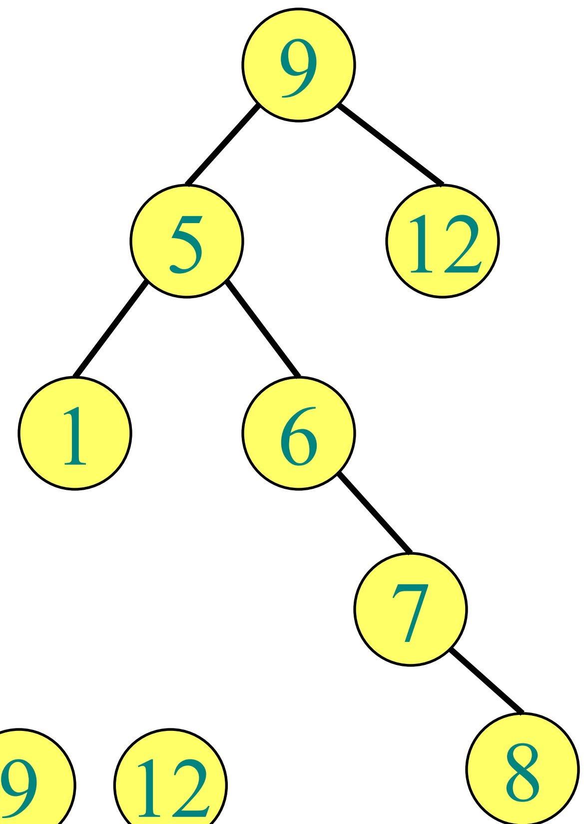


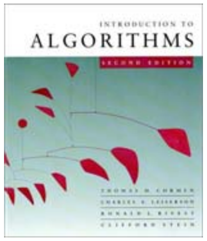


What can we do given BST ?

- Sort !
- Inorder-Walk(x):

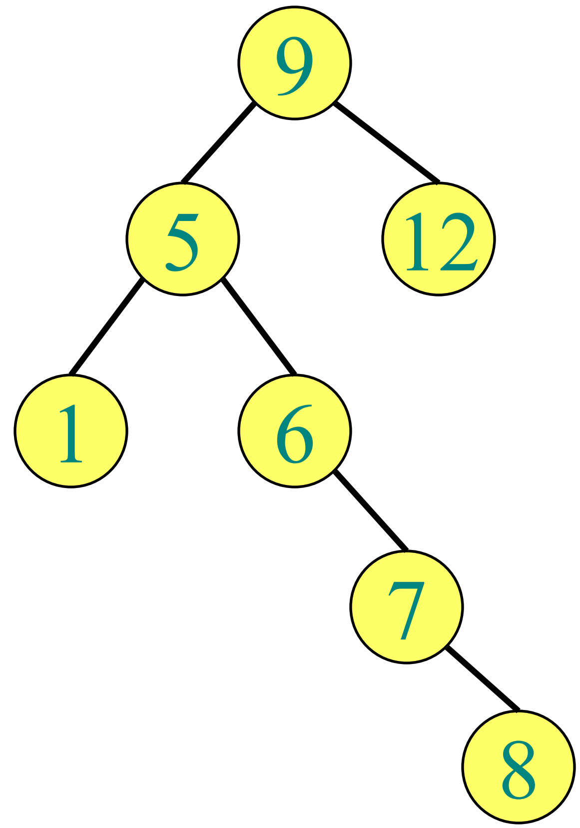
  If x≠NIL then
  - Inorder-Walk(left[x])
  - print key[x]
  - Inorder-Walk(right[x])
- Output: 1 5 6 7 8 9 12

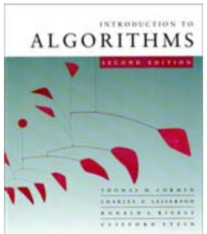




Sorting, ctd.

