## SECTION-1:

### ⑤ Correctness:

Proof by induction.

#### Base-case:

If root == NULL, the function immediately returns i.e. no key is printed.

#### Inductive-hypothesis:

Assume that for any subtree with no. of nodes $< n$, `rb_enumerate(root, a, b)` correctly prints all keys in $[a, b]$ in sorted order.

#### Inductive-step:

Consider a tree rooted at `root` with no. of nodes `n`.

**Case-1:** root → data > b

All keys in the right subtree are greater than root → data i.e. $> b$.

⇒ The entire right subtree is safely ignored and the algo. recurses only on the

left subtree, which gives the correct output according to the induction hypothesis.

Case-2: root → data < a

Similar to case-1, here the algo safely ignores the left subtree and recurses only on the right subtree, which gives the correct output according to the induction hypothesis.

Case-3: a ≤ root → data ≤ b

- The left subtree contains the keys in range [a, root → data). The recursive call rb_enumerate(root → left, a, b) handles that.
- root → data is printed as it lies in the range
- The right subtree contains keys in the range (root → data, b]. The recursive

call rb_enumerate (root → right, a, b)
handles that.

Hence, by strong induction, the algorith correctly prints all the keys in the range [a, b] in sorted order.

Time-complexity:

- Nodes not in range but are on the search paths to the first and last in-range node are visited.

$$\Rightarrow \Theta(h) = \Theta(\log n) \quad (\because \text{RBT})$$

- For every in-range node, the algorithm takes $\Theta(1)$ time (for printf and pointer movement)

$$\Rightarrow m \cdot \Theta(1) = \Theta(m).$$

$$\therefore \boxed{T(n) = \Theta(m + \log n)}$$

# SECTION-2:

① Consider the original relaxed RBT T with a red root.

⇒ The children of the root are black (∵ T is an RBT)

Let, black depth of some node N is d.

Now, the root of T is colored black.

(a) Every node is either red/black. ⇒ ✓

(b) Changing the color of the root doesn't affect the color of the leaves

⇒ Leaves are still black.

(c) New double-red problem can only occur if a new red node is introduced. No such operation is done here

⇒ No double-red problem

(d) Black depth of node N increases to d+1, irrespective of the node N.

⇒ Every nodes black depth increases by 1.

⇒ (d) is also satisfied.

∴ The resulting tree T is also a RBT.

---

② Let, a RBT T have a node x.

Consider sub-tree rooted at x, which is a relaxed RBT. Let the subtree be $T_x$.

All the leaf nodes of $T_x$ have the same black-depth i.e. the no. of black nodes between the root x and any leaf is the same, say d.

Let the distance between x and a leaf node L be p.

The shortest possible value of p is d, as there need to be 'd' black nodes in the path from x to L.

⇒ $p \geq d$ — (1)

The largest possible value of p is 2d, as along with the necessary 'd' black nodes, there can be atmost 'd' red nodes without causing double-red problem.

