CSE 101-01 5-23-23

Mid 2 : Th May 25th

PaG : ext. last time to Fri. 10PM
(last one!)

BST stuff

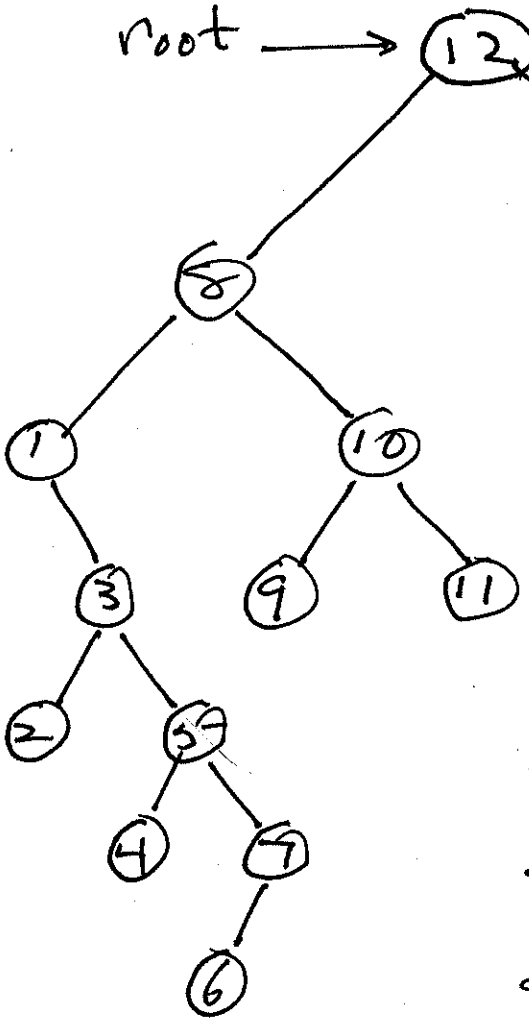
Queries

- TreeSearch(x, k) (x: node, k: target key)
- TreeMinimum(x)
- TreeMaximum(x)
- TreeSuccessor(x)
- TreePredecessor(x)

Defn

The **successor** of a node x is the next node after x to be processed in an I.O.T.W.

Ex.



I.O.T.W:

1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13

successor(1) = 2

successor(8) = 9

successor(7) = 8

successor(13) = nil

2 cases for successor:

case 1: x has a right child

successor of x is the leftmost descendant of that child

TreeMinimum(x.right)

case 2: x has no right child

successor of x is the lowest ancestor of x whose left child is also an ancestor of x.

Exercise:

- characterize predecessor of x in two cases
- write TreePredecessor(x)

