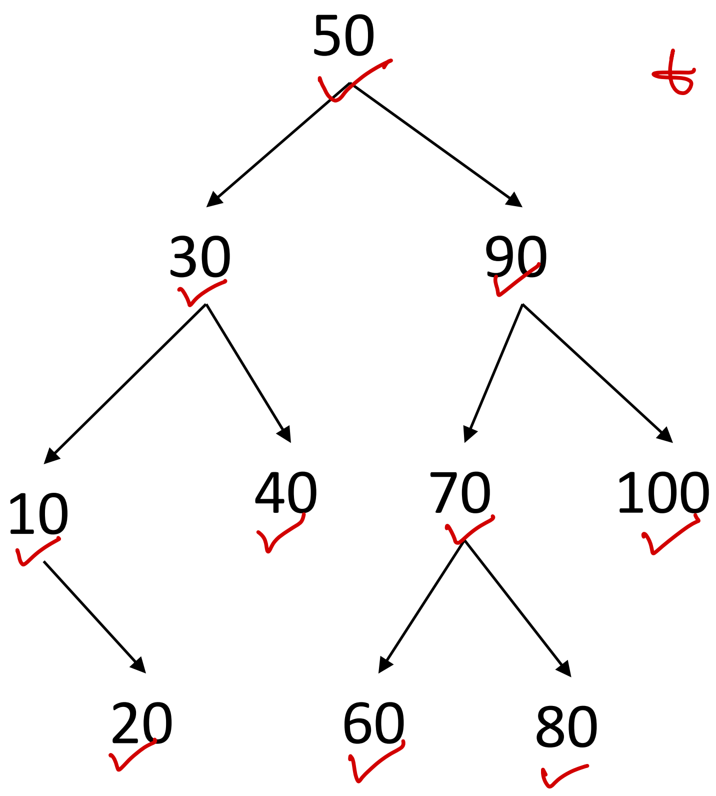
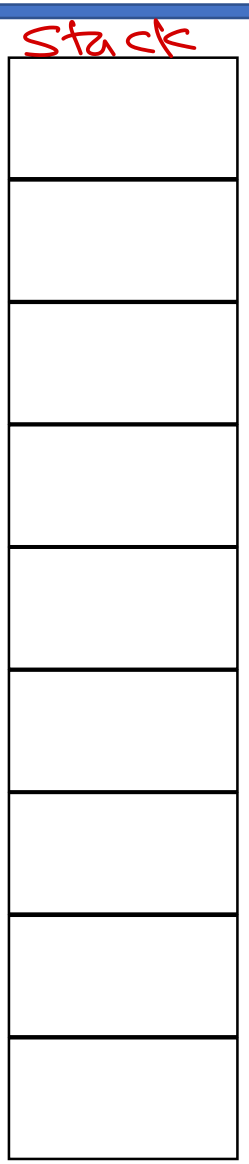
Data Structure & Algorithms

Sunbeam Infotech

BST - post order ( )






```
trav = root;
while(trav != null || !s.isEmpty()){
    while(trav != null){
        s.push(trav);
        trav = trav.left;
    }
    if(!s.isEmpty()){
        trav = s.pop();
        if(trav.right != null &&
            !trav.right.visited){
            s.push(trav);
            trav = trav.right;
        }
        else{
            print(trav.data);
            trav.visited = true;
            trav = null;
        }
    }
}
```

20 10 40 30 60

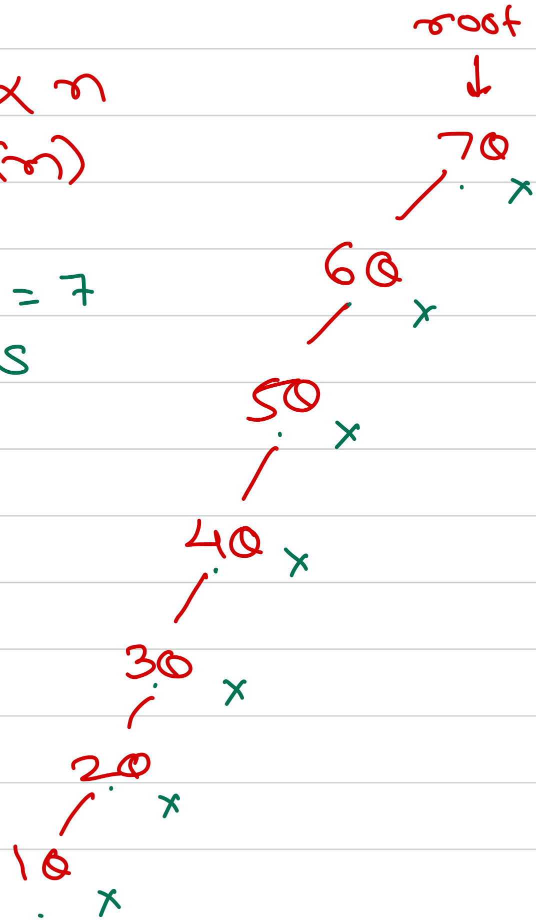
80 70 100 90 50

skewed BST → max height

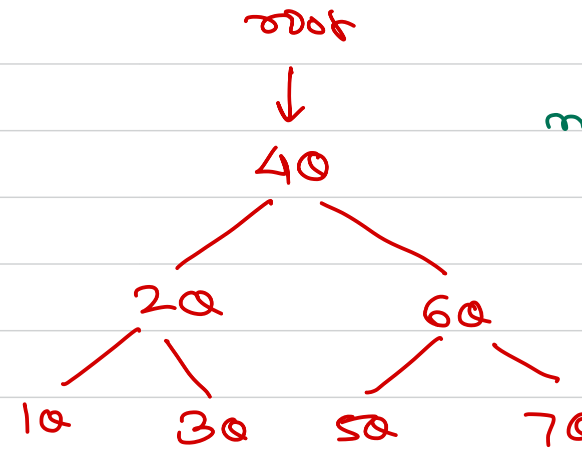
$T \propto n$

$O(n)$

max = 7 itrs

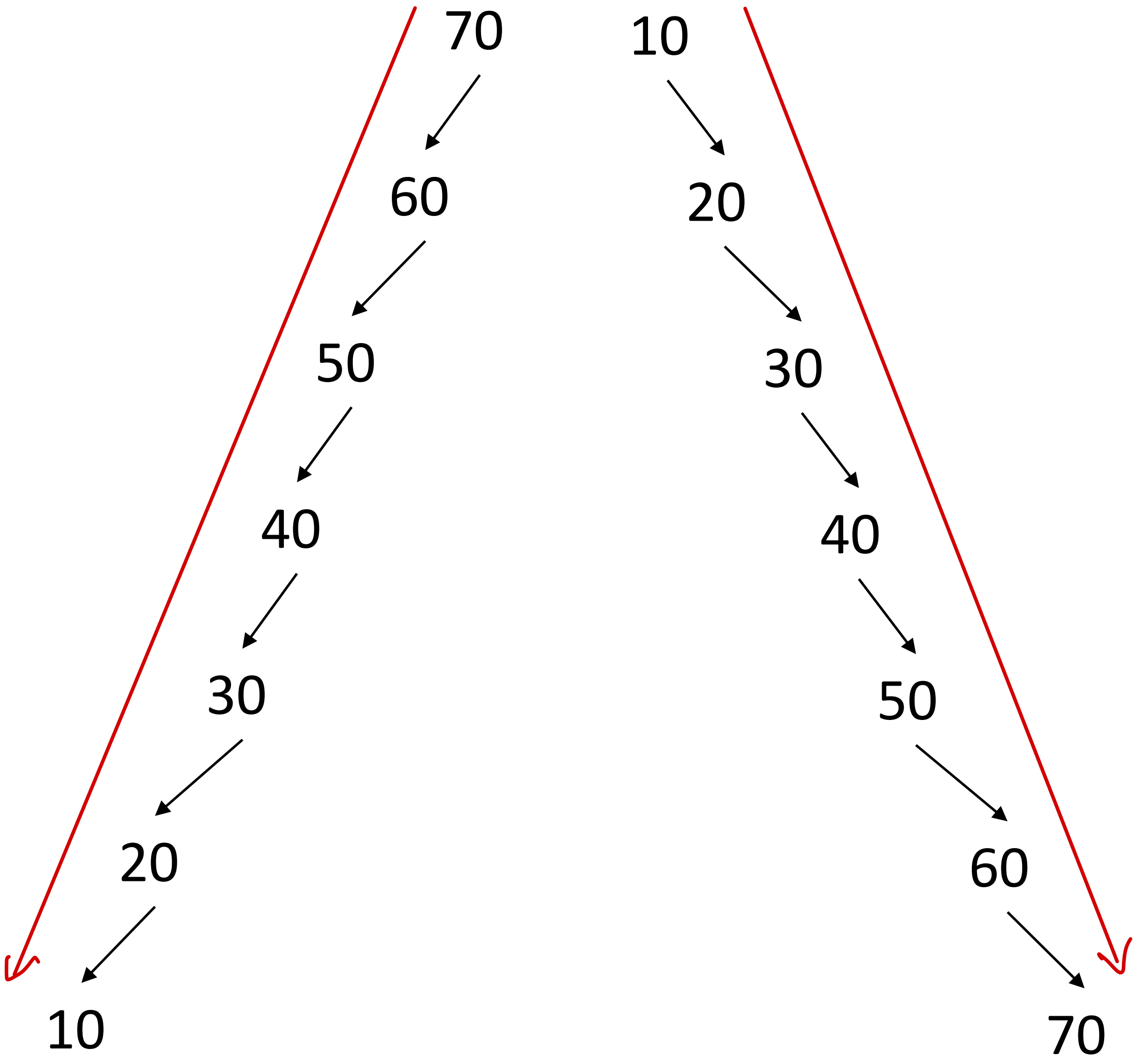


Balanced BST
min height.



