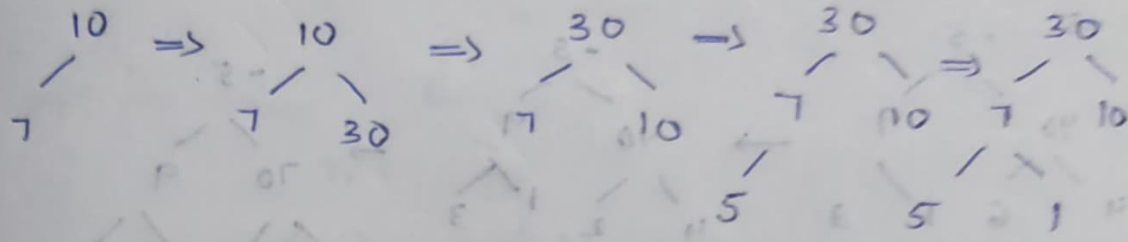
Unit - 1

→ Different types of Heaps

→ Binary Heap:

* Max heap: parent > children

10, 7, 30, 5, 1



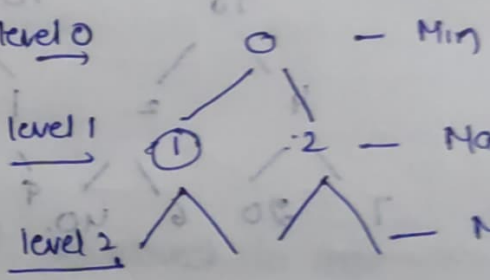
* Min Heap: parent < children.

Extract Max / → O(logn), O(1)

* Min - Max Heap:-

i) Min Max

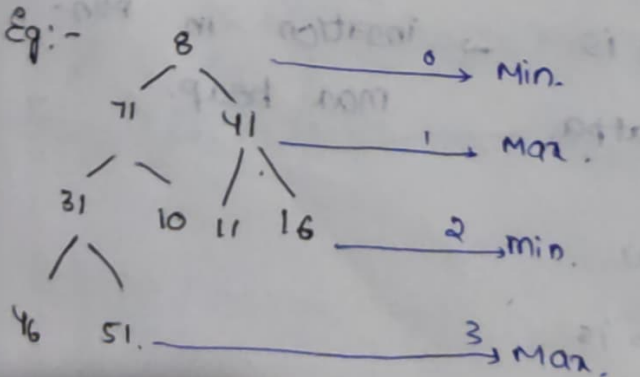
(even) (odd)



ii) Each node at even level is lesser than all of its descendants.

iii) Nodes at odd level is greater than all of its descendants

Eg:-



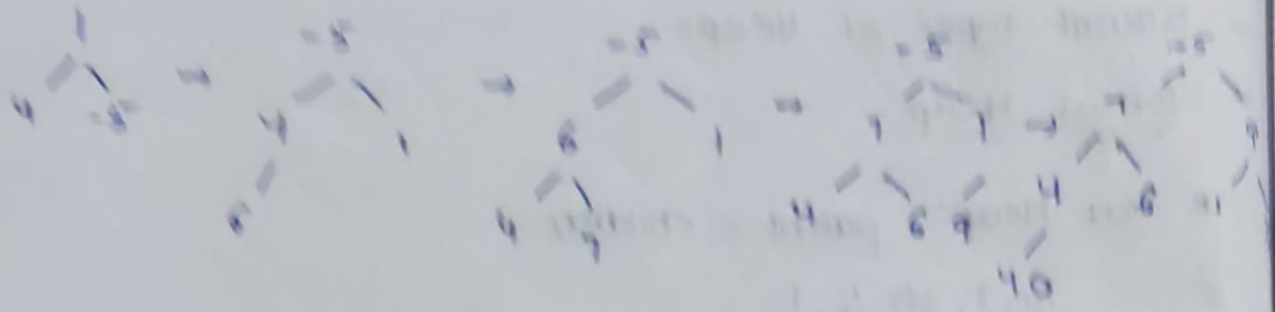
root → Smallest value.

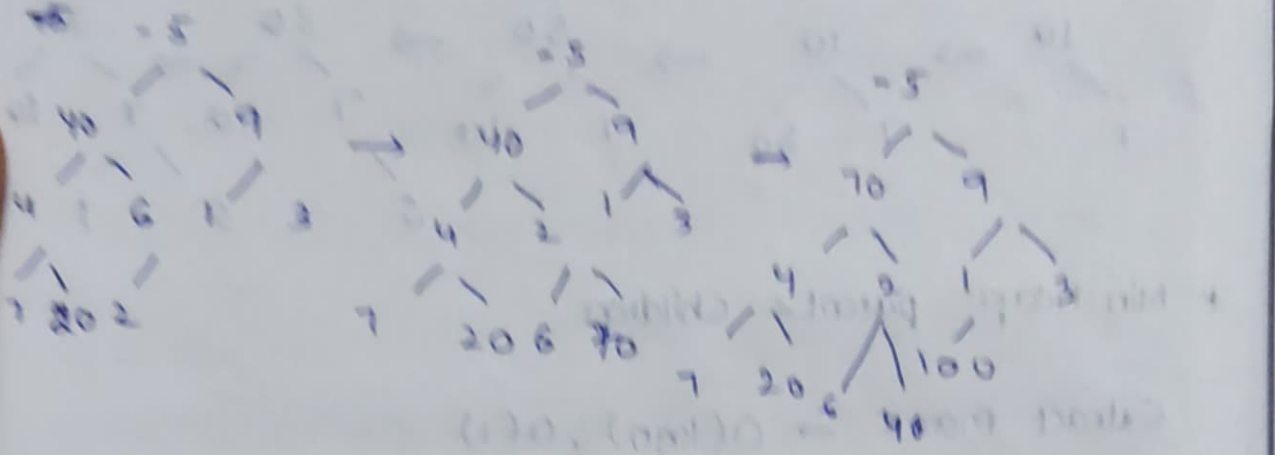
level 1 → largest value.

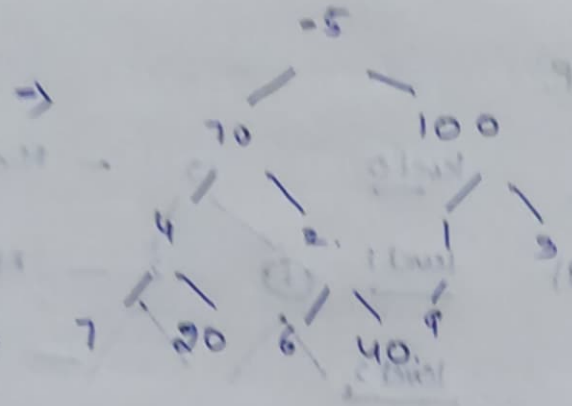
level 2 → 2nd smallest.

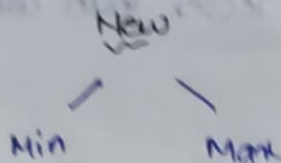
At max level → any node is largest element in the subtree rooted at the node. [same for min level, smallest]

→ Construct Min-Max heap 1, 4, 5, 6, 7, 9, 3, 40, 20, 2, 70, 100









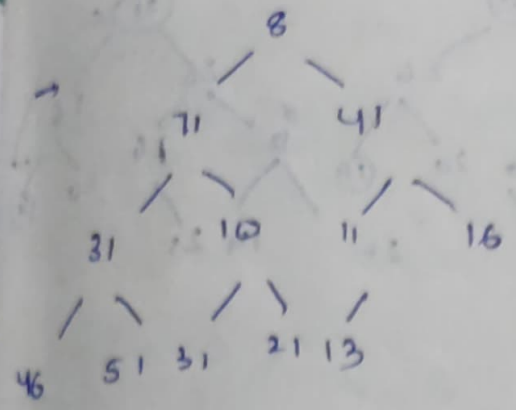
1) pa < new
Max prop is violated at pa

pa > new.
Min prop is violated at pa.

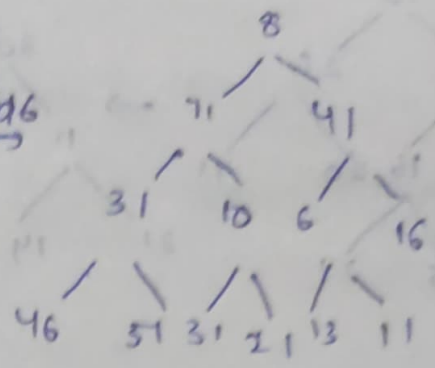
→ Insertion in Min-max heap.

2) gp > new
min prop is violated at gp

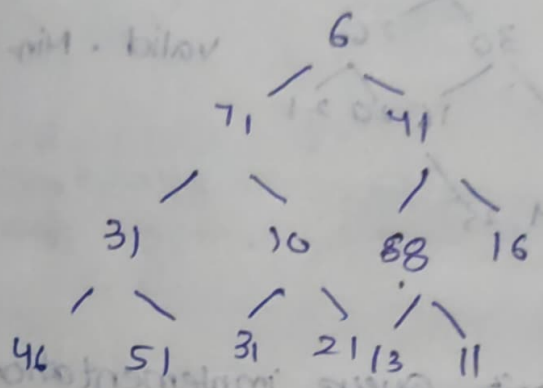
gp < new
max prop is violated at gp



add 6 →



→



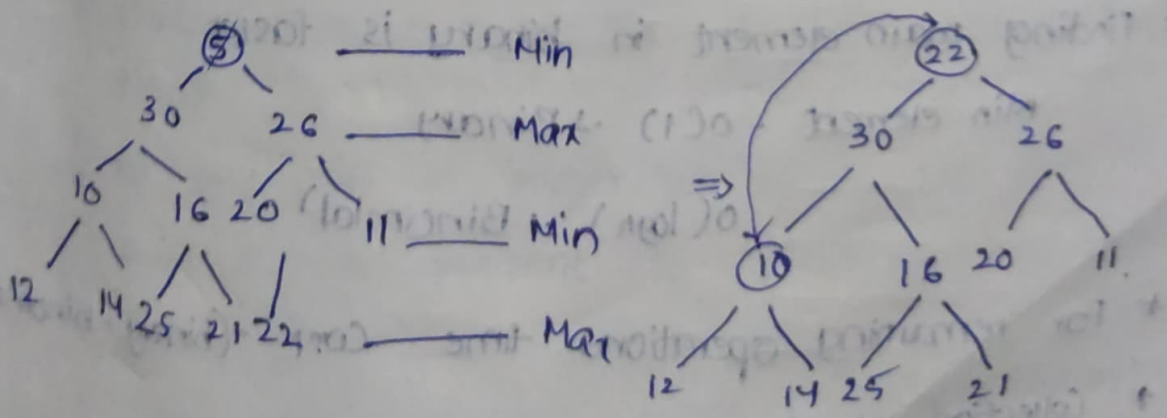
25/01/21.

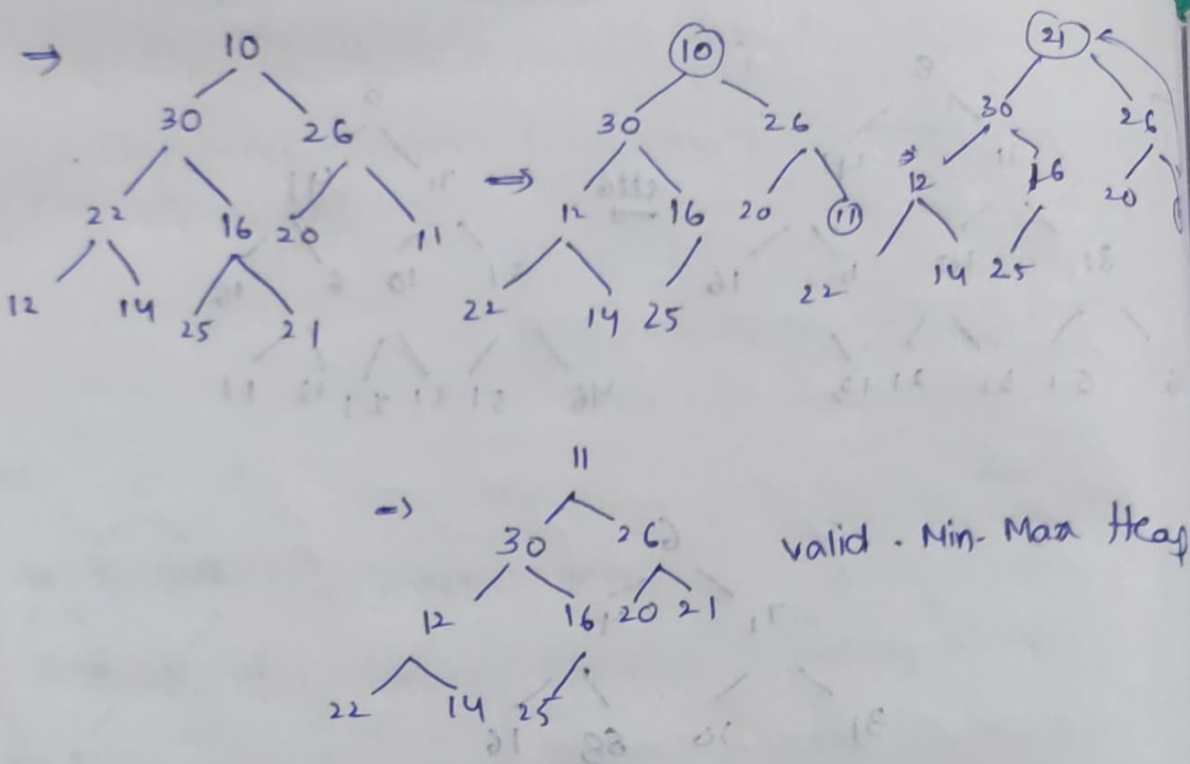
Extraction:

Extract Min:

1. Root is removed.
2. last leaf of last level is moved to root
3. Min - Max properties are checked and rearranged

Ex:





valid - Min-Max Heap

Application :-

1. Double - ended priority Queue implementation.

Assign QN: Double ended priority queue operations
How Min-Max is used here

Binomial Heap:

* It provides the faster union operation.

Union - O(n) → Binary.

- O(logn) → Binomial.

* Finding main element in binary is faster.

Min element - O(1) → Binary

O(logn) → Binomial

* For remaining operations time Comp → (binary . binomial

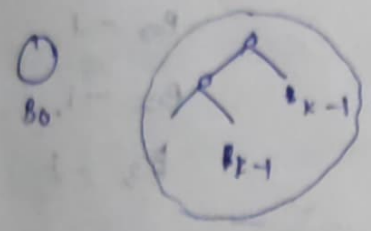
* Collection of Binomial trees.

So, Binomial heap is a forest [No specific root for Binomial heap]

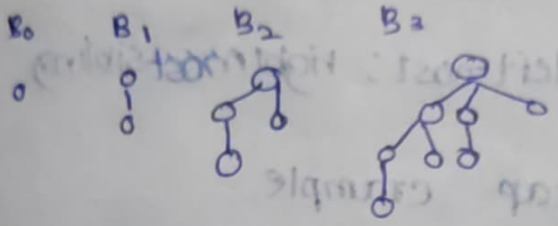
Binomial Tree : (BT) :-

BT of order → 1 node

k → 2 BT of order (k-1) each linked.



→ root of one is leftmost side of other.

* 

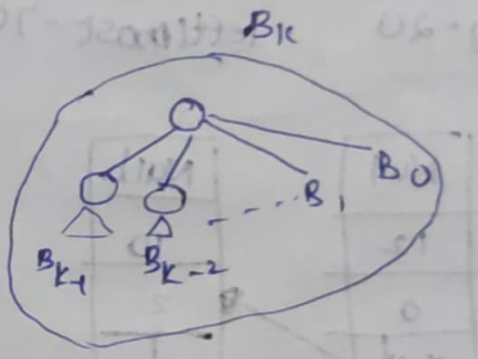
[No limit in No. of children]

* B_k :

i) 2^k values

ii) level $i \rightarrow \binom{k}{i}$ nodes

iii) k depth

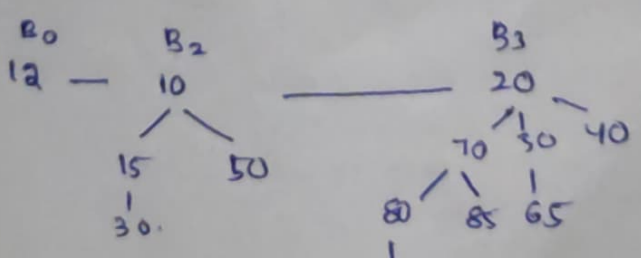


* Binomial Heap < Min (default)
Max.

· H → set of Binomial Trees → Min binomial heap.

i) Each binomial tree in the H must satisfy Min heap proper.

ii) for any non-ve integer 'k', atmost one tree of order k is present in H, whose root has degree k.

Ex: 

sorted linked representation of the roots of heap (order sorted)

- N- Nodes in Binomial Heap.

How many Binomial trees should we have and whats their order.

Ex : 13 nodes

$13 = 1\ 1\ 0\ 1 \rightarrow$ Binary rep.

