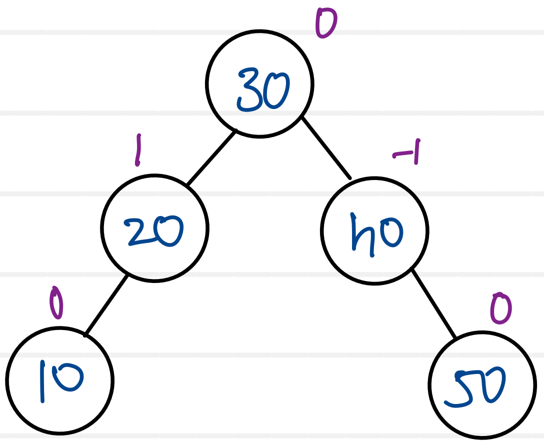
Sunbeam Institute of Information Technology
Pune and Karad

Algorithms and Data structures

Trainer - Devendra Dhande
Email – devendra.dhande@sunbeaminfo.com

Skewed Binary Search Tree

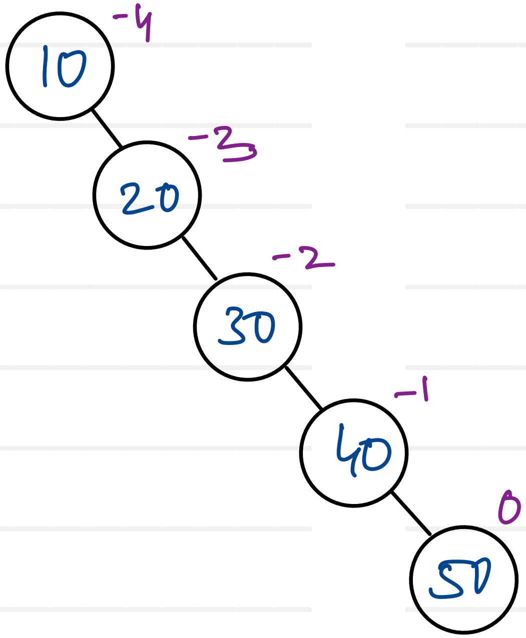
Keys : 30, 40, 20, 50, 10



$h = \log n$

$T(n) = O(\log n)$

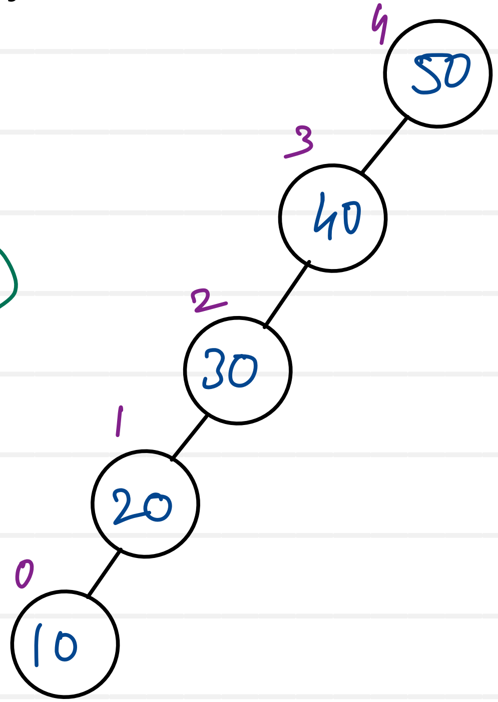
Keys : 10, 20, 30, 40, 50



$h = n$

$T(n) = O(n)$

Keys : 50, 40, 30, 20, 10



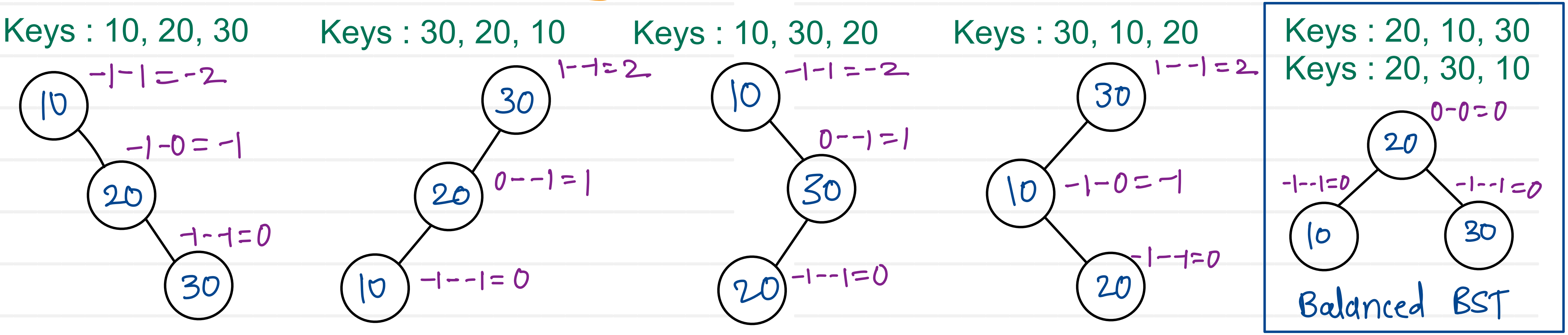
- In binary tree if only left or right links are used, tree grows only in one direction such tree is called as skewed binary tree
  - Left skewed binary tree
  - Right skewed binary tree
- Time complexity of any BST is O(h)
- Skewed BST have maximum height ie same as number of elements.
- Time complexity of searching is skewed BST is O(n)

Balanced BST → ① AVL tree ② Red-Black tree ③ Splay tree

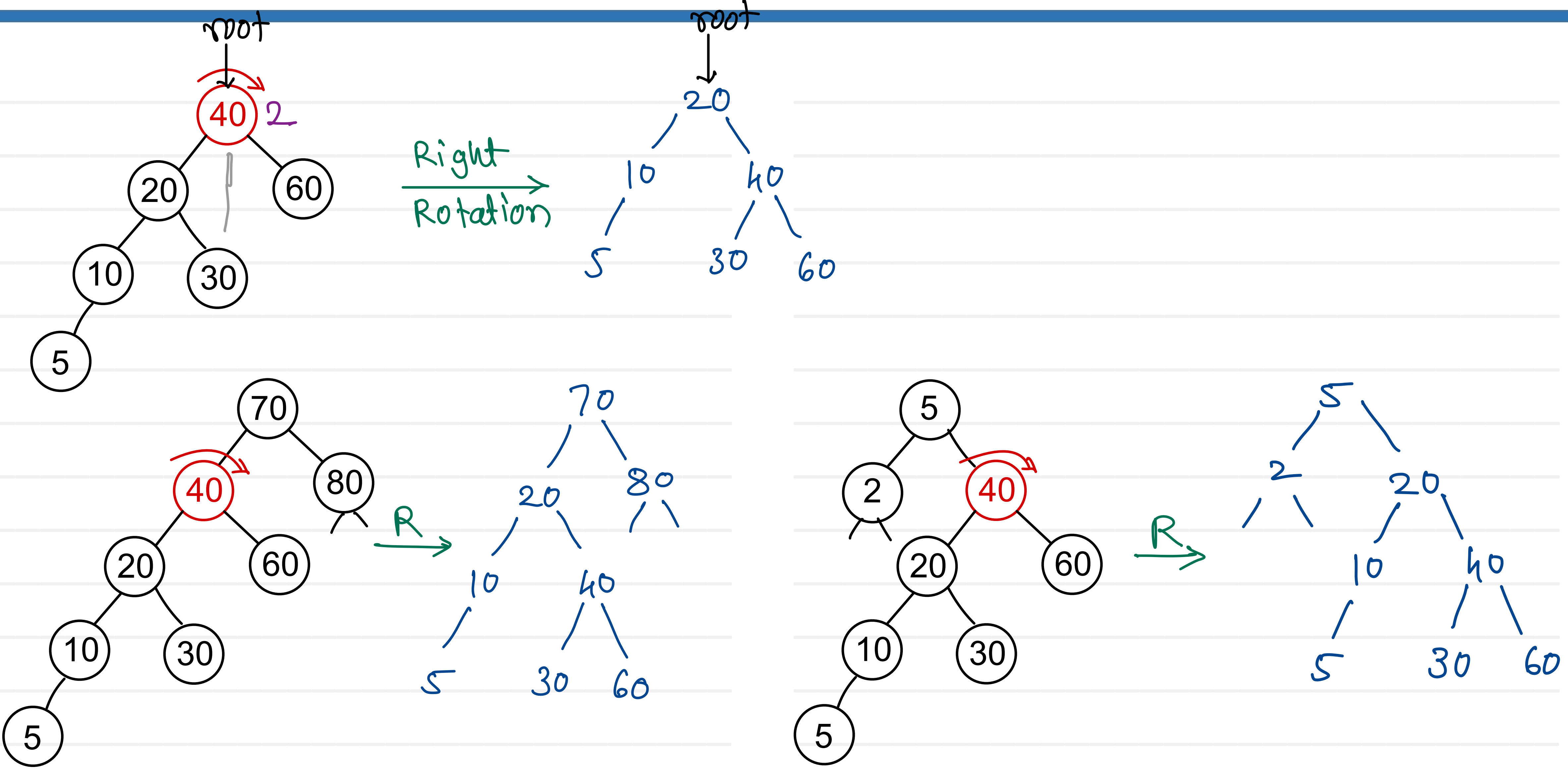
- To speed up searching, height of BST should be minimum as possible
- If nodes in BST are arranged, so that its height is kept as less as possible, is called as Balanced BST

