* Advantages of doubly link list for deletion of a node over singly link list.

=> - Deletion of a node in a singly link list we have to find the predecessor of the discarded node.
- The search was performed by chaining through the successive node. This search was necessary in order to change the link of the predecessor node to a value that would point to the successor of a node which is to be deleted.
- search is very time consuming depending on the no. of deletion & the no. of nodes in the list.
- In doubly link list no such search is required.
- Simply given the address of a node which is to be deleted & the predecessor & successor nodes are immediately node.
- So doubly link list are much more efficient w.r.t in deletion a singly link list.

* Deletion algorithm in doubly link list :-

procedure : DOUBDEL(L, R, OLD)

OLD = Address of a node which is to be deleted

1. [Underflow ?]

if R == null
then, write ('UNDERFLOW')
Return

PAGE NO:
DATE:

* Advantages of doubly link list for deletion of a node over singly link list.

⇒ - Deletion of a node in a singly link list we have to find the predecessor of the discarded node.
- The search acces performed by chaining through the successive node. This search was necessary in order to change the link of the predecessor node to a value that would point to the successor of a node which is to be deleted.
- search is very time consuming depending on the no. of deletion & the no. of nodes in the list.
- In doubly link list no such search is required.
- Simply given the address of a node which is to be deleted & the predecessor & successor nodes are immediately node.
- So doubly link list are much more efficient w.r.t in a singly link list.

* Deletion algorithm in doubly link list :-

procedure : DOUBDEL (L, R, OLD)

OLD = Address of a node which is to be deleted

1. [Underflow ?]

if R == null
then, write ('UNDERFLOW')
Return

DATE:

2. [Delete node]

if L == R (if single node in link list)

then L ← R ← NULL

else if OLD = L (left most node to be deleted)

then L ← RPTR(L)
LPTR(L) ← NULL

else if OLD = R (Right most node)

then R ← LPTR(R)
RPTR(R) ← NULL

else

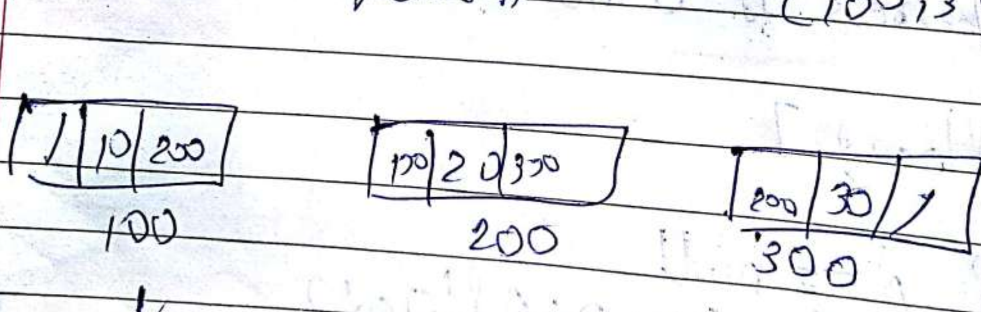
RPTR(LPTR(OLD)) ← RPTR(OLD)
LPTR(RPTR(OLD)) ← LPTR(OLD)

3. Return deleted node

Restore (OLD)

Return

L R OLD → જે delete કરવાનો હોય તેનો address
(100, 300, 100)



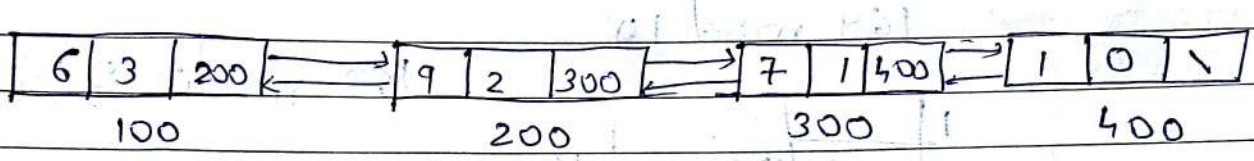
dynamic = memory allocated at run time.

SUPER
PAGE NO:
DATE:

* Application of link list:

(ii) Polynomial representation

d) $6x^3 + 9x^2 + 7x + 1$



link list can be use

(iii) to implement stack, tree, graph, queue etc.

(iii) It is use to table hash table.

(iv) Undo functionality in photoshop & word.

(v) It is useful in computer in which all the running application are kept in cd L.L & the operating system gives a fix time slot to all running process.

(vi) The O.S keeps on iterating over the L.L untill all the application are completed.

Hashing

⇒ Searching complexity O(1) Array O(n)

↳ k mod m

m = 10

⇒ 196, 182, 144, 147, 293, 1395

169 mod 10

1	
2	182
3	
⋮	
9	169
10	

182 mod 10 = 2

182 mode 10 = 2 store in 2

⇒ Suppose I want to search an element in array the time complexity is O(n)

* Comparison of searching example in Array

- It is depend on the total no. of element in that data structure so it is not desirable for large data set to deal with this issue hashing is introduced.
- Hashing is one approach in which time required to search an element doesn't depend on the no. of elem using hashing D.S an element is search with the complex ↳ T.C. (O(1))

Hash table:

Definition: It is a data structure in which keys (element) are map to the hash table position via hash fn.

- # table is a D.S in which insertion & search operation are very fast irrespective of the size of data.

* Assume this keys

196, 182, 144, 147, 293, 1395, 1495

collison → trying to store at one position

fn is = k mod m & m = 10

196 mod 10

19 = 6

10) 196

10

96

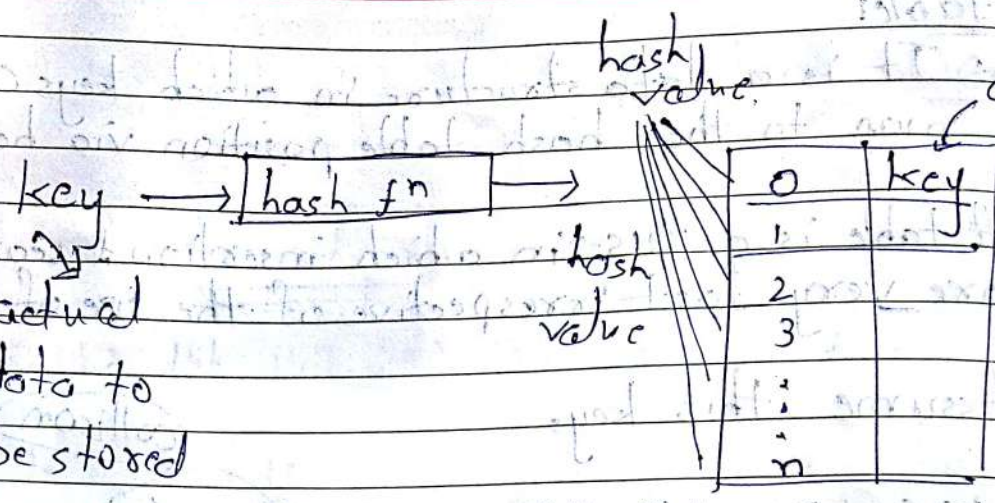
90

6

0	
1	
2	182
3	293
4	144
5	1395
6	196
7	147
8	
9	

Hashing:

The process of mapping the keys to the appropriate location in a hash table is called as hashing.



Ex: Suppose we have a set of string {"abc", "def", "ghi"}

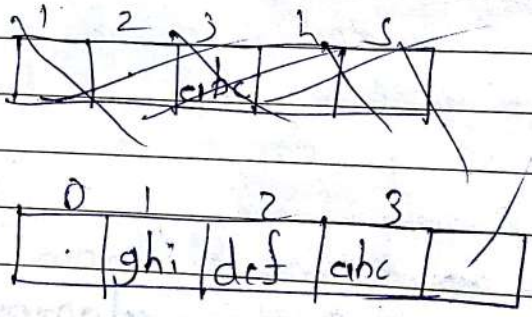
M = 5

k mod 5

(0+1+2) mod 5

3 mod 5

= 3



a	0
b	1
c	2
d	3
e	4
f	5
g	6
h	7
i	8

SUPER
PAGE NO:
DATE:

Hash function :-

A hash fn is simply a mathematical formula where when it is applied to the key, it produce an integer which can be use an an index for the key in the # table.

- The main aim of a # fn is the element should be relatively randomly & uniformly distributed.
- To achive a good hashing mechanisam it is important to have good # fn with following basic requirement

(i) It should be easy to ~~comm~~ compute.
(ii) Determinism
(iii) uniformity

(i) Easy to compute: It means the cost of computing the # fn must be small.

Ex: If sequential search algo. can search an element in O(n) times then the cost of # fn must be less than order of n (O(n)).

(ii) Determinism: It means the same # value must be generated for a given i/p value.

(iii) Uniformity: It should provide a uniform distribution across the # table.

SUPER
PAGE NO:
DATE:

* Different # fn :-

There are 4 basic # fn

(i) Division method
(ii) multiplication "
(iii) folding "
(iv) mid square "

(i) Division method :

$h(z) = z \bmod m$ — size of hash table
location actual value to be stored

Suppose m is even then if z is even then h(z) is even
= m is even, z is odd then h(z) is odd

- If even keys are more than odd key then devision method will not spread the hash value uniformly. So it is best to choose m to be a prime number.

Calcute the # value of keys 1, 2, 3, 4 & 5, 6, 4, 2.
1234 & 5672 5642

m = 97

97) 1234

16	70
5642 ...	1234

squared $(5)^2 - 1 / 2$

SUPER
PAGE NO:
DATE:

(iii) multiplication ~~model~~ : method :

step-1 choose a constant a such that a is b/w 0 to 1
2. multiplied the key k by a 3] Extract the fractional part of ka

4 multiple the result of step-3 by m & take the floor

$$h(x) = \lfloor m (kA \bmod 1) \rfloor$$

⇒ kA mod 1 gives the fractional part of KA.
⇒ m = total no. of indices in # table.

$A = 0.6180339887 = \sqrt{5} - 1 / 2$

* Given a hash table of size 1000 map the key 12345 to an appropiate location in the # table.