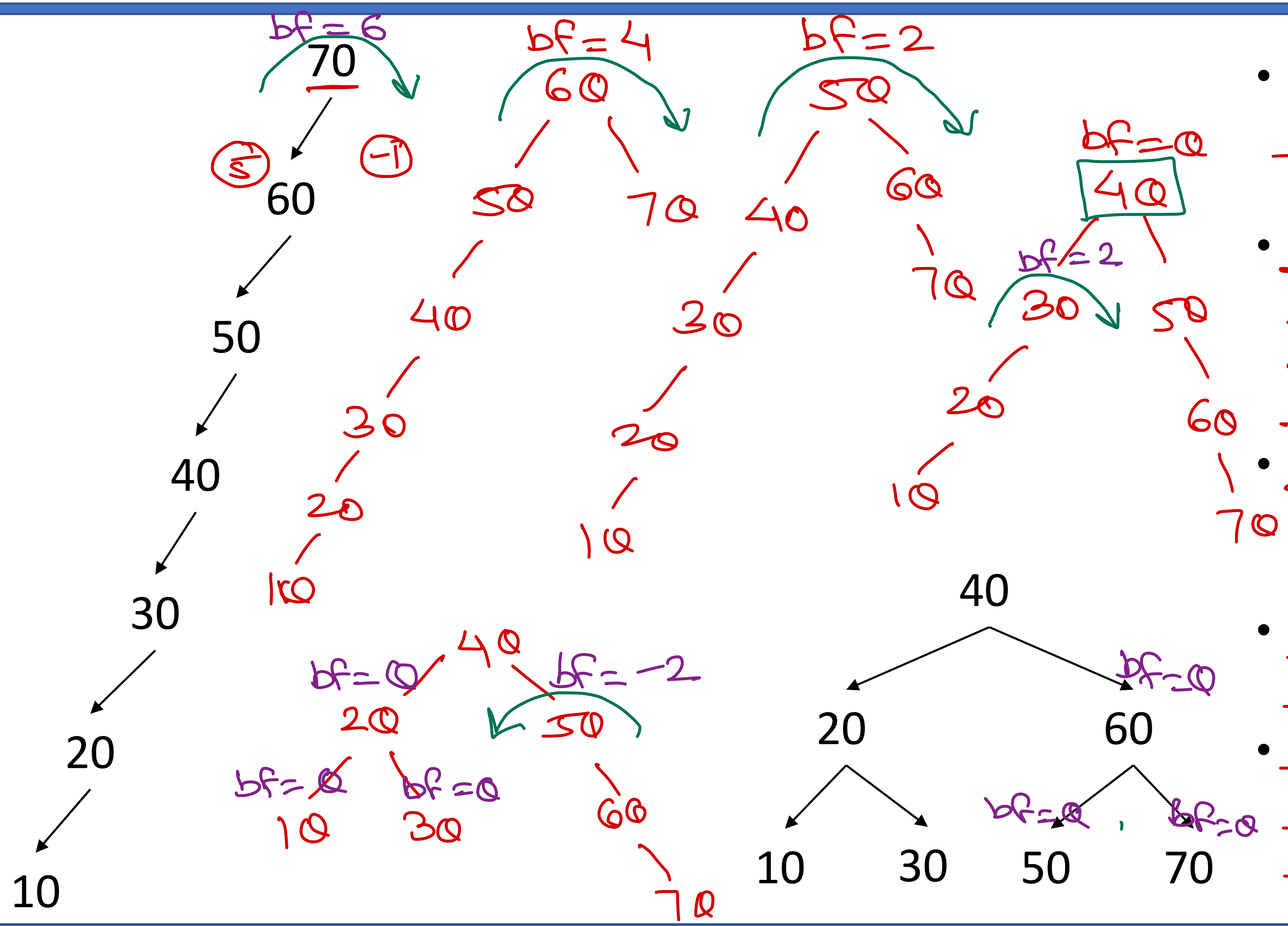
max = 3 itrs

Skewed Binary Tree



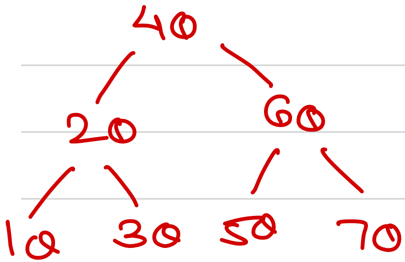
- In Binary tree if only left or only right links are used, tree grows only on one side. Such tree is called as skewed binary tree.
  - Left skewed binary tree
  - Right skewed binary tree
- Time complexity of any BST is O(h). height
- Such tree have maximum height i.e. same as number of elements.
- Time complexity of searching in skewed BST is O(n). → like linked list

Balanced BST



- To speed up searching, height of BST should 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
  - = Height of left sub tree – Height of ~~left~~ right sub tree
- In balanced BST, BF of each node is -1, 0 or +1.
- A tree can be balanced by applying series of left or right rotations on unbalanced nodes.