$$\Rightarrow p \leq 2d \quad - (1)$$

$P_L$ := Longest path length between x and a leaf

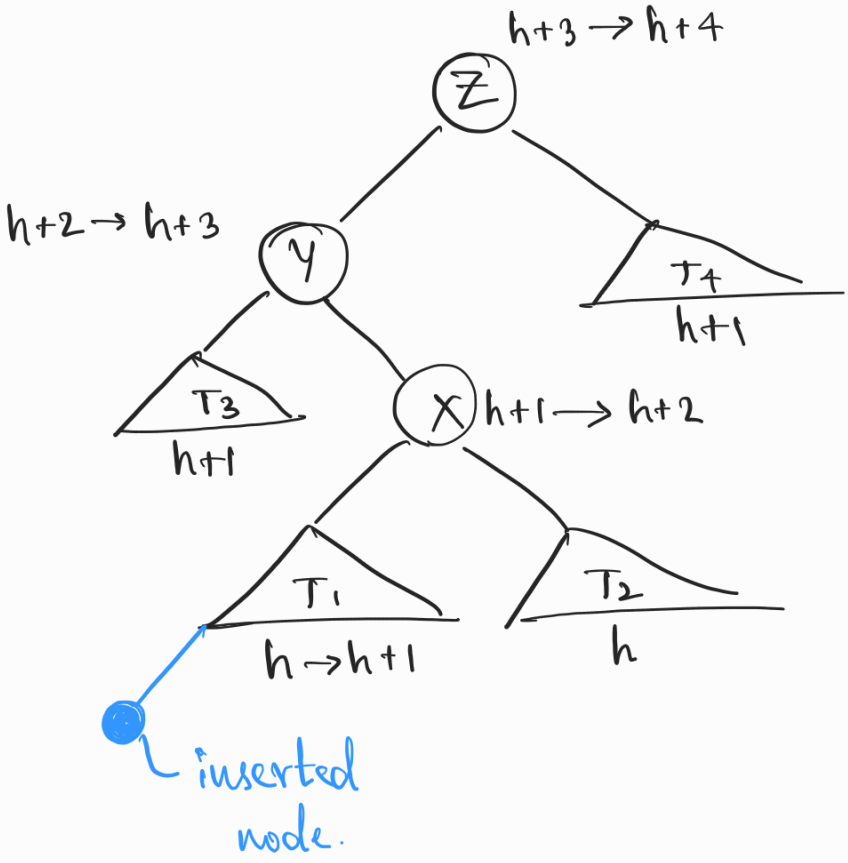
$P_s$ := Shorstest path length between x and a leaf