$3\ 2\ 1\ 0$

$\therefore B_0 - 1$

$B_2 - 1$

$B_3 - 1$

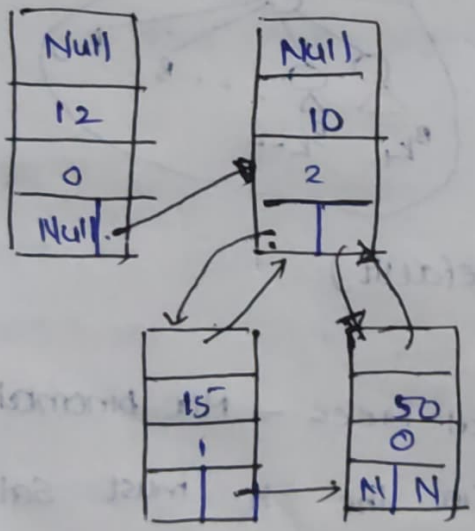
How to represent Binomial Heap:

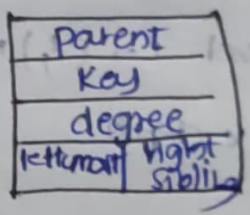
Parent; key; degree; leftmost; rightmostsibling

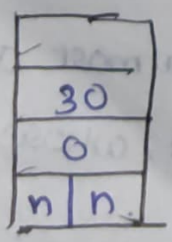
* In the min Binomial heap example

parent = null degree = 3 right = null

Key = 20 leftmost = 70.







27/1/20

* Finding Minimum:

Root list → Ascending order (of order of Binomial tree)

B_0 B_2 B_2

12 — 10 — 20

let Min = $+\infty$

B_0 → Min = 12

B_2 → Min = 10 ∴ Min = 10.

B_3 → Min = 10

Union operation:-

$H_1 \cup H_2$

Step 1:- Merge H_1 and H_2. link roots of two in ascending order

Step 2:- Same order. ptr, next -

Case 1:- Order of ptr and next

If not same → move ahead.

Case 2:- ptr, next, next → next

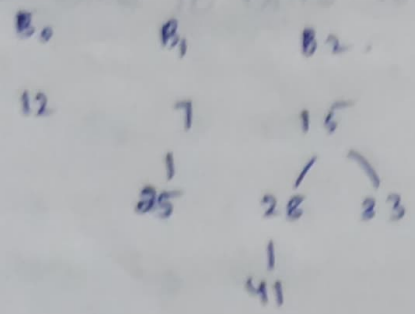
Same order for all three → move ahead.

Case 3: ptr <= next → next = child of ptr.

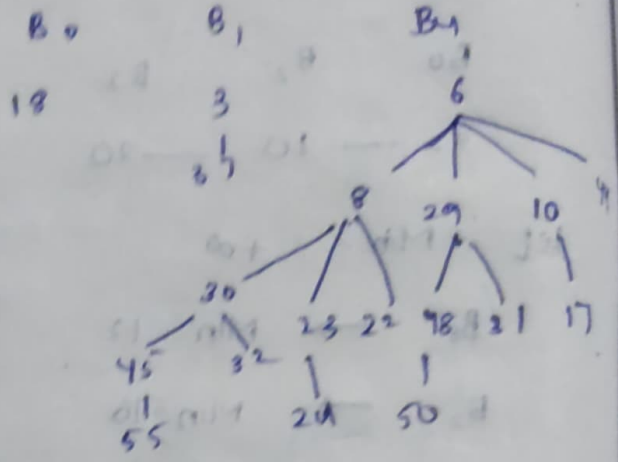
Case 4: ptr > next → ptr = child of next

Ex:-

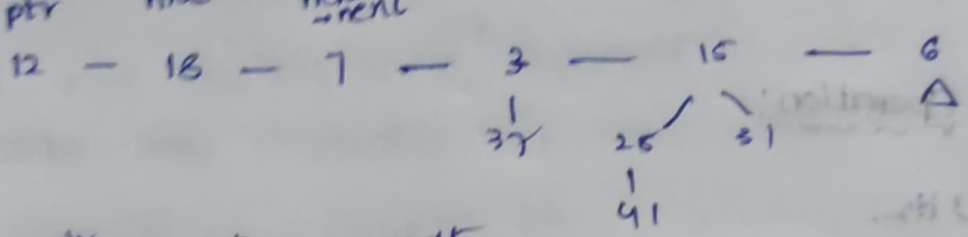
H_1:



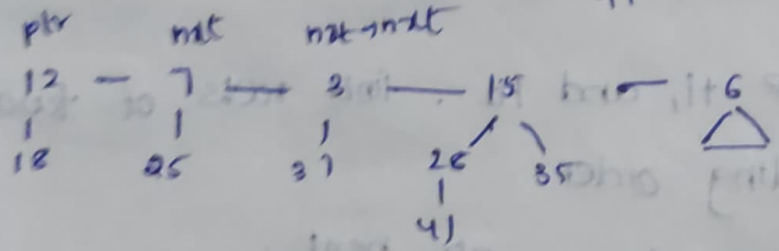
H_2



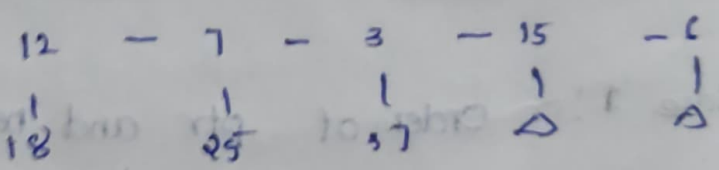
H



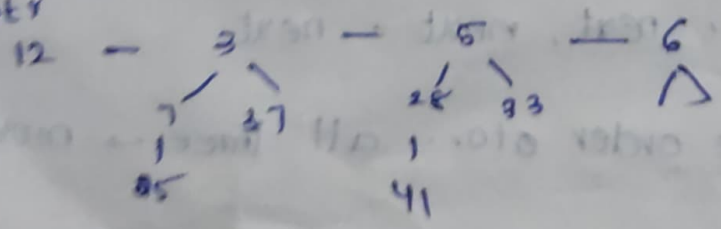
Case 2:



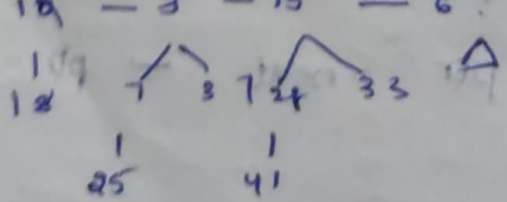
Case 0: move ahead.



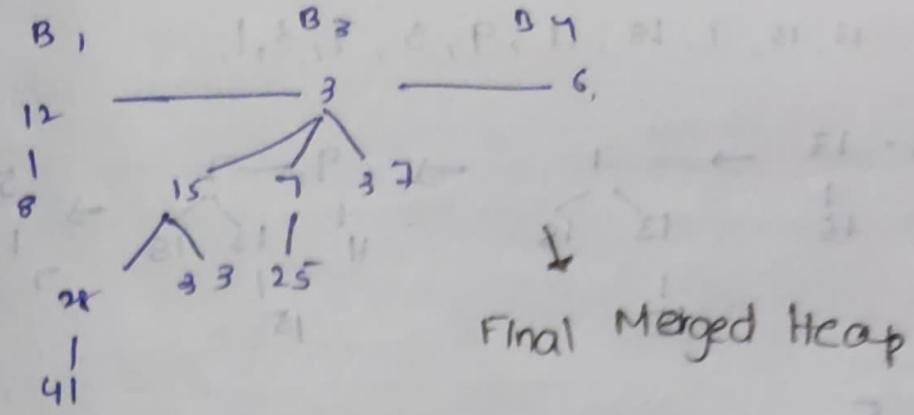
Case 4:



Case 1: Move ahead

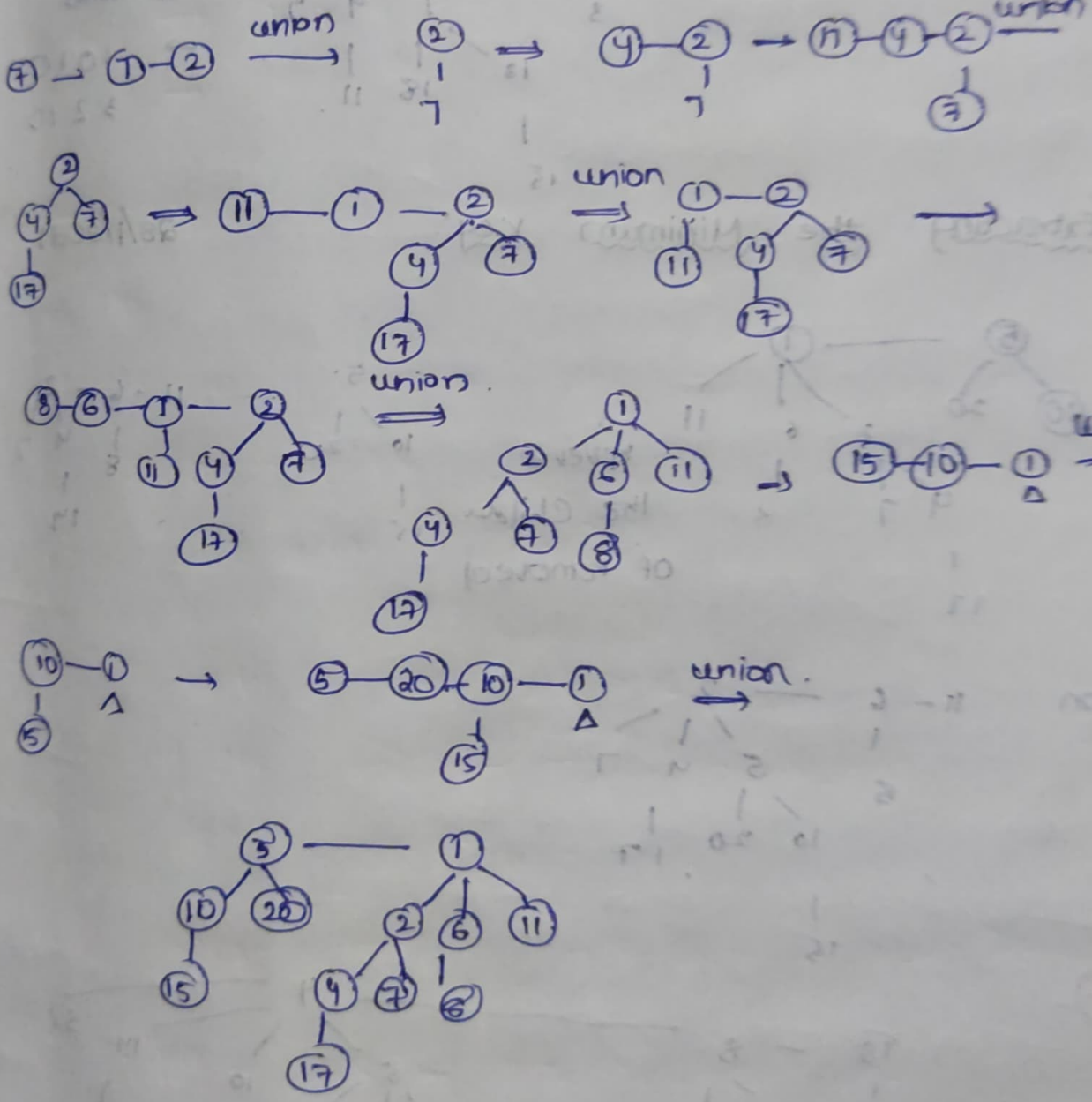


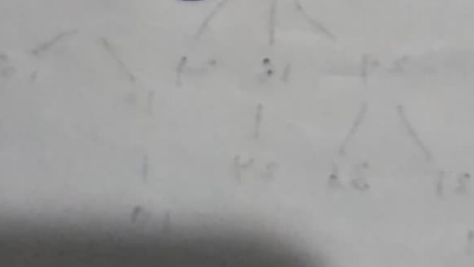
Case 3:



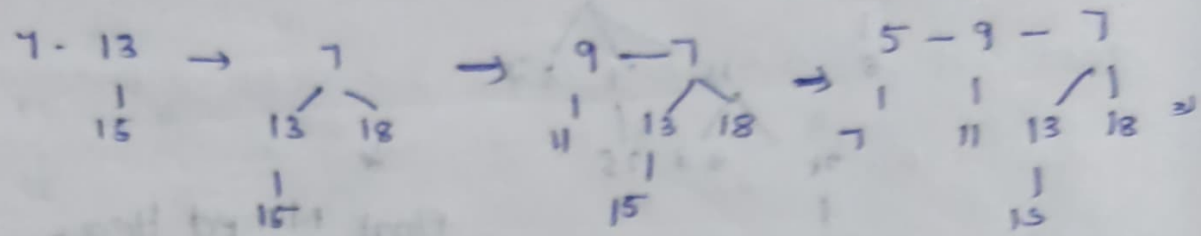
Insertion Operation:-

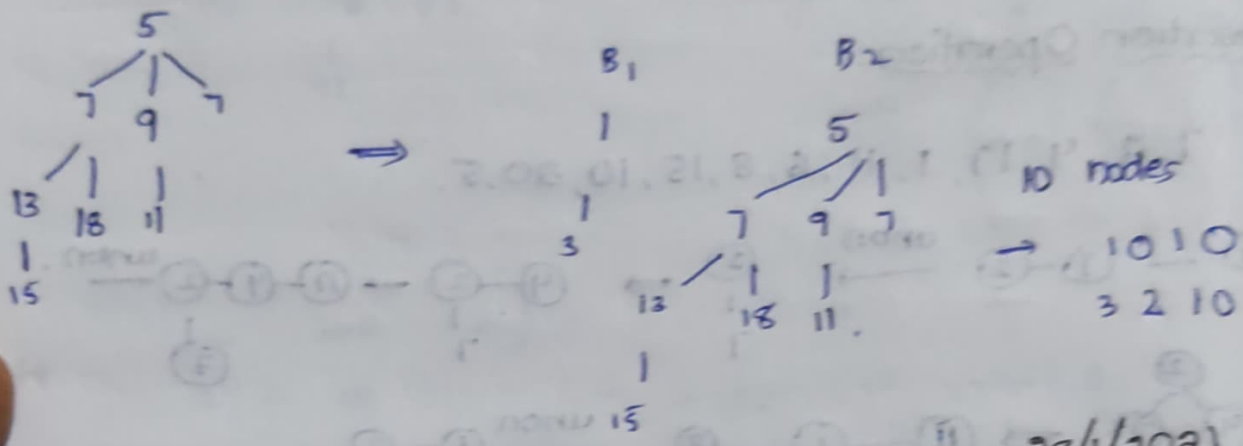
7, 2, 4, 17, 1, 11, 6, 8, 15, 10, 20, 5.



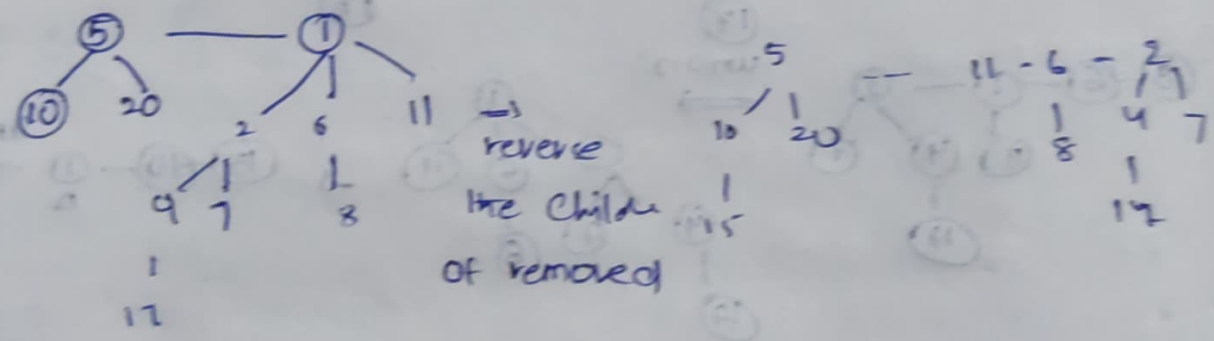


Ex: 13, 15, 7, 18, 11, 9, 5, 7, 3, 1.

