

Data Structures and Algorithms

(ESO207)

Lecture 19

Analysis of

- Red Black trees
- Nearly Balanced BST

A **Red Black** Tree is height balanced

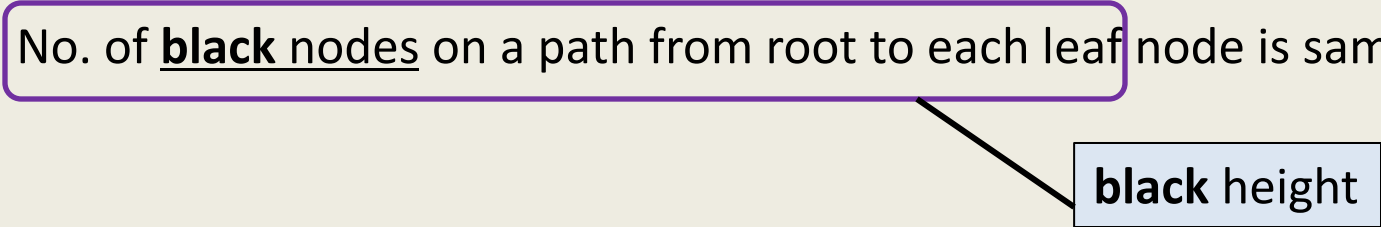
A **detailed proof** from **scratch**

Red Black Tree

Red Black tree:

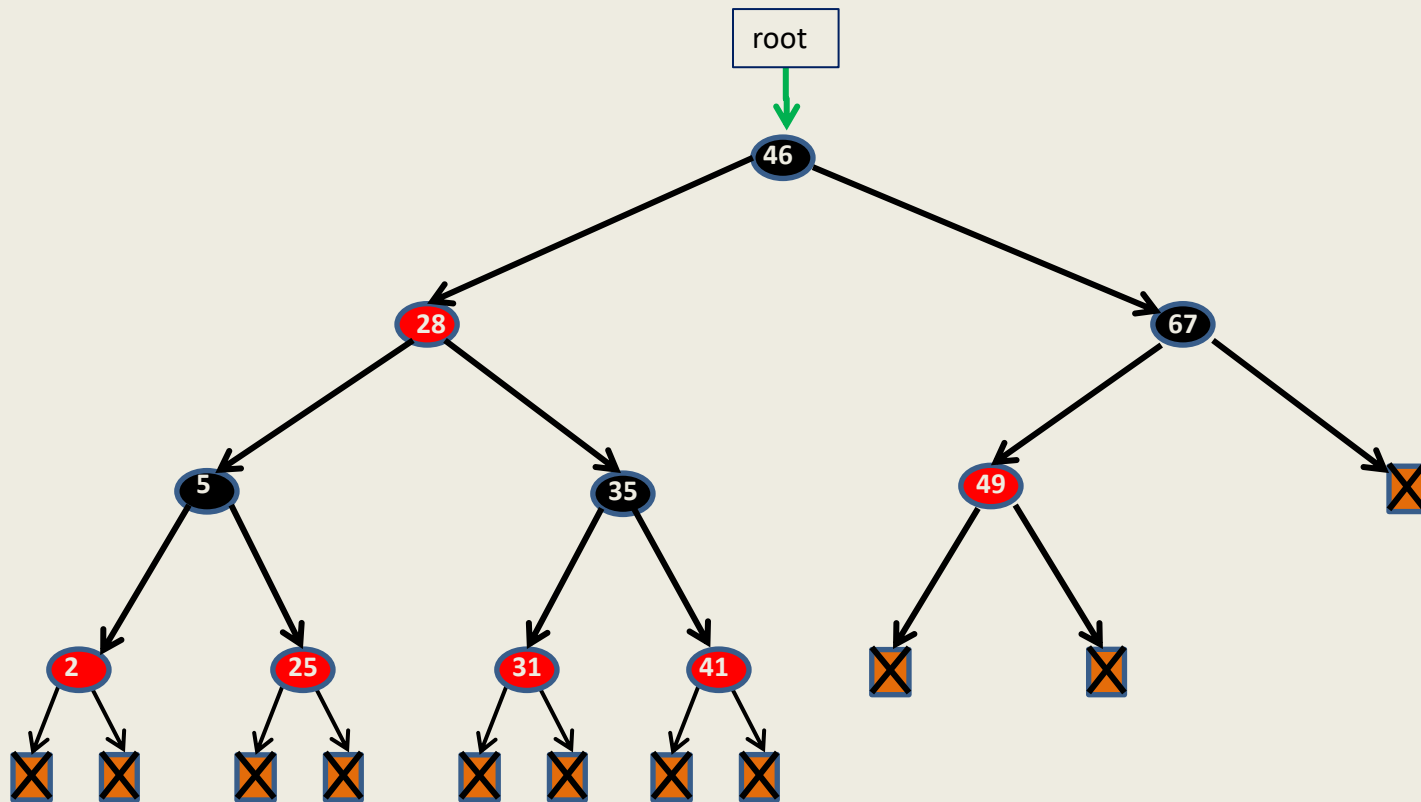
a **full** binary search tree with each leaf as a **null** node and satisfying the following properties.

- Each node is colored **red** or **black**.
- Each leaf is colored **black** and so is the root.
- Every **red** node will have both its children **black**.
- No. of **black nodes** on a path from root to each leaf node is same.