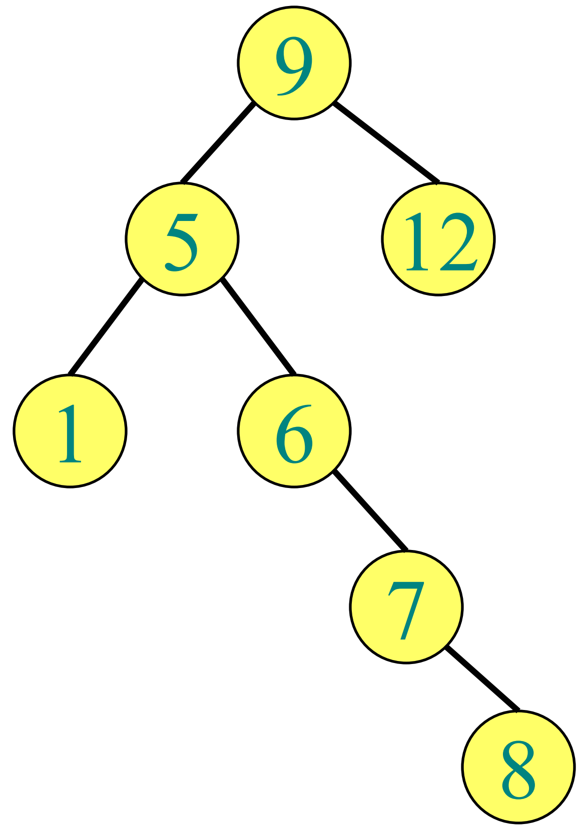
- What is the running time of Inorder-Walk?
- It is O(n)
- Because:
  – Each link is traversed twice
  – There are O(n) links

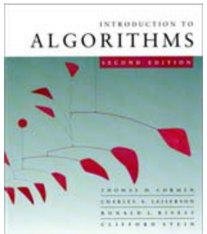




Sorting, ctd.

- Does it mean that we can sort n keys in O(n) time ?
- No
- It just means that building a BST takes Ω(n log n) time
(in the comparison model)





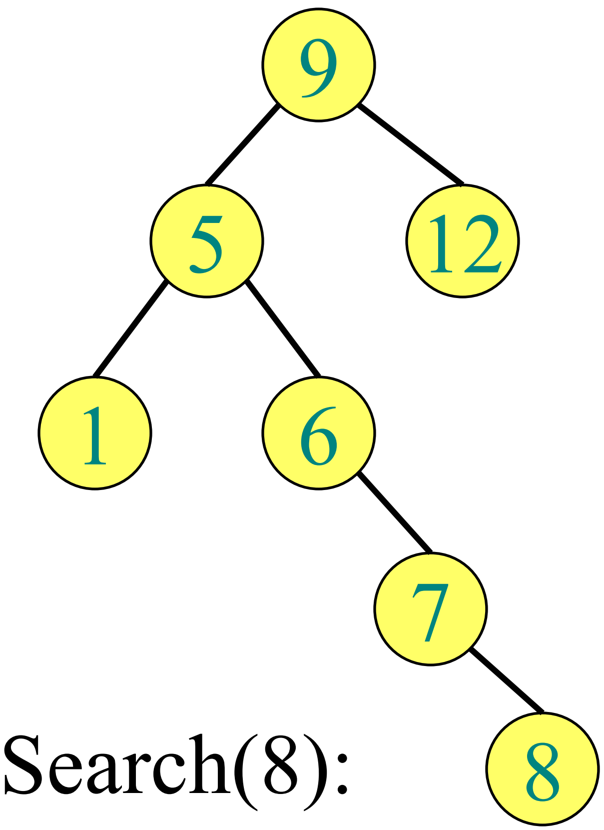
BST as a data structure

- Operations:
  - Insert(x)
  - Delete(x)
  - Search(k)

Search

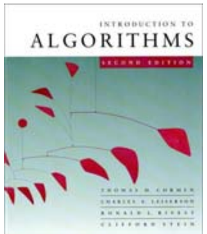
Search(x):

- If x≠NIL then
  - If key[x] = k then return x
  - If k < key[x] then return Search(left[x])
  - If k > key[x] then return Search(right[x])
- Else return NIL



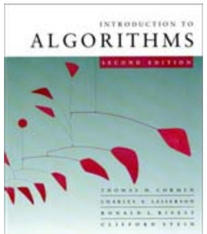
Search(8):

Search(8.5):



