9) Lecture 20-22 AVL Tree

10 September 2024 14:31

BST : time complexity of search, insertion, deletion } $O(h)$

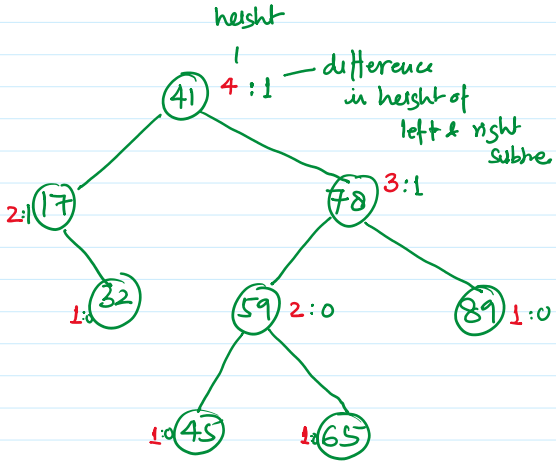
$h = O(n)$ $n :=$ # of nodes in BST.

AVL tree

(i) BST

(ii) For every internal node in BST difference between height of left subtree & right subtree is at most 1.

leaf node is at height 1



This is an AVL tree

Th: The height of AVL tree of n node is $O(\log n)$.

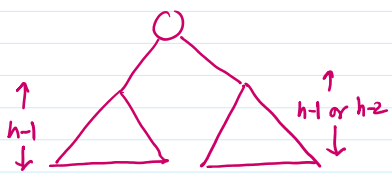
Proof: $n(h) :=$ min no. of nodes in an AVL tree of height h.

Compute $n(h)$?

$n(1) = 1$ ○ leaf node

$n(2) = 2$ ○—○ or ○—○

For $h \geq 3$; an AVL tree consist of



- root node
- one AVL tree (left or right subtree) of height h-1
- one " " of height either (h-1) or (h-2)

- one AVL tree (left or right subtree) of height $h-1$
- one " " of height either $(h-1)$ or $(h-2)$

$$n(h) = 1 + n(h-1) + n(h-2)$$

$$n(h) > n(h-1) + n(h-2) \qquad n(h-2) \leq n(h-1)$$

$$n(h) > 2\, n(h-2)$$

$$> 2^2\, n(h-4)$$

$$\vdots$$

$$n(h) > 2^i\, n(h-2i) \qquad \text{choose } i \text{ s.t } h-2i = 2$$

$$n(h) > 2^{\frac{h}{2}-1}\, n(2) \qquad \because n(2) = 2$$

$$n(h) > 2^{\frac{h}{2}}$$

$$\Rightarrow h < 2 \log n(h)$$

$$\Rightarrow \boxed{h = O(\log n(h))}$$

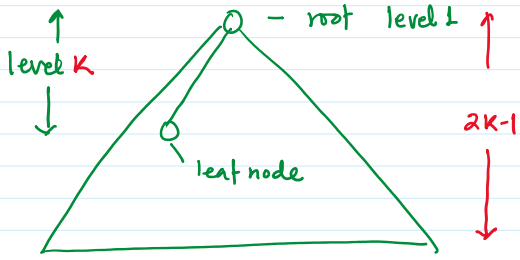
The height of AVL tree is $O(\log n)$.

Properties of AVL tree

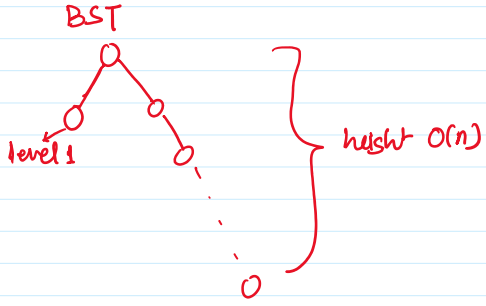
Consider an AVL tree of n nodes

Let the closest leaf node is at level K.