Balance factor (node) = Height (left sub tree) - Height (right sub tree)

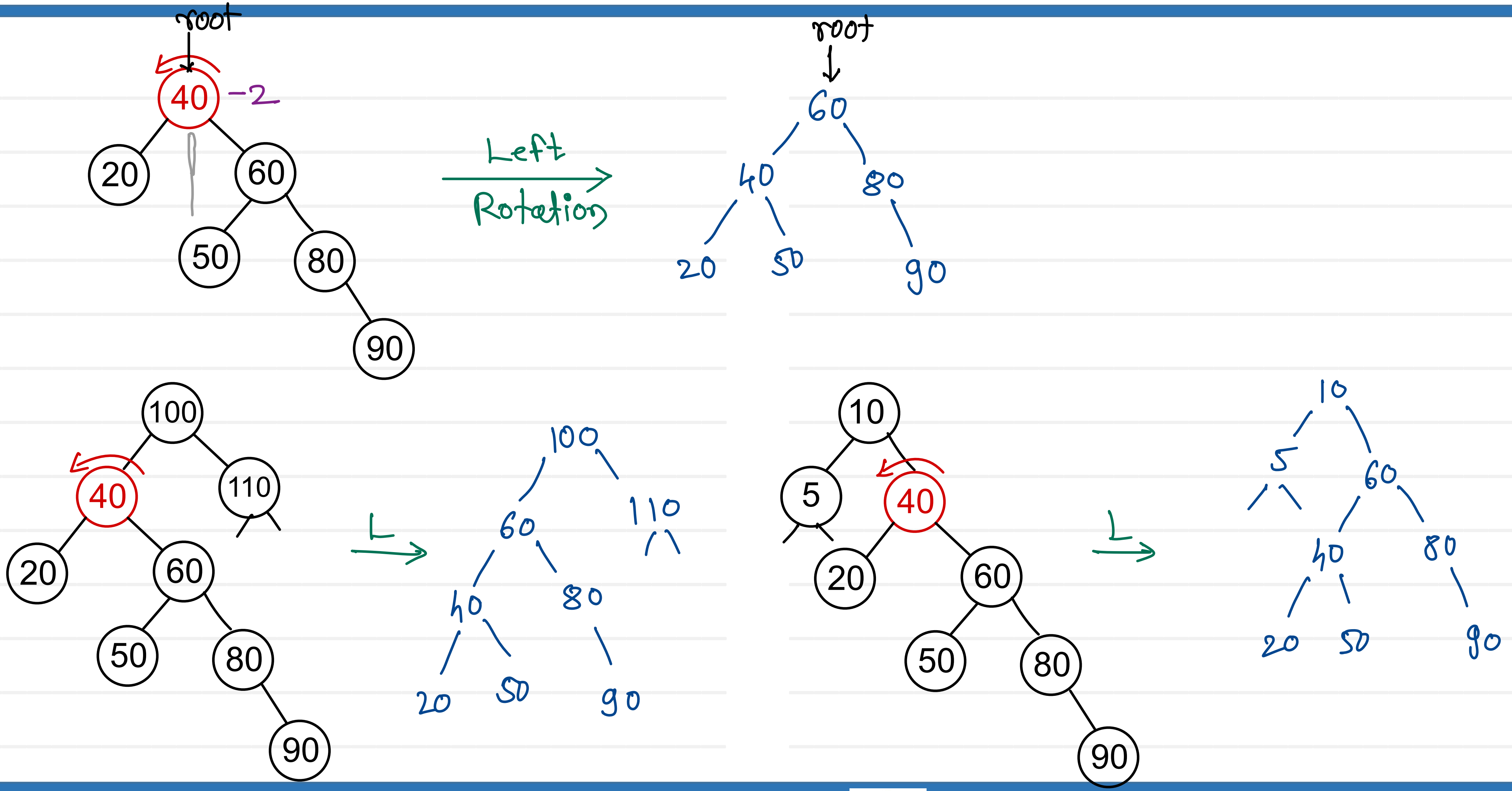
- tree is balanced if balance factors of all the nodes is either -1, 0 or +1
- balance factors = {-1, 0, +1}
- A tree can be balanced by applying series of left or right rotations on imbalance nodes (nodes having balance factor other than -1, 0, +1)



Right Rotation



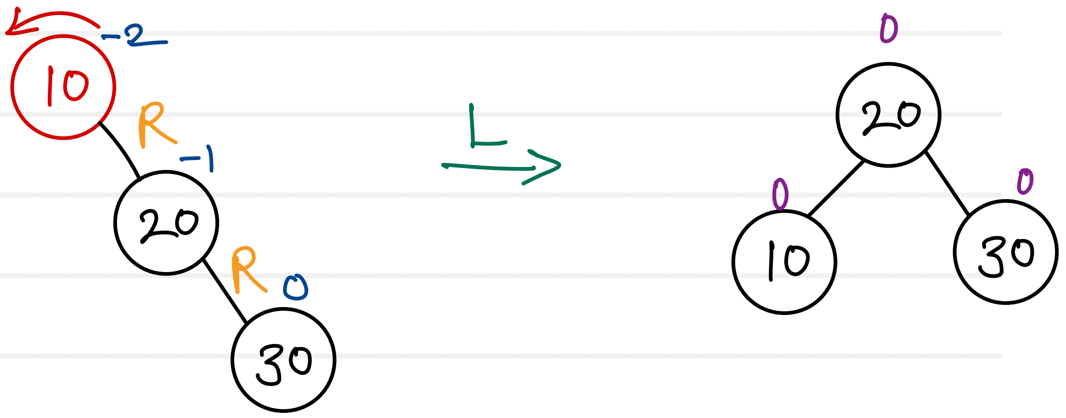
Left Rotation



Rotation cases (Single Rotation)

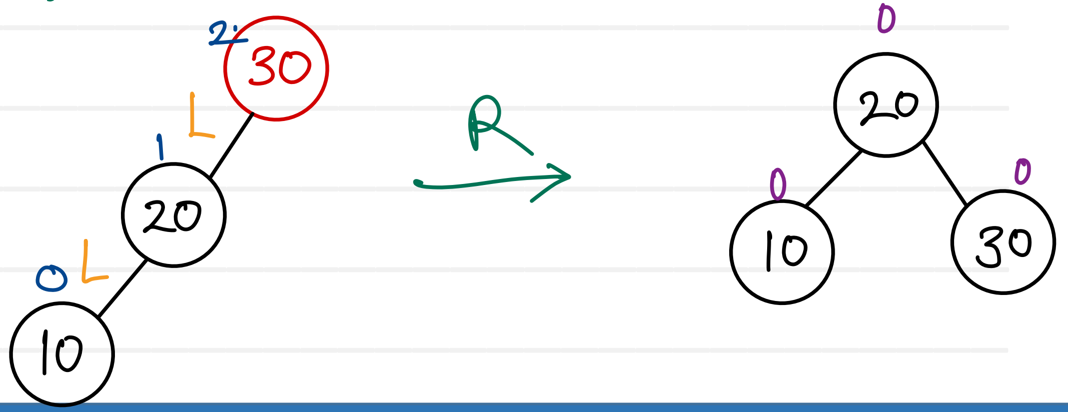
RR Imbalance

Keys : 10, 20, 30



LL Imbalance

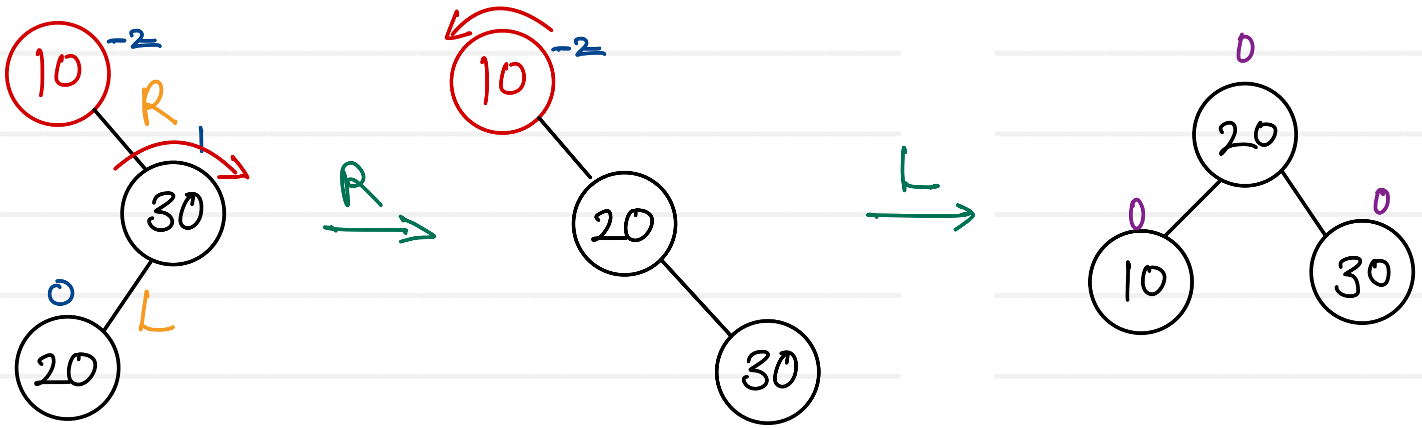
Keys : 30, 20, 10



Rotation cases (Double Rotation)