Predecessor/Successor

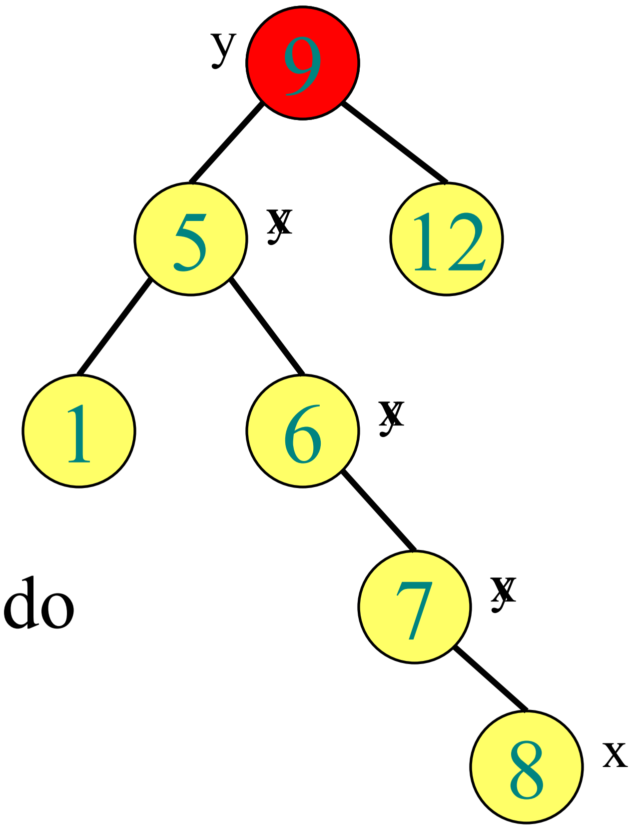
- Can modify Search (into Search’) such that, if k is not stored in BST, we get x such that:
  - Either it has the largest $key[x]<k$, or
  - It has the smallest $key[x]>k$
- Useful when k prone to errors
- What if we always want a successor of k ?
  - x=Search’(k)
  - If $key[x]<k$, then return Successor(x)
  - Else return x

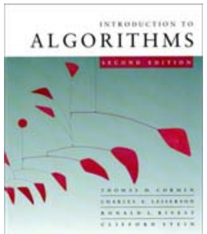


Successor

Successor(x):

- If right[x] ≠ NIL then return Minimum(right[x])
- Otherwise
  - $y \leftarrow p[x]$
  - While y≠NIL and x=right[y] do
    - $x \leftarrow y$
    - $y \leftarrow p[y]$
  - Return y

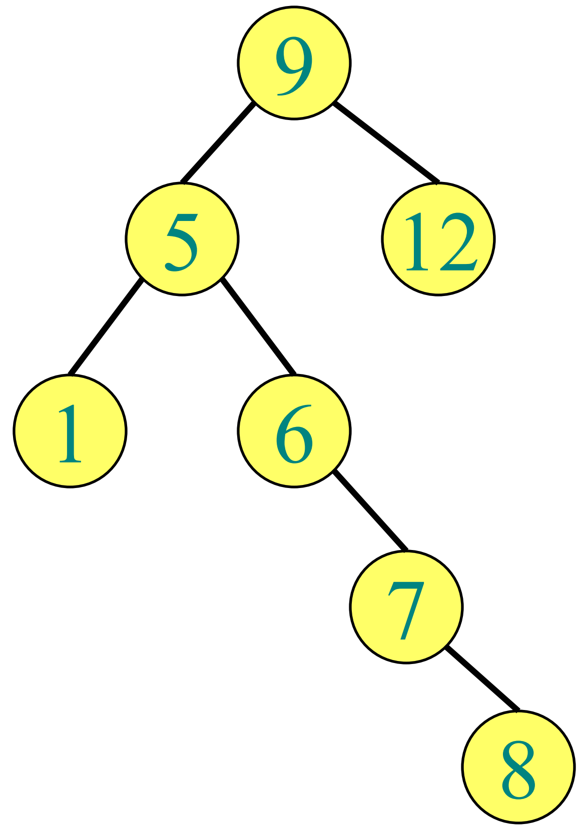


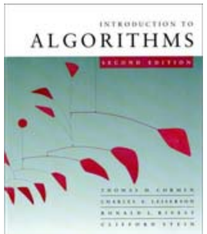


Minimum

Minimum(x)