$h(12345) = \lfloor 1000 ((12345)(0.6180339887) \bmod 1) \rfloor$

$= \lfloor 1000 * 0.62959 \rfloor$
$= \lfloor 629.59 \rfloor$

$h(12345) = 629$ floor

PAGE NO:
DATE: 6/3

* mid square method:

step 1 square the value of key that is find k^2.

step 2 Extract the middle r bits of the result obtain in step-1.

step 3 In mid square method same r bit must be choosen from all the keys therefore the hash fn is given as a $h(k) = s$

where s is obtained by selecting r bits form k

Calculate # value for the key 1234 & 5642 using the mid square method. # table has 100 memory location

$r = a - (99)$ $= 2$

1234 = 1522756
5642 = 31832164

h(1234) = 1522756 = 56
h(5642) = 31832164 = 64

* folding method:

1 Divide the key value into no. of parts that is devide k into parts $k_1, k_2, \ldots k_n$

where each part has the same no. of digit except the last part which may have lesser digit than the other part.

2. At the individual part that is obtain the sum of $k_1 + k_2 + \dots k_n$ the # value is produce by ignoring the last carry if any.

Note: Note that the no. of digit in each part of the key will depending on the size of the # table.

Given a hash table of 100 location calculate a # value using folding method for keys 5678, 321 & 34567

1st	key	5678	321	34567
depend on location	parts	56 & 78	32 & 1	34, 56 & 7
2nd	sum	134	33	97
	Hash value	34 (carry generate)	33	97

Collision:

- Collision occur when the # fn next to diff key to the same location. This process is called as collision
- Therefore a method used to solve the problem of collision is called as collision resolution technique

There are two technique

(i) Open addressing
(ii) chaining

(i) There are four technique
- linear probing
- quadratic "
- double hashing
- Rehashing

* Collision resolution by open addressing:

1 Once a collision take place open addressing or closed hashing computes new position using prob sequence and next record is stored in that position.

2 # table contain 2 types of value
(i) Central Value
(ii) Data "

3 The present of central value indicate that the location contain no data value and it can be used to hold any value (key).

- When a key is mapped to the particular memory location then the value it holds its checked. If it contain continal value then the location is free. However if the location has already has some data value then other slot are examine systemetically in the forward direction to find free slot.
- The process of examine a memory location in the # table is called as probing.

a) Linear probing:

$$h(k,i) = [h'(k) + i] \bmod m$$

$h'(k) = k \bmod m$ → size of hash table

↳ actual value

$i = 0$ to $m-1$

↓

(prob sequence)

* Consider a # table of size 10 using linear probing insert the key 72, 27, 36, 24, 63, 81, 92, 101 into the table.

⇒

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

i) $h(72,0) = [(72 \bmod 10) + 0] \bmod 10$

$= 2 \bmod 10$

$= 2$

0	1	2	3	4	5	6	7	8	9
-1	-1	72	-1	-1	-1	-1	-1	-1	-1

iii) h(27,7) =

h(27,0) = [(27 mod 10) + 0] mod 10

= 7 mod 10

= 7

Values of i always 0, when collision occure it increase by 1.

0	1	2	3	4	5	6	7	8	9
-1	-1	72	-1	-1	-1	-1	-1	27	-1

0	1	2	3	4	5	6	7	8	9
-1	-1	72	-1	-1	-1	-1	27	-1	-1

iii) h(36,0) = [36 mod 10] + 0] mod 10

0	1	2	3	4	5	6	7	8	9
-1	81	72	63	24	92	36	27	101	-1

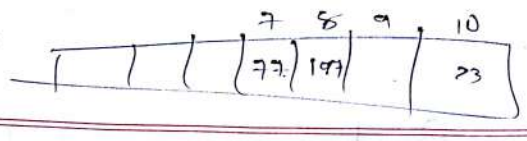
iii) Problem with linear probing:

- It is primary clustering (group) problem
- Suppose 7, 77, 177 & the # table size 10

0	1	2	3	4	5	6	7	8	9
							7	77	177

primary clustering problem.

- Here one of the portion of # table is field with the keys and other portion remain unuse. So this is called as primary clustering problem.
- To avoid p.c other technique such as quadratic probing & double hashing are use.



1937 SUPER
PAGE NO:
DATE:

Searching a value using linear probing :-

⇒ Searching in linear probing is same as storing.

⇒ While searching for a value in a # table index is recomputed and the key of the element stored at that location is compared with the value that has to be searched

⇒ if match is found then search operation is successful then the T.C. O(1)

⇒ If the key doesn't match then the search function begin a sequential search until the first one the value is found. ~~The second~~

(i) the search fn encounter a vacent location in the array or hash table

(ii) The search fn terminate because it reach to the end of the table.

(hash value index) - Worst case of the search operation have n-1 comparision. So the run time of ~~algorithm~~ algorithm is O(n)

Worst case encounter when after scanning all the n-1 element the value is either present at the last location or not present at all

- We can see that with the increasing no. of collision the distance b/w array index computed by the # fn & the actual location of element increases. thereby increasing search time

(ii) Quadratic Probing :-

$h(k,i) = [h'(k) + c_1 i + c_2 i^2] \bmod m$

$h'(k) = k \bmod m$

c_1 & c_2 = constant such that c_1 & $c_2 \neq 0$
& c_1 & c_2 is decided by expert

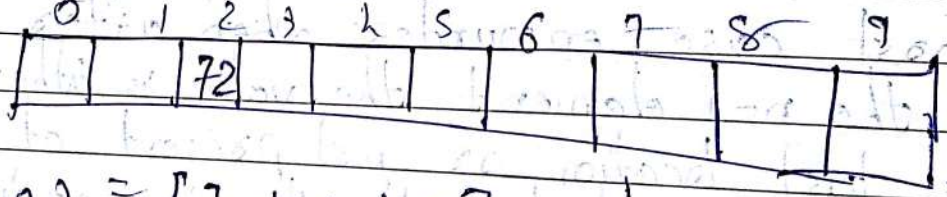
⇒ Quadratic probing eliminates the primary clustering problem of linear probing.

Consider a # table of size 10 using quadratic probing insert a keys 72, 27, 36, 24, 63, 81, 101, 92, $c_1 = 1$ & $c_2 = 3$

⇒ $h(72,0) = [2 + (1)(0) + (3)(0)] \bmod 10$

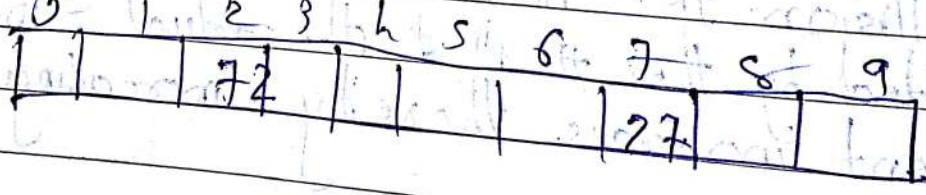
$i = 0 + 0 = 0$

$= 2$



$h(27,0) = [7 + 0 + 0] \bmod 10$

$= 7$



$h(36,0) = [6+0+0] \bmod 10$

$= 6$

0	1	2	3	4	5	6	7	8	9
		72				36	27		

$h(24,0) = [4] \bmod 10$

$= 4$

0	1	2	3	4	5	6	7	8	9
		72	24		36	27			

$h(63,0) = 3$

$=$

$h(81,0) = 1$

0	1	2	3	4	5	6	7	8	9
	81	72	63	24		36	27		

$h(101,1) = [1 + 1(1) + 3(1)] \bmod 10$

$= 5 \bmod 10$

$= 5$

0	1	2	3	4	5	6	7	8	9
	81	72	63	24	101	36	27		

$h(92,1) = [2+5] \bmod 10$

$= 6 \bmod 10$

$= 6$

$h(92,2) = [2+2+12] \bmod 10$

$= 16 \bmod 10$

$= 6 \bmod 10$

$h(92,3) = [2+3+9\times3] \bmod 10$

$= 2$ (16₃)

$h(92,4) = [2+4+48] \bmod 10$

$= 4$

$h(92,5) = [2+5+75]$

$= 82 \bmod 10$

$= 2$

$h(92,6) = [2+6+105] \bmod 10$

$= 116 \bmod 10$

$= 6$

$h(92,7) = [2+7+117] \bmod 10$

$= 126 \bmod 10$

$h(92,8) = [2+8+192] \bmod 10$

$= 202 \bmod 10$

$h(92,9) = [2+9+243] \bmod 10$

$= 4$

$h(92,10) = [2+10+300] \bmod 10$

$= 2$

$h(92,11) = [2+11+363] \bmod 10$

$= 6$

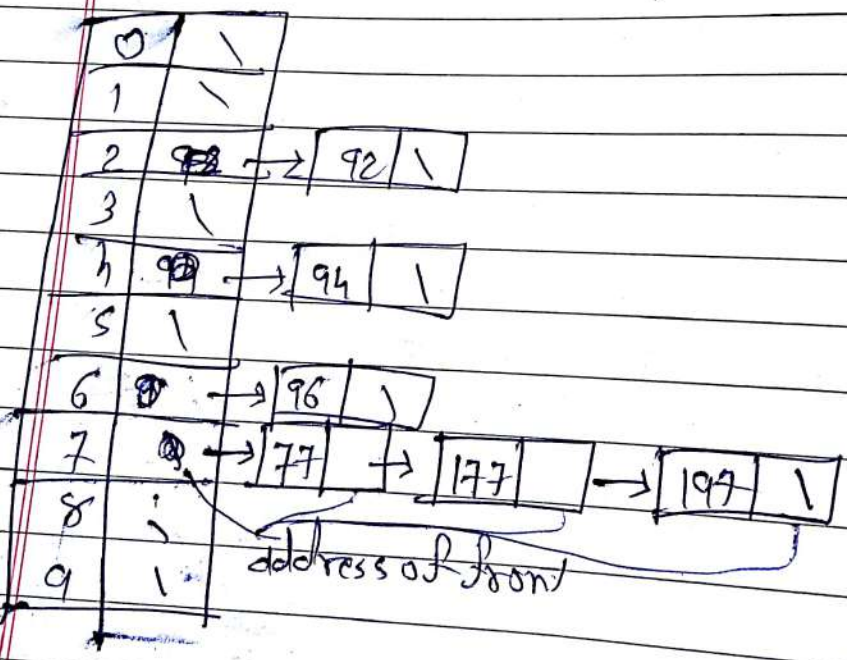
SUPER
PAGE NO:
DATE:

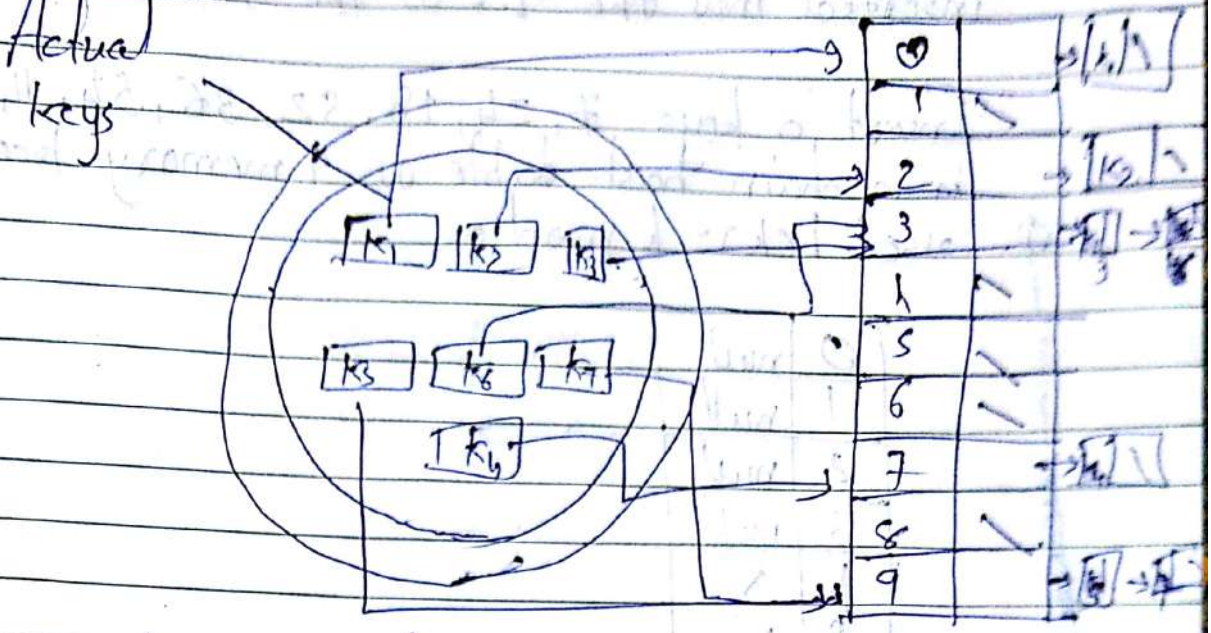
* Collision resolution by chainning:

⇒ In chainning each location in a hash table stores the pointer to a linklist, contains all the key values that work # to the particular location. that is location one in the # table points to the head of the LL of all the keys value that has to one. If no key value has to one then location one in the # table contains null.

77, 177, 197, 92, 94, 96 Initially null

77 mod 10
7





Operation on chain # table:

⇒ Searching for a value in a chain # table is as sample as scanning a linklist insertion operation appends the key to the ~~end~~ End or front of the L.L pointed by the # location.

⇒ Deleting a key required searching the list and removing the element. chain hash table with L.L are widly used due to the simplicity of the algorithm to insert, delete and search a key.

- The code for this algorithm is exactly same as for inserting, deleting and searching a value in a singly L.L

⇒ Cost of insertion is O(1) and cost of deletion is O(n) searching n is a no. of element in the list of the location.

⇒ In the worst case searching require O(n) n = no. of key store in the chain/hash table.

(This case arise when all the key values are inserted into the L.L of the same location.

Insert 8 keys 7, 24, 18, 52, 36, 54, 11, 23, 60 in a chain hash table of 9 memory location

else. h(k) = k mod m

0	0	null
1	1	null
2	2	null
3	3	null
4	4	\
5	5	\
6	6	\
7	7	\ → 7 \ \
8	8	\

a) Insert 7 — 7 mod 9 = 7

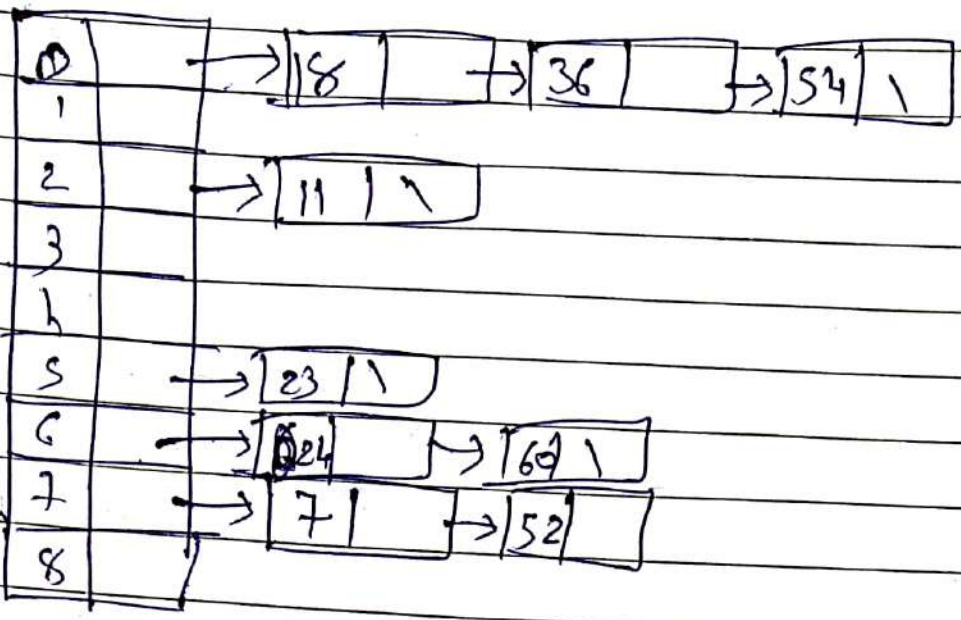
0	0	
1	1	
2	2	
3	3	
4	4	
5	5	
6	6	
7	7	→ 7
8	8	

(b) 24 — 24 mod 9 = 6 — 54 = 54 mod 9

18 — 18 mod 9 = 0 — 11 ⇒ 11 mod 9

52 — 52 mod 9 = 7 — 23 ⇒ 23 mod 9

36 — 36 mod 9 = 0 — 60



SUPER
PAGE NO:
DATE:

SUPER
PAGE NO:
DATE:

(-1,0,1) difference

AVL Tree :- (Height balanced tree)

⇒ AVL tree is a self bt balancing binary search tree invented by G.M Adleson, Velsky, E.M. Landis in 1962

- In an AVL tree the height of two subtree of a node may be differ by atmost one. Due to this AVL tree is also known as height balanced tree.

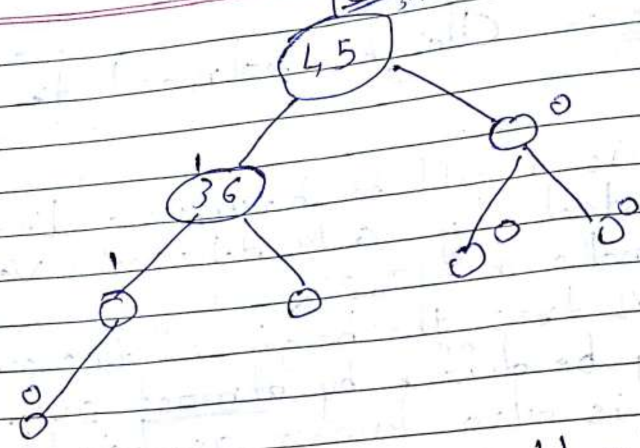
- The key advantage of using AVL tree is that it takes O(logn) time to perform insert, or delete and search operation in an average case as well as in worst case. Hight of the tree is limited to O(logn).

- In this tree every node as a balanced factor associated with it

Balanced factor = Height (left subtree) - Height (Right subtree)

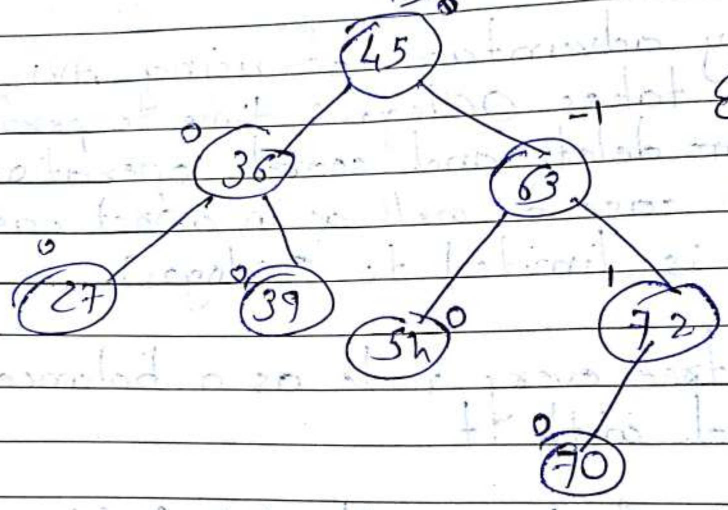
- A binary search tree in which ~~no~~ every node has a balanced factor of -1, 0, 1 is said to be AVL tree or height ~~bem~~ balanced tree.

2 = left subtree hight more



-1 → Right subtree hight more

1.



Example of Right heavy tree

⇒ A node with any other balance factor is considered as unbalanced & required rebalancing of a tree

1 left heavy tree :- If the balance factor of a node is 1 Then it means the left subtree of the tree is one level higher than that of the right sub tree. Such tree called as a left heavy tree.

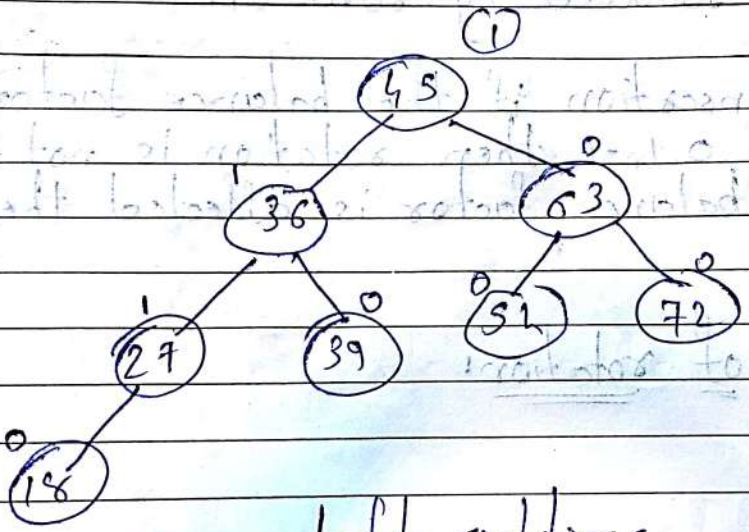
2 Right heavy tree : If the balance factor of a

node is −1 then it means that left subtree of the tree is one ~~label~~ level lower than that of a right subtree.
Such tree is known as Right heavy tree

3. Balanced tree :- If the balance factor of a node is zero than it means the height of the left subtree is equal to the height of the right " .