Runtimes: let n = # nodes

- Traversals: In, Pre, Post

  cost = $\Theta(n)$

- Queries: Search, Min, Max, Successor, Predecessor

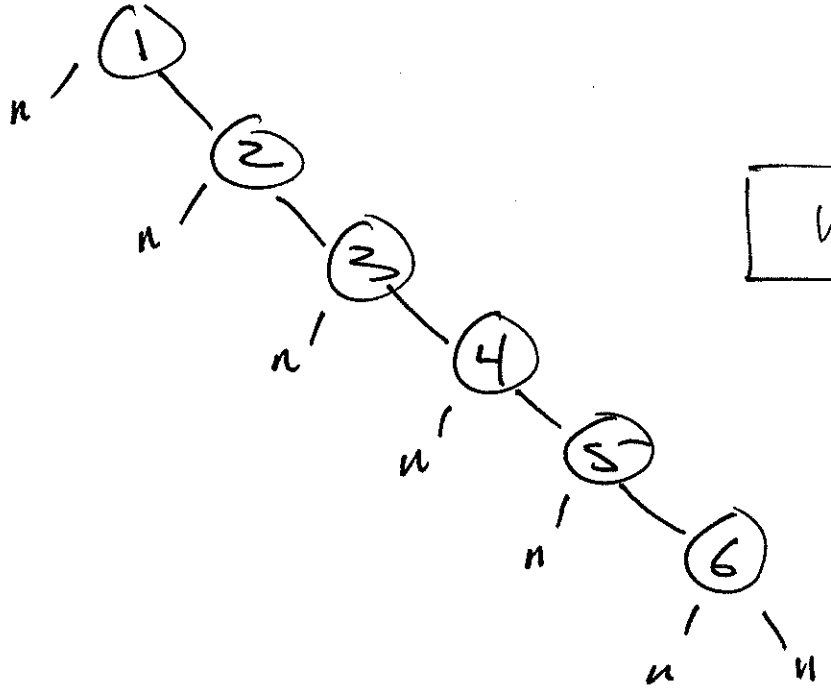
Defn

- height(T) = dist. from root to the deepest leaf.
- height(x) = height of subtree rooted at x.

cost of Queries (worst case)

$= \Theta(\text{height}(T))$

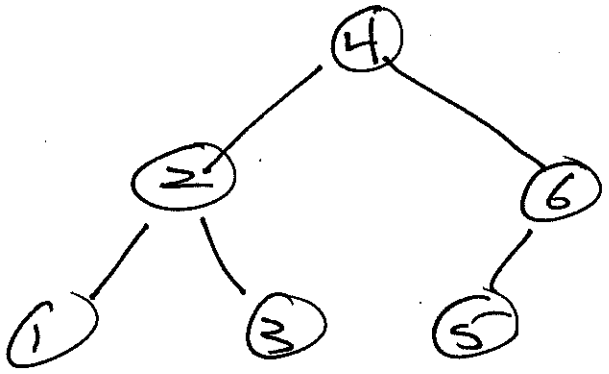
Worst Possible BST for Queries:



$n = 6$

cost: $\Theta(n)$
since height = 5

Best Possible BST!



$n = 6$

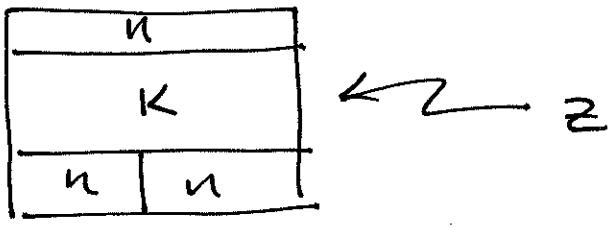
height = 2

In general: $\underbrace{\log n}_{\text{balanced tree}} \leq \text{height}(T) \leq n-1$

12.3 in CLRS: Insertion & Deletion

to insert and maintain BST prop.

- set z.key = k
- set z.left = z.right = z.parent = nil.



- find a node to adopt z as left or right child

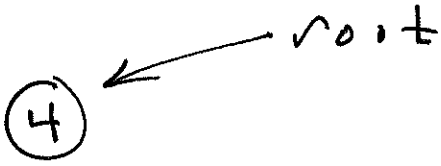
how?

search for key k in T, find nil child where k belongs, and put it there.

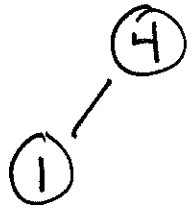
Ex. Keys: 4 1 7 2 6 3 5

T.root = nil

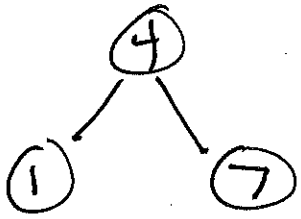
insert(4) :



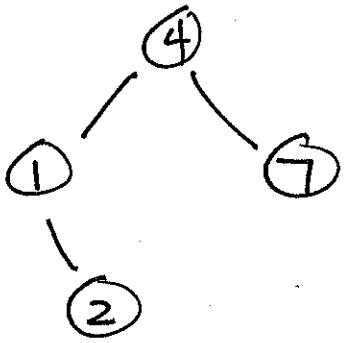
insert(1) :