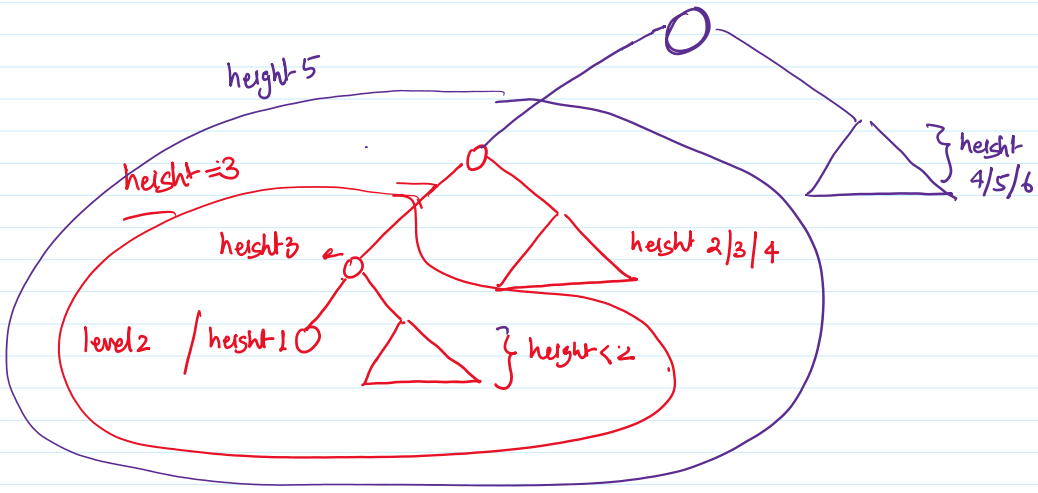
The height of AVL tree is at most $2K-1$.



AVL tree



Proof by image



$K = 2$ $\quad h \leq 3$

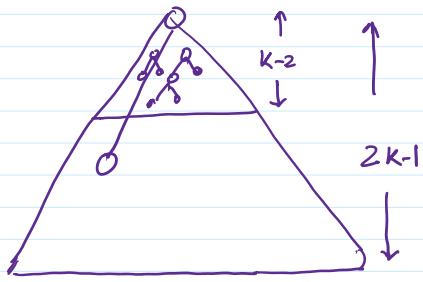
$K = 3$ $\quad h \leq 5$ $\quad \Rightarrow \boxed{h \leq 2K-1}$

$K = 4$ $\quad h \leq 7$

Proposition If the closest leaf is at level K, then all nodes till $1 \ldots (K-2)$ have 2 children.

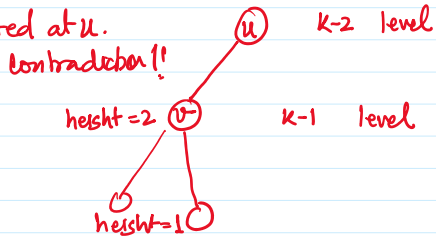
Proof By contradiction

Suppose $\exists$ u at level $(K-2)$ that has only one child v (say, w.l.o.g left child)



v is at level $K-1$, so it can't be a leaf (By assumption)

$\Rightarrow$ Height balance property is violated at u.



Contradiction!!

QED

BST # of nodes can be linear in height of BST.

$$2^{k-1} \leq \#\text{ of nodes in AVL tree} \leq 2^{2k-1} \qquad \because 2k-1 = h$$

$$2^{\left(\frac{h-1}{2}\right)} \leq \#\text{ of nodes in AVL tree} < 2^{h}$$

Search | Insertion | deletion in AVL tree

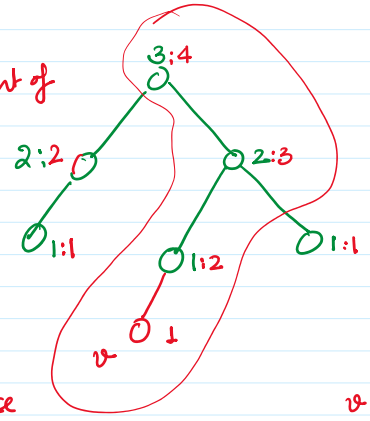
(i) Search - exactly BST

(ii) Insertion in AVL tree :

Inserting a node in AVL tree T changes height of some nodes in T.

