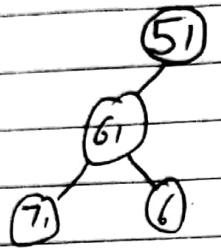
TREE

23/3/18

$T = (V, E)$ → Edges
↳ Vertices

- There should be no loop inside tree

Eg.



$E = \{(51,61), (61,71), (61,6)\}$
$V = \{51, 61, 71, 6\}$

In bidirectional tree
$(61,51) = (51,61)$
$(61,71) = (71,61)$
$(6,61) = (61,6)$

Eg (7)

$V = \{7\}$
$E = \phi$

$\boxed{E \subset V \times V}$

Eg $V = \{7,8\}$ Tree should always be connected
$E = \phi$ — ~~Not a tree~~

Tree with 'n' nodes has:
$|V| = n$
$|E| = n-1$

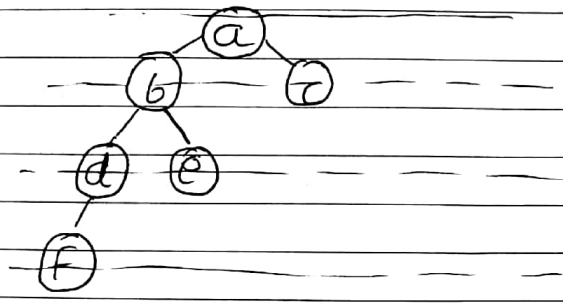
- To move from one node to another node there is only one path
- Collection of Tree is Forest

Null tree:
$V = \phi$
$E = \phi$
$h = -1$

__/__/____

Binary tree

- There should be a root node
- There showed be atmost two child



Level	height
0	1
1	2
2	3
3	4

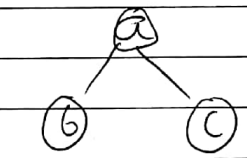
a is root of tree
a, b are ancestors of f
f, e, c are leaves
d, e are siblings
b, c are children of a
b is parent of d, e
b is grandparent of f

$l+1 = h$

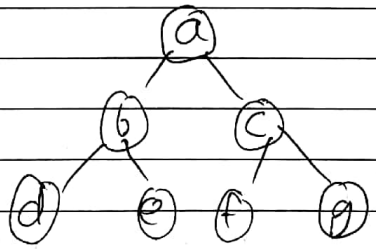
28-3-18

Full Binary tree:-
Tree with each level full of nodes

eg



$2^2-1 = 3$



$2^3-1 = 7$

Formula for no. of nodes = $2^h - 1$

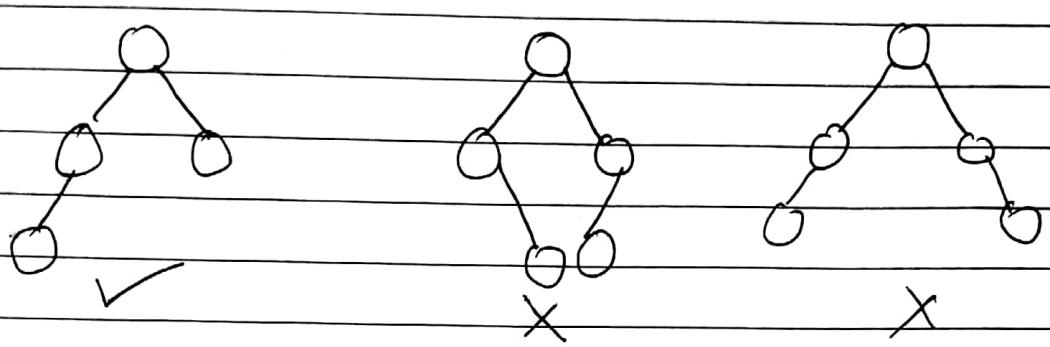
$$\lceil \log_2(n+1) \rceil \leq \text{Height of Binary tree} < n$$

where n is no. of nodes

Complete Binary tree :-

Except last level all level should be full and also last level should be filled from left to right.

eg

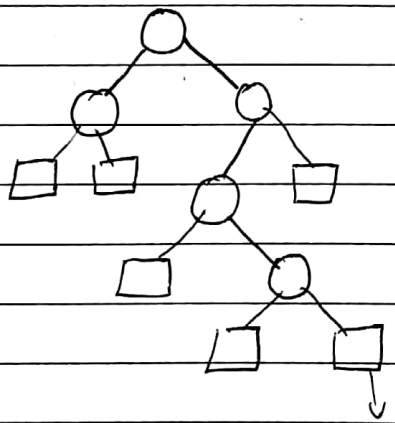


$F \subseteq C$

Full binary tree — Complete Binary tree

Extended Binary Tree :-

It is tree in which each node has 0 or 2 children



Nodes with 0 children are external nodes '□'