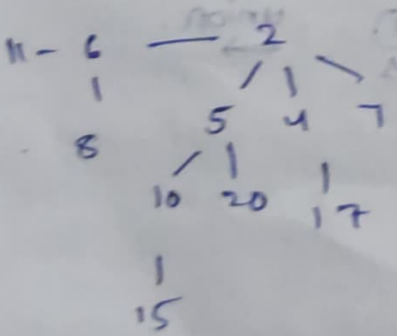




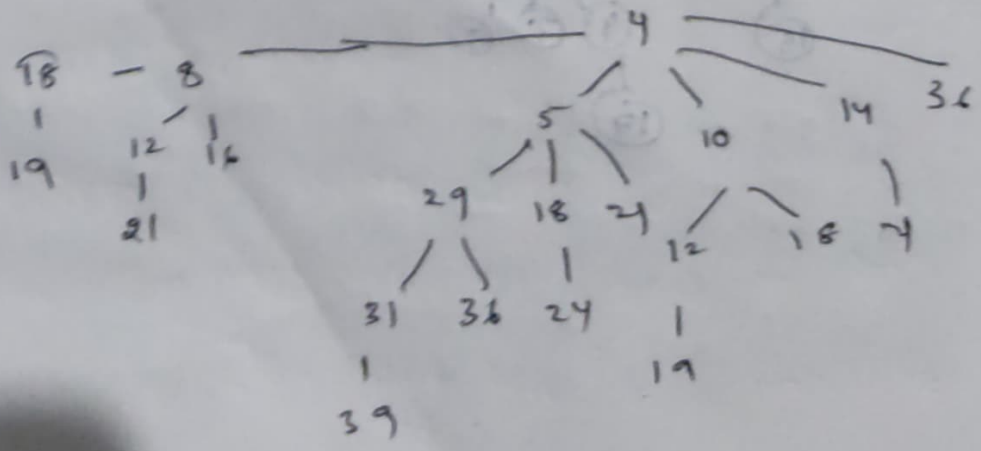
* Extracting the Minimum Key:

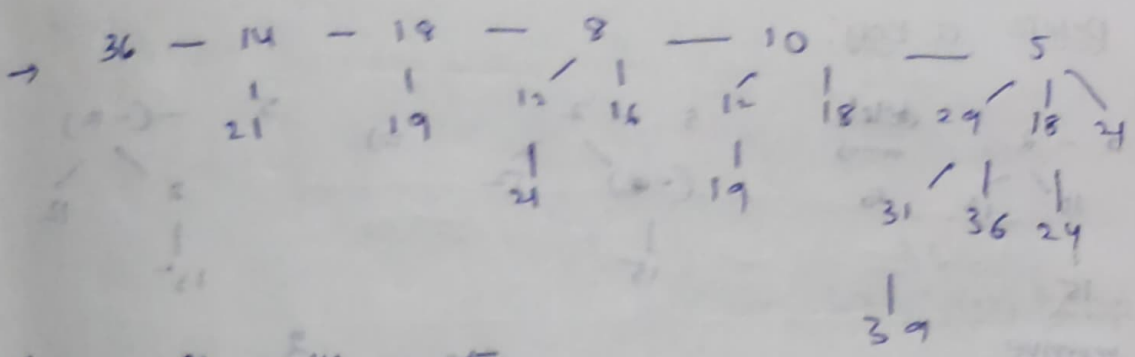


union

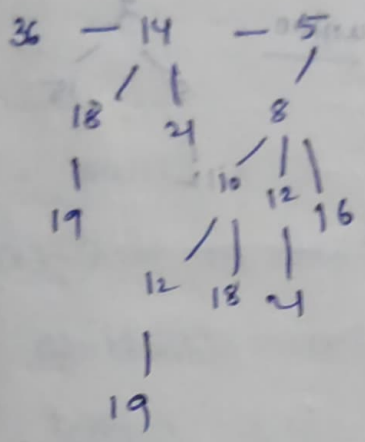


Ex:-

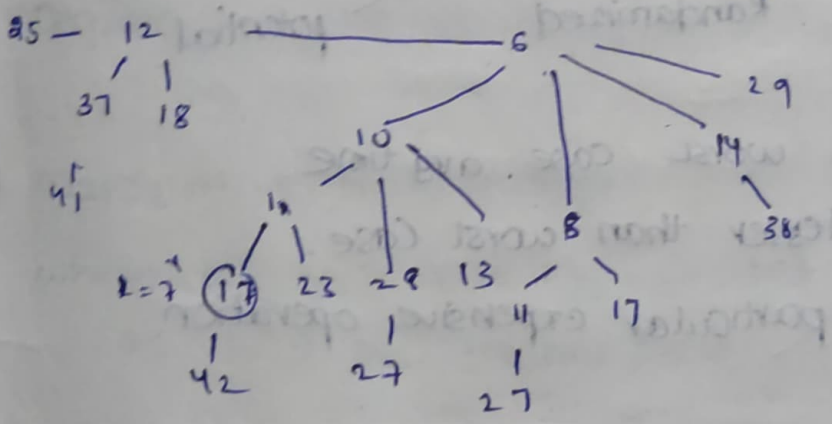




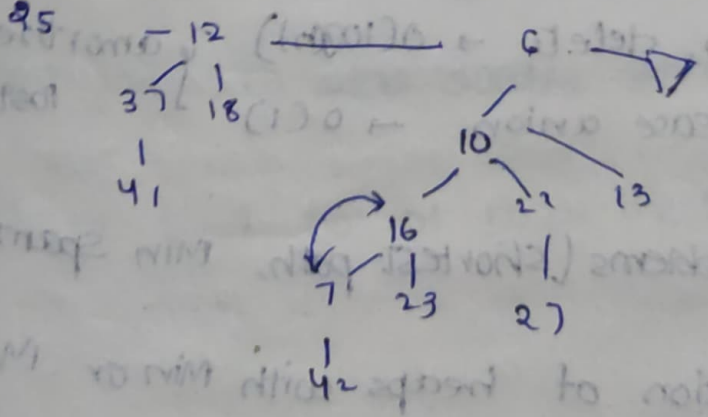
union ⇒



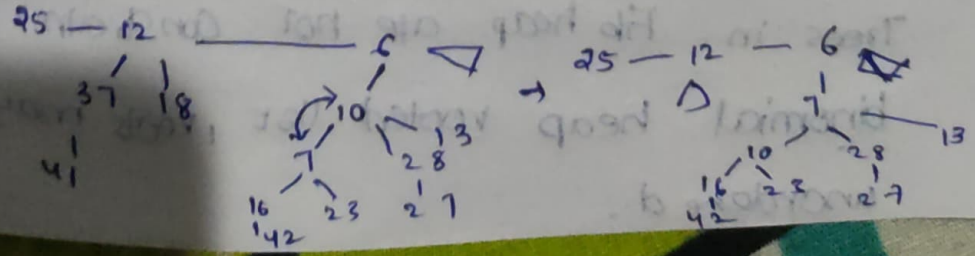
* Decreasing a key: (H, x, k)



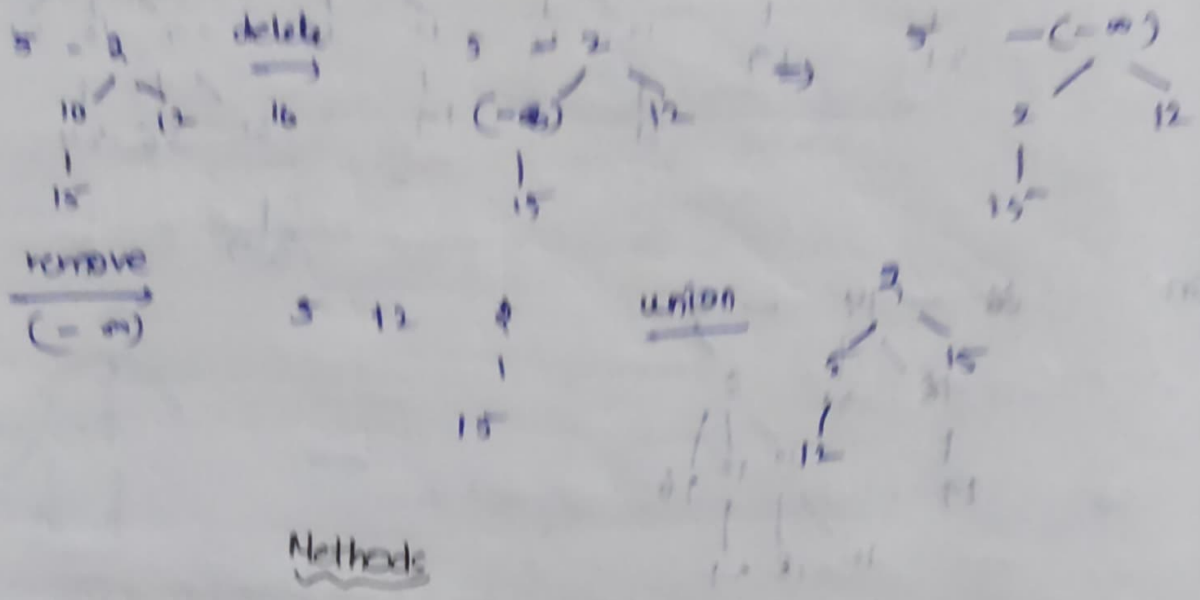
⇒



→



* Delete a key:



Methods

* For algo: Asymptotic
  - Amortized
    - aggregated
    - accounting
    - potential
  - Randomized

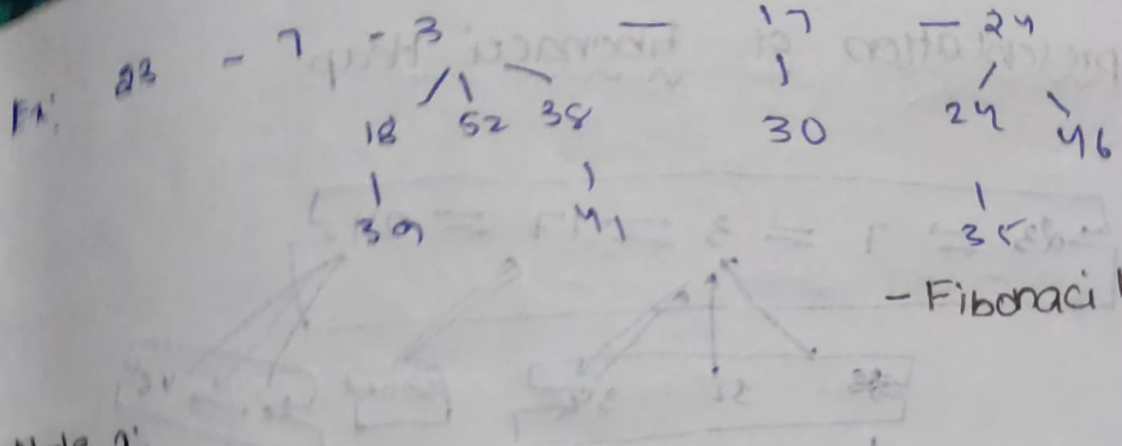
Gives worst case avg time is lesser than worst case of particular expensive operation.

Finonacci Heap:

* Extract Min, delete → O(logn) } amortized
  insert, decrease union → O(1) } last time

* Graph problems (shortest path, Min spanning tree)

* A collection of heaps with Min or Max properties
  Trees in Fib heap are not constrain through the binomial heap rooted tree, roots may be unordered.



- Fibonaci heap

Node x:

1) P(x) → parent of x.
2) child(x) → ptr to any one of x's children

childlist - circular doubly linked list

left(y) → y's left sibling

right(y) → y's right sibling

3) Degree(x) → no. of children in the child list of x.

4) mark(x) → true/false.

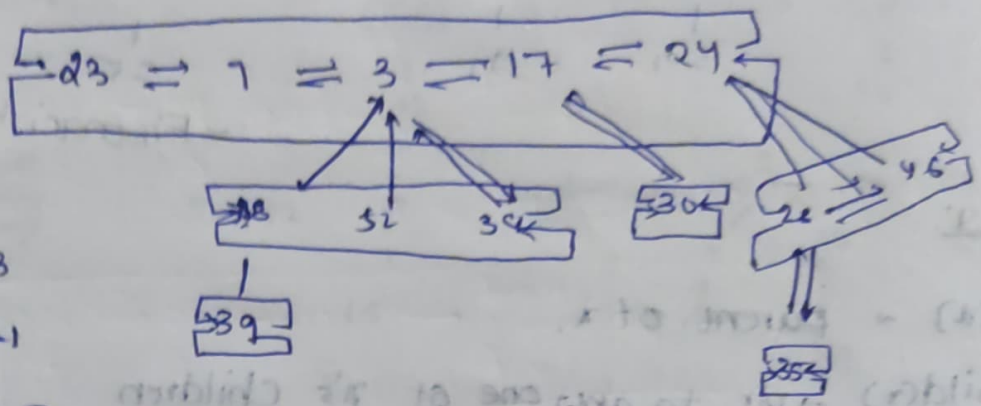
where a new node is created → always false as child of mother node

5) min(H) → root of the trees containing the min key.

root list - circular doubly linked list

6) n(H) → total no. of nodes in the Heap.

* Representation of Fibonacci Heap:



deg(3) = 3

deg(18) = 1

deg(52) = 0

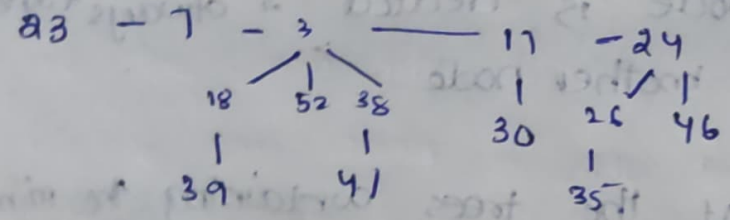
deg(38) = 1

~~01/02/24~~

1/2/21

* For fibonacci series:

$$F(n) = F(n-1) + F(n-2) \quad : \quad n > 2.$$



The trees are constructed in such a way that the tree of order 'n' has atleast F(n+2) nodes in it

Fibonacci series: 0 1 1 2 3 5 8 . . .

0 1 2 3 4 5 6 . . .

* Insertion:

- time comp → O(1)

1. create a node x.
2. If the heap is empty

→ one node in trees

1 min.

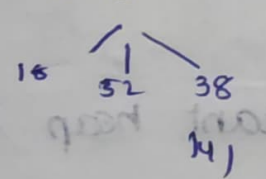
If the heap is not empty

→ into root list

(left of min)

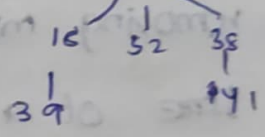
and update the min.

Ex: 23 - 7 - 3 ← min



insert 12 →

23 - 7 - 12 - 3 ← min



ex: 12, 50, 3, 15, 20.

↓min ↓min ↓min ↓min.

12 → 50 → 12 → 50 - 3 - 12 → 50 - 15 - 20 - 3 - 12

* Finding the Min:

- Returning the Min pointer.

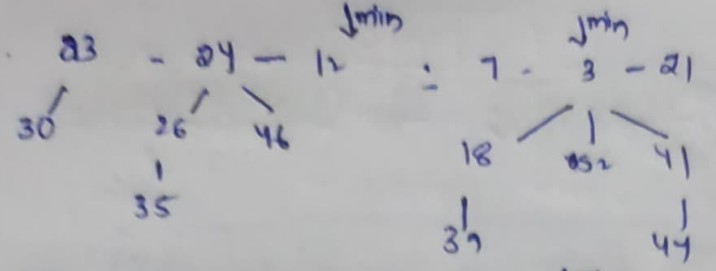
* Union Operation:

1. Merge root list H_1 & H_2.
2. Min(H_1) < Min(H_2) Min(H_2) as right of last root of H_1.

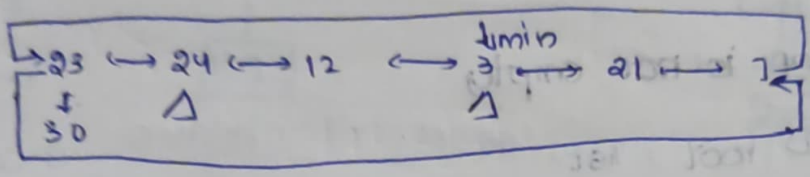
Min(H_1) = H_1.

else Min(H_1) = H_2

Ex: 23 - 24 - 12 : 7 - 3 - 21



union ⟶ 23 - 24 - 12 - 7 - 3 - 21



It is not correct in prgm point of view

temp 1 · min [H1]
temp 2 → min (H2)

temp 1 → right = temp 2.
temp 2 → left = temp 1.
temp 2 → left.

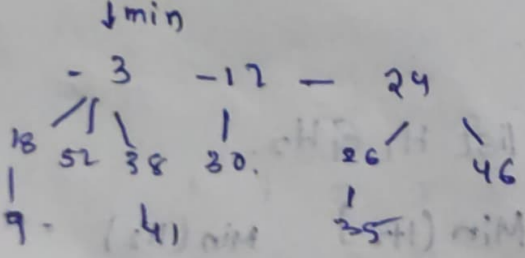
* Extract Min:

After removing min, The resultant heap should have almost three of a particular order.

1. delete min. → add its children to root list
2. set the min ptr to nxt node previously in root list.
3. Create an array of size $\log_2[n(H)]+1$
4. Map the deg to the deg in array

Map the deg to the next root

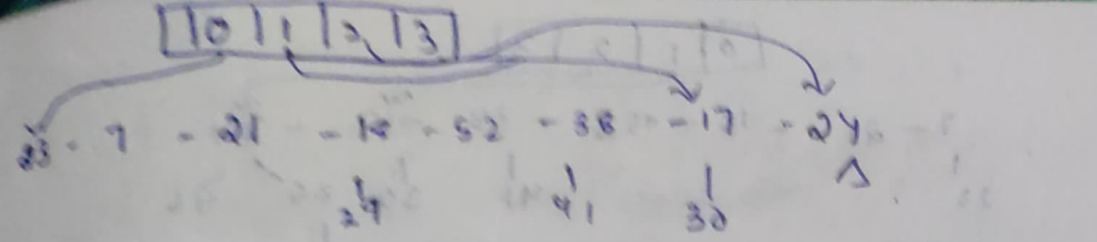
Ex: 23 - 7 - 21 - 3 - 17 - 24

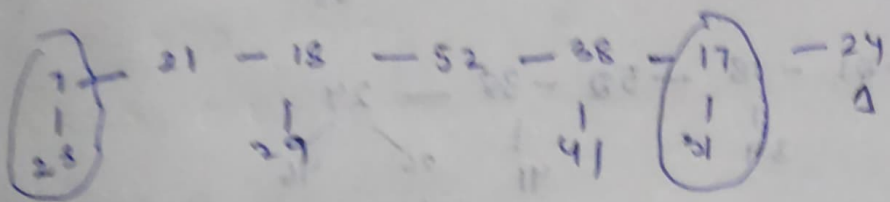


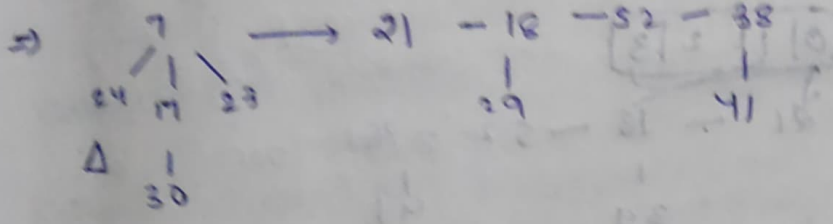
23 - 7 - 21 - 18 - 52 - 38 - 17 - 24

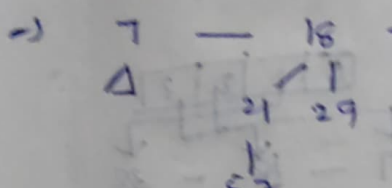
41 30

Array → $[\log_2 15]+1 \simeq 4$.