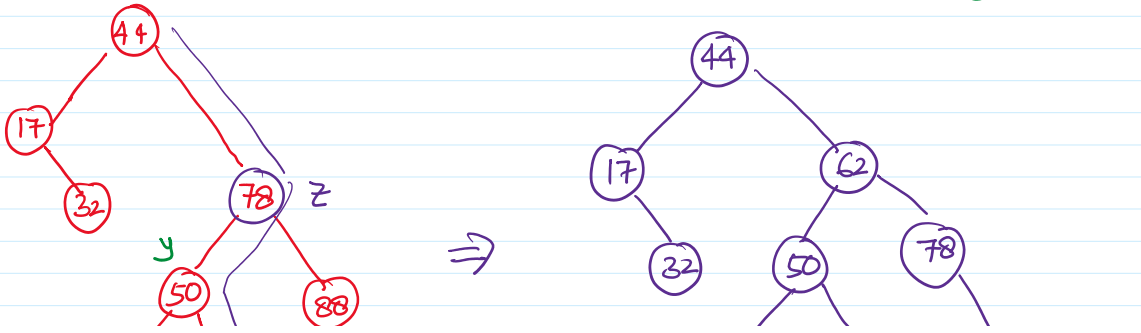
Nodes that are ancestor of v, their height is changed.

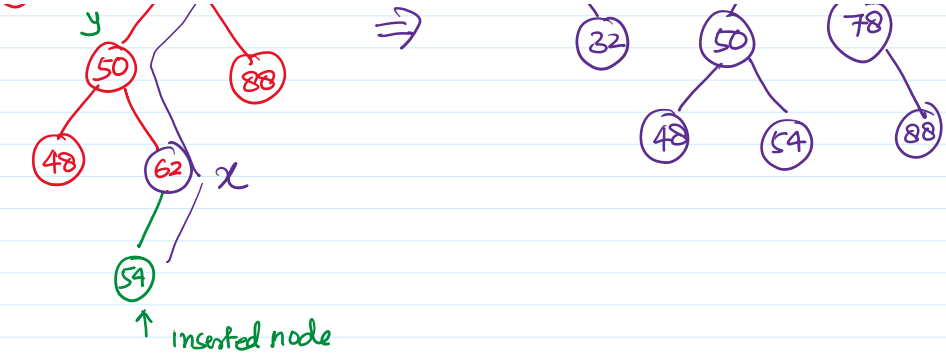
Height balance property can violate at those nodes (ancestor of v)



Traverse up toward root from v, until we find the first node on which height balance property is violated (z)

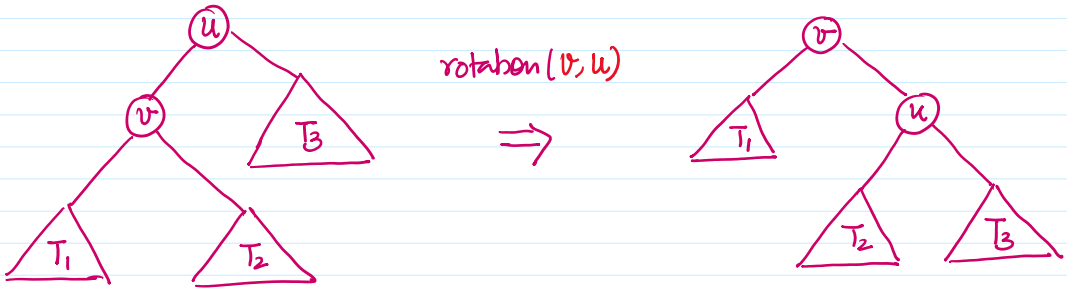
let z has child and grandchild y, x toward newly inserted node)





Rotation : - A process of locally reorganizing BST

- u, v are two nodes s.t u = parent(v)



$key(T_1) < key(v) < key(T_2) < key(u) < key(T_3)$

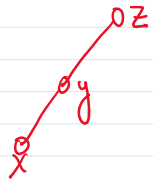
Insertion in AVL tree

z be the 1st node (traversing from inserted node towards root) on which height balance property is violated.

y & x are child and grandchild of z while traversing from inserted node towards root.