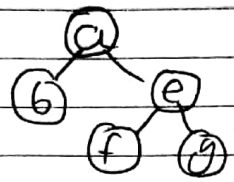
Nodes with 2 children are internal nodes '○'

Special node or orphan child

29/03/18

Binary tree traversal :-

1. Pre order
2. Post order
3. In order

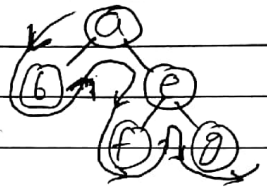


- **Pre order**

1. Traverse Root.
2. Perform Pre order on left subtree of Root.
3. " " " Right " " "

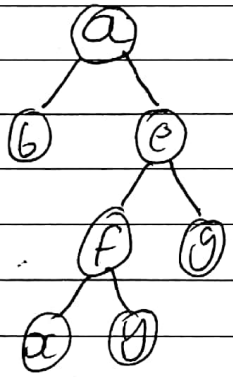
a, b, e, f, g

Shortcut method:-



Post order

1. Perform Post order on left subtree of Root
2. " " " " Right " " "
3. Traverse Root

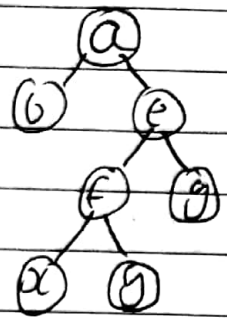


b, x, y, f, g, e, a

__/__/____

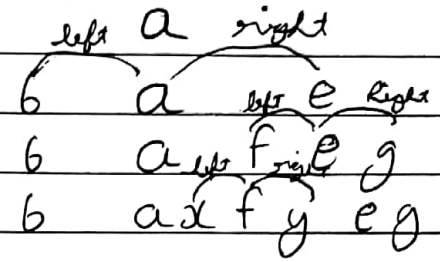
Inorder:

1. Perform Inorder on left subtree of Root
2. Traverse Root
3. Perform Inorder on Right Subtree of Root



b, a, x, f, y, e, g

Shortcut method:-

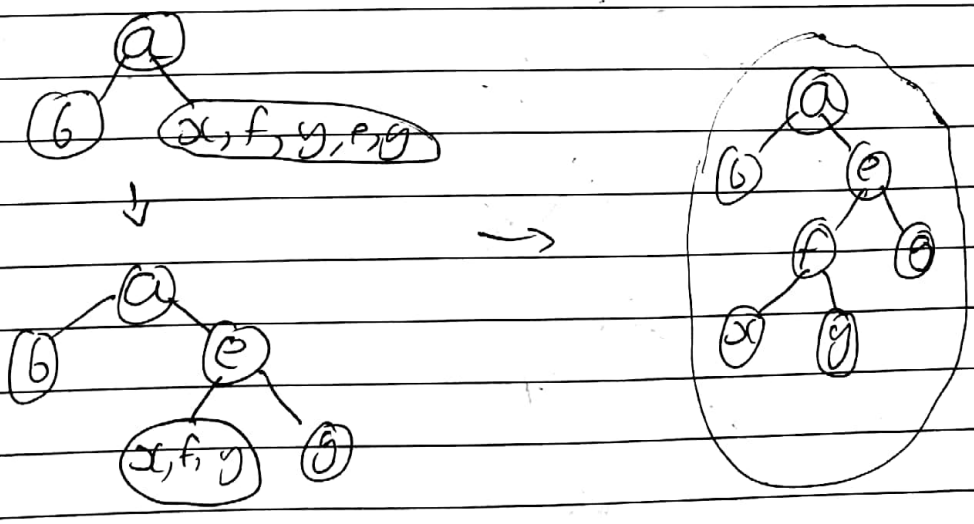


b, a, x, f, y, e, g

We can't tell exact unique tree from preorder or postorder or inorder.