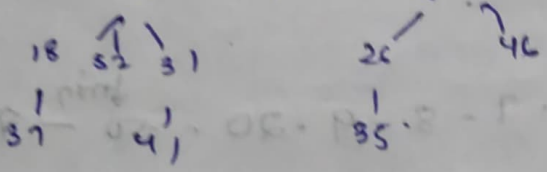
→ 

→ 

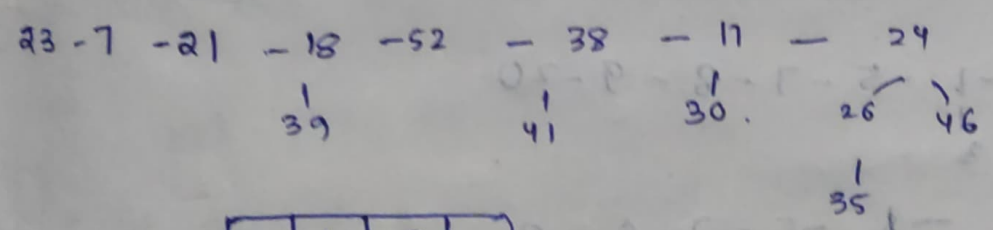
→ 

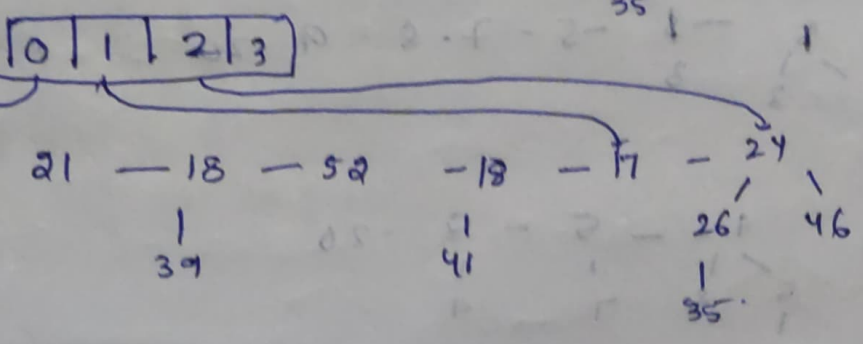
3/2/24

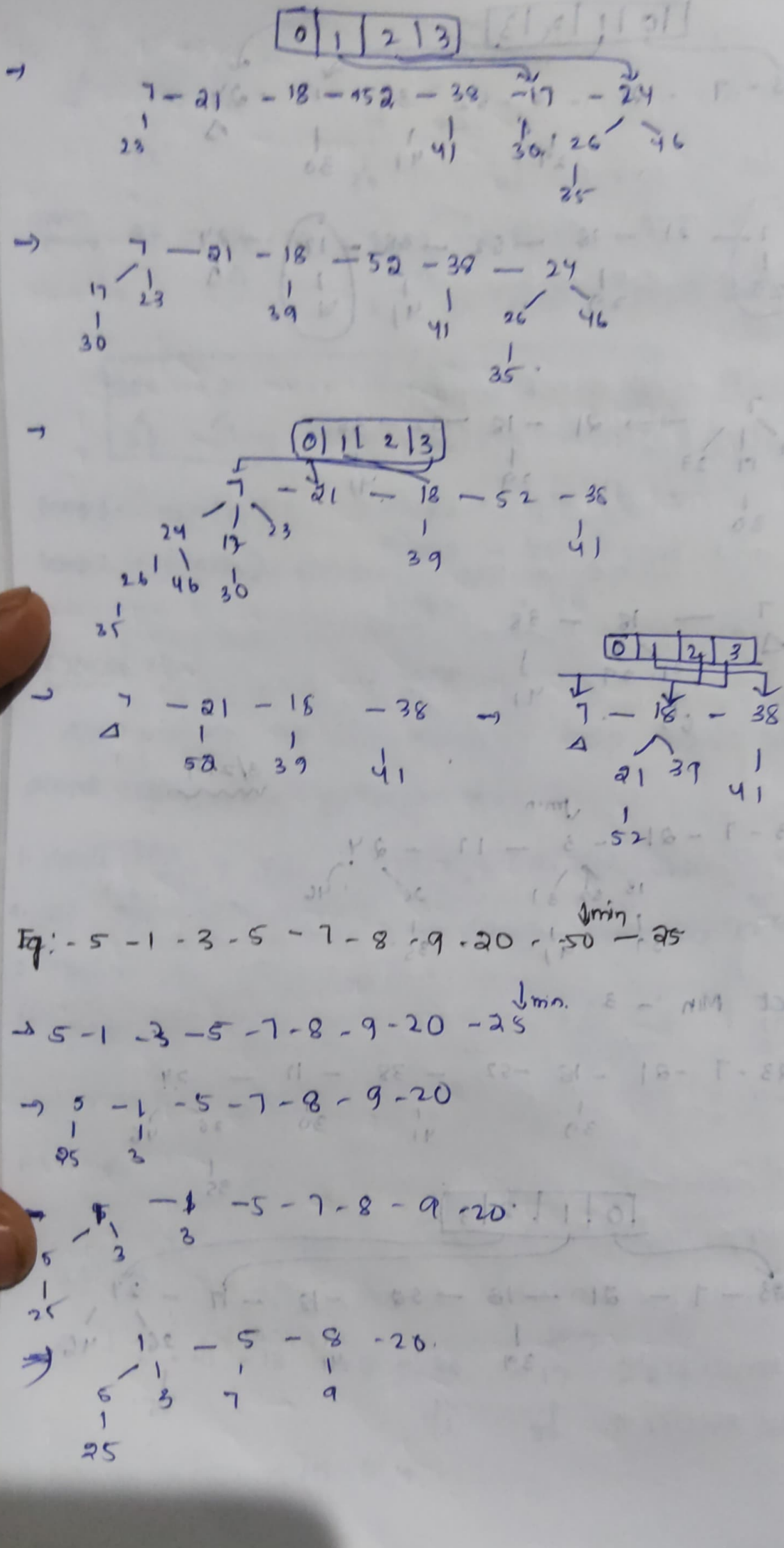
ex: 23 - 7 - 21 - 3 - 17 - 24.

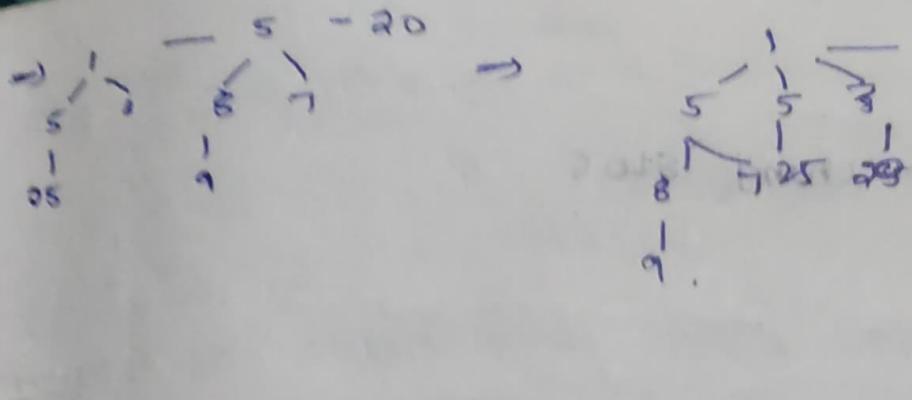


Extract Min → 3






0 1 2 3
7 - 21 - 18 - 52 - 38 - 17 - 24
23
41
30
26
46
25
7 - 21 - 18 - 52 - 38 - 24
17
23
39
41
26
46
30
35
0 1 2 3
7 - 21 - 18 - 52 - 38
24
17
23
26
46
30
39
41
25
0 1 2 3
7 - 21 - 18 - 38
52
39
41
7 - 18 - 38
21
39
41
52
Eg: - 5 - 1 - 3 - 5 - 7 - 8 - 9 - 20 - 50 - 25
min
min.
5 - 1 - 3 - 5 - 7 - 8 - 9 - 20 - 25
5 - 1 - 5 - 7 - 8 - 9 - 20
25
3
1 - 5 - 7 - 8 - 9 - 20
5
3
8
25
1 - 5 - 8 - 20
6
3
7
9
25



9 nodes.

1001

3 0.

* Decreasing a key value:-

H, x, k

case 1: min property

case 2: x and p(x) unmarked.

- cut the link btwn x & p(x)
- mark p(x)
- x is added to root list [update min].

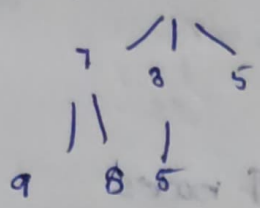
case 3: x and p(x) marked.

- cut link b/w x & p(x)
- add x to root list, min is updated.
- ~~add x to root list, n~~

* cut link b/w p(x) & p(p(x))
* add p(x) to root list
- if p(p(x)) is unmarked, mark it
- if p(p(x)) is marked.

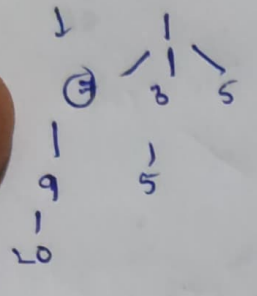
cut off p(p(x)).

Ex: 1 - 25



→ chang 8 to 6.

unmark

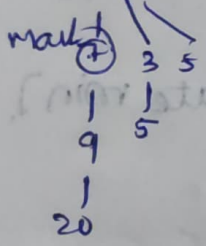


⇒ Marked 1 - 6 - 25

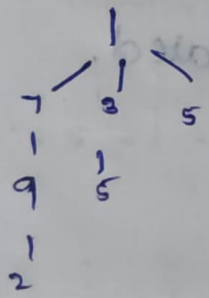
Min heap property satisfied

Valid fibonacci heap

Ex: 1 - 6 - 25



→ Chang 20 to 2.



1 - 2 - 6 - 25

mark 7 3 5

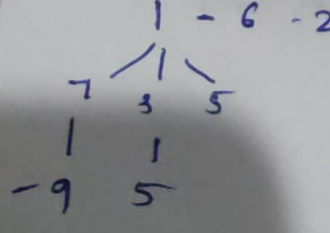
mark 9

Ex: 1 - 6 - 2 - 25

mark 7 3 5

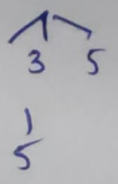
mark 9 5

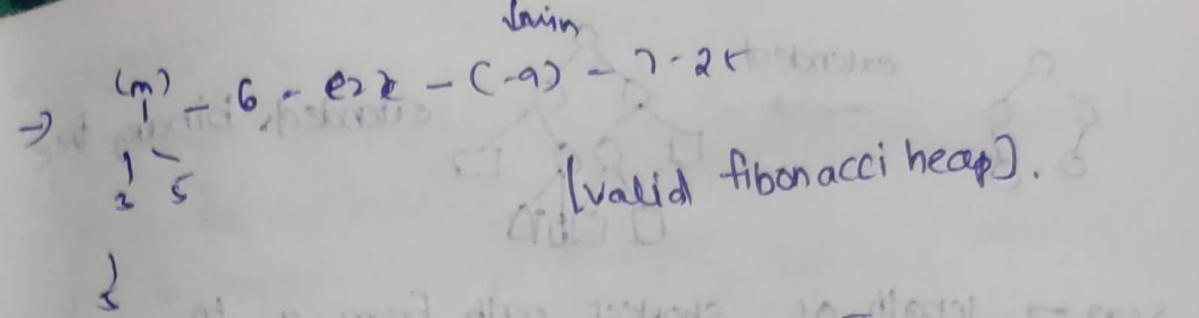
→ Chang 9 to -9



min

1 - 6 - 2 - (-9) - 25





[valid fibonacci heap].

purpose of mark → we denote status of node (it costs it's one of children)

when another children also gets removed, it moved to root list and gets unmarked.

* Delete Operation:

Delete the node required and replace it with $(-\infty)$ and then extract min [ie - extract $(-\infty)$] we get the resultant

4/2/21.

→ leftist heap:

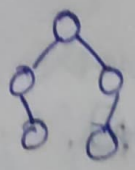
priority Queue — Min / Max.

* used for implementing Single ended priority Queue

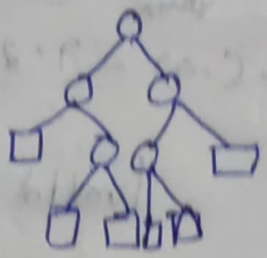
leftist — Hight biased — Min / Max
\ weight biased : Min / Max

* Extended Binary Tree:-

Binary trees with special nodes [external nodes] replaces empty subtrees.



extended →



extended binary tree

$s(x)$ → length of shortest path from x to an external HS subtree.

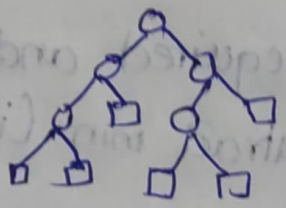
$s(x)$ — External → 0

Internal → $1 + \min\{s(L), s(R)\}$

* HBLT → $s(L) \geq s(R)$

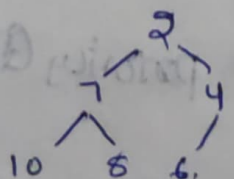
highest biased leftist tree/heap

Ex:

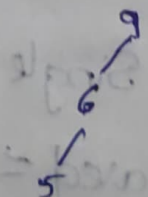


No. of x nodes in the subtree of x is atleast $2^{s(x)}-1$ internal.

* HBLT — Min
  Max.



Min HBLT



Max HBLT

operations on Max HBLT

Insertion:-

1. New max HBLT → $x \to H_1$

$H_1 \cup H_2$.

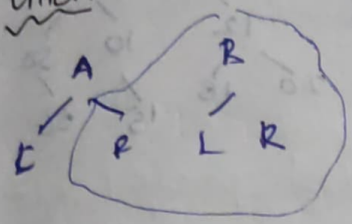
H_2 is previously HBLT to which we add x.

* Deletion:

Deletion of Max.

- remove root
- union of left & right subtree

* Union:



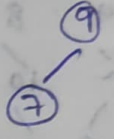
→ Max HBLT

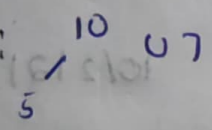
A ⊃ B.

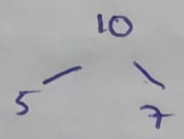
If A > B ⇒ combine R of A with B.

Ex: (9) ∪ (7) → (9) Max satisfied

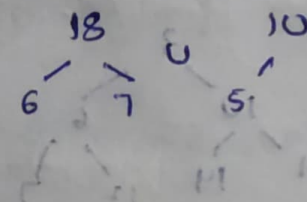
(7) HBLT not Satisfied → change position

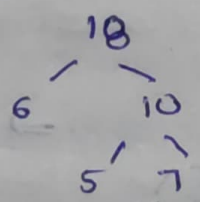


Ex: 

→ 

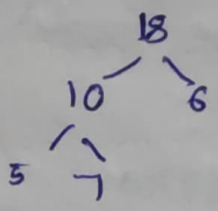
Max HBLT satisfied.

Ex: 

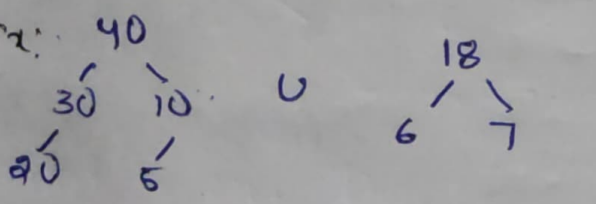
→ 

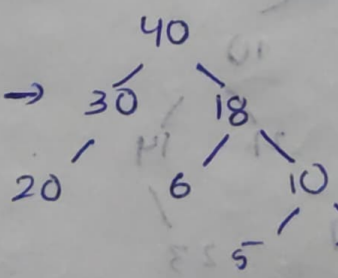
Max Satisfied

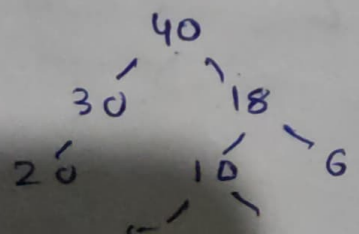
HBLT not satisfied

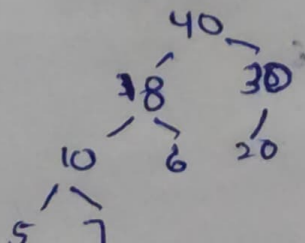
for 18, HBLT is not satisfied → 

interchanging subtrees of 18.

