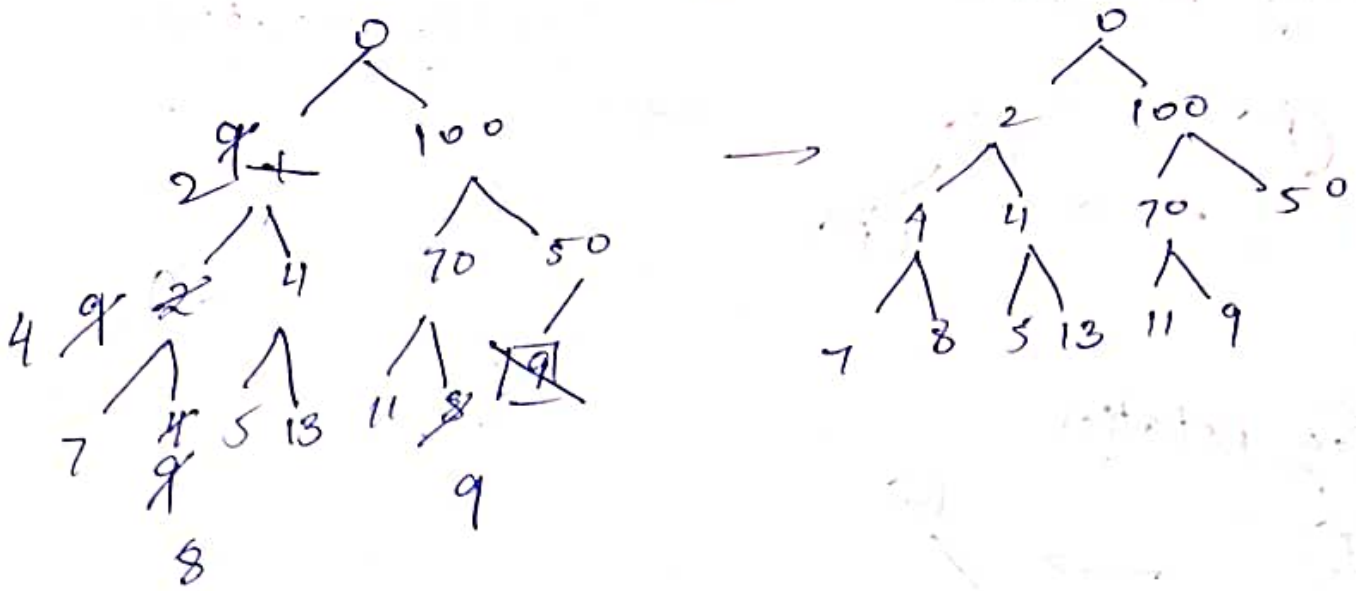
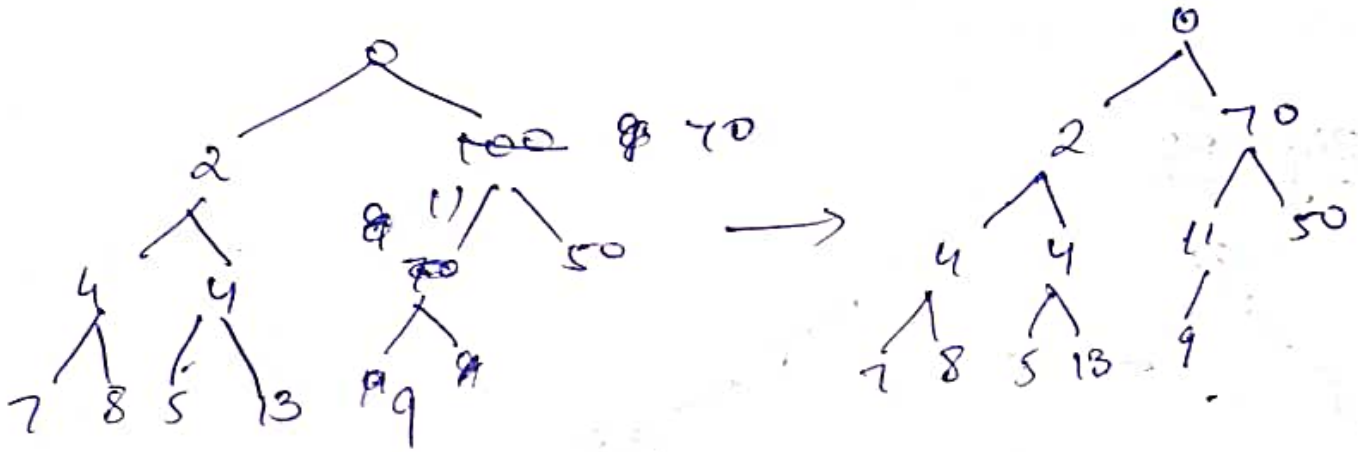
Delete -1



Delete 100



Compare each heap.

Splay tree

BST

Splaying sequence of rotations of node to root.

Not tightly balanced.

few data more frequntly used.

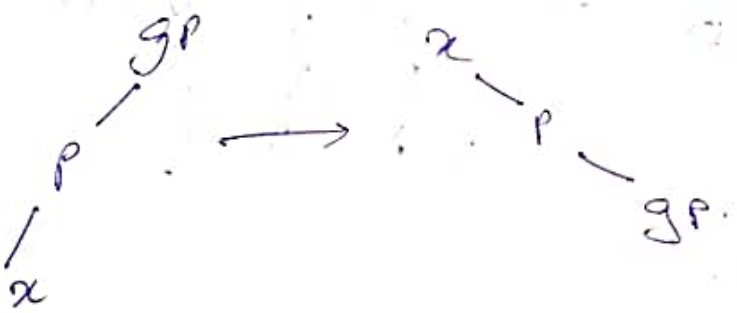
Storing at root or near root.

Zig Rotation

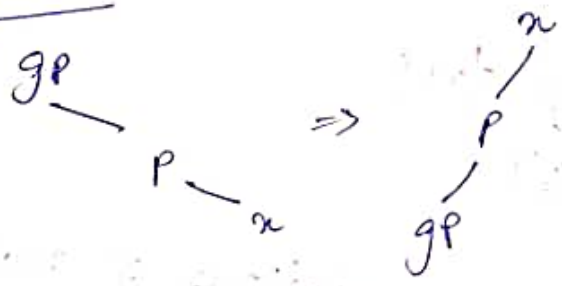
6 rotation

Zag rotation

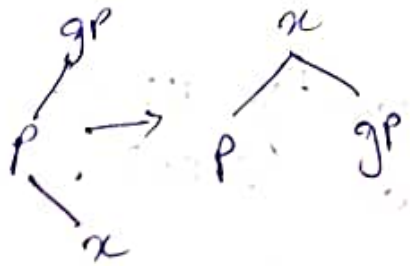
Zig zig



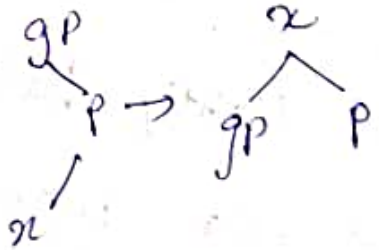
Zag zag



Zig zag



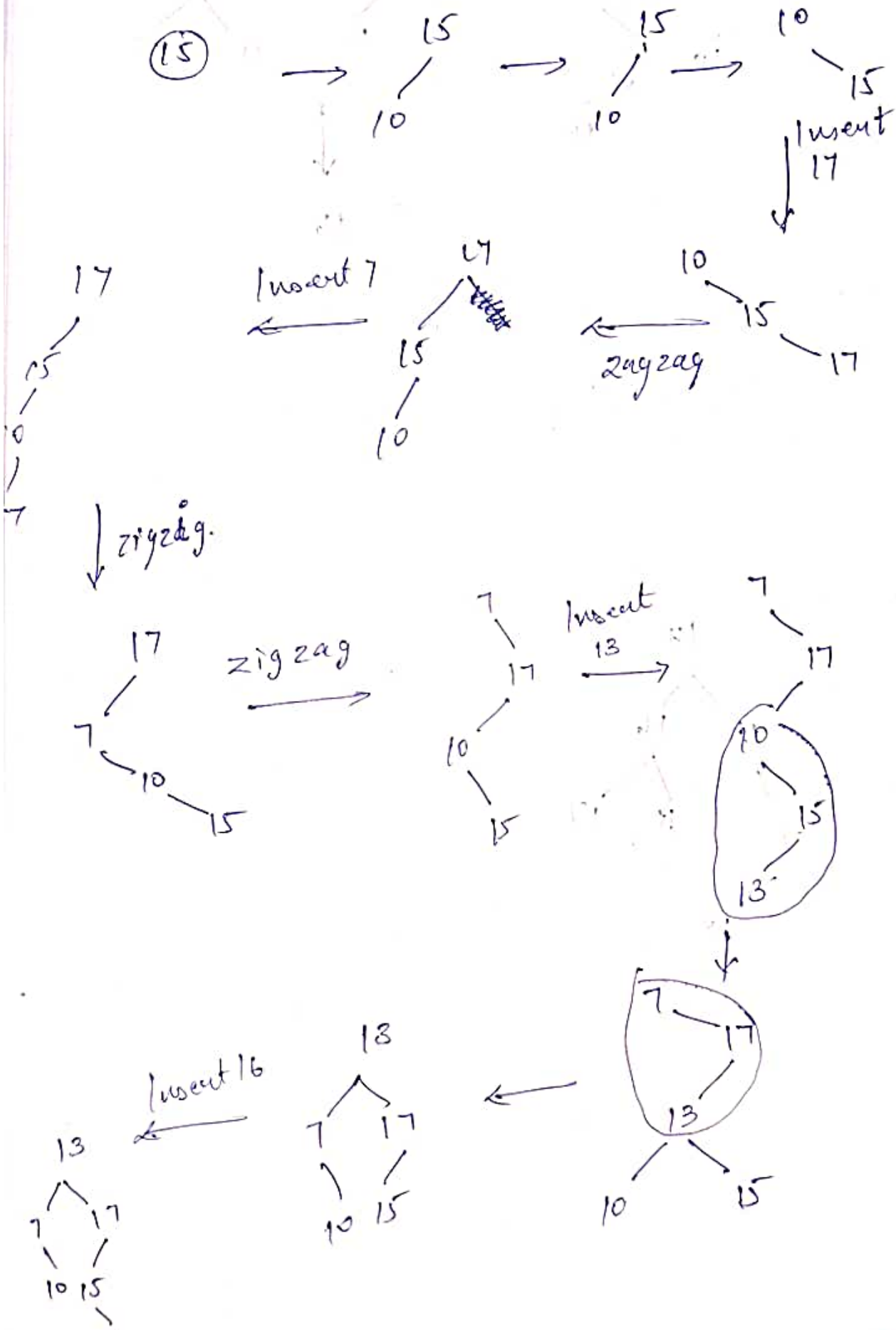
Zag zig

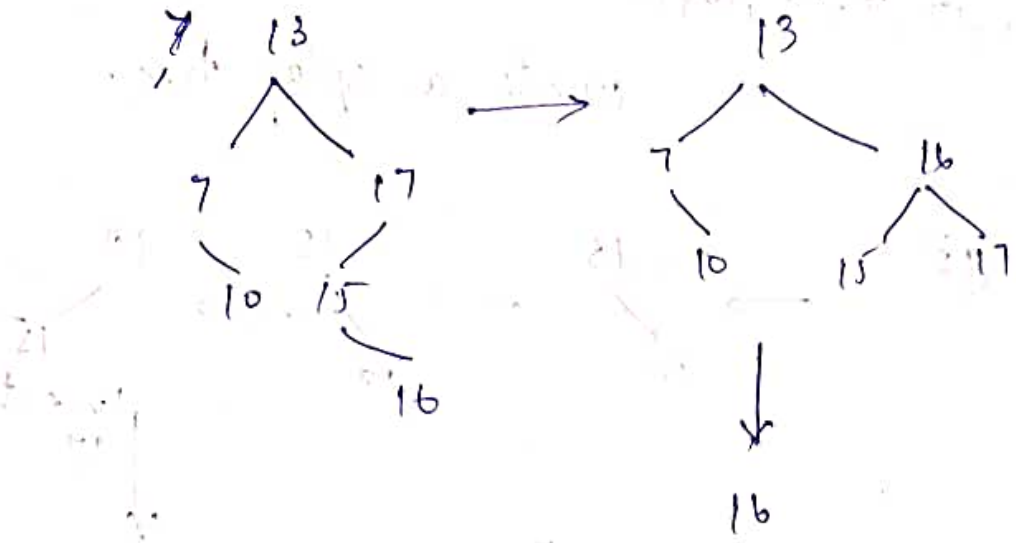


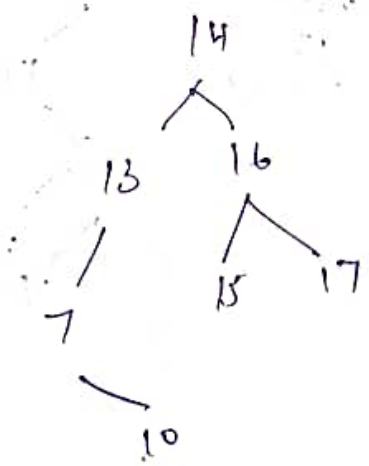
Insert:

15, 10, 17, 7, 13, 16, 14

create a splay tree.




13
7
17
10
15
16
13
7
16
10
15
17
16

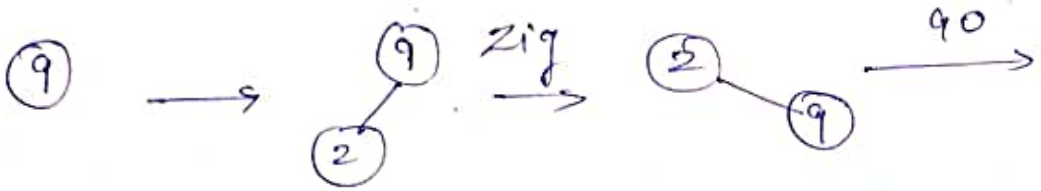

14
13
16
7
15
17
10

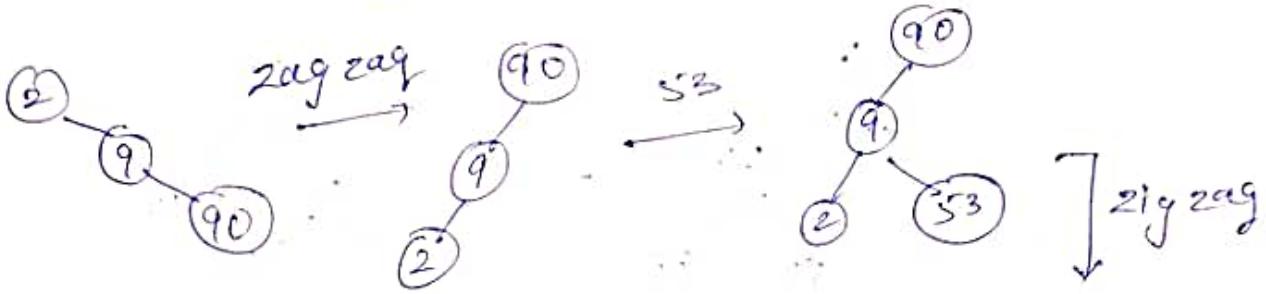
8/2/23

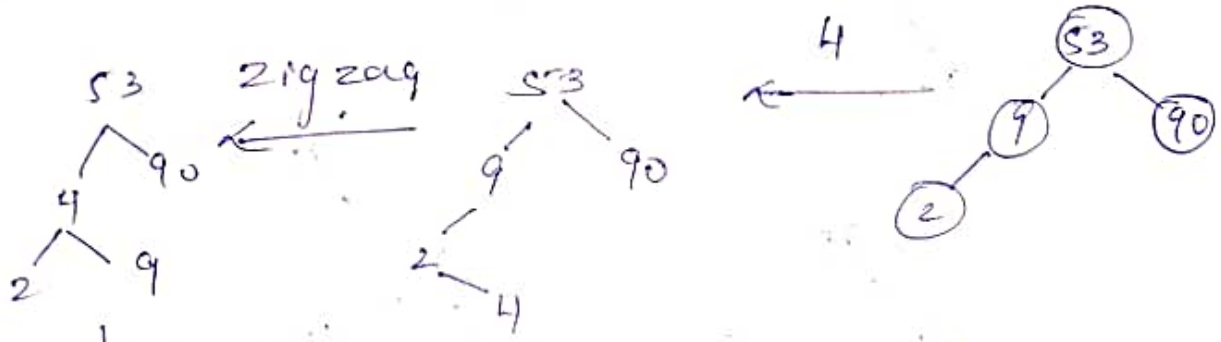
Splay tree

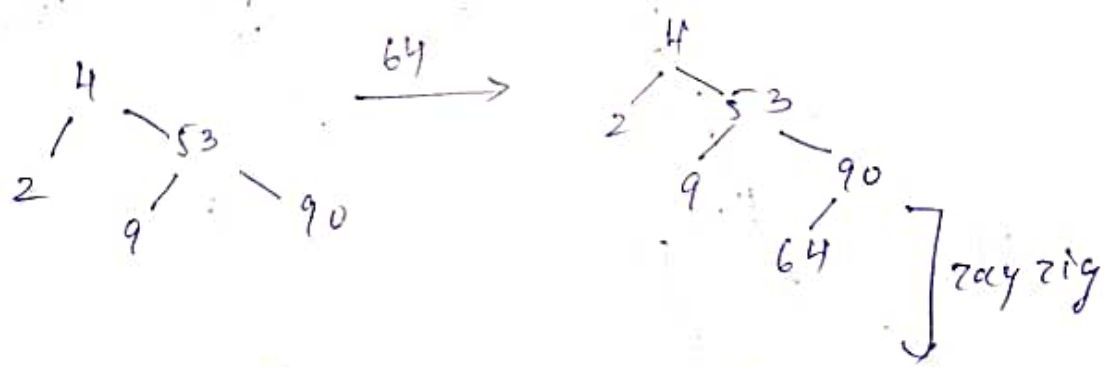
Insert:

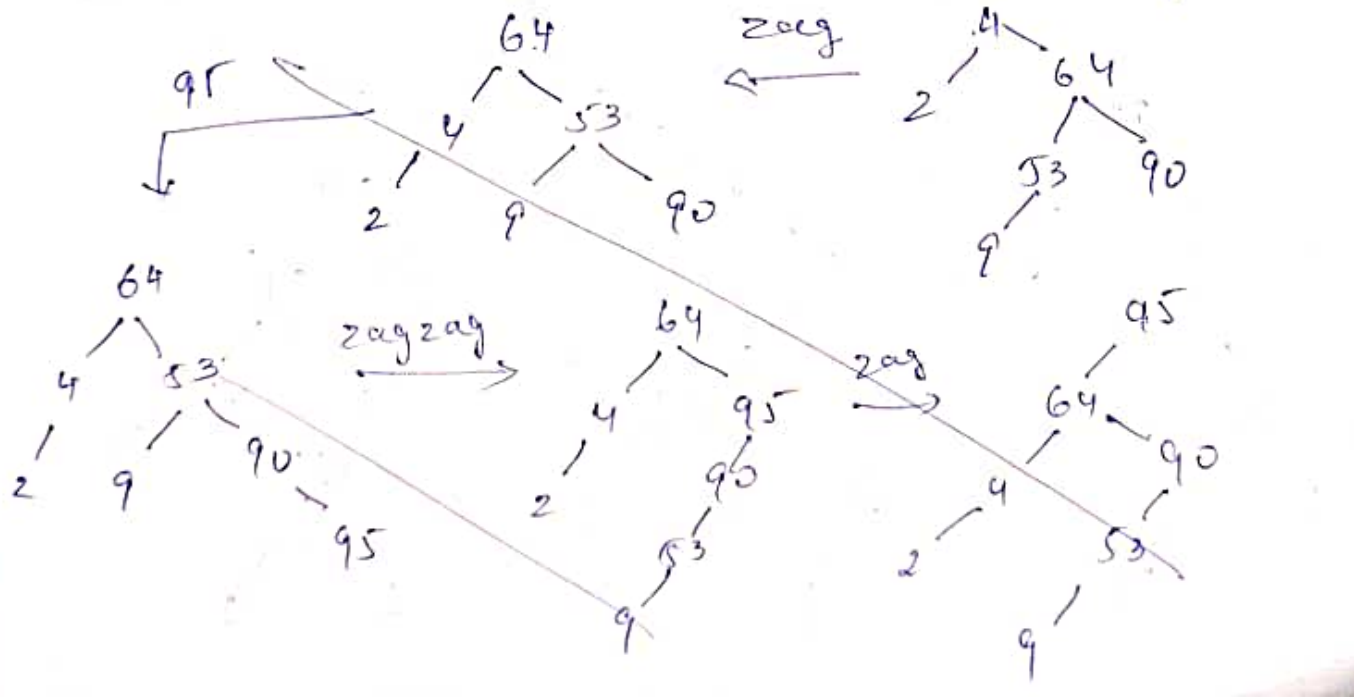
9, 2, 90, 53, 4, 64, 95, 59

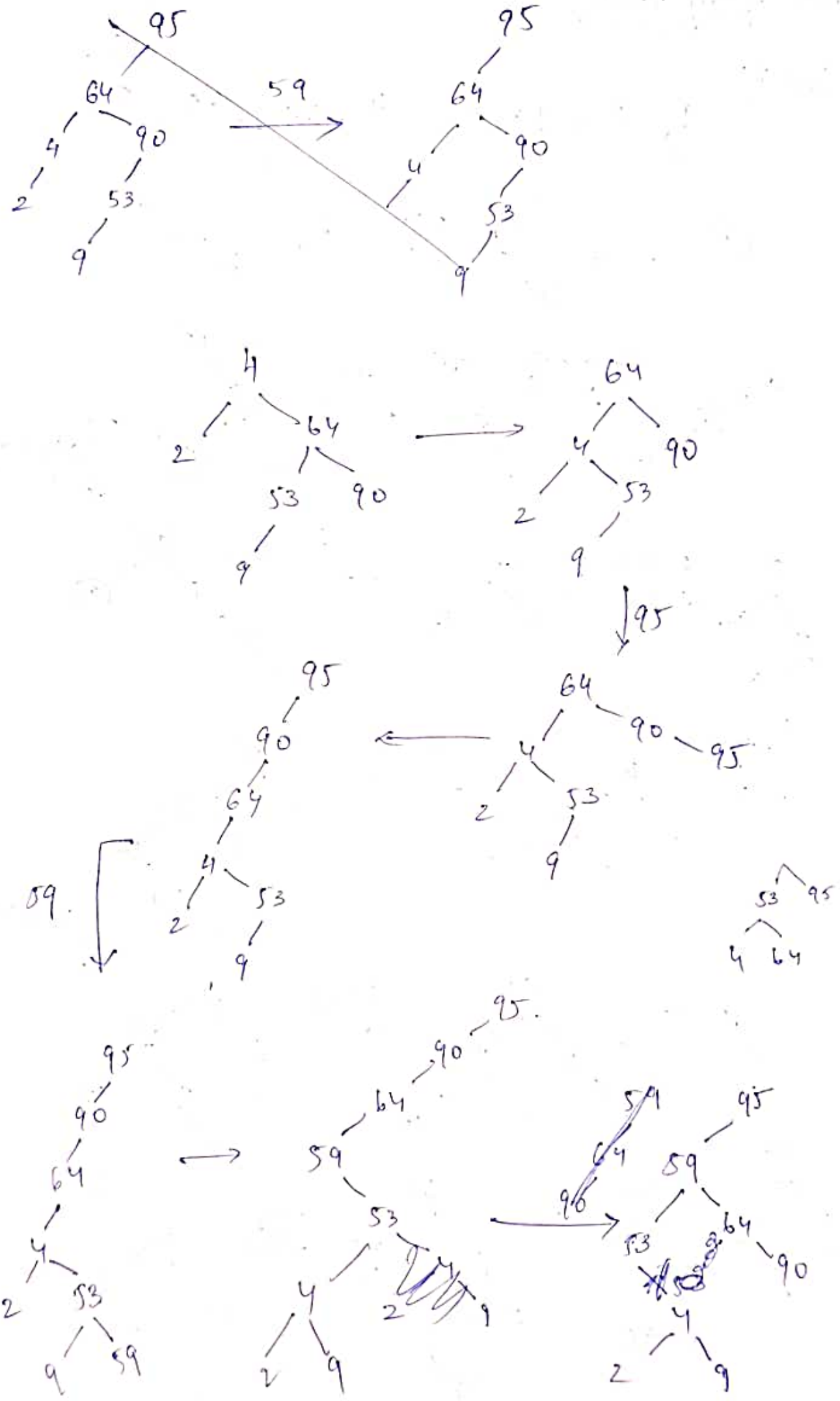


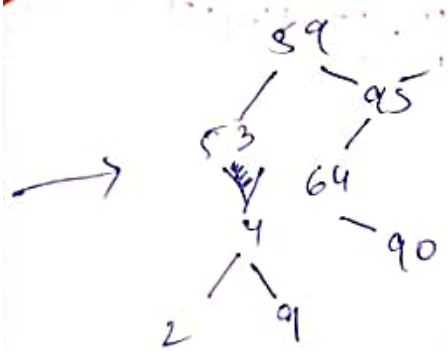








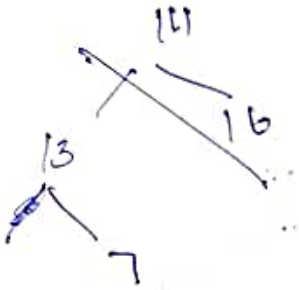

95
64
4
90
2
53
9
59
95
64
4
90
53
9
4
2
64
53
90
9
64
4
90
2
53
9
95
95
90
64
4
59
2
53
9
64
4
90
95
2
53
9
53
95
4
64
95
90
64
4
2
53
9
59
95
90
64
59
53
4
2
9
59
64
96
95
59
53
64
90
4
2
9
4
2
9

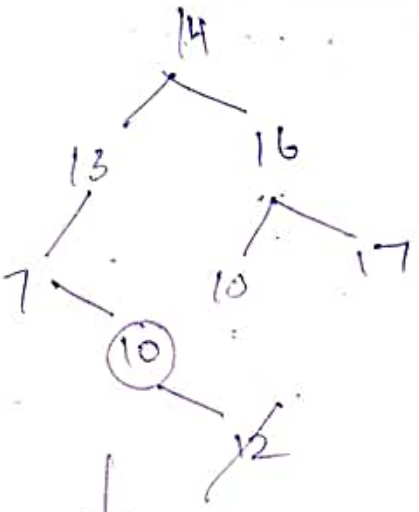


* All BST will be splay tree.