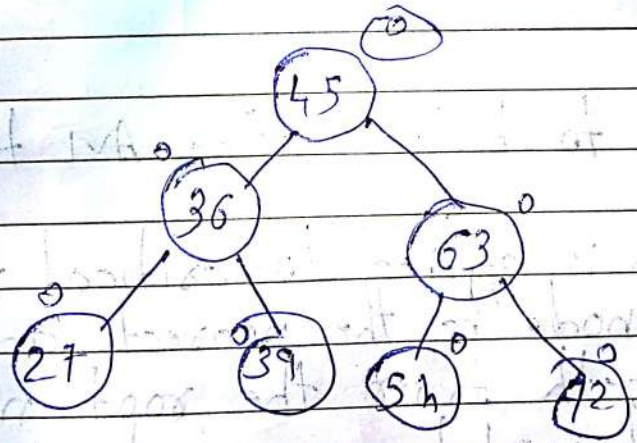
Ex:

1.



left subtree

2.



Balanced tree

Operation on AVL tree:

(i) Searching :-

⇒ Searching operation is perform same as the in binary search tree.

- Complexity :- O(logn) (every cases) B.S.T searching O(n)

(ii) Insertion :-

- Insertion operation is done in the same ways as BST followed by rotation.

- After insertion if the balance factor of every node is 0, 1, -1 then rotation is not required but if the balance factor is affected the rotation is required.

4 - types of rotation:

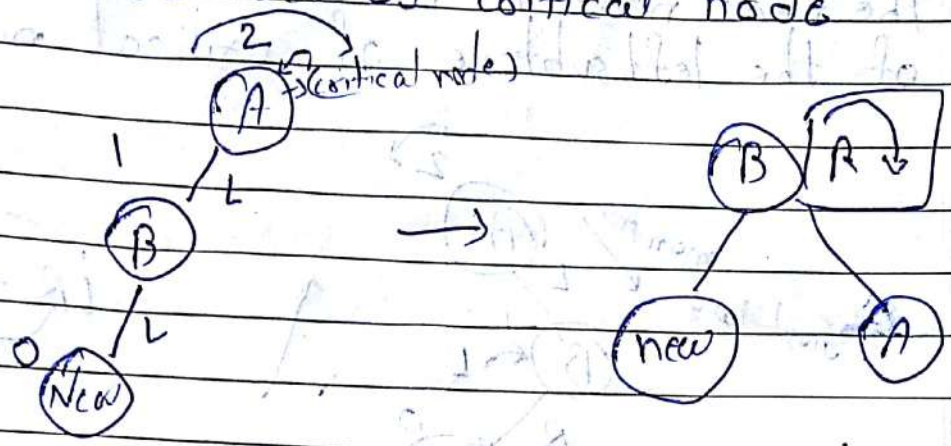
(i) LL
(ii) RR
(iii) LR
(iv) RL

are used to balance the AVL tree.

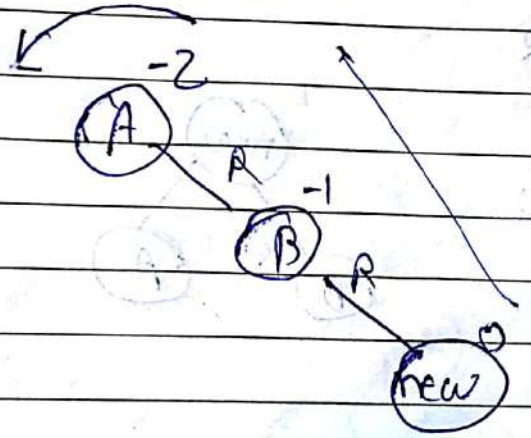
- Rotation is applied to critical node.
- Critical node is the nearest ancestor (de) on the path from the root node to inserted node whose balancing factor is neither -1, 0 or 1.

(i) LL rotation :-

- The new node is insert in the left subtree of the left subtree of critical node

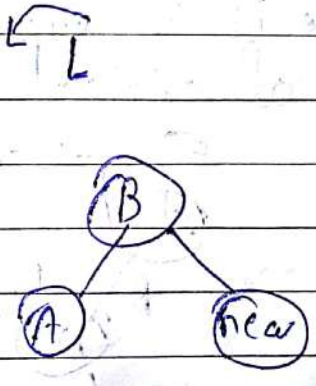


(ii) R-R Rotation :



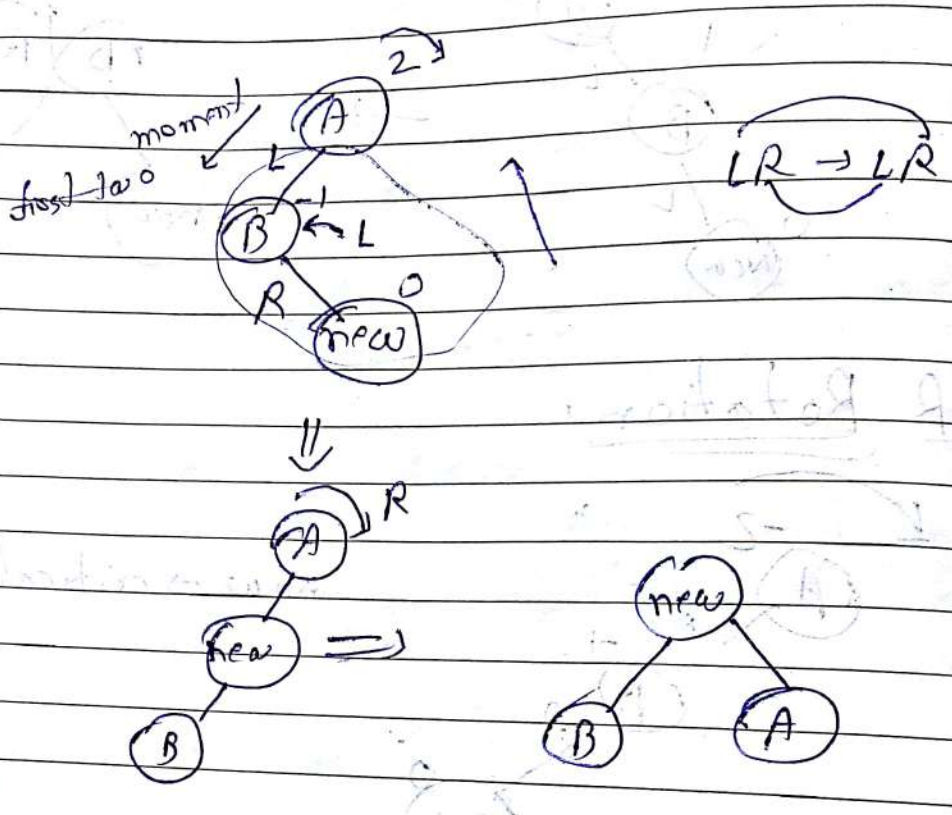
new → critical node ralag

The new node is insert in the right subtree of the right subtree of critical node



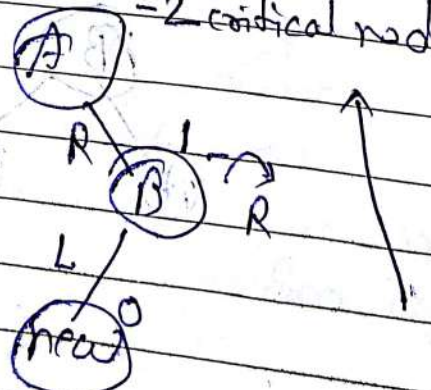
3. L-R rotation:

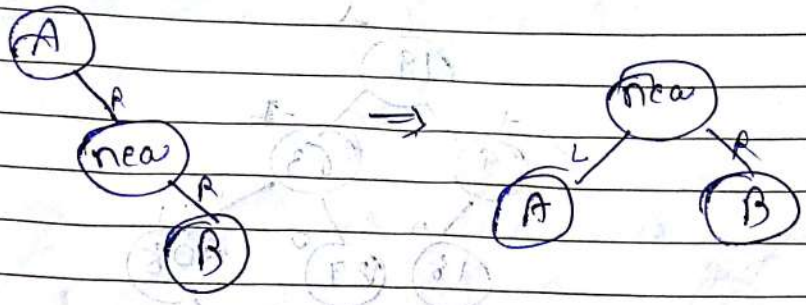
The new node is inserted in the right subtree of the left subtree of critical node.



(iv) R-L rotation:

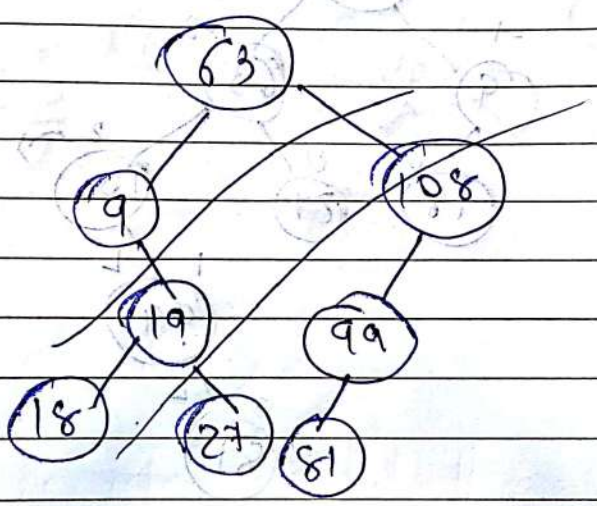
The new node is inserted in the left subtree of the right subtree of critical node.