Ex: 

→ 

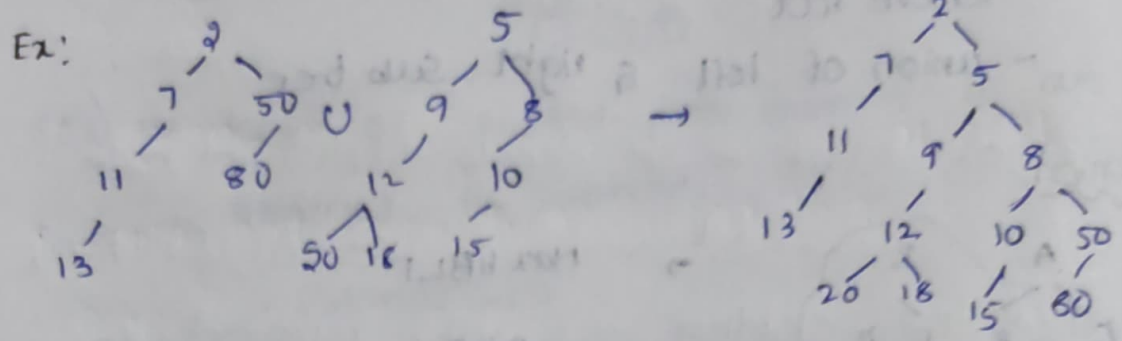
→ 

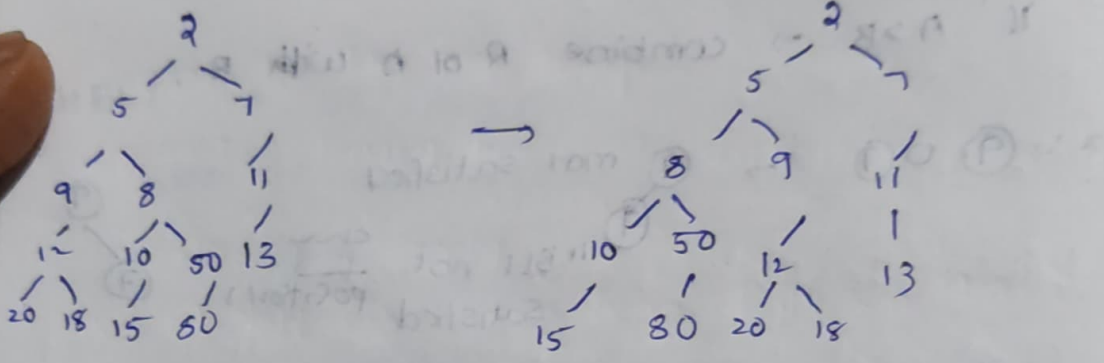
→ 

Top to bottom parse - for merging

bottom to top parse - to swap subtrees

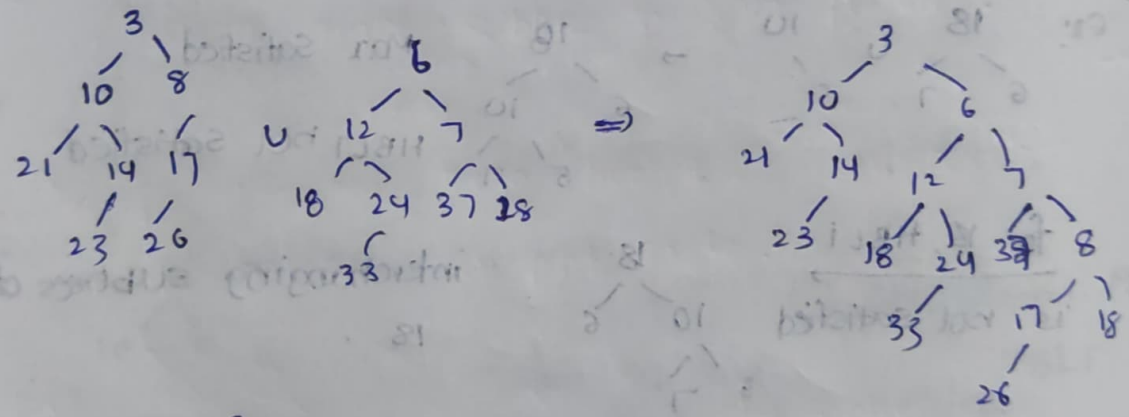
Ex:

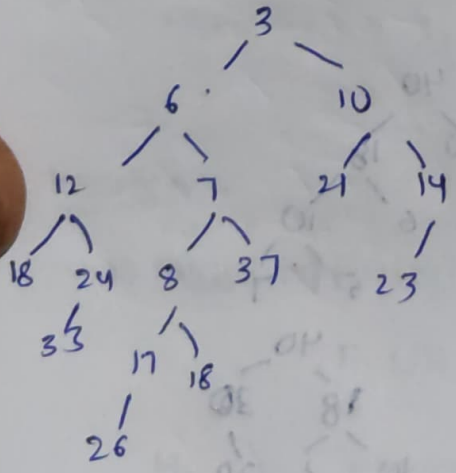




10/2121.

Ex: HBLT (Min):

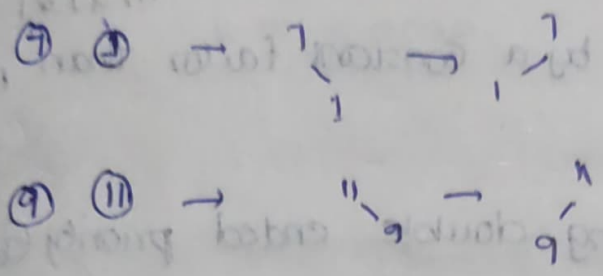


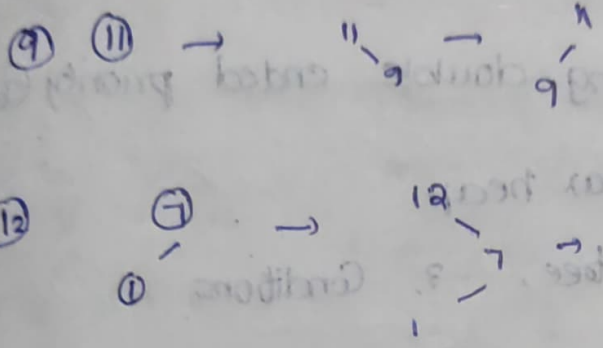


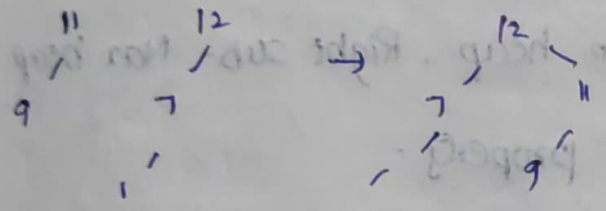
→ Intialization: [HBLT, Max]

* Creates simple array
  each node as a seperate HBLT.

Ex: 7, 1, 9, 11, 12







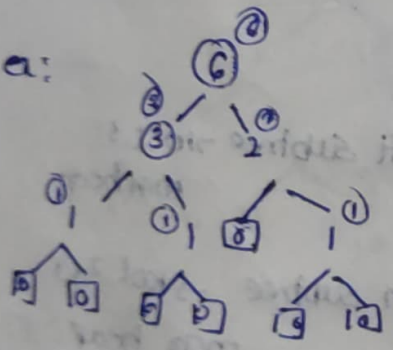
Final HBmaxLT for given number

Weight Biased leftist tree:

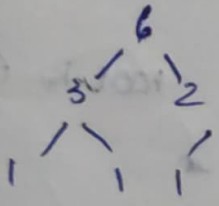
w(1) - 0 - external nods.

If Sum of weights of its Children → internal nods

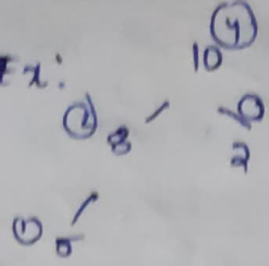
a:



w(l) > w(r) → weight biased leftist tree.



→ weight biased

Ex: 

→ weight biased.

DEAPS:-- 11/2/21.

* simpler and faster by a constant factor than minmax heaps.
* used for implementing double ended priority Queue.
* similar to Min-Max heap.

complete binary tree, 3, Conditions:

i) Root is empty.

ii) left sub - Min heap. Right sub - Max heap

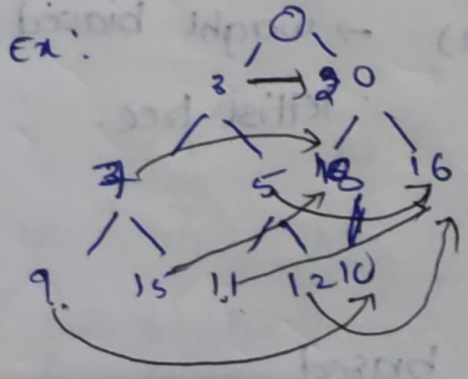
iii) Correspondence property.

x in left subtree, its correspondant node is

y in right subtree in same position.

$x \leq y$.

If 'y' is not there then it's corrosponding node is its parents correspond node.

Ex: 

for

30 → 20

7 → 18

5 → 16

9 → 10

15 → 18

11 → 16

12 - 16

left subtree → root 3 min heap

high subtree → root 20 max heap

root is Empty-

x	3	7	5	9	15	11	12
y	20	18	16	10	18	16	16

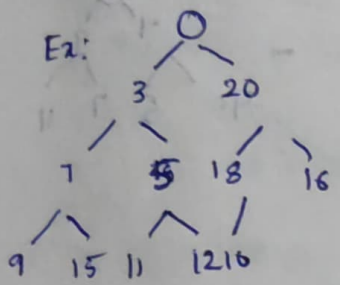
$x \leq y$.

All conditions are satisfied

∴ it is a deap.

* Inserstion:-

Ex:

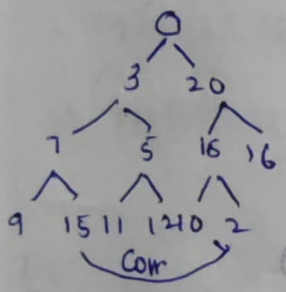


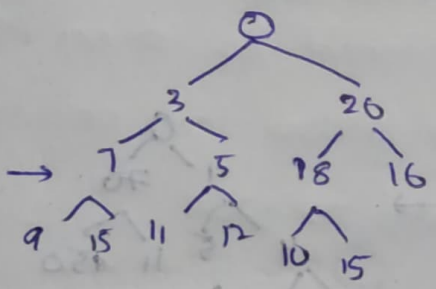
i) As it is complete BT we have to insert at last level.

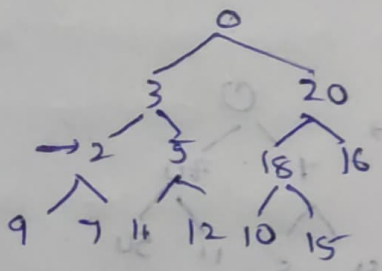
ii) After inserting check the correspond ding property.

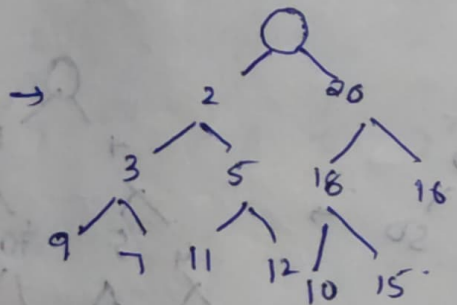
iii) After correspondence we have to check heap property.

2.

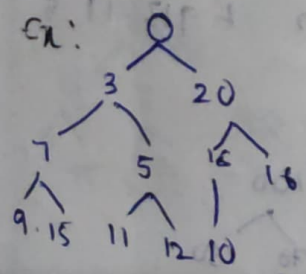


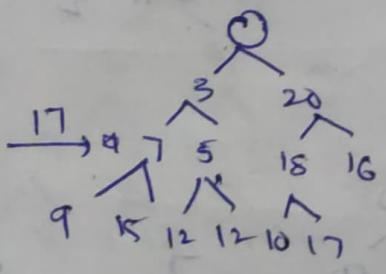






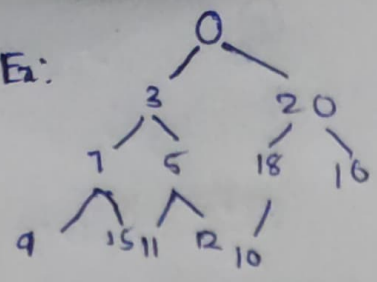
Ex:

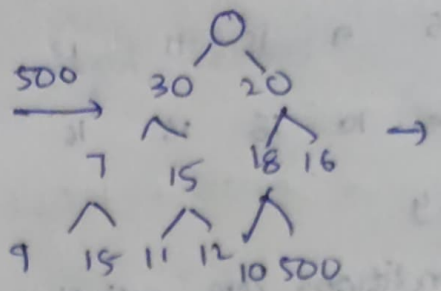


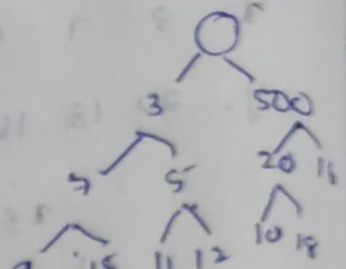


All properties satisfied

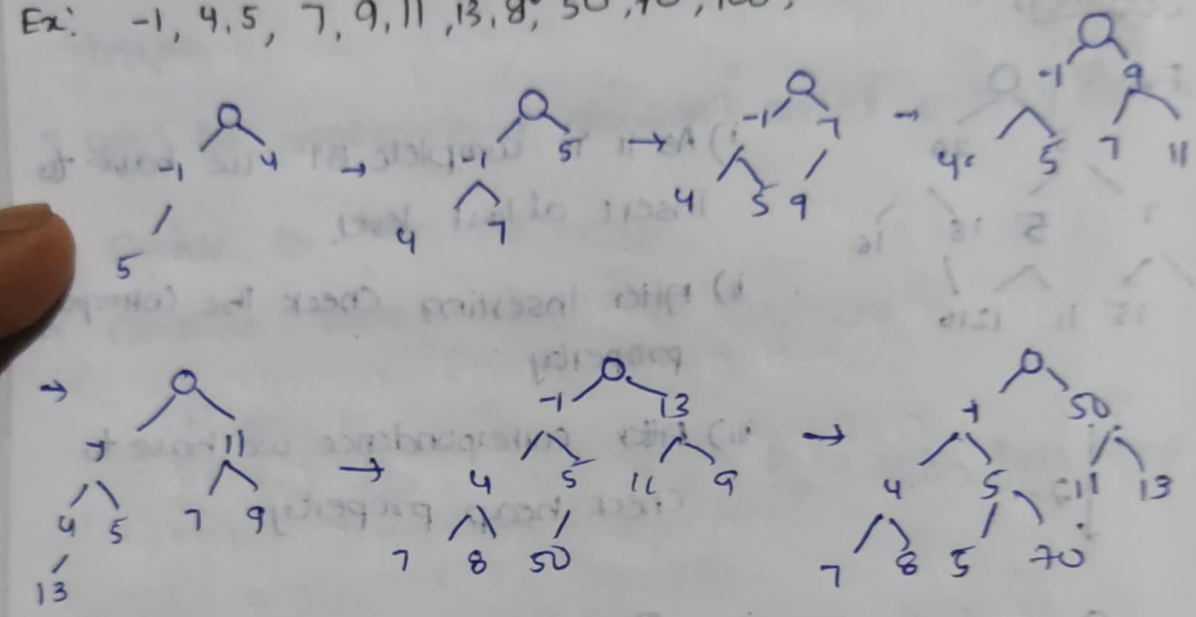
Ex:

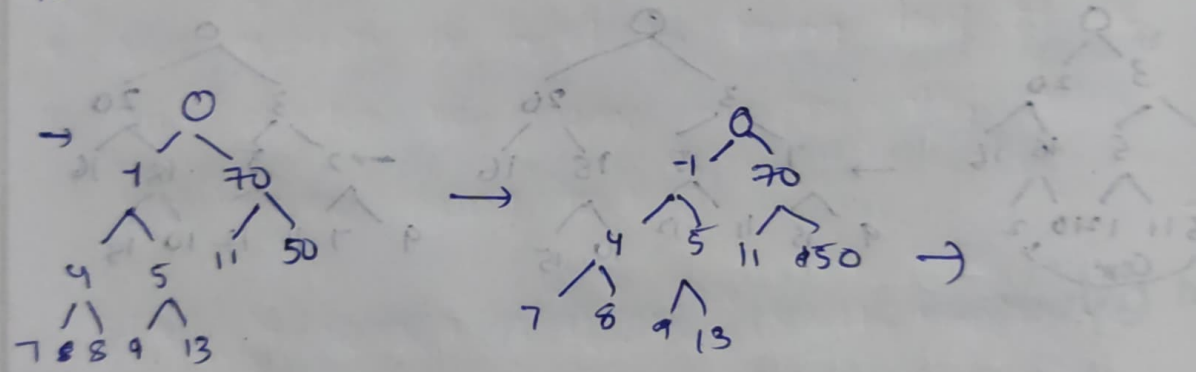


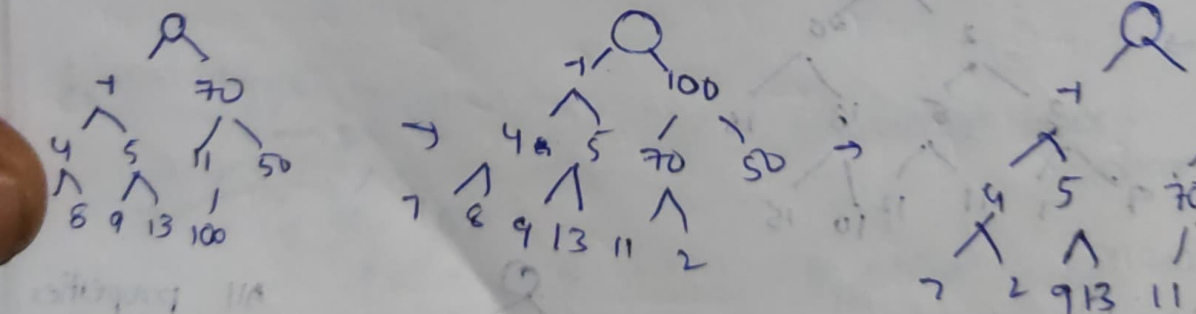


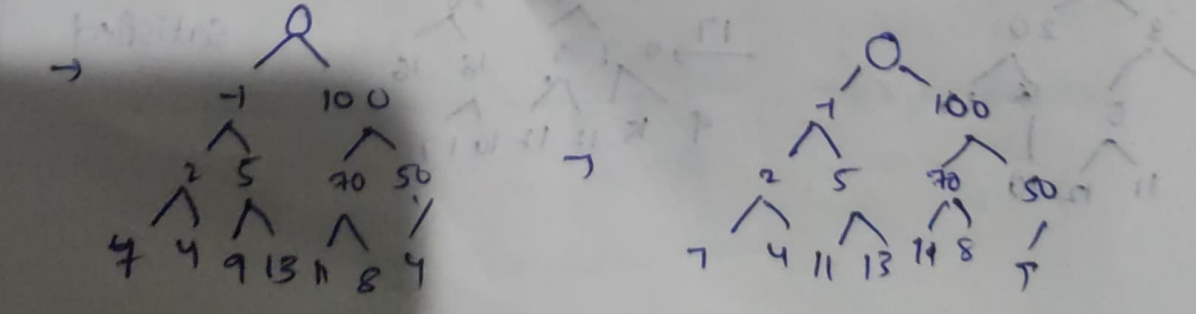


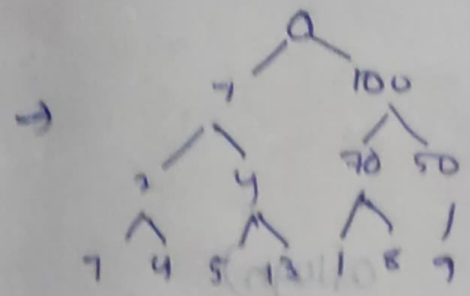
Ex: -1, 4, 5, 7, 9, 11, 13, 8, 50, 70, 100, 2, 4.