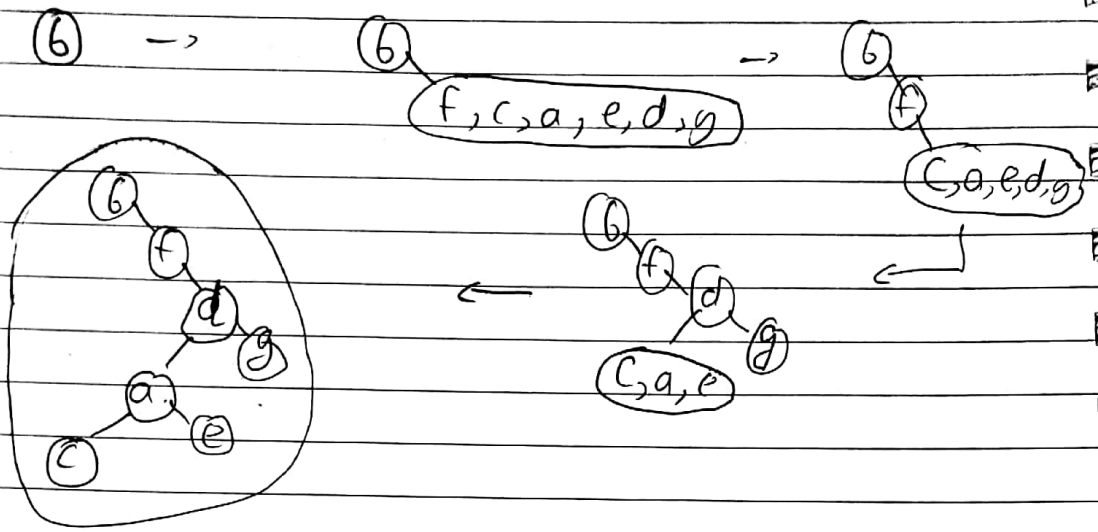
We can find a unique tree from preorder and inorder or postorder & inorder

eg. Pre: a, b, e, f, x, y, g
In: b, a, x, f, y, e, g

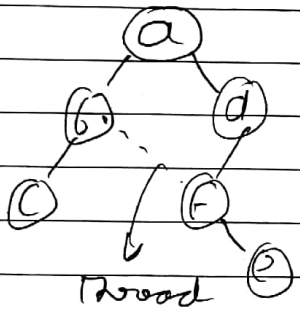


__/__/____

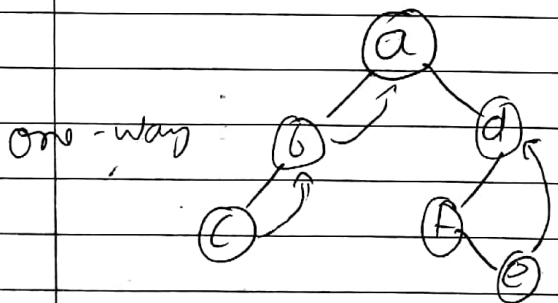
eg Post: C, e, a, g, d, f, b
In: b, f, c, a, e, d, g



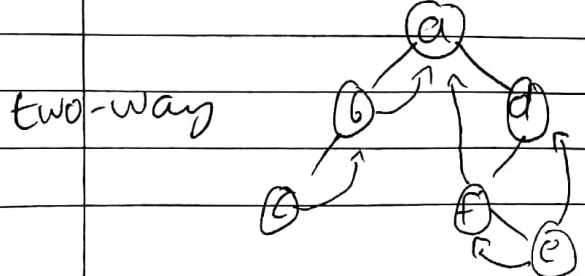
Threaded Binary tree:



Inorder Threading:



Inorder
C, b, a, f, e, d

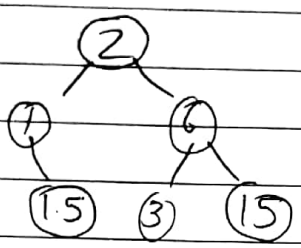


__/__/____

Binary Search Tree (BST)

- Should be Binary tree
- at each node, right side child should be smaller than parent and left side child should be bigger than parent.

eg



Operation :-

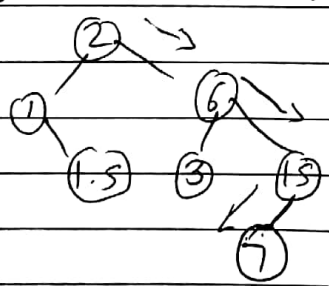
1) Search :-

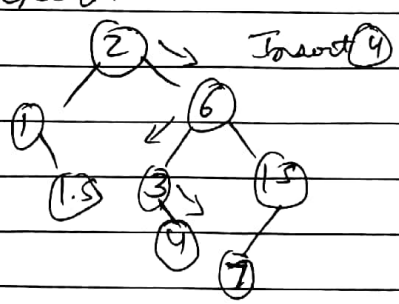
- To search we will start from root like if we want to search 15.
  15 > 2 hence we should move to right subtree of 2
  15 > 6 hence we " " " " " "
- like if we want to search 1.5
  1.5 < 2 hence we should move to left subtree of 2
  1.5 > 1 " " " " " right " " 1