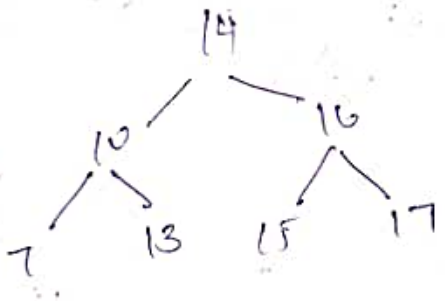
** Deletion.

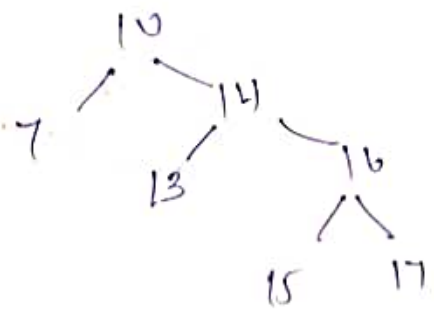
12, 14, 16, 20, 17



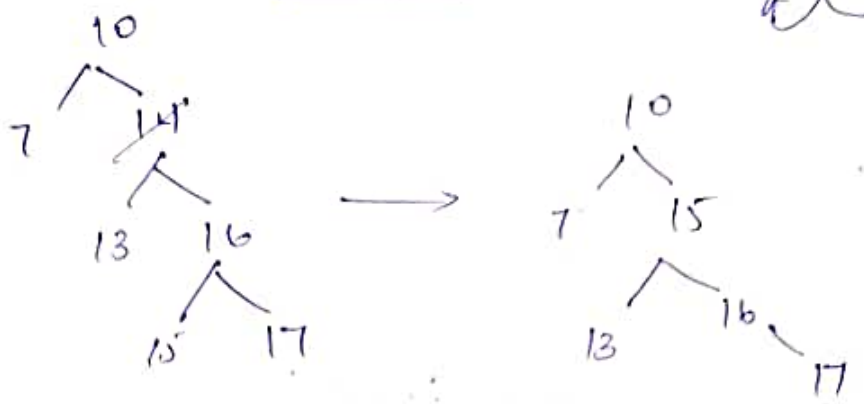


Delete 12





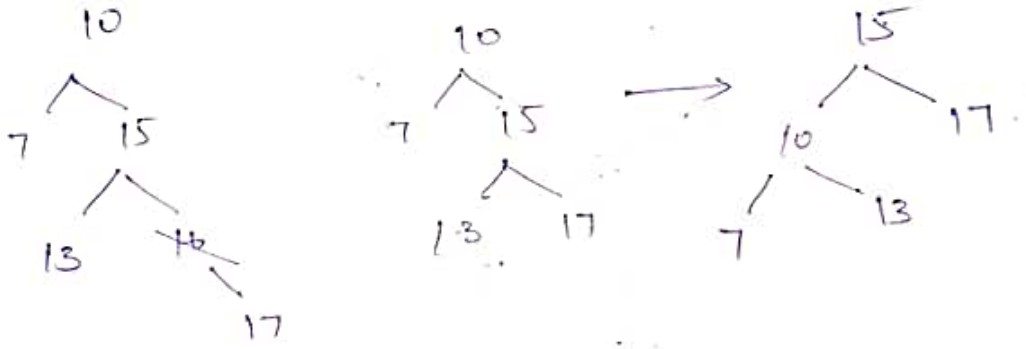
Inorder: 7, 10, 13, 14, 15, 16, 17



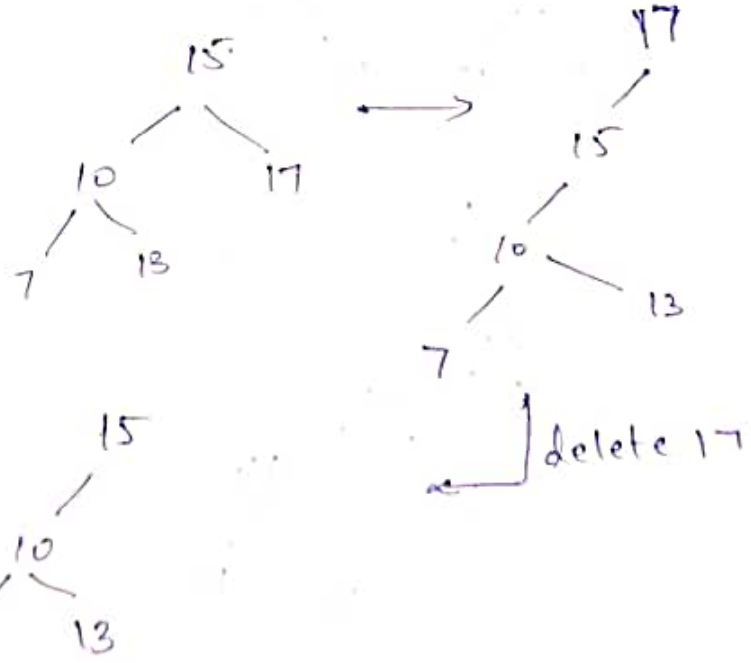
2 possibilities

→ Splay operation with parent

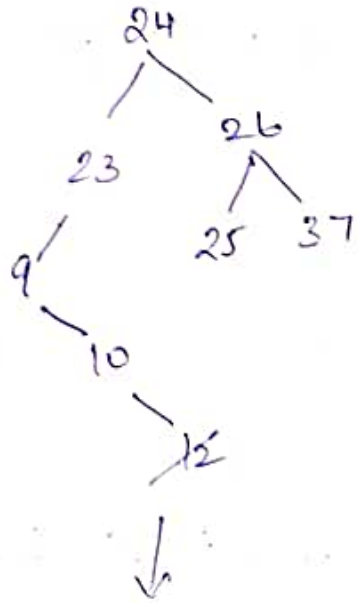
→ Splay operation with inorder successor.



If element not found, do splay operation with last seen node

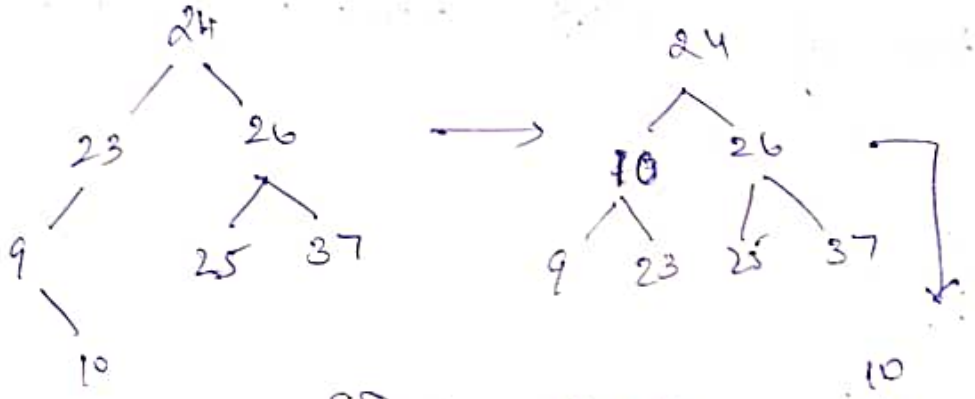


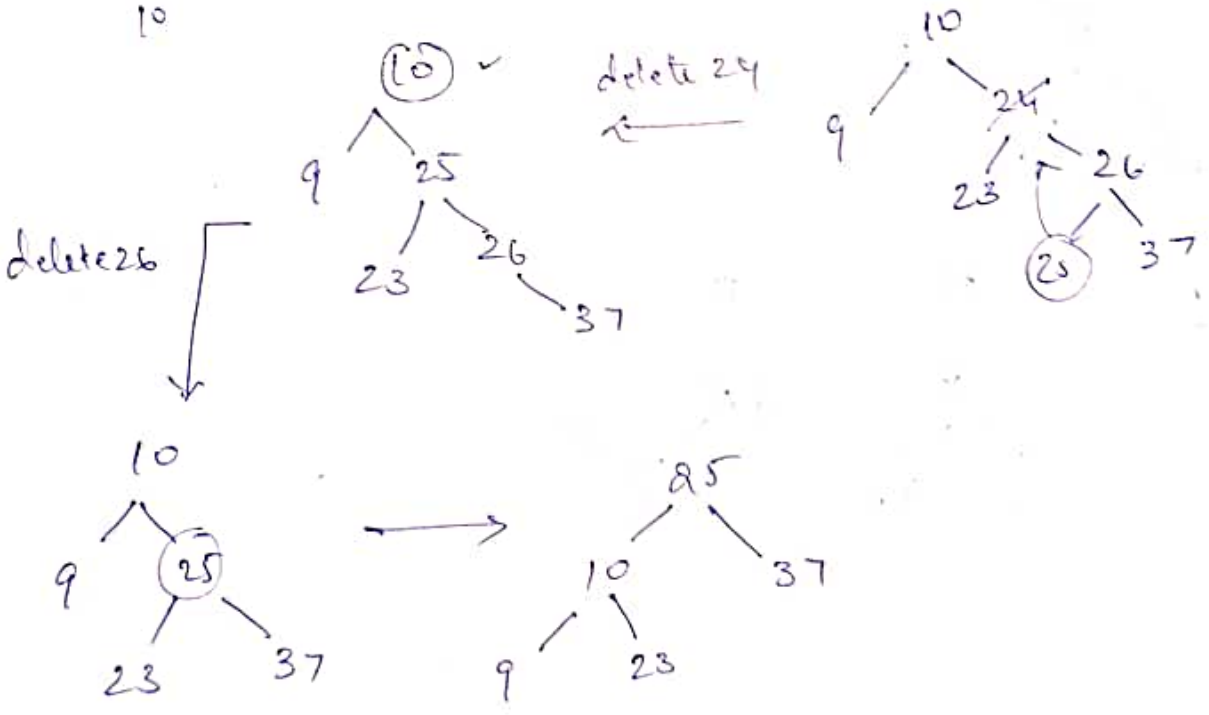
delete 12, 24, 26



Inorder Successor splay with parent node

Inorder: 9 10, (12), 23, 24 25, 26, 37





For insertion — 2 traversals. top - bottom
bottom - up

Top-down splaying

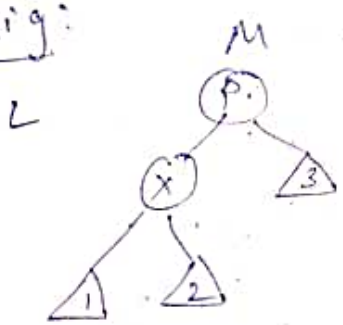
→ no need to maintain parent node pointer

L	M	R
empty	splay	empty

L: right subtree of largest element in L.

R: left subtree of smallest element in R.

zig:

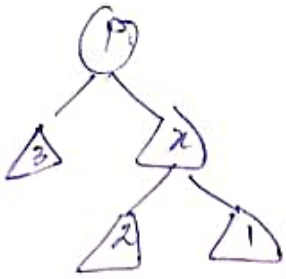


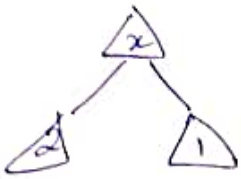
Zig:

L

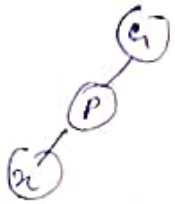
M

R





empty

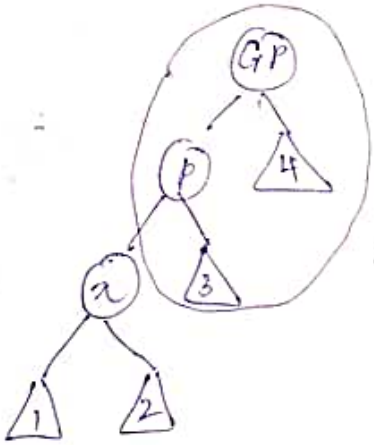


Zig Zig:

Left

Middle

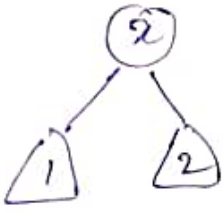
Right



Left

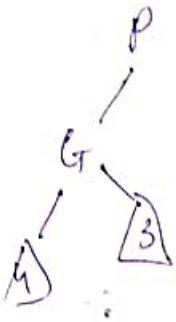
Middle

Right

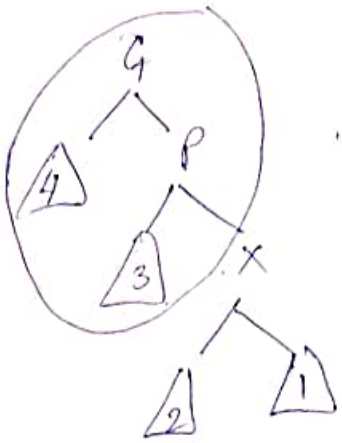


Zagzag

L



M

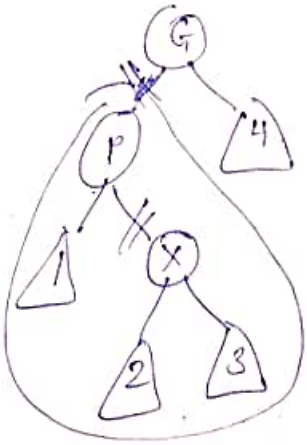


R

Zigzag

L

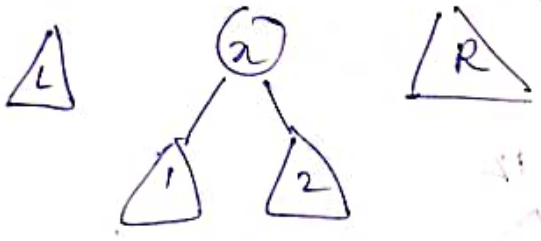
M

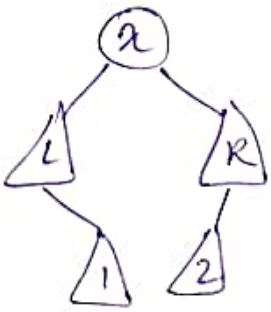


R

* bring only G to R

Zagzig ?

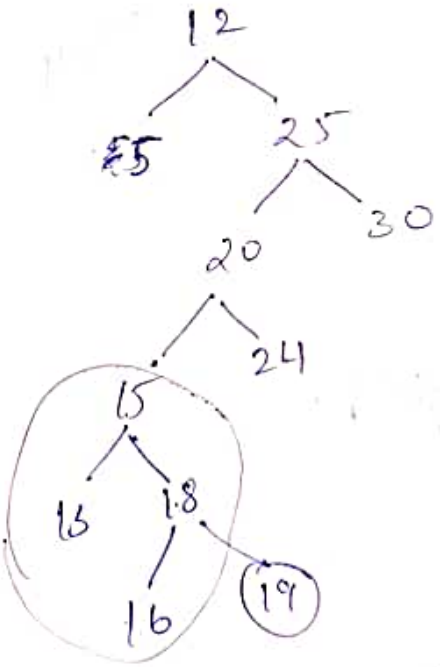


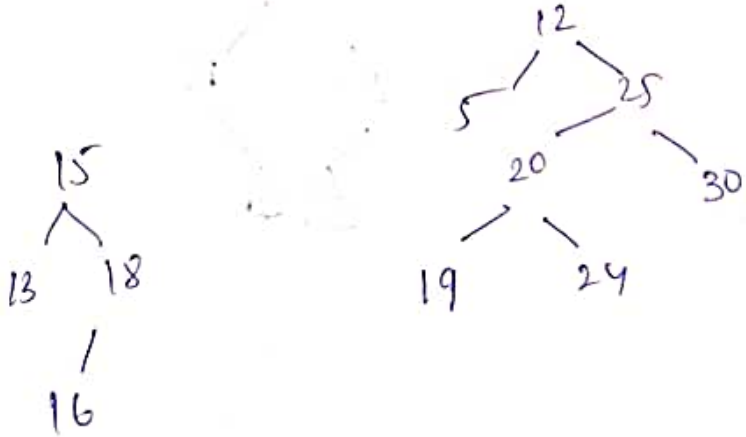


Top down splay tree; → avoid parent pointer / having stack.

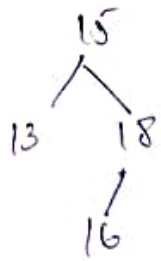
Insert 19:

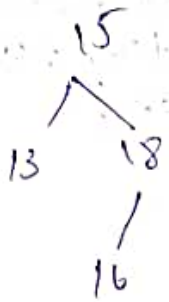
Left Middle Right

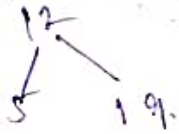


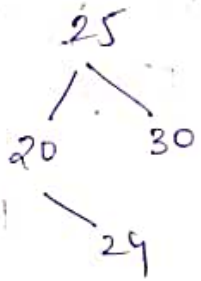


Lyt Middle Right





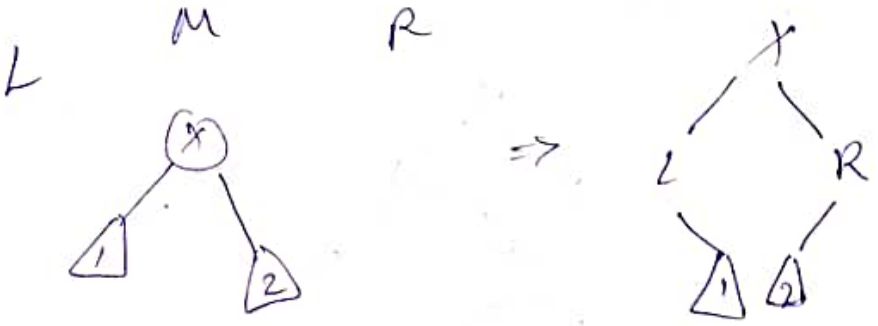




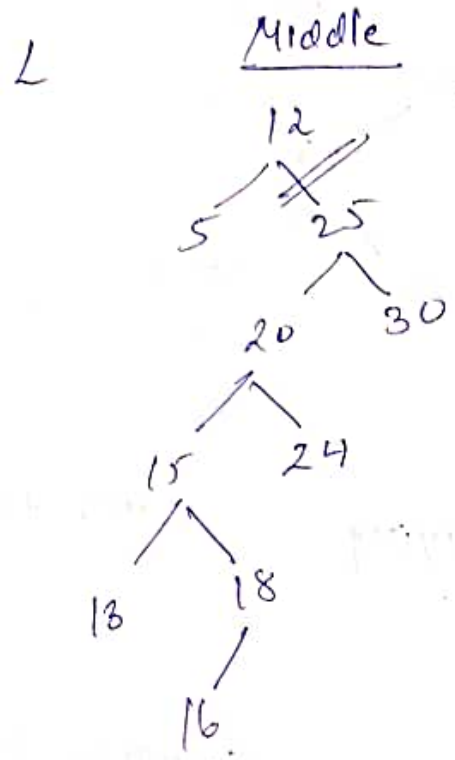
10/2/23

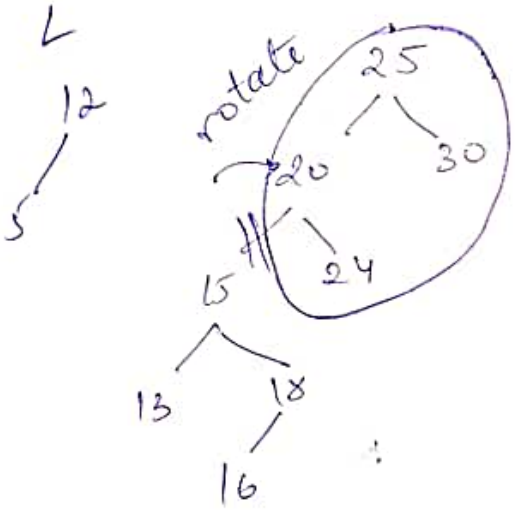
Top down splay

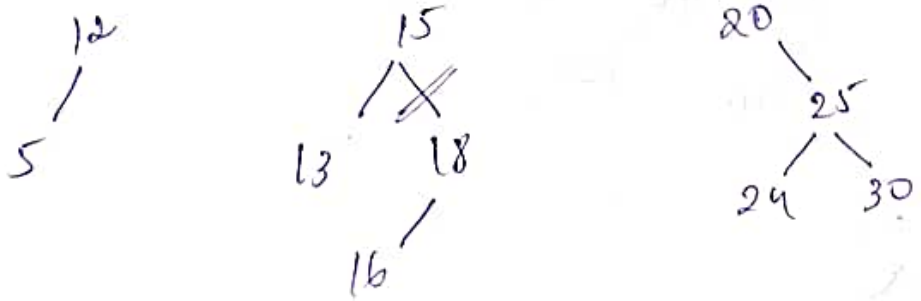
zig zag
zigzig zagzag.
Reassemble

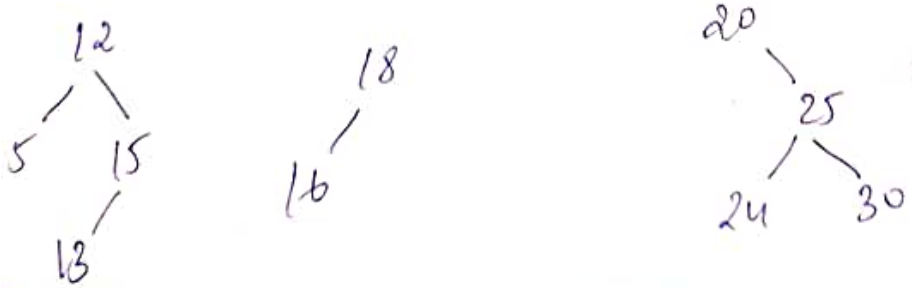


Search 19 :

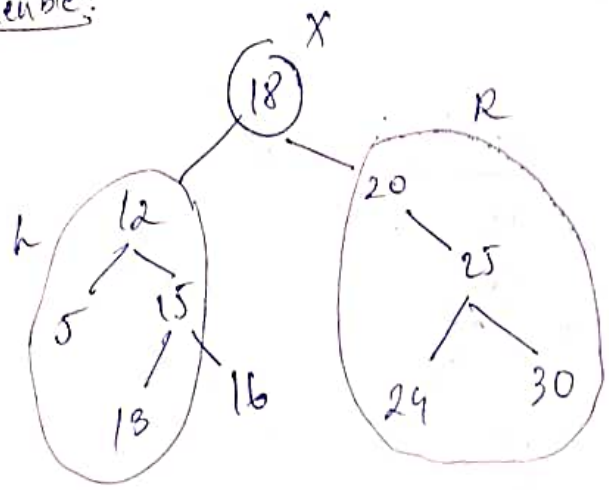

L
Middle
R
12
5
25
20
30
15
24
13
18
16


L
R
12
5
rotate
25
20
30
24
15
13
18
16


12
5
15
13
18
16
20
25
24
30


12
5
15
13
18
16
20
25
24
30

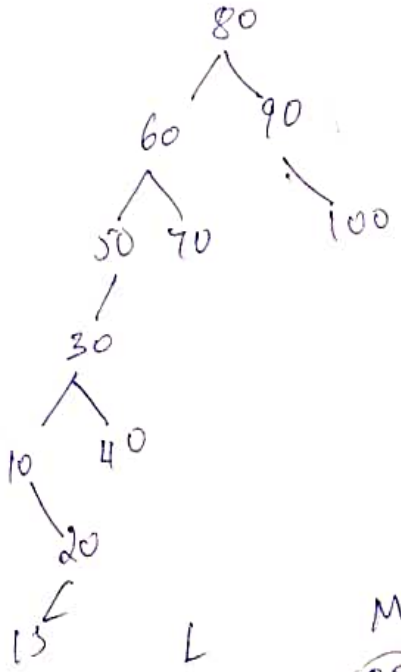
Reamenble:



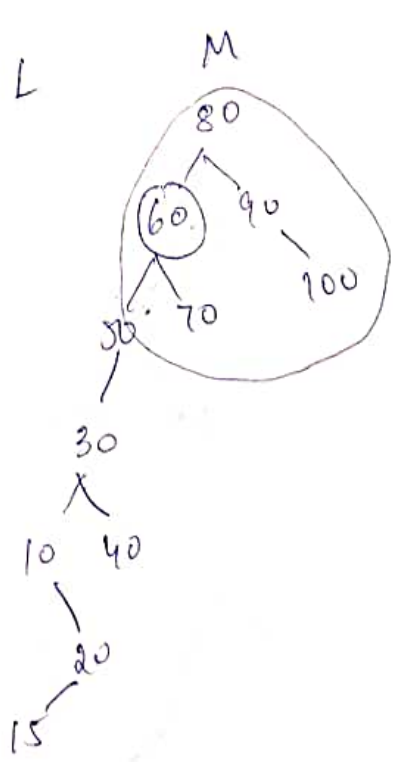
Only one traversal

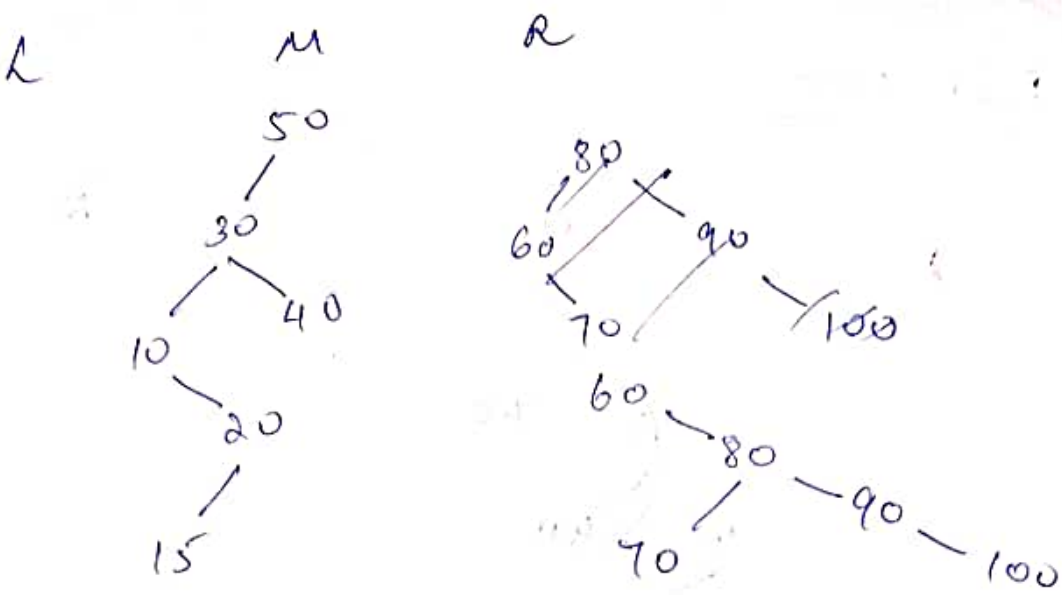
H/W: bottom up splaying on same tree.