height = 2



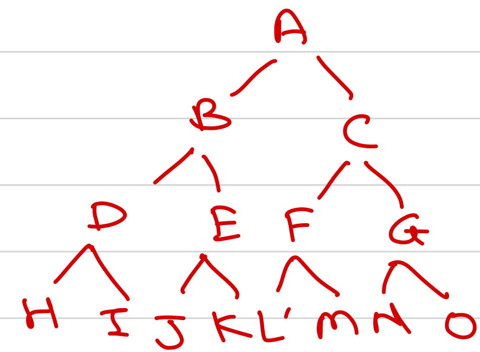
$n = 7$

$n = 2^{h+1} - 1$

leaf nodes (external) $= 2^h$

non-leaf nodes (internal) $= 2^h - 1$

height = 3

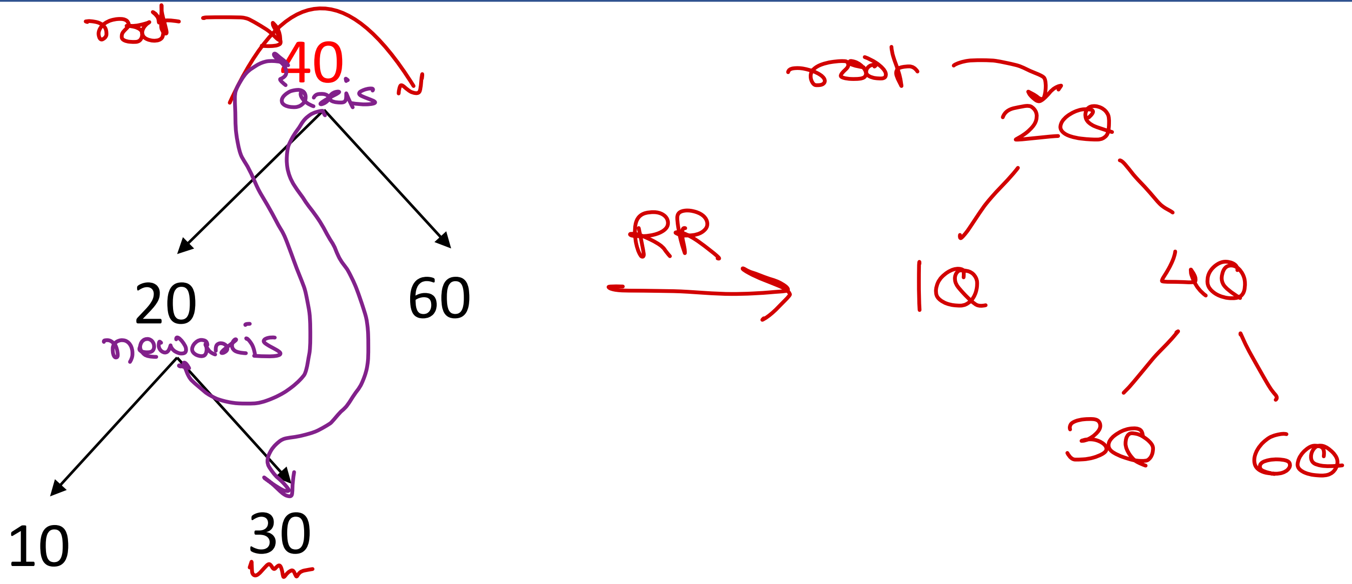


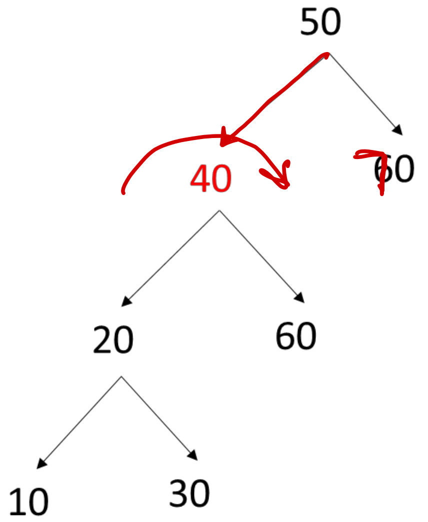
$n = 15$

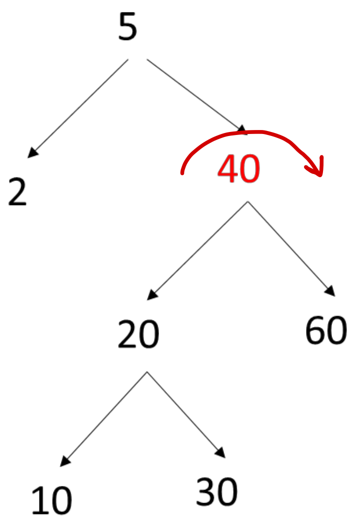
leaf $= 8$

non-leaf $= 7$

Right rotation

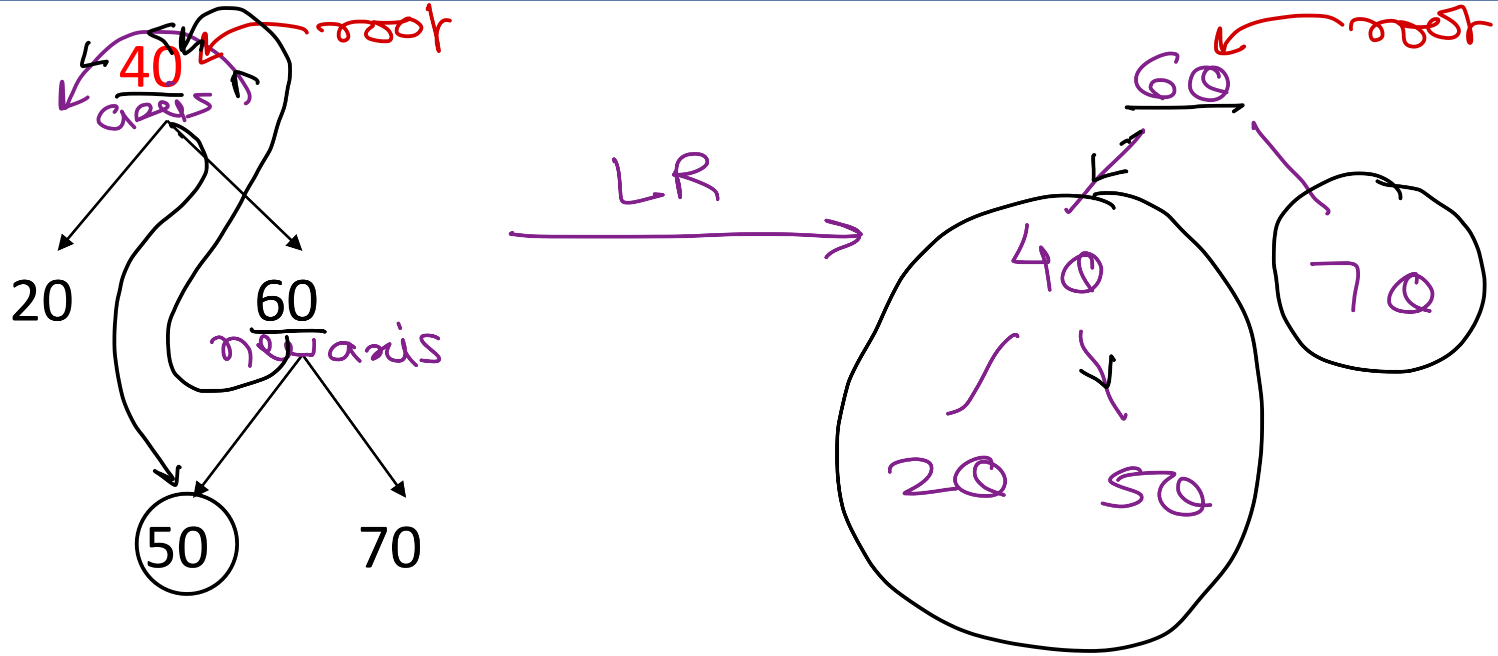


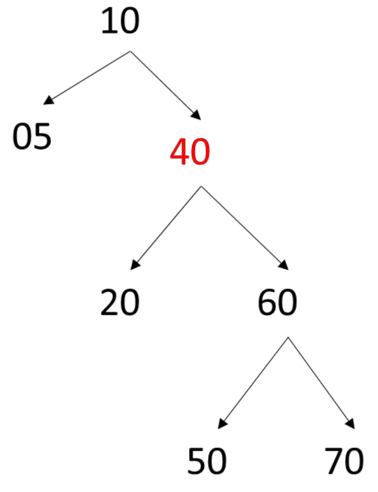


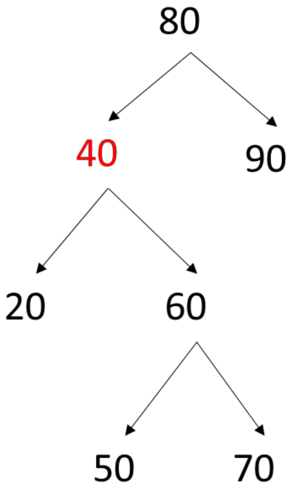


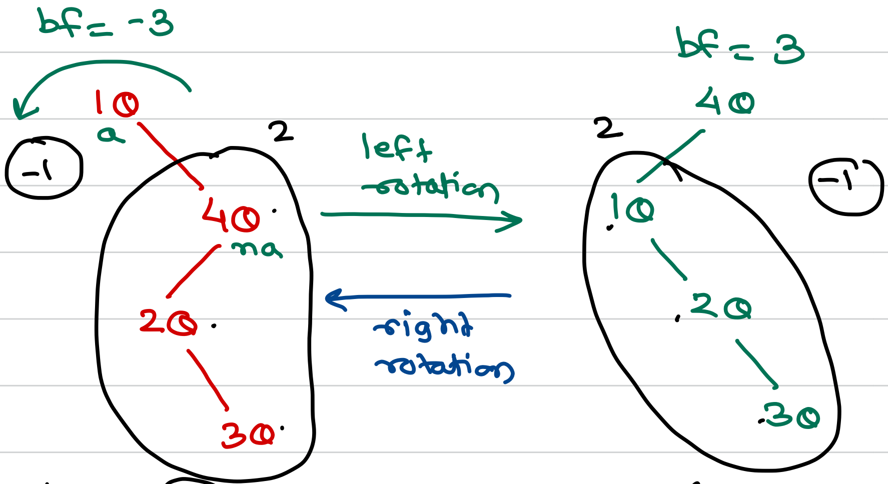

```
newaxis = axis.left;
axis.left = newaxis.right;
newaxis.right = axis;
if(axis == root)
    root = new axis;
—
—
```

Left rotation







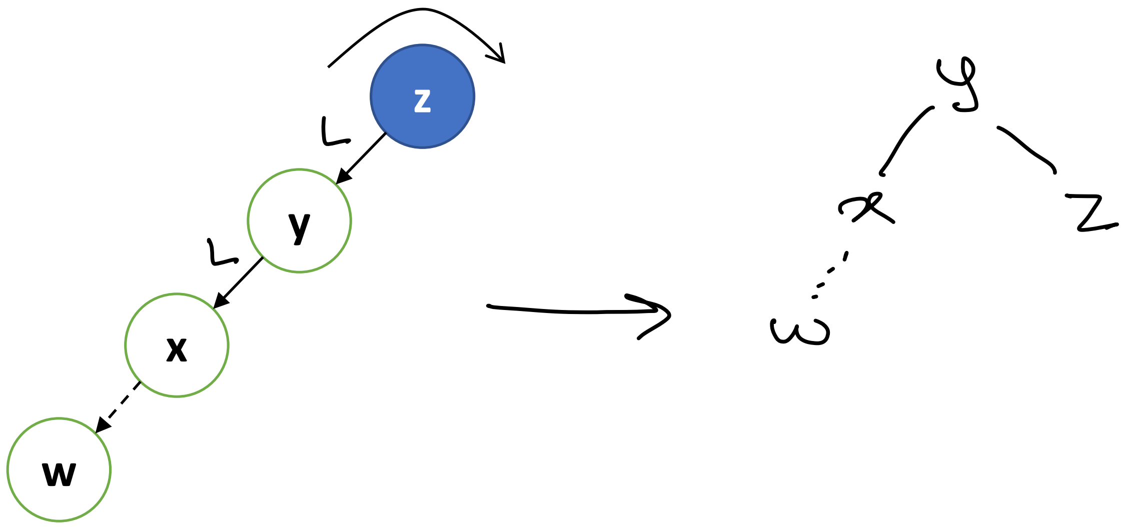


bf = -3
10
a
-1
2
40
na
20
30
left rotation
right rotation
bf = 3
40
2
-1
10
20
30

