Design and Analysis of Algorithms

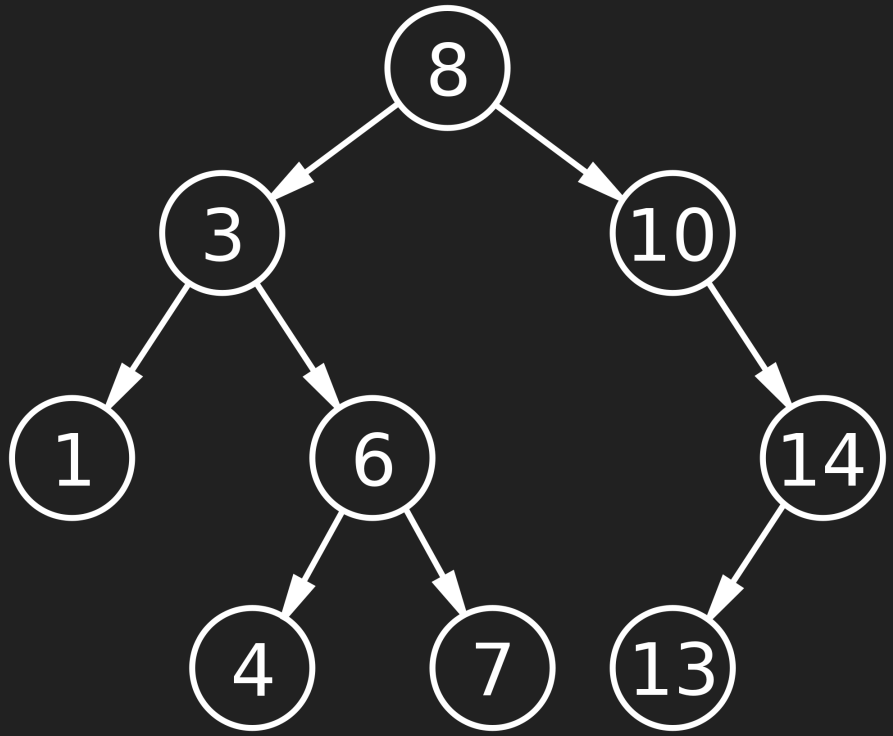
Search Trees

Mr. Channa Bankapur

Binary Trees

Binary Search Trees (BST)

What do we “conquer” by transforming a **Binary Tree** into a **Binary Search Trees** (BST)?

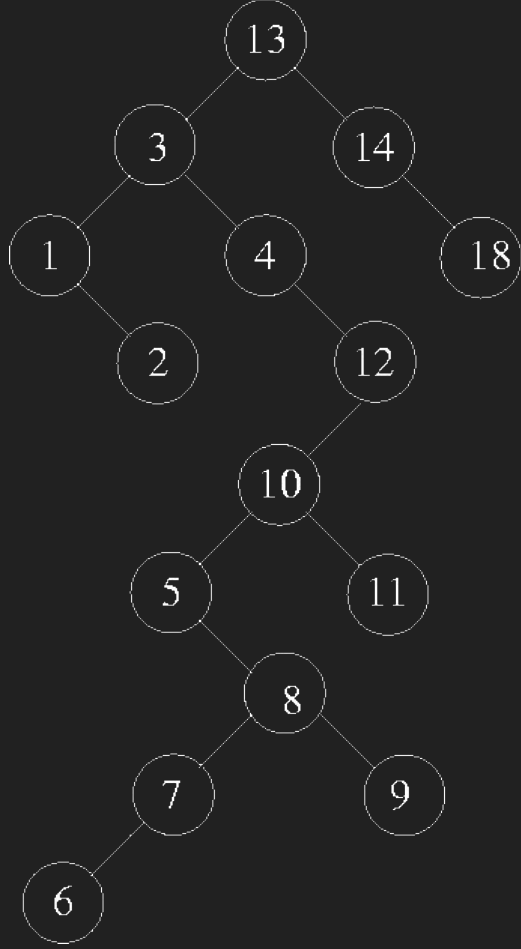


Binary Search Trees (BST)

What do we “conquer” by transforming a Binary Tree into a BST?
Search!

Time complexity of

- Inserting an element into a BST: **O(?)**
- Searching for an element in a BST: **O(?)**

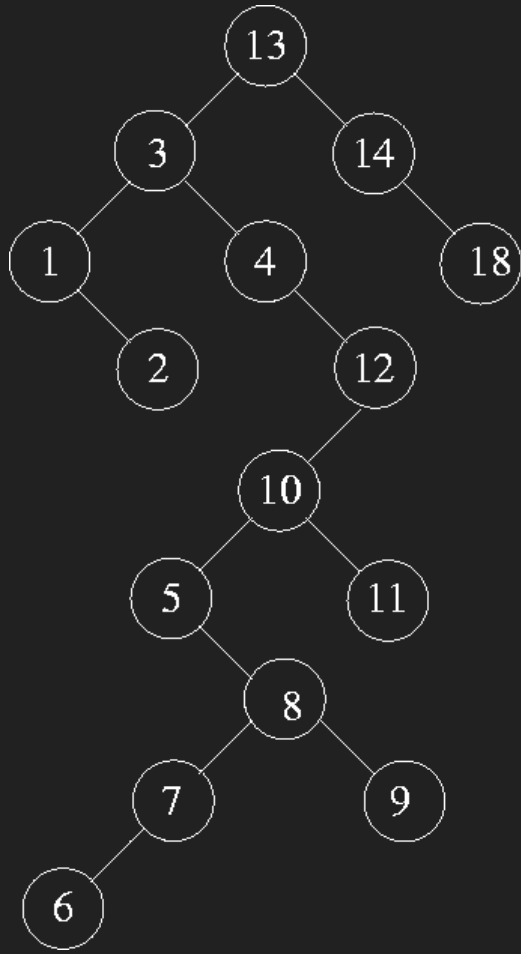


Binary Search Trees (BST)

What do we “conquer” by transforming a Binary Tree into a BST?
Search!

Time complexity of

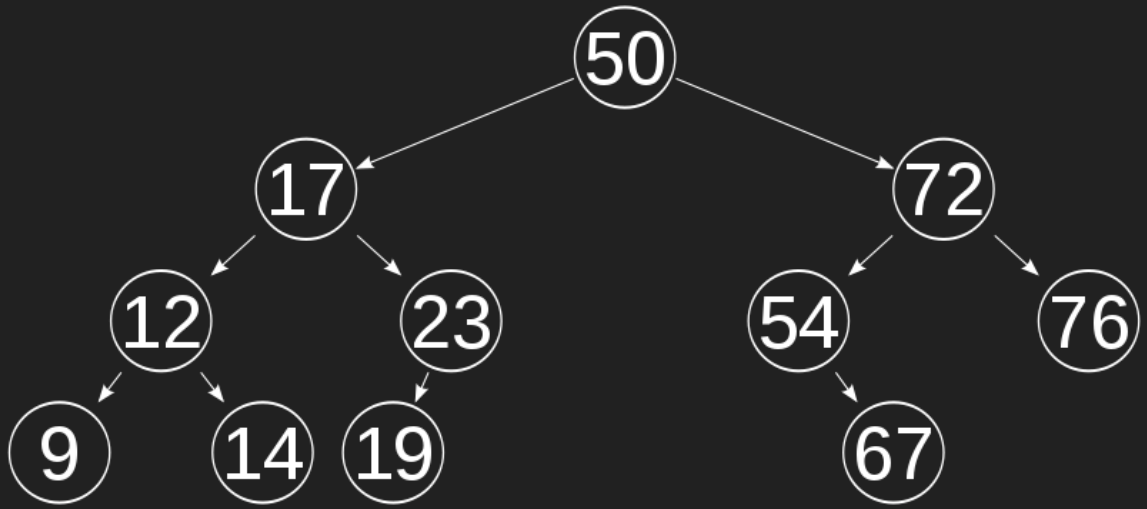
- Inserting an element into a BST: **O(n)**
- Searching for an element in a BST: **O(n)**
- **Because height = O(n)**