(2)

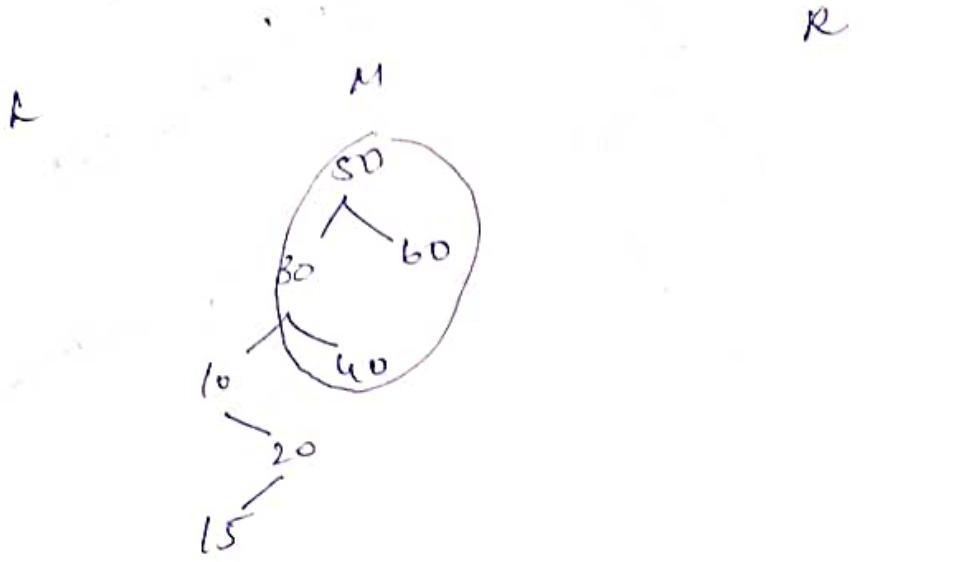


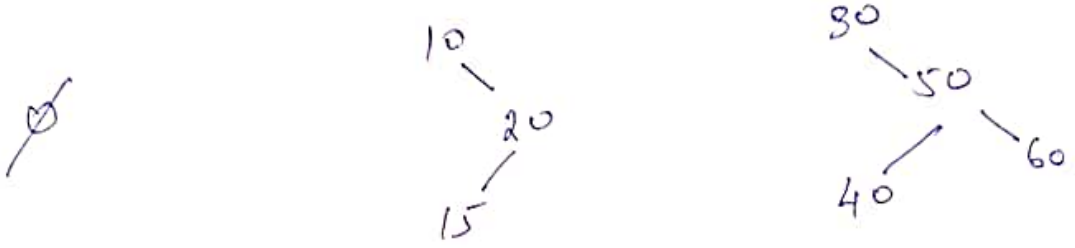
search 30



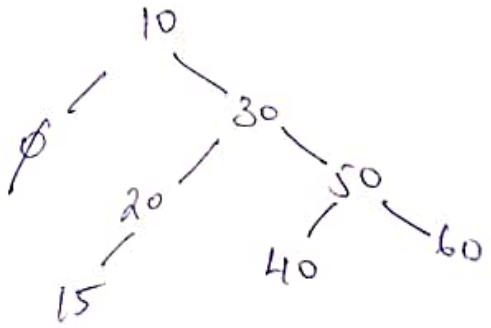

L
M
R
50
30
40
10
20
15
80
60
90
70
100
60
80
90
70
100

Find Min





Reassemble:

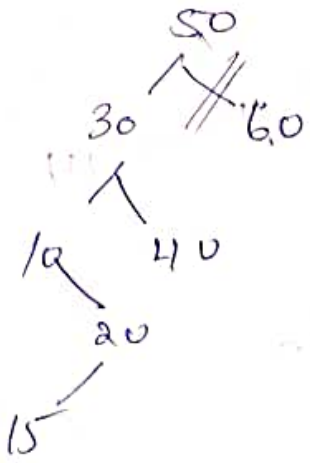


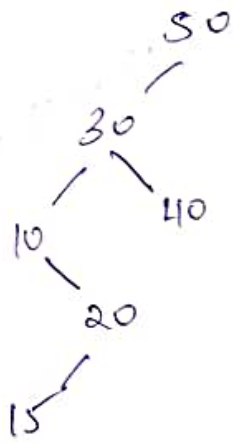
Delete Min

(1) Find Min.
(2) remove min
clean right subtree as the root.

Find max

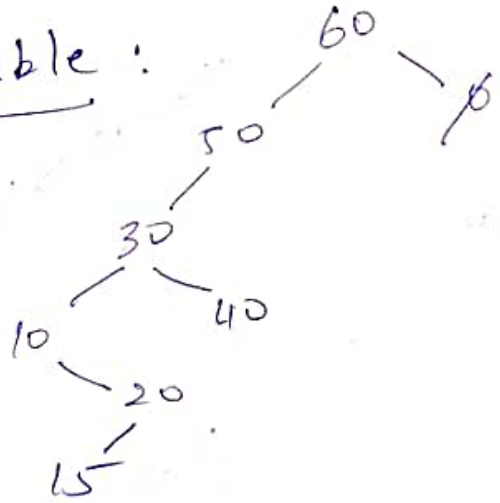
L M R





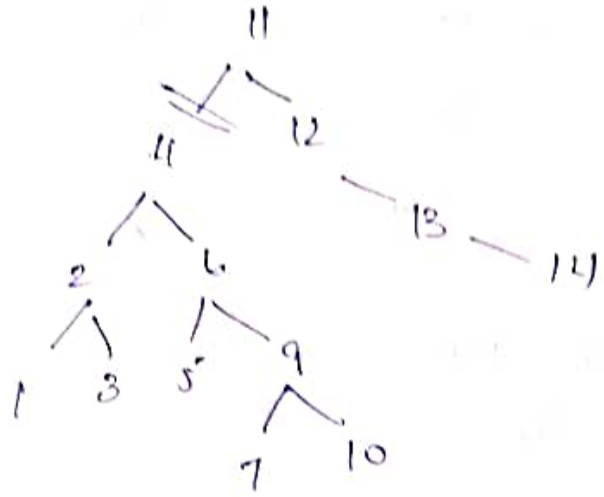
60 ∅

Reamemble :

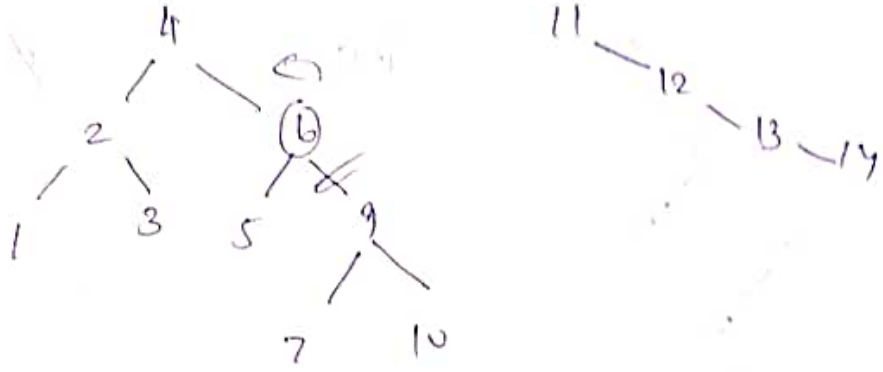


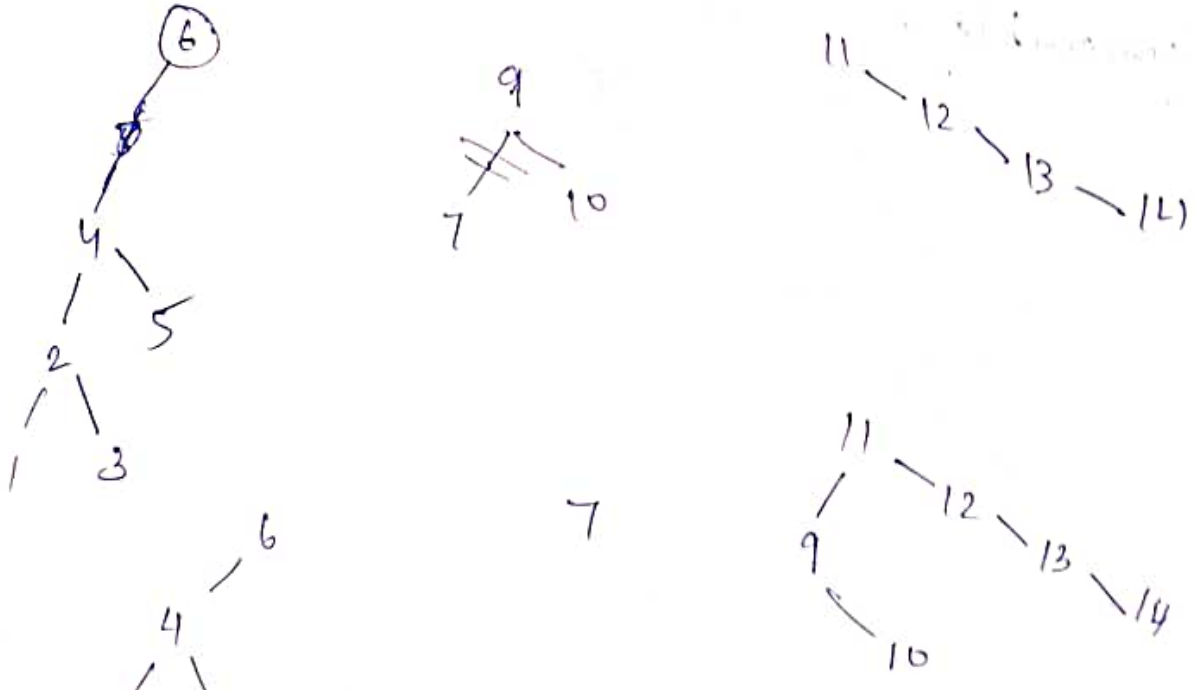
Delete node: (7)

L

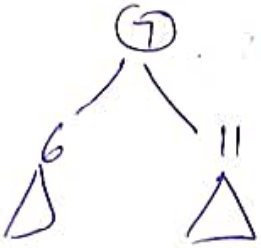


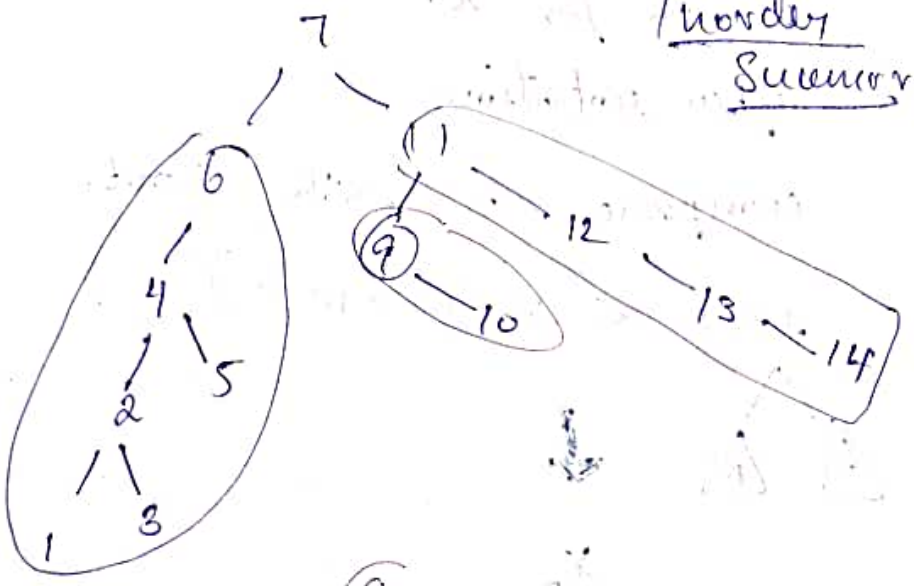
R



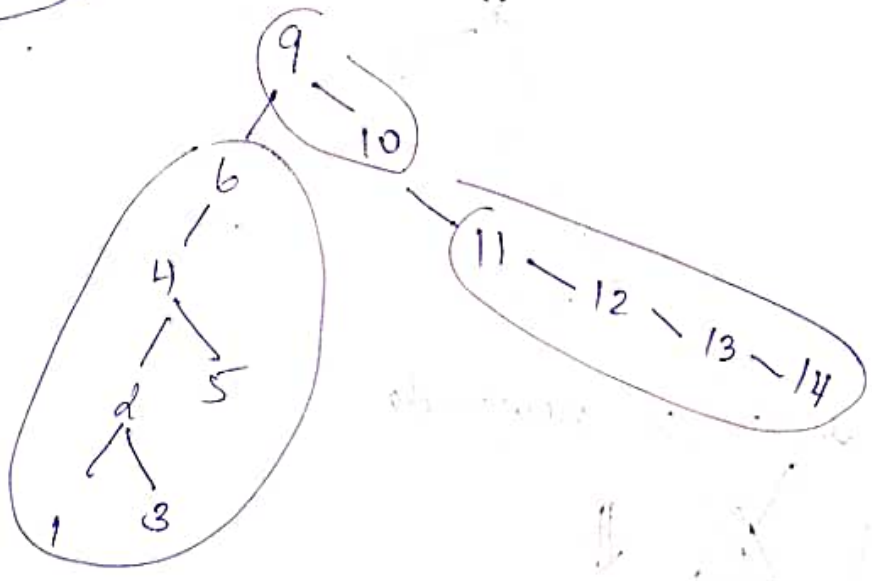


Reassemble:





Inorder Successor: 9



Top Down Splaying

Insertion:

9, 2, 40, 53, 4, 64, 95, 59

Insert: x

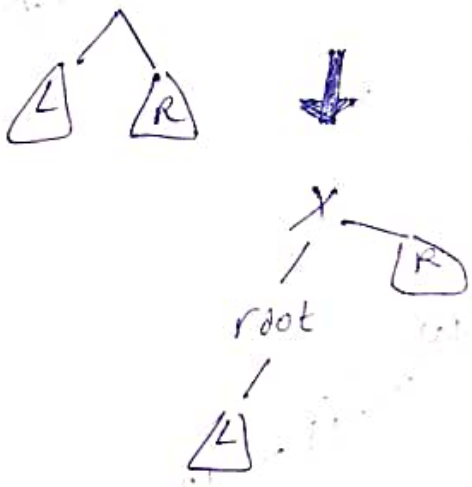
procedure:

search for x.

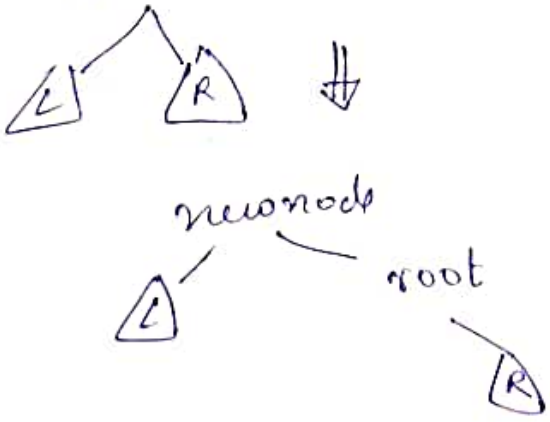
after rotations

compare x with root.

root < x (new node)



root > new node



L M R.

a)

b)

c)

2 × 9

(2) (9)

Reassemble:

9

2

9 < 40.

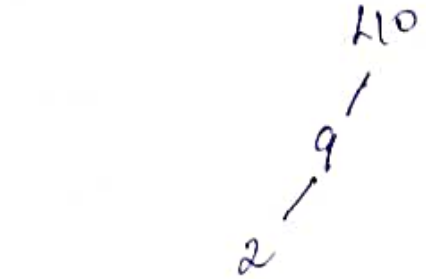
40

9

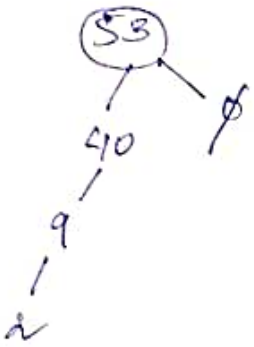
2

do rotation
do reassembling
compare root

L M R

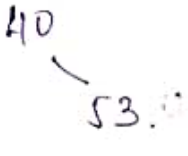


(after reassembling)

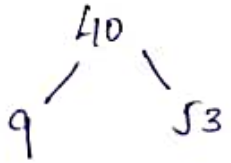


Insert 4 :

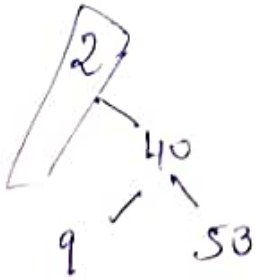
L M R



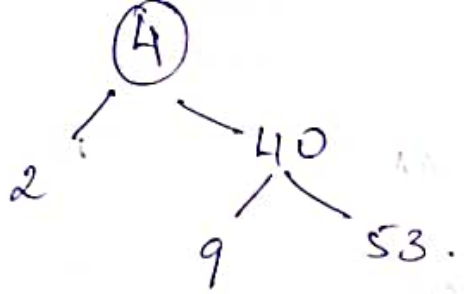
2



Reassemble :

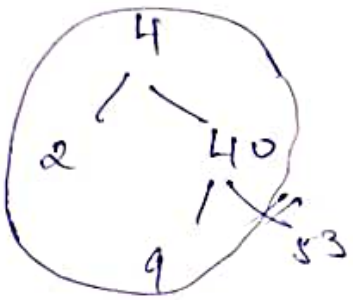


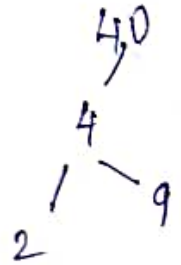
2 < 4



Insert 64:

L M R

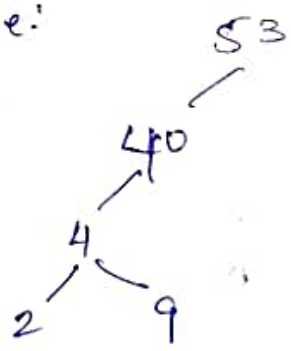




53

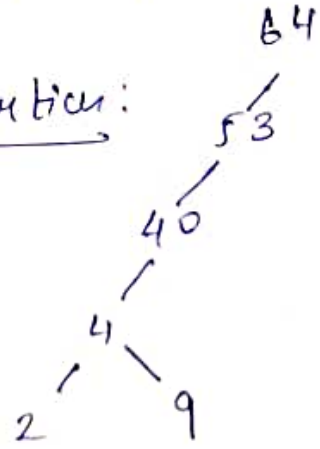
$\emptyset$

Reassemble:



53 < 64

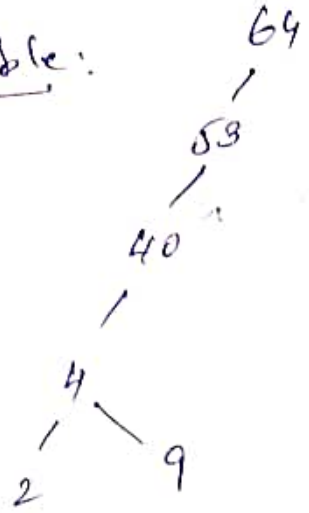
After Insertion:

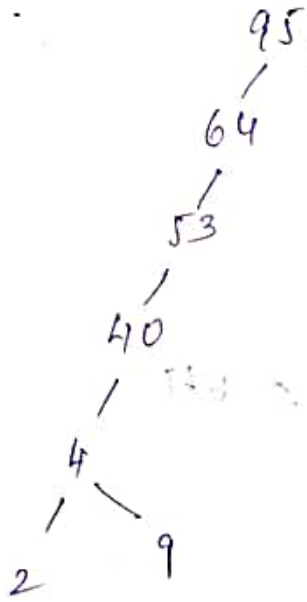


Insert 95:

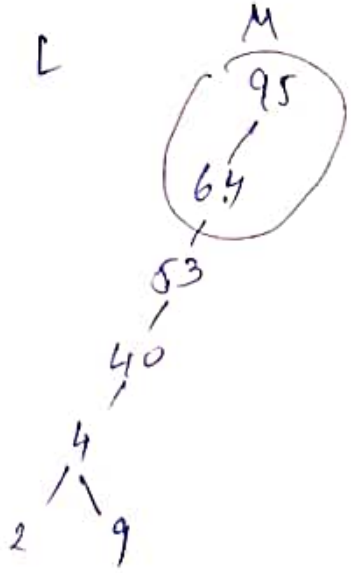
L M R

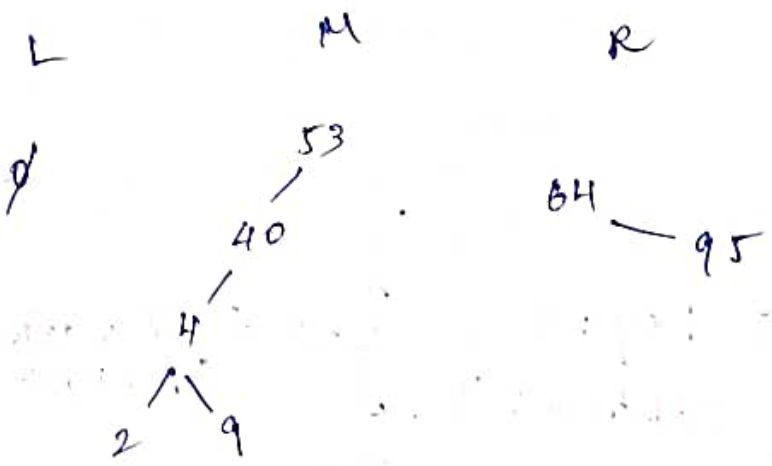
Reassemble:



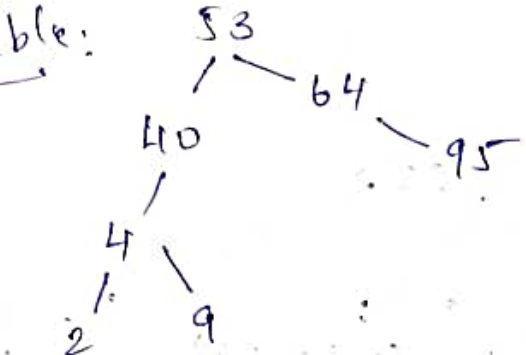


Insert 59: L M R

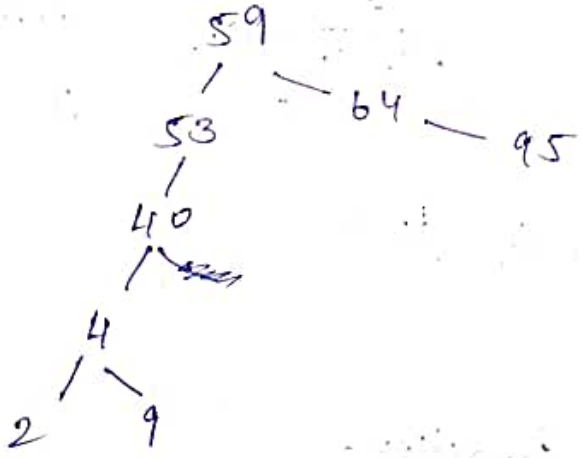




Reassemble:



53 < 59



H/W do insertion in bottom up playing.

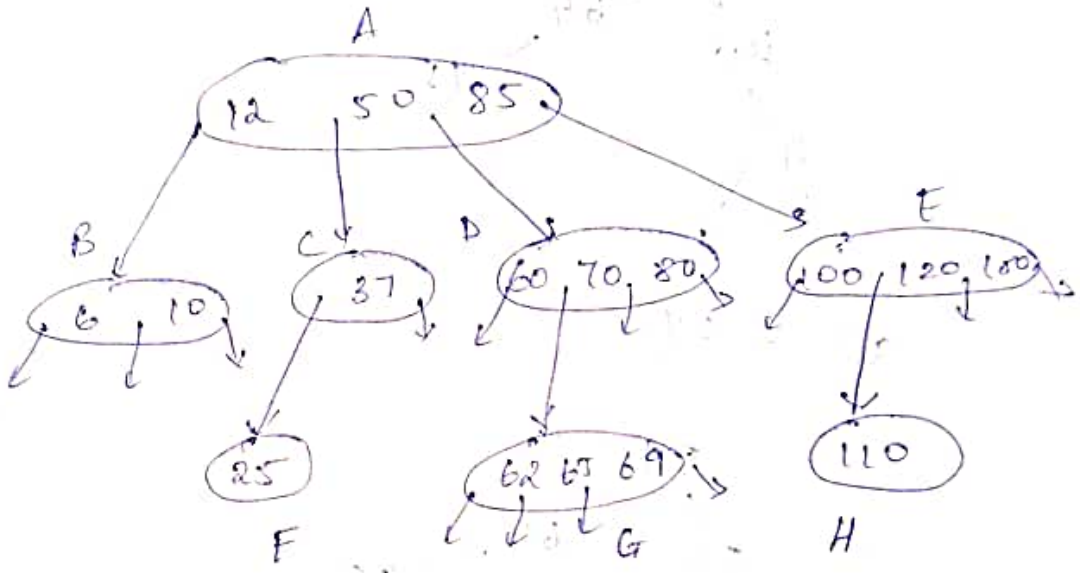
H/W: 9, 2, 90, 53, 4, 64, 95, 59

13/2/23

Order 4

(1) Key : 3
Subtree : 4
} ⇒ Full nodes {A, D, E, G}

Key :- ascending order.



Multiway Search tree

B - Tree : Conditions:

(i) Multiway Search Tree.

→ all elements non decreasing order.

B tree of order n

(ii)

		Max	Min
Root	key	n-1	1
	Subtree	n	2
Non Root	key	n-1	$(n+1)/2$
	Subtree	n	$(n+1)/2$

n — (n-1) tree max number → #subtree
or min number → key
(n-1) → n tree.