black height

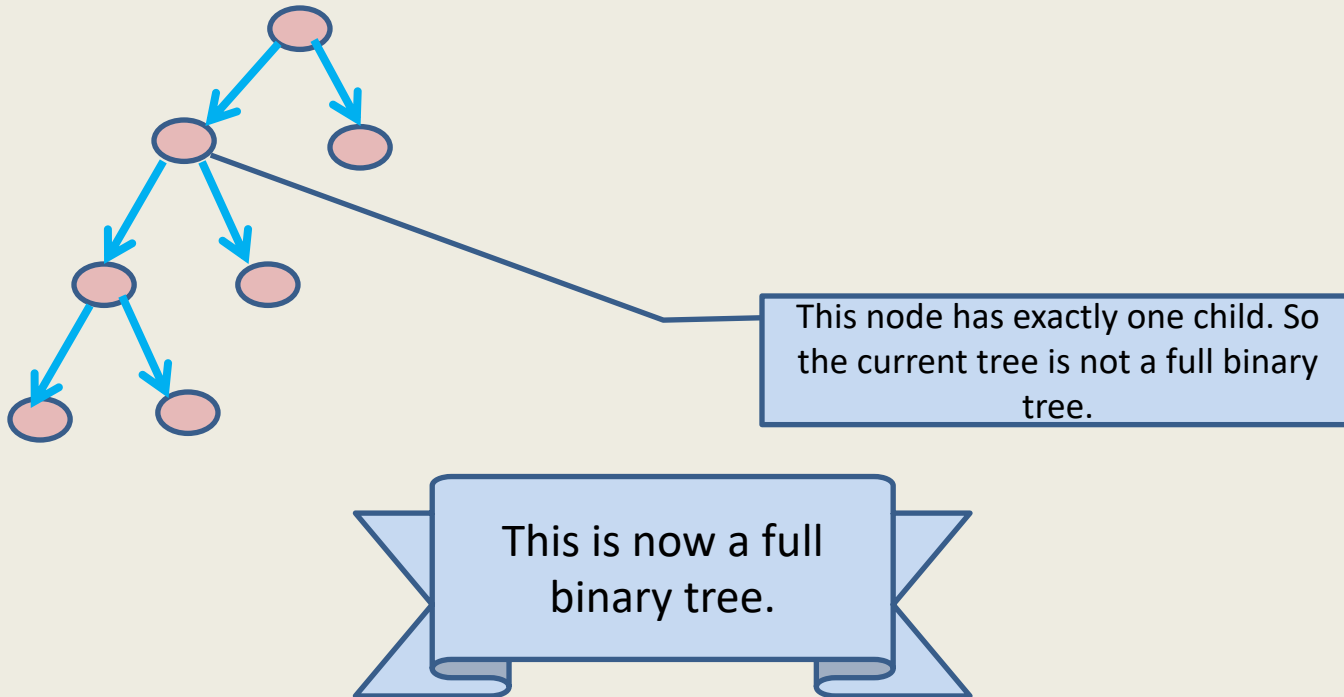
A red-black tree



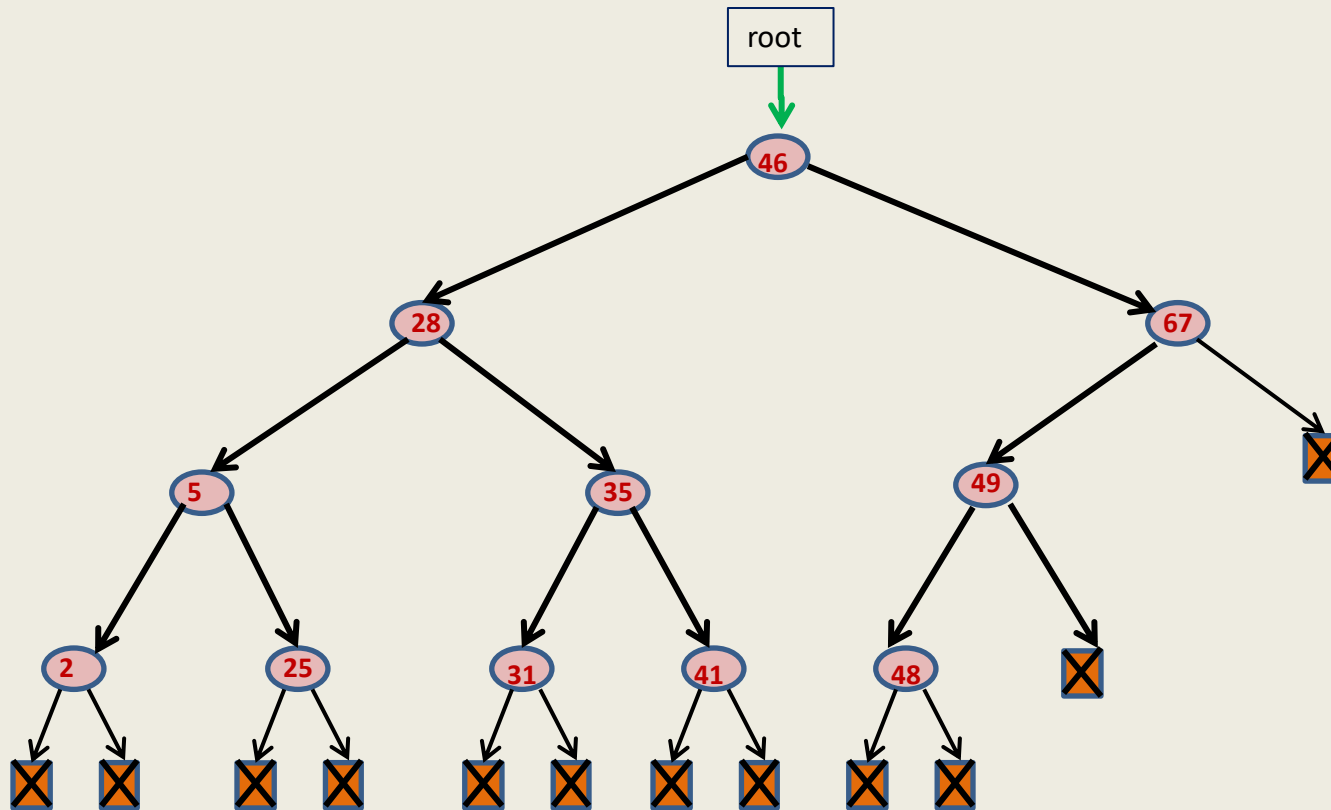
Terminologies

Full binary tree:

A binary tree where every internal node has exactly two children.

