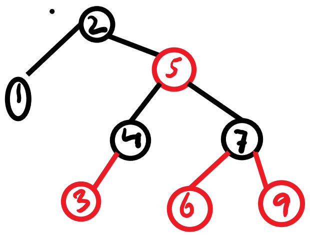



# CS 201 Data Structure II (L2 / L5)

# Binary Tree / Treap

**Section 6.1, 6.2, 7.2**

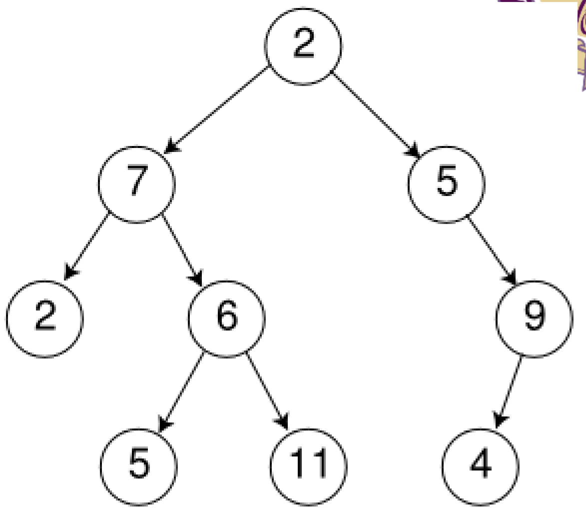
**Muhammad Qasim Pasta**

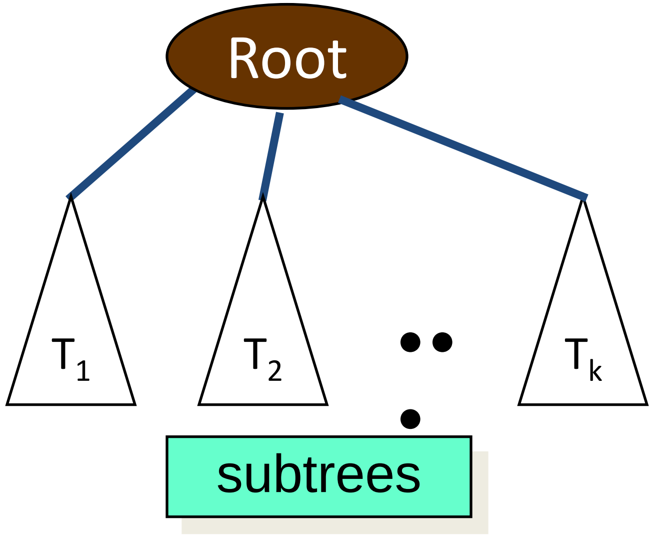
qasim.pasta@sse.habib.edu.pk

Some details are not mentioned as these slides are designed to be filled during the lectures using board and markers. Should not be used as reading.

