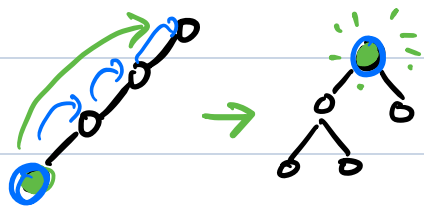
BST
- traditional BST
- AVL
- Red Black tree
- Splay tree



BST(ADT) : Find / Delete / Insert amortized cost of $O(\log n)$, occasionally $O(n)$ will happen.

As we traverse the tree, we "compress" so that subsequent traversals become cheaper.

Beneficial when we have "data locality"

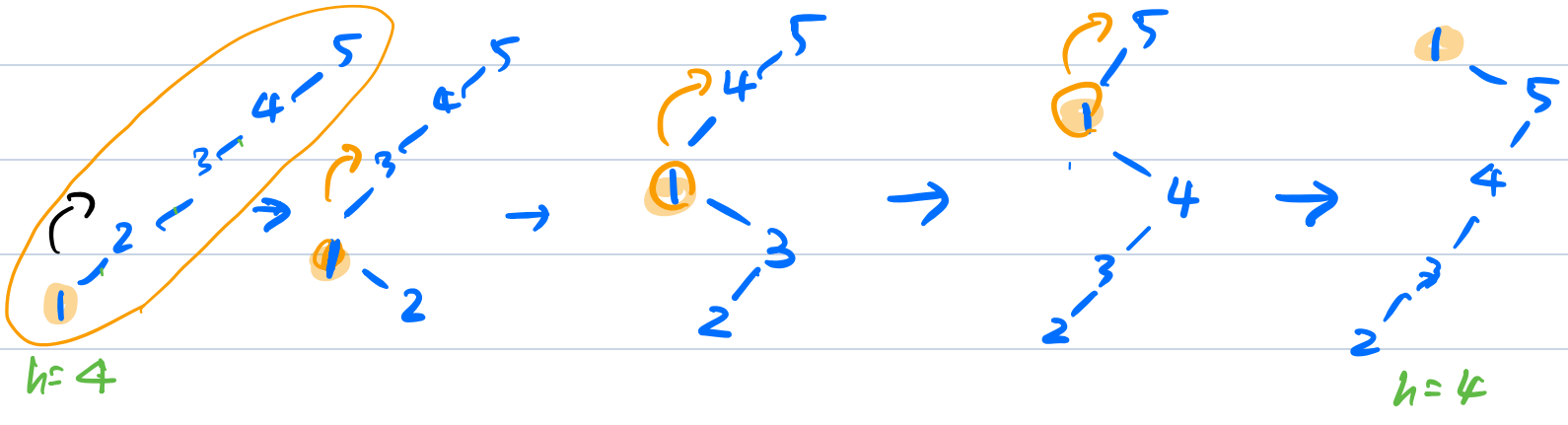
keeping popular items toward / closer to the root will make future operations cheaper / faster.

Splay tree does not rely on this data locality.

It still works $O(\log n)$ in the worstcase w/o data locality.

Splaying
1. move the element accessed to the root
2. compress the tree, making tree height shorter.

1. "Move to root heuristic" ✗ Find(1) x

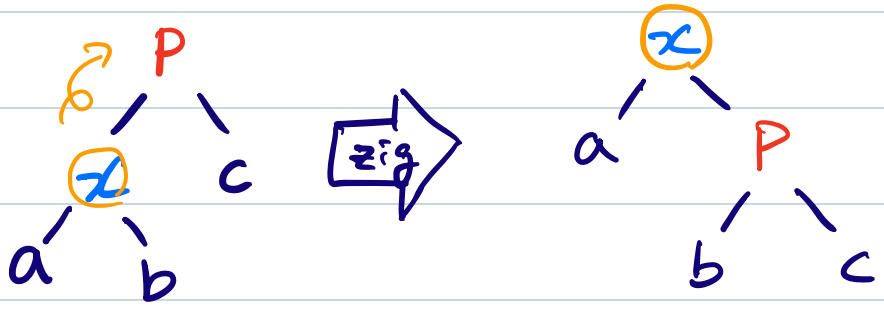


2. Splay operation (Sleator, Tarjan, 1985) Find(x)

Do below operations recursively until x reaches to root

If x is not found still splay x' (x' is the last node accessed)

case 1 : zig If x's parent is the overall root, do a single rotation to make x the new root