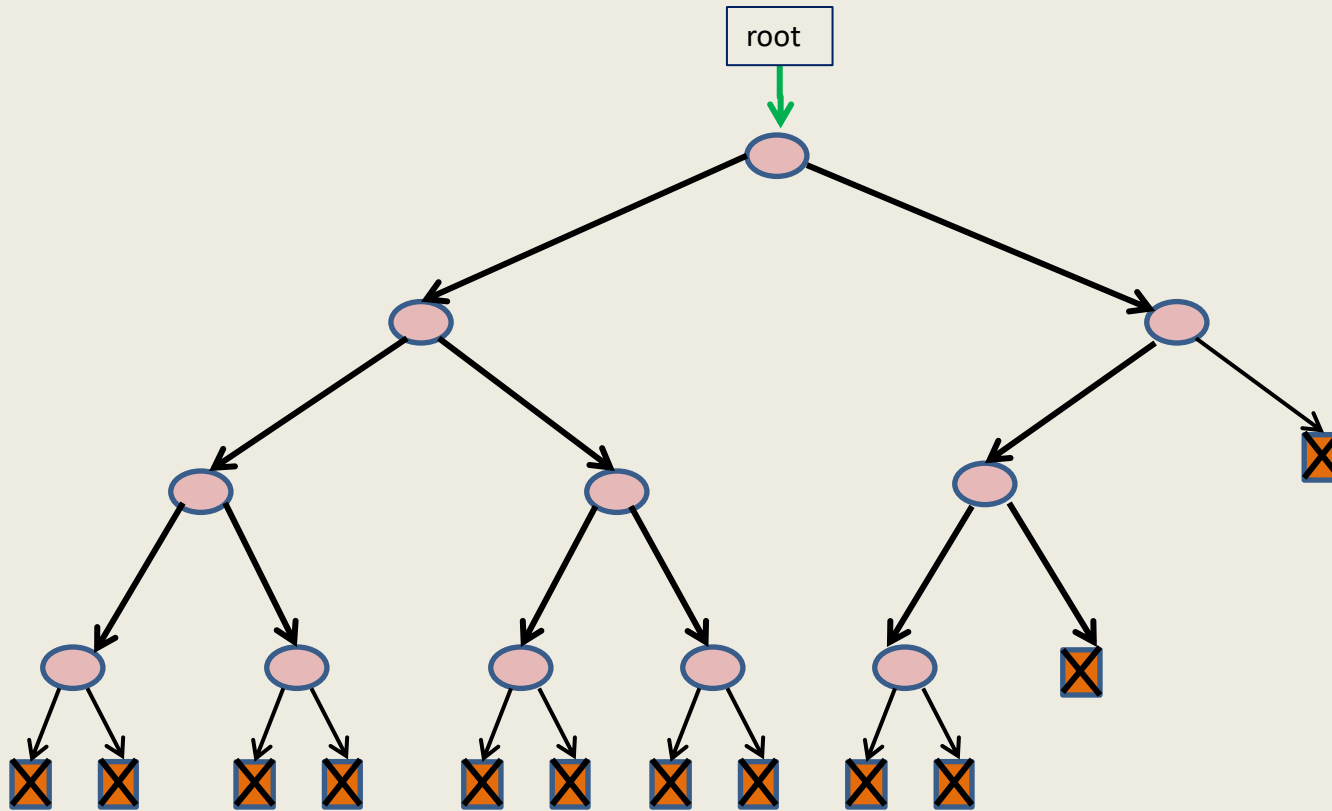


Red-black tree: as a Full Binary Tree



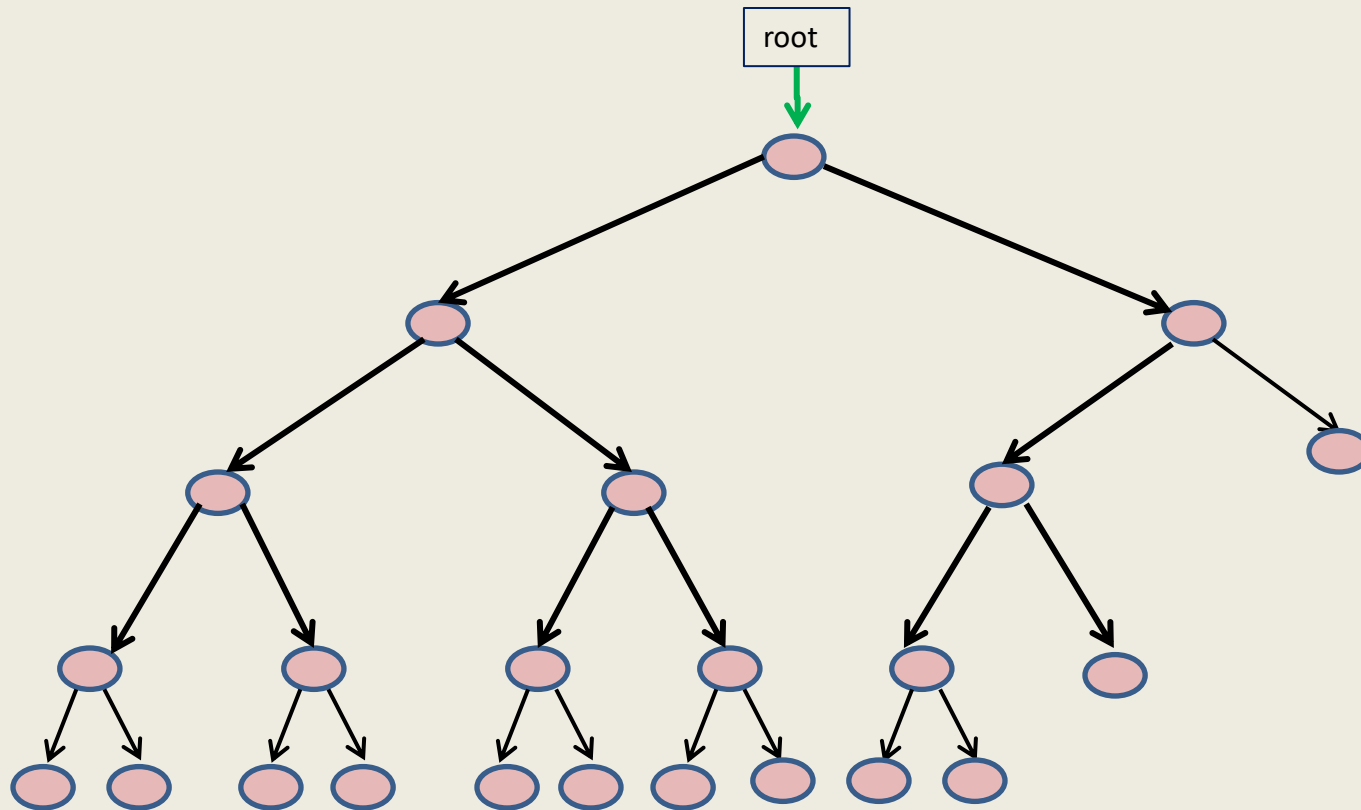
Ignore the values

Red-black tree: as a Full Binary Tree



Ignore the distinction
between internal nodes
and leaf nodes

Red-black tree: as a Full Binary Tree



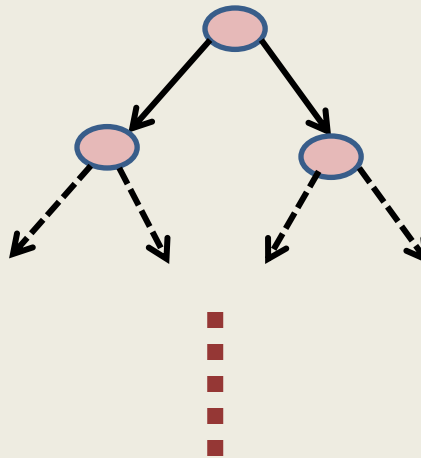
Properties of a **Red**-Black Tree viewed as a full binary tree

Relationship between

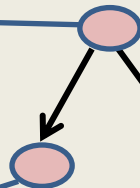
Number of **leaf nodes** and

Number of **internal nodes**

A full binary tree



This node must have a left child since the tree is a full binary tree

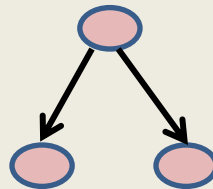
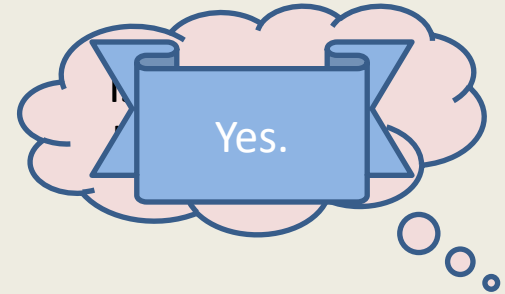
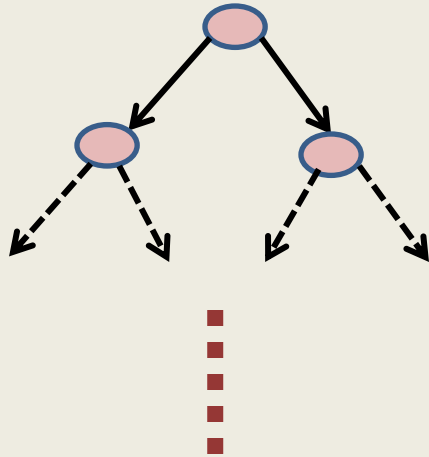


This node must be a leaf node. Give reason.