- While left[x]≠NIL do
  - x ← left[x]
- Return x



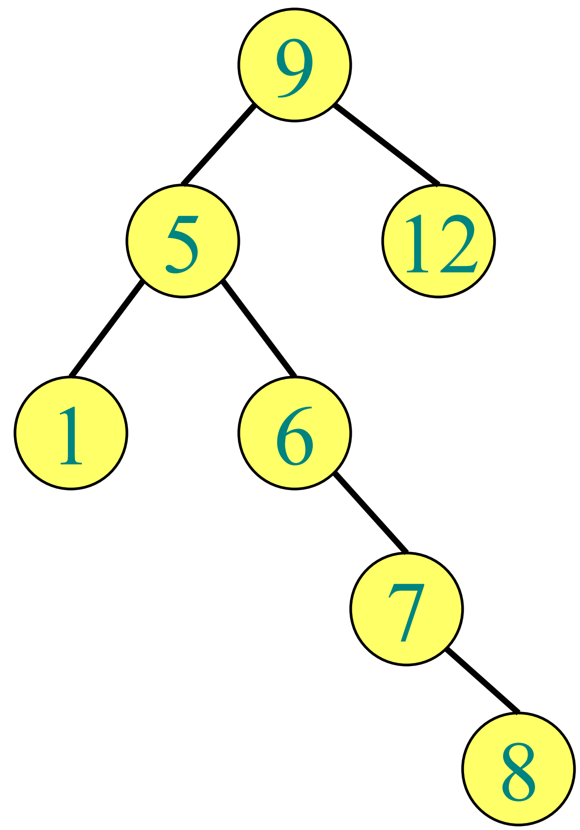


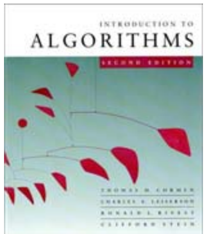
Nearest Neighbor

- Assuming keys are numbers
- For a key k, can we find x such that |k-key[x]| is minimal ?
- Yes:
  - key[x] must be either a predecessor or successor of k
  - y=Search'(k) //y is either succ or pred of k
  - y' =Successor(y)
  - y''=Predecessor(y)
  - Report the closest of key[y], key[y'], key[y'']

Analysis

- How much time does all of this take ?
- Worst case: O(height)
- Height really important
- Tree better be balanced