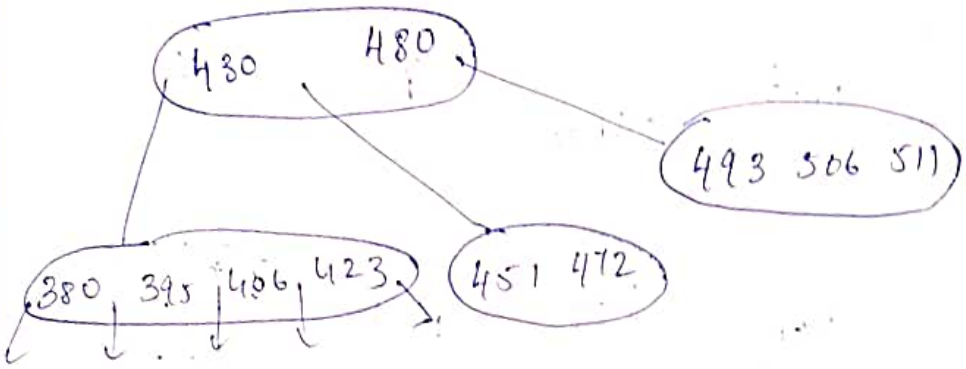
(iii) Leaves → Same level.

Insertion:

B-Tree n → order.
no. of keys → n-1

left < root
root < right.

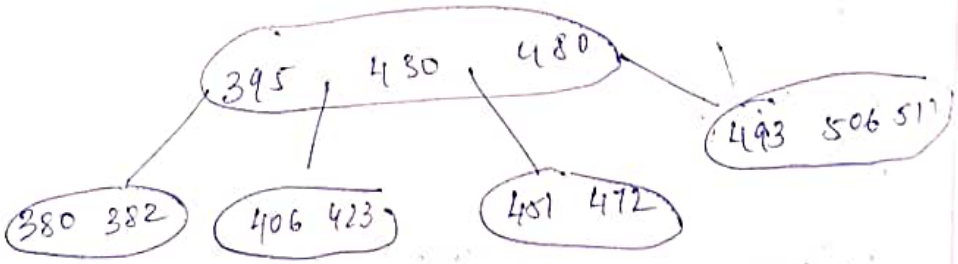
Order : 5



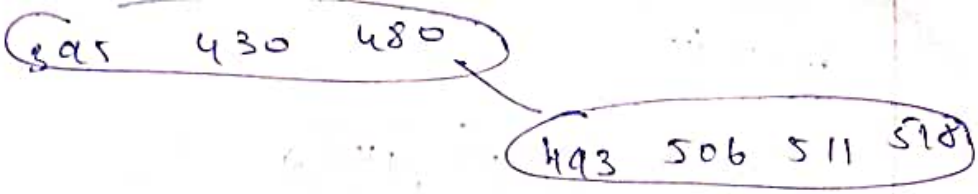
Insert : 382 ✓ 518 508 387 385
390

Insert: 382

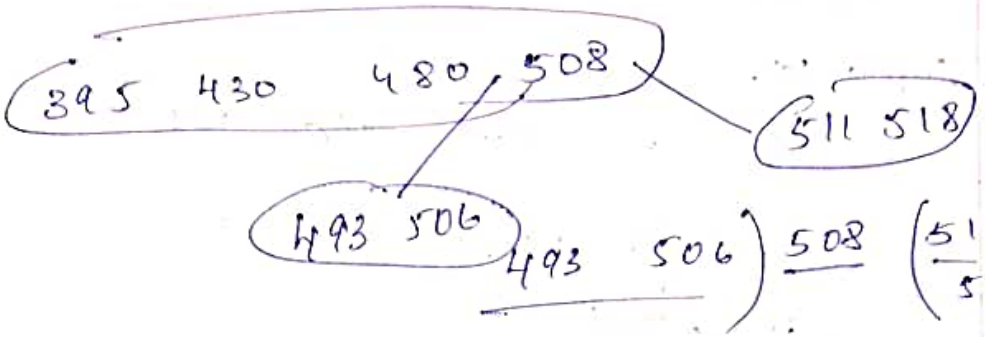
380 382 395 406 423



n/2 — left
n/2 — right
middle — parent.

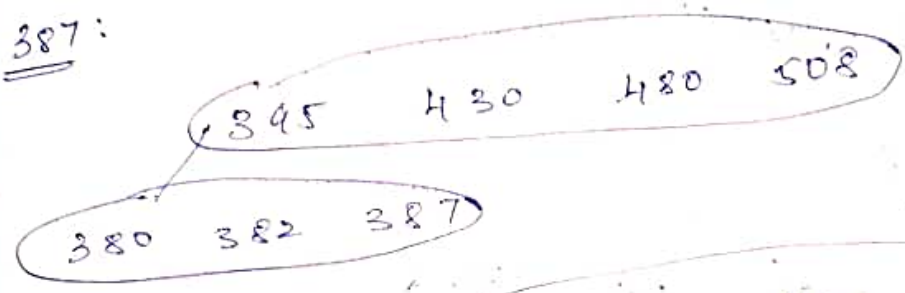


508:

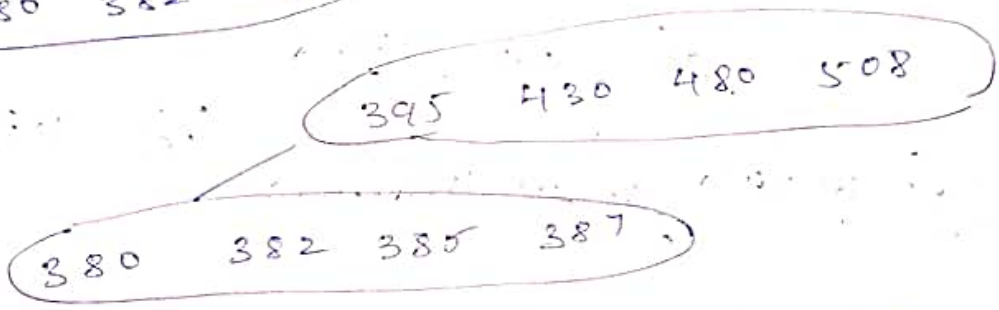


493 506 508 511 518

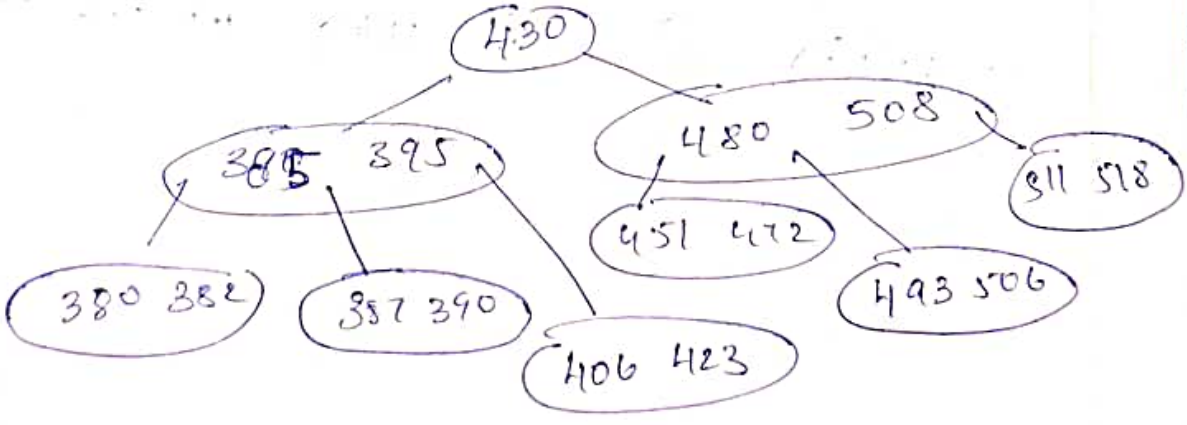
387:



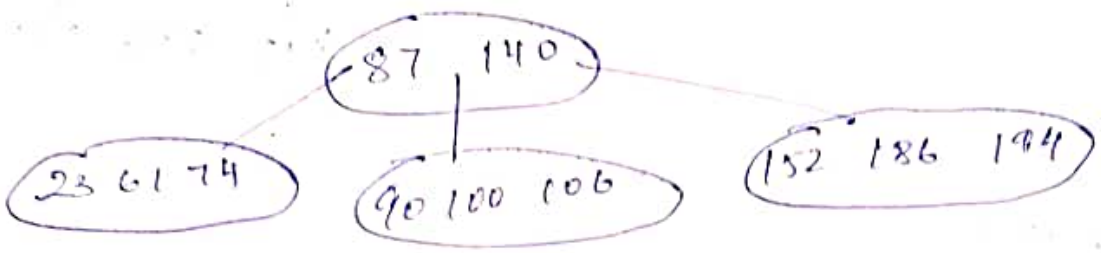
385:



390: 380 382 385 387 390 395 430 480 508



order 4:

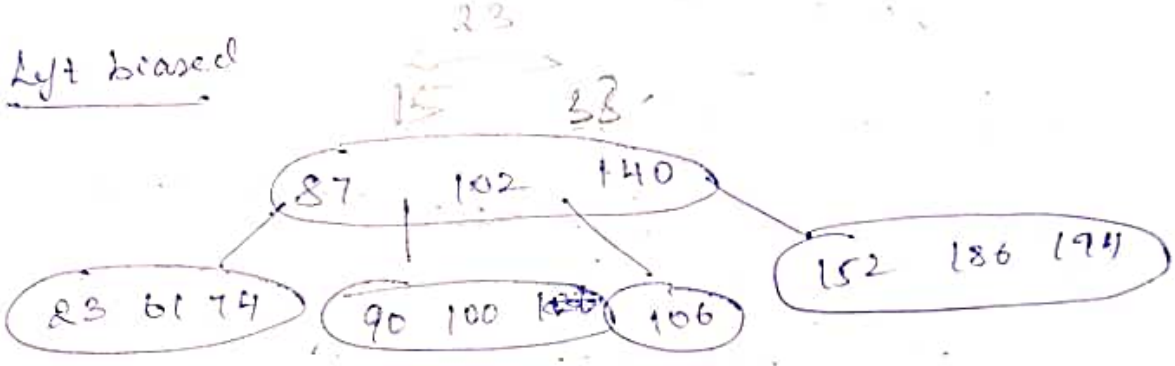


Insert 102:

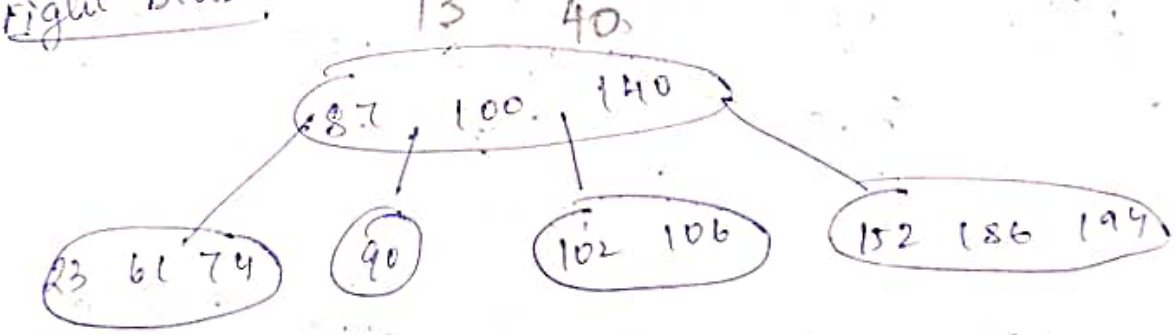
90 100 102 106

(i) left biased
(ii) right
(iii) alternative.

Left biased

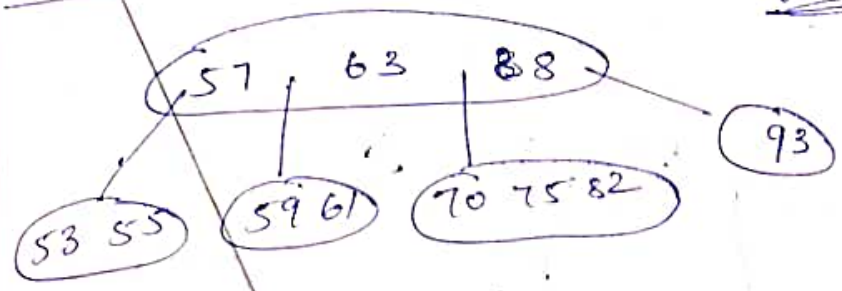


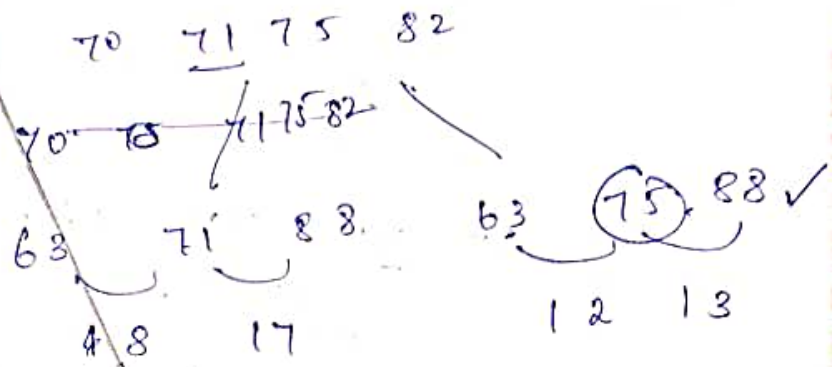
Right biased



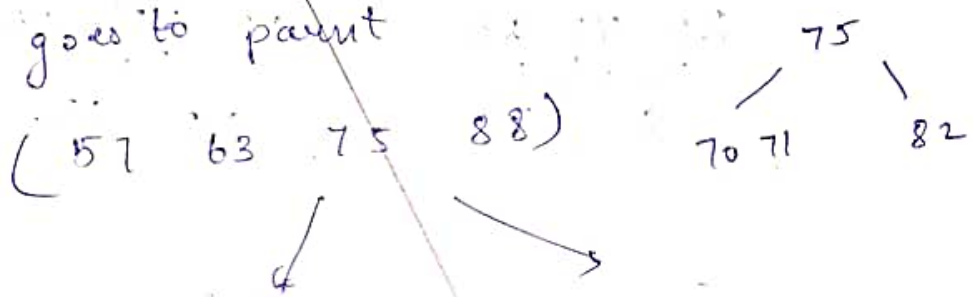
order 4

Insert: 71

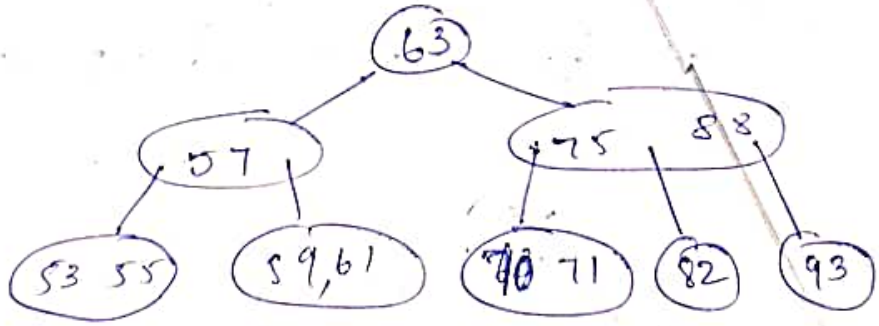




75 goes to parent



63 goes to parent

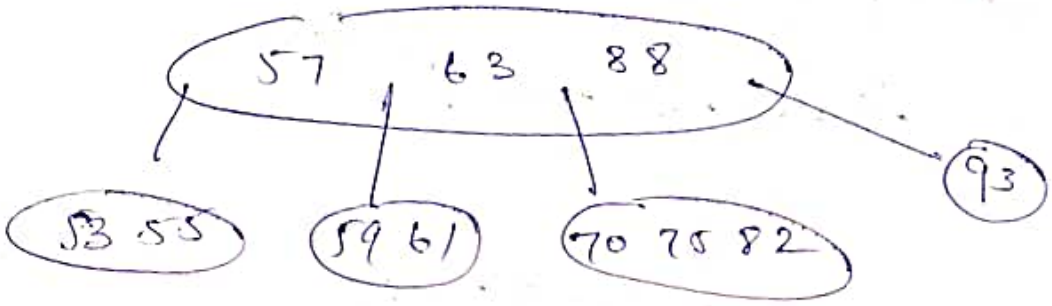


Root node : interval $0 - \infty$.

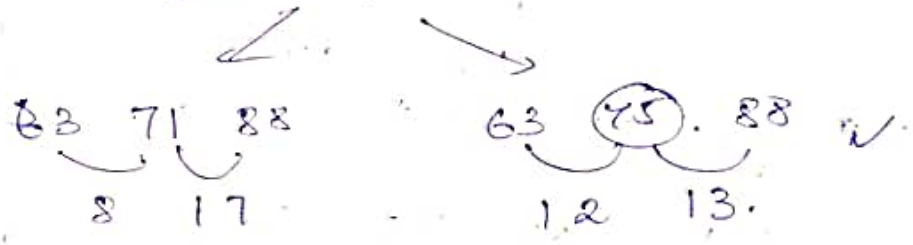
Insert: (Order: 5)

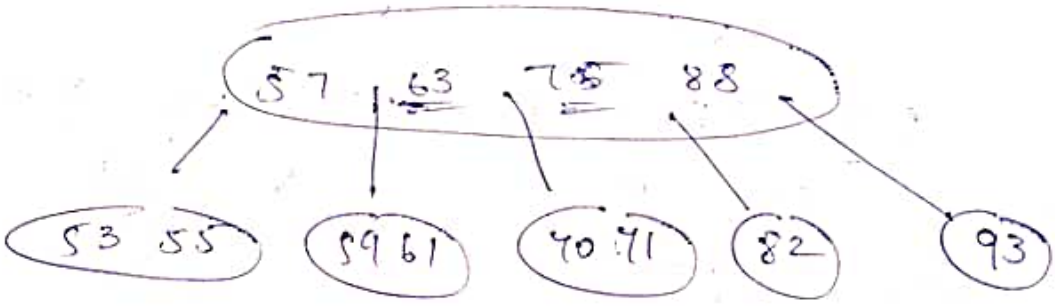
→ c n g a h e k q m f w l t z d
p r x y s.

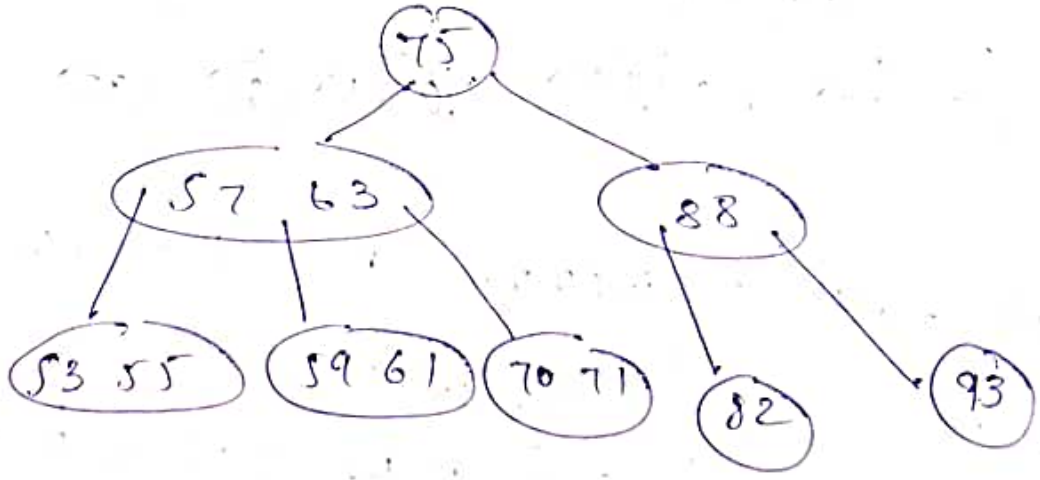
Insert 71 :-



70 71 75 82 :

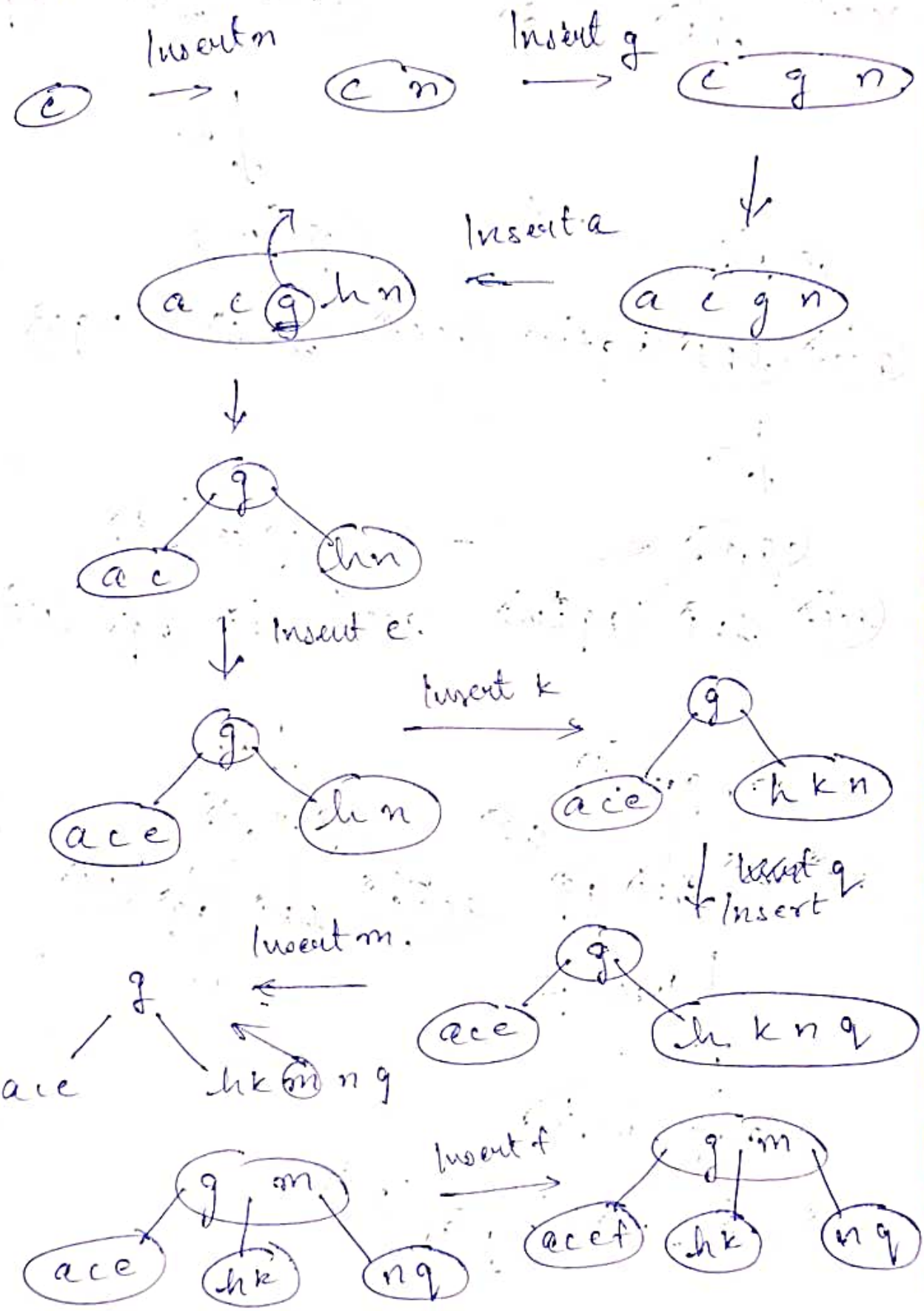




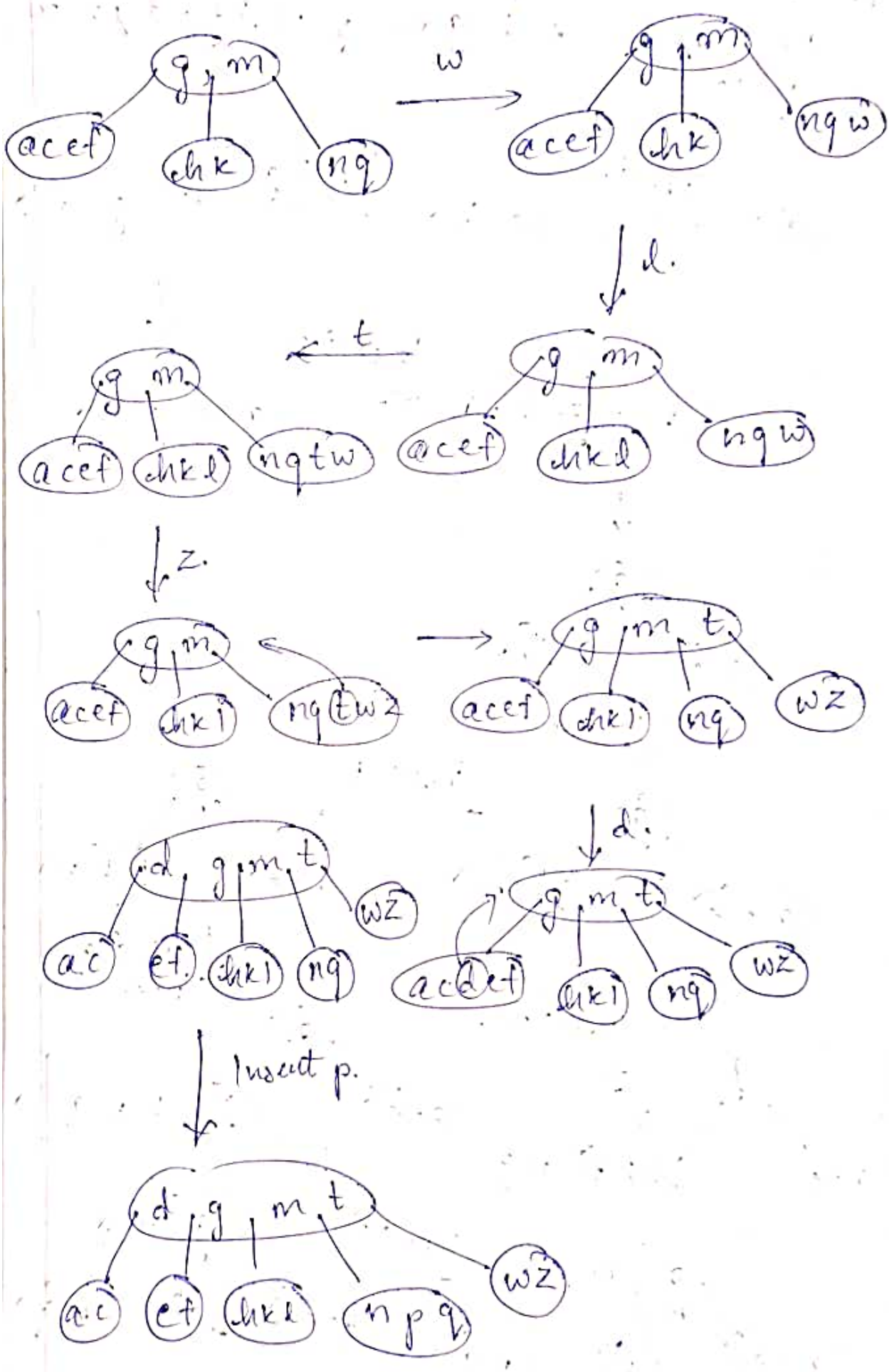


Insert: H/W Order: 5

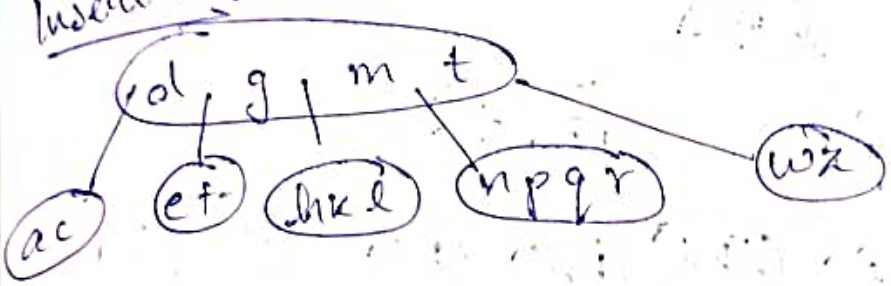
c n g a h e k g m f w l t z d p r x y s



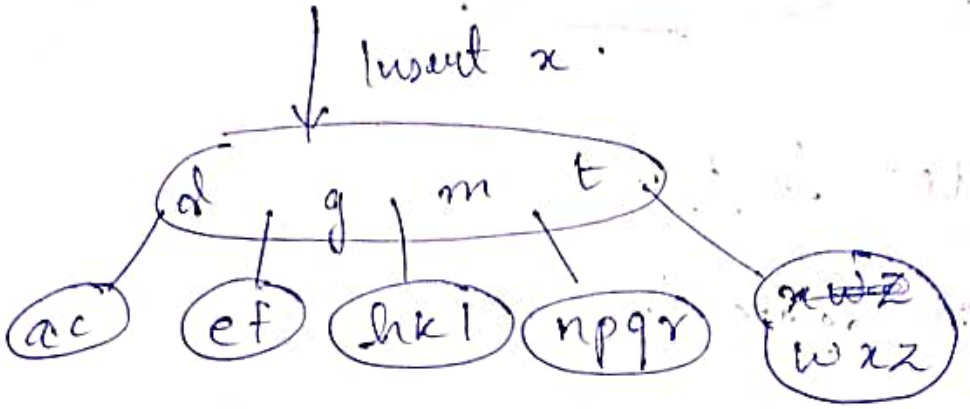
Insert w l t, z d p r x y s:



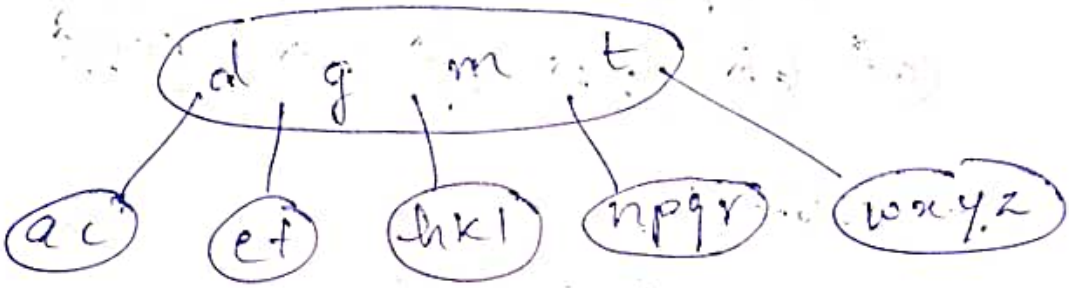
Insert r:



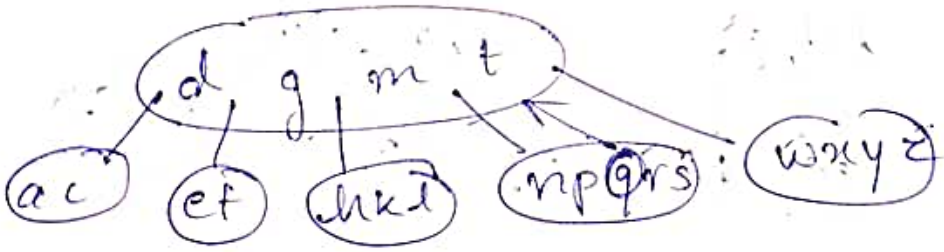
Insert x.

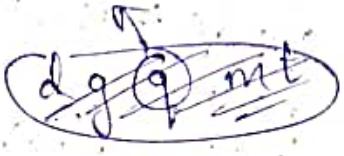


Insert y



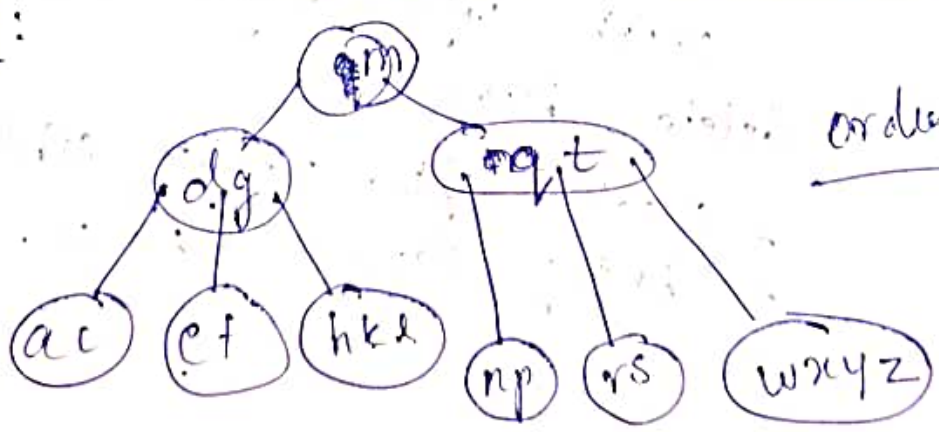
Insert s.



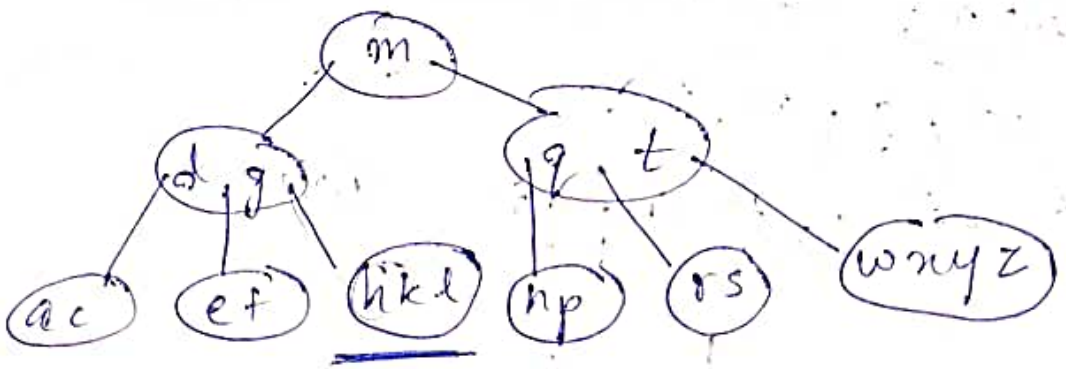


d g m q t

Final:

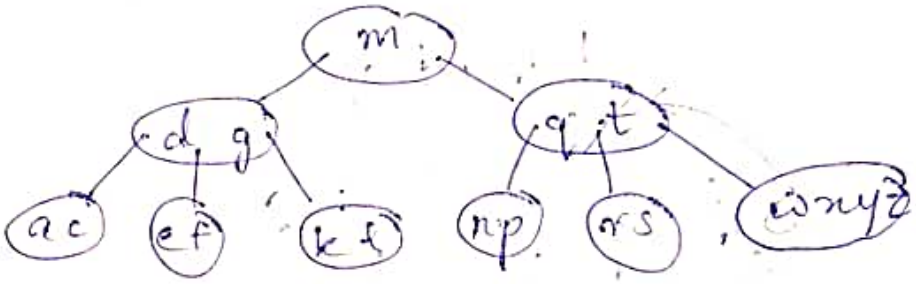


order = 5



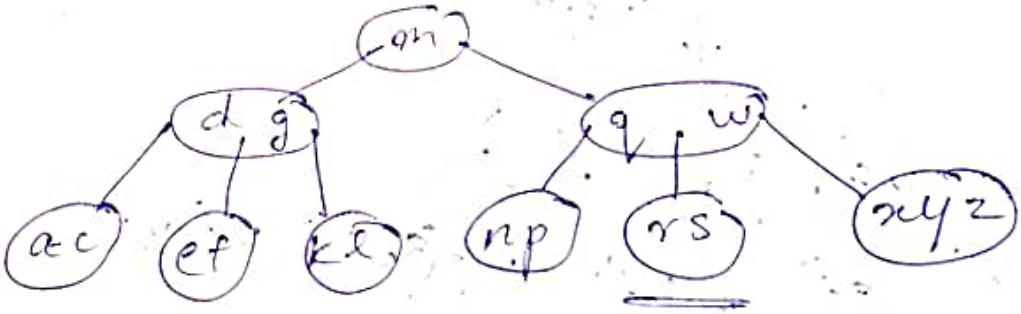
Delete: h, t, r, e

(i) delete h



(ii) delete t.

t → w.



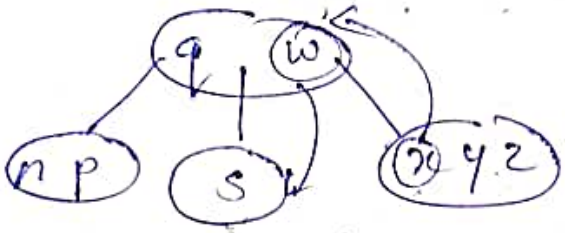
(iii) delete r. Redistribution

delete parent

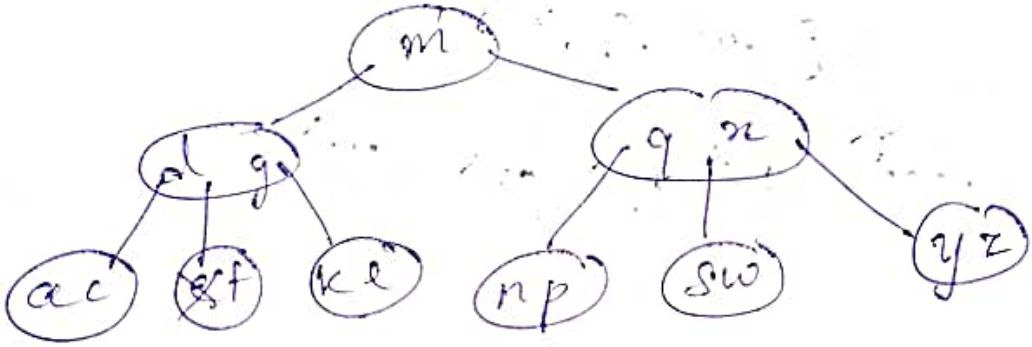
check left, right child.

delete leaf

* check left, right sibling.
* bring down parent.



(iv) delete 'e':



Merging + Redistribution.

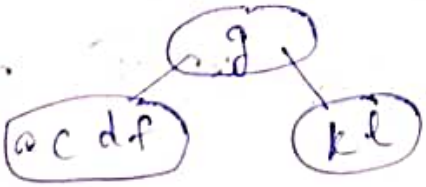
No extra keys in parent.

=>

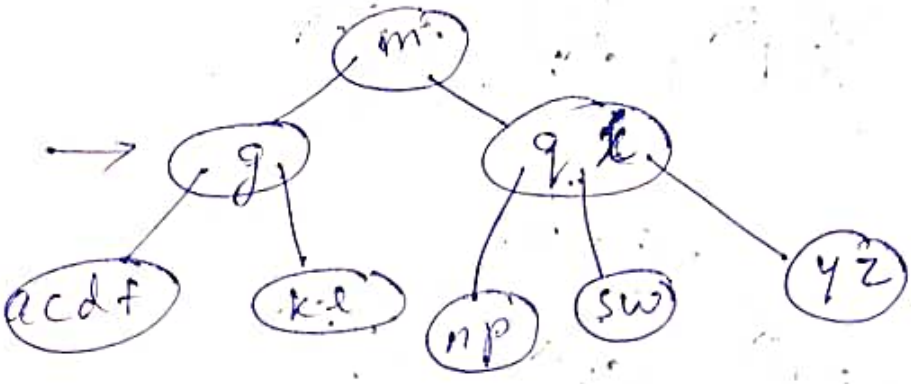
a c d f

acdf

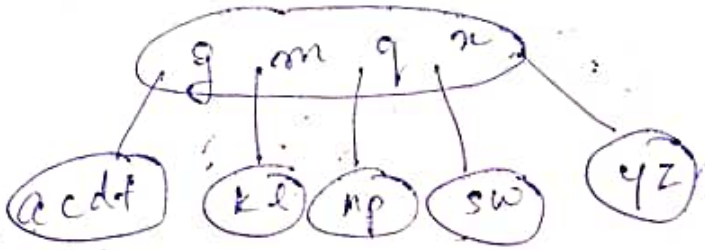
f g k l.



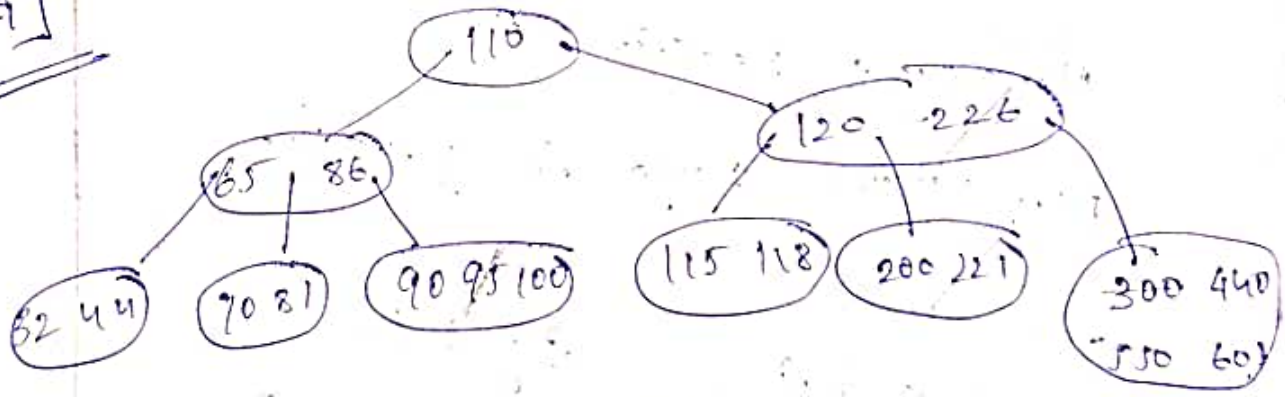
(*) see sibling of g.



(*) Not possible to have all distributes

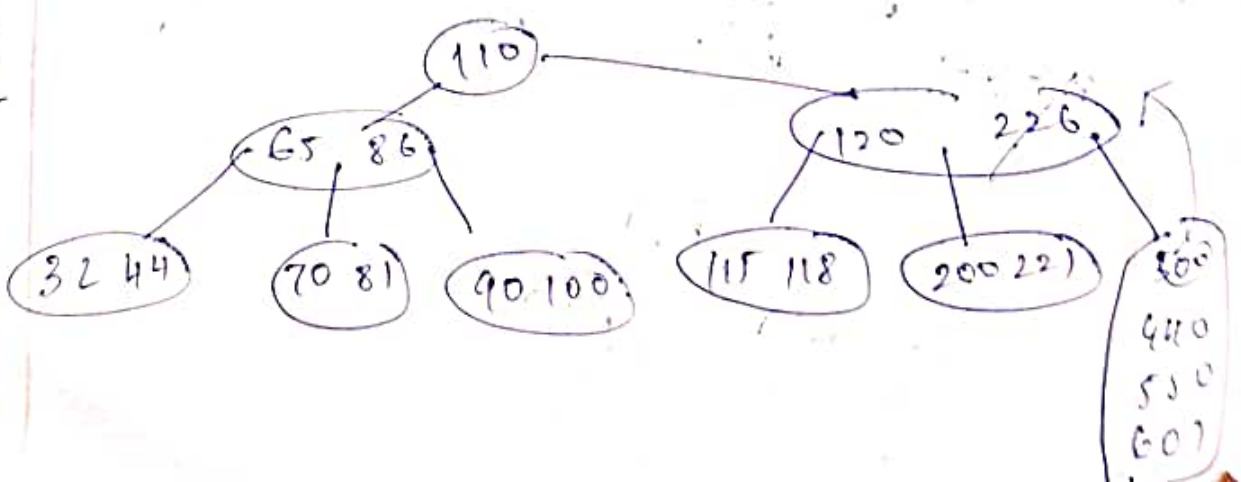


4]

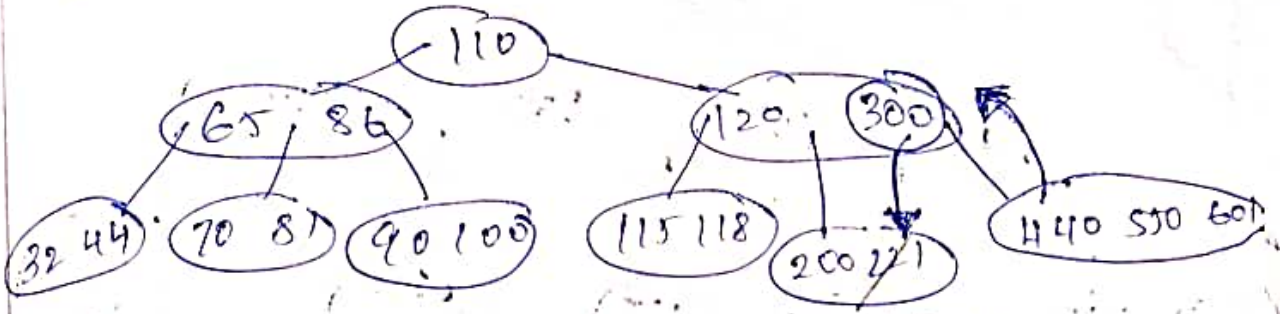


(*) delete 95, 226, 221, 70

B Tree - 5

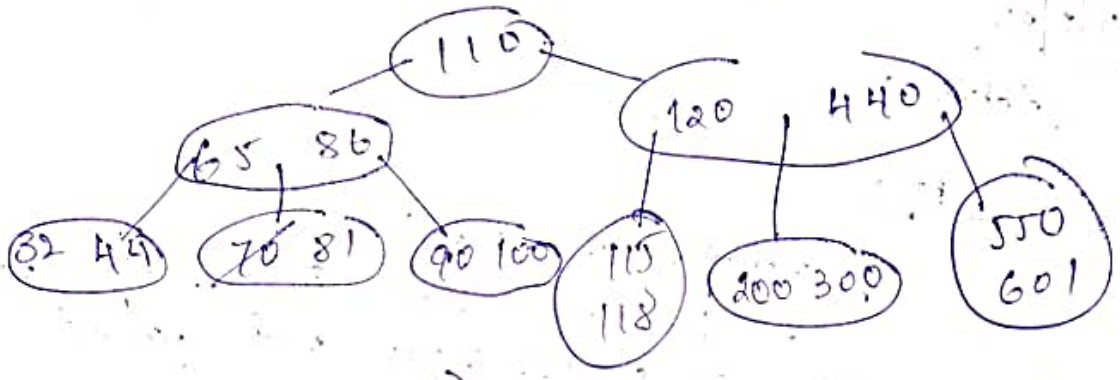


delete : 226



delete 221 :

difference
[data structur.
dbms]



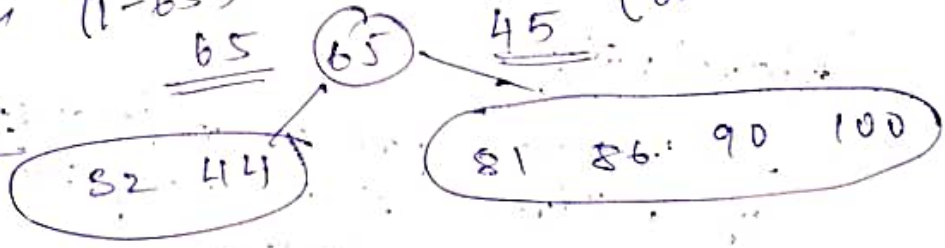
delete 70 :

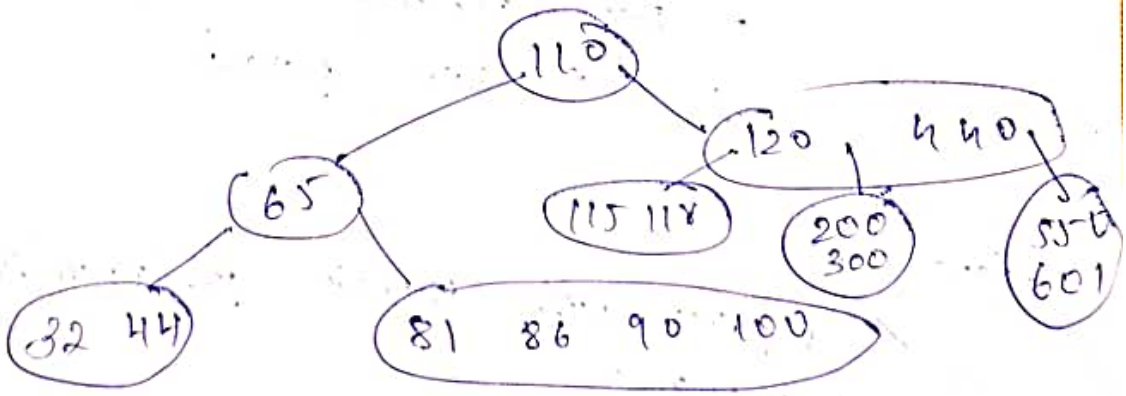
case 1:

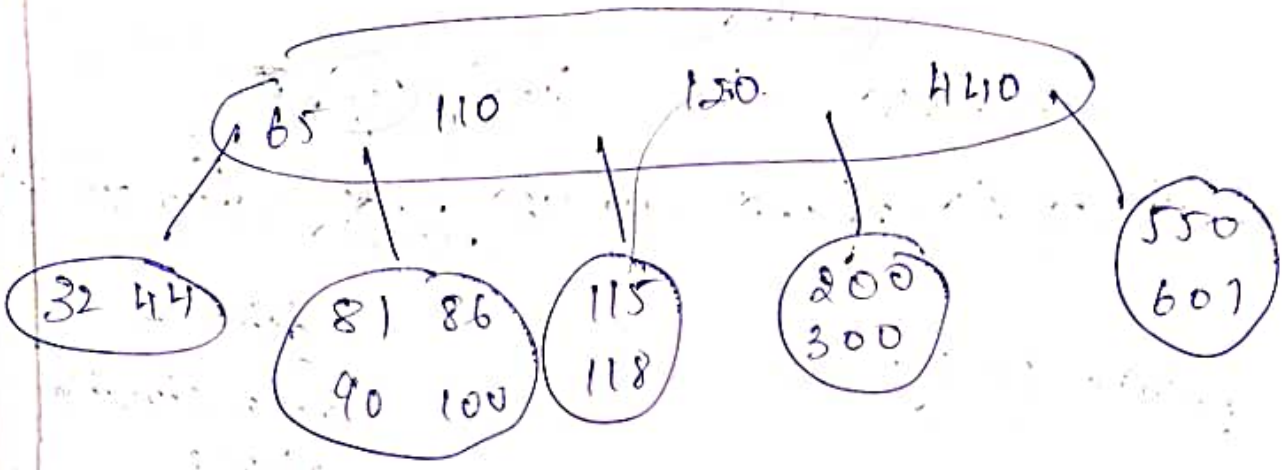
(1-86) 86 86 24 (87-110)

✓ (1-65) 65 65 45 (66-110)

case 2:







14/2/23

deletion:

leaf:

- If no. of nodes < min
  - borrow key from left / right
  - If excess keys not present
    - go to parent
      - If parent has less keys
      - merging.

internal node:

- inorder predecessor / successor.

If children ≤ min no of keys

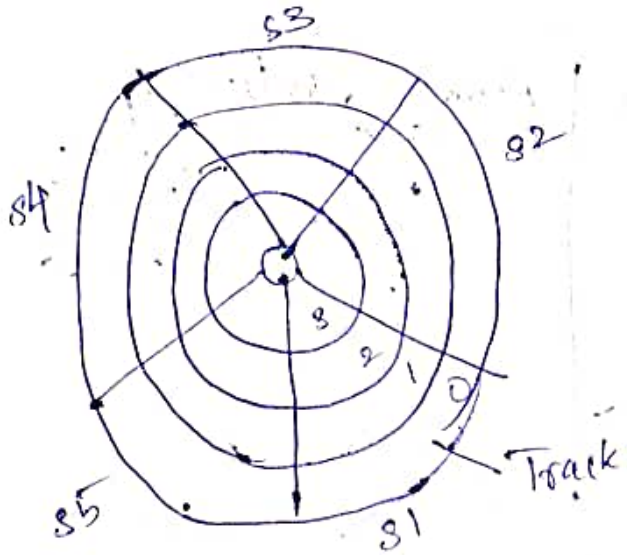
- see sibling
- else see parent

delete root:

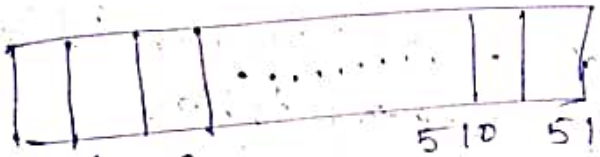
replace with inorder predecessor / successor.