# Tree and Binary Trees

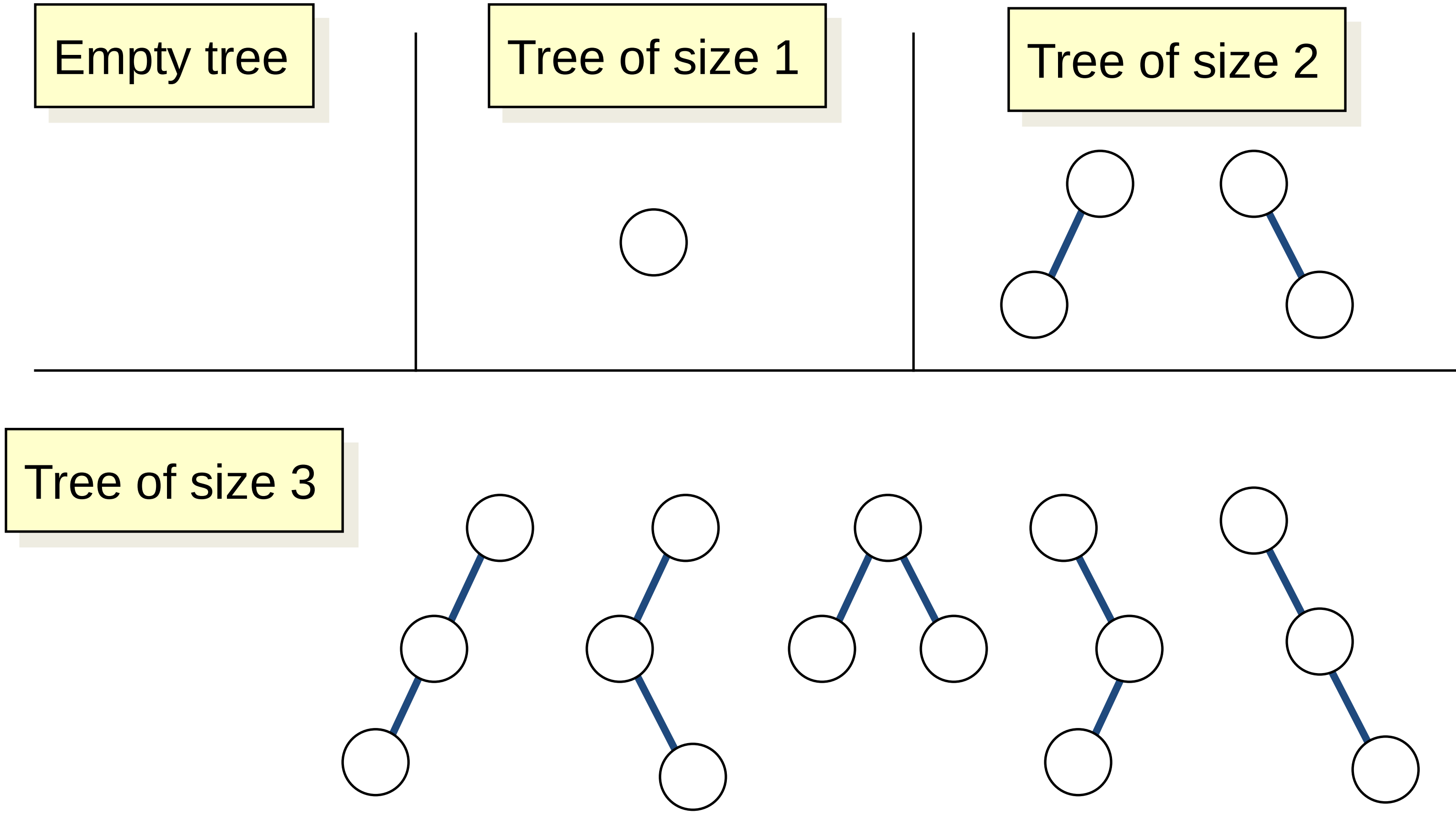
- Hierarchal data structure (non-linear)
- A set nodes and a set of directed edges that connects pair of nodes (non-recursive definition)
- Either a tree is empty or it consists of a root and zero or more nonempty subtrees $T_1$, $T_2$, … $T_k$, each of whose roots are connected by an edge from the root.(recursive definition)
- Acyclic



# Binary Tree

- A tree with no node more than two children
- A binary tree is either **empty**, or it consists of a node called the **root** together with **TWO binary trees** called the left subtree and the right subtree of the root.