4 possible cases – y can be left/right child of z

x " " " y

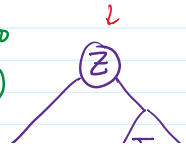


Case (i) y is left child of z and x is left child of y.

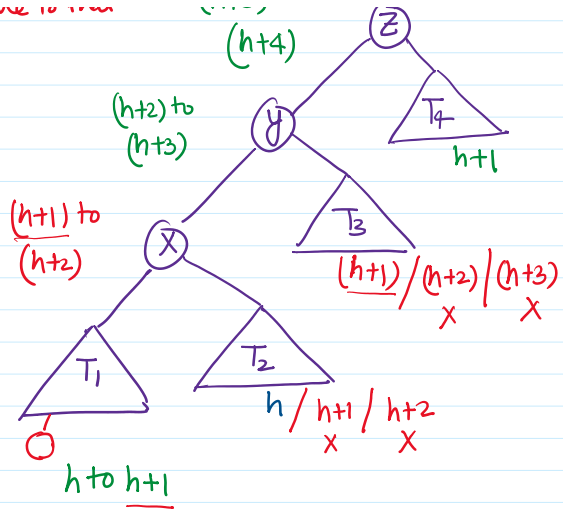
Suppose originally $h_t(T_1) = h$.

Let insertion happen at T_1, and due to that $h_t(T_1)$ increased from h to $h+1$

$(h+3)$ to $(h+4)$



$h_t(T_1)$ increased from h to h+1



Since X remain balance $h_t(T_2)$ could be

- h, (h+1), (h+2)

If $h_t(T_2) = h+2$ then X was originally imbalanced (contradiction)

- If $h_t(T_2) = h+1$, then $h_t(X)$ doesn't increase

⇒ z is balanced (contradiction)

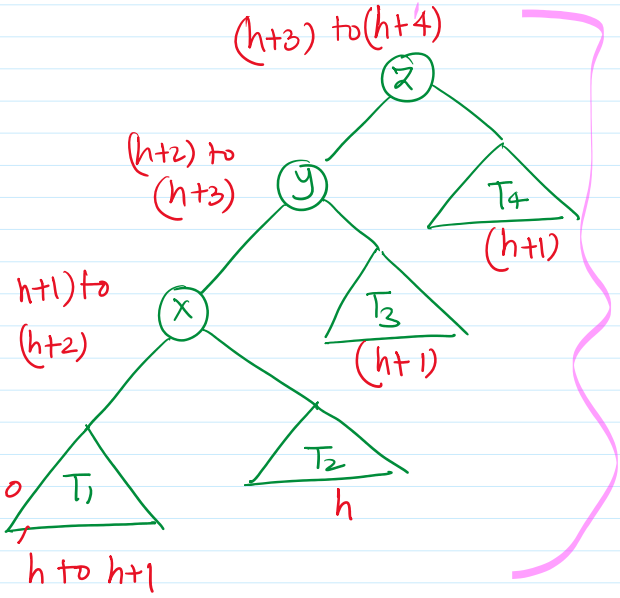
⇒ $h_t(X)$ increased from (h+1) to (h+2)

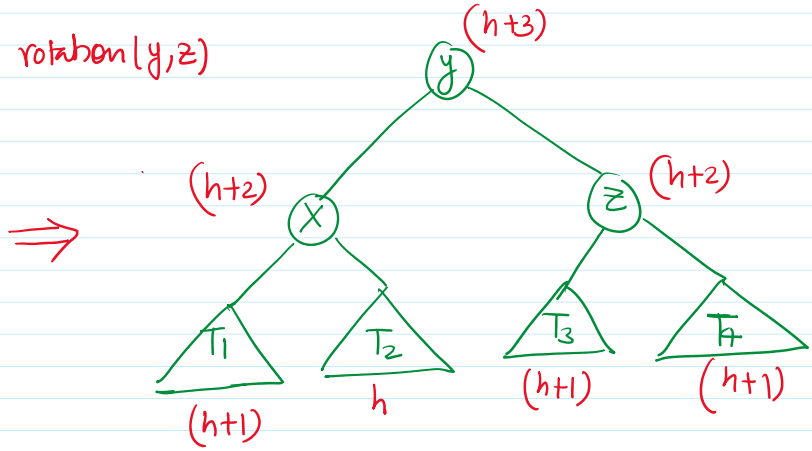
Since y remain balanced $h_t(T_3)$ can be (h+1), (h+2) (h+3)