Balanced Binary Search Trees:

Time complexity of worst-case of search in a BST is **O(n)**

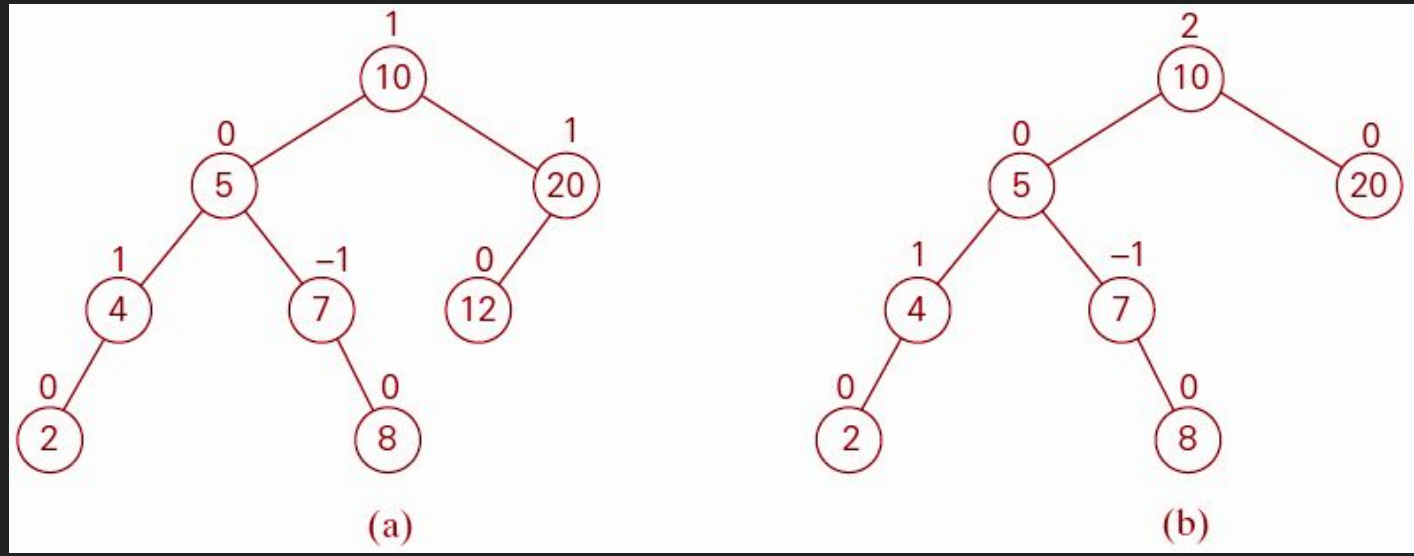
If we can we keep the BST balanced (that is, the height of the tree is limited to **O(log n)**), then the worst-case of search will be in **O(log n)**.



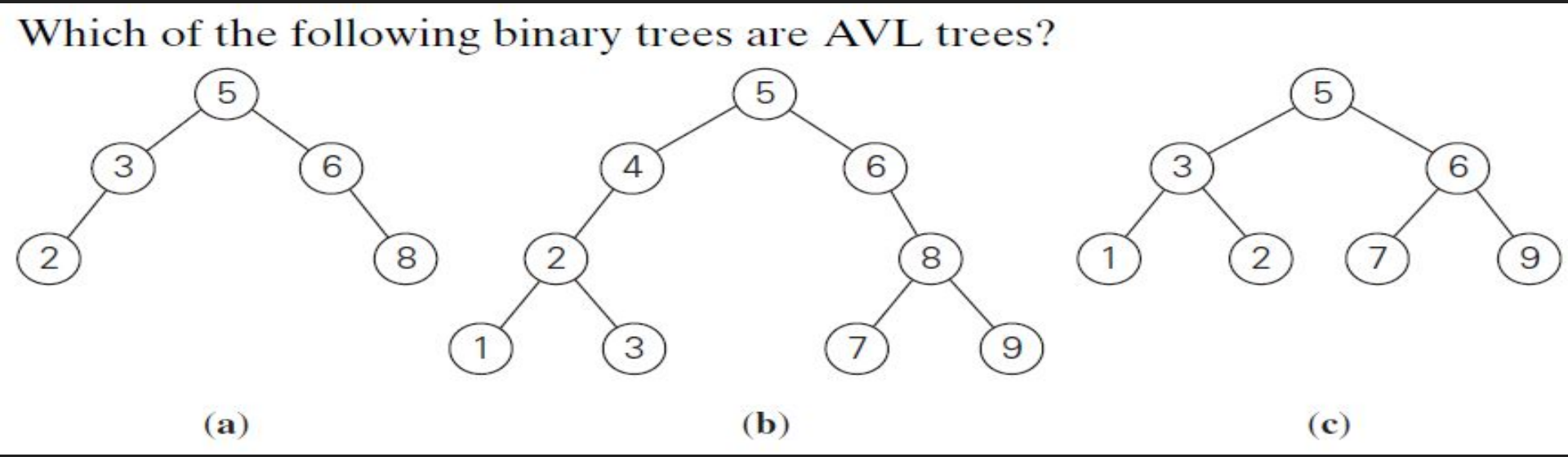
Balanced Binary Search Trees:
If we can we keep the BST balanced (that is, the height of the tree is limited to **O(log n)**), then the worst-case of search will be in **O(log n)**.