Otherwise this node won't be the deepest node.

Any deepest node

A full binary tree



What happened to the number of internal nodes ?

Reduced by one

What happened to the number of leaf nodes ?

Reduced by one

A full binary tree

Analyze the process:

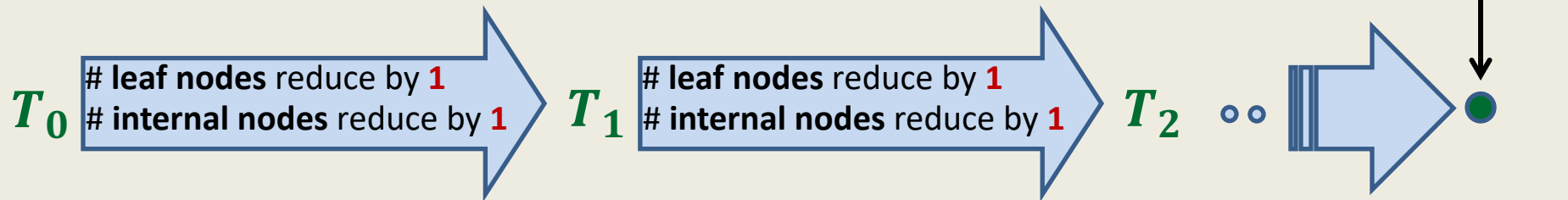
Repeat

{ Delete the deepest node and its sibling }

until only **root** remains

Let T_0 be the full binary tree before the process starts.

Let T_1, T_2, \dots be the full binary trees after 1st, 2nd, ... iterations of the process.



Question: What might be the relation between leaf nodes and internal nodes in T_0 ?

Answer: No. of leaf nodes in T_0 = No. of internal nodes in T_0 + 1.

A full binary tree

Question: If i is the number of internal nodes in a full binary tree T , what is the size (number of nodes) of the tree ?

Answer:

$$2i + 1$$

Question: What is the size of a **Red Black** tree storing n keys ?

Answer:

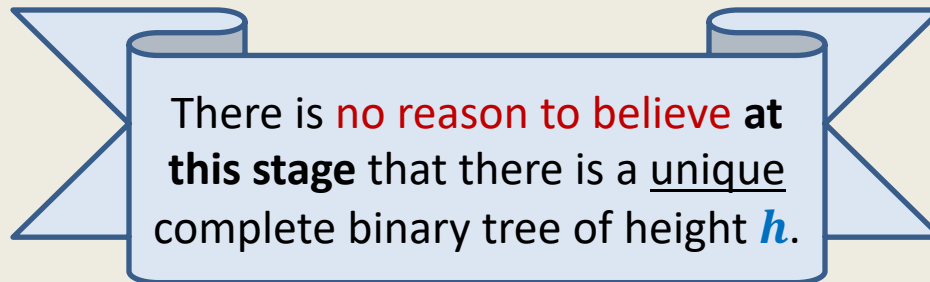
$$2n + 1$$

A complete binary tree of height h and its Properties

A complete binary tree of height h

Definition:

A full binary tree of height h is said to be a **complete** binary tree of height h if every leaf node is at depth h .



Question: How will any complete binary tree of height h look like ?

A complete binary tree of height h

Definition:

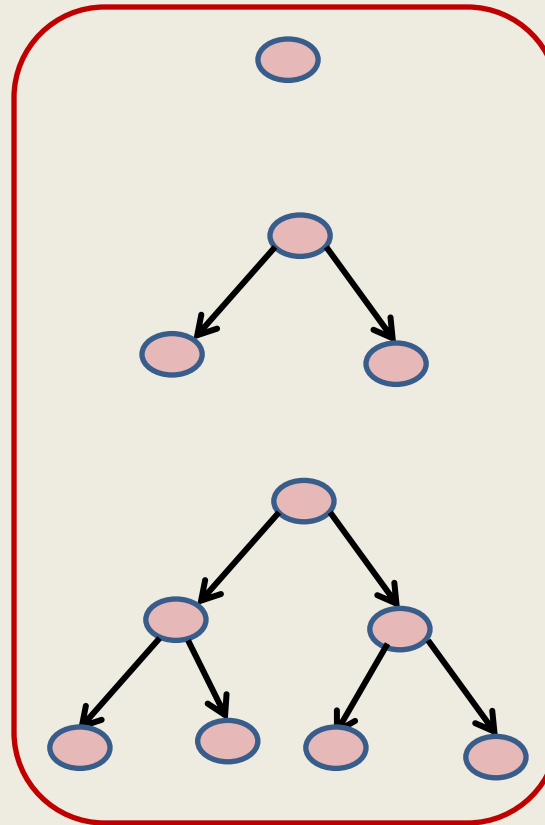
A full binary tree of height h is said to be a complete binary tree of height h if every leaf node is at depth h .

A complete binary tree of height h

Complete binary tree of height 1 ?

Complete binary tree of height 2 ?

Complete binary tree of height 3 ?



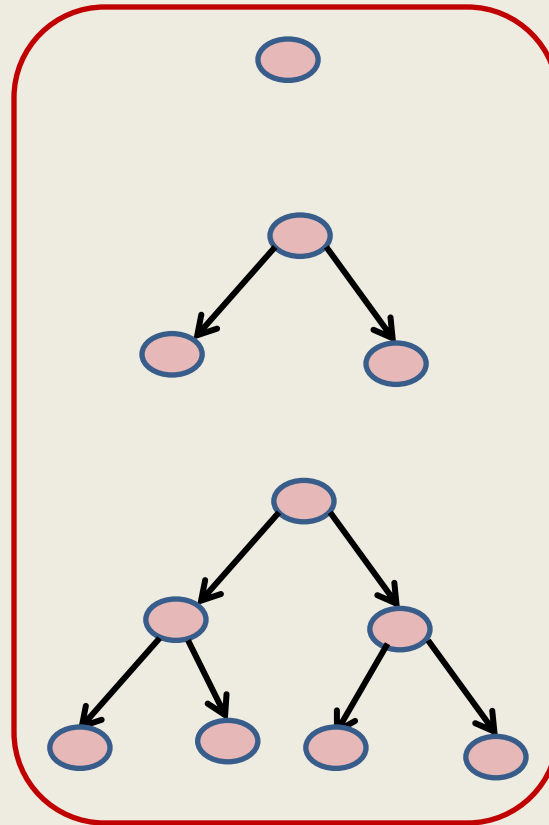
Try to generalize the type of tree shown here to tree of height h .