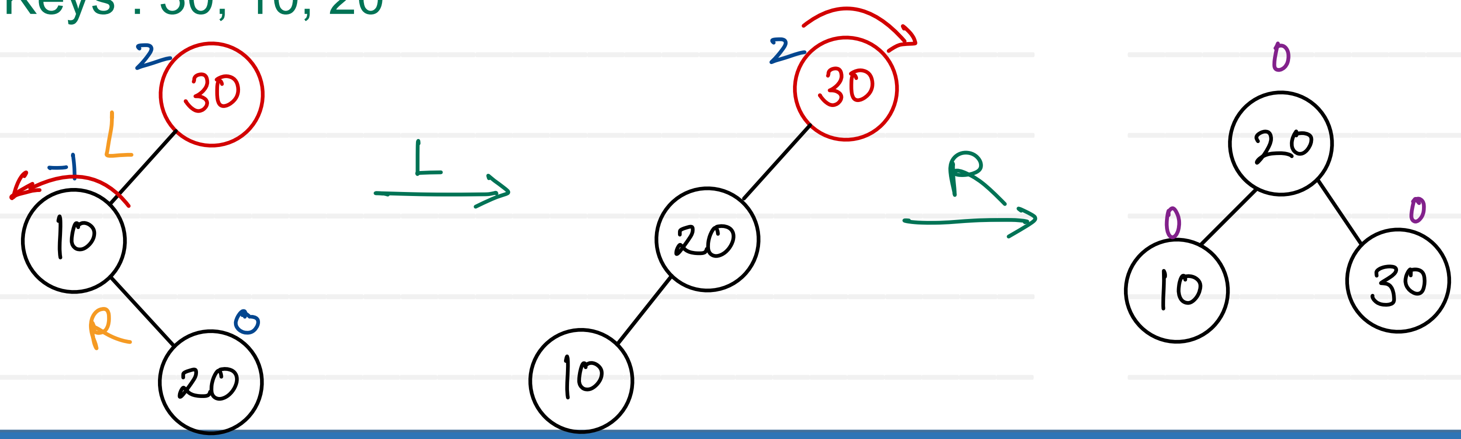
RL Imbalance

Keys : 10, 30, 20



LR Imbalance

Keys : 30, 10, 20



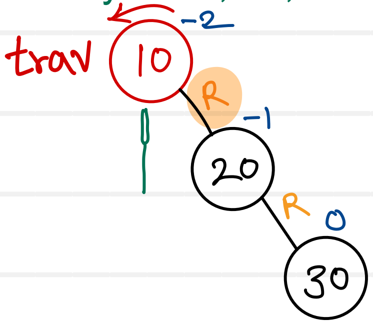
$bf = \{-1, 0, +1\}$

$bf < -1$

$bf > +1$

RR Imbalance

Keys : 10, 20, 30

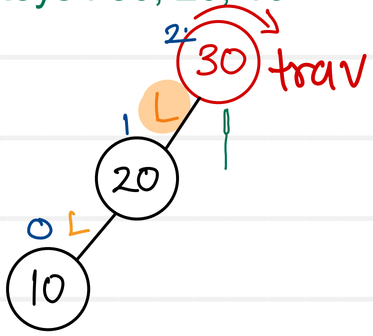


value > trav.right.data

(30 > 20)

LL Imbalance

Keys : 30, 20, 10

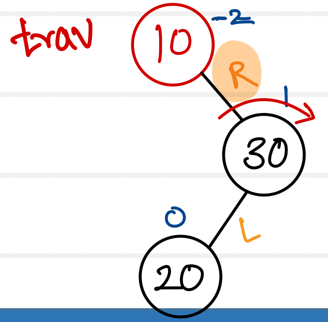


value < trav.left.data

(10 < 20)

RL Imbalance

Keys : 10, 30, 20

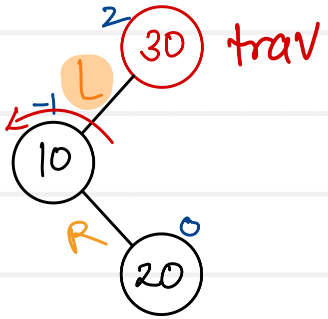


value < trav.right.data

(20 < 30)

LR Imbalance

Keys : 30, 10, 20



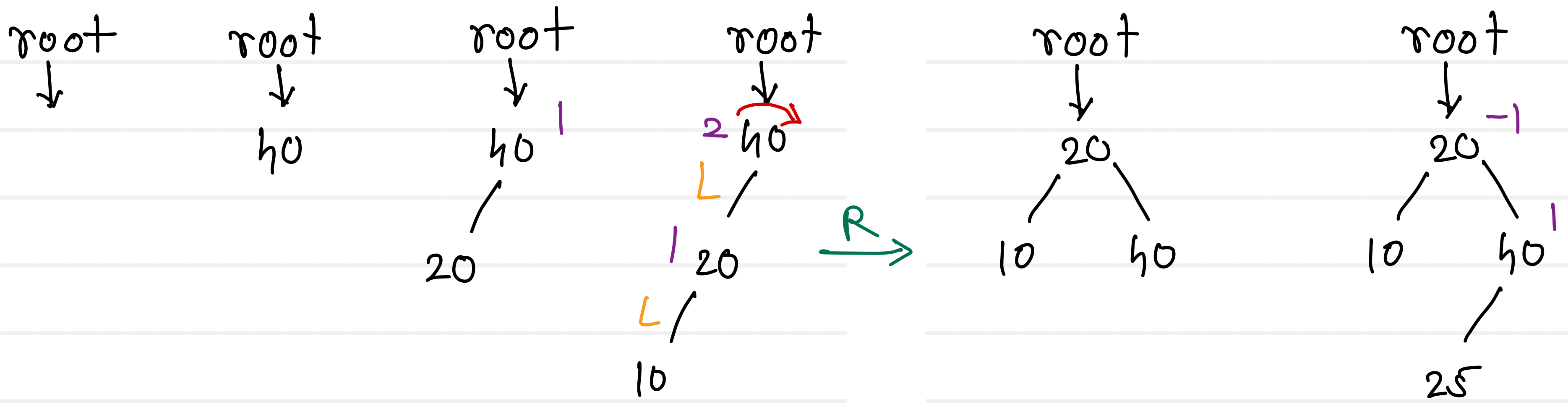
value > trav.left.data

(20 > 10)