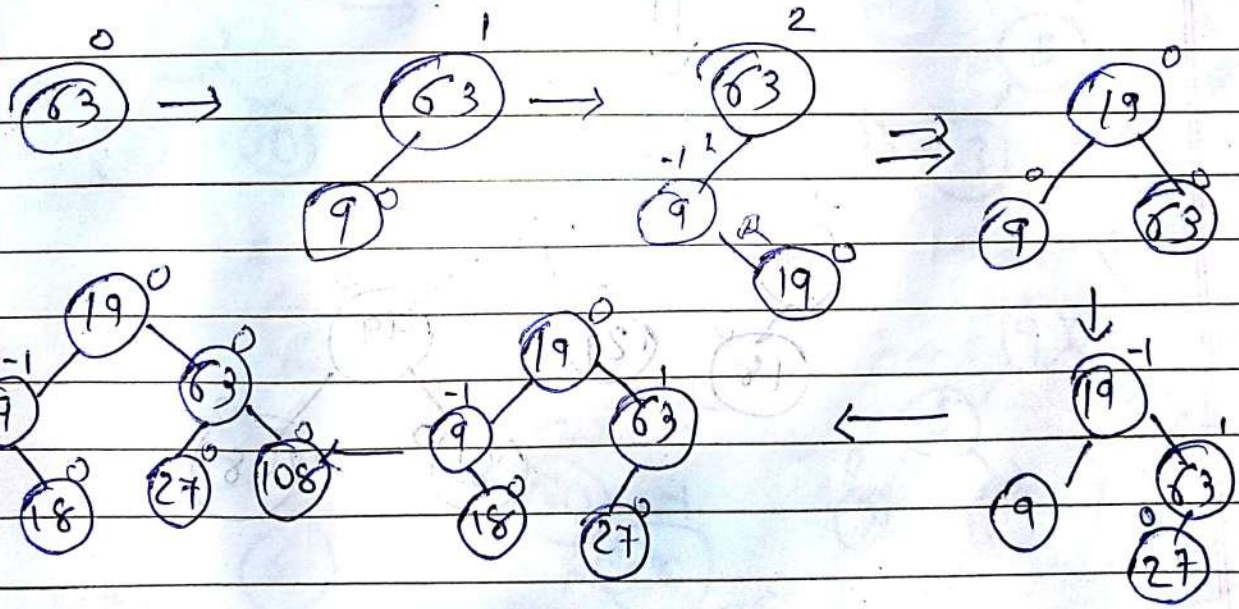


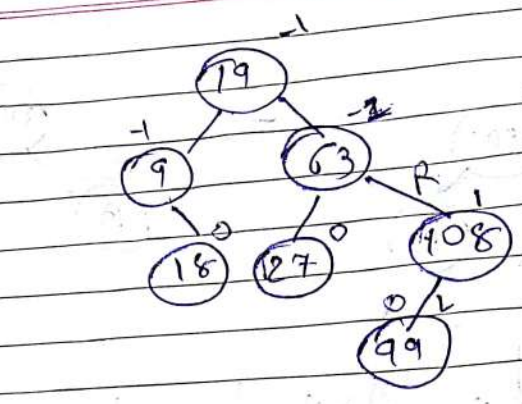


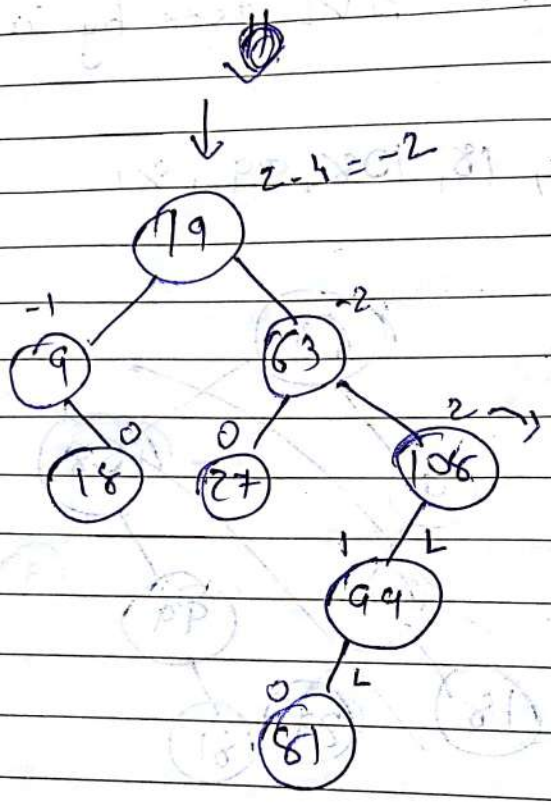
* Construct the AVL tree by inserting following element.