- merging

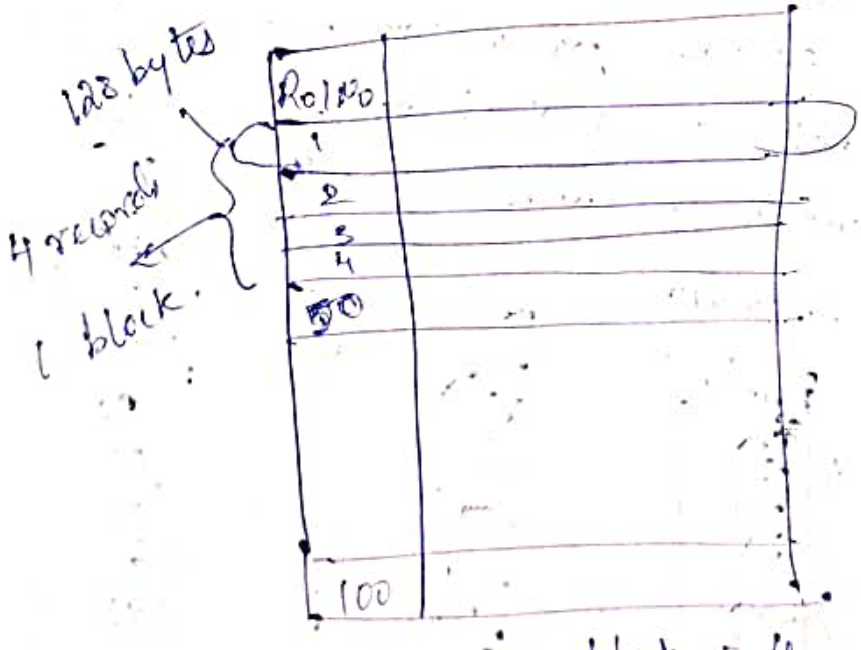
DS → organize data in : main memory.
data in secondary memory → dbms.



Block : 512 bytes.



Offset:



If ~~not~~ sorted.

No. of records per block = 4.

(*) 100 records := 25 blocks.

Indexing table

Roll Number	Record pointer
1	
2	
3	
:	
50	
:	
100	

each row : 16 bytes

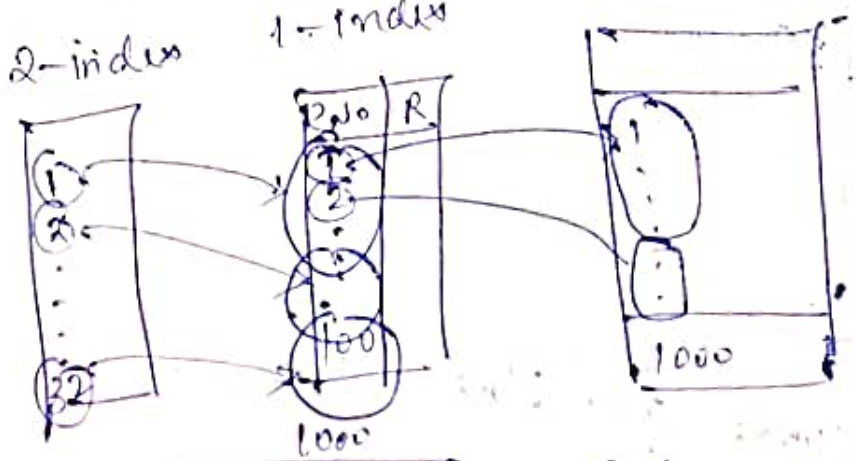
no. of rec / block := 32 records / block for indexing table.

100 records = 4

4+1 = 5 blocks to be accessed to search for record.

25 blocks ⟶ 5 blocks

If no. of records = 1000

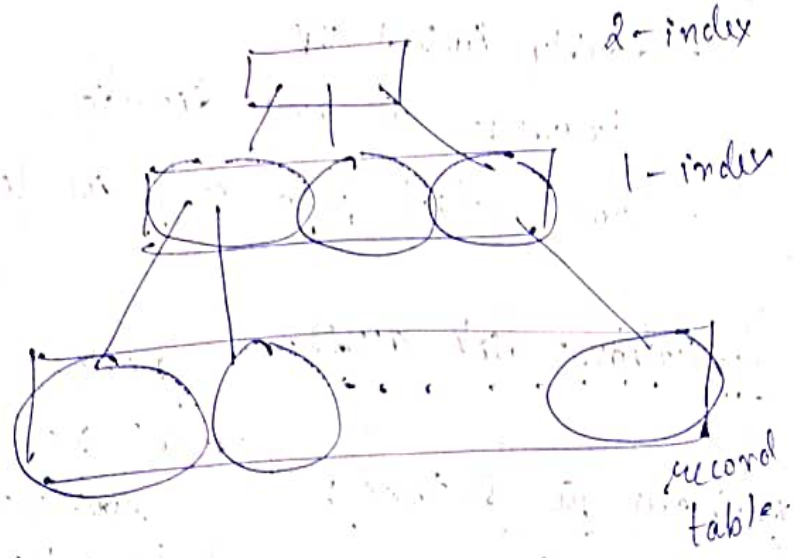


992 / 31 = 32 / 96

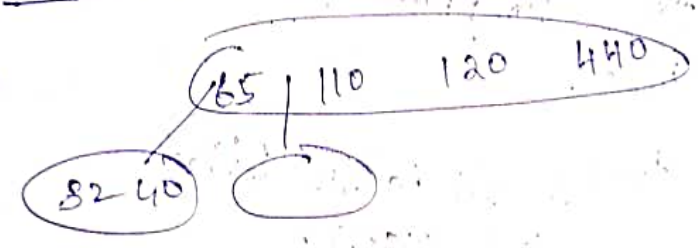
1000 / 31 = 32 / 8

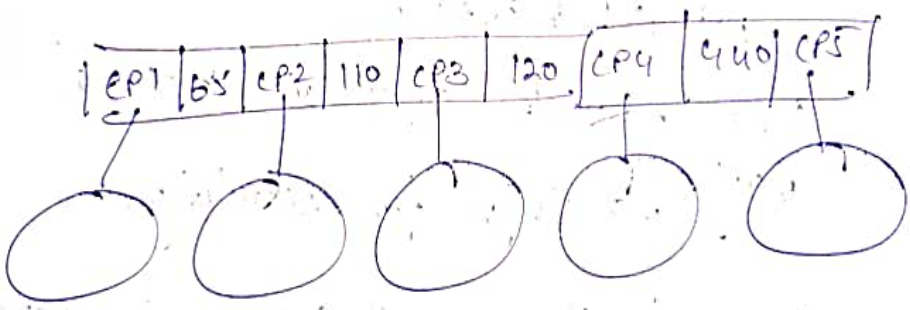
[1+1+1] := Only 3 blocks needed to access

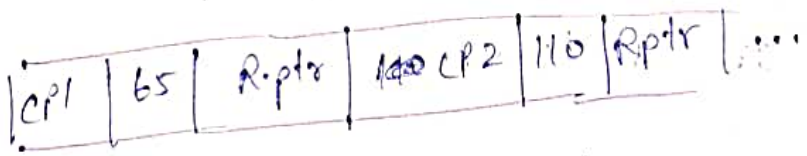
B Tree:



Example:







(*) Self adjusting Binary tree.

B tree.
B+ tree
B* tree.

} Used in DBMS.

B+ trees

- Doubly linked list
  traverse in any direction.
- store actual info only in leaf.

20/2/23

Applications: disk access

Problems in B tree :-

* More operations when insertion / deletion compared to B+ tree.
* B+ tree: all leaves linked. sequential access.

Not in B tree:

Some elements copy in internal node. => Useful for searching.

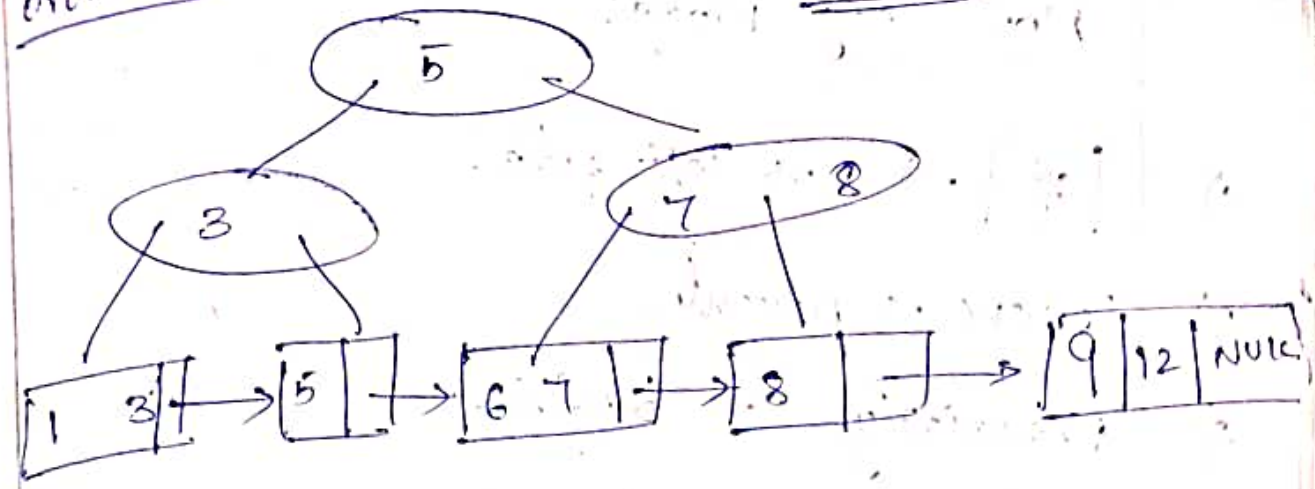
Min and Max no. of keys. is same as B tree.

B+ tree

* all key in leaves.
* Some keys replicated in internal nodes.

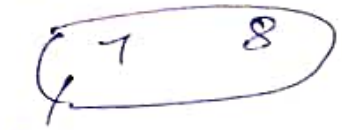
order: 3

B+ tree:



Search: 6

root: 5 go right.



6 < 7, go left of 7.

6 is found.

Search 5:

root == 5 ✓ <=

go to left

5 > 3 goto right.

Accessing sequentially : Not possible in B trees.

Leaf:

1) $\left\lceil \frac{n-1}{2} \right\rceil$ left side.

2) Copy smallest of remaining → parent.

3) remaining → right.

Non leaf Node:

1) $\left\lceil \frac{n}{2} \right\rceil - 1$ ⇒ left side.

2) move ⇒ parent

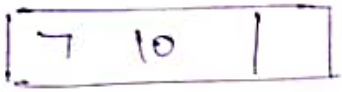
3) remaining ⇒ right.

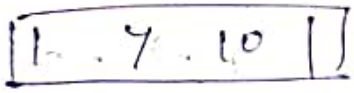
Order 5 Min: 2 Max: 4

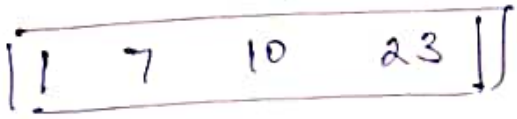
Insert:

7, 10, 1, 23, 5, 15, 17, 9, 11, 39, 35, 8, 40, 25

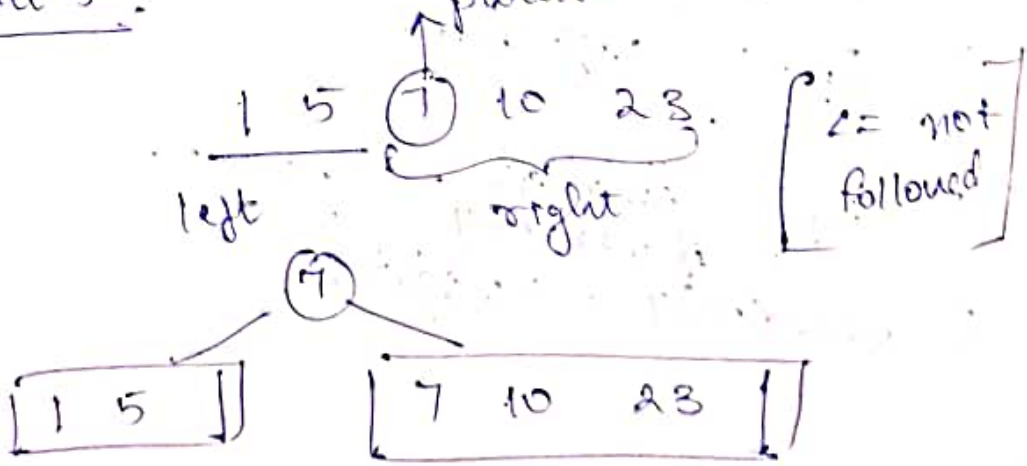
(1) Insert 7

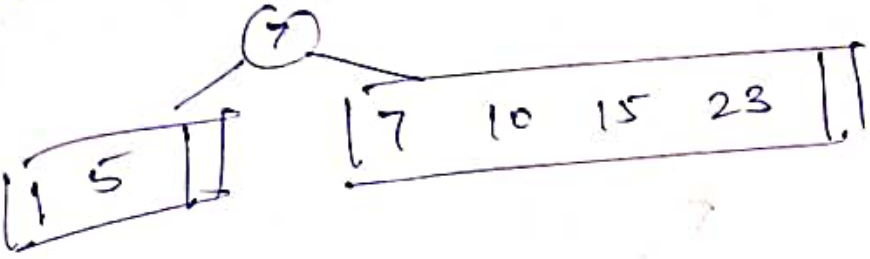


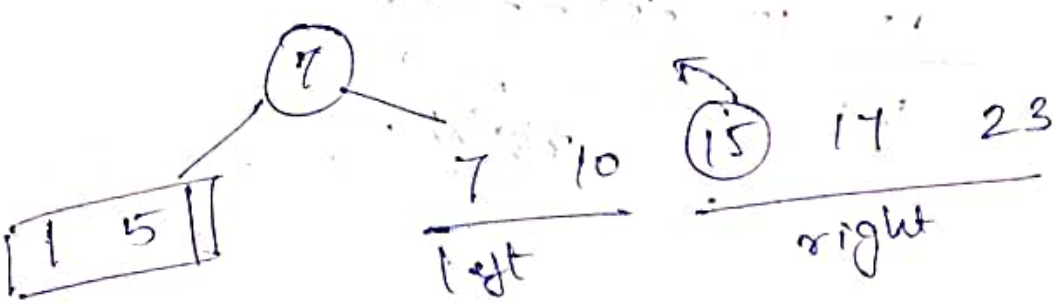


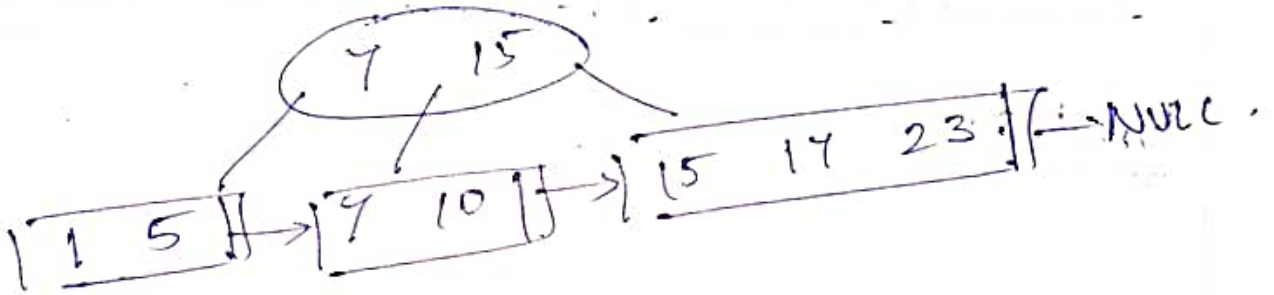


Insert 5:

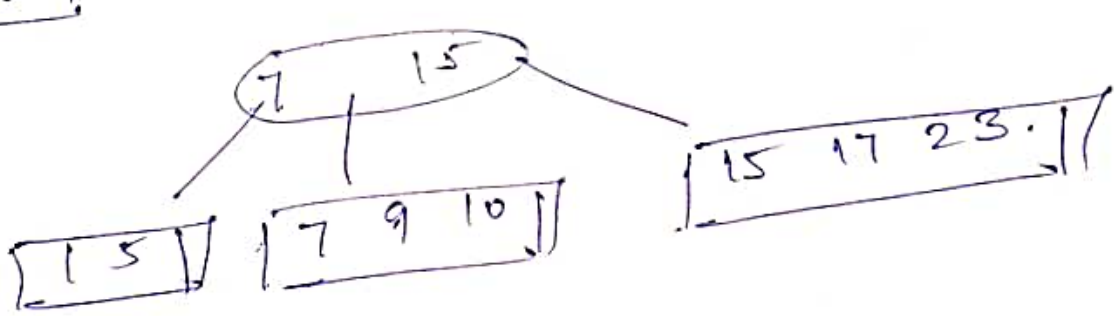




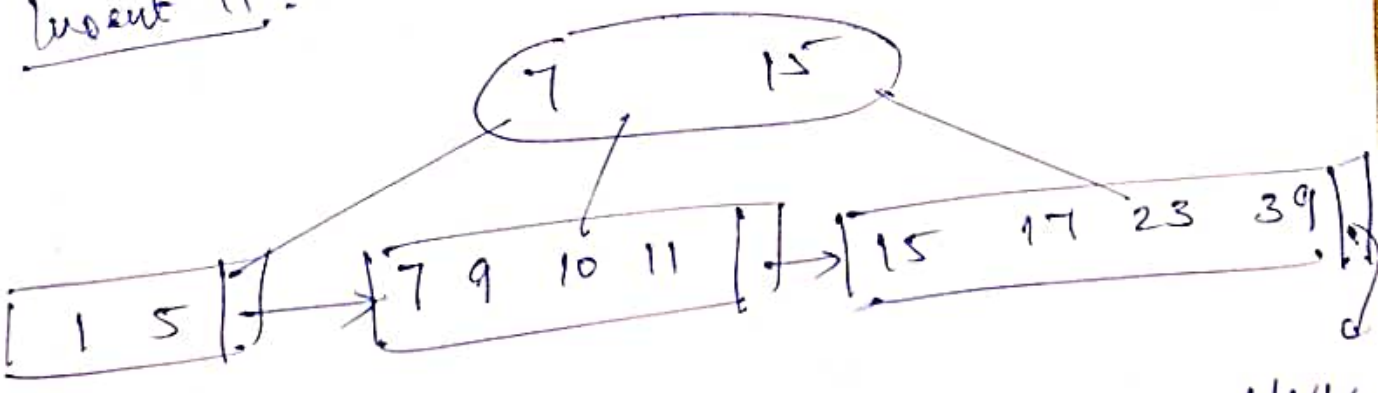




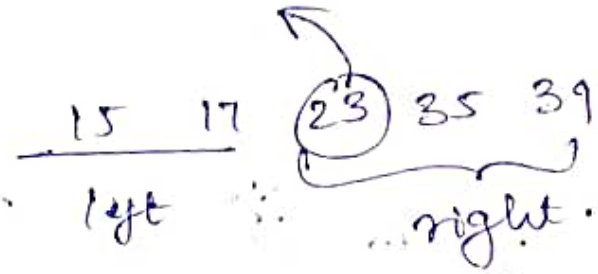
Insert 9:

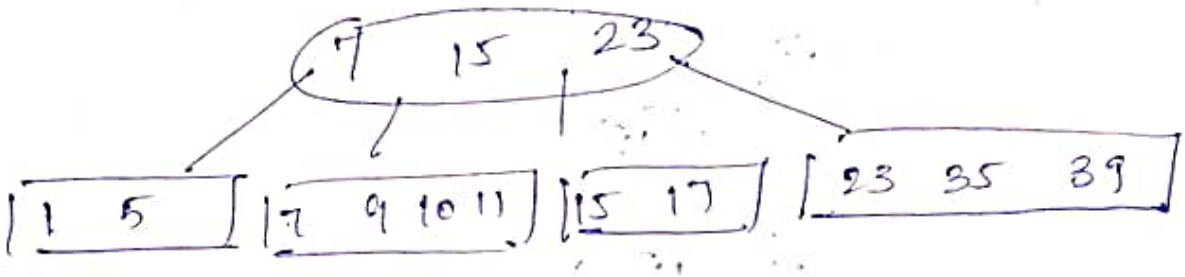


Insert 11:



Insert 85:



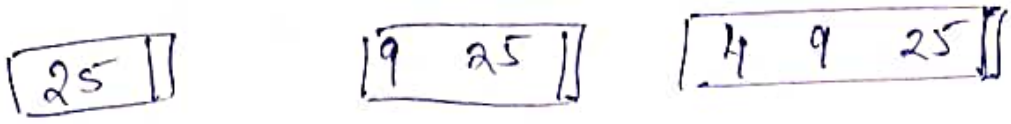


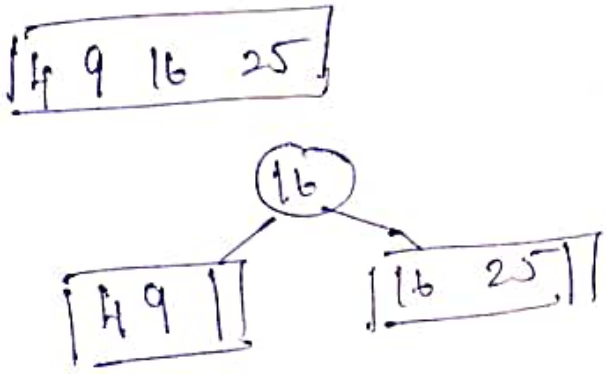
Insert 8:

Insertion: Order: 4

25 9 4 16 1 20 13

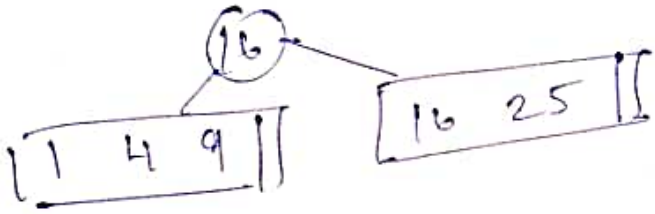
15 10 11 12 18.

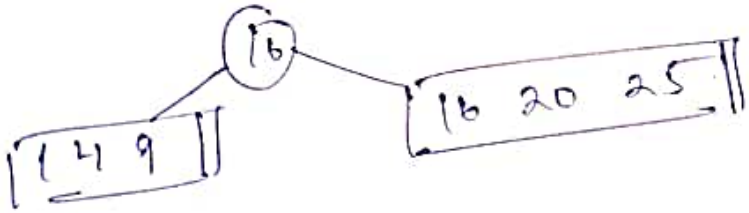




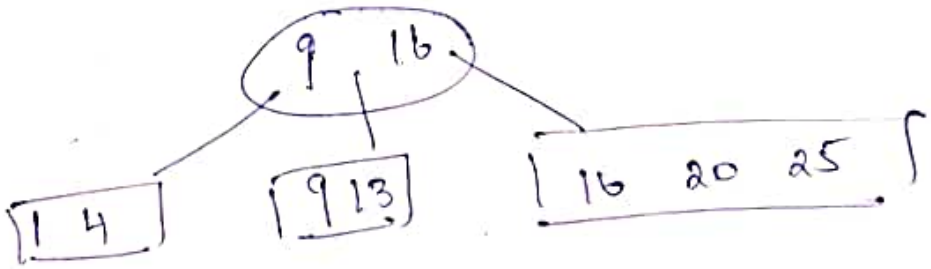
$\left\lceil \frac{4-1}{2} \right\rceil = 2$

$\left\lceil \frac{4}{2} \right\rceil - 1 = 1$



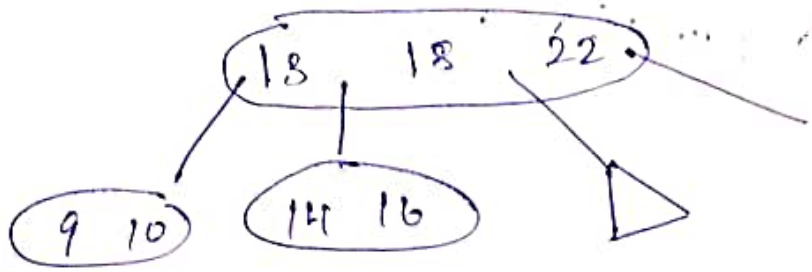


$\frac{1\,4}{2}$ (9) 13

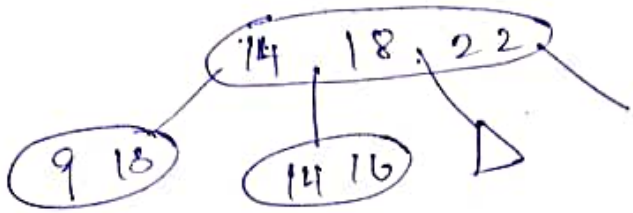


Deletion:

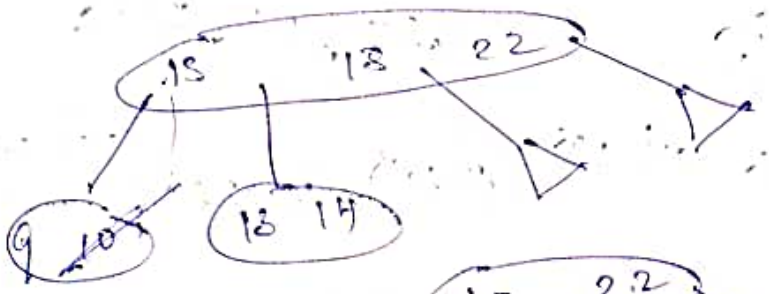
1)



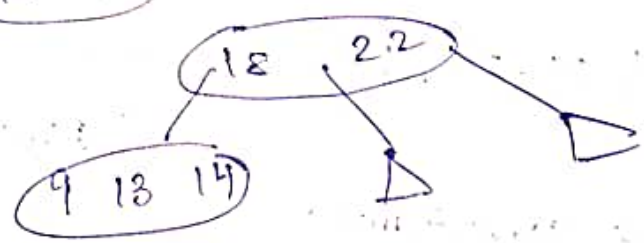
delete 10:



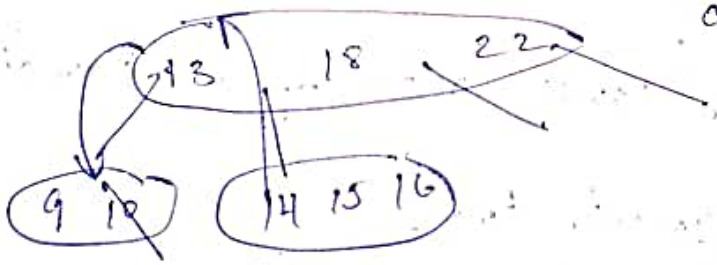
2)



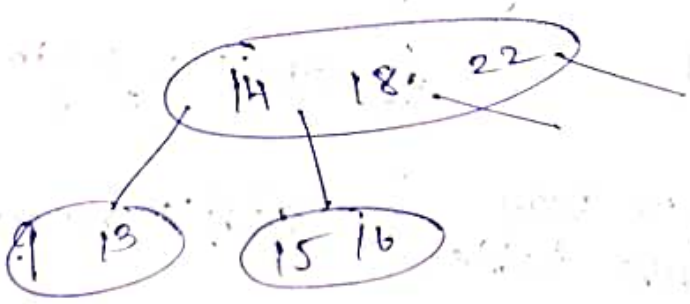
delete : 10.