$-1-2 = -3$

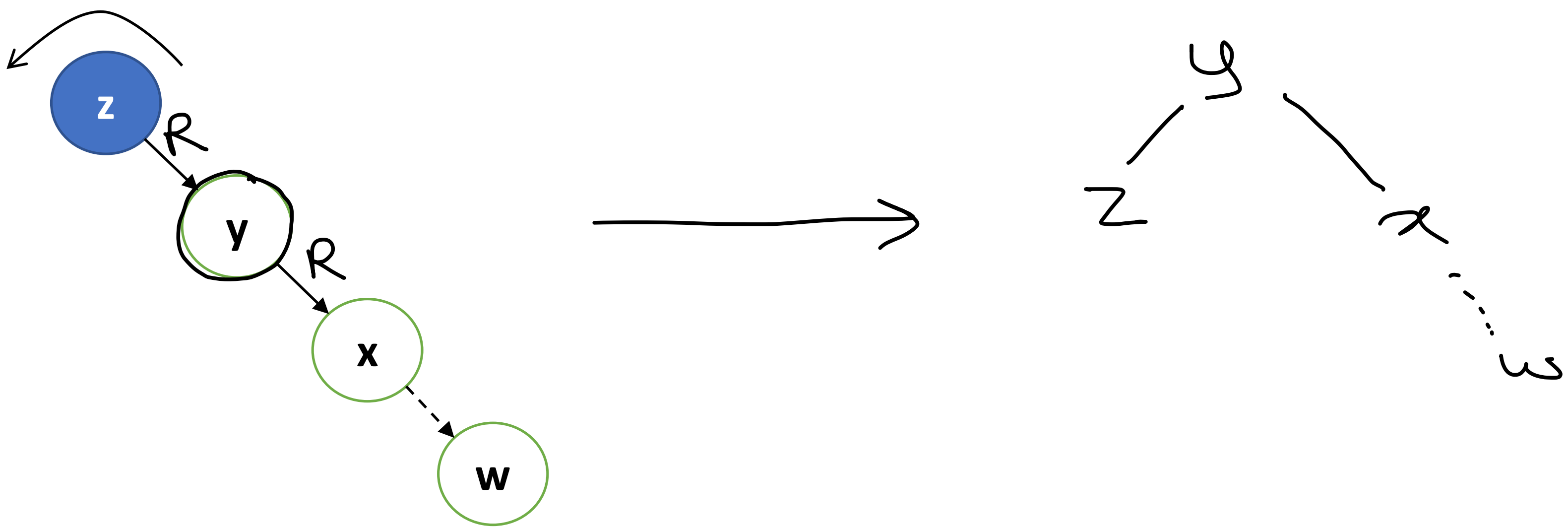
$2-(-1) = 3$

Rotation cases



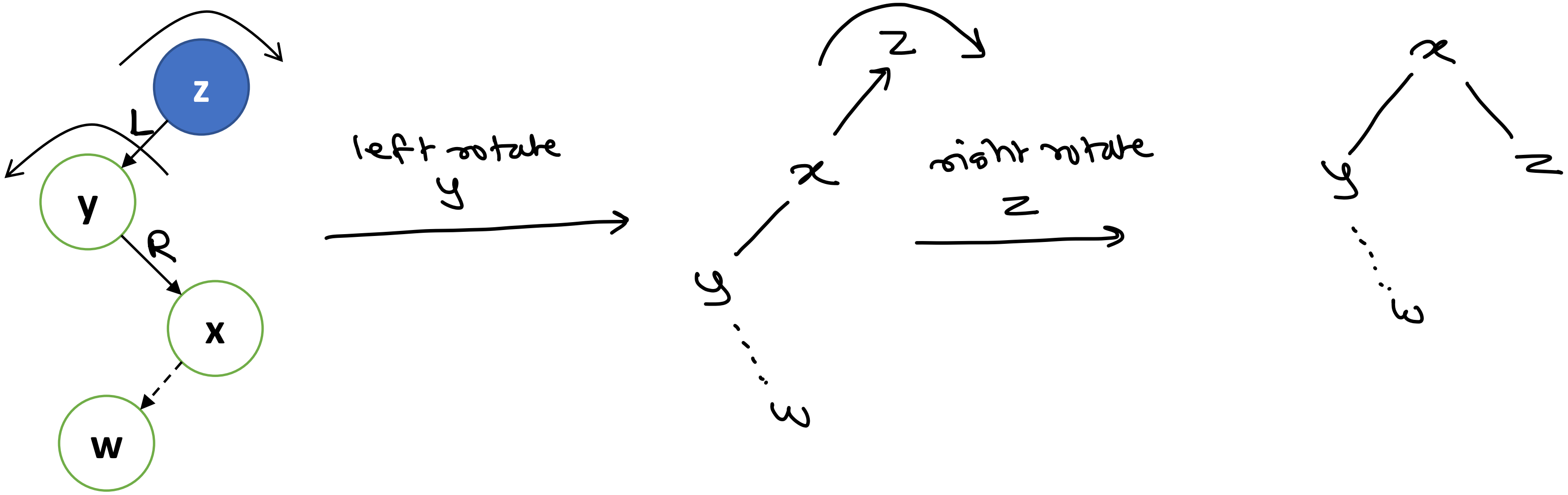
Left-Left case

Rotation cases



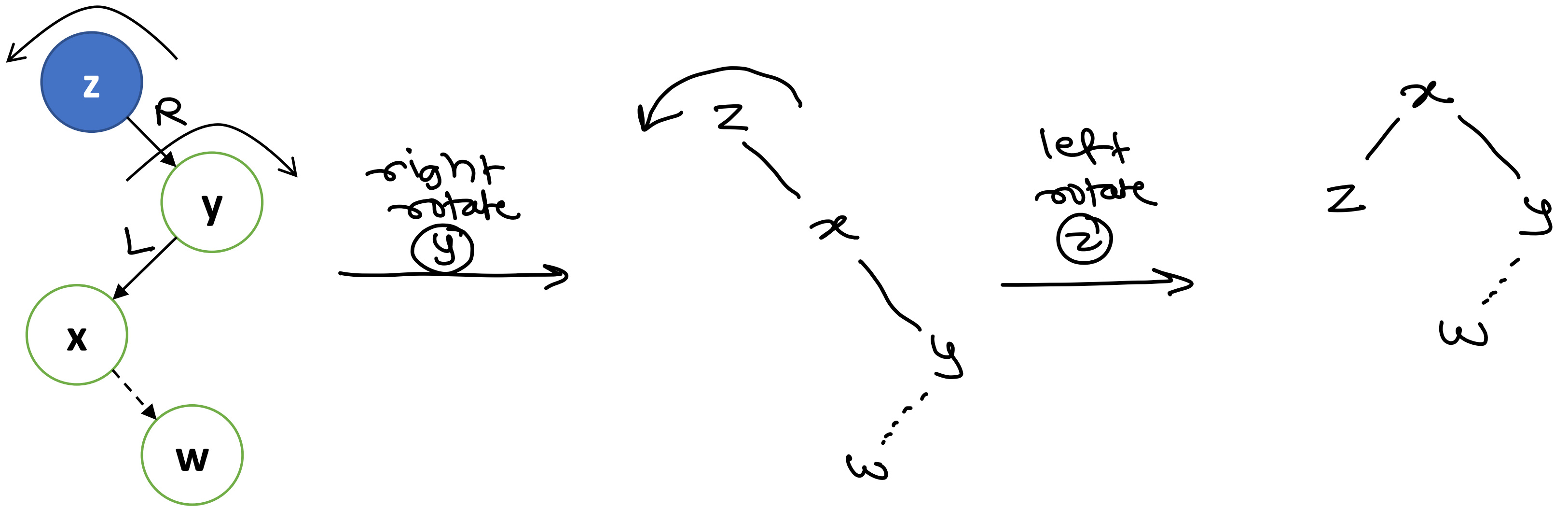
Right-Right case

Rotation cases

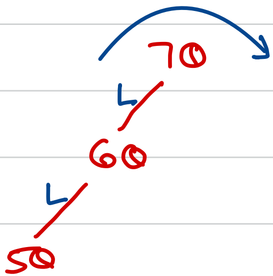


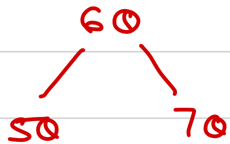
Left-Right case

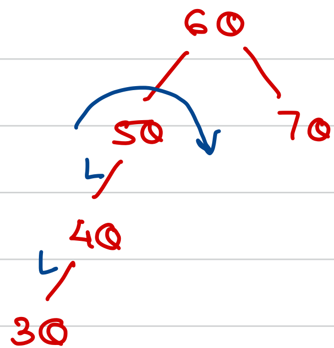
Rotation cases

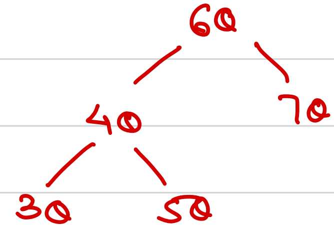


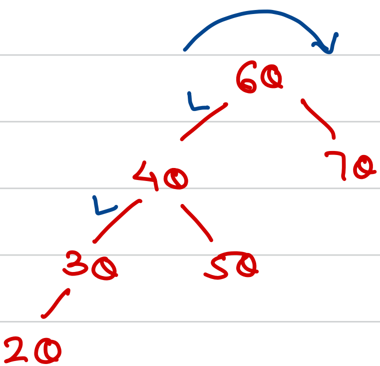
Right-Left case

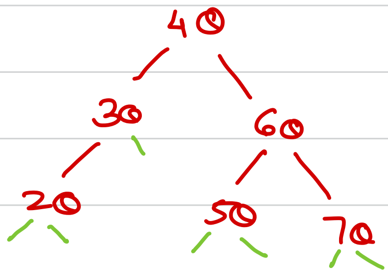

70
L
60
L
50


60
50
70


60
50
70
L
40
L
30


60
40
70
30
50


60
L
40
70
L
30
50
20


40
30
60
20
50
70

AVL Tree

- AVL tree is a self-balancing Binary Search Tree (BST).
- The difference between heights of left and right subtrees cannot be more than one for all nodes.
- Most of BST operations are done in O(h) i.e. O(log n) time.
- Nodes are rebalanced on each insert operation and delete operation.
- Need more number of rotations as compared to Red & Black tree.