Images are taken from slides of Saurav Karmakar

# Binary Tree Representation

[how to represent binary tree using node class]

# Terminologies

- Root node
- Leaf / parent / sibling
- Sub-tree
- Path/Path-length
- Height of a node / tree
- Size of a node / tree
- Depth of a node / tree

# Size / Height of a binary tree

[recursive code for size/height]

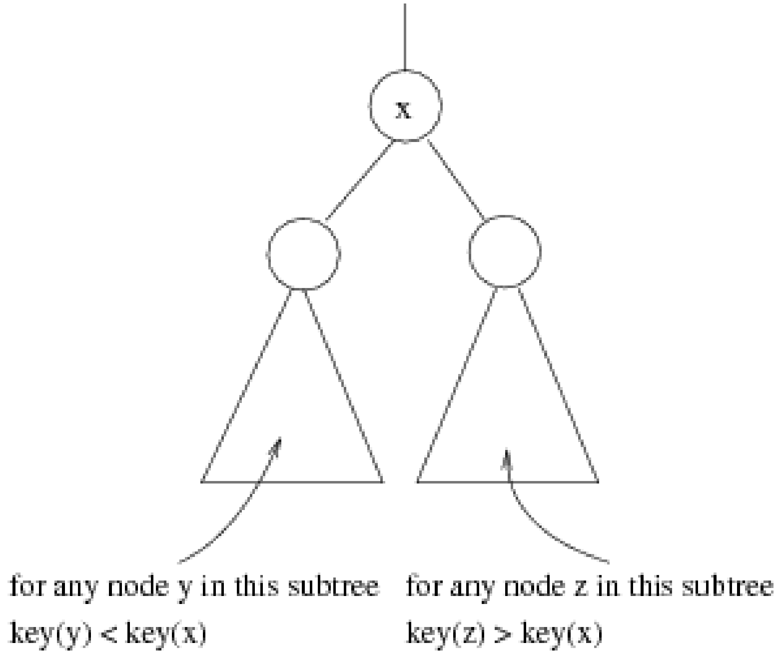
# Traversal

- Pre/Post/In-order

[traversal through activity]

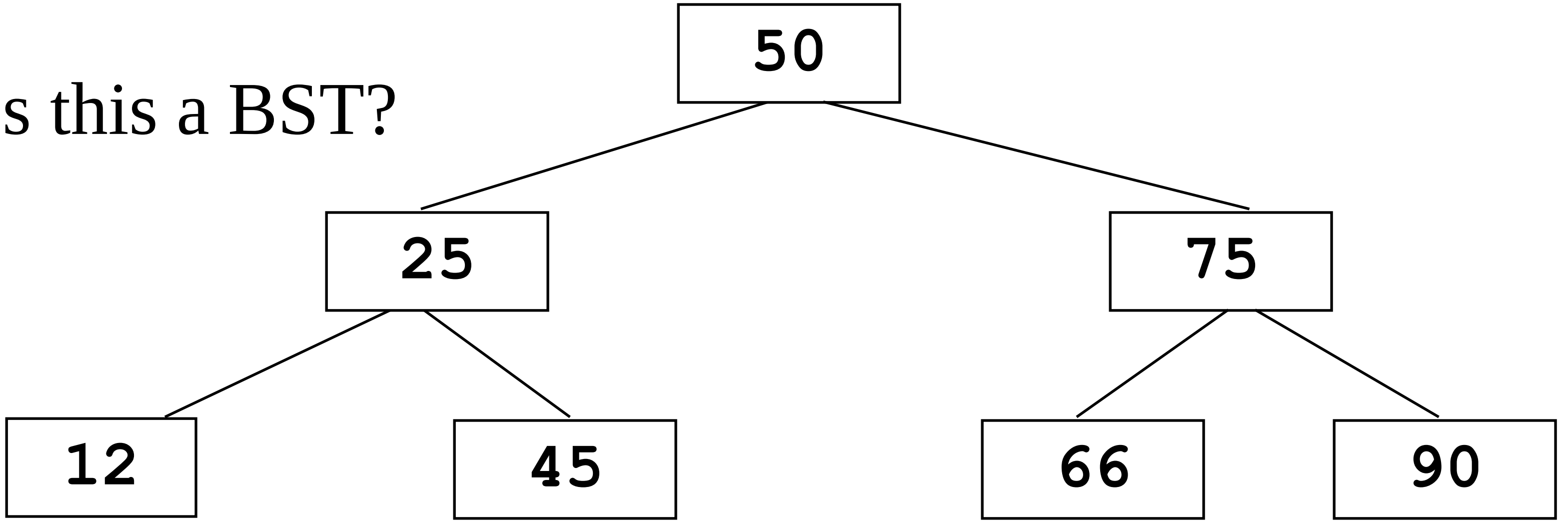
# Binary Search Tree

- A Binary Tree with following properties
- For every node X
  - All the keys in its left sub-tree are smaller than the key value in X
  - All the keys in its right sub-tree are larger than the key value in X