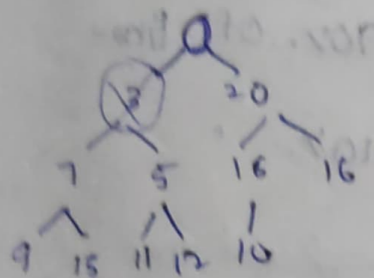




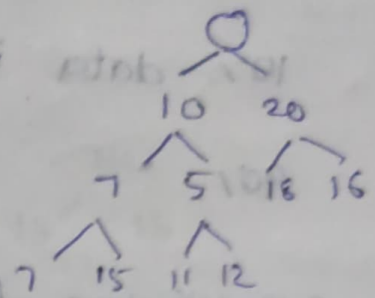


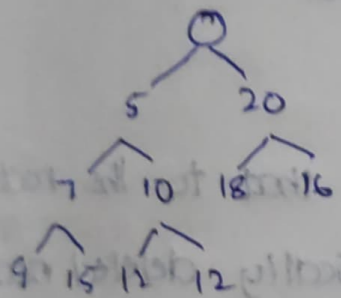


* Deletion: (Extract Min)



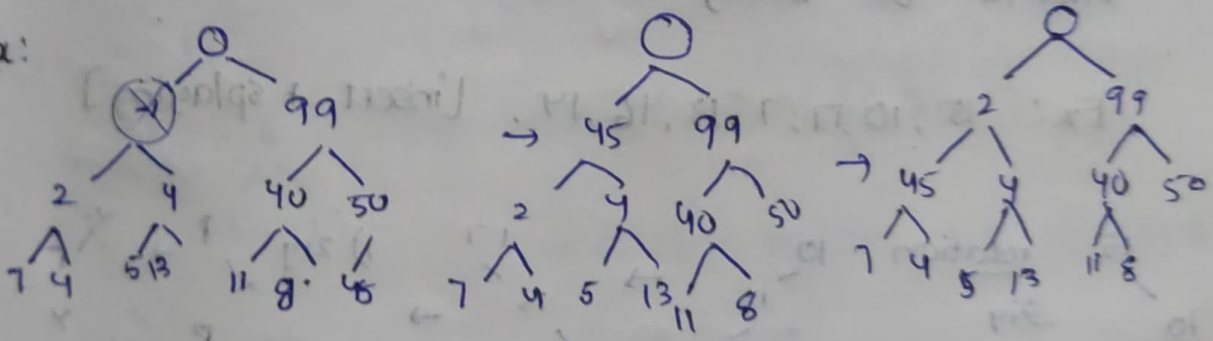
bring the last leaf at the deleted position

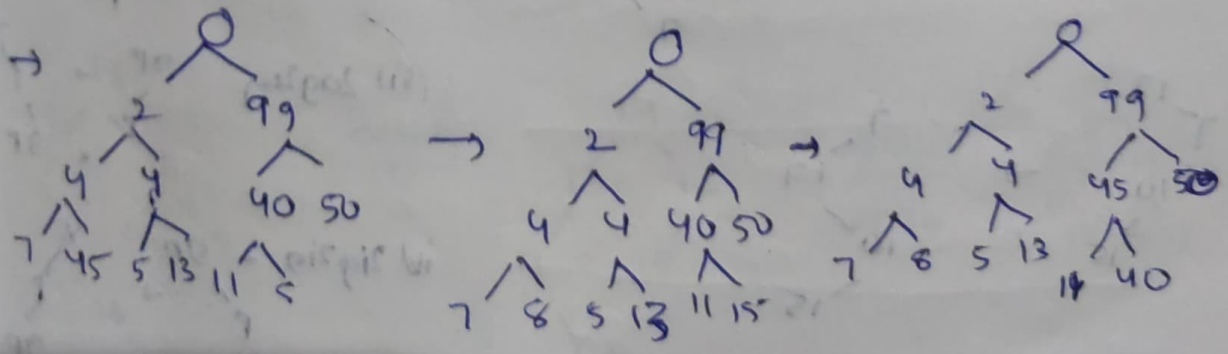




all properties are satisfied.

Ex:





Unit - 2

→ Splay Tree:

* Splay Tree → worst case O(logn)
* Roughly balance as compared to RB, AVL Trees
* Based on principle of locality.

  10% data is accessed 90% of time

  90% " " 10% " "

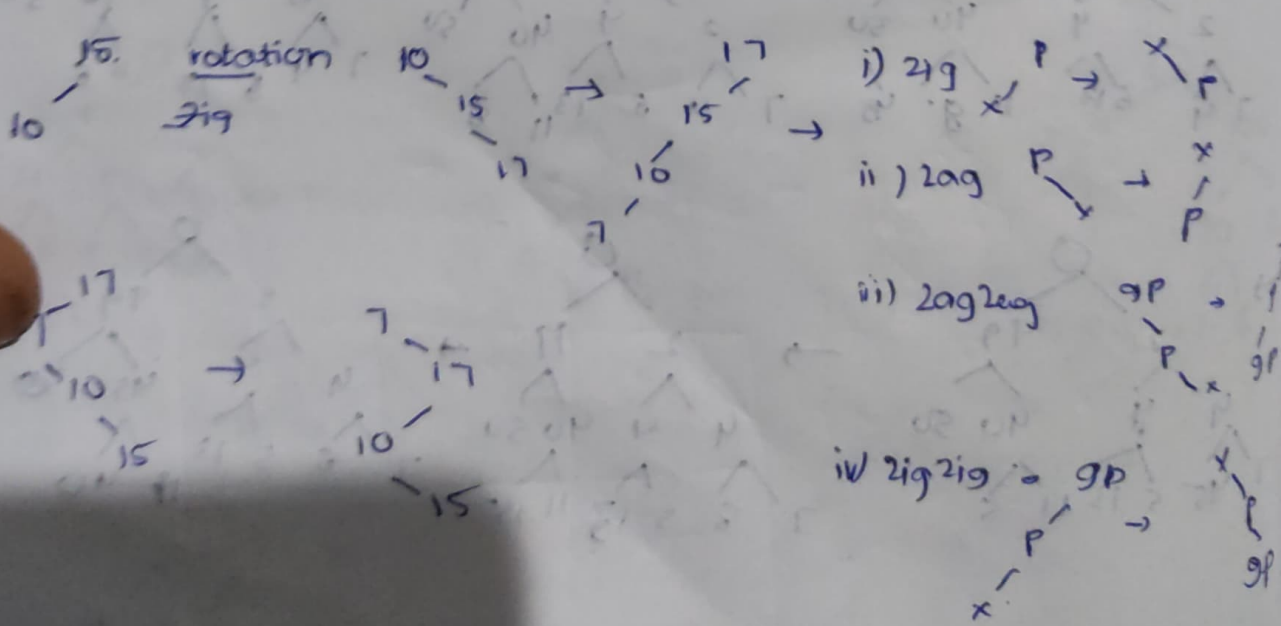
* Self adjusting Binary search tree:

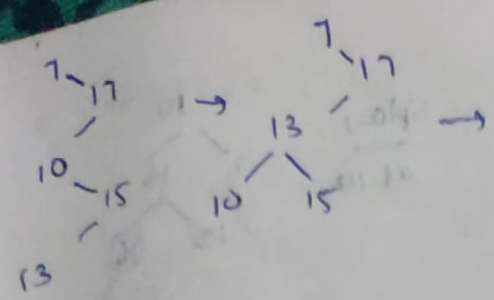
  After every insertion, splay operation is done

  splay → rotation sequence.

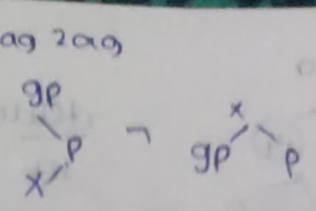
* After insertion, we need to bring that to the root.
* After deletion, the parent of physically deleted node should be bring to root.

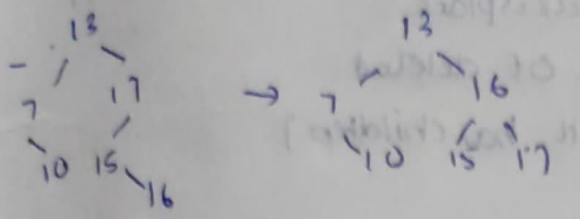
Ex: 15, 10, 17, 7, 13, 16, 14. [insert + splaying]



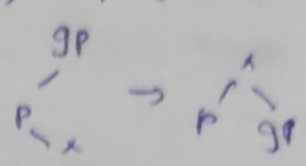


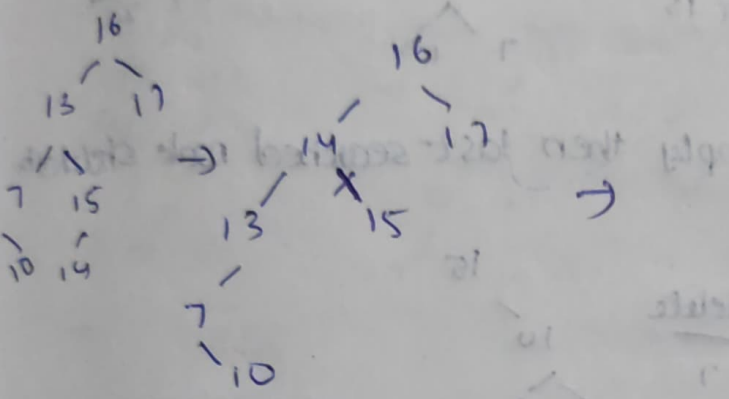
v) zag zag





vi) zig zag

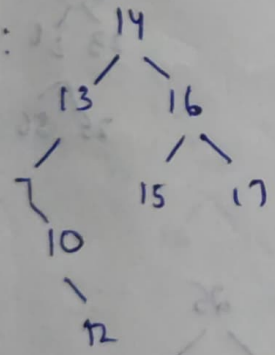




* Deletion:-

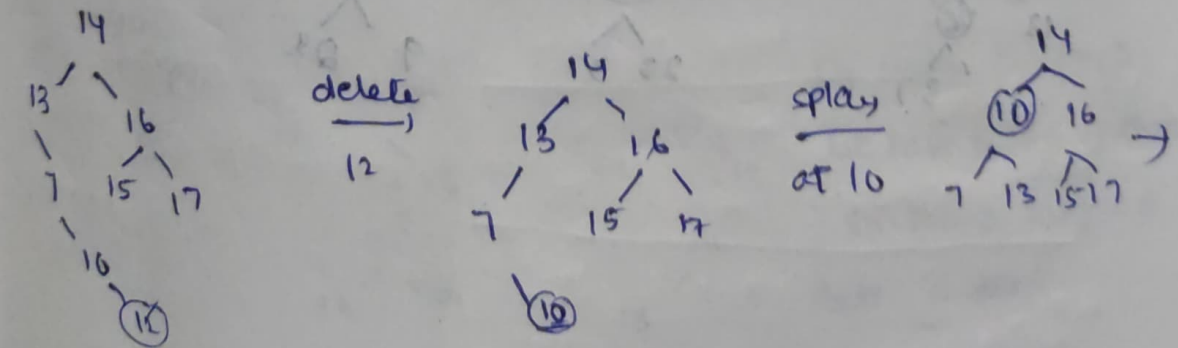
Splaying done at parent of deleted node.

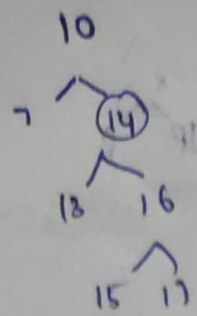
Ex:



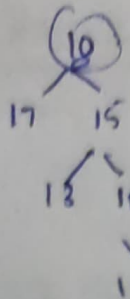
delete 12, 14, 16, 20, 17

default - bottom-up approach is followed

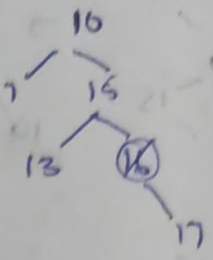




delete 14

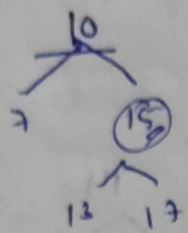


Splay at 10

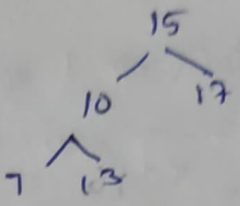


[Successor takes place the place of deleted node if it has children]

delete 16



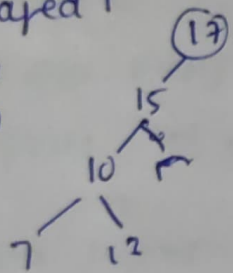
Splay a 15



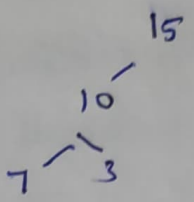
delete 20

(ended up with empty then last searched node should be splayed)

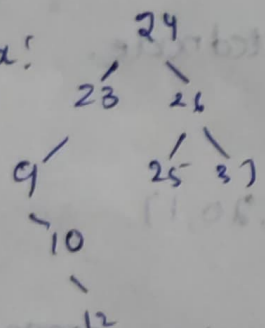
Splay at 17



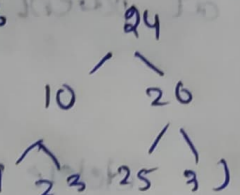
delete 17



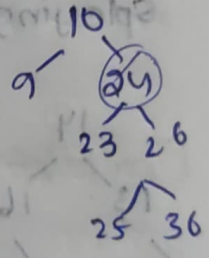
Ex:



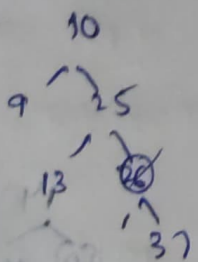
12, 24, 26 →



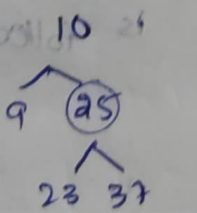
→



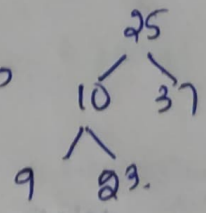
→



→

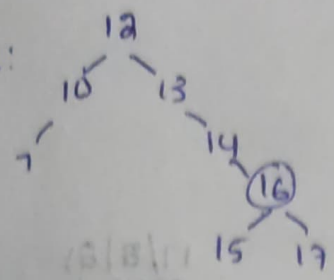


→



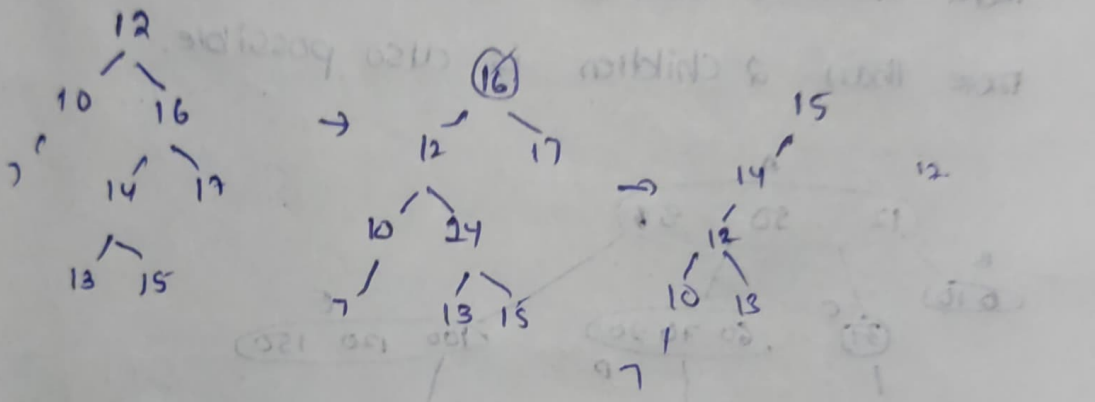
* Deletion :- (Top-down splaying)

Ex:

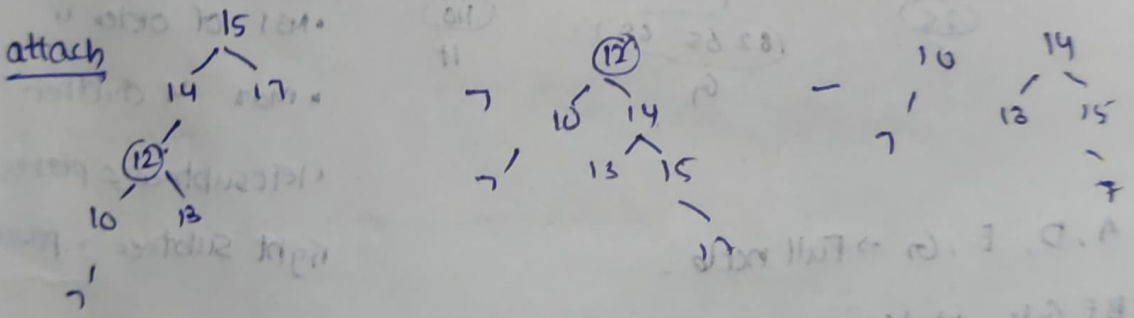


16, 12, 7, 17

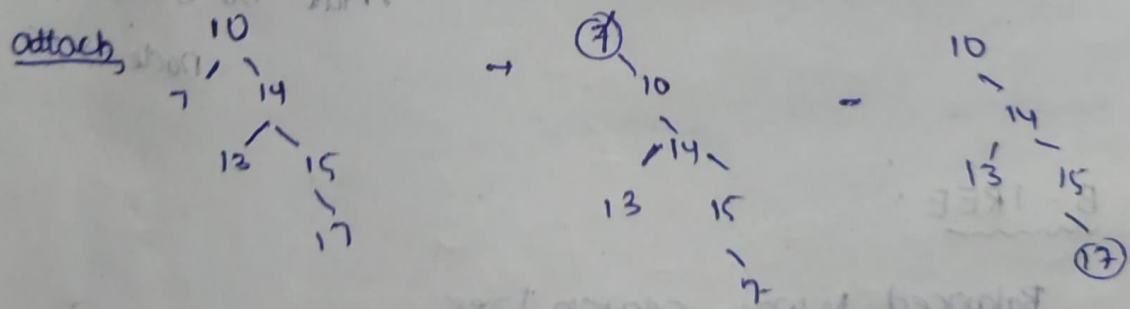
The node to be deleted is bring the root and deleted then merge left & right subtrees. Before merge, largest in left is bring to the root.