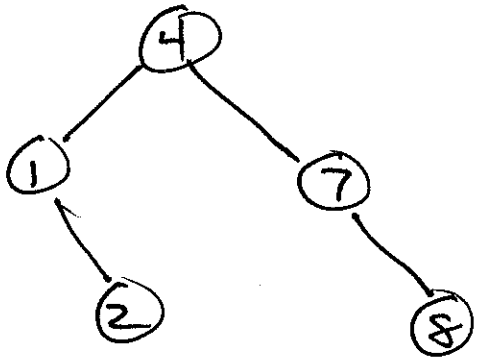
insert(7) :



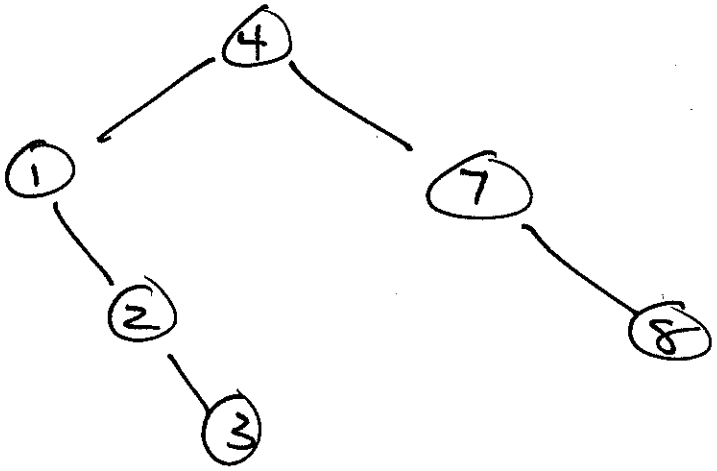
insert(2) :



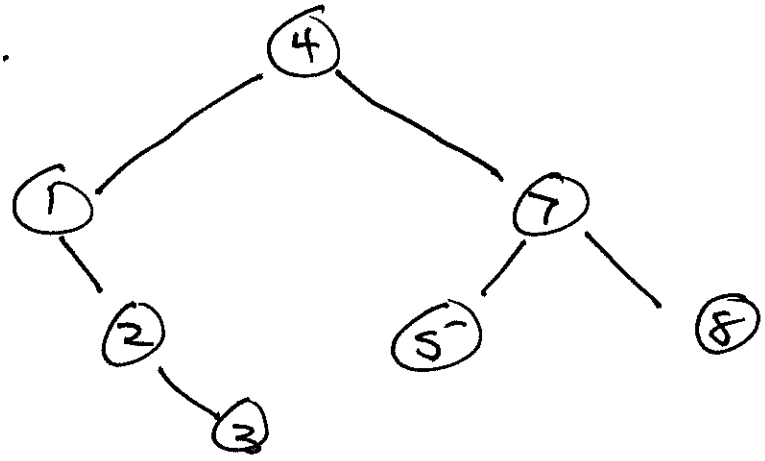
insert(8):



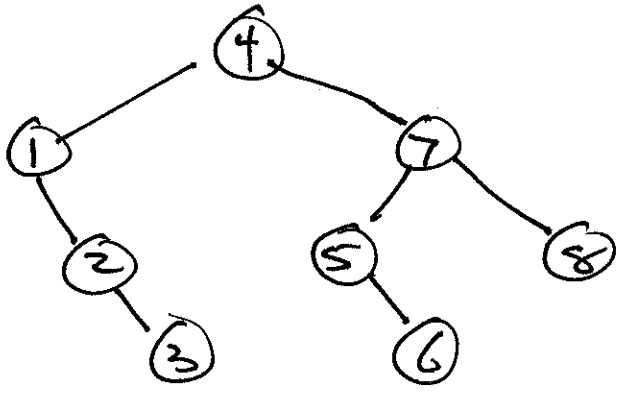
insert(3):



insert(5):



insert(6):



note:

- Insert Keys in Pre O.T.W will re-produce the BST

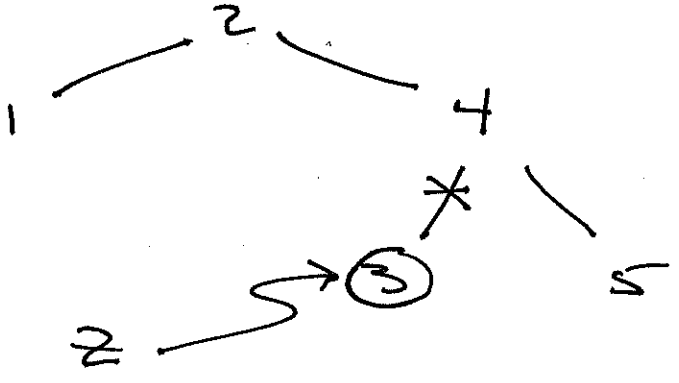
- Insert Keys in reverse Post O.T.W. order will also reproduce BST.

Deletion: 3 cases

goal: delete z from T.

case 1: z has no children.

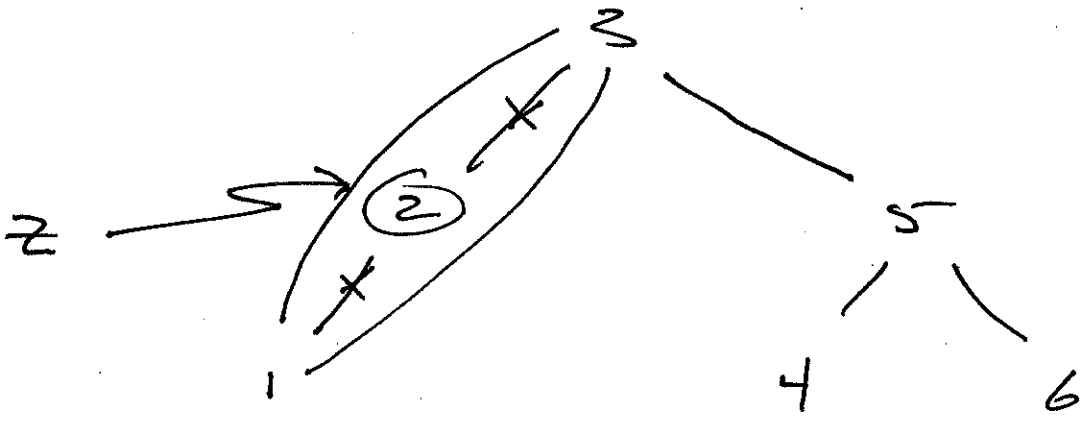
detach z from T



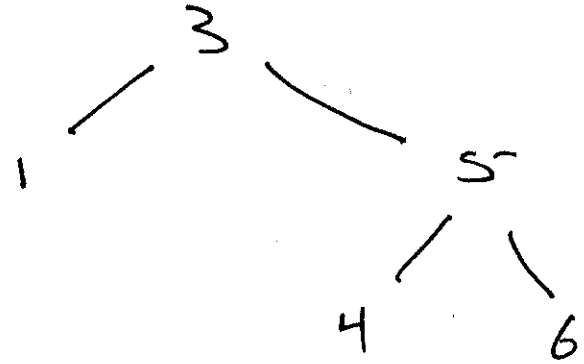
case 2 : z has 1 child