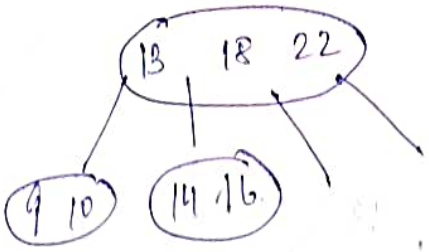
3)



delete : 10.

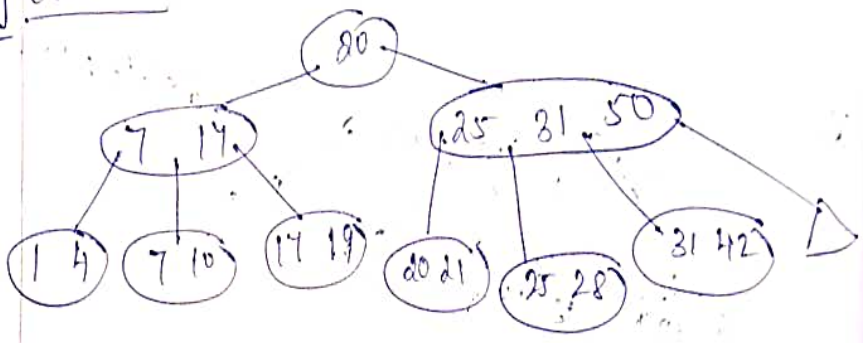


4)



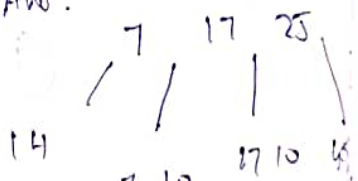
delete 10. →

5] order 5:



28, 31, 21, 25, 19.

Ans:



(*) Range operations:

Q. (5, 9). ?

min
sum
avg
mean

includes lower and upper bounds.

Segment tree

Range operations.

Q (5, 9)

dynamic : segtree

↳ prefix sum
Sparse table
BIT

static array

Binary Indexed Tree.

Dynamic array → segment tree.

22/2/23

Prefix Sum array :-

	0	1	2	3	4	5
A:	1	3	1	0	2	1

Sum(2,4)

Reduce number of access in array.

Step ①: preprocessing in array:

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

$Sum(a,b) = sum(0,b) - sum(0,a-1)$

Initial condition

$sum(0,-1) = 0$

$Sum(2,4) = sum(0,4) - sum(0,1)$

$= 7 - 4$

$= 3.$

* This idea can be extended to N-dimensional array.

Sparse table:

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

(1) preprocessing

Number of columns = $\lfloor \log_2 N \rfloor + 1$

$N = 6$

Colu $= \lfloor \log_2(6) \rfloor + 1$

$= 2 + 1$

$= 3$

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

i \ j	0	1	2
0	0	0	(2)
1	1	2	2
2	2	2	2
3	3	4	
4	4	4	
5	5		

$i=0 \quad 2^j \quad 2^0 = 1$

$i=1 \quad 2^j = 2^0 = 1$

$i=0 \quad 2^1 = 2$

$i=1 \quad 2^1 = 2$

min (3, 5)

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

No. of elements $\Rightarrow 3$

$i \Rightarrow 3$

$j = \lfloor \log_2 3 \rfloor \Rightarrow 1 \quad (3, 1)$

$= 4$

$i = 4 \quad j = 1$

min (0, 5) & 6 elements

$i = 0$ $j = \lfloor \log_2(6) \rfloor = 2$.

4 elements considered.

$a[0] = 1$

2 more elements to check.

$i = i + 2$.

$\therefore 2, \quad j = 2$.

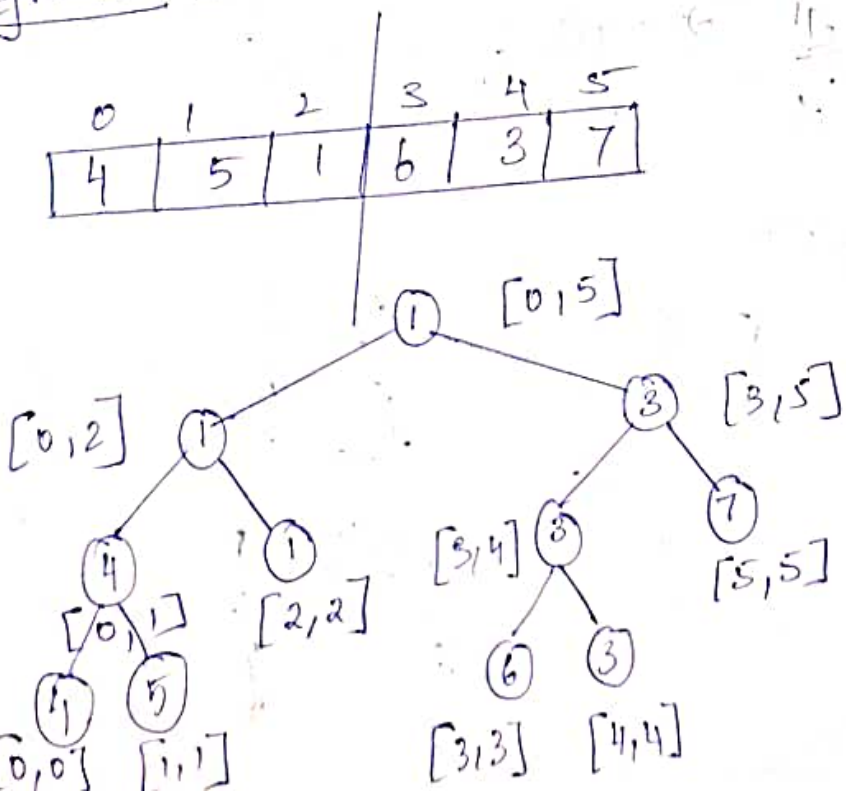
$2^2 = 4$

$(2, 2) = 2$.

$a[2] = 1$

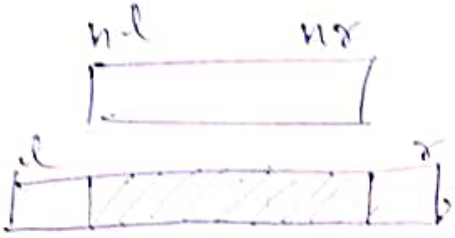
Minimum $= \min(1, 1) = 1$

Segment Tree:



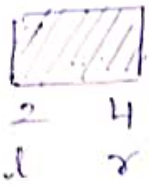
Complete overlapping
Partial Overlapping
No Overlapping ⟶ return value based on type of query

l r.
min(2, 4)

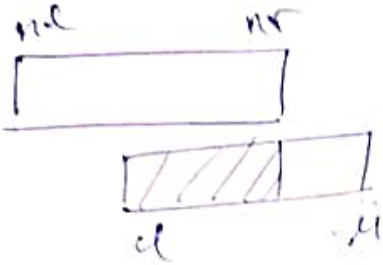


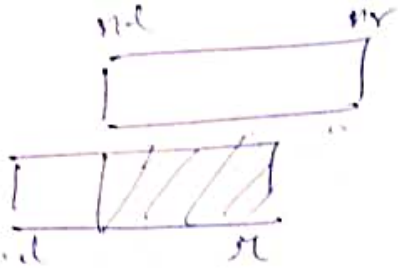
Complete overlapping

partial.

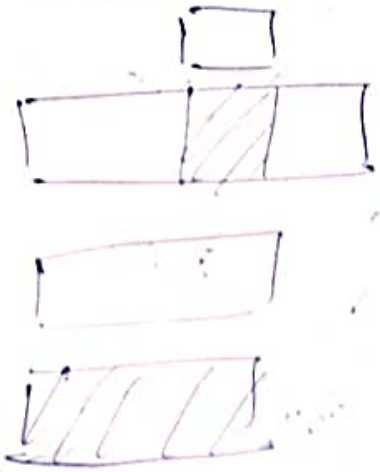


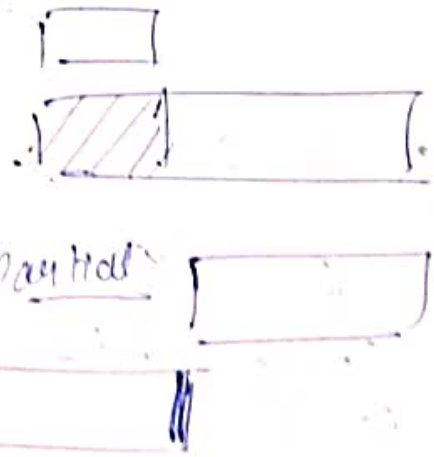
Partial:-





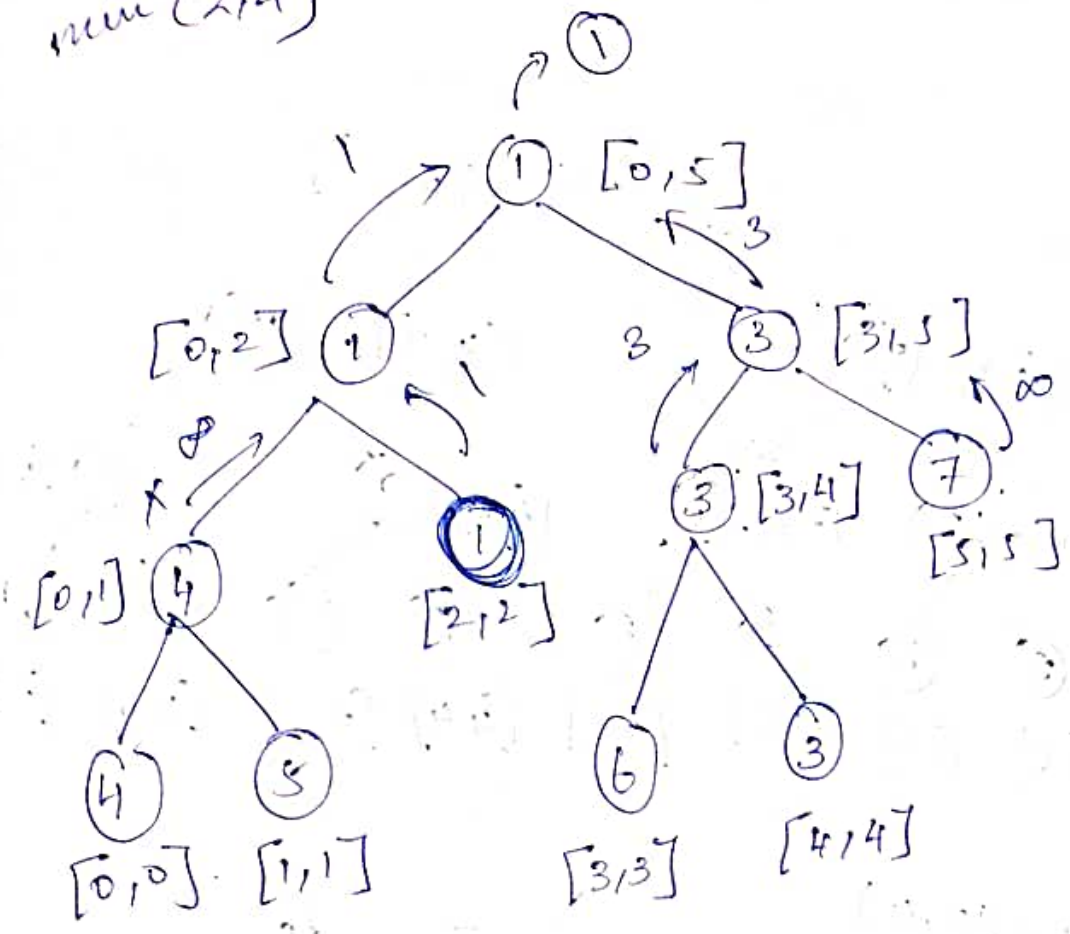
Complete:





Partial

min (2,4)



Minimum = 1

2

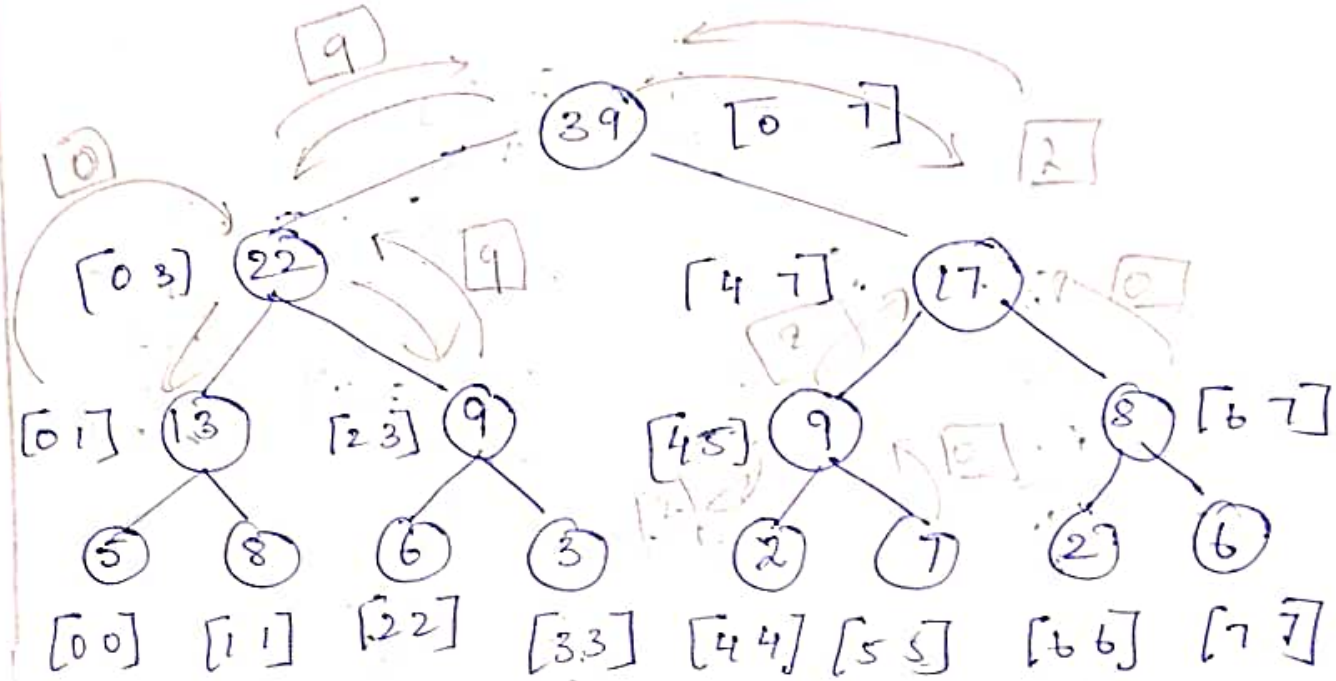
Partial Overlapping → go left, right

total overlapping → return value.

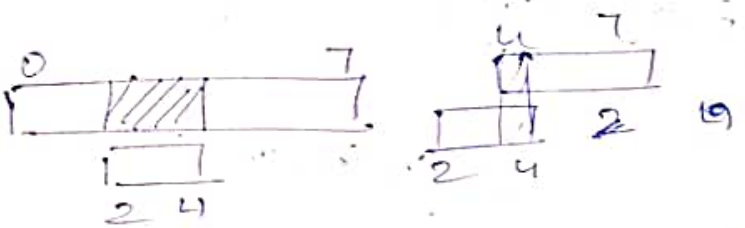
no overlapping → return ∞.

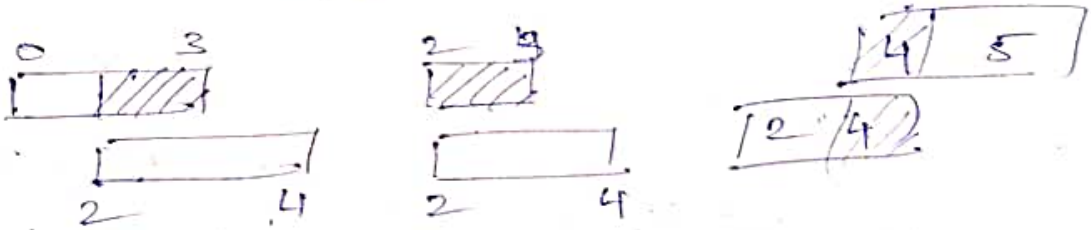
[5 8 6 3 2 7 2 6]

Sum (2,4)

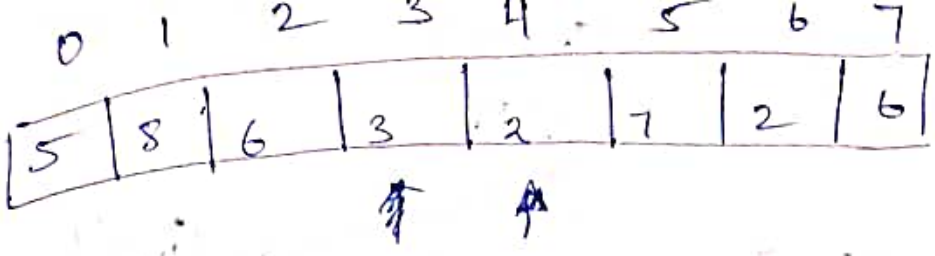


sum (2, 4)



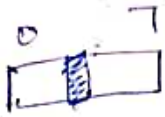


Ans: 11 = 9 + 2



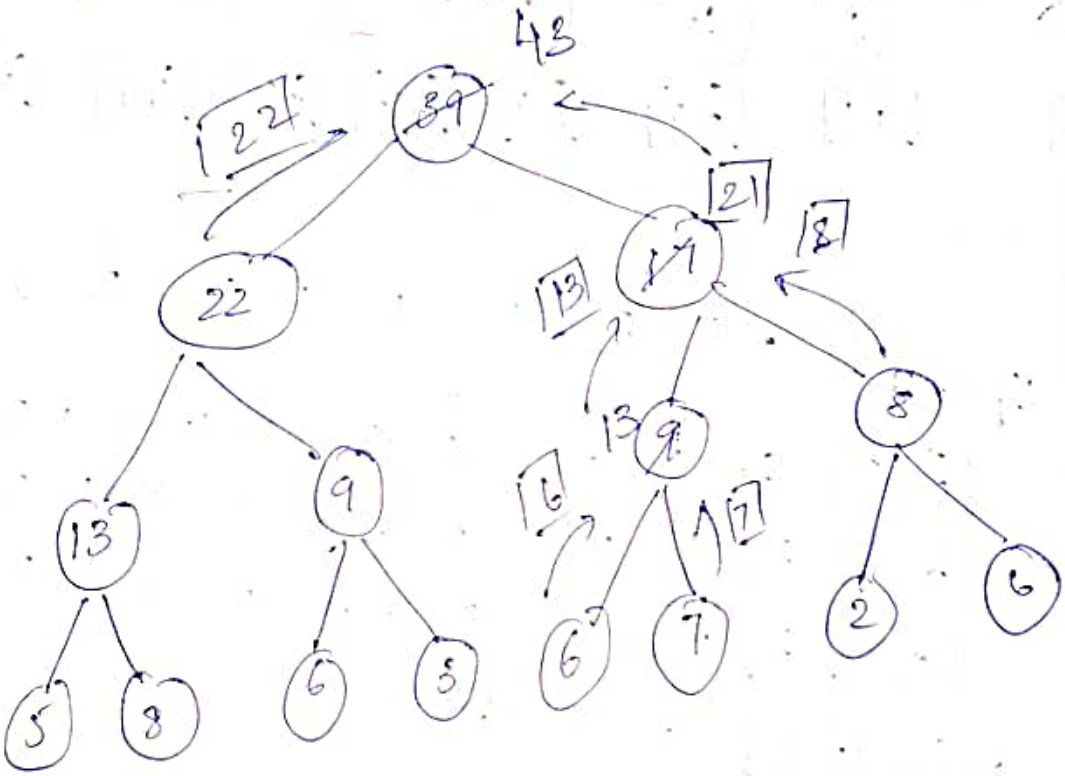
Update 2 with 4.

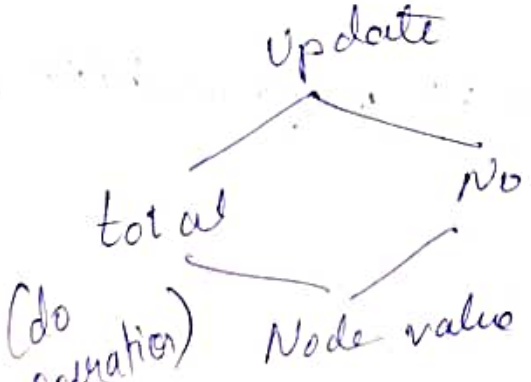
[4, 4]



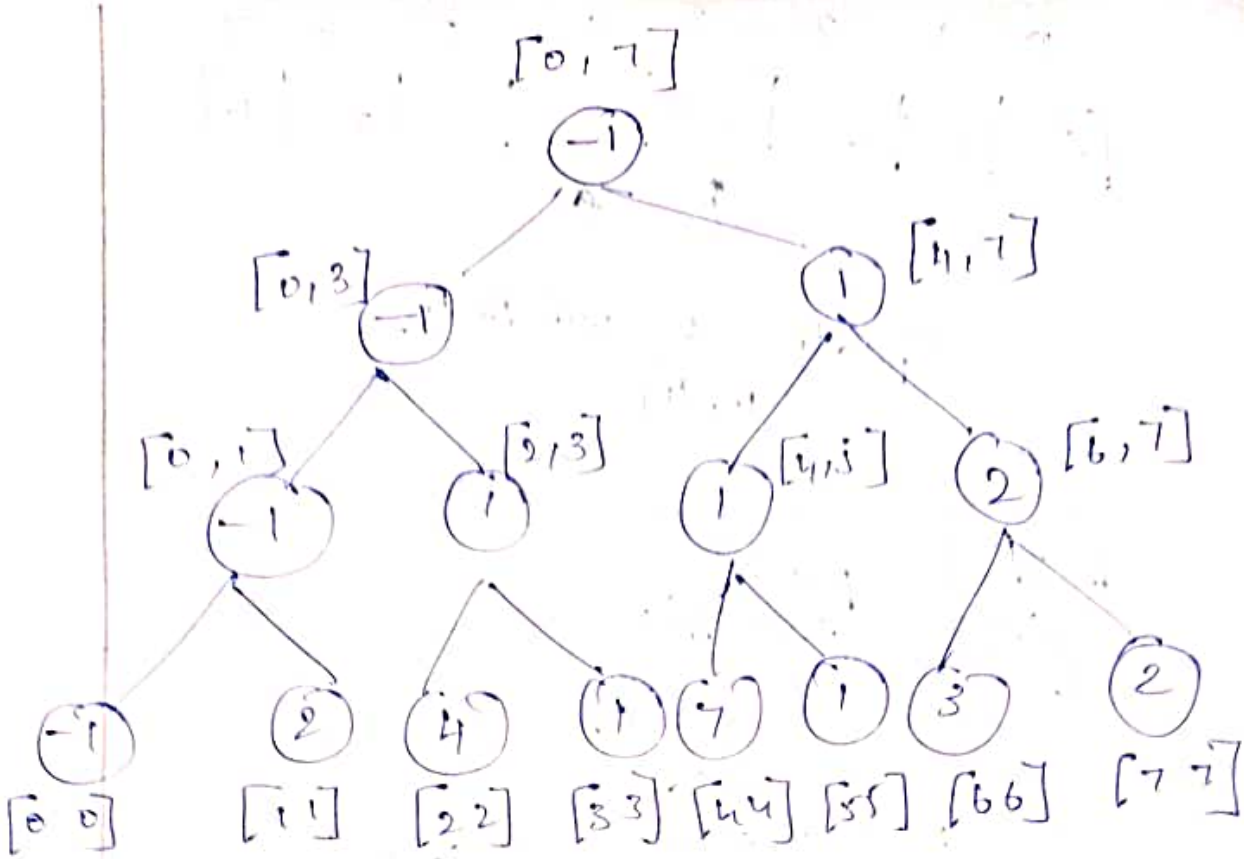
[4 4]

partial overlapping





return updated value.