(a, b, c are arbitrary subtrees)

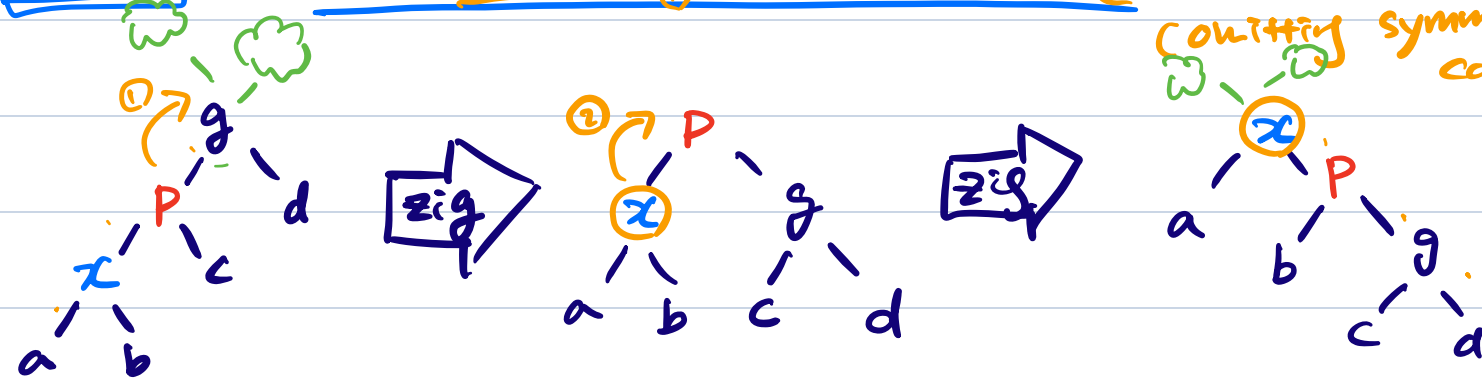
omitting symmetric case

case 2 : zig zig (x's parent is Not the overall root)

if x and p are both left children (right children)

perform double rotation in the same direction.

order ! : left p first, then left x



(omitting symmetric case)

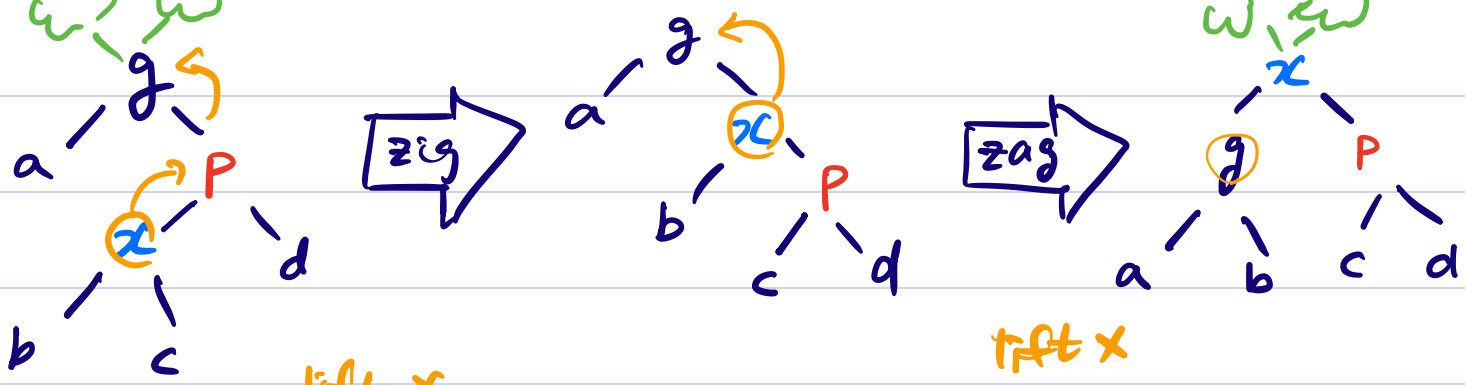
case 3: zig zag double rotation in the opposite direction (AVL)