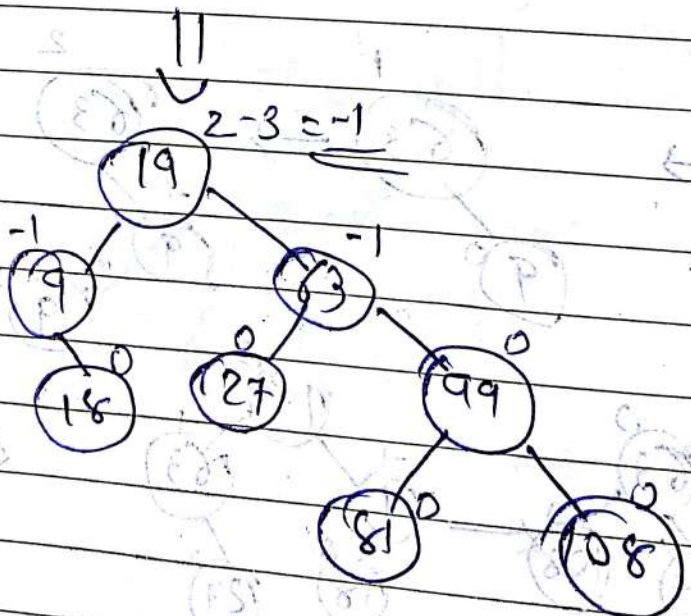
63, 9, 19, 27, 18, 108, 99, 81



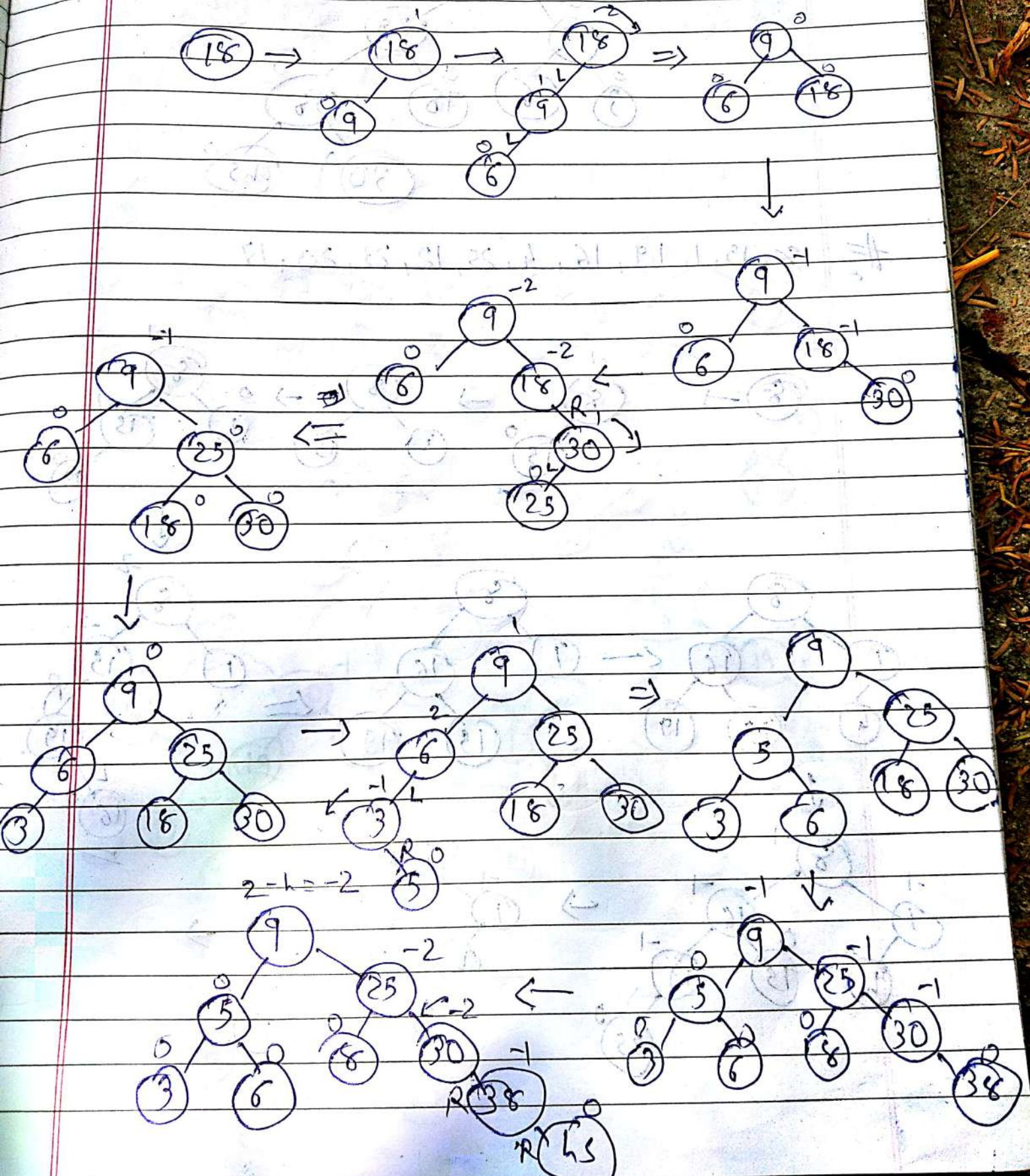


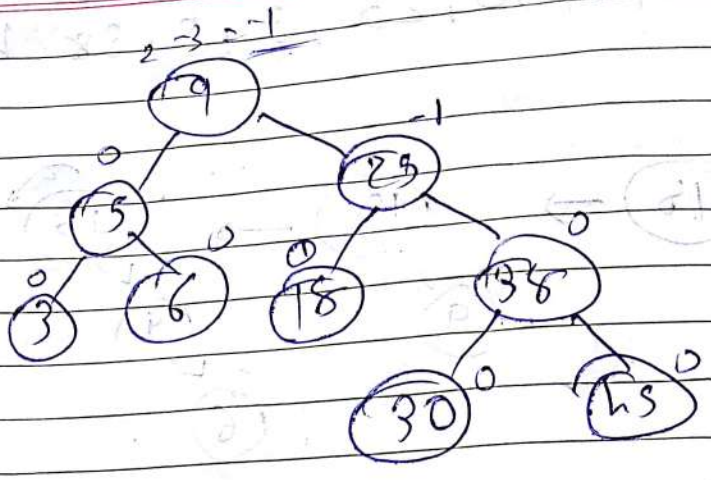




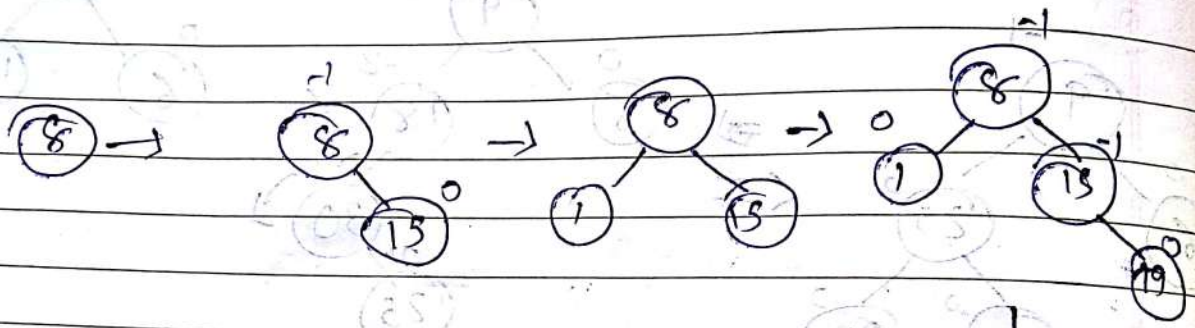


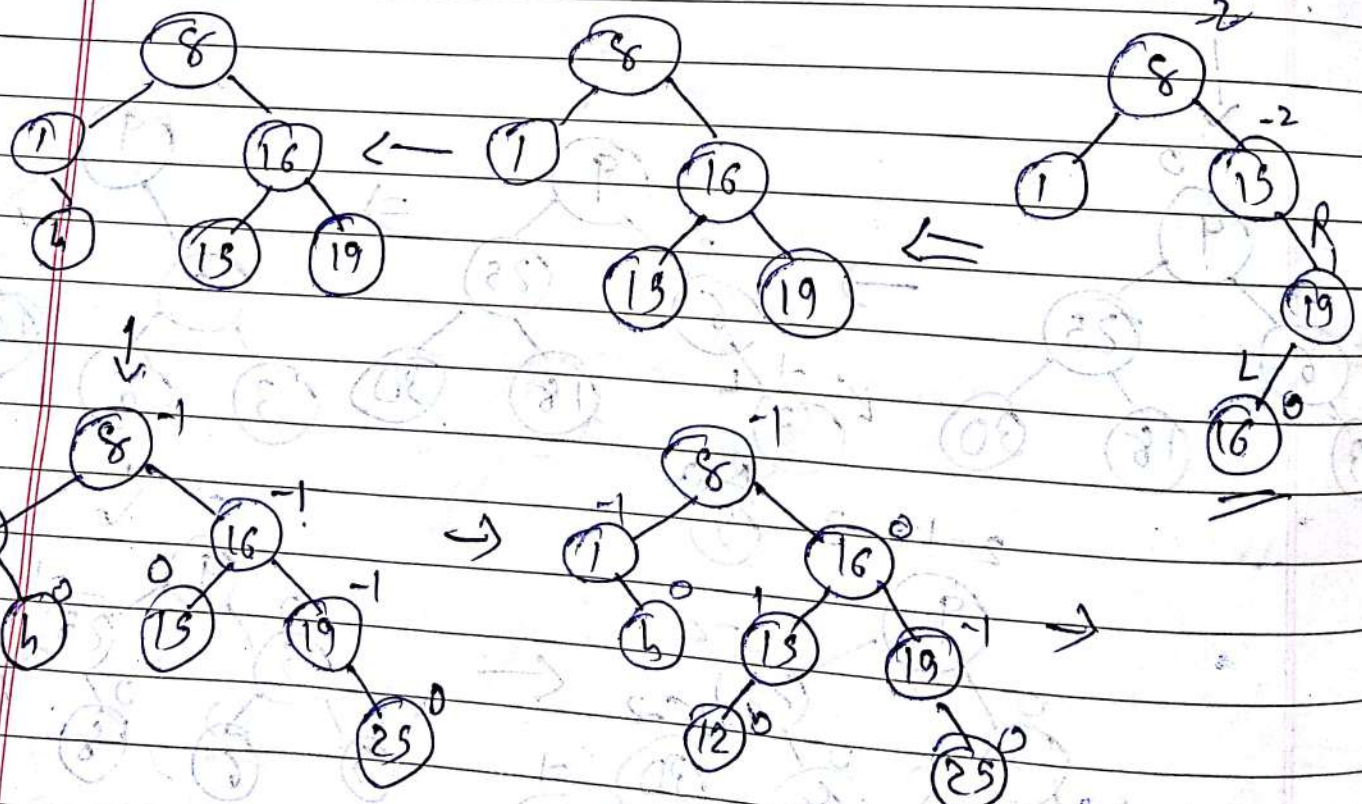
2. 18, 9, 6, 30, 25, 3, 5, 38, 45

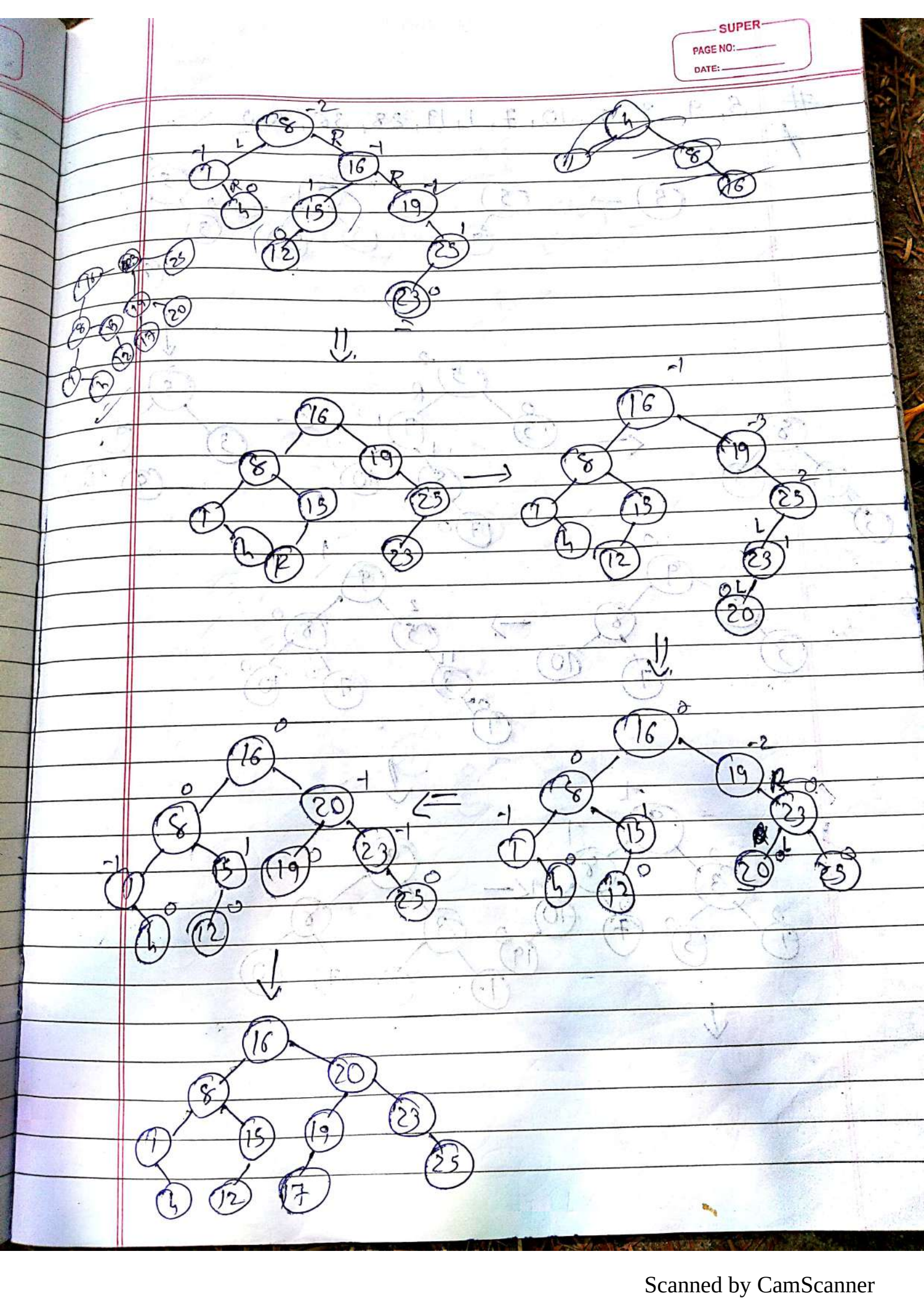




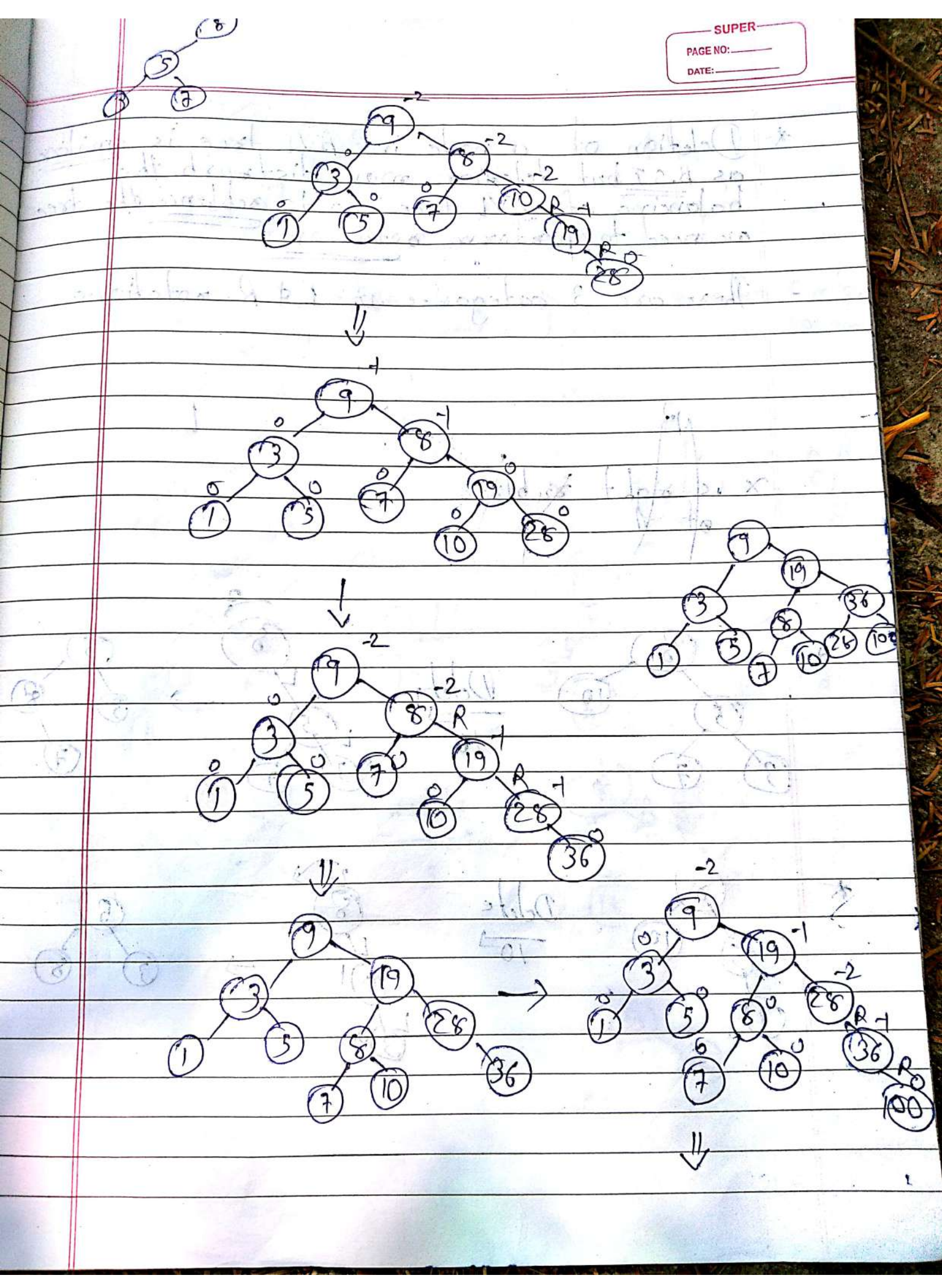
8, 15, 1, 19, 16, 4, 25, 12, 23, 20, 17





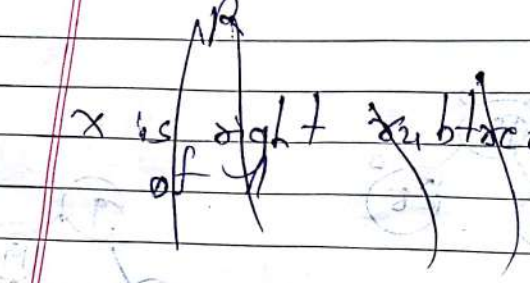


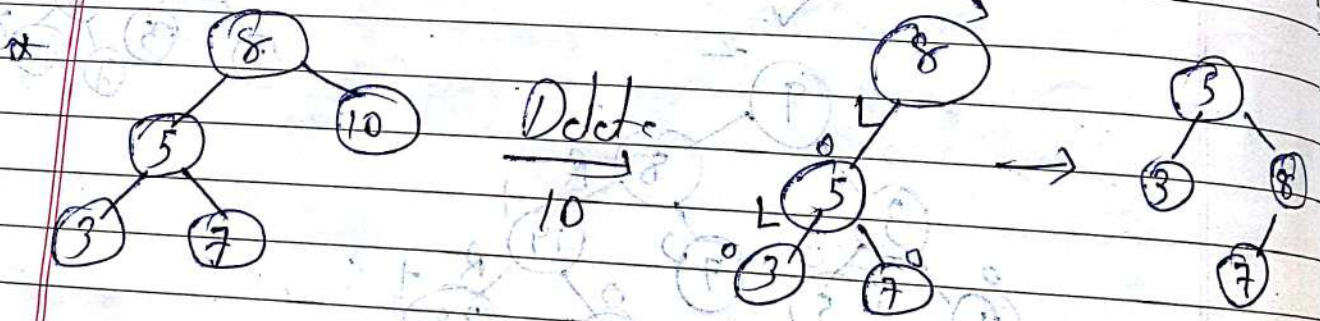
5, 9, 3, 8, 10, 7, 1, 19, 28, 36, 100

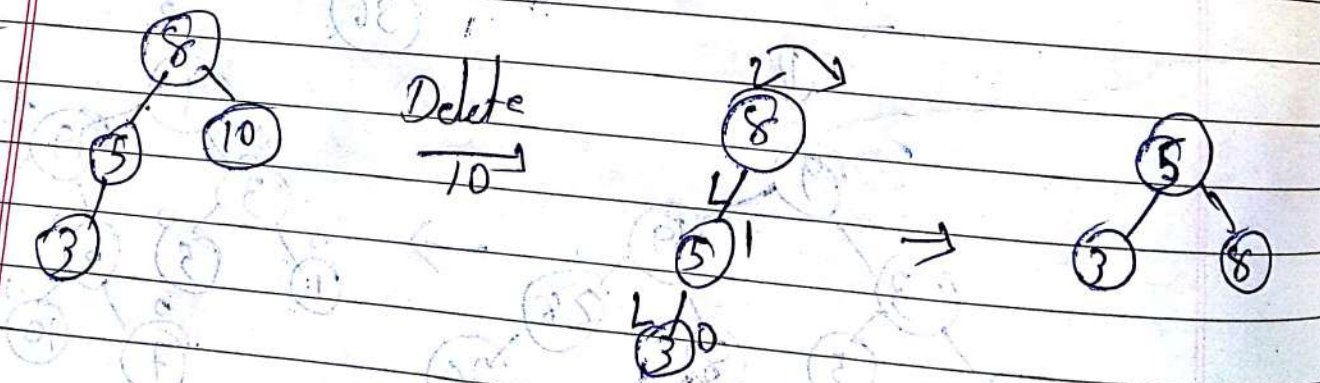