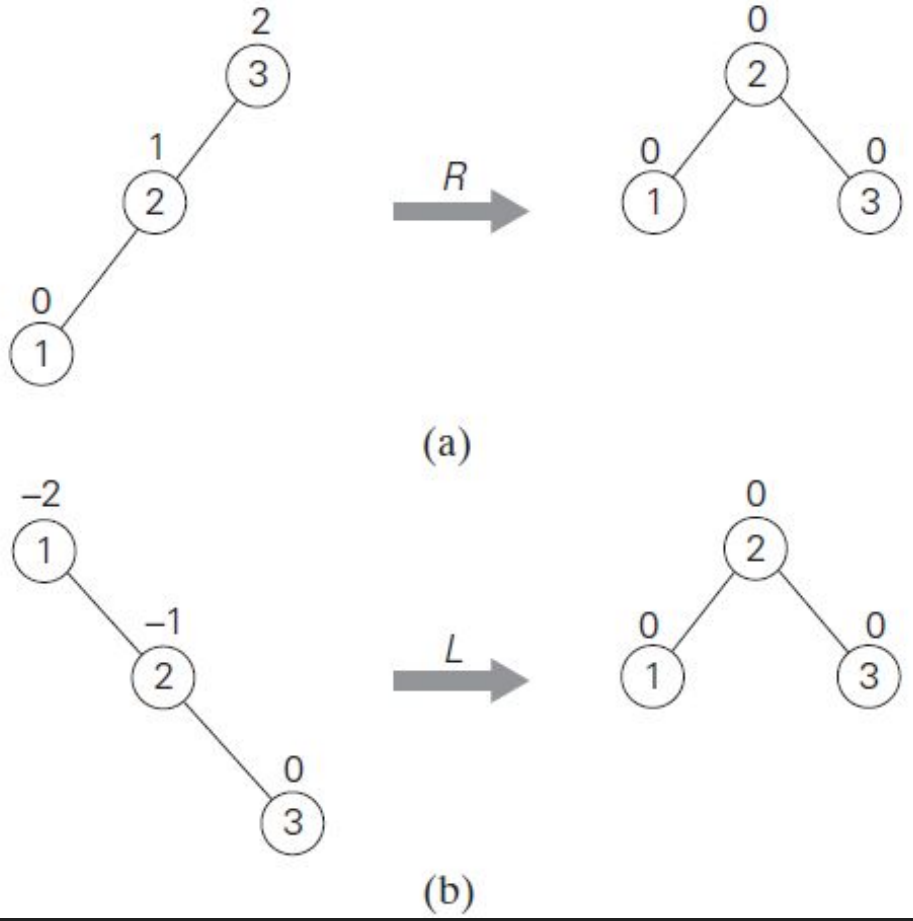
1. AVL Trees
2. Red-Black Trees
3. 2-3 Trees
    a. Not a binary tree, it’s a **Balanced Search Tree**.
4. B Trees
    a. Not a binary tree, it’s a **Balanced Search Tree**.

- **AVL Trees:** An AVL tree is a BST in which the **balance factor** of every node is either 0, +1 or -1.
- **Balance factor:** The difference between the heights of the node's left and right subtrees.
- **Height of a tree:** The longest path from the root to a leaf. The height of a null tree is defined as -1.

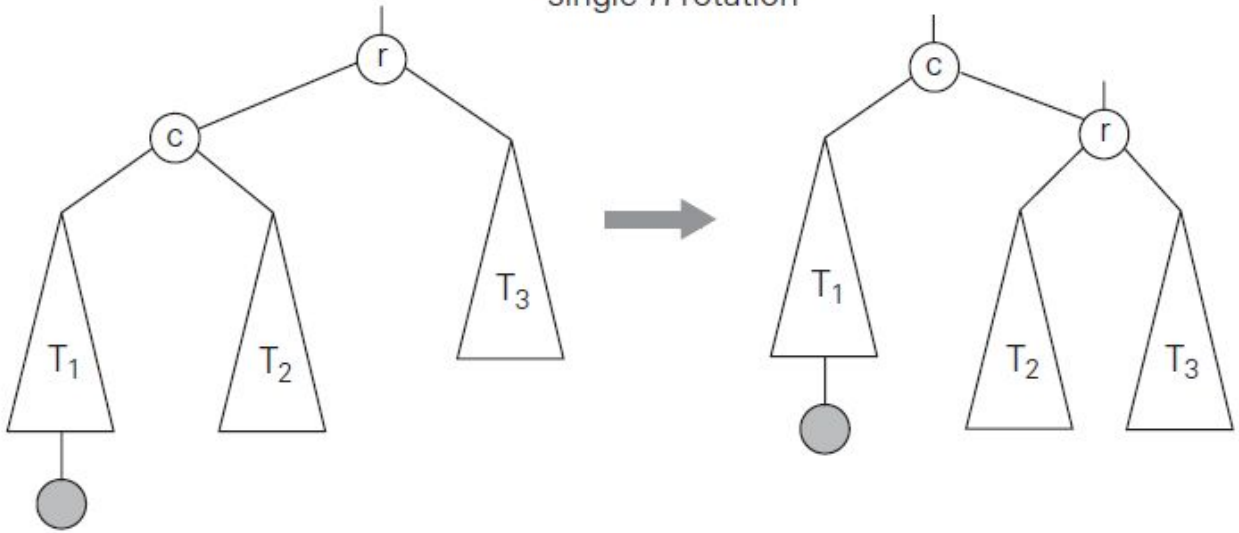


- **AVL Trees:** An AVL tree is a BST in which the **balance factor** of every node is either 0, +1 or -1, where the balance factor is the difference between the heights of the node's left and right subtrees.

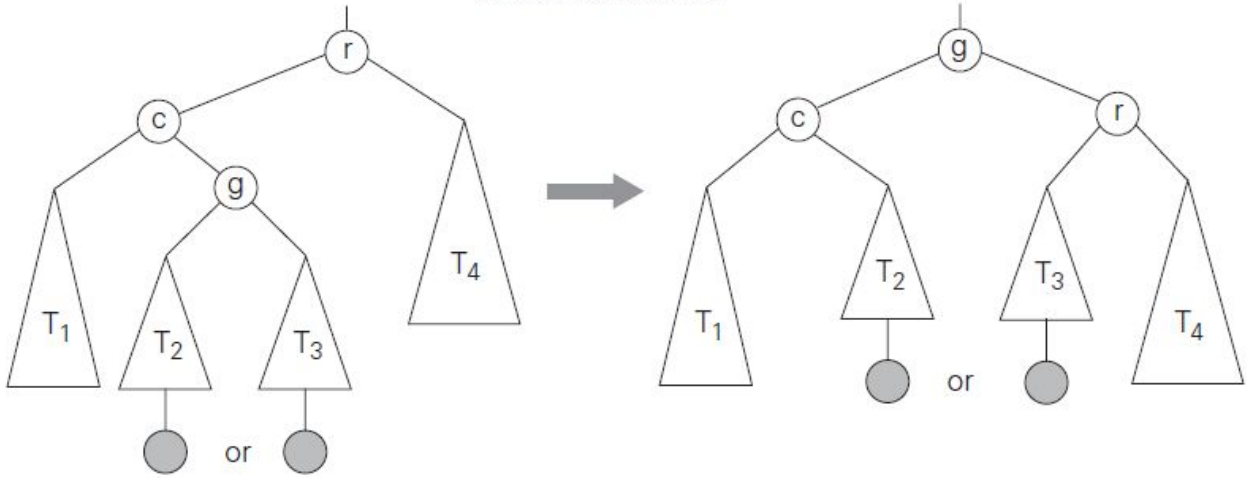



2
3
1
2
0
1
R
0
2
0
1
0
3
(a)
–2
1
–1
2
0
3
L
0
2
0
1
0
3
(b)
2
3
–1
1
0
2
LR
0
2
0
0
1
3
(c)
–2
1
1
3
0
2
RL
0
2
0
0
1
3
(d)