(p is Not the overall root)

If x is a left child and p is a right child (vice versa)

do double rotations in the opposite direction.

(left x twice)



3 Example of extreme cases

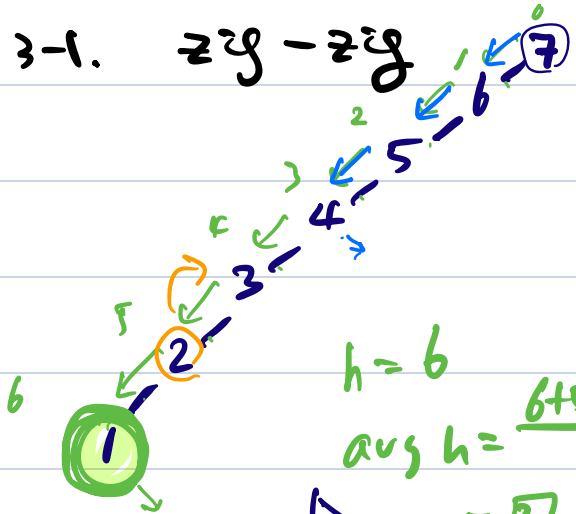
Find(1)

3-1. zig-zig

Find (1.5) → splay 1

Find (4.2) → splay 4

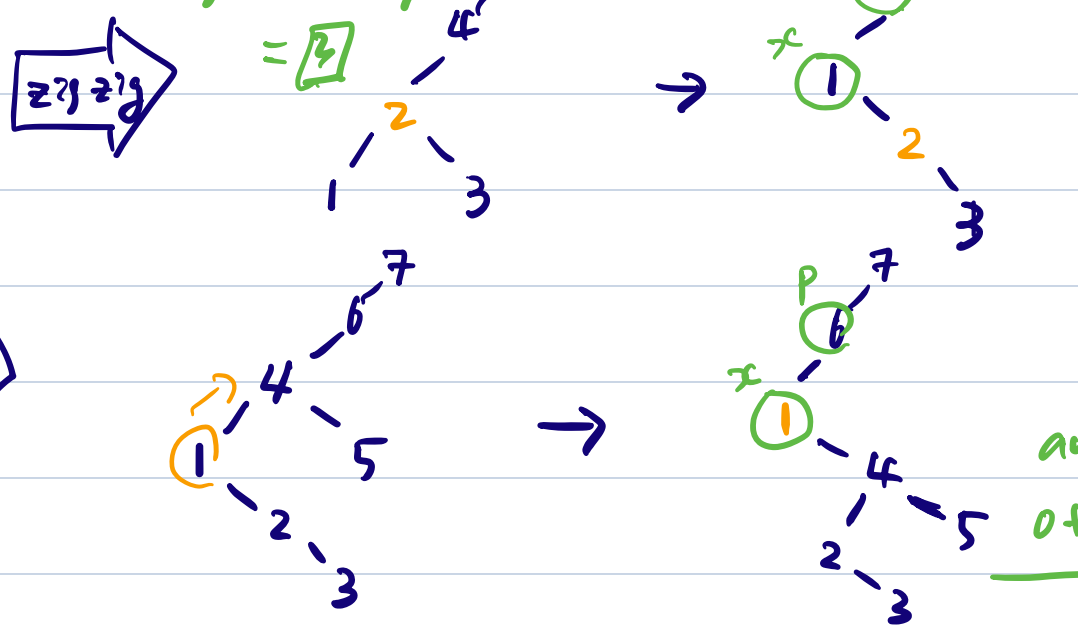
Find (0) → splay 7



h = 6

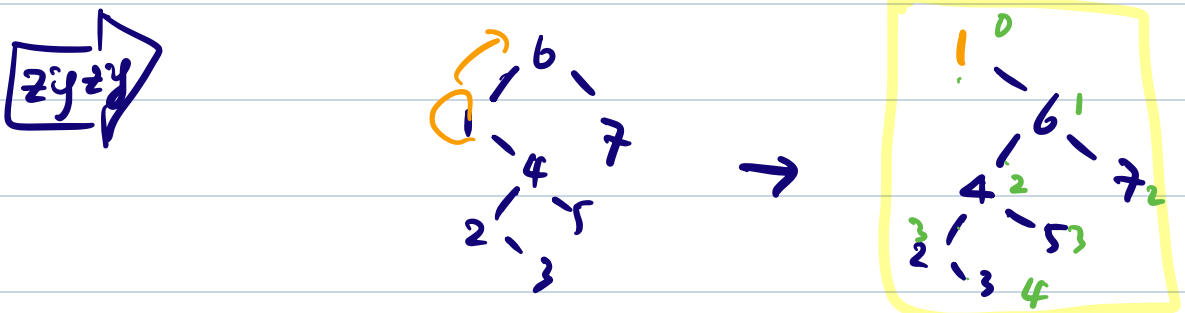
avg h = $\frac{6+5+\cdots+0}{7}$

= 3



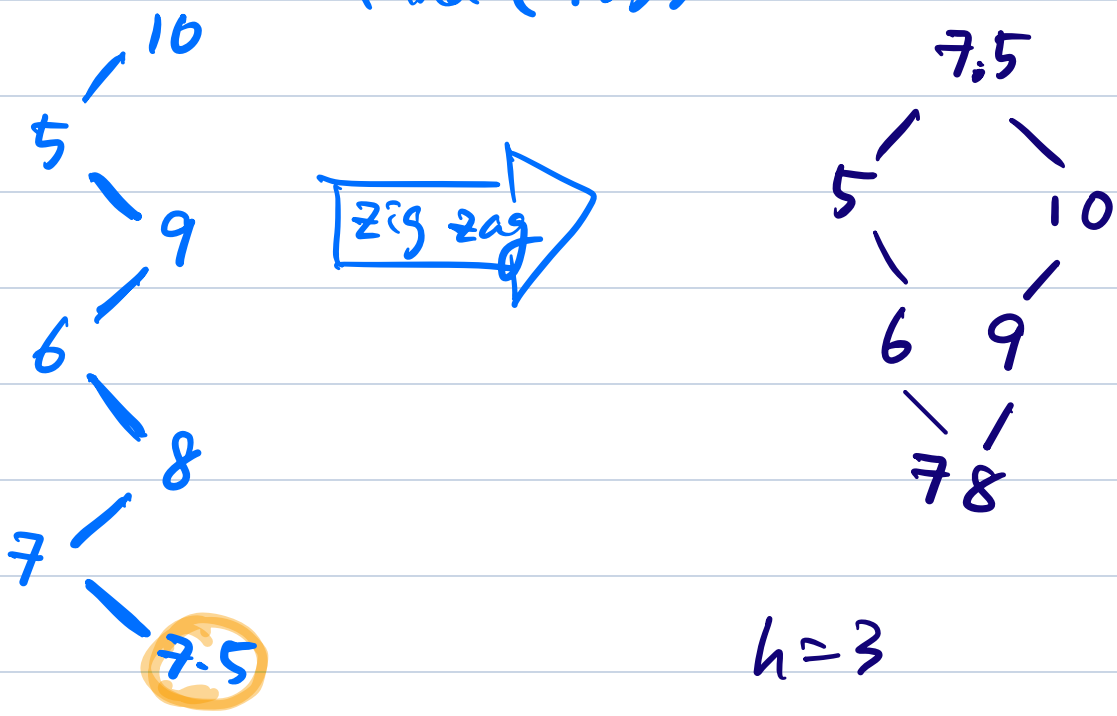
avg = $\frac{0+1+2+2+3+3+4}{7}$

= 2.1



3-2.