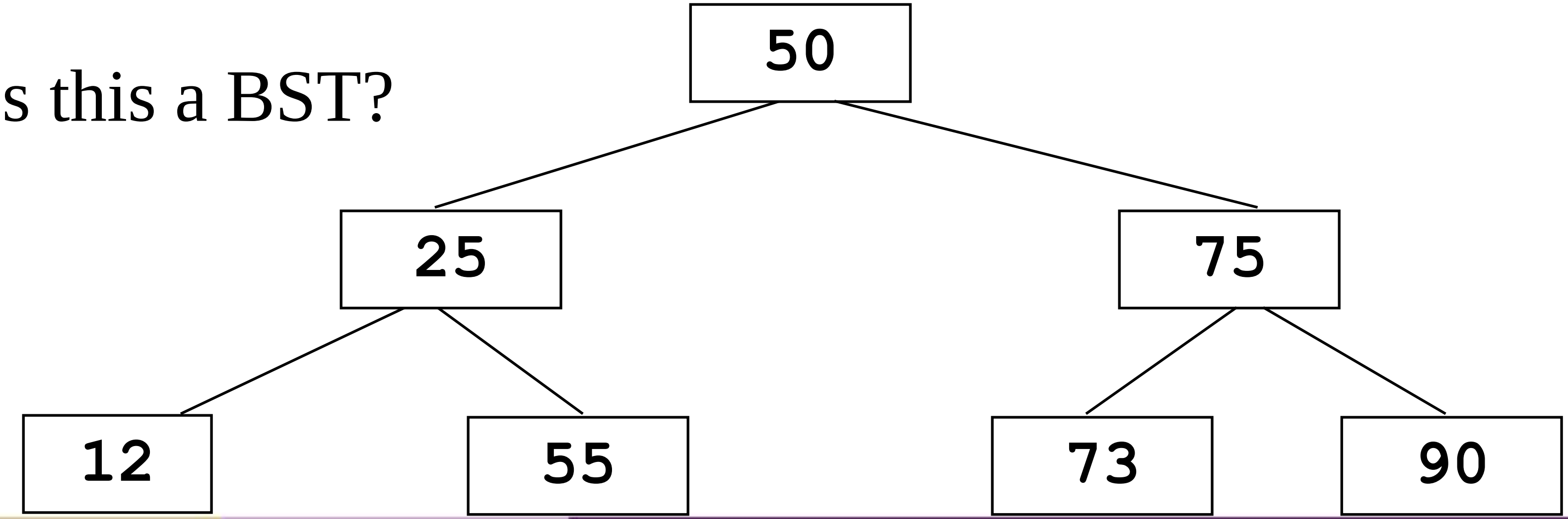



Images are taken from slides of Saurav Karmakar

Is this a BST?




Is this a BST?






Images are taken from slides of Saurav Karmakar

# Operations on BST

- Search
- Insert
- Delete
  - Case 1 – leave
  - Case 2 – one child
  - Case 3 – two children

- SSET as BST??

# Randomizes Binary Search Tree

- Search – O(h)
- Insert – O(h)
- Delete – O(h)
- The height of the tree depends on the sequence of numbers
  - 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14
    - h = n
  - 7,3,11,1,5,9,13,0,2,4,6,8,10,12,14
    - h  = log n
- How to prevent h = n?