$$2^h \simeq n$$

$$h \log 2 = \log n$$

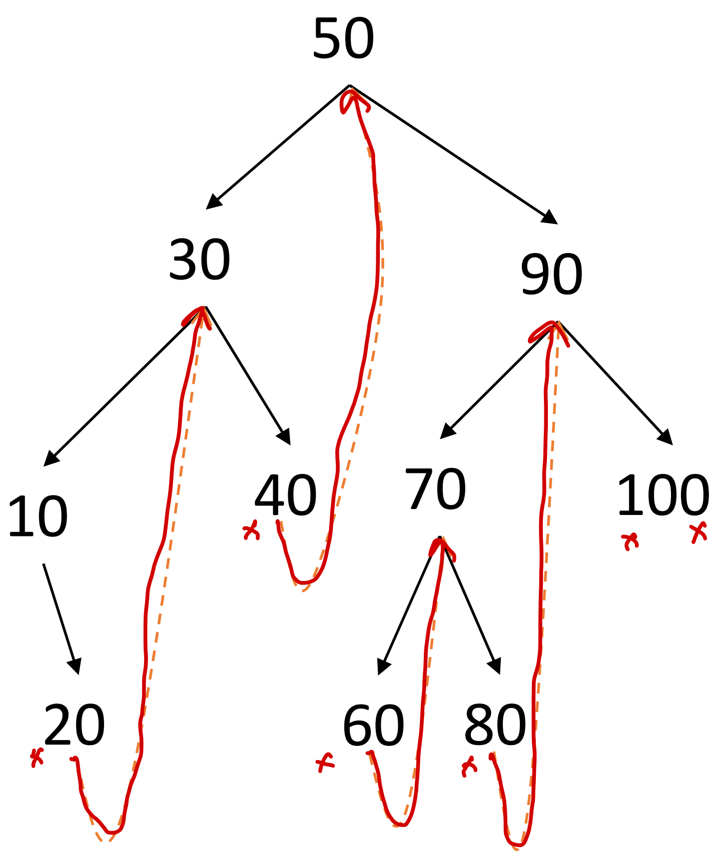
$$h = \frac{\log n}{\log 2}$$

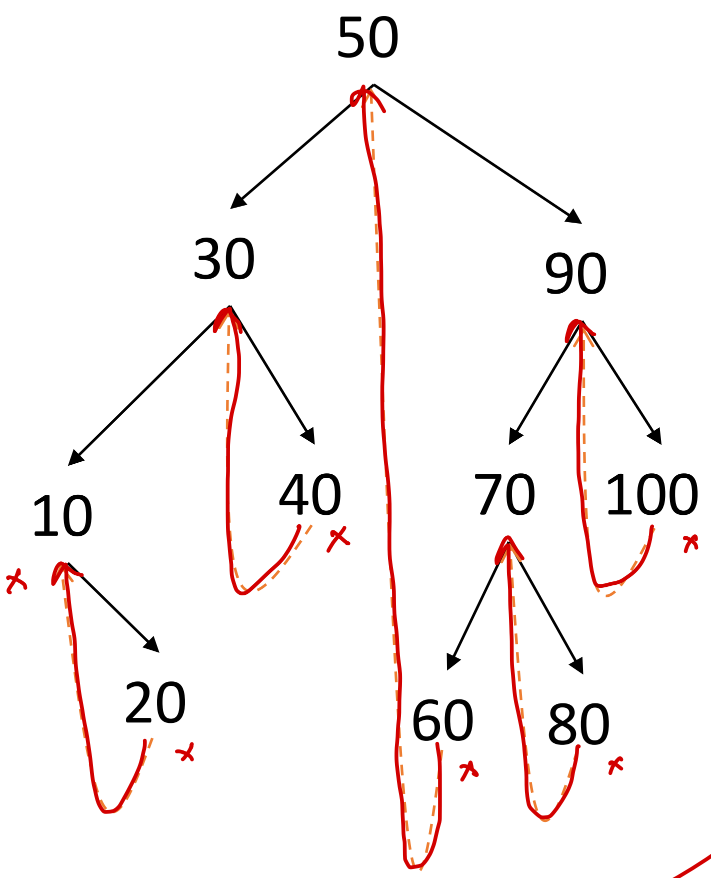
$$T \propto h$$

$$T \propto \frac{\log n}{\log 2}$$

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

Threaded BST

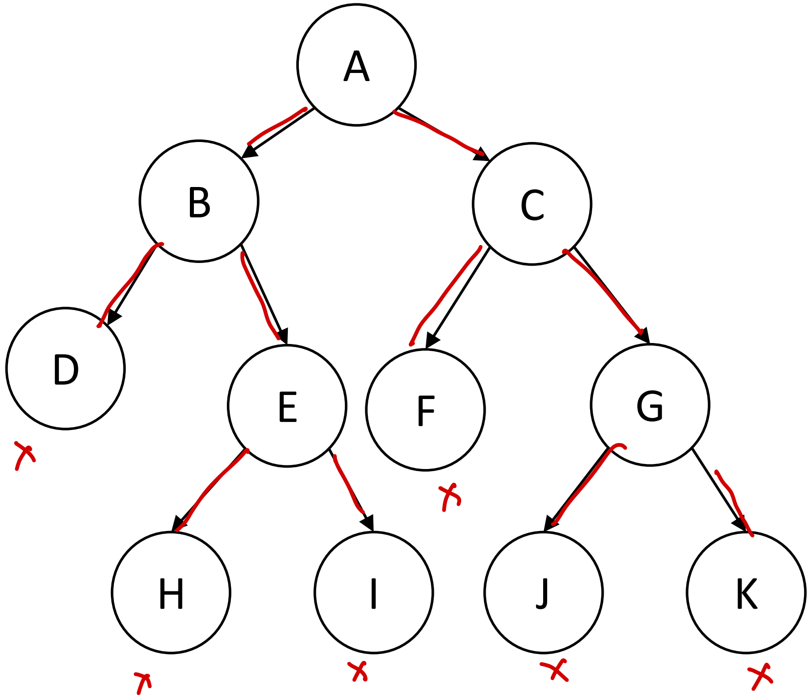




- Typical BST in-order traversal involves recursion or stack. It slows execution and also need more space.
- Threaded BST keep address of in-order successor or predecessor addresses instead of NULL to speed up in-order traversal (using a loop).
- Left threaded BST
- Right threaded BST
- In-threaded BST