- If $h_t(T_3) = (h+3)$ then y was originally imbalance (contradiction)
- If $h_t(T_3) = (h+2)$, then $h_t(y)$ doesn't increase ⇒ z is balanced (contradiction)

Since z was originally balanced $h_t(T_4)$ can be (h+1), (h+2), (h+3)

Since z is imbalanced now so $h_t(T_4) = h+1$





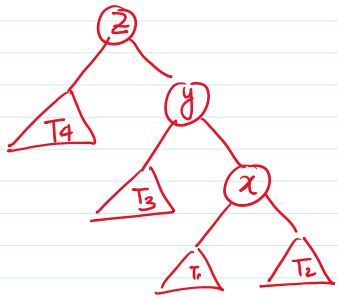
Complexity O(1)

Original height of subtree rooted z was (h+3)

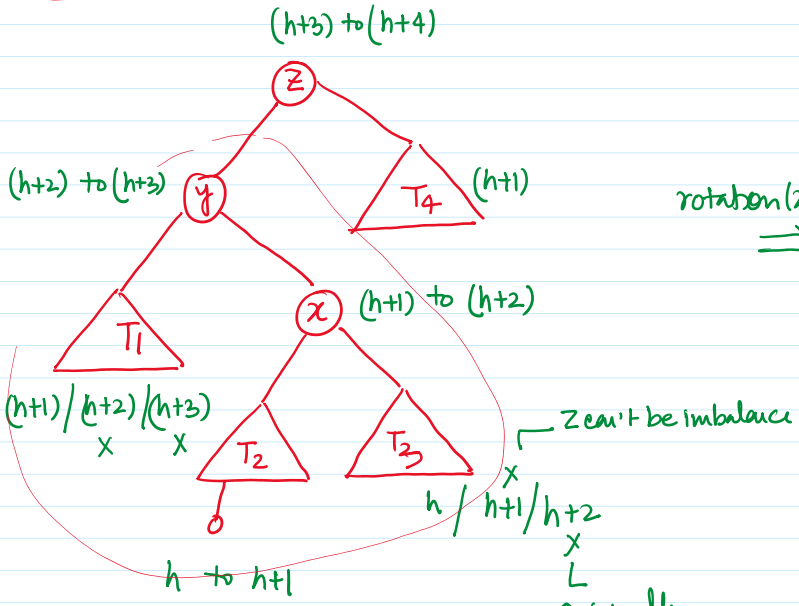
Updated height of the subtree is (h+3).

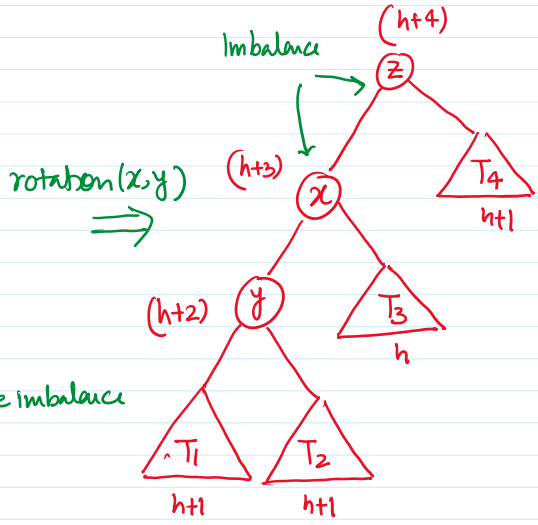
⇒ No need to proceed towards root.