-1 2 4 1 7 1 3 2

(i) Update

[0,3] by 3 => 2 5 7 4 7 1 3 2

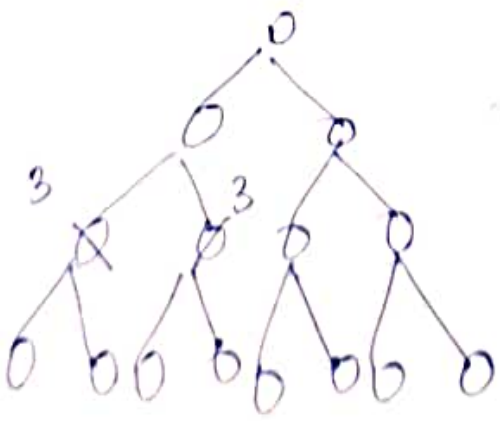
[0 3] by 1 => 3 6 8 5 7 1 3 2

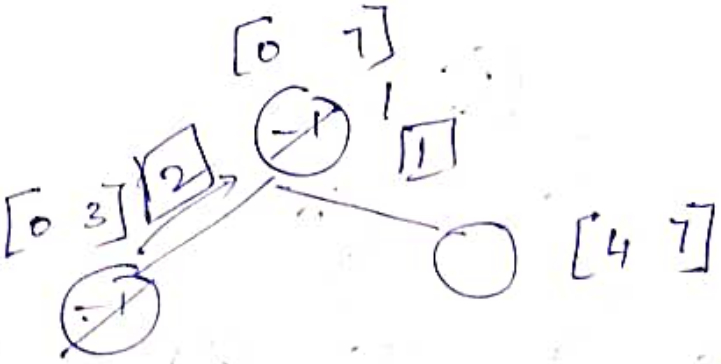
[0 0] by 2 => 5 6 8 5 7 1 3 2

min [3 5] ⟹ (1)

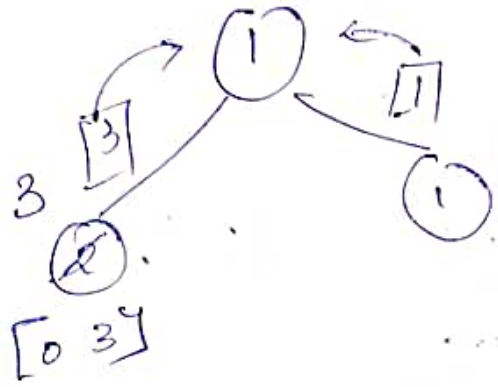
Lazy Tree

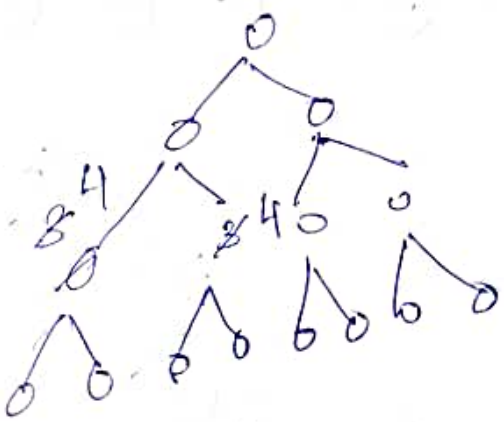
Implementation: array representation,

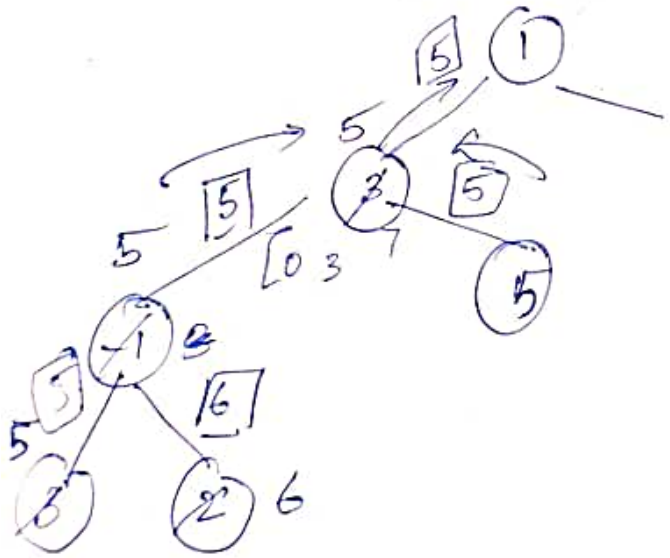


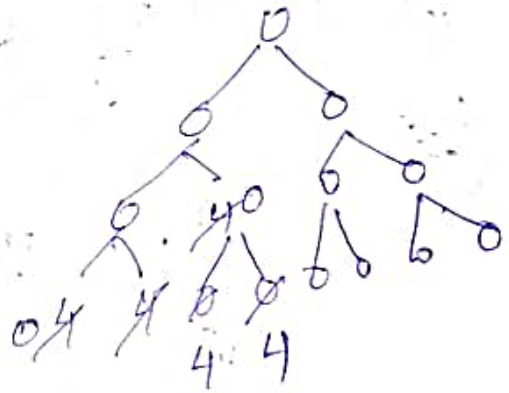


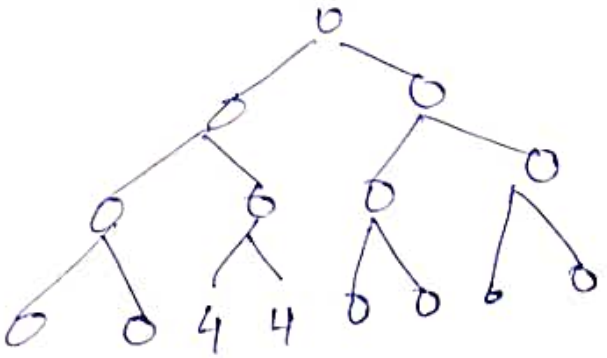
O(N) + O(A)

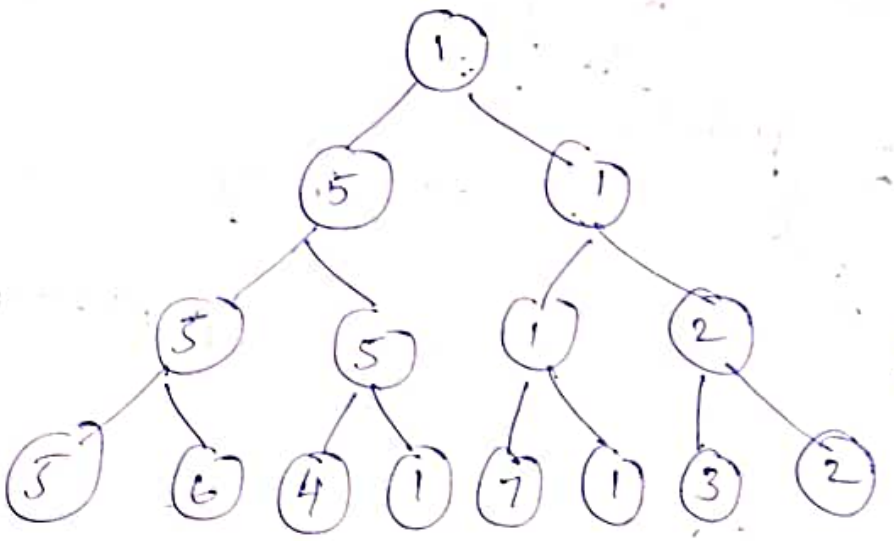


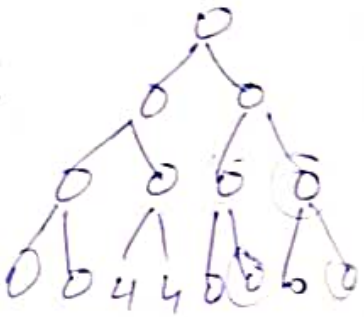


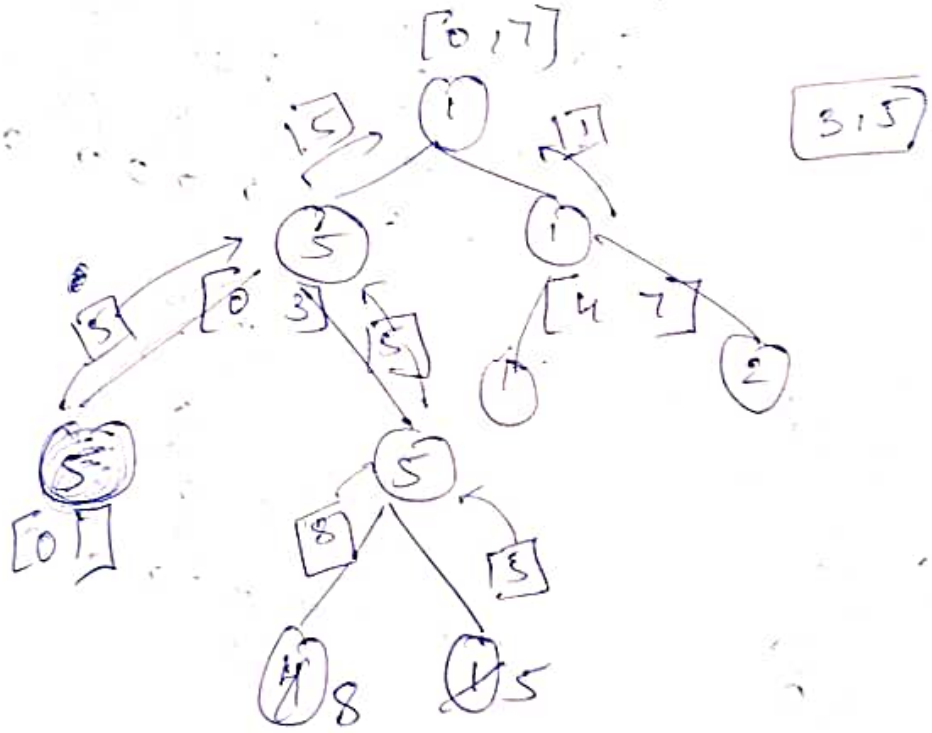








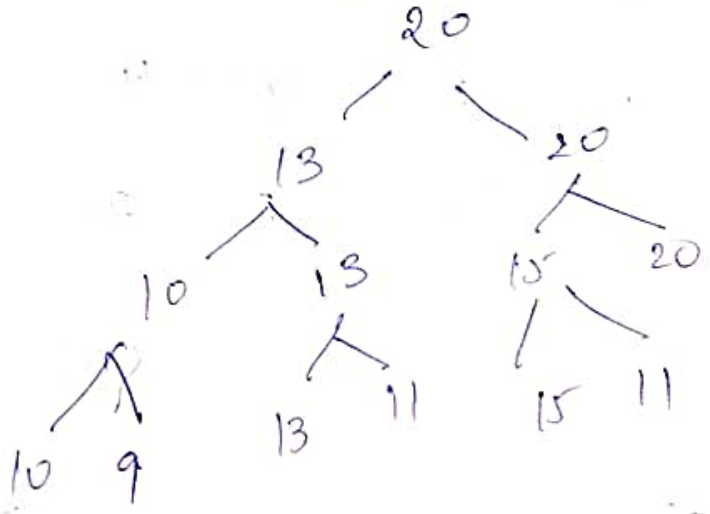




[3,5]

A/W :

Q) Max Segment tree



do following operations,

Update.

1) [0, 3] by 3
2) [0, 3] by 1
3) [0, 0] by 2
4) max [3, 5] => ?

Knapsack problem

Dynamic Programming
Greedy Approach
Branch and Bound
:

Maximisation (profit)

Minimization (weight)

Greedy Approach :-

objects :-

	1	2	3	4	5	6	7
profit:	5	10	15	7	8	9	4
Weight:	1	3	5	4	1	3	2

$W = 15$

$n = 7$

(1) To get the $\max$ profit using profits

(2) To get max profit using weight

(3) To get max profit using $\left(\frac{\text{profit}}{\text{weight}}\right)$ ratio

Solution 1:

profits:

(i)

objects	profit	weight	remaining weight
3	15	5	15 − 5 = 10
2	10	3	10 − 3 = 7
6	9	3	7 − 3 = 4
5	8	1	4 − 1 = 3
4	7 × 3/4 = 21/4 = 5.25	4 × 3/4 = 3	3 − 3 = 0

Total profit = 54.25

47.25

25
17
12.25

29.25
25
54.25

Solution 2:

Select the least weight object.

objects	profit	weight	remaing weight
1	5	1	15 − 1 = 14
5	8	1	14 − 1 = 13
7	4	2	13 − 2 = 11
2	10	3	11 − 3 = 8
6	9	3	8 − 3 = 5
4		4	5 − 4 = 1
3	15 × 3/4 = 11.25	3 × 1/4 = 1	1 − 1 = 0

Solution 8:

$\frac{\text{profit}}{\text{weight}}$

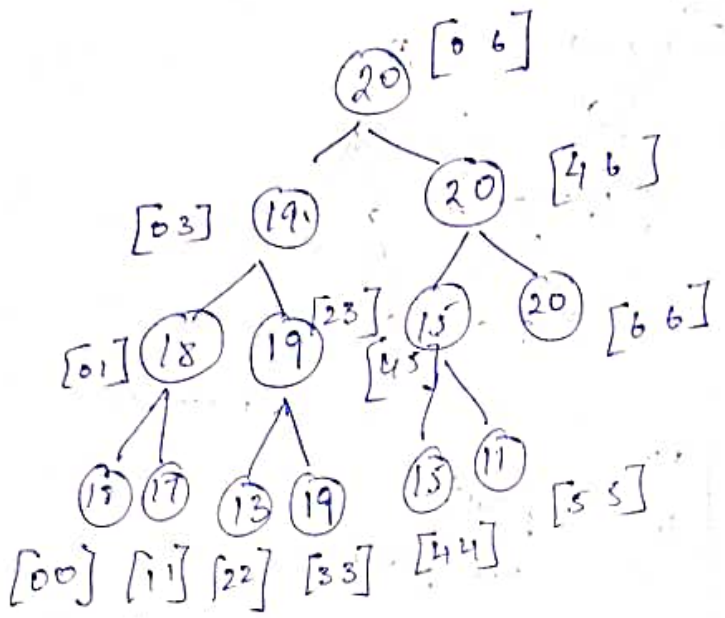
	✓ 1	✓ 2	✓ 3	4	✓ 5	✓ 6	✓ 7
objects :	1	2	3	4	5	6	7
profit :	5	10	15	7	8	9	4
weight :	1	3	5	4	1	3	2
P/W :	5	3.33	3	1.75	8	3	2

Object	profit	weight	rem-weight
5	8	1	15 - 1 = 14
1	5	1	14 - 1 = 13
2	10	3	13 - 3 = 10
3	15	5	10 - 5 = 5
6	9	3	5 - 3 = 2
7	4	2	2 - 2 = 0
	51		

1
13
25
13
51

1/3/25

Range Update → lazy propagation
lazy tree.

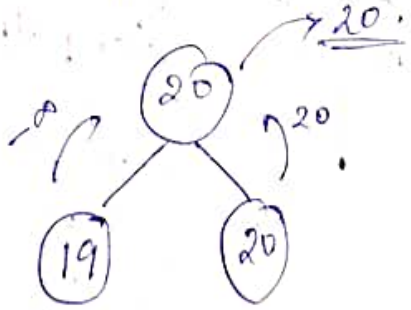


(i) max [4 6]
(ii) Update [0 3] by 3
(iii) Update [0 3] by 2
(iv) Update [0 0] by 1
(v) max [3 5].

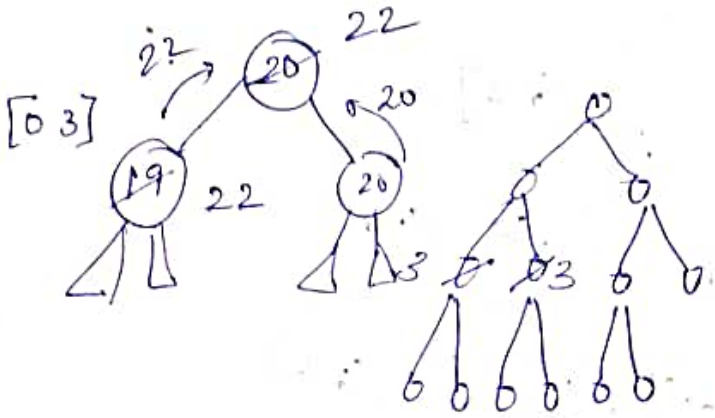
(i) max [4 6]

nl	nr
6	6
0	3
4	6

partial.
No overlapping
Complete



(ii) update [0 3] by 3:



Max (22, 20) = 22

nl	nr	
0	6	partial
0	3	complete
4	6	No

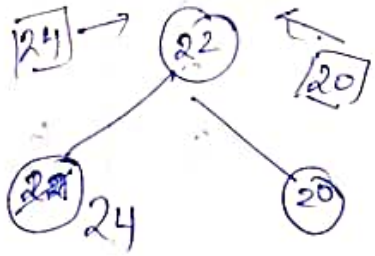
Updation return node value.

actual : [21 20 16 22 15 11 20]

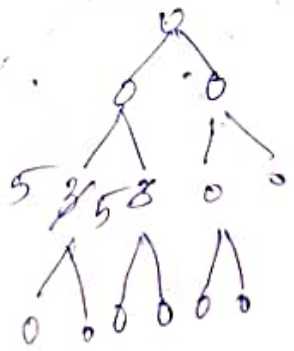
(iii) Update [0 3] by 2.

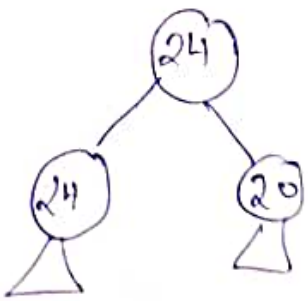
[23, 22, 18, 24, 15, 11, 20]

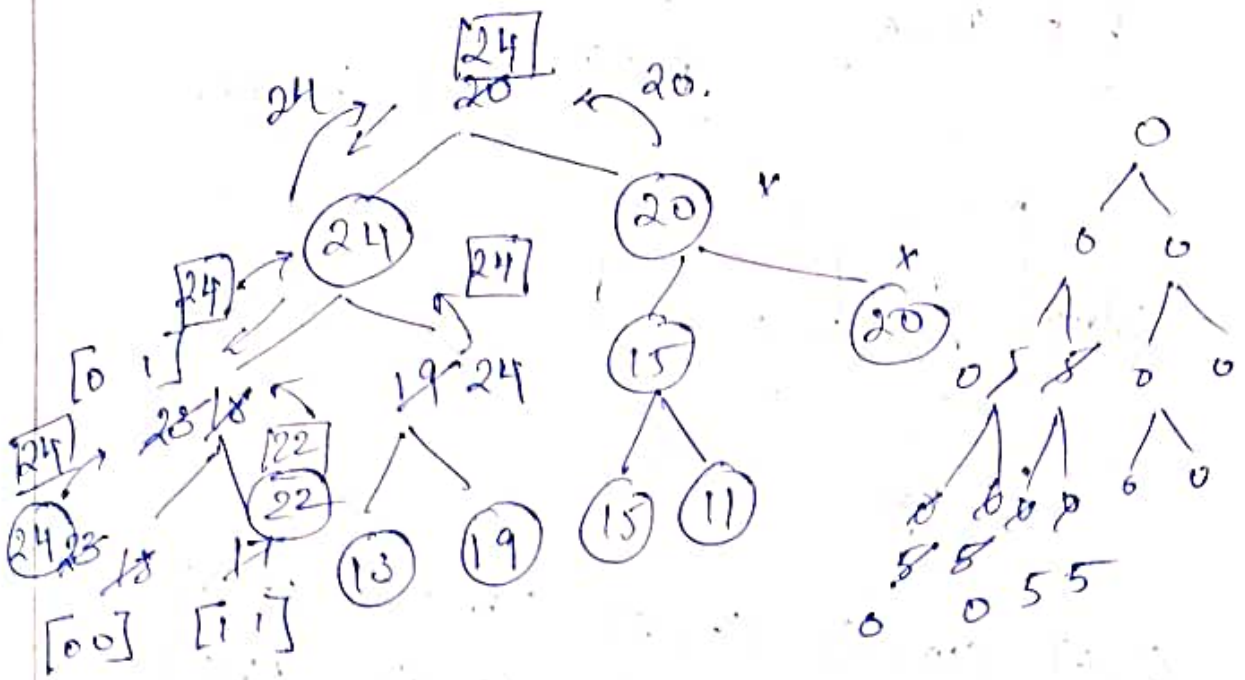
nl	nr	
0	6	partial
0	3	Complete



Max = 24





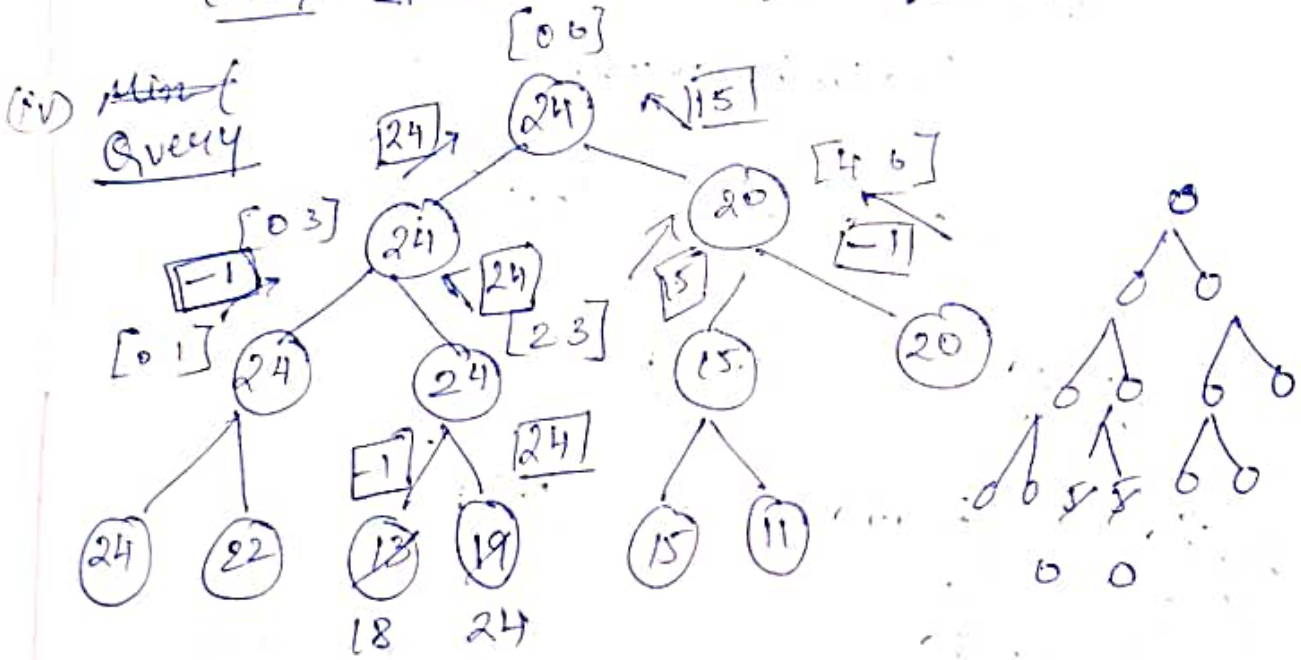


$18 + 5 + 1 = 24$

(Lazy) (updation)

Max [3, 5].

(iv) ~~Min~~ Query



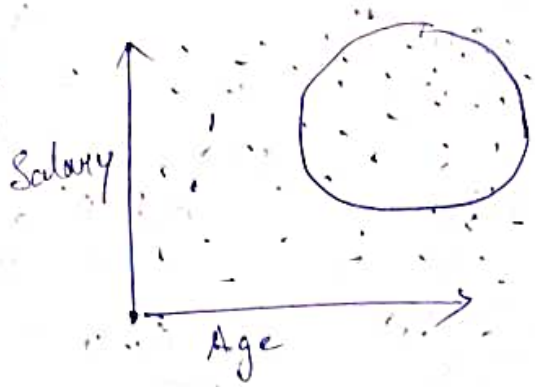
nl	nr	
0	6	partial
0	3	partial
0	1	No.
2	3	partial
2	2	No
3	3	Complete
4	6	partial.
4	5	Complete

Max (24, 15)

= 24

KD Tree

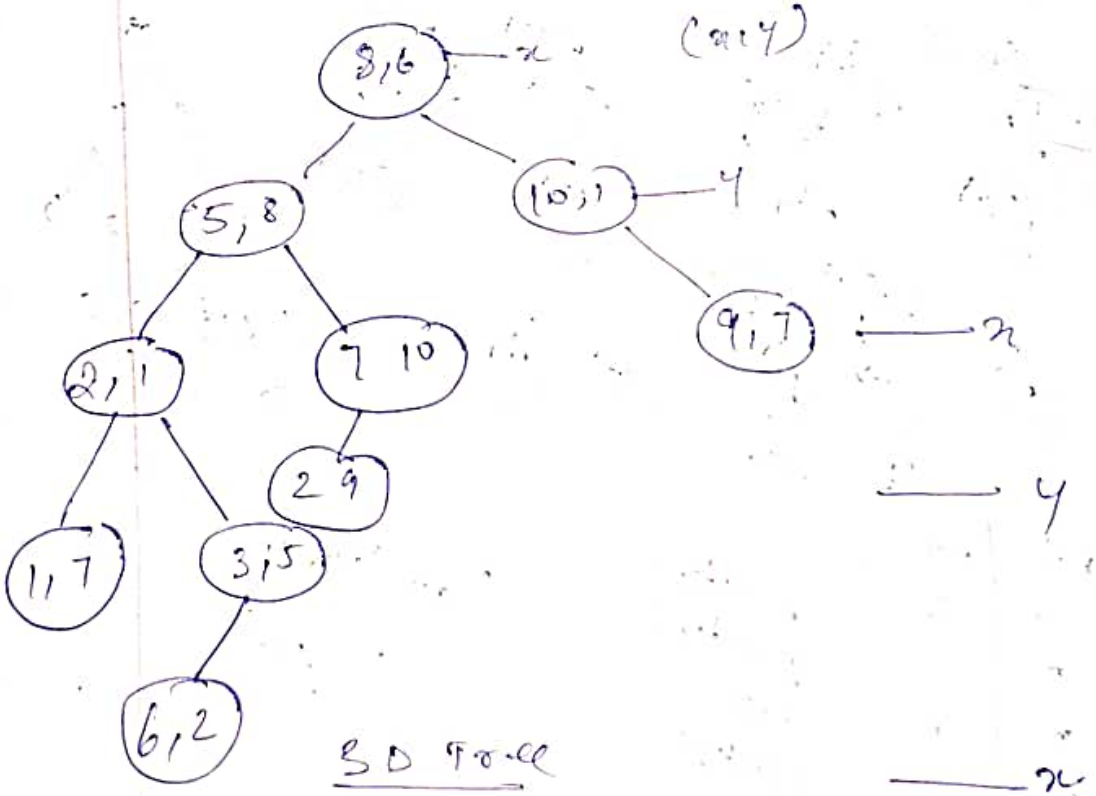
BST — 2 children



(8,6) (10,1) (5,8) (9,7) (2,1)
(3,5) (11,7) (7,10) (2,9) (6,2)

KD Tree

↳ Unbalanced.



SD Tree

x
y
z
x
y
z