A **complete** binary tree of height h

Complete binary tree of height **1** ?

Complete binary tree of height **2** ?

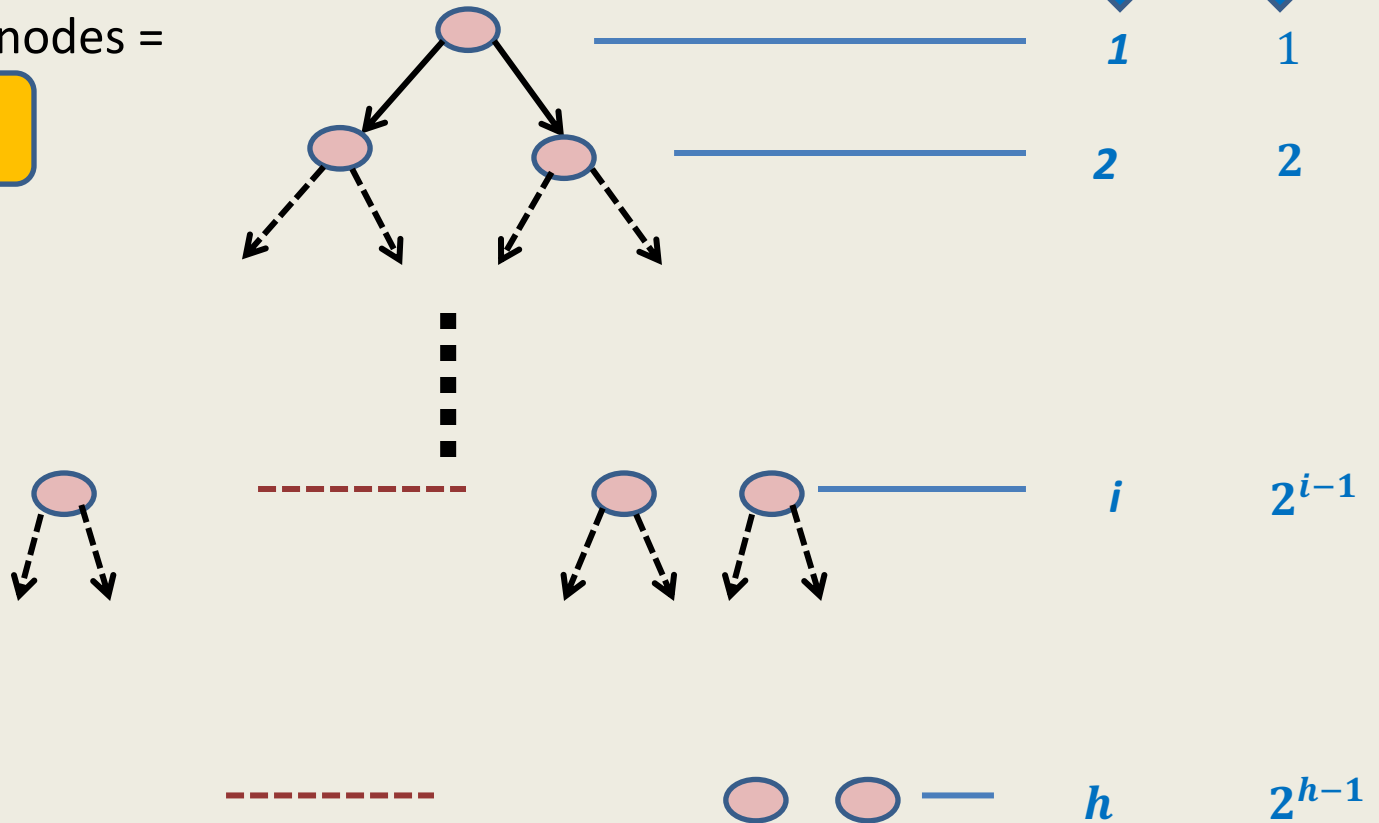
Complete binary tree of height **3** ?



A complete binary tree of height h

Total number of nodes =

$$2^h - 1$$



Certainly this tree is a complete binary tree of height h .
We shall now show that this is **the only possible** complete binary tree of height h .

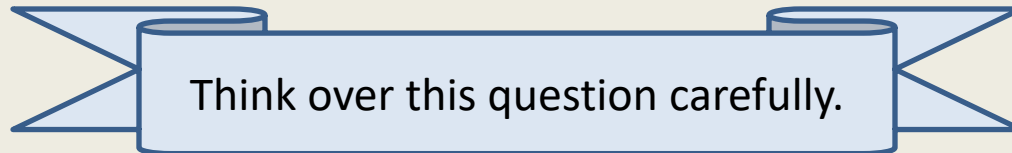
Uniqueness of a complete binary tree of height h

Let T^* be the complete binary tree of height h shown in previous slide.

Notice that this is **densest** possible tree of height h .

Let T be any other complete binary tree of height h different from T^* .