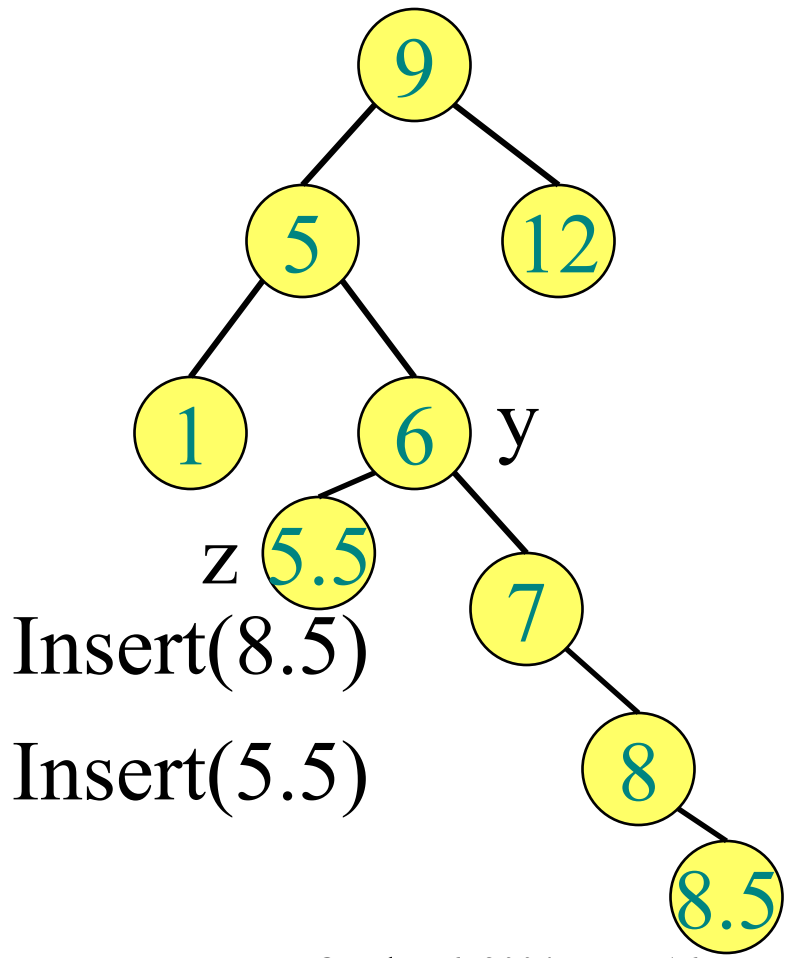




Constructing BST

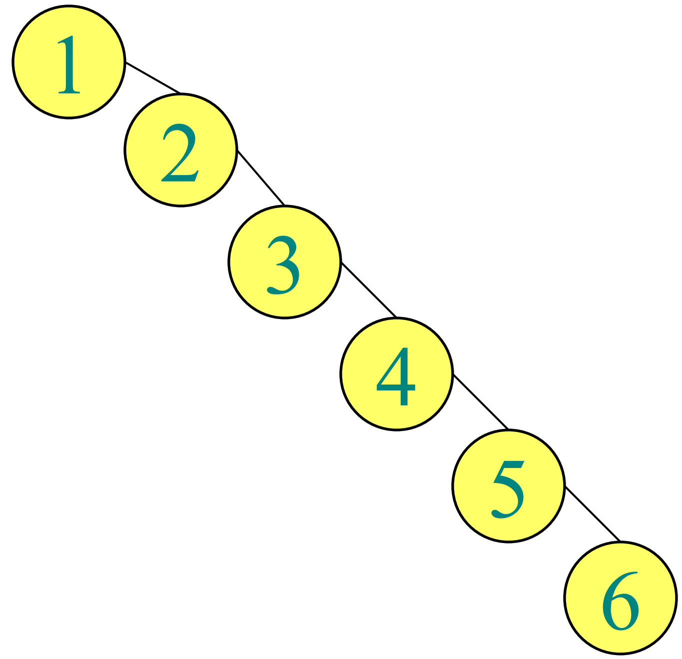
Insert(z):

- y ← NIL
- x ← root
- While x ≠ NIL do
  - y ← x
  - If key[z] < key[x]
    then x ← left[x]
    else x ← right[x]
- p[z] ← y
- If key[z] < key[y]
  then left[y] ← z
  else right[y] ← z



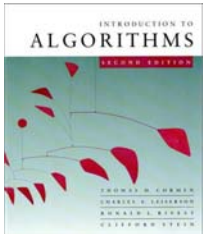
Analysis

- After we insert n elements, what is the worst possible BST height ?
- Pretty bad: n-1