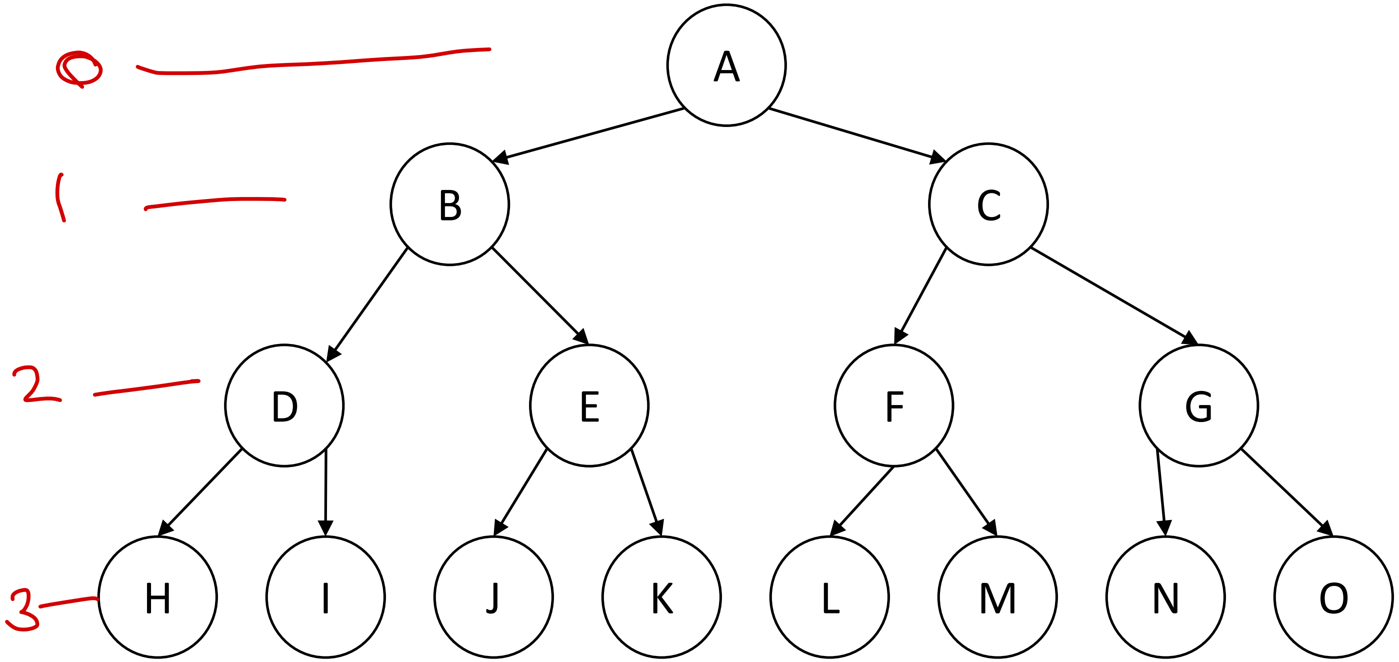
left + right

Strict Binary Tree

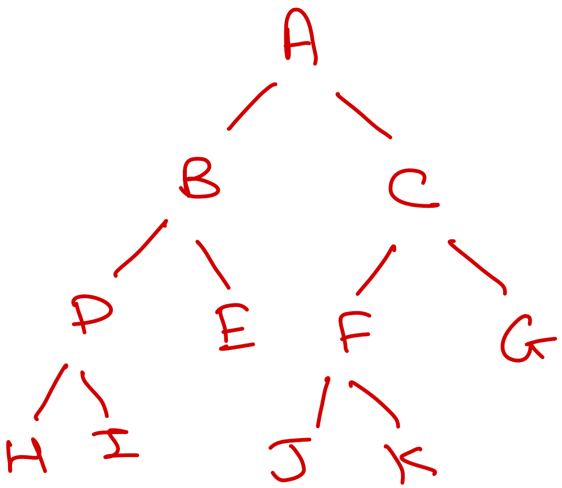


- Binary tree in which each non-leaf node has exactly two child nodes.

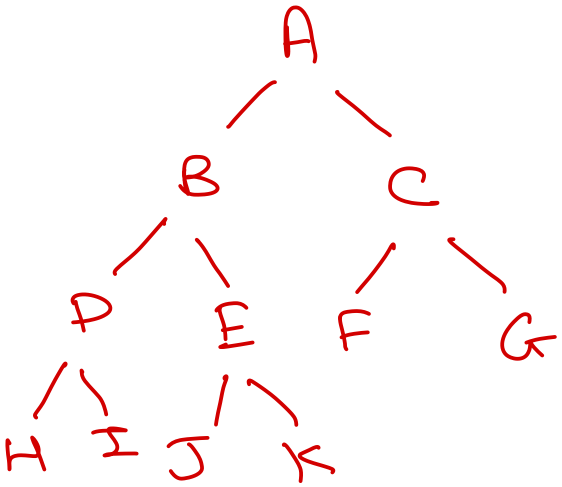
Perfect Binary Tree



- Binary tree which is full for the given height i.e. contains maximum possible nodes.
- Number of nodes = $\cancel{2^h - 1}$ $2^{h+1} - 1$