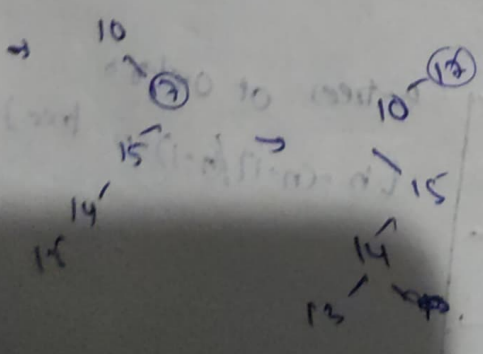


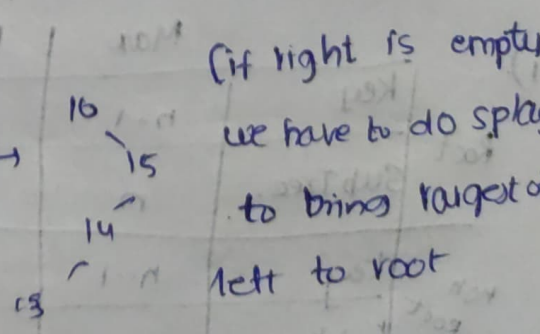
attach



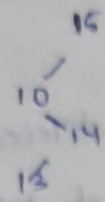
attach







(If right is empty we have to do splaying to bring largest of left to root

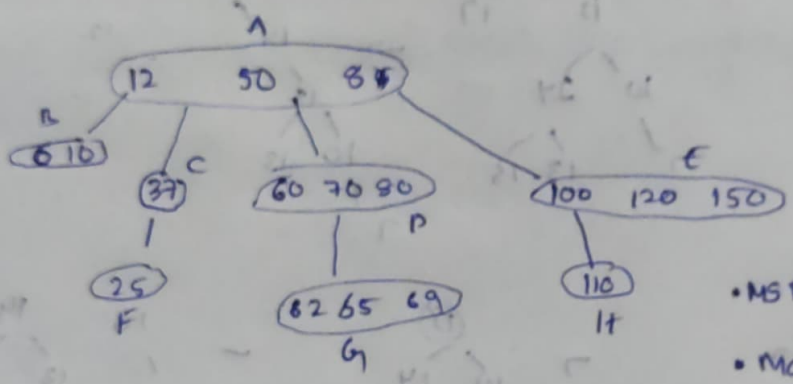


final splaying tree.

17/8/21

- Multiway search Tree:

More elements can be stored in each node. more than 2 children is also possible.



- MST of order 4
- Max 4 children
- left subtree ≤ parent
  right subtree ≥ parent

- A, D, E, G → Full node.
- B F G H → leave.

- Max no. of key per node → 3

* B-TREE:

Balanced M-way search Tree.

1)

		Max	Min
Root	Key	n-1	1
	SubTree	n	2
non Root	Key	n-1	$\frac{n-1}{2}$
	Subtree	n	$\frac{n+1}{2}$

B-trees of order n
[n-(n-1)/(n-1)-n tree]

2. All leaves should be in the same level.

Ex. B-tree of order 3

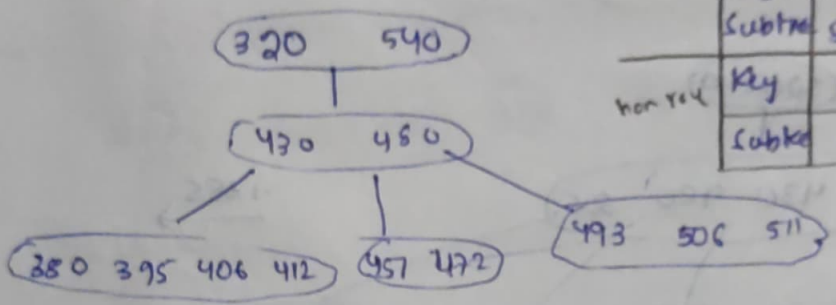
4-5 tree (or) 5-4 tree

→ Insertion

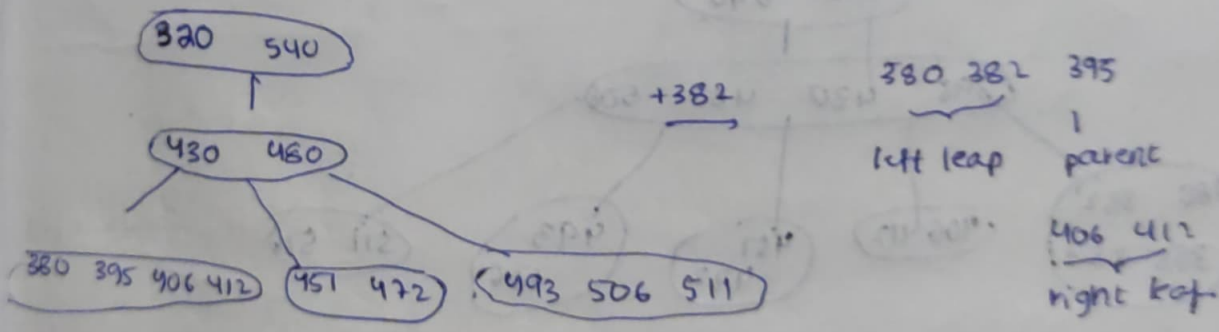
- Identify whether the node is full node or not
- If it is full, then split the elements

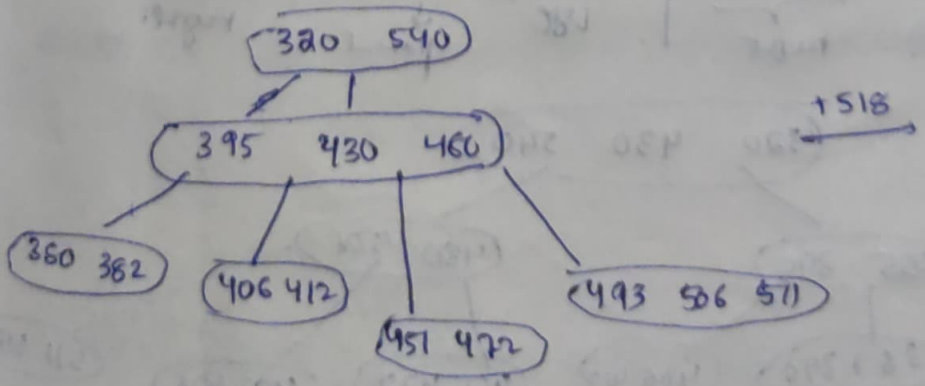
Ex: B-tree of order 5

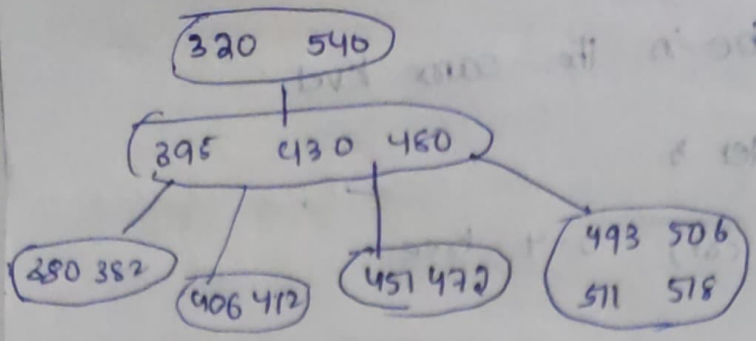
		Max	Min
root	Key	4	1
	Subtree	5	2
non root	Key	4	2
	Subtree	5	3



Order 5 → odd number.