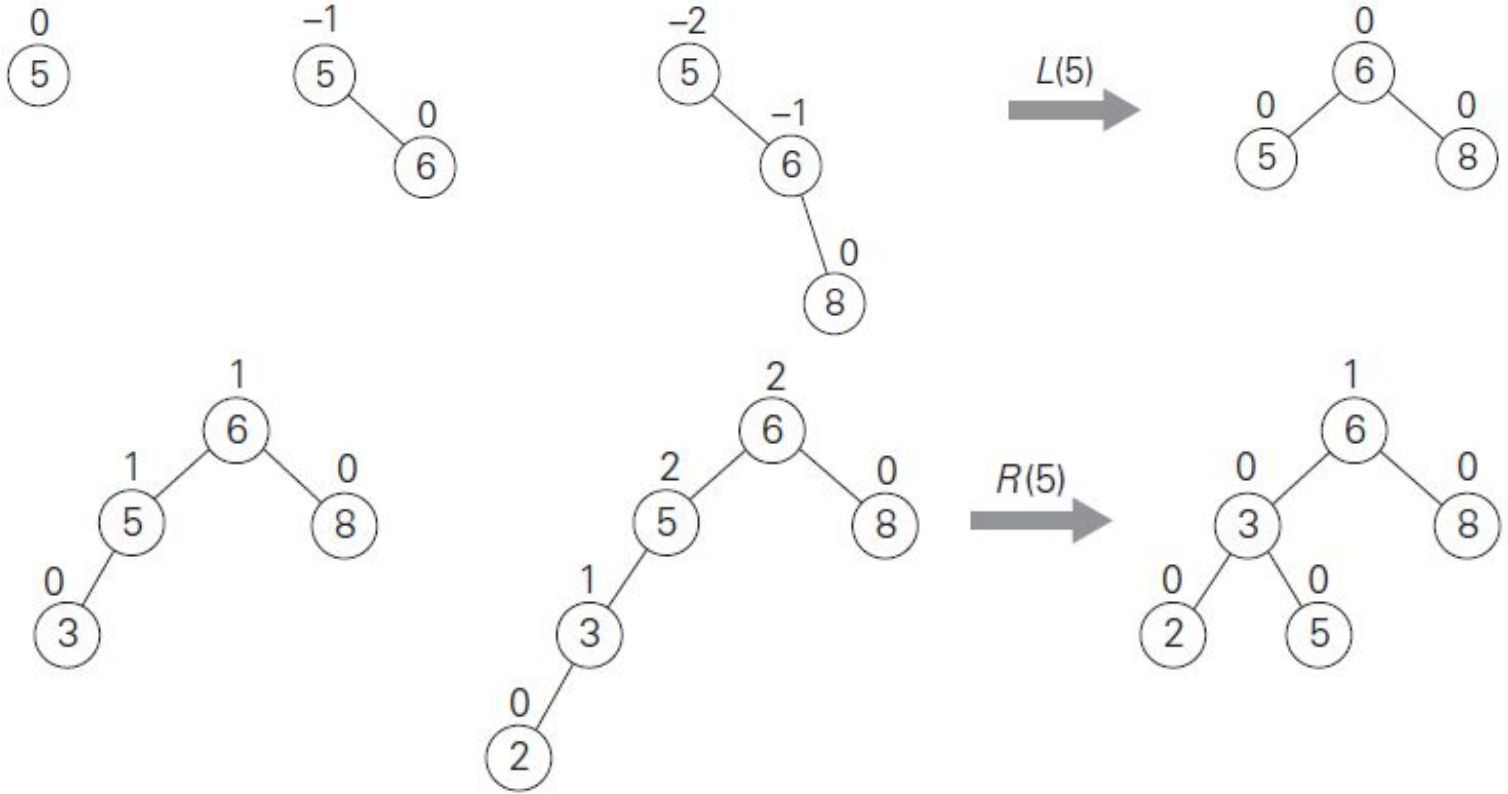
single *R*-rotation



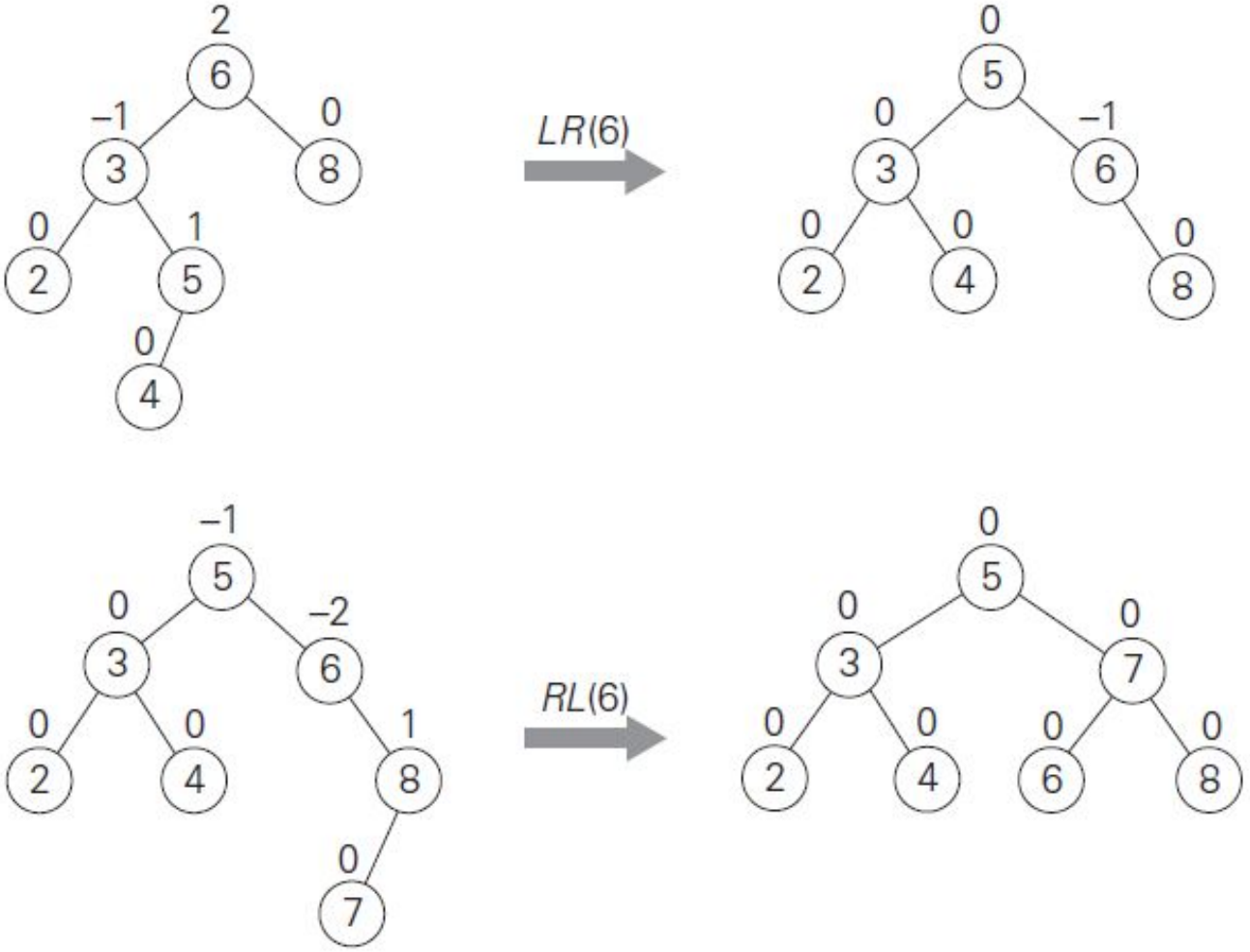
double *LR*-rotation



Construction of an AVL tree for the list 5, 6, 8, 3, 2, 4, 7



Construction of an AVL tree for the list 5, 6, 8, 3, 2, 4, 7

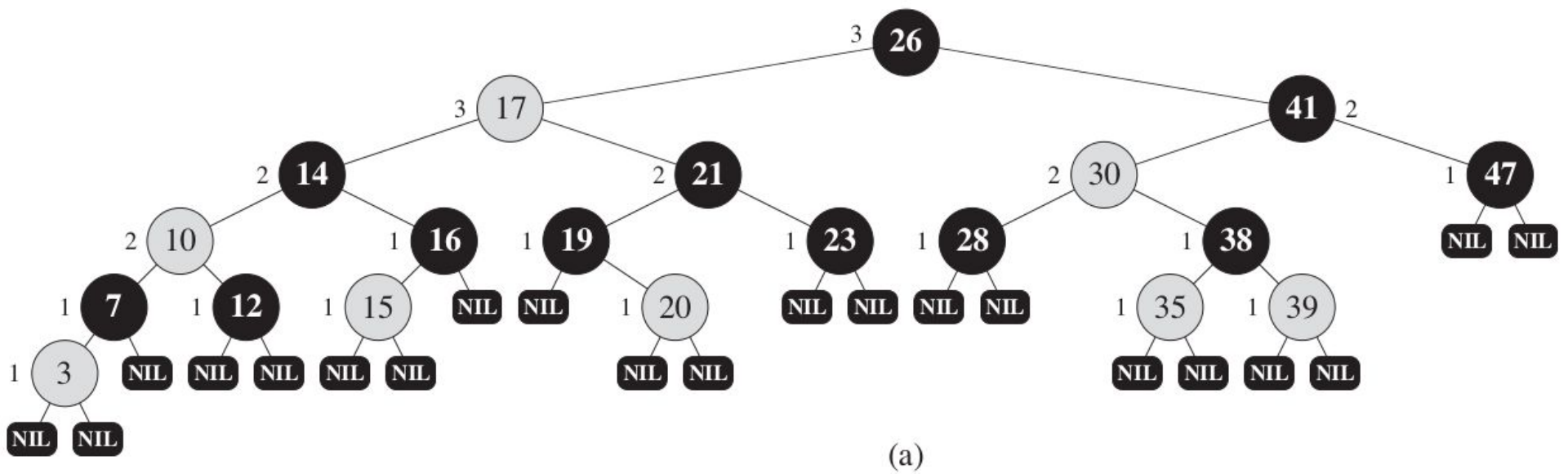


Search time efficiency of AVL trees:
Search is bounded by the height h of the tree.
Height is bounded both above and below by logarithmic functions.

$$\lfloor \log_2 n \rfloor \leq h < 1.4405 \log_2(n+2) - 1.3277$$

Red Black Tree

- A red-black tree is a binary search tree with one extra bit of storage per node: its color, which can be either RED or BLACK.