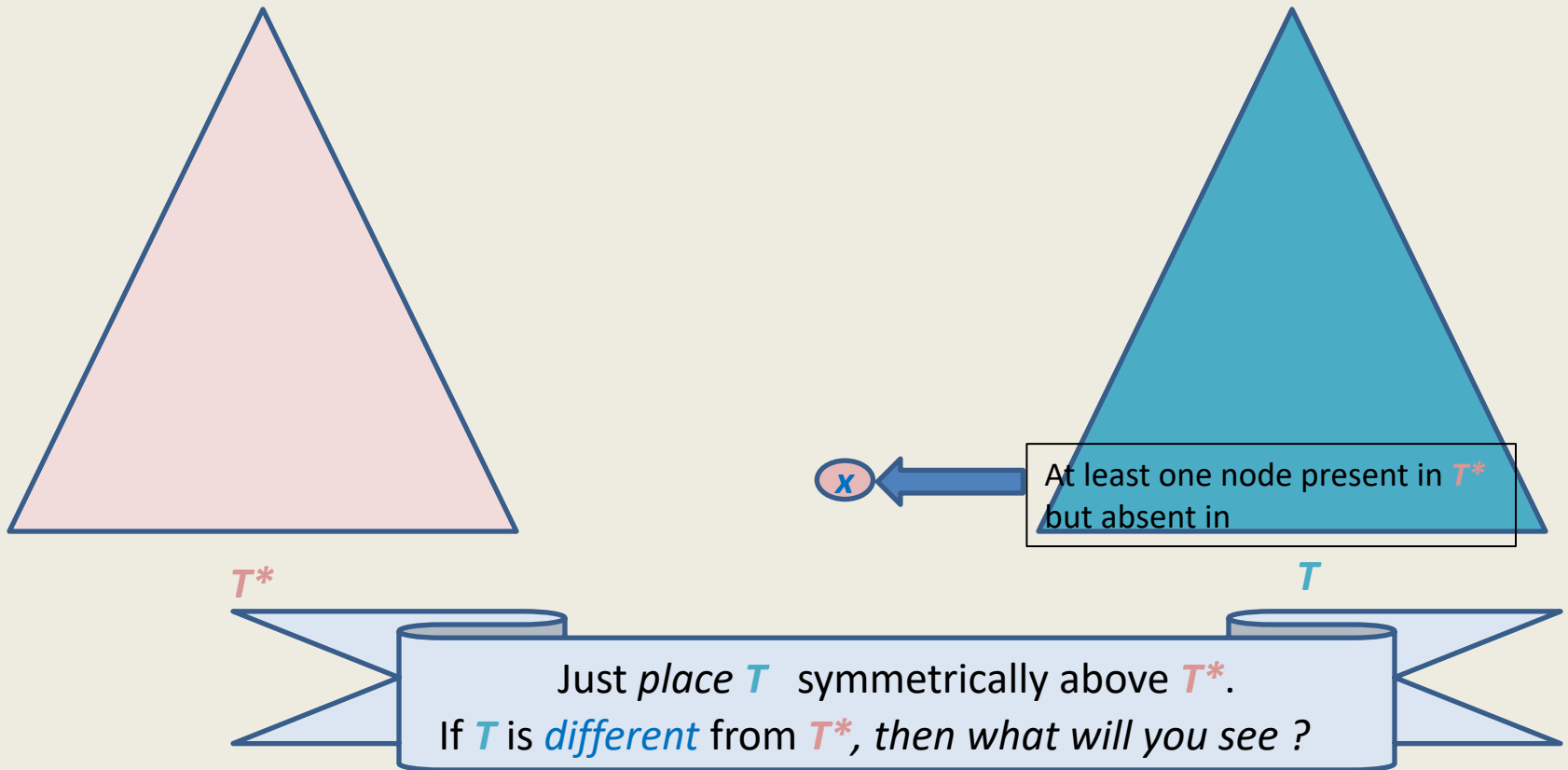
Question: How to show that T can not exist ?



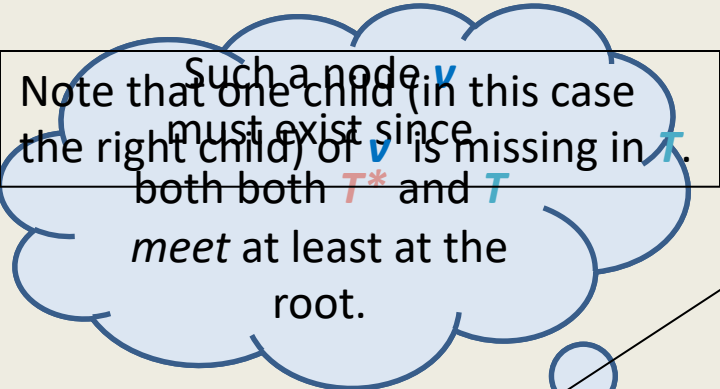
Watch the following slide carefully.

Uniqueness

of a **complete** binary tree of height h



Uniqueness



Depth of v must be less than h since v is ancestor of x .

Any node present in T^* but absent in T

Since **T** is a full binary tree and **right child** of **v** is missing,

→ v can not be an internal node in T .

→ **v** must be leaf node.

Hence v is a leaf node of T at depth $< h$

Hence

T is not a complete binary tree of height h

Hence there is no **complete** binary tree of height h different from T^* .

➔ There exists a unique **complete** binary tree of height h .

Theorem:

A complete binary tree of height h has exactly $2^h - 1$ nodes.

A **Red Black** Tree is height balanced

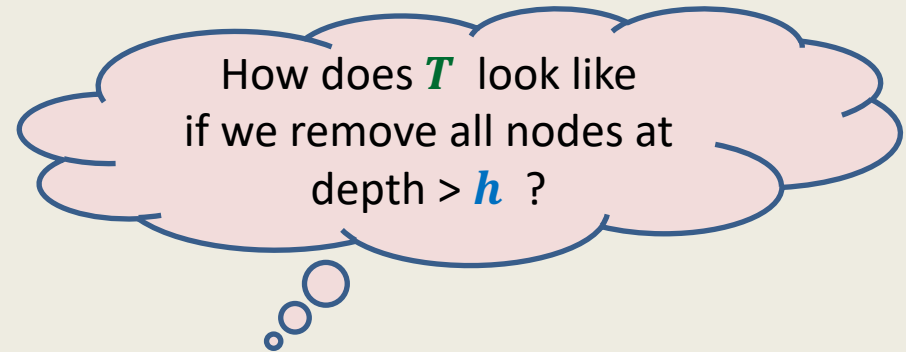
The final proof

T : a red black tree storing n keys.

Total number of nodes = $2n + 1$

h : the black height

Every leaf node is at depth $\geq h$



a complete binary tree of height h

Hence $2n + 1 \geq 2^h - 1$

$$\rightarrow 2^h \leq 2n + 2$$

$$\rightarrow h \leq 1 + \log_2(n + 1)$$

So Height of $T \leq 2h - 1 \leq 2 \log_2(n + 1) + 1$

Analysis

NEARLY BALANCED BST

Nearly balanced Binary Search Tree

Terminology:

size of a binary tree is the number of nodes present in it.

Definition: A binary search tree **T** is said to be nearly balanced at node **v**, if

$$\text{size}(\text{left}(\mathbf{v})) \leq \frac{3}{4} \text{size}(\mathbf{v})$$

and

$$\text{size}(\text{right}(\mathbf{v})) \leq \frac{3}{4} \text{size}(\mathbf{v})$$

Definition: A binary search tree **T** is said to be **nearly balanced** if it is nearly balanced at each node.