* Deletion of a node in AVL tree is similar as B.S.T but deletion may disturb the balancing of AVL tree so to reblance the tree we need to perform rotation

- There are 3 categories of L & R rotations





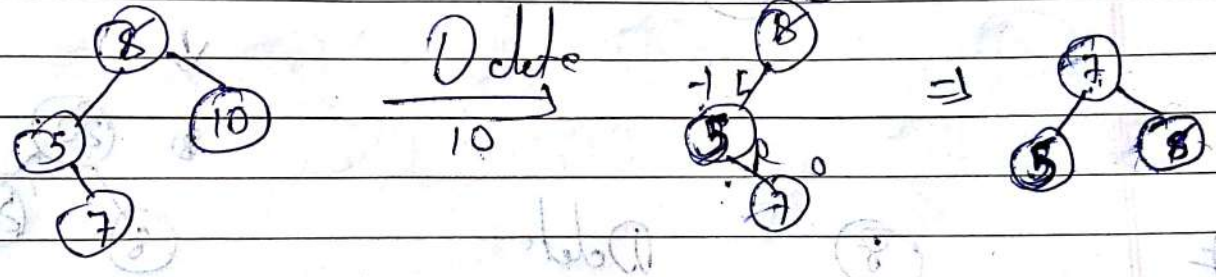


R L

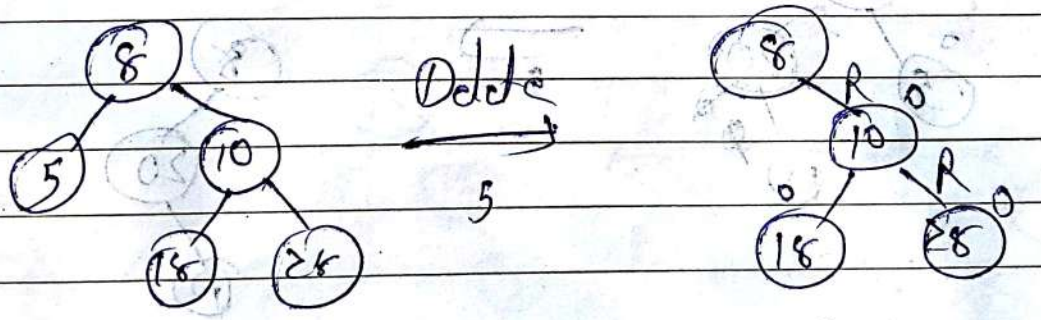
x is right subtree of A — x is in the left subtree of A

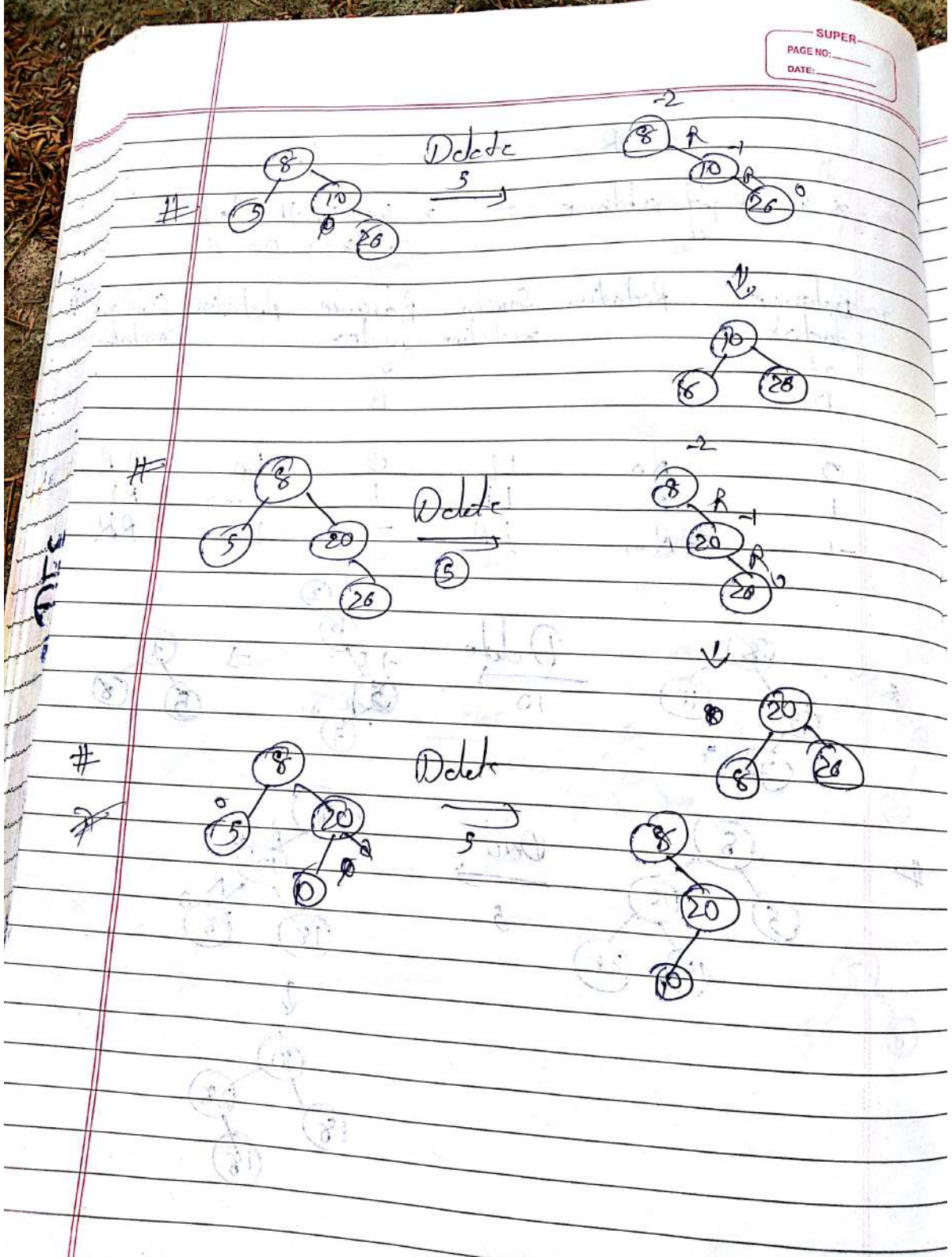
Balance factor of B	Rotation	Similar rotation	Balance factor of B	Rotation	Similar rotation
0	R_0	LL	0	L_0	RR
1	R_1	LL	1	L_1	RL
-1	R_{-1}	LR	-1	L_{-1}	RR