- By constraining the node colors on any simple path from the root to a leaf, red-black trees ensure that no such path is more than twice as long as any other, so that the tree is approximately balanced.



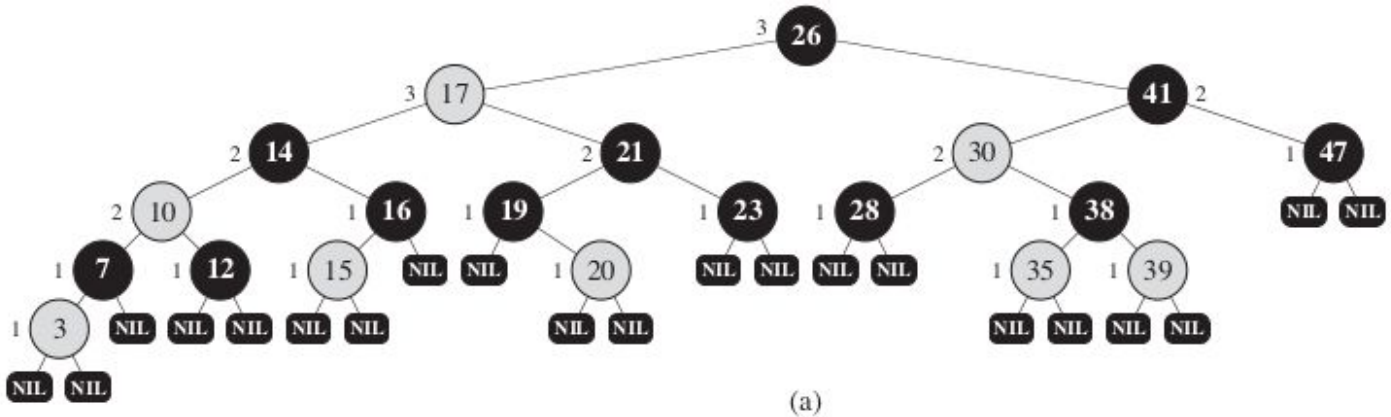
(a)

Red Black Tree

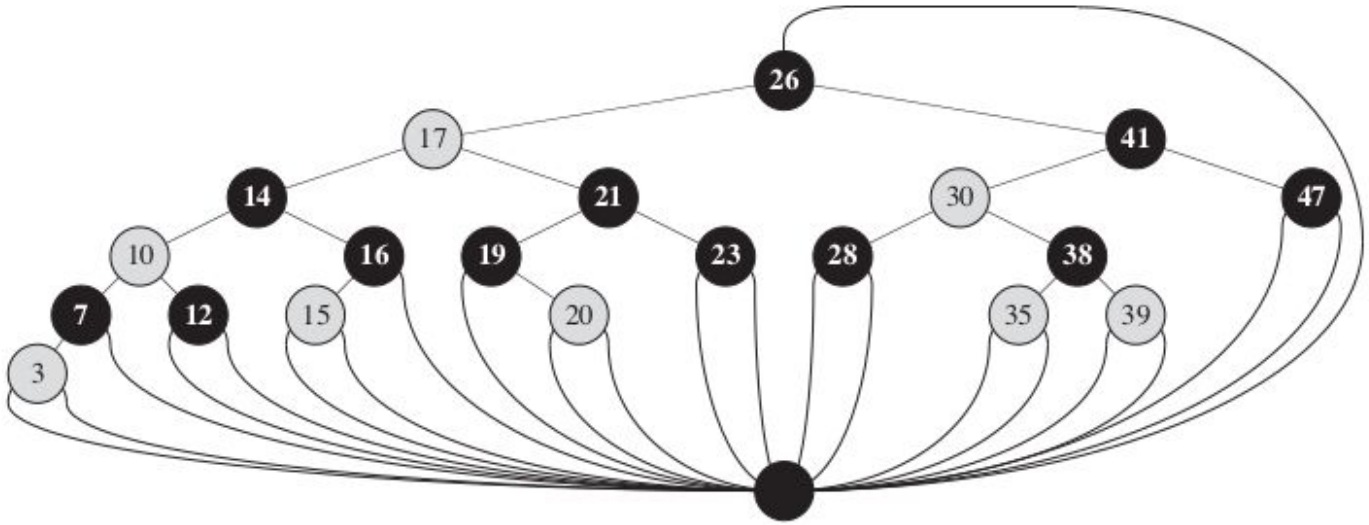
A red-black tree is a binary search tree that satisfies the following red-black properties:

1. Every node is either red or black.
2. The root is black.
3. Every leaf (NIL) is black.
4. If a node is red, then both its children are black.
5. For each node, all simple paths from the node to descendant leaves contain the same number of black nodes.