Theorem: Height of a **nearly balanced** BST on n nodes is $O(\log_{4/3} n)$

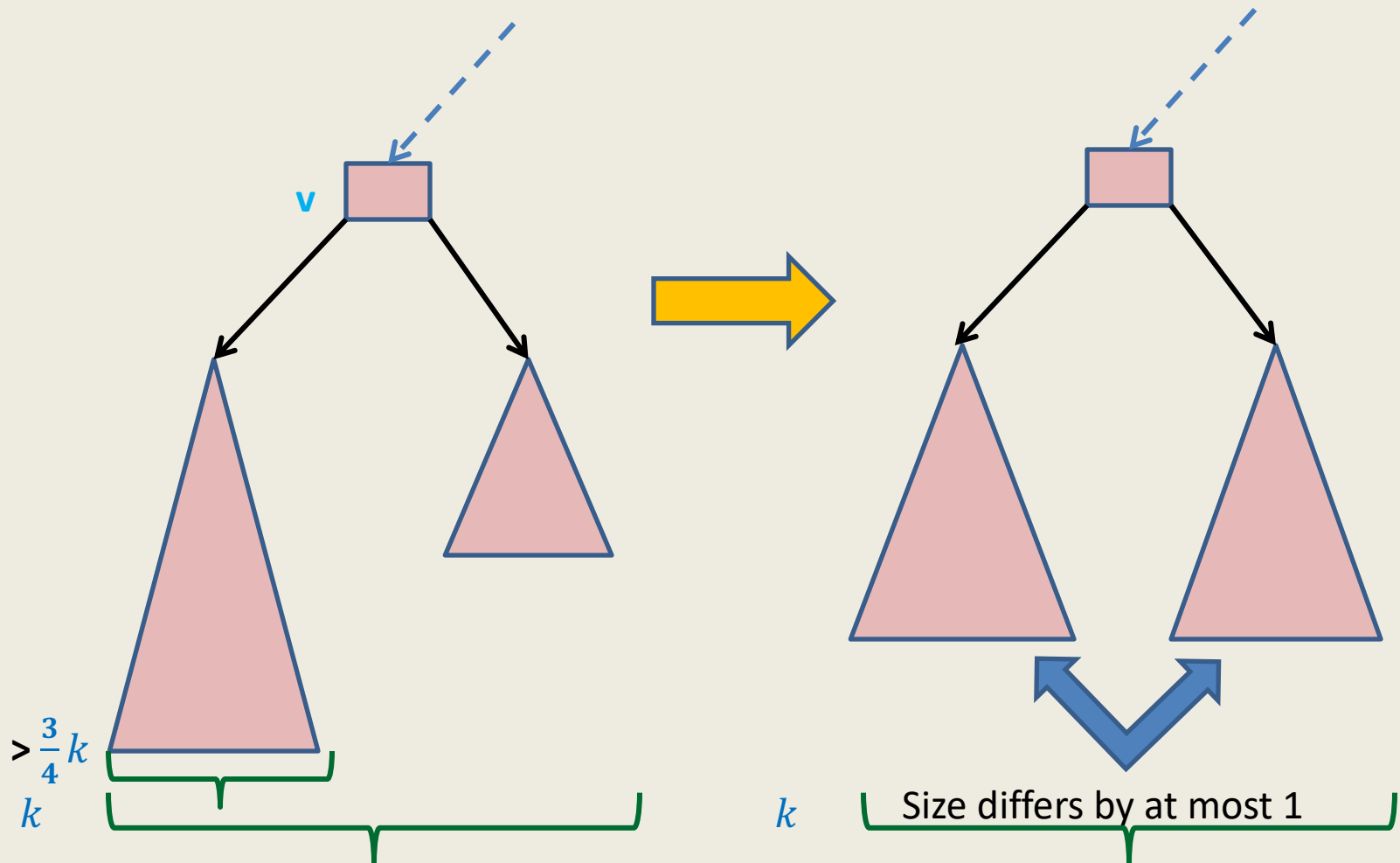
Nearly balanced Binary Search Tree

Maintaining under Insertion

Each node v in T maintains additional field $\text{size}(v)$ which is the number of nodes in the $\text{subtree}(v)$.

- Keep $\text{Search}(T, x)$ operation unchanged.
- Modify $\text{Insert}(T, x)$ operation as follows:
 - Carry out normal insert and update the size fields of nodes traversed.
 - If BST T ceases to be **nearly imbalanced** at any node v , transform $\text{subtree}(v)$ into **perfectly balanced** BST.

“Perfectly Balancing” subtree at a node v



Nearly balanced Binary Search Tree

Observation :

It takes $O(k)$ time to transform an imbalanced tree of size k into a perfectly balanced BST. (It was given as a Homework.)

Observation: Worst case search time in nearly balanced BST is $O(\log n)$

Theorem:

For any arbitrary sequence of n operations, total time will be $O(n \log n)$.

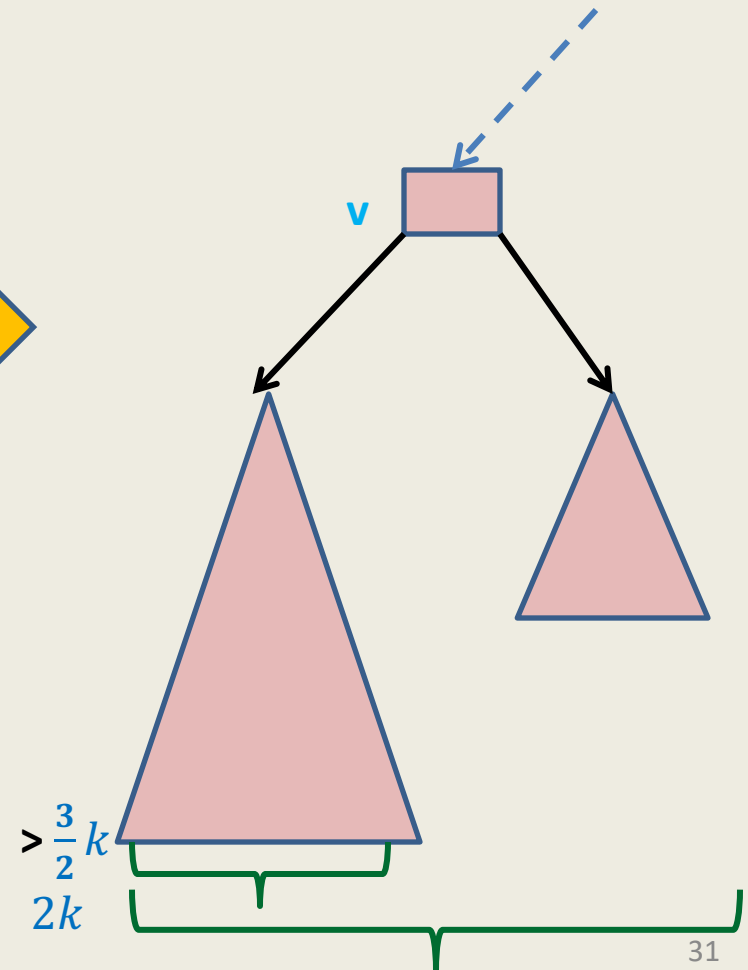
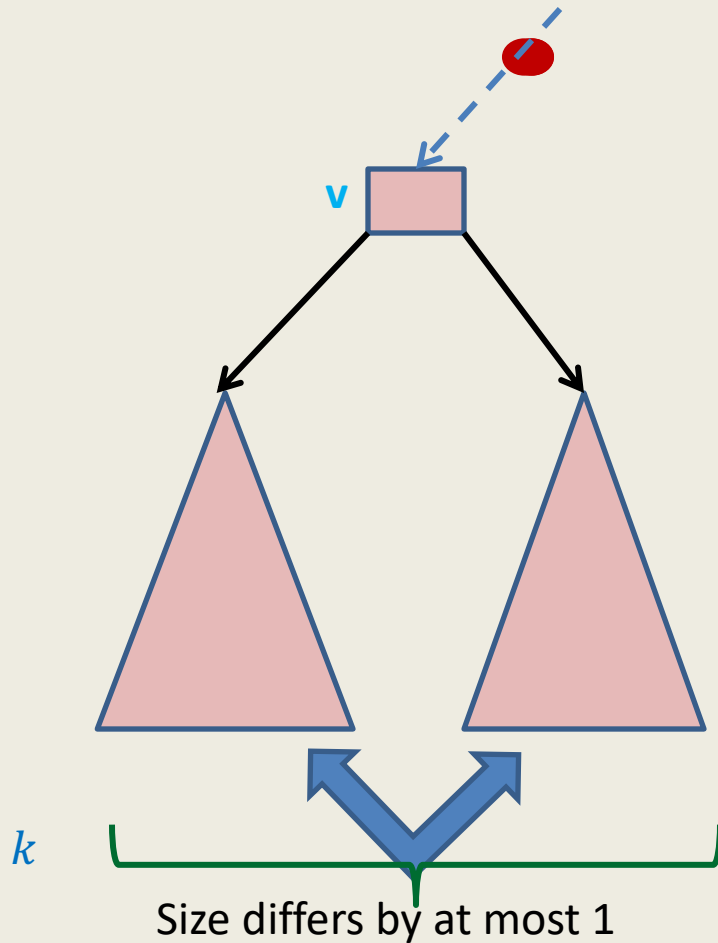
We shall now prove this theorem formally.