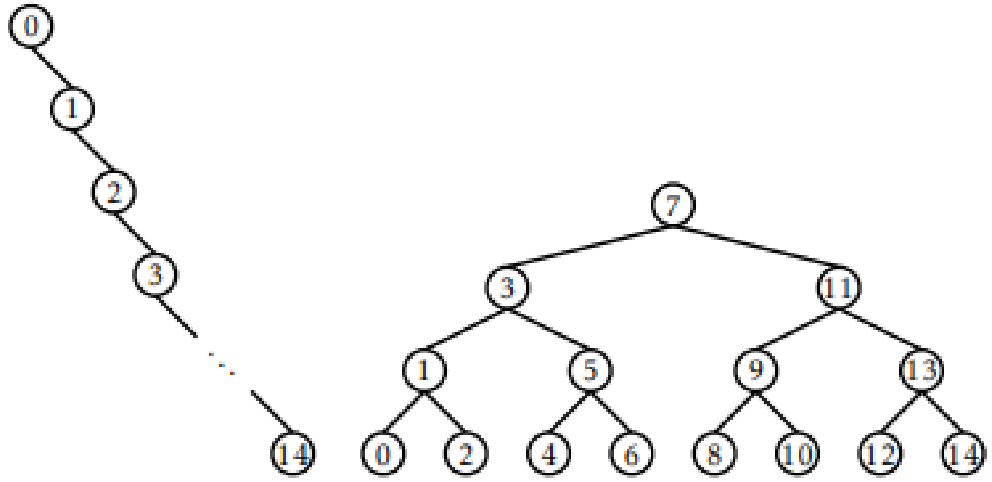
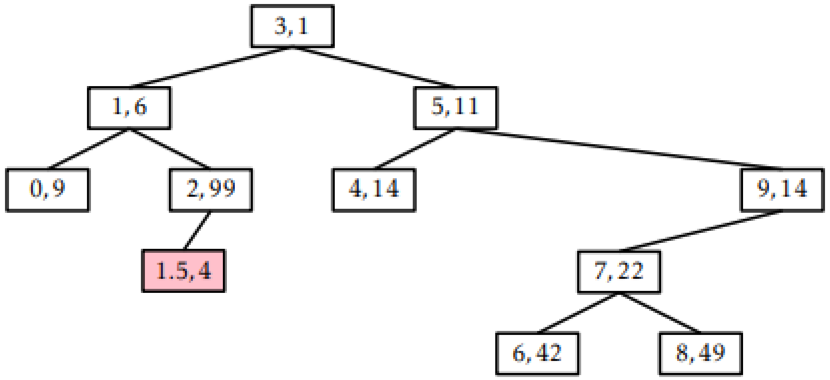



Figure 7.1: Two binary search trees containing the integers 0,…,14.

# Treap: Tree + Heap

- A Binary Search Tree with an addition value ‘p’ for each node and one additional property
  - At every node u, except the root, u.parent.p < u.p (Heap Property)
  - No parent should have a higher 'p' value than its children
  - The value of p – assigned randomly and unique
- add(x):
  - Create a new node ‘u’ with value x
  - Assign a random value for p
  - Insert using add(x) of BST – ‘u’ should be a leave
    - BST property maintains – heap property might not
  - Fix by Bubble Up



# Rotations

- We can fix heap property by performing rotations
- Rotate Right = make the left child as a parent
- Rotate Left = make the right child as a parent
- Decrease (/increase) the depth by one