AVL Tree

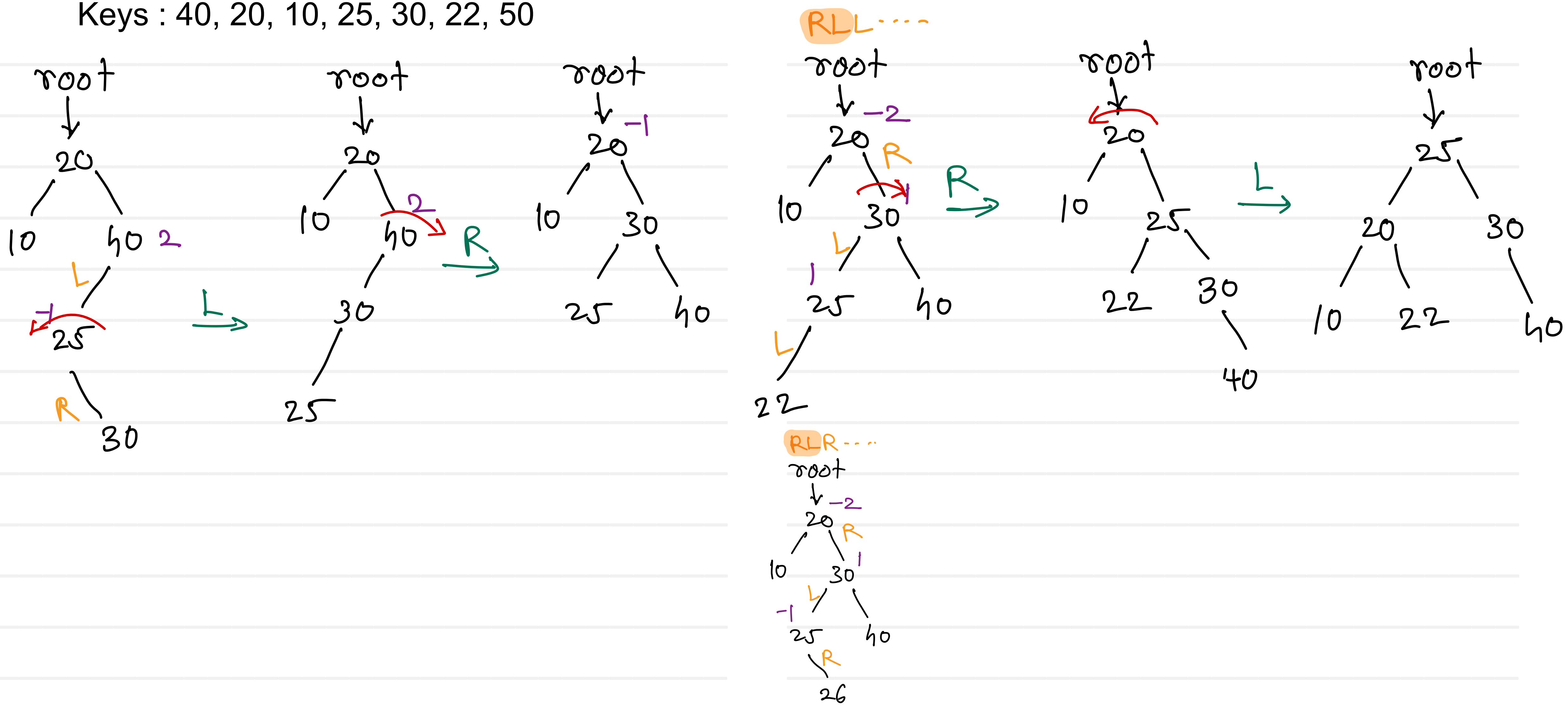
- self balancing binary search tree
- on every insertion and deletion of a node, tree is getting balanced by applying rotations on imbalance nodes
- The difference between heights of left and right sub trees can not be more than one for all nodes
- Balance factors of all the nodes are either -1 , 0 or +1
- All operations of AVL tree are performed in O(log n) time complexity