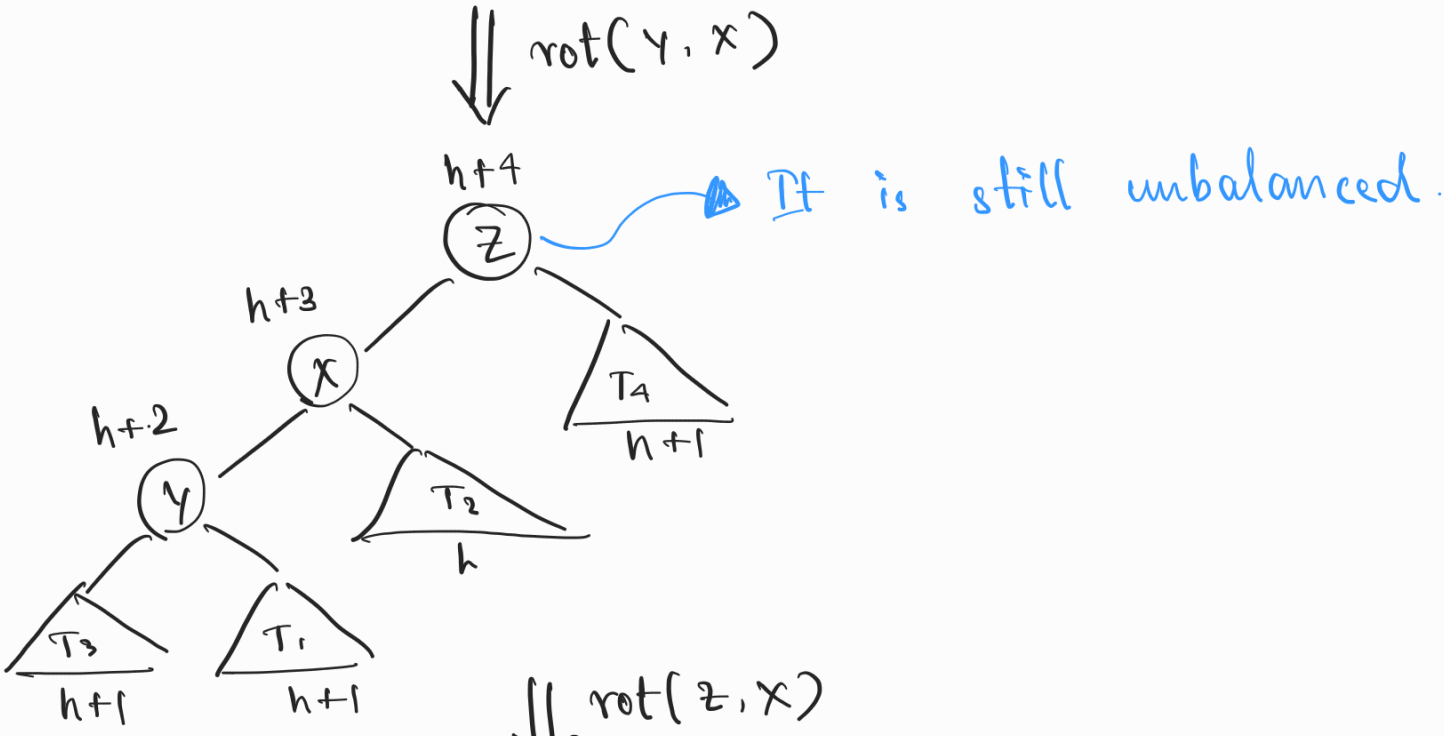
$$P_L \leq 2d$$

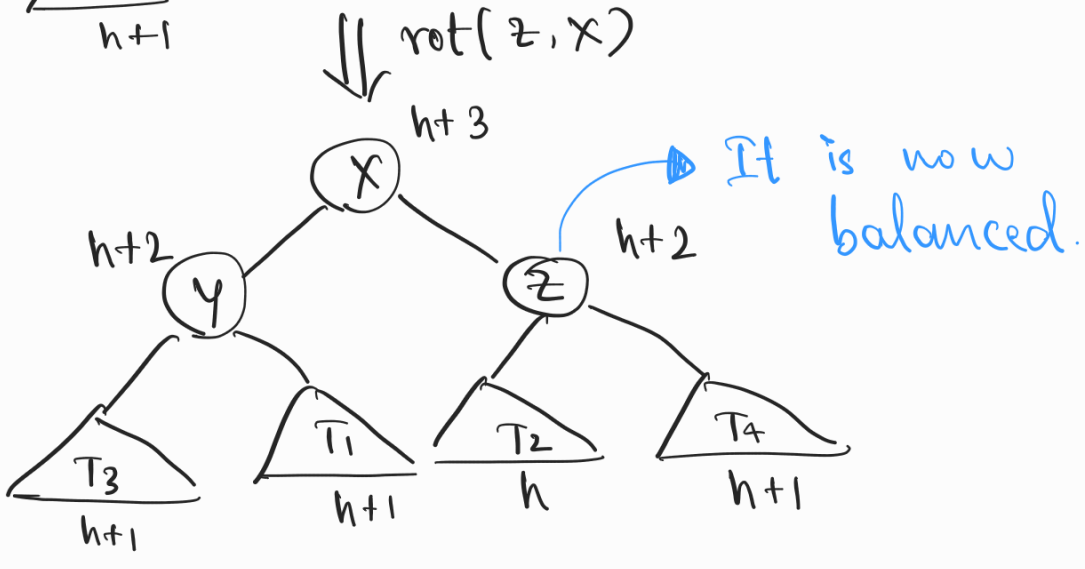
$$P_s \geq d \Rightarrow \frac{1}{P_s} \leq \frac{1}{d}.$$

$$\Rightarrow \boxed{\frac{P_L}{P_s} \leq 2.}$$

(3)


h+3 → h+4
Z
h+2 → h+3
Y
T4
h+1
T3
h+1
X
h+1 → h+2
T1
h → h+1
T2
h
inserted node.


rot(Y, X)
h+4
Z
It is still unbalanced.
h+3
X
T4
h+1
h+2
Y
T2
h
T3
h+1
T1
h+1


rot(Z, X)
h+3
X
It is now balanced.
h+2
Y
Z
h+2
T3
h+1
T1
h+1
T2
h
T4
h+1

The node (z), which was unbalanced due to insertion could not be balanced by a single rotation.

⇒ Double rotation was performed to achieve height balance.