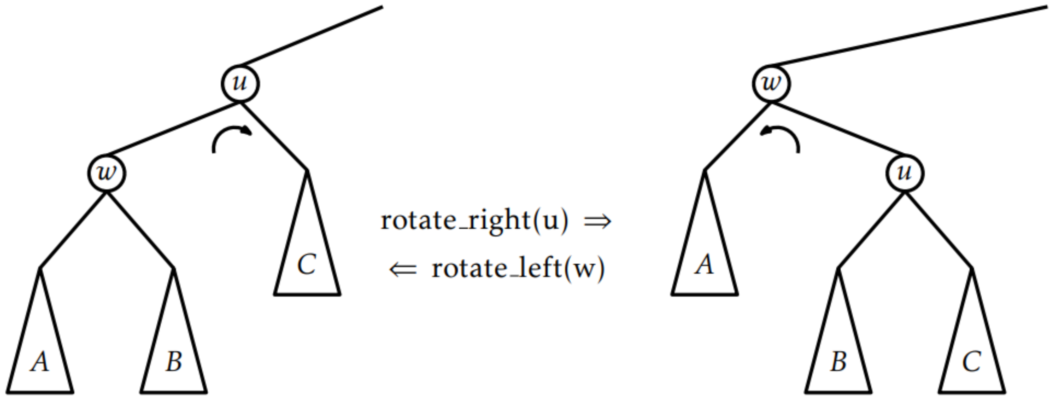
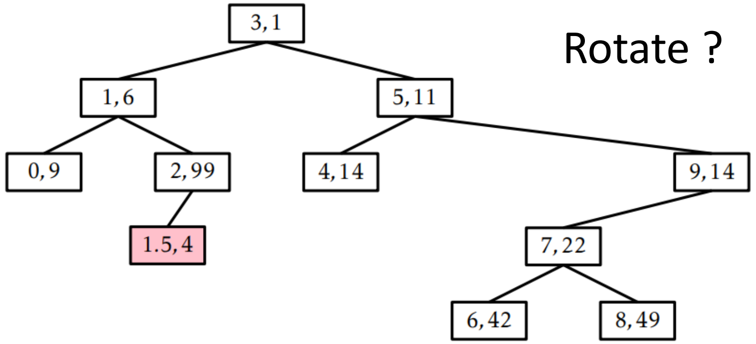


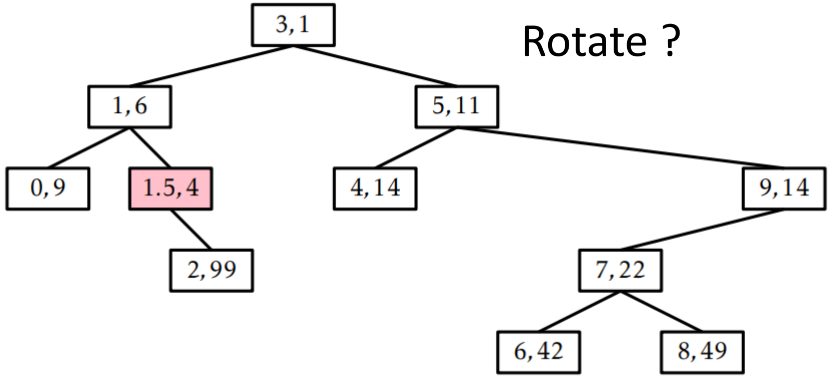

Figure 7.6: Left and right rotations in a binary search tree.

# Bubble Up

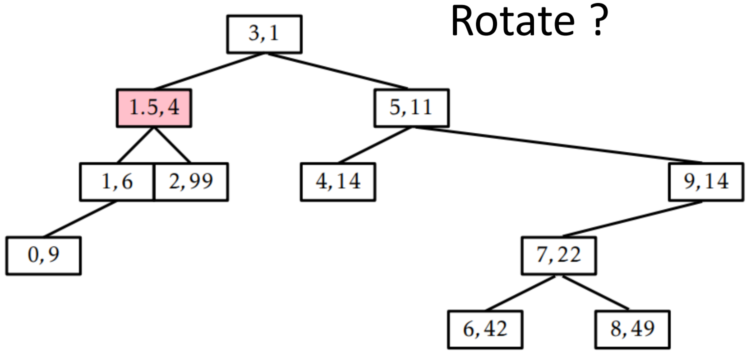
- Move element to up by performing rotations




Rotate ?