Find (7.5)



$h = 6$

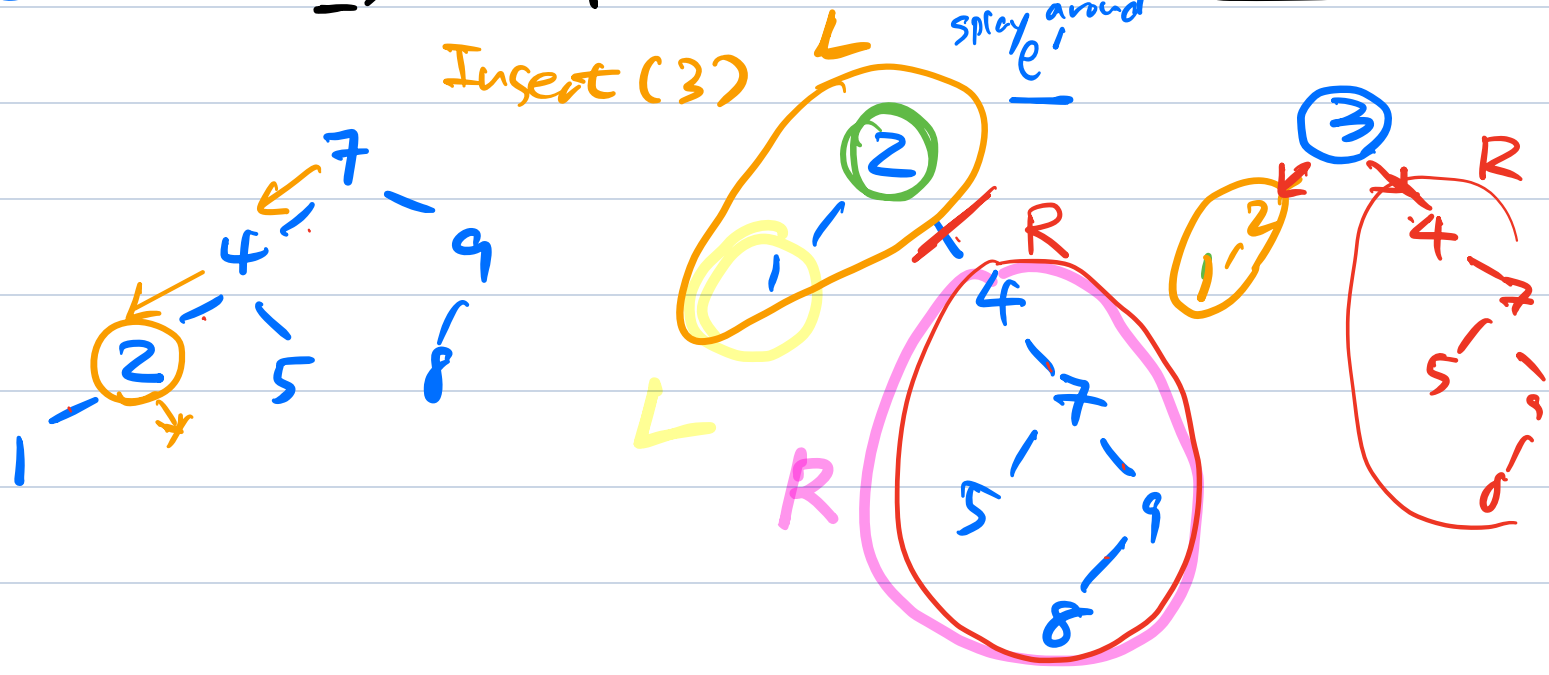
$h = 3$

4. ① Find : Step 1. Ordinary BST search $O(h)$

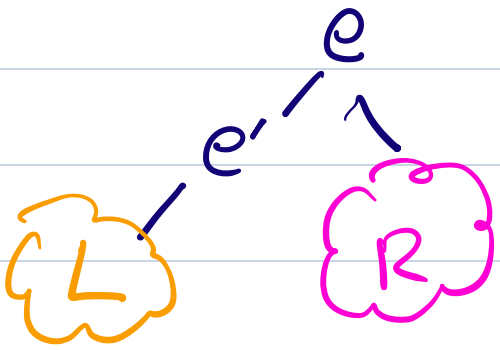
Step 2. Splay around x, x is the last node accessed

$O(h)$

② Insert(e) : Step 1. BST search on e: Find(e)



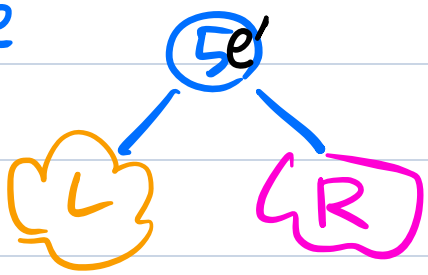
case1. $\frac{e'}{2} < \frac{e}{3}$

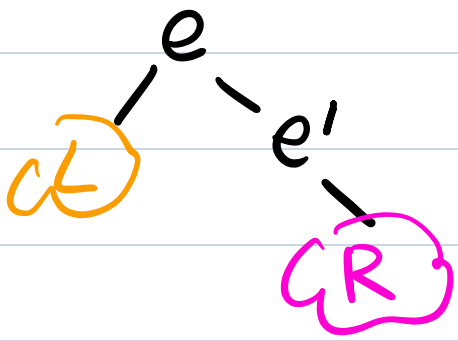


step2. create a new node e and make e the root form the tree as shown

case2. $e' > e$

insert(45) $e' = 5$