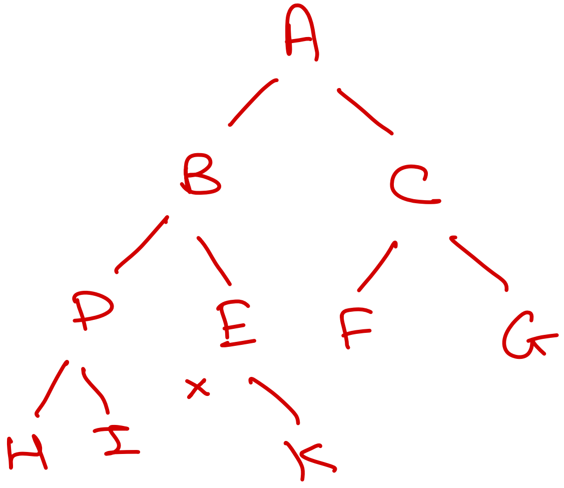


strict



strict

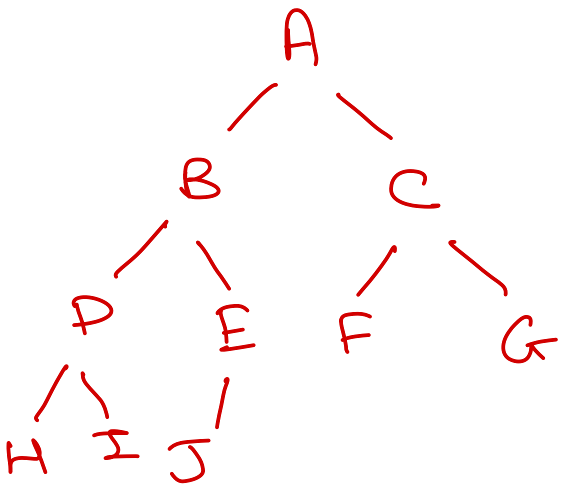
complete



Strict X

Complete X

almost complete X

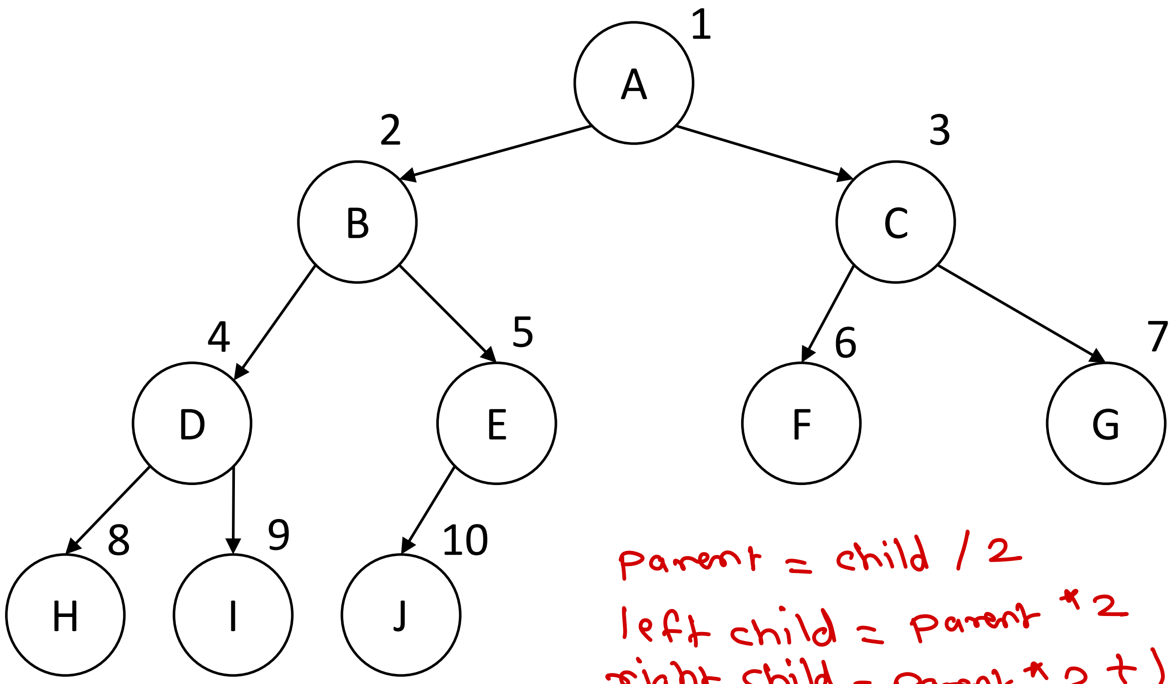


strict X

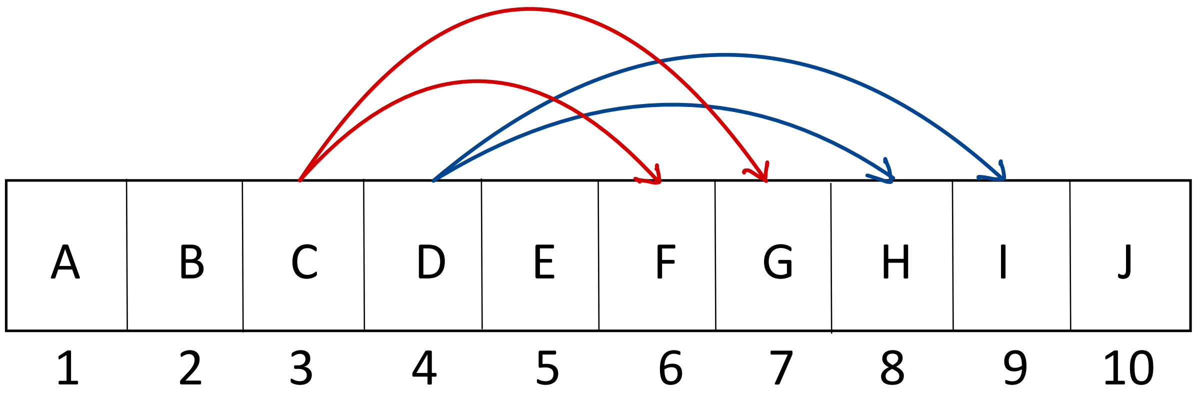
complete ✓

✓ almost complete

Complete Binary Tree and Heap



parent = child / 2
left child = parent * 2
right child = parent * 2 + 1

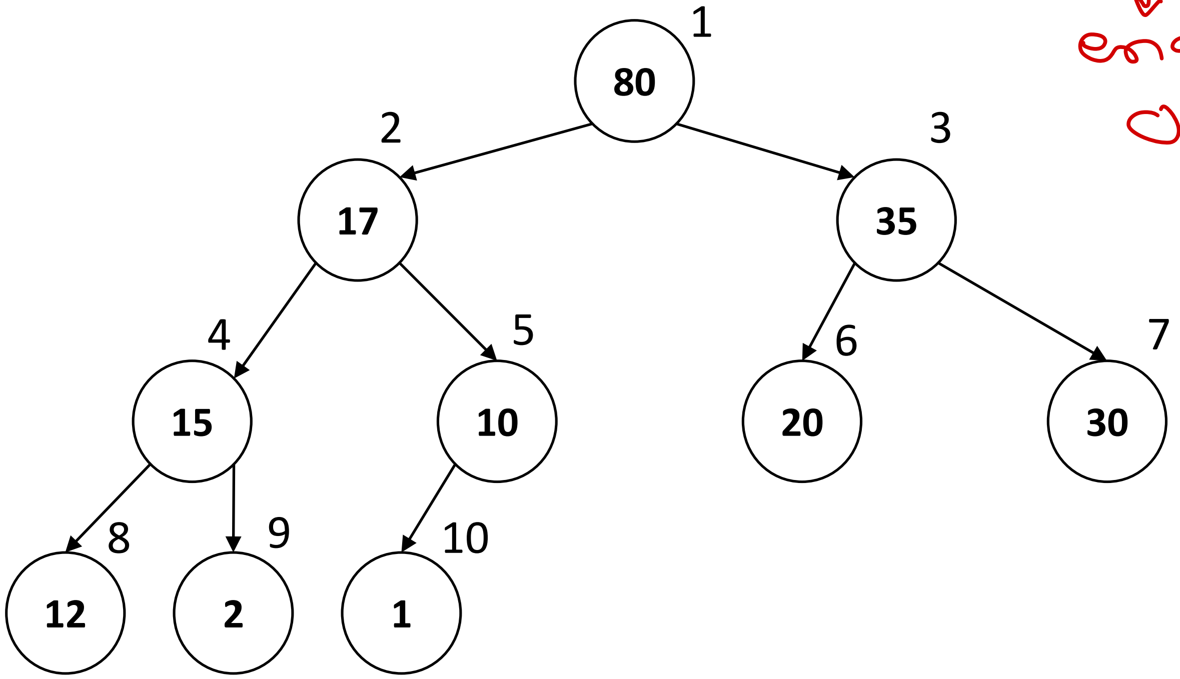


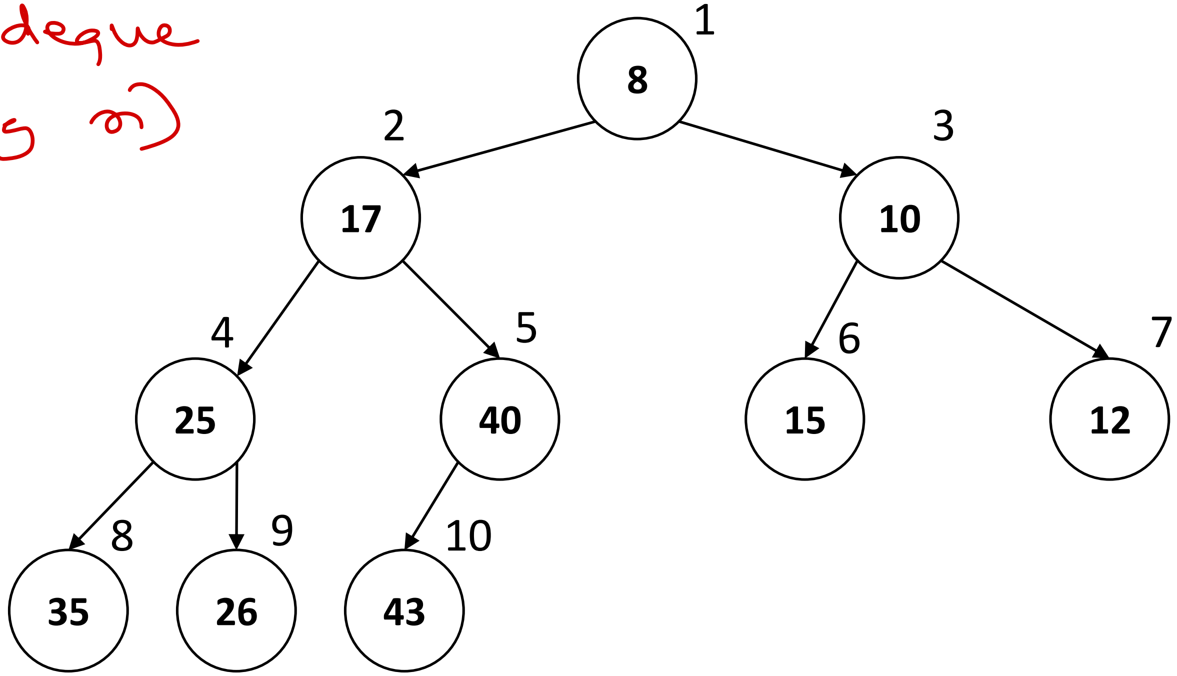
- A complete binary tree (CBT) is a binary tree in which every level, except possibly the last, is completely filled, and all nodes are as far left as possible.
- Almost complete binary tree is similar to CBT, but may not be strictly binary tree.

- Heap is array implementation of complete binary tree.
- Parent child relation is maintained through index calculations
  - parent index = child index / 2
  - left child index = parent index * 2
  - right child index = parent index * 2 + 1

Max Heap & Min Heap → used to implement priority queue.

enque/deque
O(log n)

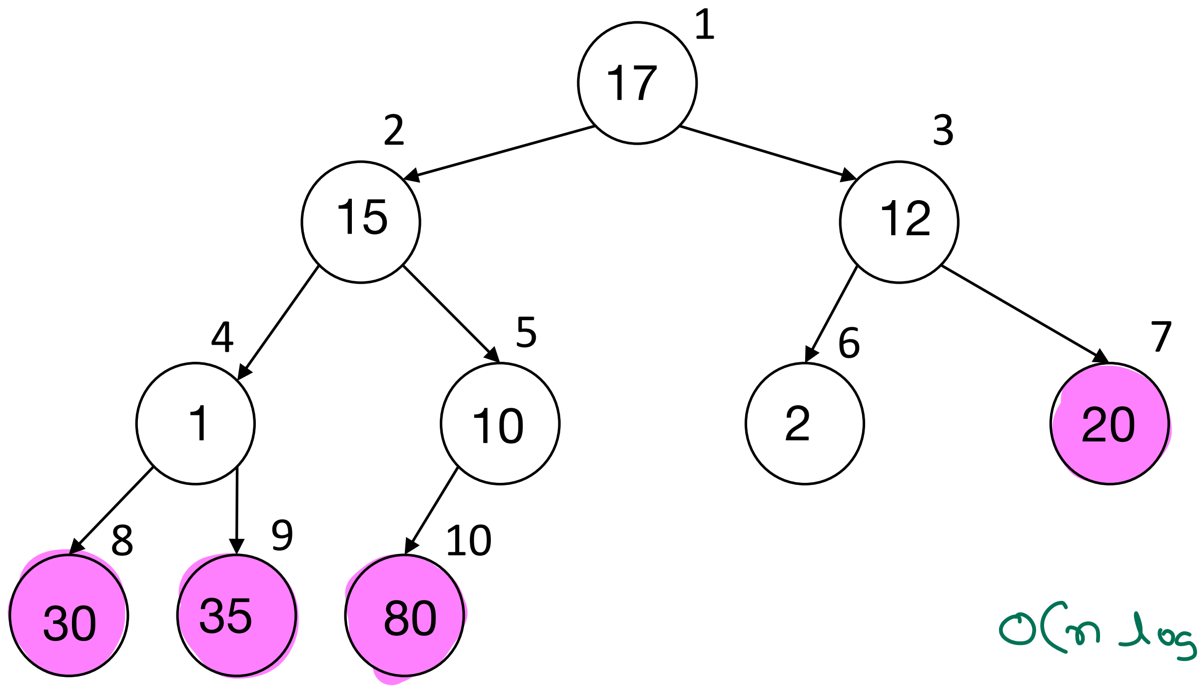




- Max heap is a heap data structure in which each node is greater than both of its child nodes.