2) Insert :-

It is same as search we will traverse until we reach a node with needed child.

eg





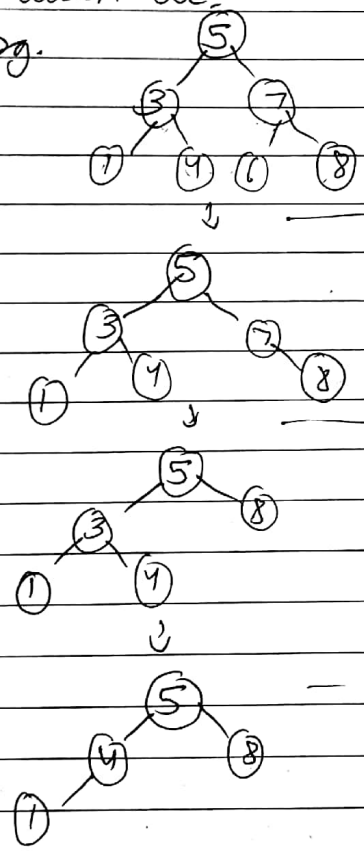
__/__/____

3) Deletion:-

There are three cases of deletion:-

- If the node is a leaf:
  remove the node directly
- If the node has only one child:
  replace or change the Node with its child
- If the node has two children:
  Find the inorder successor and replace with it

eg.



Inorder 1, 3, 4, 5, 8

Inorder successor of 3

Scanned by CamScanner

30/3/18

Adelson-Velskii and Landis (AVL) Trees

- It is BST
- Each node has balance factor 0, 1 or -1.

Balance Factor = $H_{left} - H_{right}$

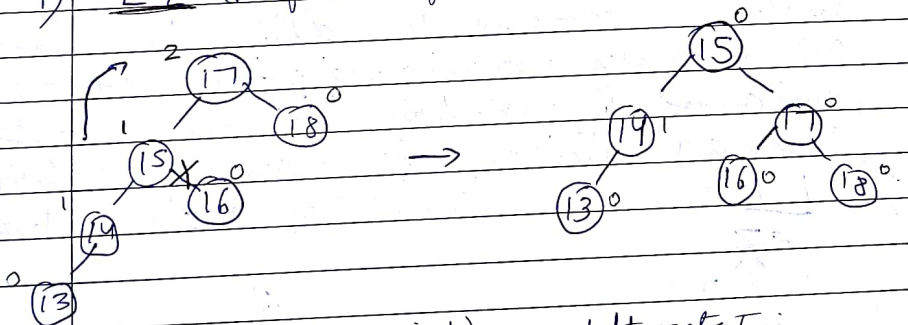
2/ B.F is +ve left side is heavy.
" " " -ve Right " " " .

Insertion:-
Sometimes insert function causes B.F to become 2 or -2 for some node which is needed to adjust by rotation.