2.1 : z has right but no left

2.2 : " " left " " right

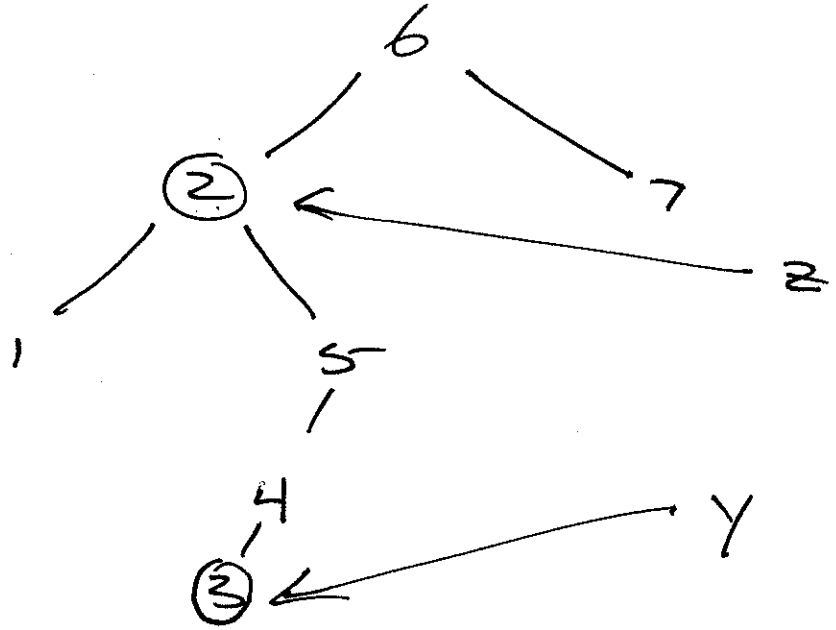


result of delete(z)

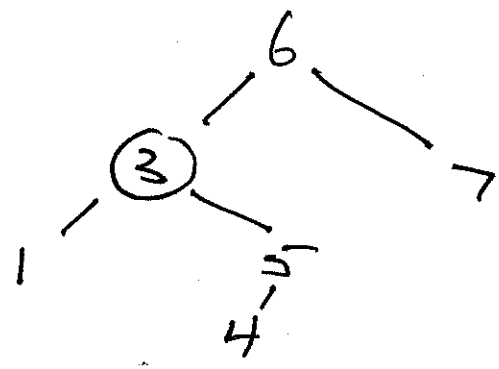


case 3 : z has 2 children

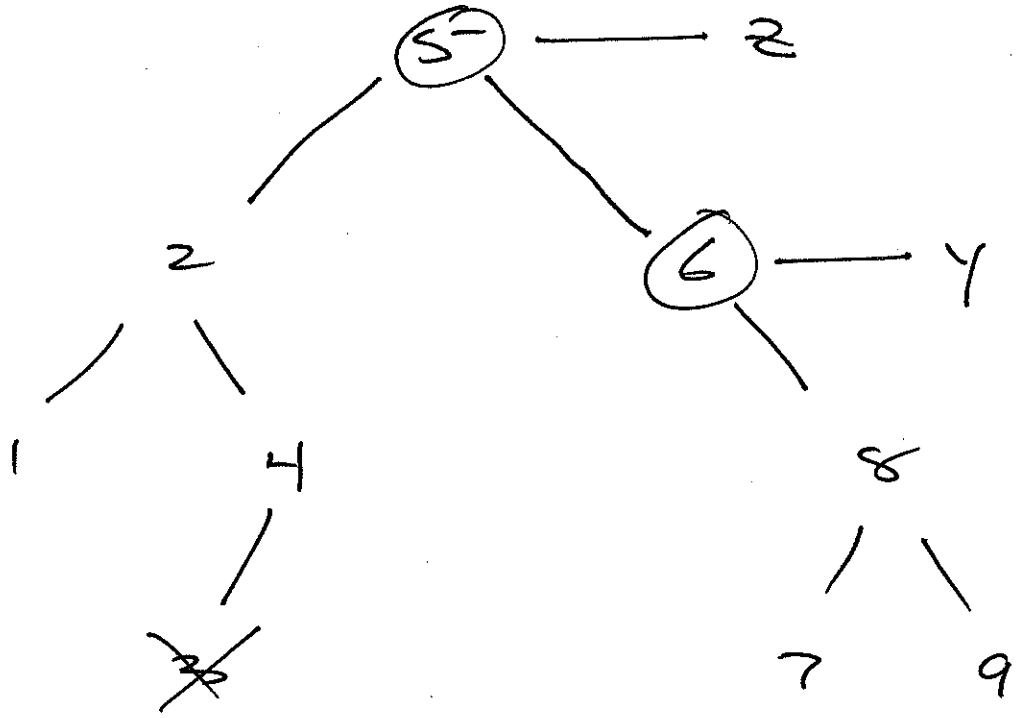
⇒ splice out z's successor y (which has no left child), then replace z by y.



replace (2) by (3)



Ex. insert: 5 2 6 4 3 8 7 1 9



delete: 3, 5