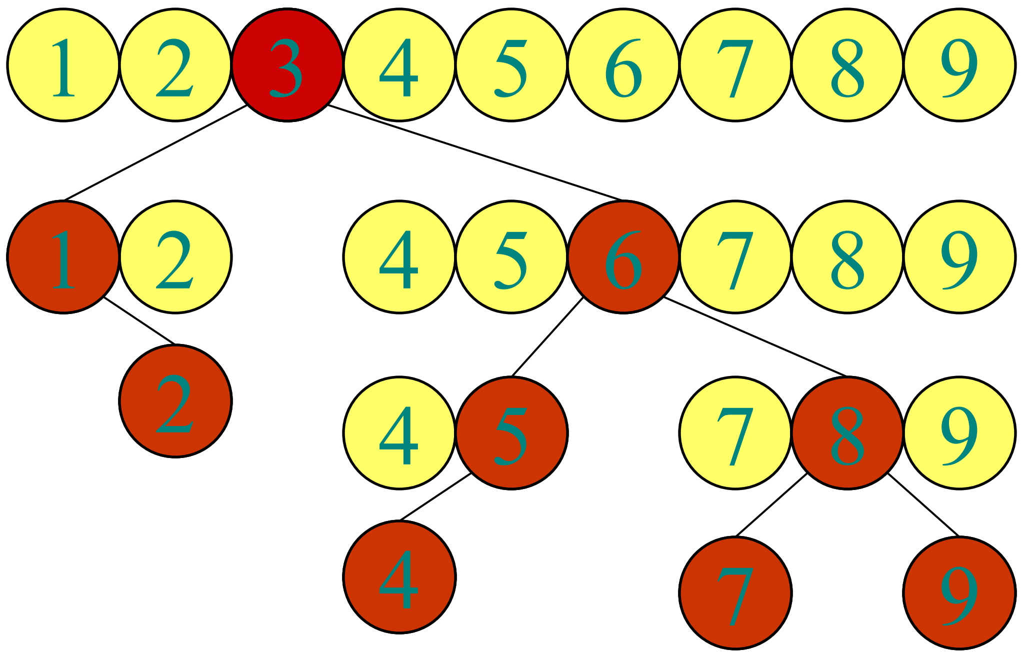
Average case analysis

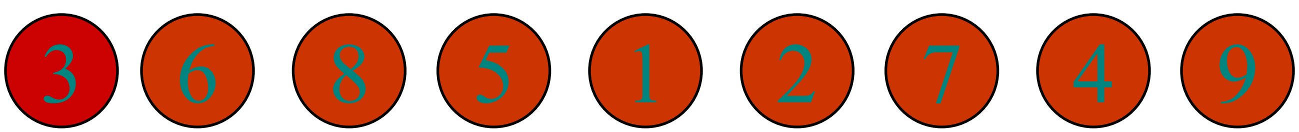
- Consider keys 1,2,…,n, in a random order
- Each permutation equally likely
- For each key perform Insert
- What is the likely height of the tree ?
- It is O(log n)

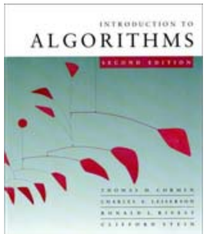


Creating a random BST

- n=9





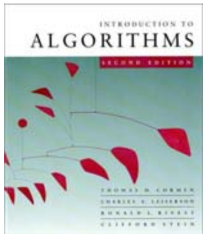


Observations

- Each edge corresponds to a random partition
- Element x has height $h \Rightarrow x$ participated in h partitions
- Let h_x be a random variable denoting height of x
- What is $\Pr[h_x > t]$, where $t = c \lg n$?

Partitions

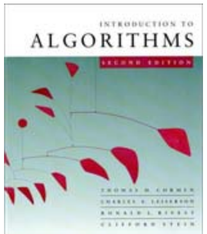
- A partition is lucky if the ratio is at least 1:3, i.e., each side has size $\geq 25\%$
- Probability of lucky partition is ½
- After $\log_{4/3} n$ lucky partitions the element becomes a leaf
- $h_x > t \Rightarrow$ in $t = c \log_{4/3} n$ partitions we had $< \log_{4/3} n$ lucky ones
- Toss $t = c \log_{4/3} n$ coins, what is the probability you get $< k = \log_{4/3} n$ heads ?



Concentration inequalities

- CLRS, p. 1118: probability of at most k heads in t trials is at most $\binom{t}{k}/2^{t-k}$

$$
\begin{aligned}
\Pr[h_x > t] &\le \binom{t}{k}/2^{t-k} \\
&\le (et/k)^k/2^{t-k} \\
&= (ce)^{\log_{4/3} n}/2^{(c-1)\log_{4/3} n} \\
&= 2^{\lg(ce)\log_{4/3} n}/2^{(c-1)\log_{4/3} n} \\
&= 2^{[\lg(ce)-(c-1)] * (\lg n)/\lg(4/3)} \\
&\le 2^{-1.1 \lg n} = 1/n^{1.1}, \text{ for sufficient } c
\end{aligned}
$$



Final Analysis

- We know that for each x, $\Pr[h_x > t] \leq 1/n^{1.1}$
- We want $\Pr[h_1 > t \text{ or } h_2 > t \text{ or } \ldots \text{ or } h_n > t]$
- This is at most

$$\Pr[h_1>t]+\Pr[h_2>t]+\ldots+\Pr[h_n>t]$$

$$\leq n * 1/n^{1.1}$$

$$= 1/n^{0.1}$$

- As n grows, probability of height $> c \lg n$ becomes arbitrarily small

Summing up

- We have seen BSTs
- Support Search, Successor, Nearest Neighbor etc, as well as Insert
- Worst case: O(n)
- But O(log n) on average
- Next week: O(log n) worst case