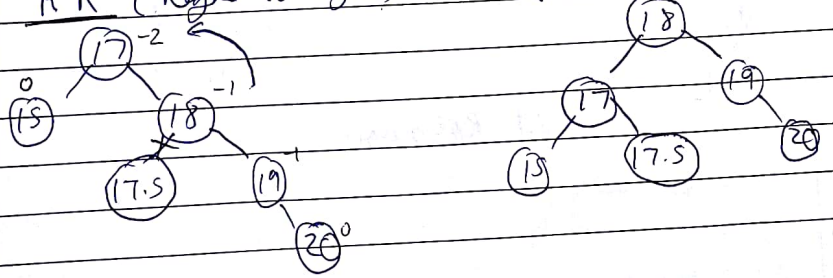
Rotation:-

Outside cases (Single rotation):-

1) LL (Left to Left) → Right rotation



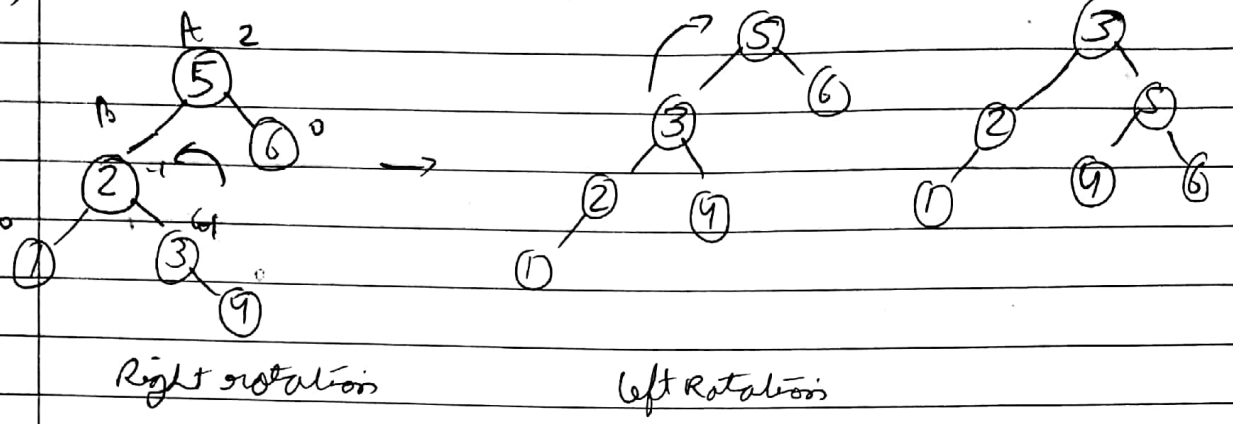
2) RR (Right to Right) → Left rotation



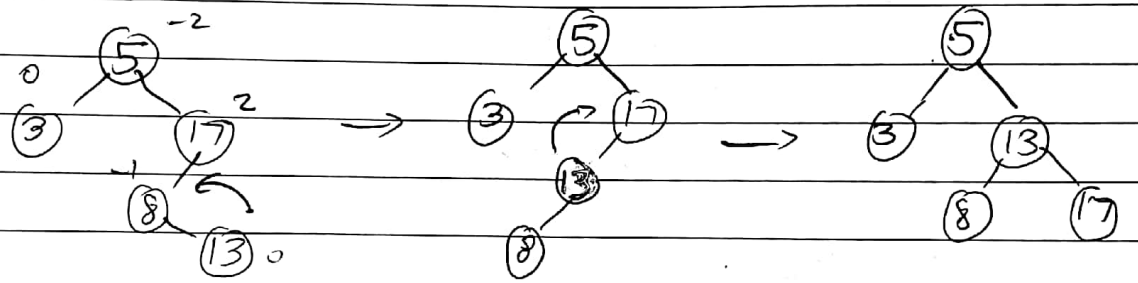
4/9/18

Inside cases :- (Double Rotation)

1) LR



2) RL

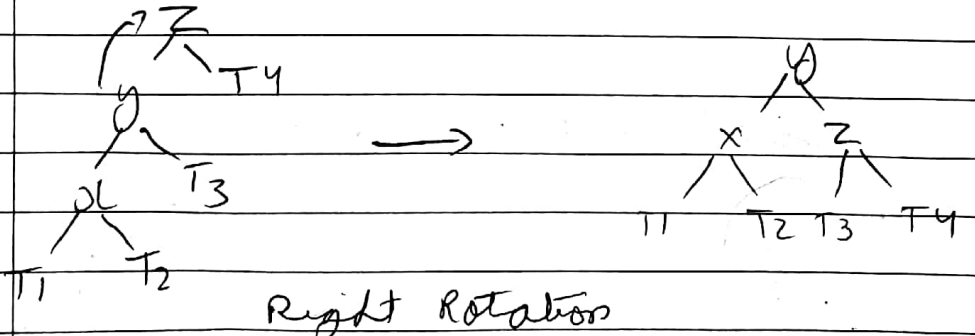


Deletion :-

Deletion is same as that of BST but sometimes Deletion causes unbalance which need to be balanced by rotation like in Insertion. There are similarly four cases of Rotation :