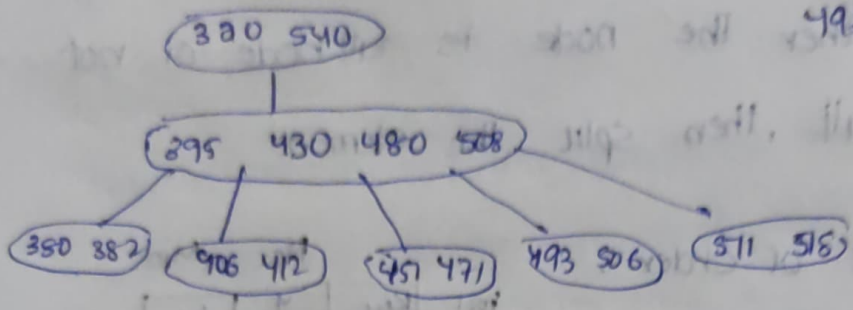




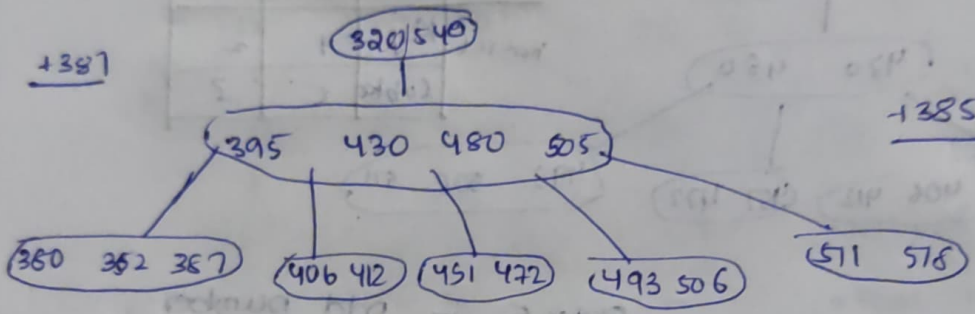


508, 387

385, 390

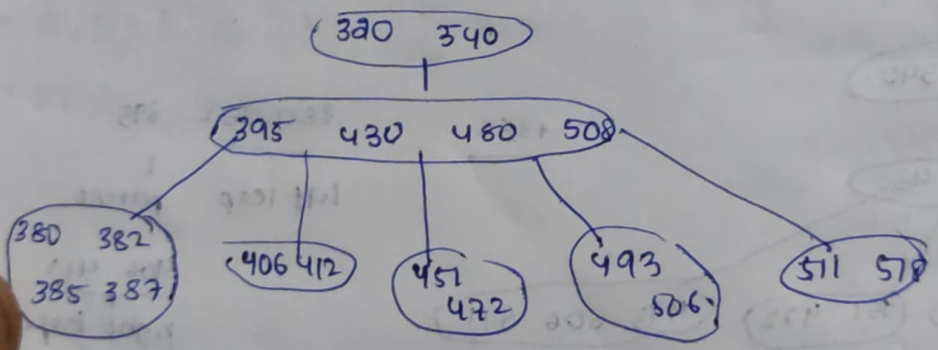


493 506 508
511 518



387

385



380 382 — left

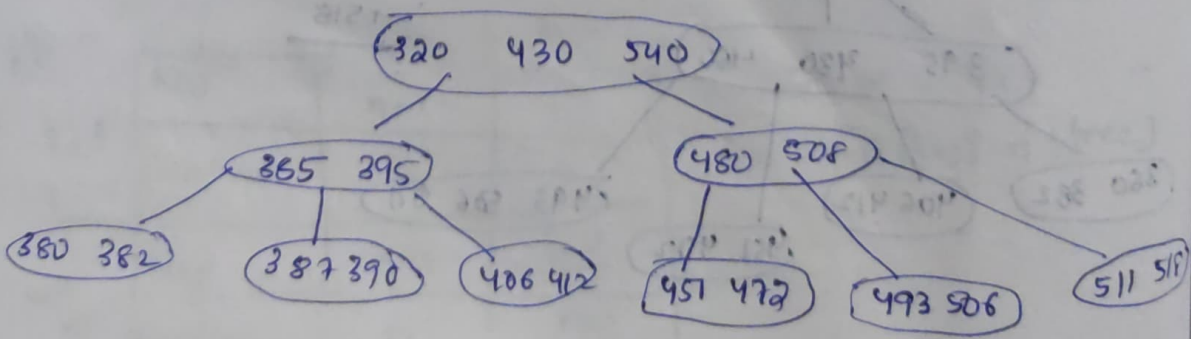
385 — parent

387 390 — right

385 395 — left

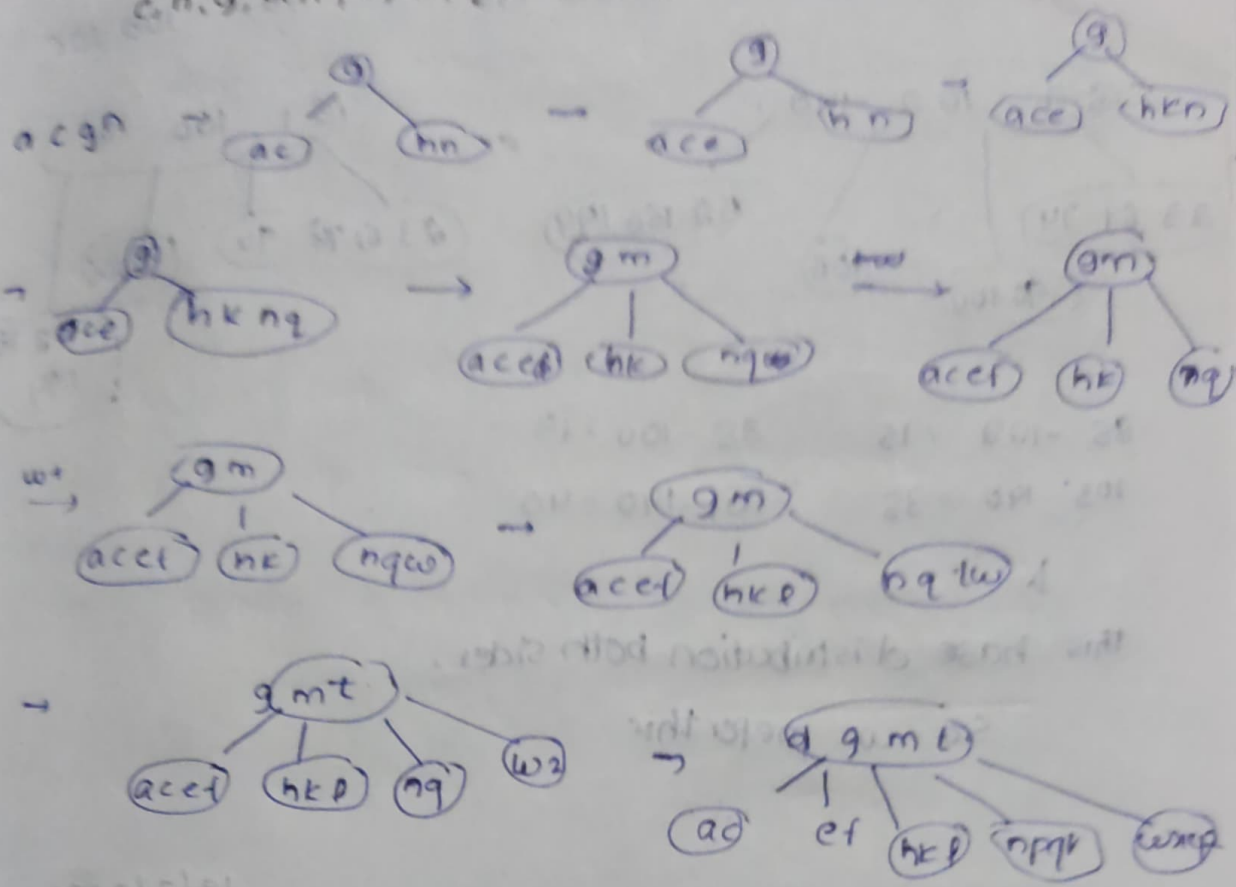
430 — parent

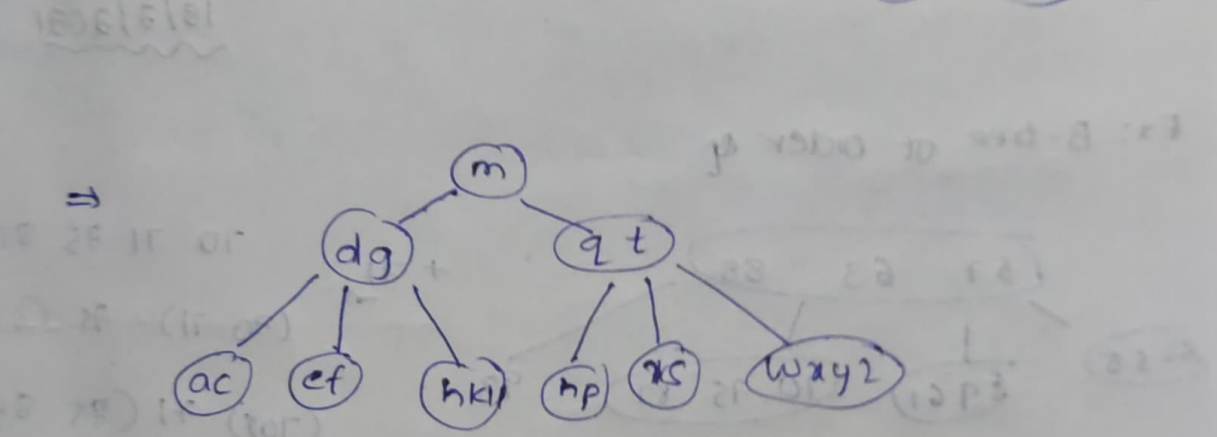
480 508 — right



Ex: B-tree of Order 5

c, n, g, a, h, e, k, q, m, f, w, l, t, z, d, p, r, x, y, s





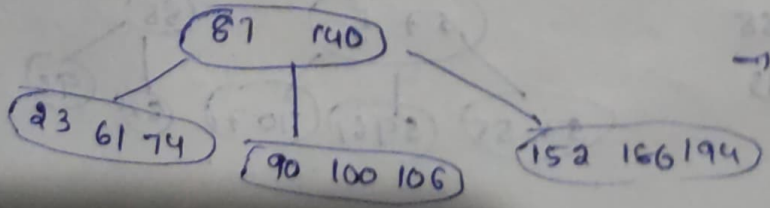
⇒

Ex: even order (n):

Max (n-1) keys (odd).

If we add a new element in full node then divide the group in unequal sized groups.

$n/2$, $n-1/2$.

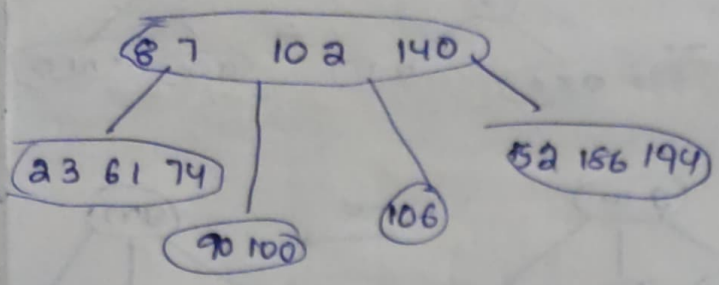


→ order 4

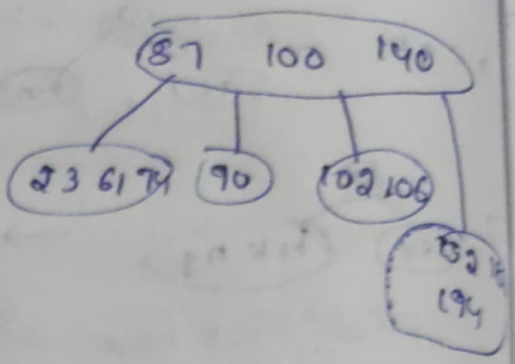
add 102

90 100 102 140 → 102 / 90 100 | 106

or 100 / 90 | 102 106



or



88 - 102 = 15 88 - 100 = 13

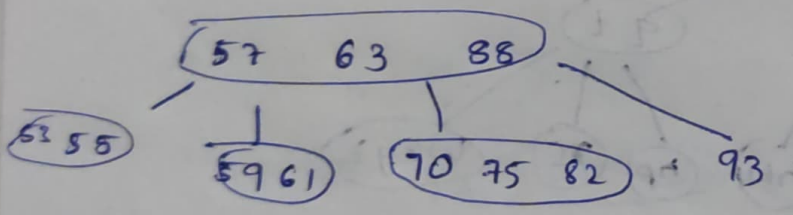
103 - 140 = 35 101 - 140 = 40

this base distribution both sides.

So we prefer this

18/2/2021

Ex: B-tree of order 4



+ 71 →

70 71 75 82

(70 71) 75 (82)

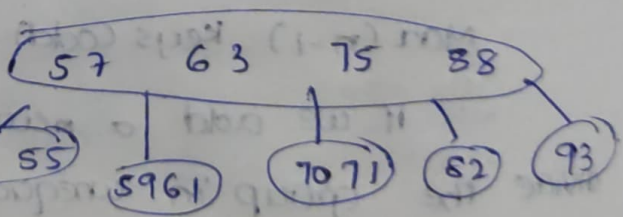
(70) 71 (75 82)

57 63 75 88.

12 13

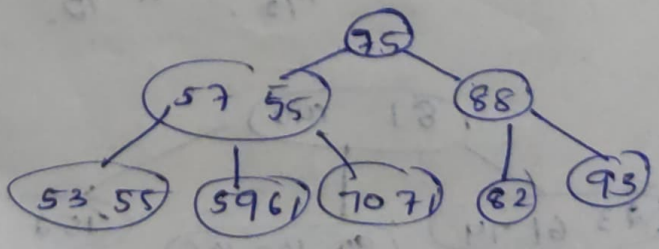
57 63 71 88

8 7



51 63 75 88

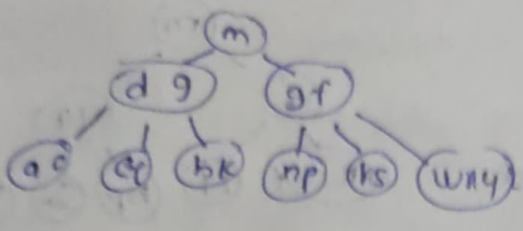
6 12 13

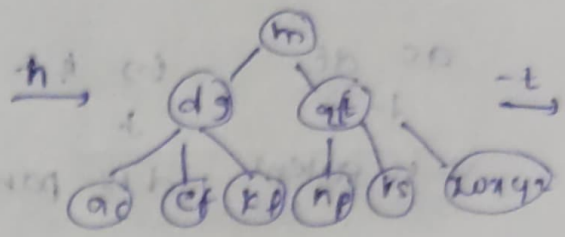


→ Deletion:

* Check whether the node going to have min no of keys after deletion.

Ex: B tree of order 5.



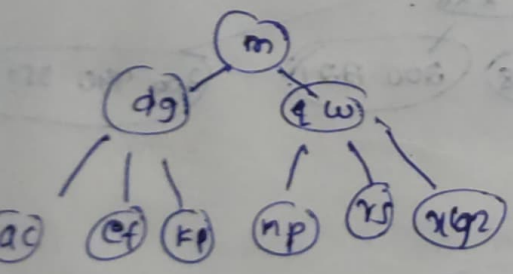


root → min 1 key
max 4 key
min 2 sub
Max 4 sub

non root → min $\frac{n-1}{2}$ = 2 key
max 4 key
min $\frac{n+1}{2}$ = 2 sub
max 5 sub.

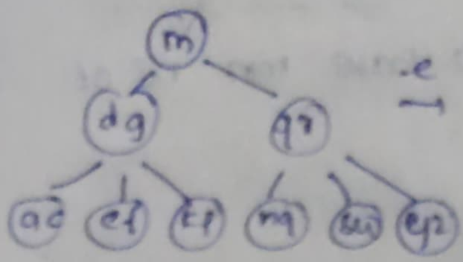
remove t → if left sub has more than required no of keys 't' predessor will replace 't'.

if right sub has more than required no of keys 't' - successor will replace 't'.

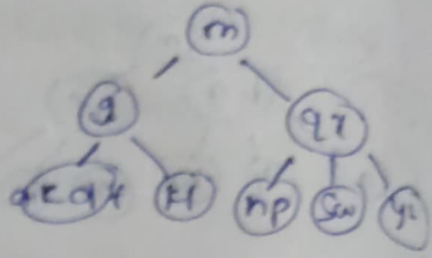


If 'r' is deleted → Min no of keys is then check left and right sibling which has more keys, it will donate.

successor will join the node and parent will also be changed.



No sibling have extra keys. So we have to merge that node parent & any of the sibling after deletion.



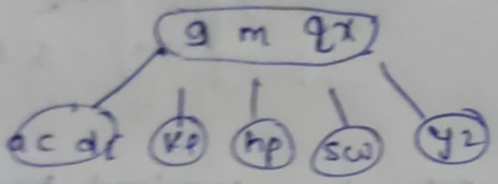
ac df ↓ g is parent

fg kl ↓ d is parent

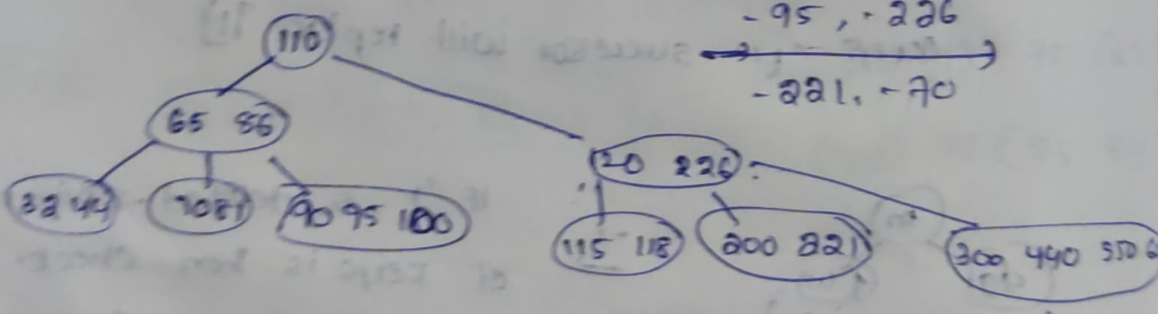
avg h to m

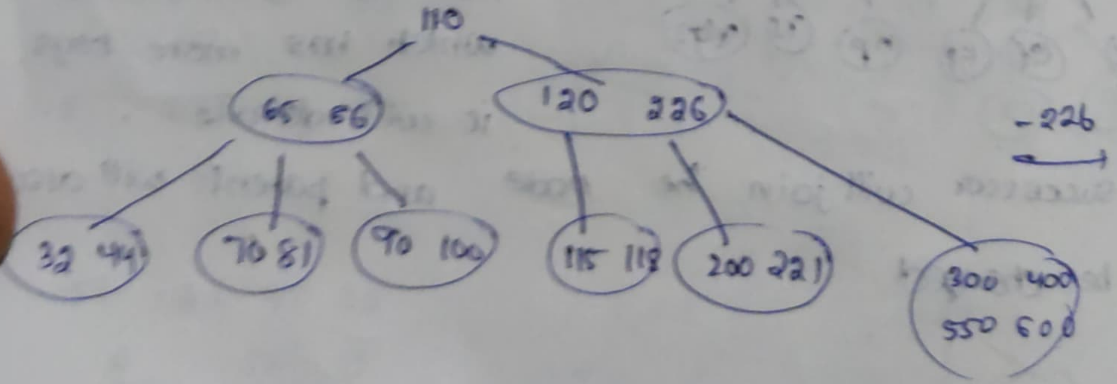
a to d, e to m

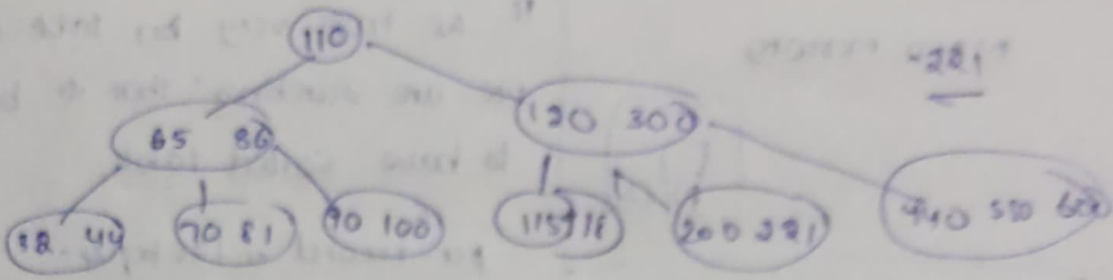
g to merged with m qr. → gm qr

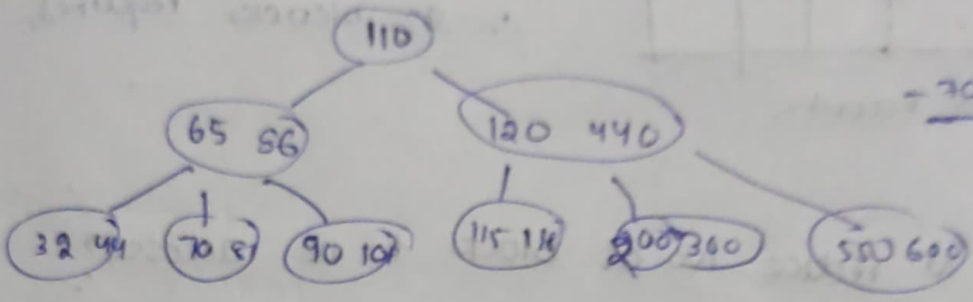


Ex: B-tree of order 5.



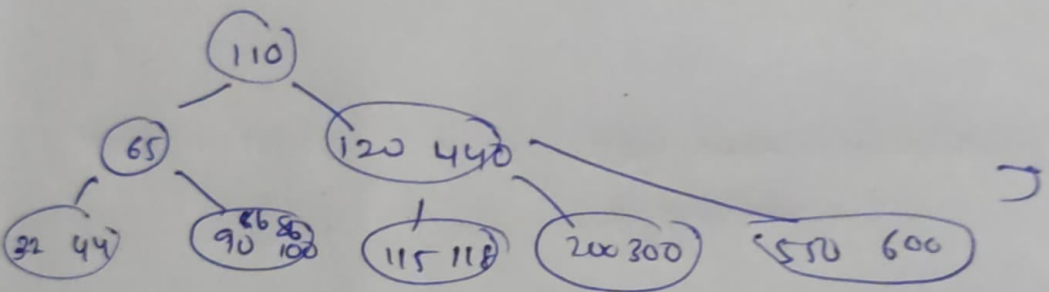


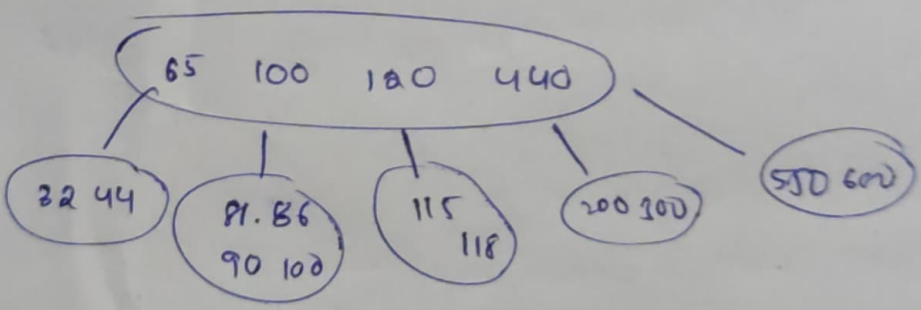




0-65, 66-110

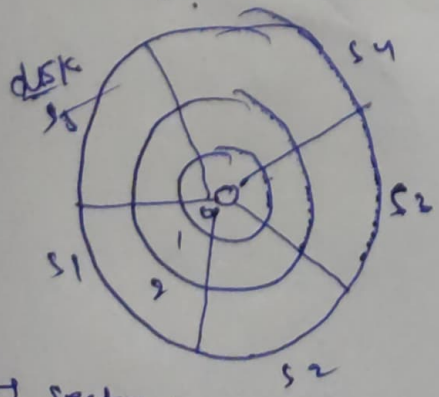
0-86, 87-100



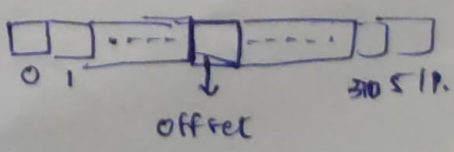


* DS → organising the data in main memory.

DBMS → organising the data in disk

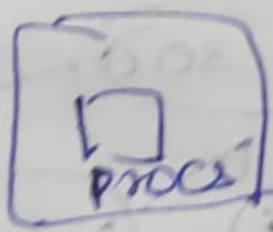


Block size = 512 bytes



sector
track } → address
offset

Main memory



If we have very big table and we are searching, then it better to have sorted table.

per-record = 128 bytes.

*

10	50	10	50	8
ID	Name	DOB	Addr	Rank
⋮				

100 Records.

∴ 4 records per block.

∴ 25 blocks required.

Since No of blocks to be accesed?