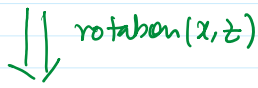
Case(ii) Analogous to case (i)

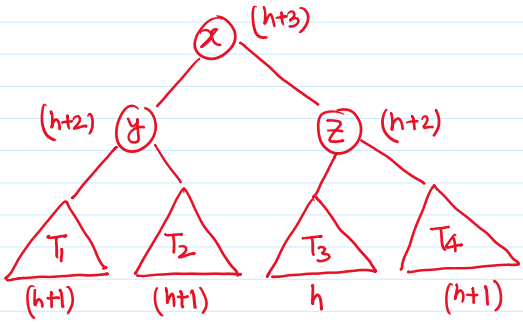


Case(iii)





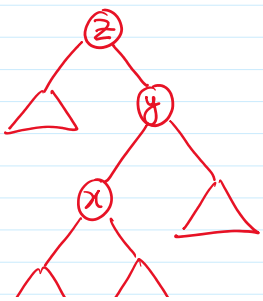




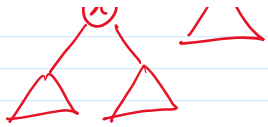
Final tree has the same height as original subtree rooted z.

So we don't need to go up further.

Case (iv)



Identical to case (iii)

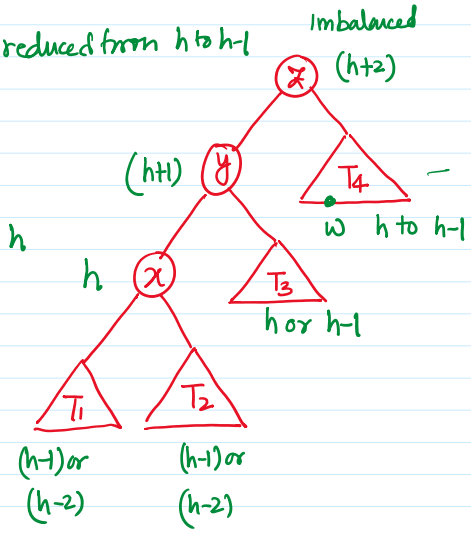


Time complexity – inserting like BST $O(\log n)$
– identifying x, y, z $O(\log n)$
– single/double rotation $O(1)$
} $O(\log n)$

Deletion in BST

- let w be the deleted node
- let z be the first imbalanced node in the path from w to root.
- y be the child of z with larger height
- x " " y " " "

Case (1) deletion happens at T_4, $ht(T_4)$ reduced from h to h-1

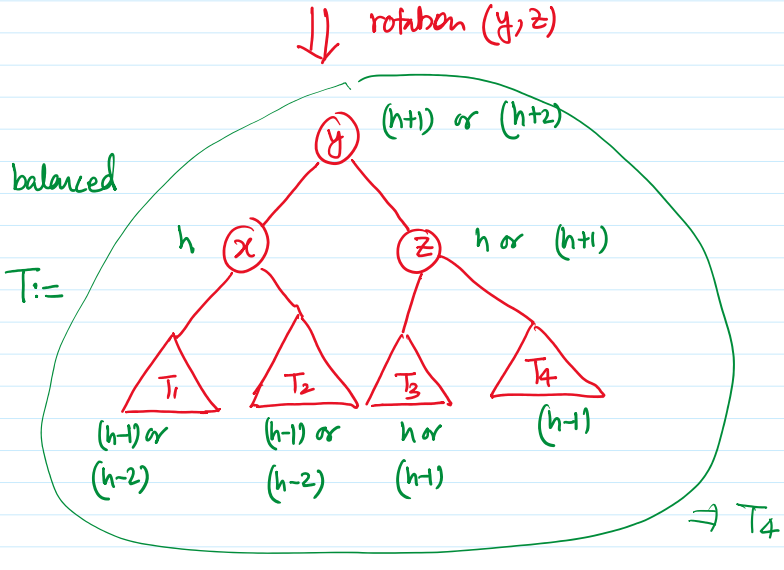


z is imbalanced $\Rightarrow ht(y) = h+1$

x has larger height than T_3 so $ht(x) = h$

Since $ht(x) = h$, $ht(T_1)$, $ht(T_2)$ can be either (h-1) or (h-2)

Both T_1 & T_2 can't have height (h-2).



After rotation ht(T) can be reduced. So need to continue up further

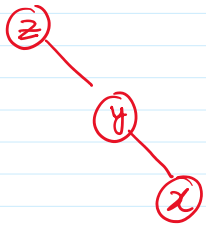
After rotation ht(T) can be reduced. So need to continue up further

Repeat the algo considering T as new T4.