Keys : 40, 20, 10, 25, 30, 22, 50

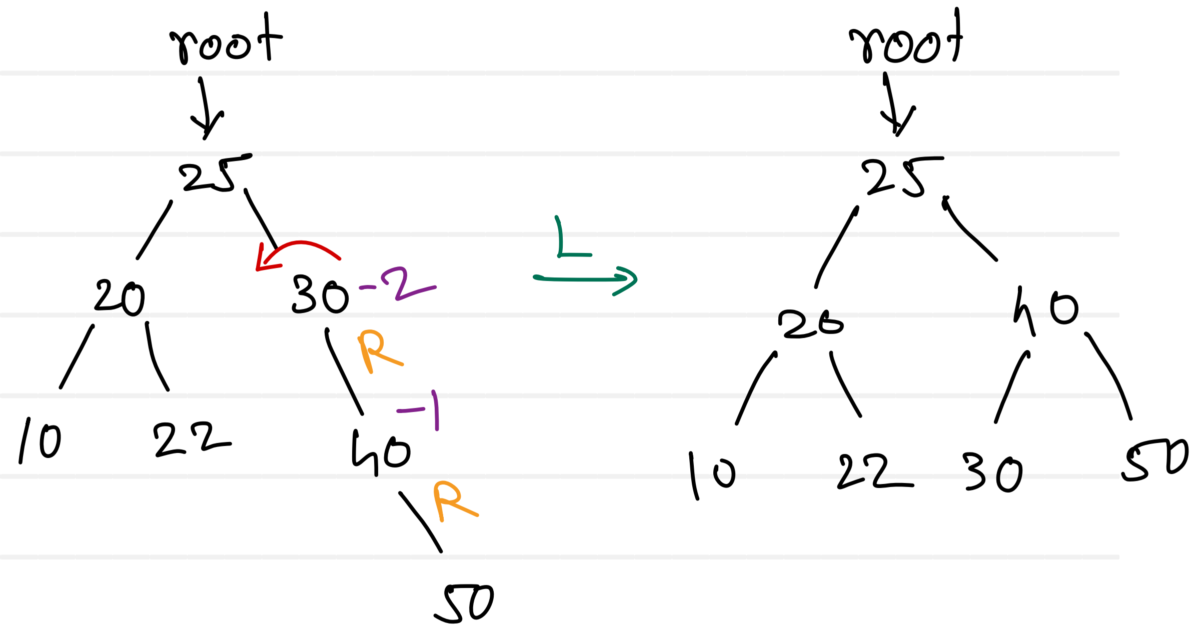


AVL Tree

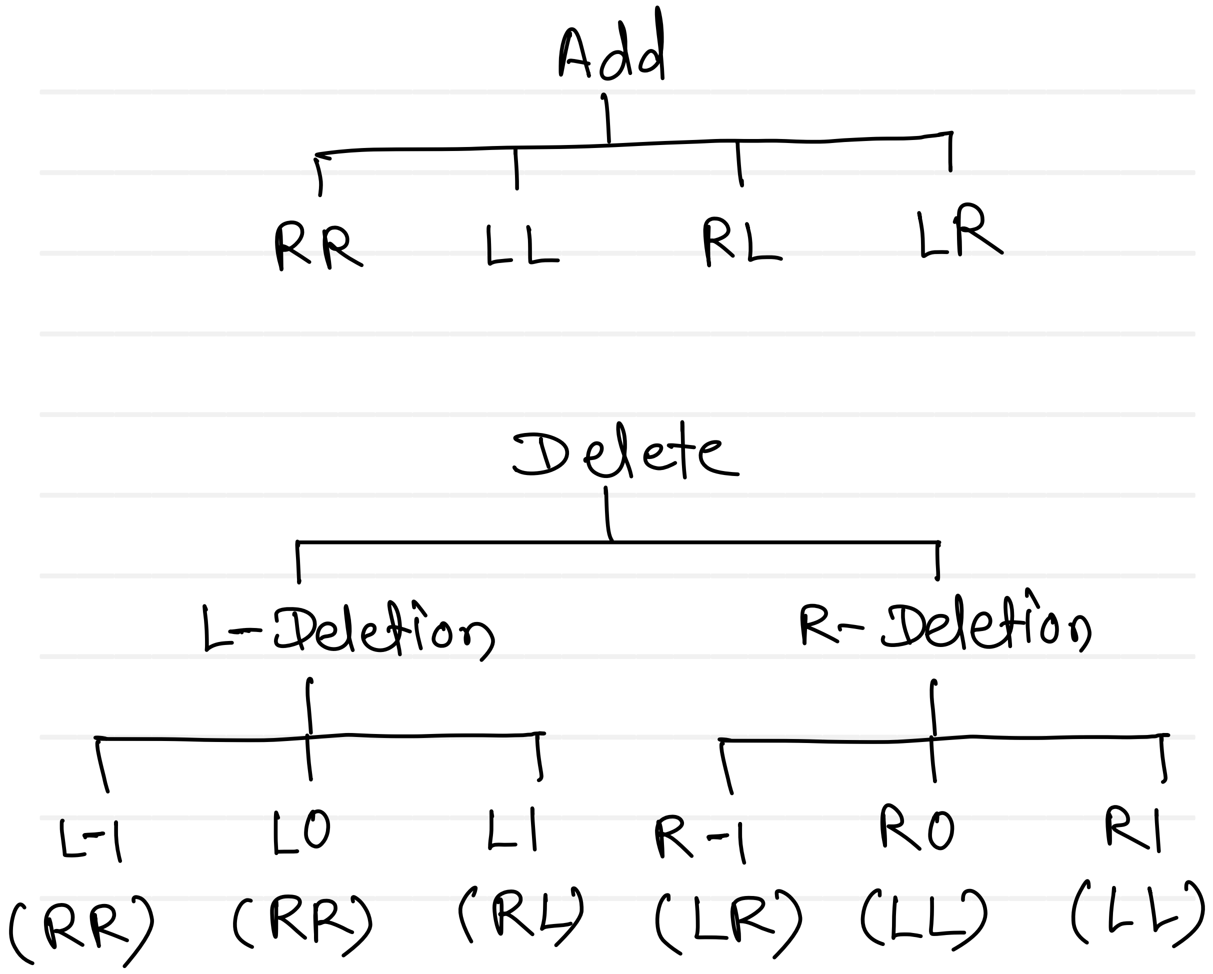
Keys : 40, 20, 10, 25, 30, 22, 50

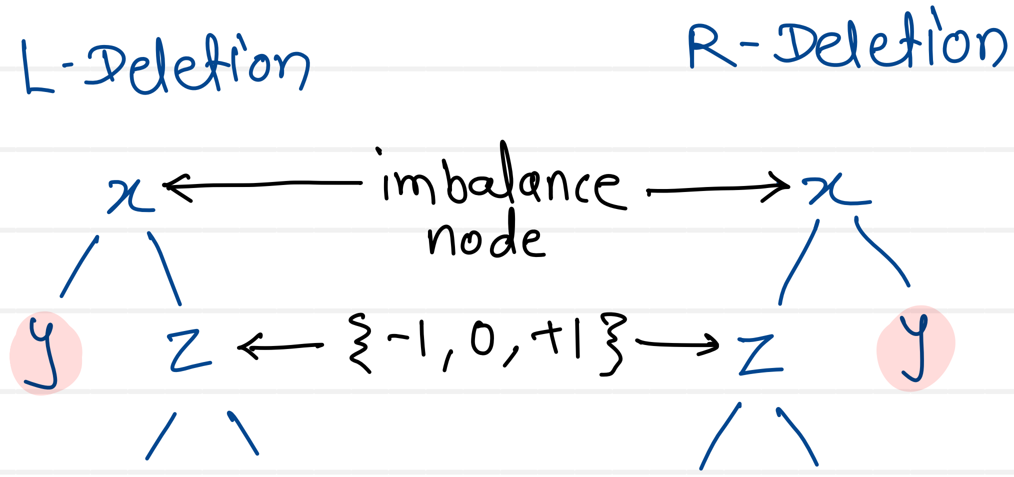


Keys : 40, 20, 10, 25, 30, 22, 50

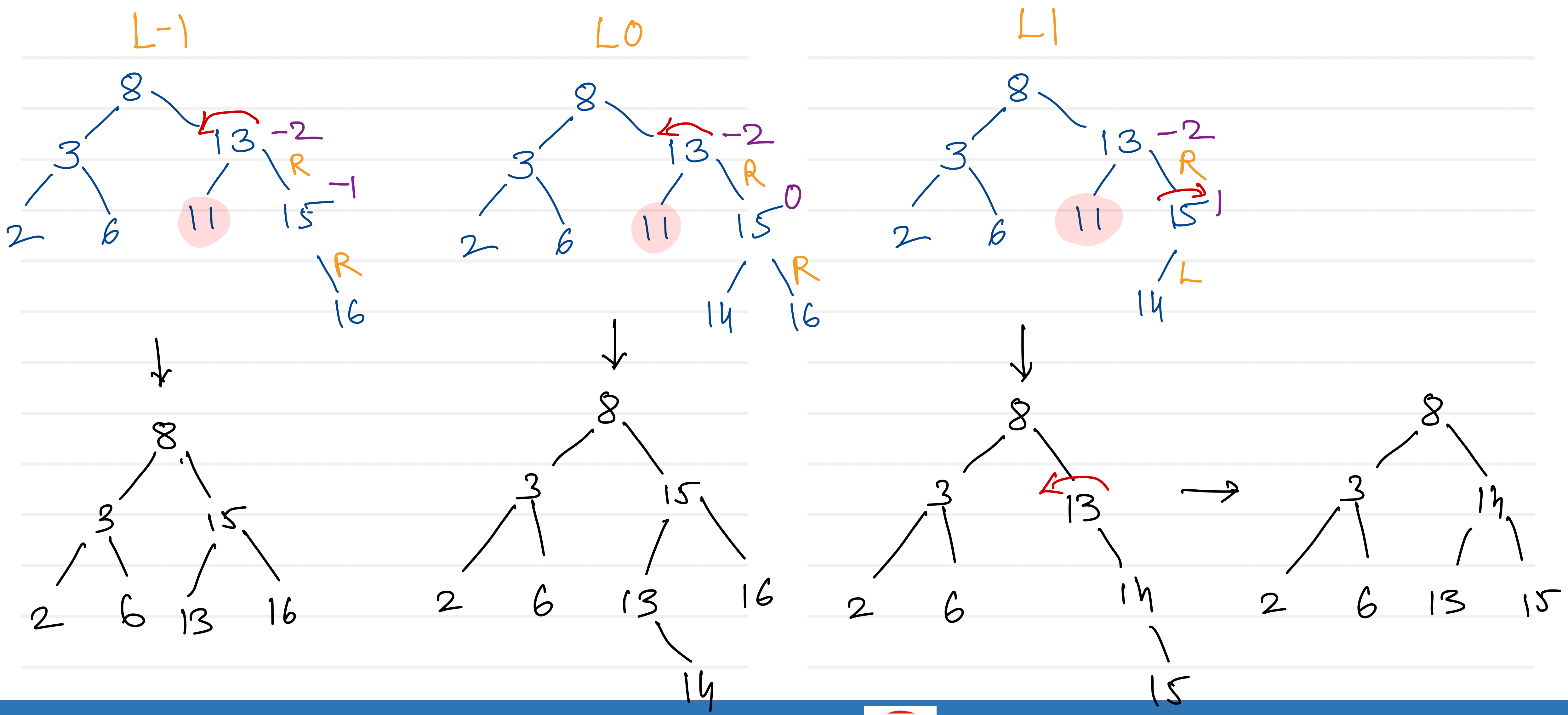


AVL Tree


Add
RR
LL
RL
LR
Delete
L-Deletion
R-Deletion
L-1
L0
L1
R-1
R0
R1
(RR)
(RR)
(RL)
(LR)
(LL)
(LL)

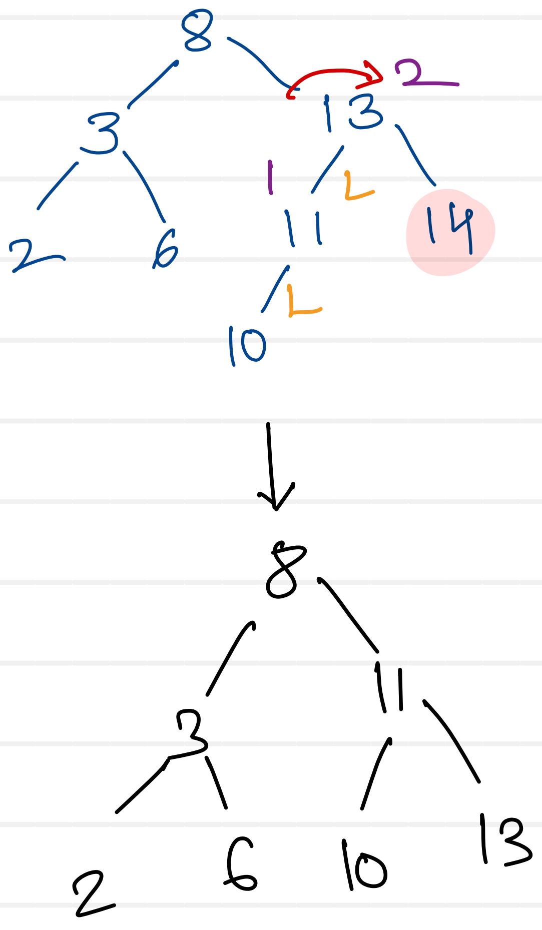

L-Deletion
R-Deletion
x
imbalance node
x
y
z
{-1, 0, +1}
z
y

AVL Tree (L- Deletion)

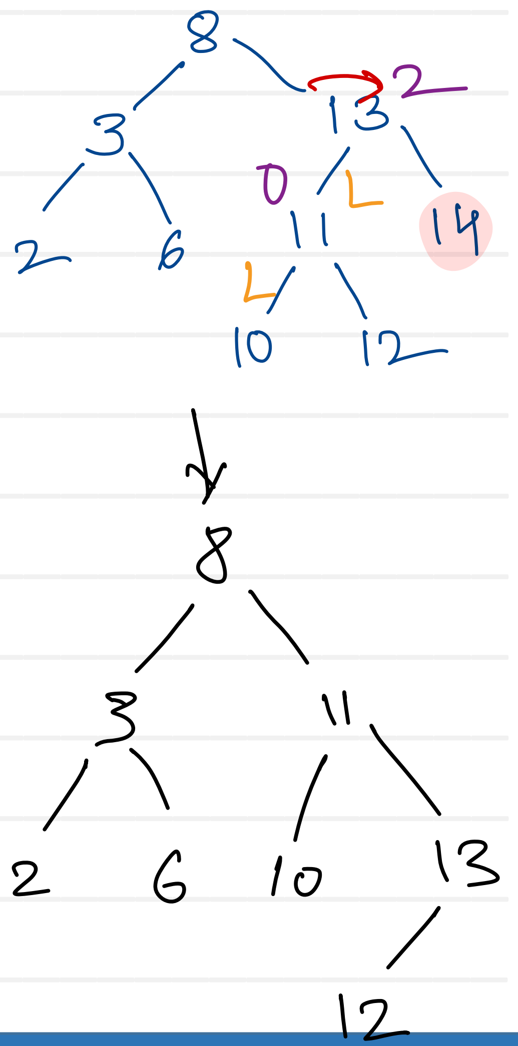


AVL Tree (R Deletion)