Red
Black
Tree



(a)



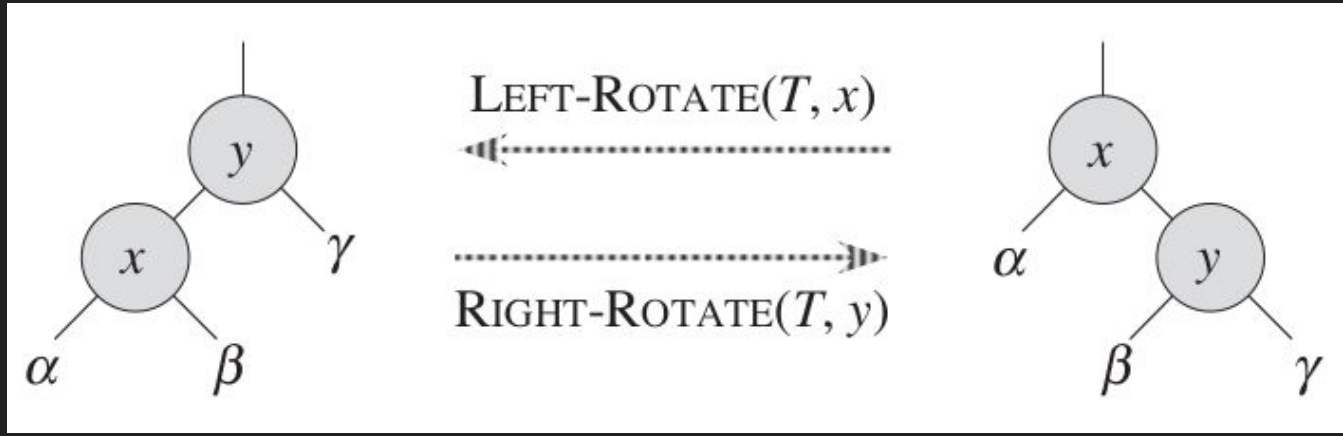
(b)

Red Black Tree: Insert operation

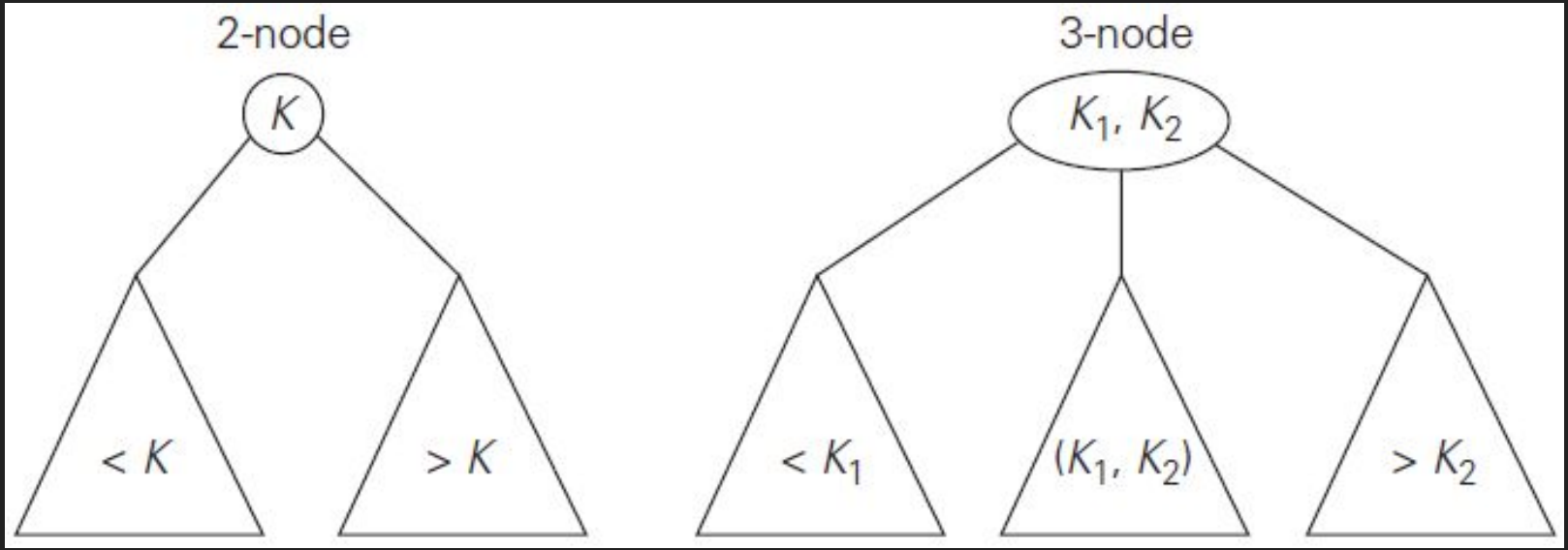
1. Insert in the usual way of inserting a node into a BST.
2. Color the new node red.
3. Fix the colors if the parent is also red.
4. If it needs a rotate, use LEFT-ROTATE(T, x) or RIGHT-ROTATE(T, y).
5. Go to step 3.

O(log n)