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

Heap Sort



Heap Sort → O(n log n)

1. make max heap; → O(n log n)
2. while (heap is not empty)

{
del ele from heap; → O(log n)
put val at end;
}

O(n log n)

17	15	12	1	10	2	20	30	35	80
1	2	3	4	5	6	7	8	9	10

Merge Sort

8 3 9 1 5 7 2 6 4

l m m+1 r

0	1	2	3	4	5	6	7	8
8	3	9	1	5	7	2	6	4

? ?

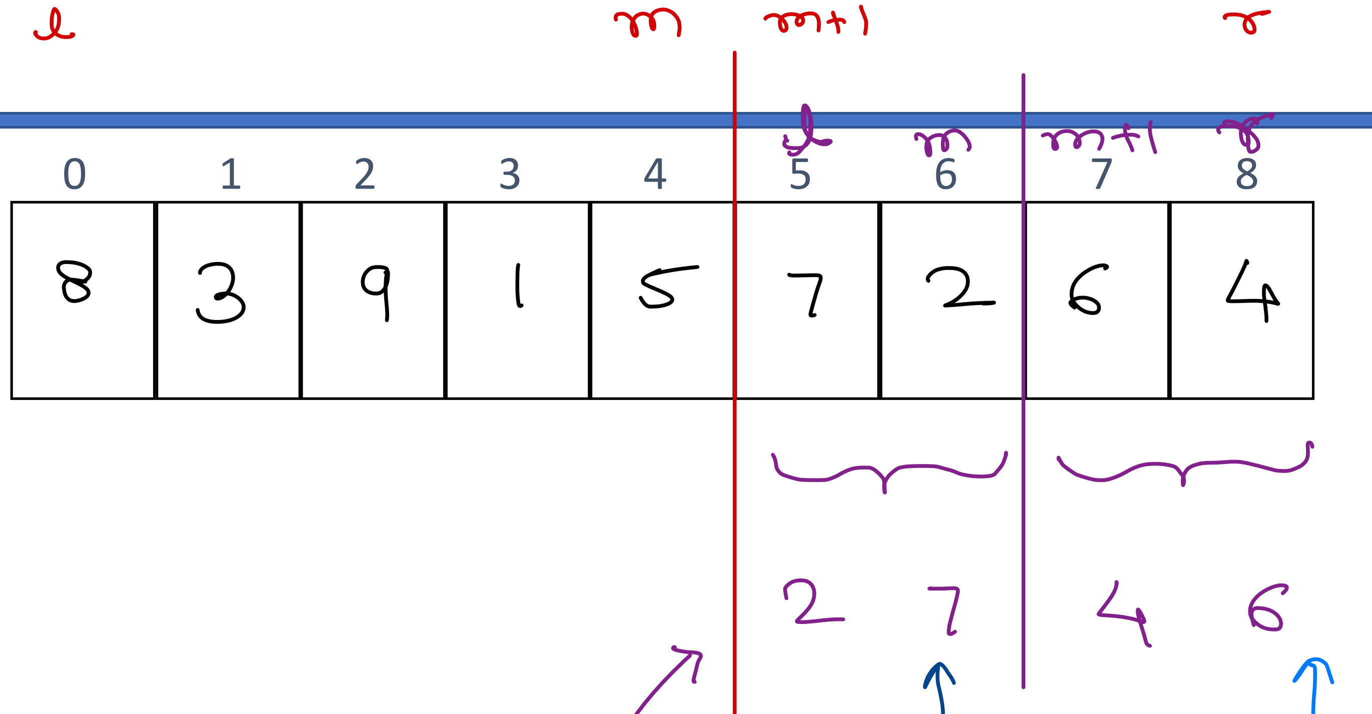
1 3 5 8 9 | 2 4 6 7

i j

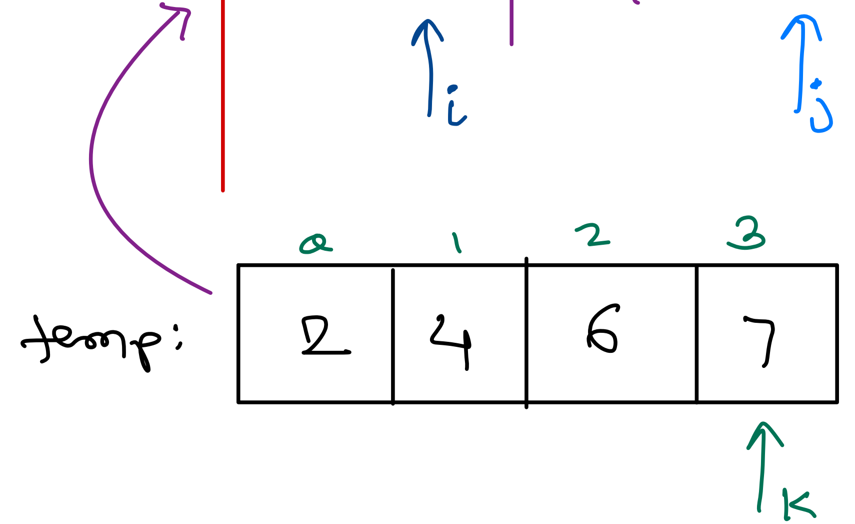
temp: 1 2 3 4 5 6 7 8 9

k

Merge Sort

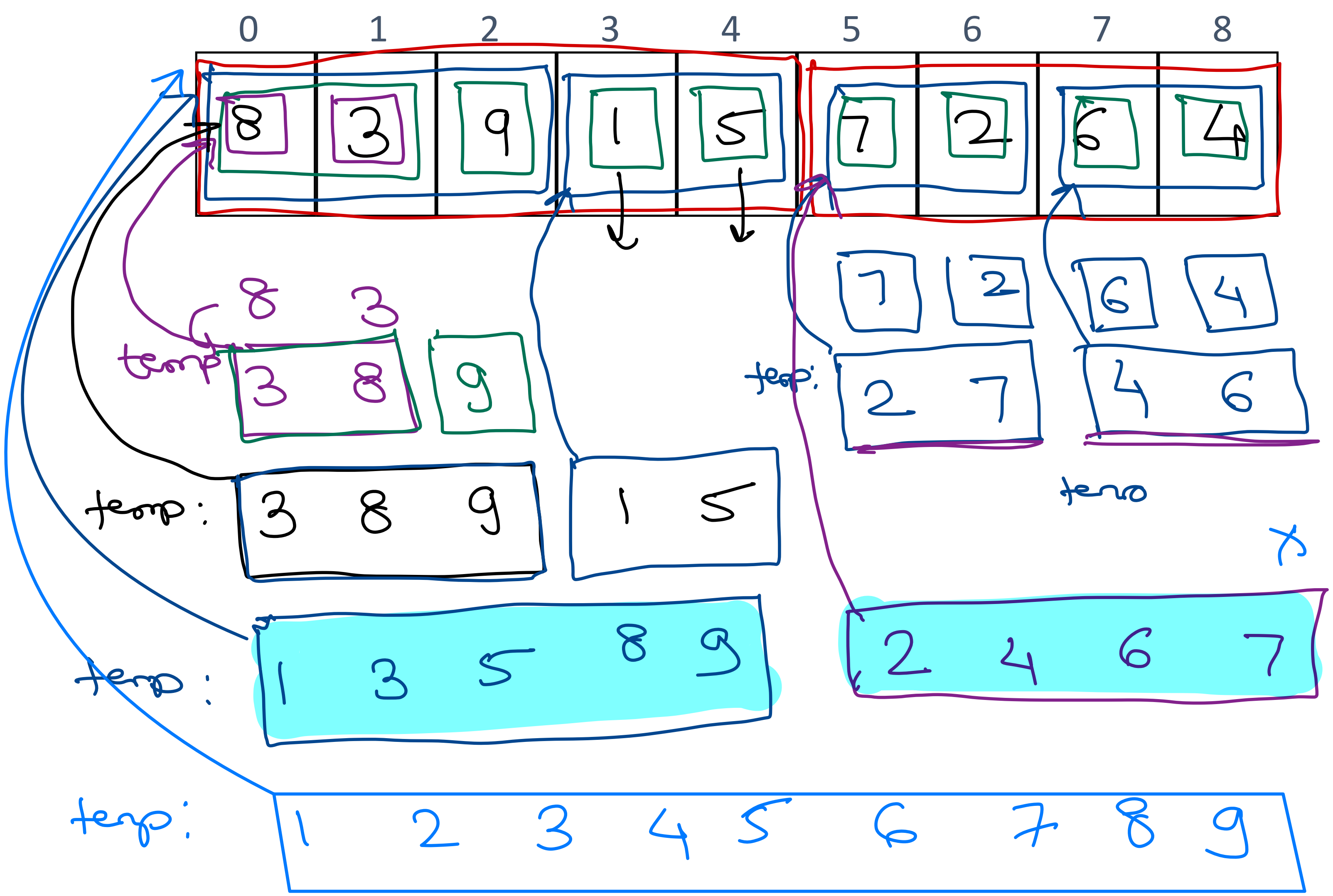



```
for (i=0; i< temp.len; i++)
    arr[left+i] = temp[i];
```



Merge Sort

l=0 m=4 | 5 r=8



Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>