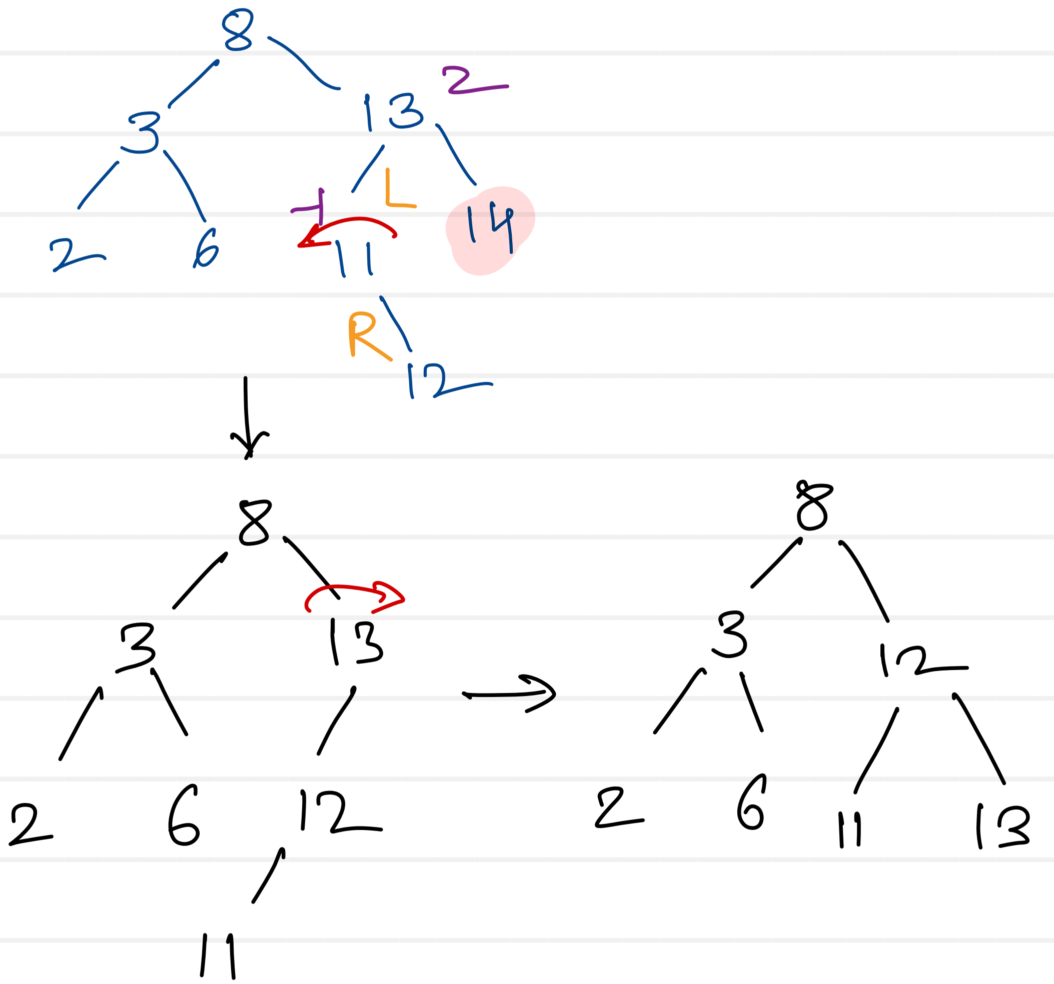
(R1)



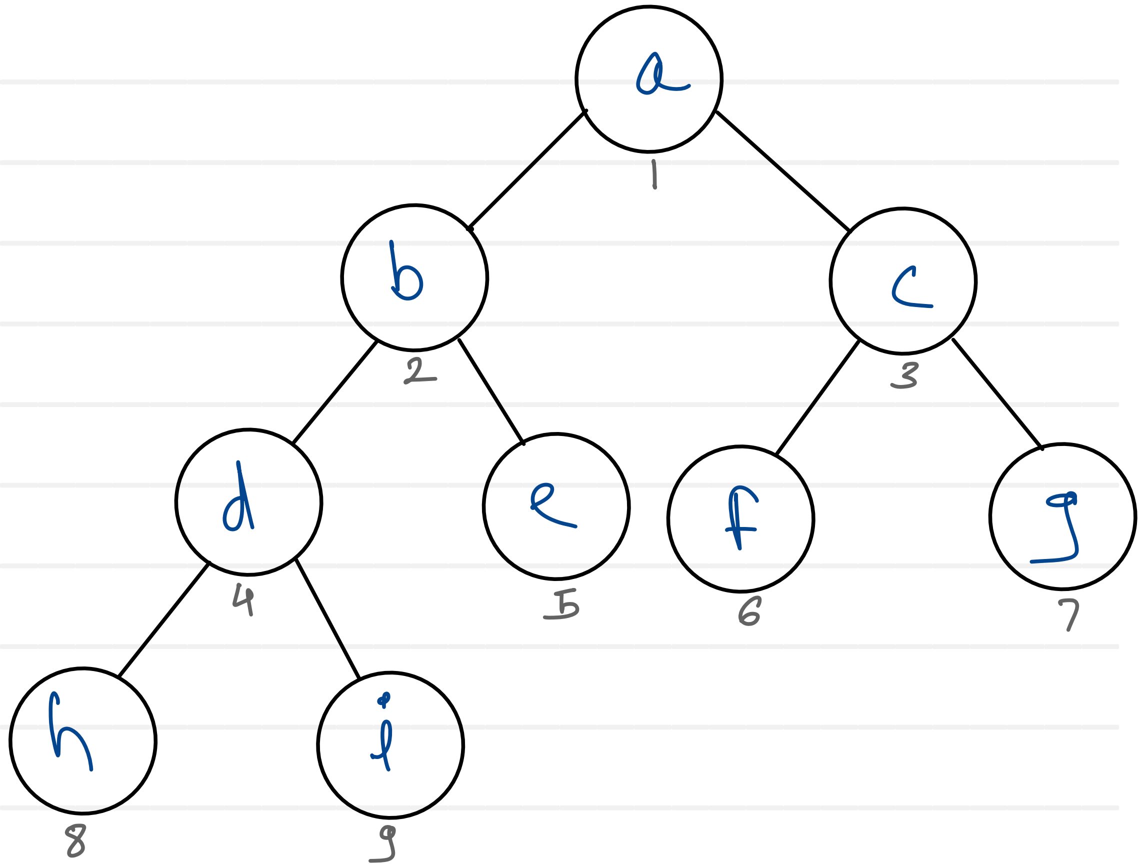
(R0)



(R-1)



Complete Binary Tree or Heap



a	b	c	d	e	f	g	h	i
1	2	3	4	5	6	7	8	9

- Complete Binary Tree (height = h)
- All levels should be completely filled except last
- All leaf nodes must be at level h or h-1
- All leaf nodes at level h must aligned as left as possible

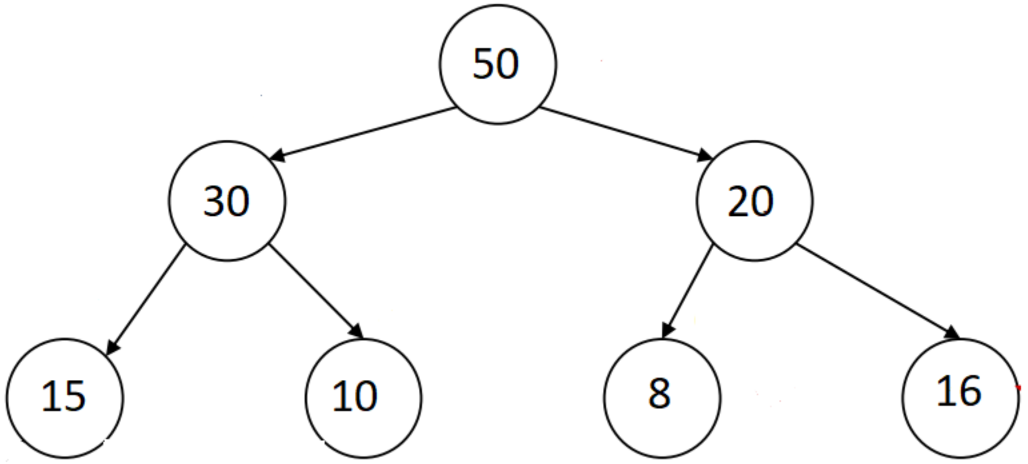
- Array implementation of Complete Binary Tree is called as heap

- Parent child relationship is maintained with the help of array indices

Node - i^{th} index
Parent - $i/2$ index
left child - $i*2$ index
right child - $i*2+1$ index