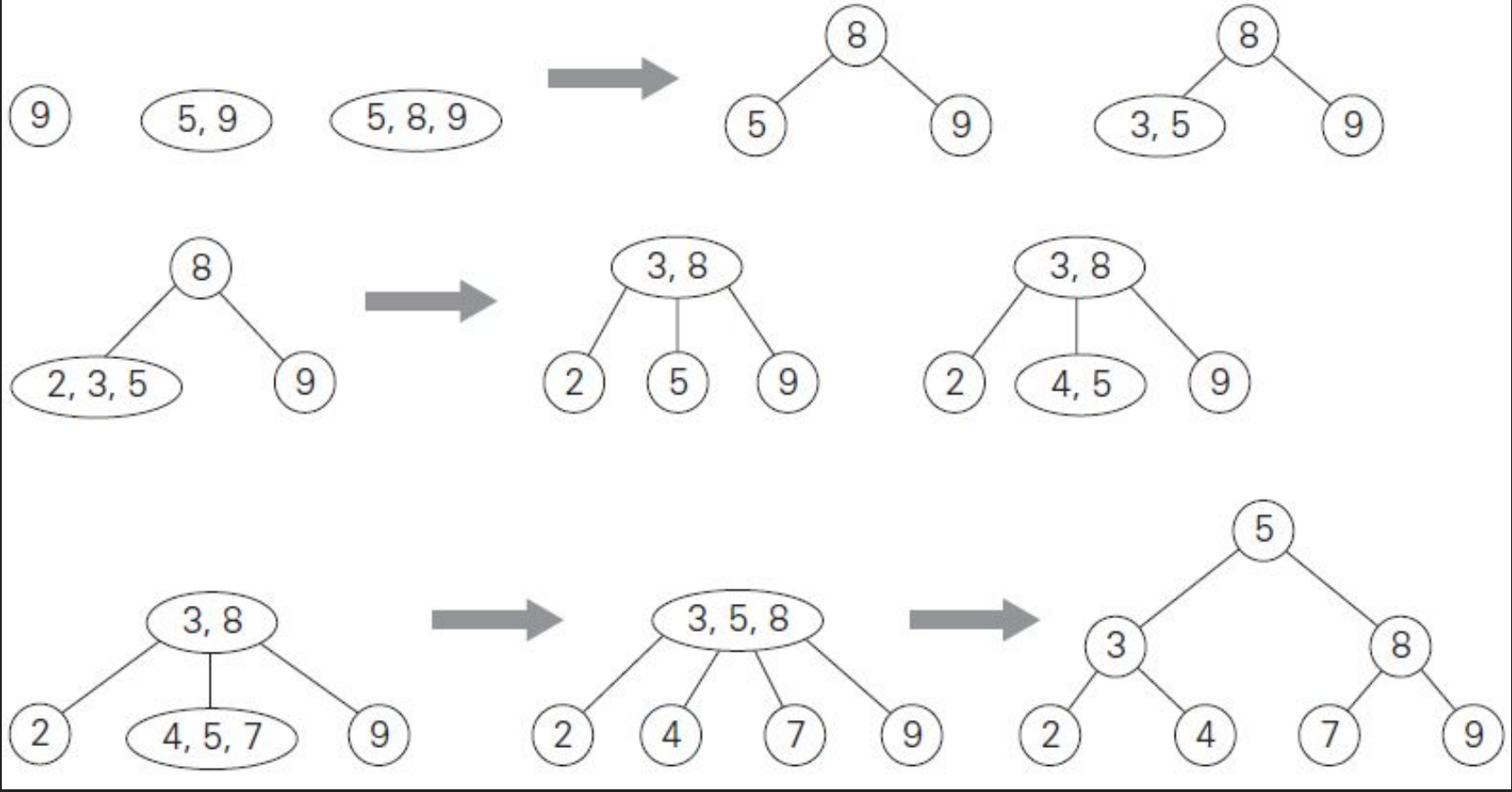
Refer CLRS book

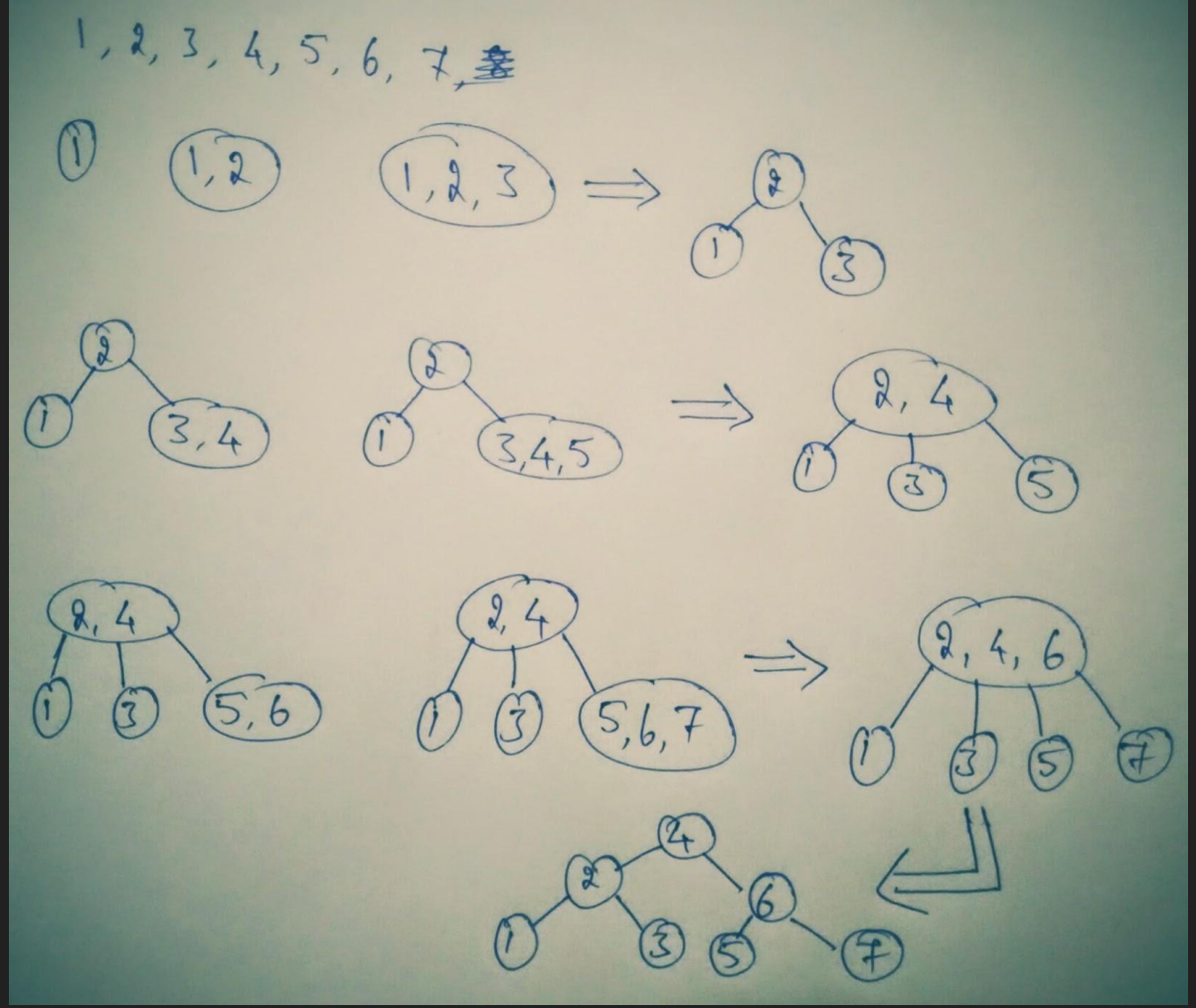


2-3 Trees: A 2-3 tree is a tree that can have nodes of two kinds; 2-nodes and 3-nodes. All its leaves must be on the same level so that a 2-3 tree is always height-balanced.

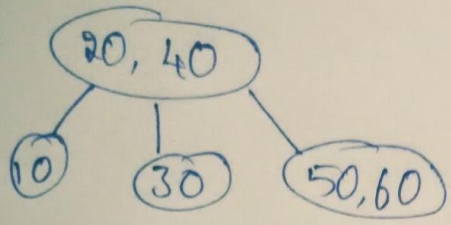


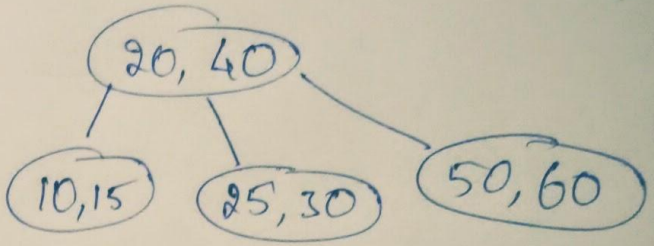
Construction of a 2-3 tree for the list 9, 5, 8, 3, 2, 4, 7.