Watch the **next** slide slowly to get a useful insight.

How many new elements to make $T(v)$ imbalanced ?

$$\geq k$$

Suppose $T(v)$ is perfectly balanced at some moment.



The intuition for proving the Theorem

“A perfectly balanced subtree $T(v)$ will have to have large number of insertions before it becomes unbalanced enough to be rebuilt again.”

We shall transform this intuition into a formal proof now.

Notations

$\text{size}(v)$: no. of nodes in $T(v)$ at any moment.

For k th insertion,

$$I_k(v) = \begin{cases} 1 & \text{if } k\text{th insertion increases } \text{size}(v) \\ 0 & \text{Otherwise} \end{cases}$$

Question: For a nearly balanced BST, what is

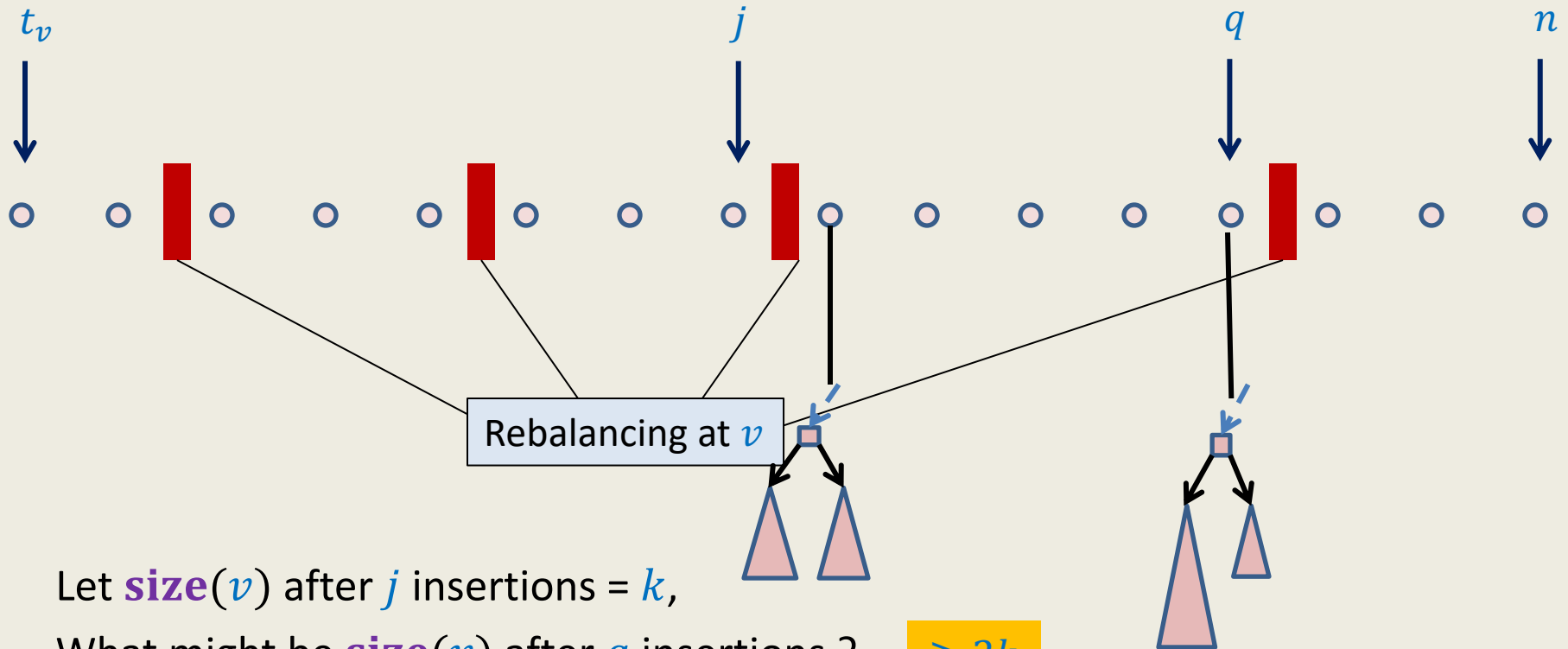
$$\sum_v I_k(v) = O(\log k)$$

$$\sum_{k=1 \text{ to } n} \sum_v I_k(v) = O(n \log n)$$

This is because

- T , being nearly Balanced, has $O(\log k)$ height.
and
- an insertion can increase **size** field for only the nodes lying along a root to leaf path.

Journey of an element/node v during n insertions



Let **size**(v) after j insertions = k ,

What might be **size**(v) after q insertions ? $\geq 2k$

Time complexity of rebalancing $\mathbf{T}(v)$ after q th insertion = $\mathbf{O}(k)$

What might be $\sum_{r=j+1}^q \mathbf{I}_r(v)$ $\geq k$

→ Time complexity of rebalancing $\mathbf{T}(v)$ after q th insertion = $\mathbf{O}(\sum_{r=j+1}^q \mathbf{I}_r(v))$

Time complexity of n insertions

For a vertex v ,

Time complexity of **rebalancing** $T(v)$ during n insertions = $\sum_{r=t_v}^n I_r(v)$

For all vertices,

the time complexity of **rebalancing** during n insertions = $\sum_v \sum_{r=t_v}^n I_r(v)$

After swapping these two “summations”

$$= \sum_{k=1 \text{ to } n} \sum_v I_k(v) = O(n \log n)$$

Theorem:

For any arbitrary sequence of n insert operations, total time to maintain nearly balanced BST will be $O(n \log n)$.