1) LL

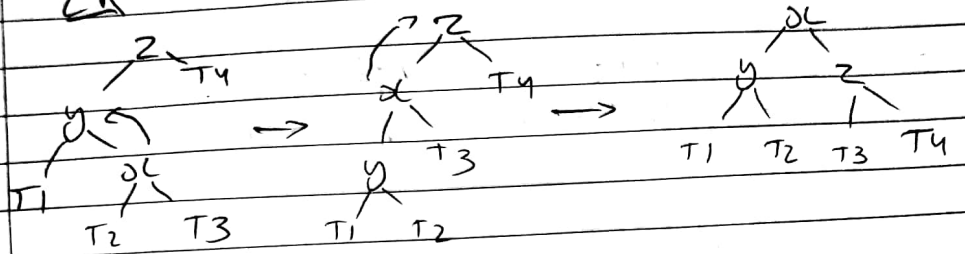
T1, T2, T3, T4 are Subtrees



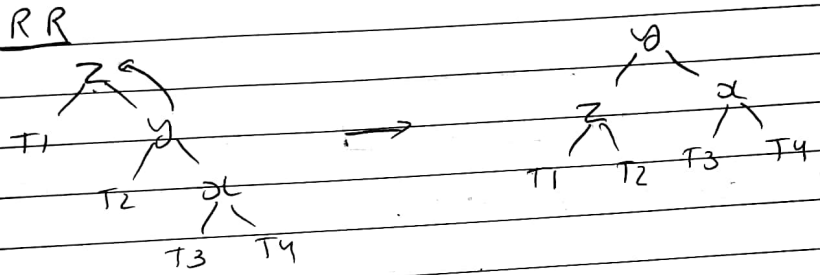
18

__/__/__

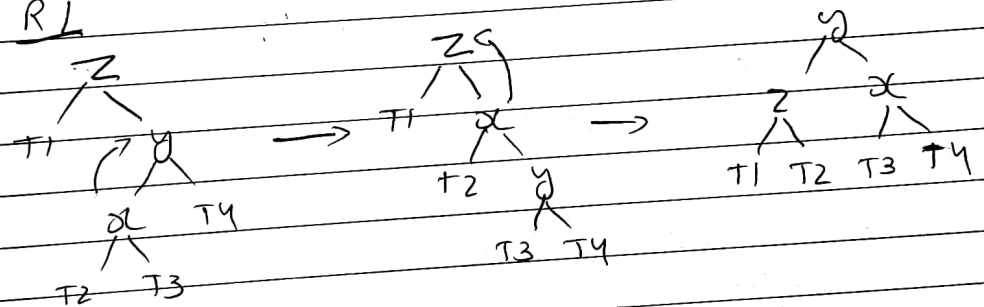
2) LR



3) RR



4) RL



21/9/18

M-way Tree (Multiway tree)

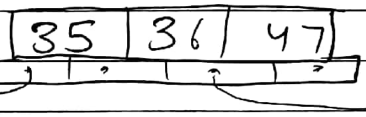
- B/w 1 to (M-1) values in each node
- At most children (M) per node

Binary tree - 2 way tree

eg 4 way - tree

Node contains two arrays one for keys other for links

M-way search tree :-

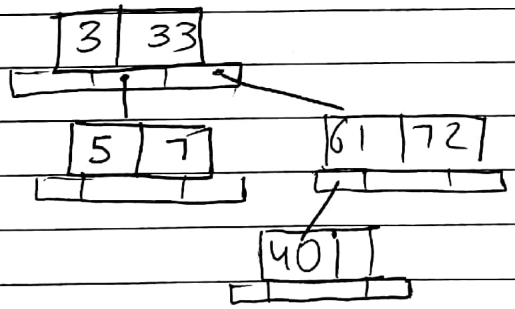


$k_1 < k_2 < k_3 < k_4$ keys

3-way search tree

Insertion :-

3, 33, 5, 7, 61, 72, 40



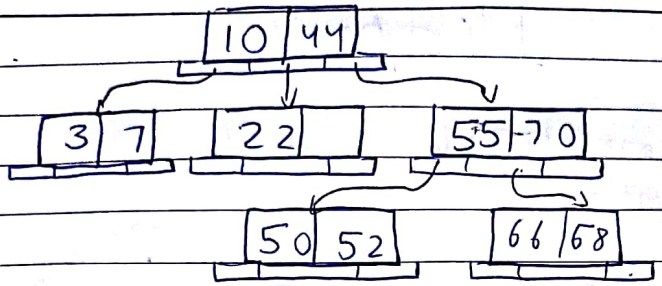
Deletion :-

Same as BST

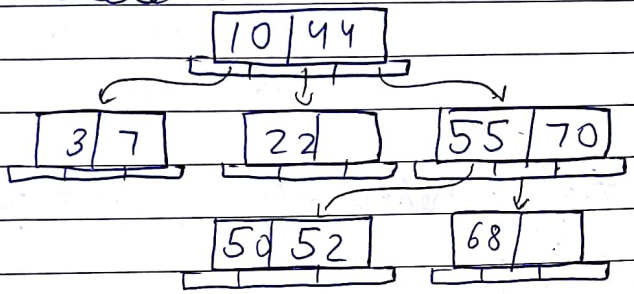
for Deletion of internal node replace by its Successor ↗ not a leaf

__/__/____

eg.