In the worst case repeation need to be done O(log n) time.

height of tree

Case (ii)



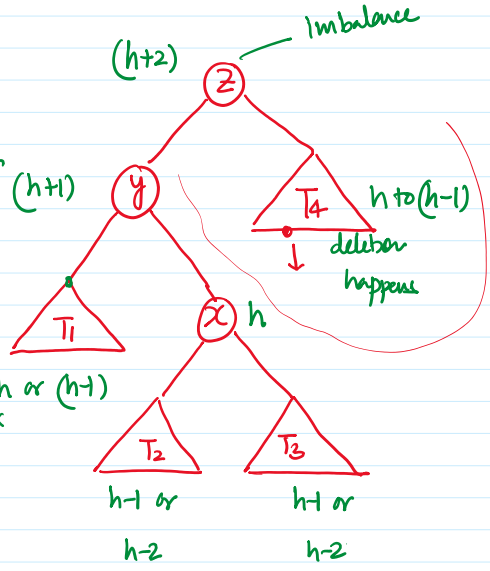
analogous to case (1)

Case (iii)

Ac before $h_t(y) = h+1$

Since y balanced, $h_t(x) = h$ { child with larger height }

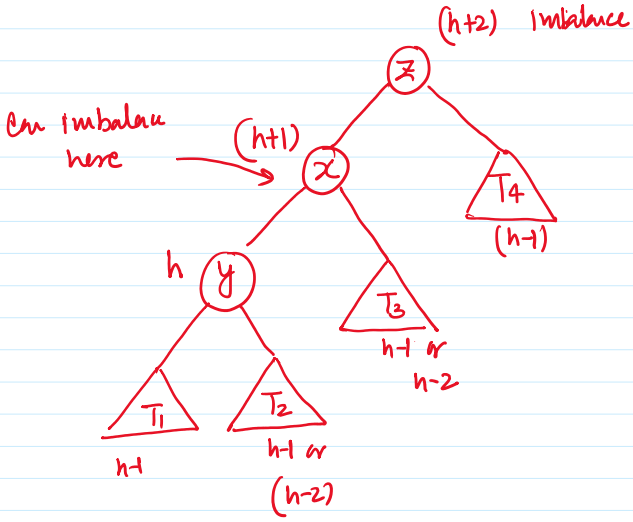
$h_t(T_1)$ is h or h-1



if $h_t(T_1) = h$ then we would have picked root T_1 as x.

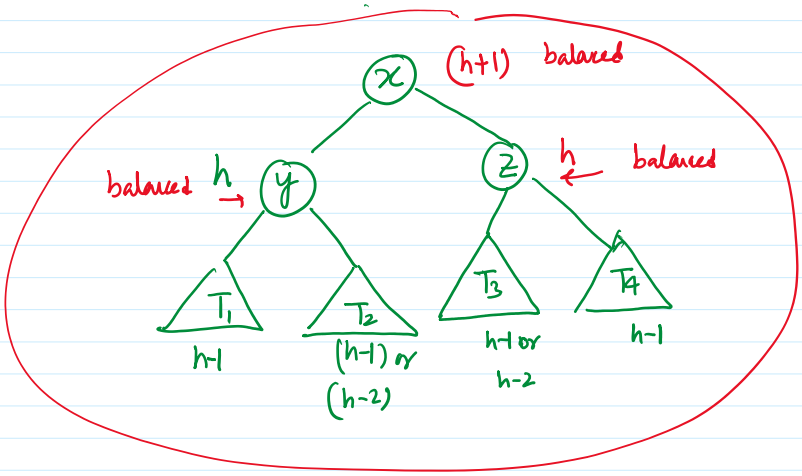
Both $h_t(T_2)$, $h_t(T_3)$ can't be (h-2).

rotation(x, y)



rotation (x, z)

T :=



Height of T reduces from (h+2) to (h+1)

repeat considering T as new T4

case iii

analogous to previous case

Time complexity - Deletion using BST O(log n)

- In the worst we need to go up to the root - O(log n)

  L in each we need to perform O(1)] rotation