Heap Types – Max and Min

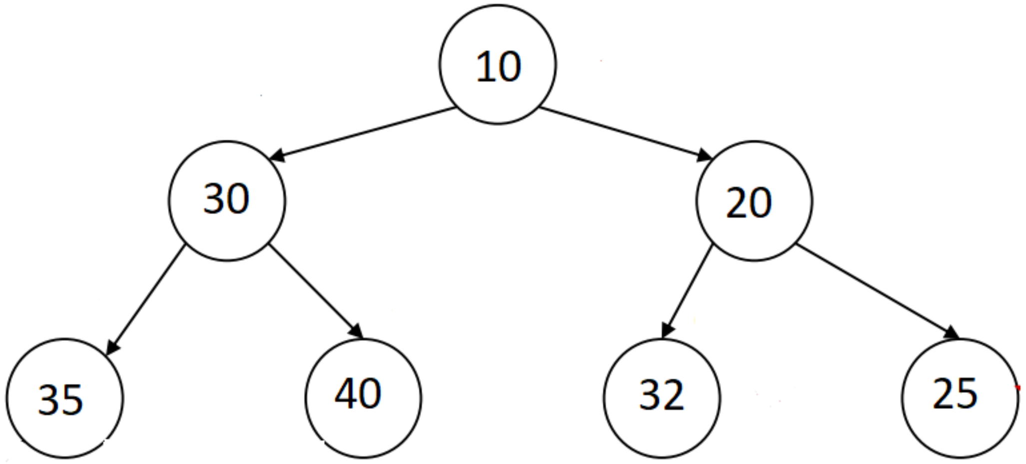
Max Heap



50	**30**	**20**	**15**	**10**	**8**	**16**
1	2	3	4	5	6	7

- Max heap is a heap data structure in which each node is greater than both of its child nodes.

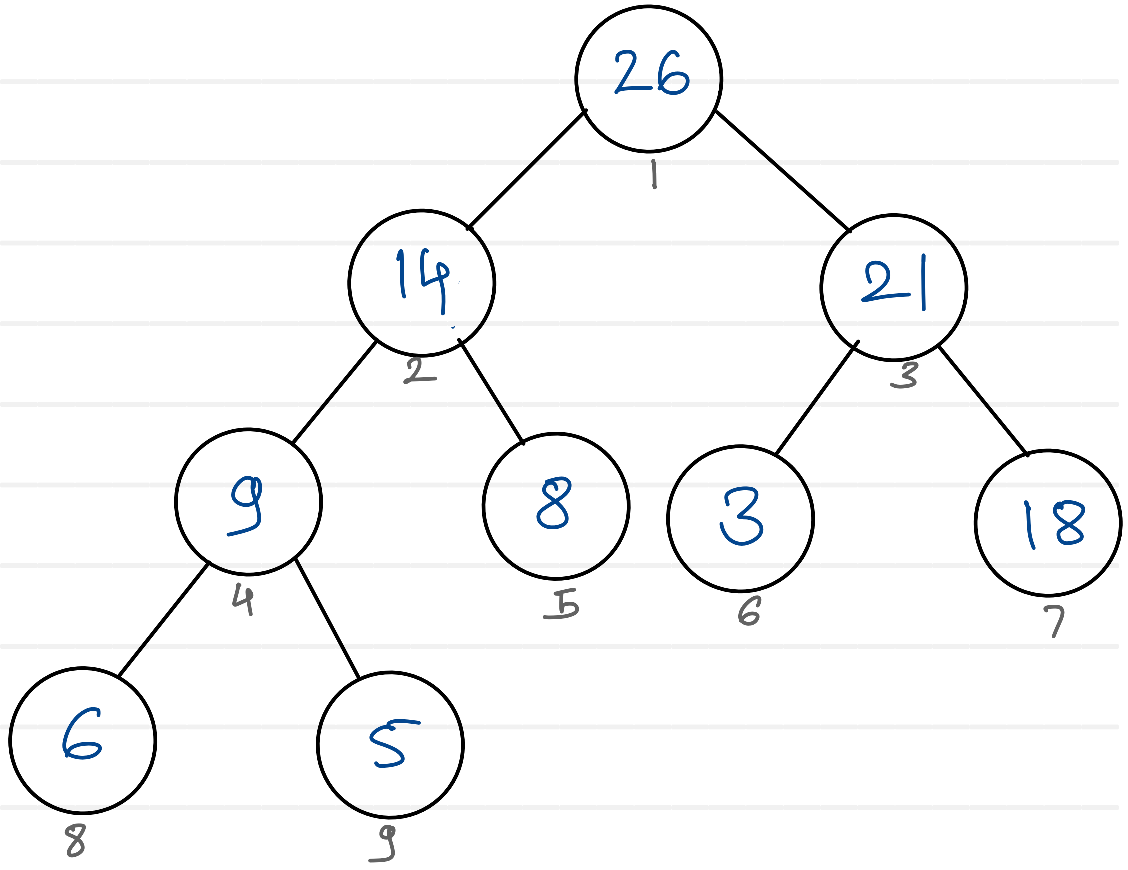
Min Heap



10	**30**	**20**	**35**	**40**	**32**	**25**
1	2	3	4	5	6	7

- Min heap is a heap data structure in which each node is smaller than both of its child nodes.

Heap - Create heap (Add)



26	14	21	9	8	3	18	6	5
1	2	3	4	5	6	7	8	9

Keys : 6, 14, 3, 26, 8, 18, 21, 9, 5

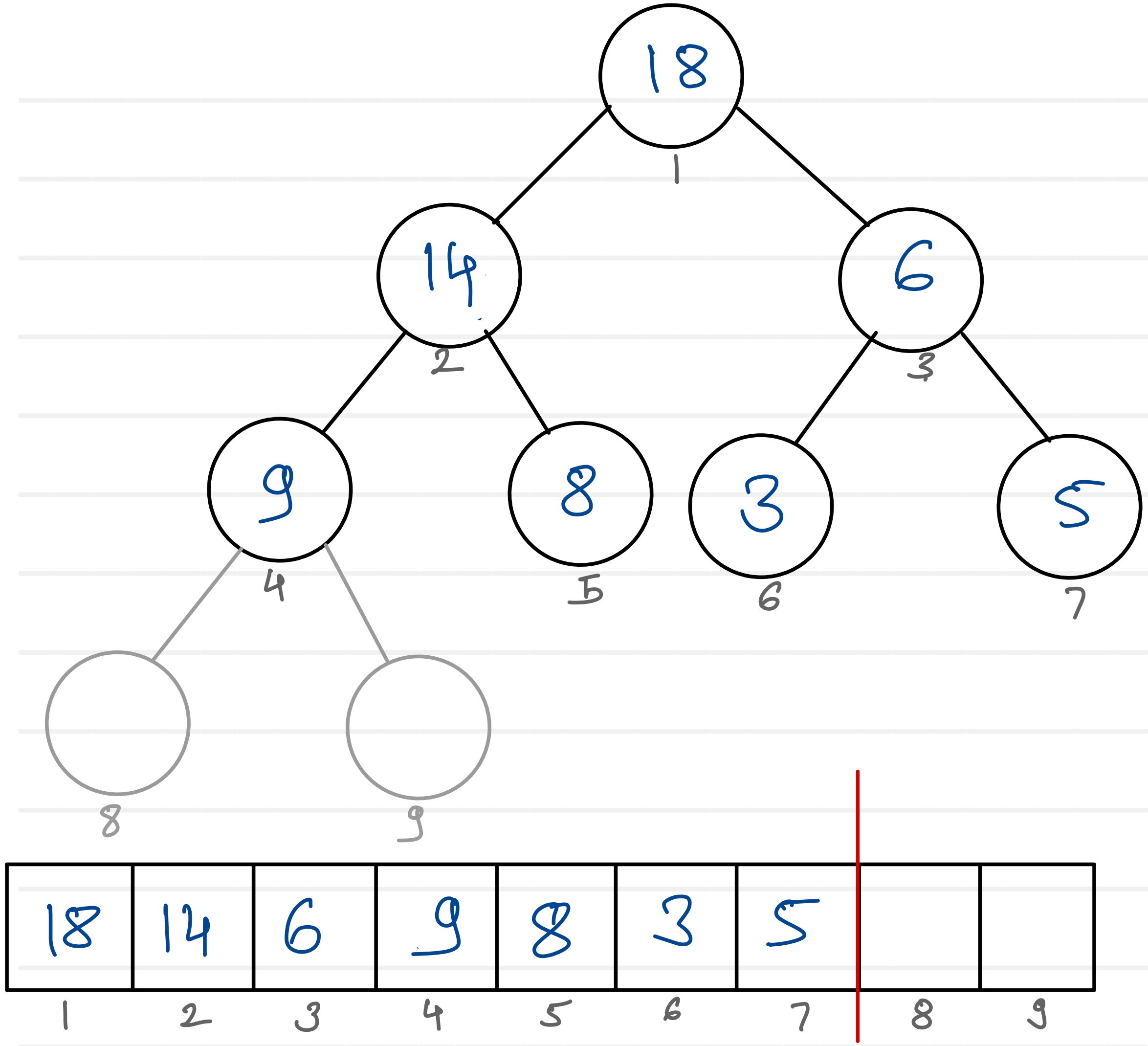
i. add new value on first empty index from left side

ii. adjust position of newly added value by comparing it with all its ancestors.