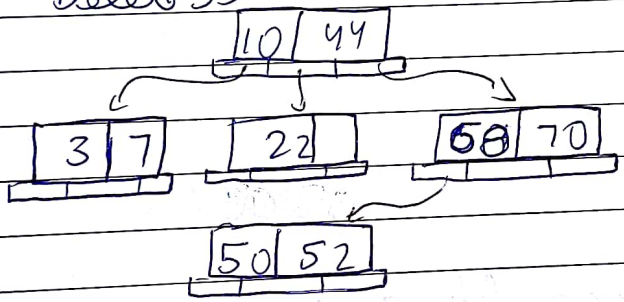


Delete 66

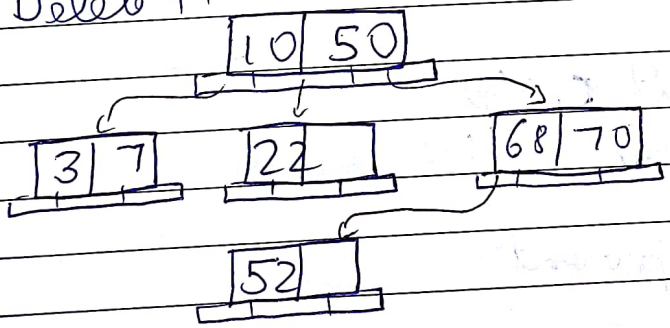


To find Successor first go right then go left... left... left.

Delete 55



Delete 44



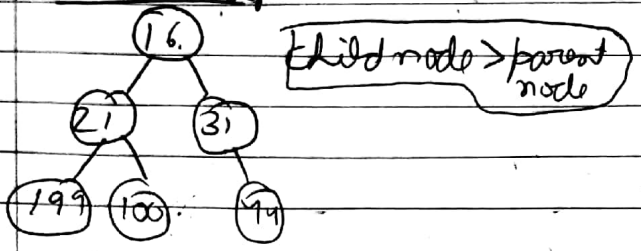
CS. usfca.edu

5/7/18

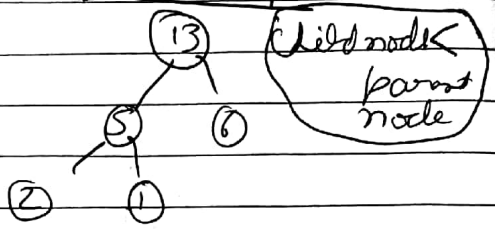
Heap:-

- It is complete binary tree
- Two types:

Min heap

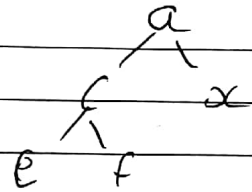


Max heap



- Stored in array or represented by array

eg | a | c | x | e | f |



Formula to find left and right subtrees

→ If (n) is node

- $2n$ is left child
- $2n+1$ is right child
- $\lfloor n/2 \rfloor$ is parent

→ If n - even

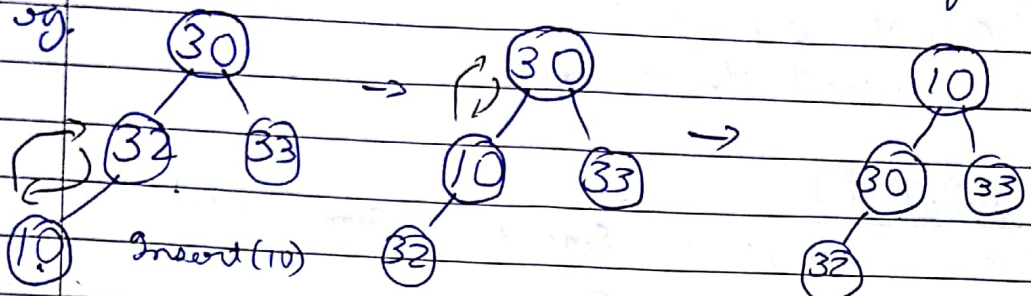
n+1 is sibling and vice versa

Operations :-

1) **Insert() :-**

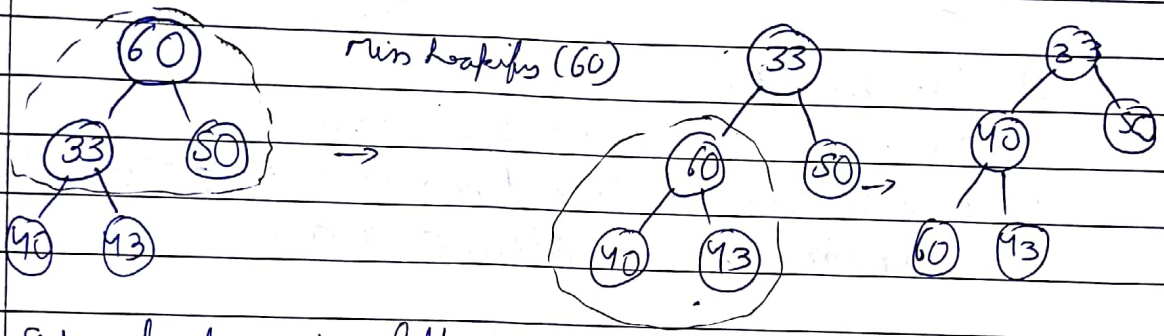
Insert the node at last and check with its parent and swap it if violates cond^n of heap.

eg.