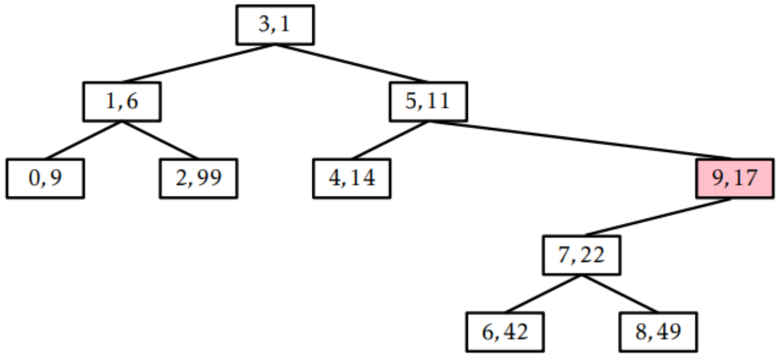

Rotate ?

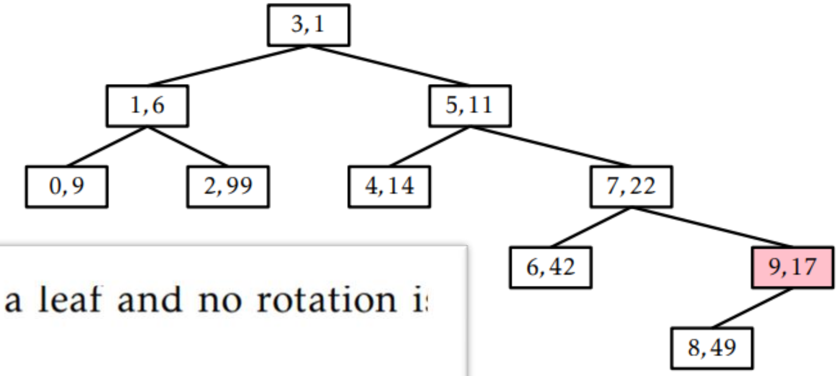


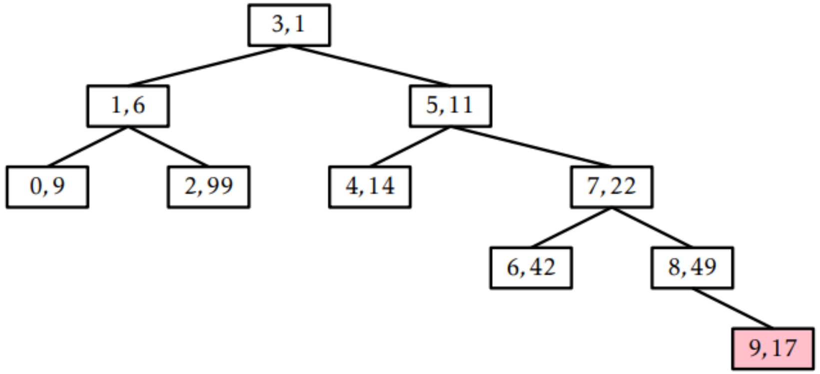

Rotate ?

# Remove: Trickle Down (reverse of Bubble Up)

- Move the element as a leaf by performing rotations




1. If *u.left* and *u.right* are both *nil*, then *u* is a leaf and no rotation i performed.
2. If *u.left* (or *u.right*) is *nil*, then perform a right (or left, respectively rotation at *u*.
3. If $u.left.p < u.right.p$ (or $u.left.p > u.right.p$), then perform a right ro tation (or left rotation, respectively) at *u*.

# Analysis of Operations

- find(x) = $O(h)$ => $O(\log n)$ [expected]
  - Same as BST
- add(x) = $O(h)$ => $O(\log n)$ [expected]
  - Insertion – O(h)
  - Bubble up – O(h)
- remove(x) = $O(h)$ => $O(\log n)$ [expected]
  - Search + trickle down – O(h)

# Construct treap for the following sequences:

1. (2,7), (3,6), (5,4), (8,12)

2. Generate priority randomly.

(2, _ ), (3, _ ), (5, _ ), (1, _ ), (7, _)