- to add value into heap, need to traverse from leaf to root position

$\boxed{T(n) = O(\log n)}$

Heap - Delete heap (Delete)



Property : can delete only root node from heap

1. in max heap, always maximum element will be deleted from heap.
2. in min heap, always minimum element will be deleted from heap.

max = 26

max = 21

i. place last element of heap on root position

ii. adjust position of root by comparing it with all its descendents.

- to adjust position need to traverse from root to leaf positions.

$$\boxed{T(n) = O(\log n)}$$

pi ci

30	26	21	14	8	3	18	6	9
1	2	3	4	5	6	7	8	9

child index	parent index
9	4
4	2
2	1
1	0

ci = size

pi ci

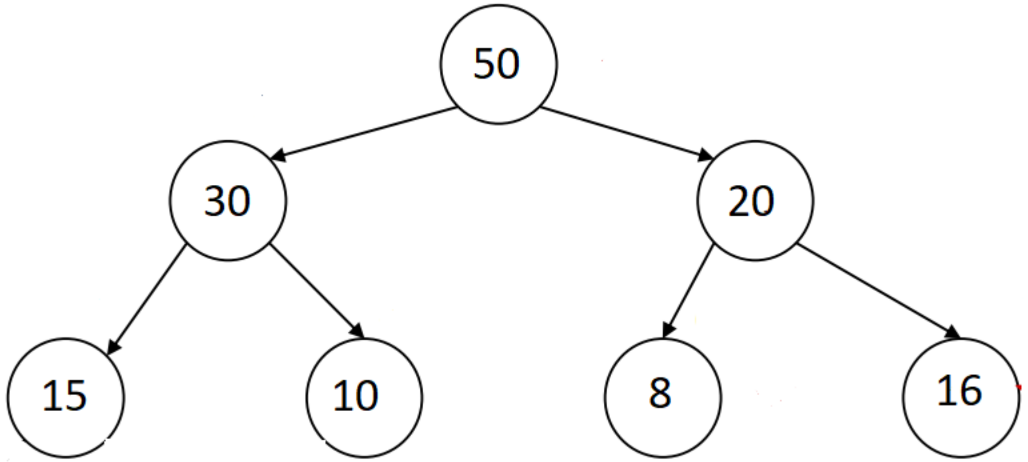
26	14	21	9	8	3	18	6	
1	2	3	4	5	6	7	8	9

parent index	child index
1	2, 3
2	4, 5
4	8, X

pi = 1

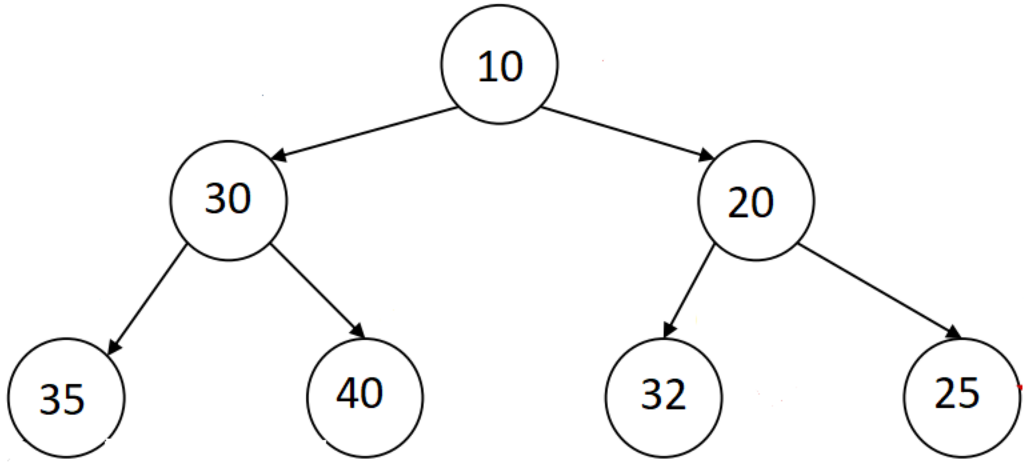
Priority Queues

Higher number
Higher priority



- In Max heap always root element which has highest value is removed

Lower number
Higher priority



- In Min heap always root element which has lowest value is removed

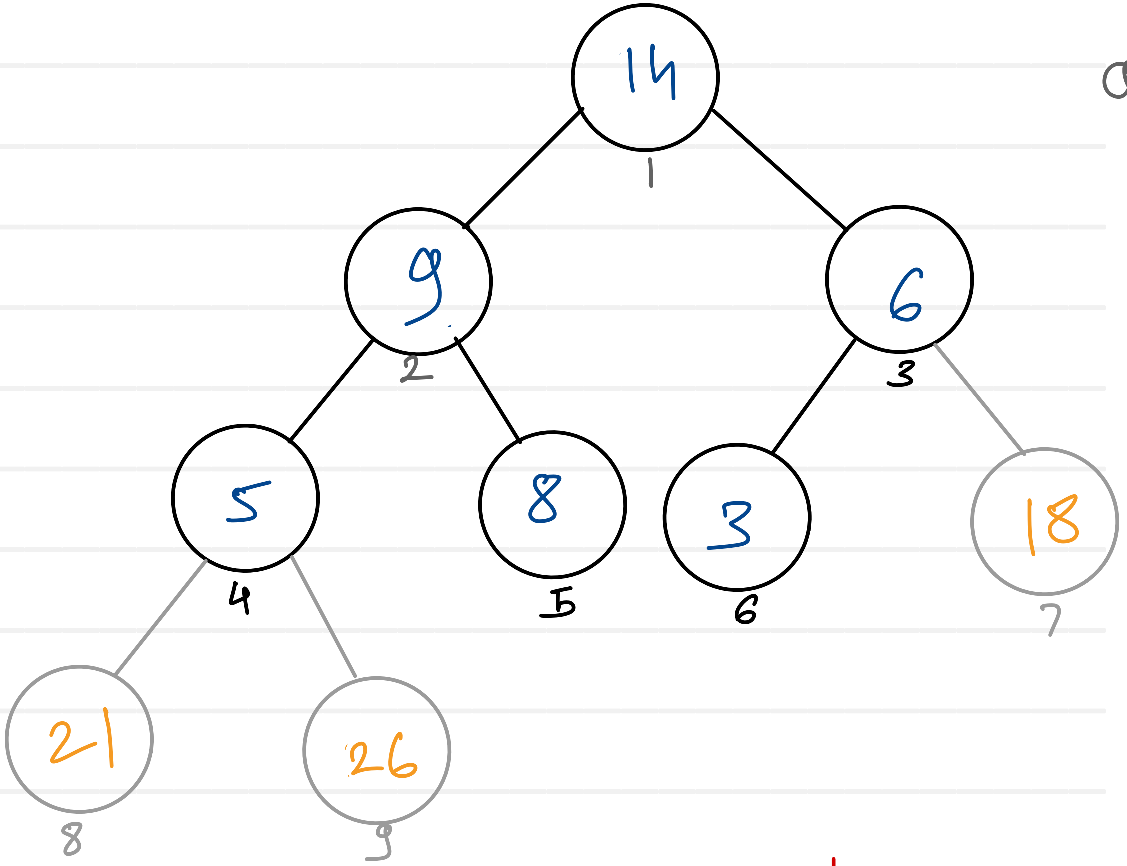
Priority Queue

- Always high priority element is deleted from queue
- value (priority) is assigned to each element of queue
- priority queue can be implemented using array or linked list.
- to search high priority data (element) need to traverse array or linked list
- Time complexity = $O(n)$

- priority queque can also be implemented using heap. because, maximum / minimum value is kept at root position in max heap & min heap respectively.
- push, pop & peek will be performed efficiently

max value → high priority → max heap
min value → high priority → min heap

Heap sort



arr

6	14	3	26	8	18	21	9	5
1	2	3	4	5	6	7	8	9

1. create heap (max/min) from given array.
2. delete all elements from heap one by one and place them on vacant locations from right side

create heap = n log n
delete heap = n log n

$2n\log n$

Time $\propto 2n\log n$

$\boxed{T(n) = O(n \log n)}$ Best Avg Worst

$\boxed{S(n) = O(1)}$

arr

14	9	6	5	8	3	18	21	26
1	2	3	4	5	6	7	8	9

SUNBEAM

Thank you!!!

Devendra Dhande

devendra.dhande@sunbeaminfo.com