2) **Max/min heapify() :-**

eg.

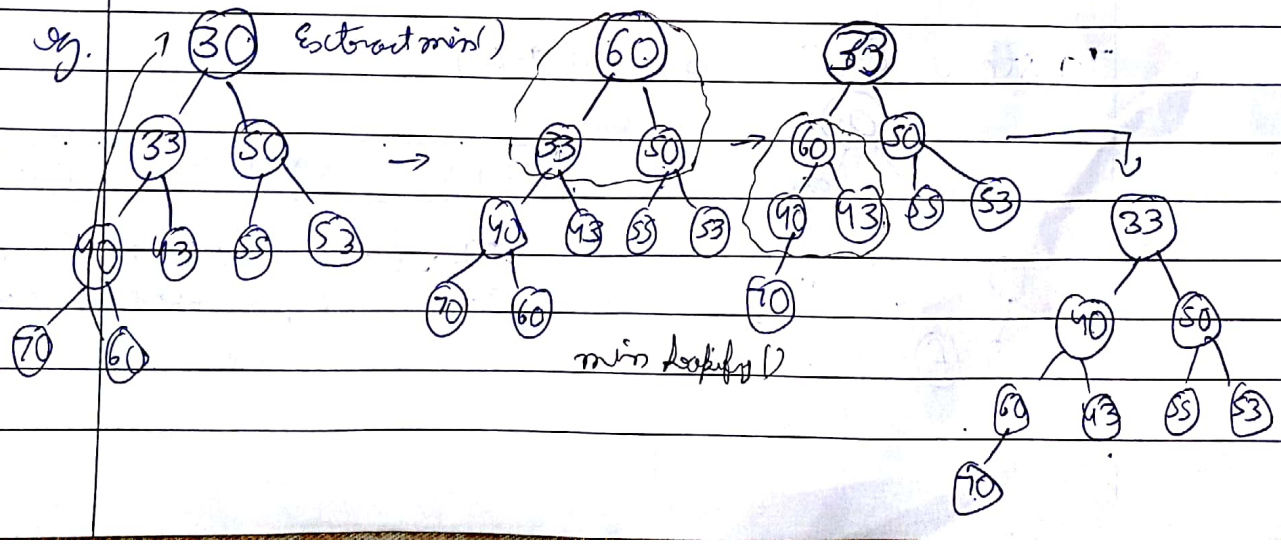


It checks the left and right child and swap with minimum (or maximum) value from there and same procedure goes on till swapping fails or stops.

3) **Extract-min/max() :-**

Take out root and replace it with last element in array and apply heapify() on root.

eg.



__/__/____

4) Decrease / increase Key ():-
Modify any key of Node to increase it if it is Max heap and decrease if it is Min-heap.

- Modify value and swap with its parent as done in insert operation.

5) Delete ():-
- Search (n); → Search is same as done in array (linear searching, Binary searching)
- decrease-key (n, -∞);
- Extract-min ();

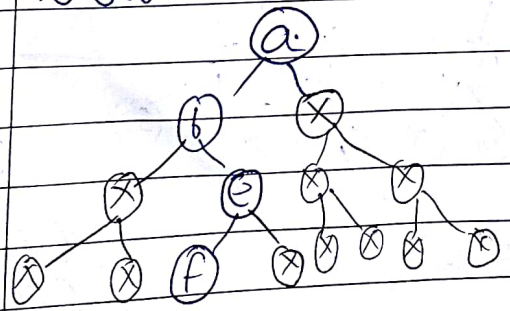
6) Heapsort ():-
To convert heap in sorted heap
- Loop to call Extract min() and store the minimum in other array.

7) Build heap ():-
To convert an array into heap

Method ①
- Make another array and insert elements of first array to other by insert ();

Method ②
- Apply min-heapify for each index

To store BST into array:-



a	b	x	x	e	x	x	x	x	f	x	x	x	x	x

It takes lots of storage, so we use pointers to store