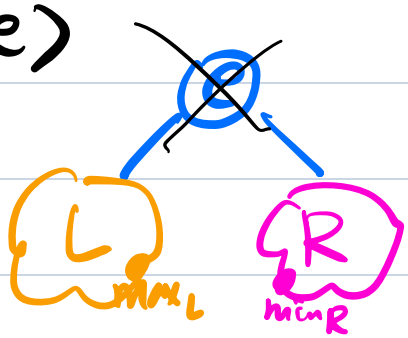
time complexity $2 \times O(h) + O(1)$

③ Delete(e)

step1. BST search on e splay around e

Find(e)

step2. Node to delete e is at the root



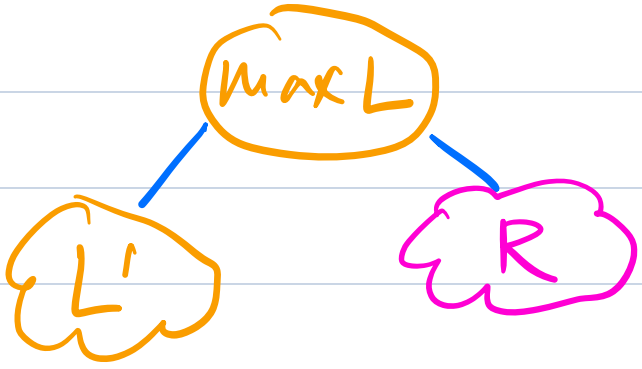
step3. delete root. $O(h)$

Find max on subtree L + splay around the max L

time complexity of delete

$O(h) + O(h) + O(1) + O(h) + O(h)$

- $O(h)$: BST search once
- $O(h)$: splay around e
- $O(h)$: findmax
- $O(h)$: splay around maxL



$L' = L - \{maxL\}$

Motivation for the top-down approach!!