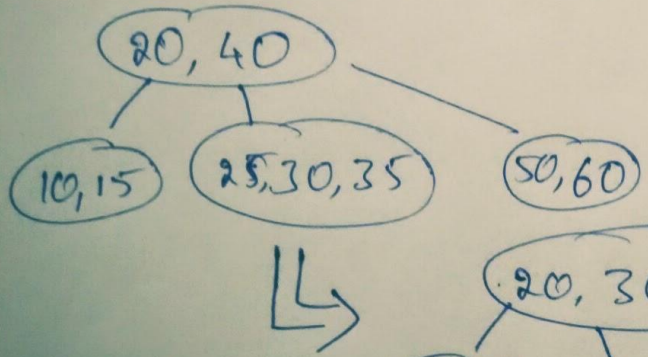


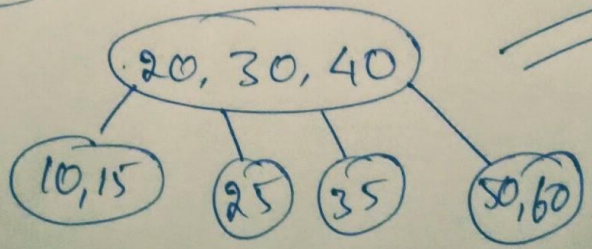

1, 2, 3, 4, 5, 6, 7, 奏
1
1, 2
1, 2, 3
2
1
3
2
1
3, 4
2
1
3, 4, 5
2, 4
1
3
5
2, 4
1
3
5, 6
2, 4
1
3
5, 6, 7
2, 4, 6
1
3
5
7
4
2
6
1
3
5
7

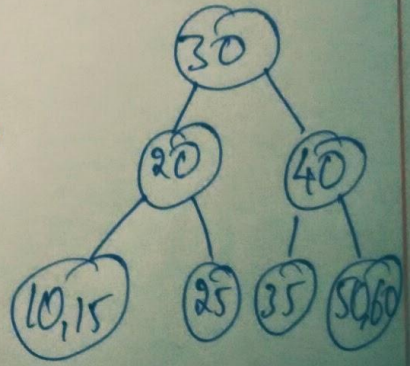
10, 20, 30, 40, 50, 60, 15, 25, 35

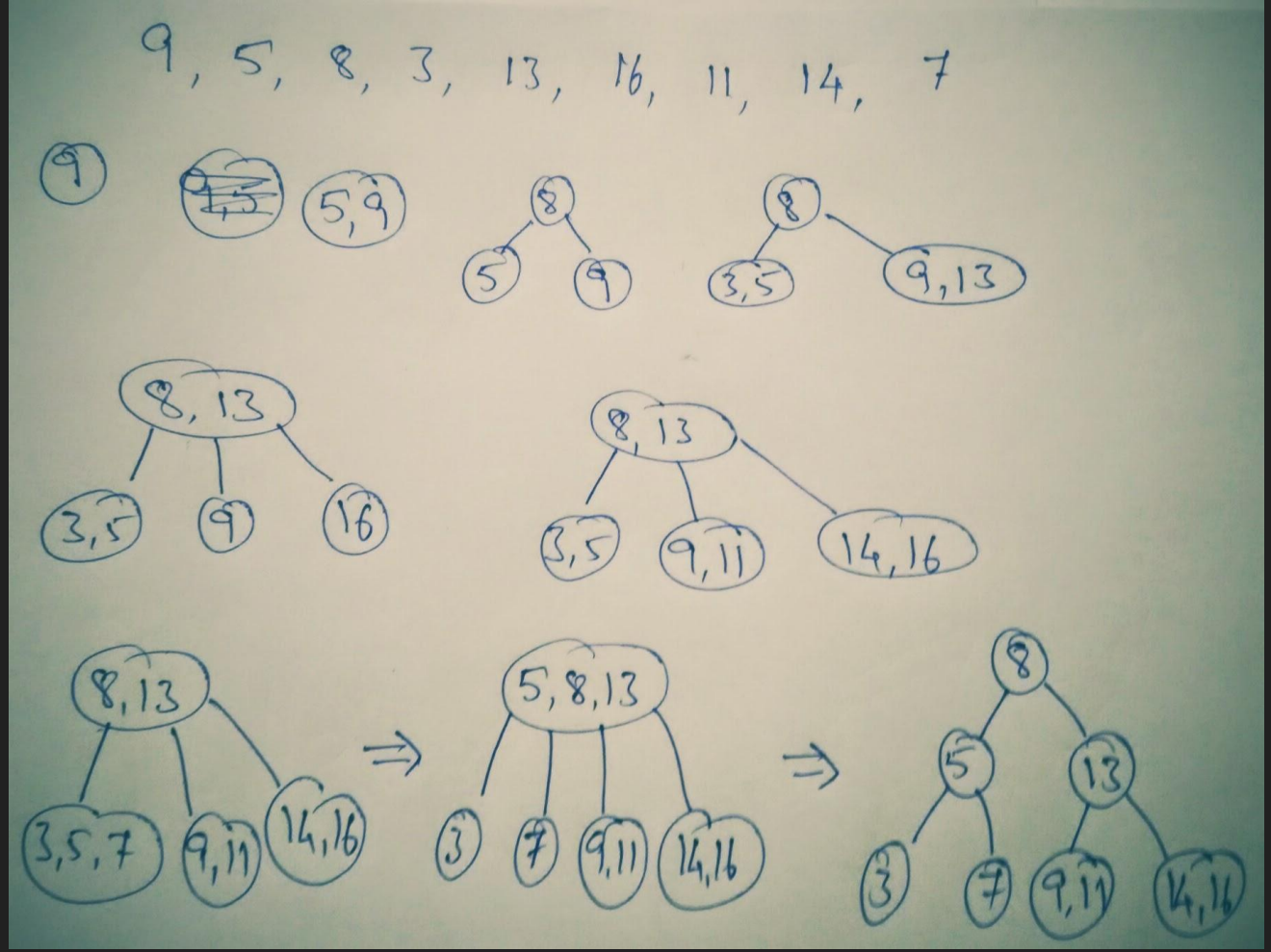











9, 5, 8, 3, 13, 16, 11, 14, 7
9
5,9
8
5
9
8
3,5
9,13
8,13
3,5
9
16
8,13
3,5
9,11
14,16
8,13
3,5,7
9,11
14,16
5,8,13
3
7
9,11
14,16
8
5
13
3
7
9,11
14,16

Analysis of 2-3 Tree:

Lower Bound on the number of keys for height h:

$$n \geq 1 + 2 + \cdots + 2^h = 2^{h+1} - 1$$

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

Upper Bound on the number of keys for height h:

$$n \leq 2 \cdot 1 + 2 \cdot 3 + \cdots + 2 \cdot 3^h = 2(1 + 3 + \cdots + 3^h) = 3^{h+1} - 1$$

$$h \geq \log_3(n+1) - 1$$

$$\log_3(n+1) - 1 \leq h \leq \log_2(n+1) - 1$$

Analysis of 2-3 Tree:

Lower Bound on n for a given h

$$n \geq 1 + 2 + 4 + \cdots + 2^h$$
$$n \geq 2^{h+1} - 1$$
$$n+1 \geq 2^{h+1}$$
$$\log_2(n+1) \geq \log_2 2^{h+1}$$
$$\log_2(n+1) \geq h+1$$
$$h \leq \log_2(n+1) - 1$$

Upper Bound on n for a given h

$$n \leq 2 \cdot 1 + 2 \cdot 3 + 2 \cdot 3^2 + \cdots + 2 \cdot 3^h$$
$$\leq 2(1 + 3^1 + 3^2 + \cdots + 3^h)$$
$$\leq 3^{h+1} - 1$$
$$n+1 \leq 3^{h+1}$$
$$\log_3(n+1) \leq \log_3(3^{h+1})$$
$$\log_3(n+1) \leq h+1$$
$$\log_3(n+1) - 1 \leq h$$

$$\therefore \boxed{\log_3(n+1) - 1 \leq h \leq \log_2(n+1) - 1}$$

Thank You :-)

Mr. Channa Bankapur