#



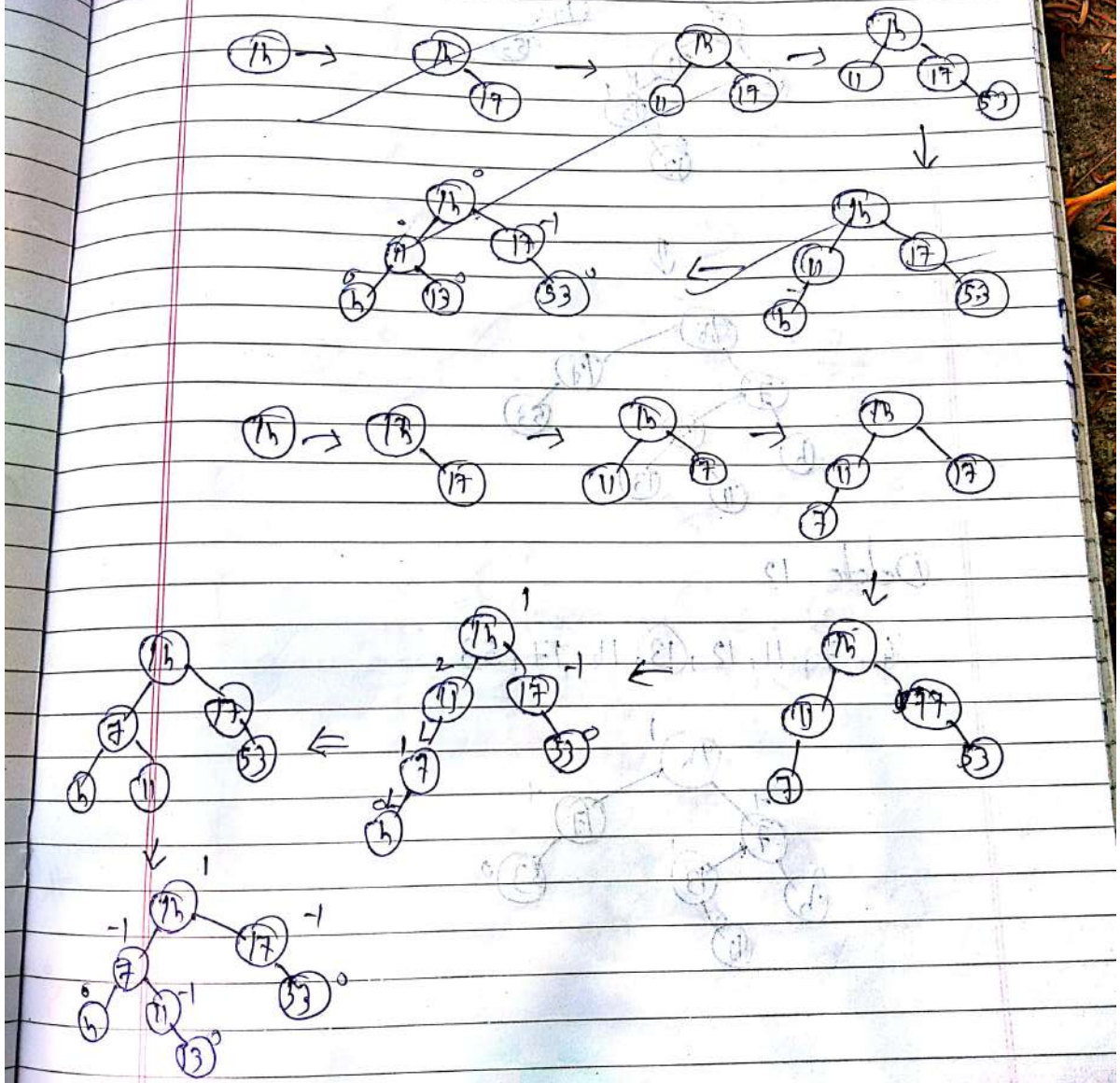
#



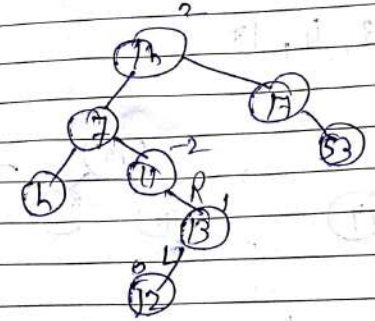


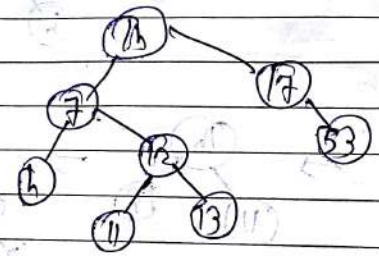
AVL tree

14, 17, 11, 7, 53, 4, 13



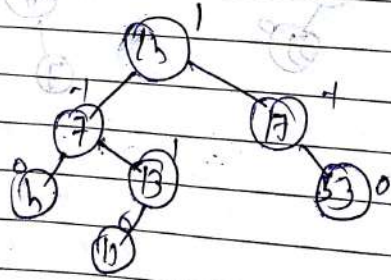
Insert 12

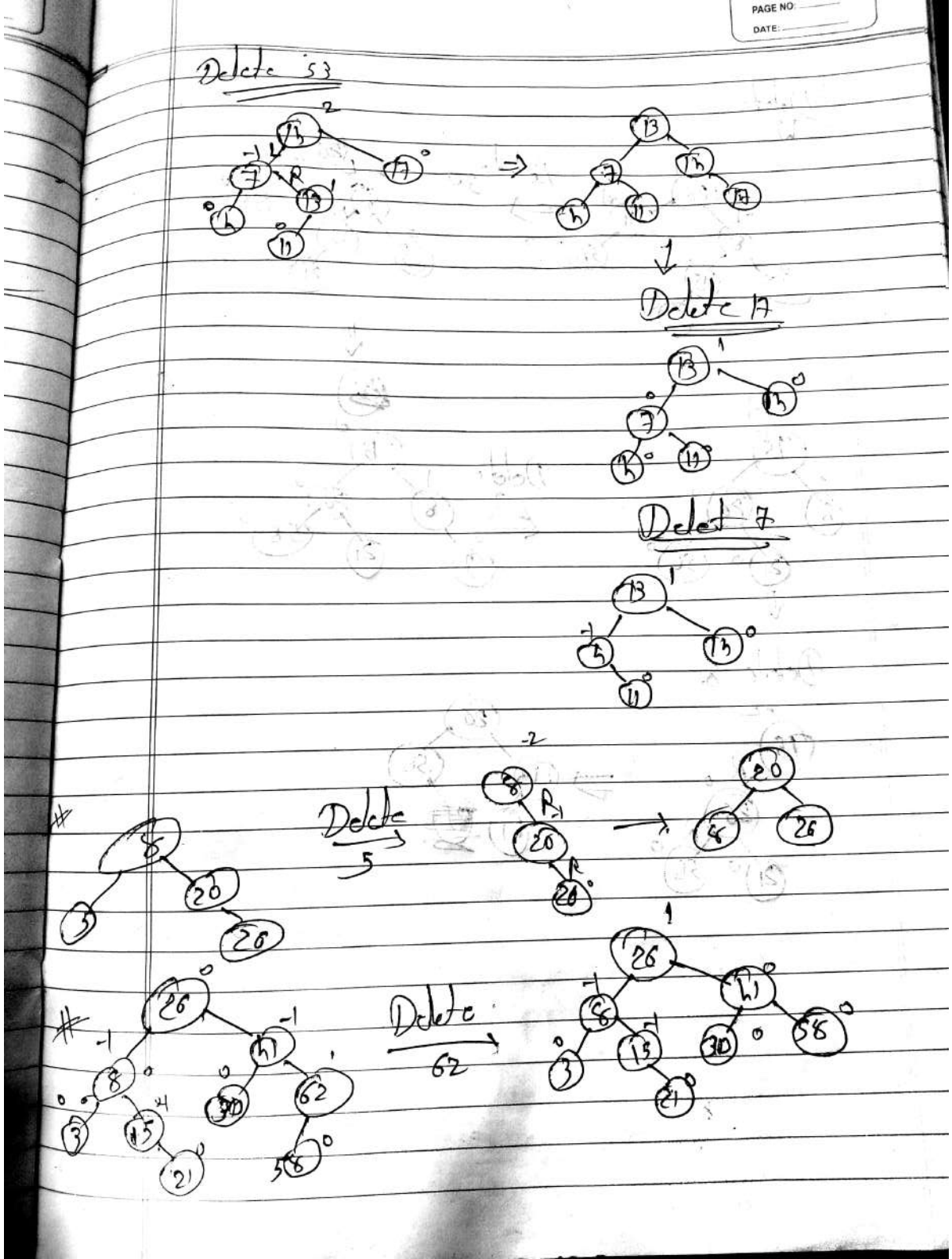




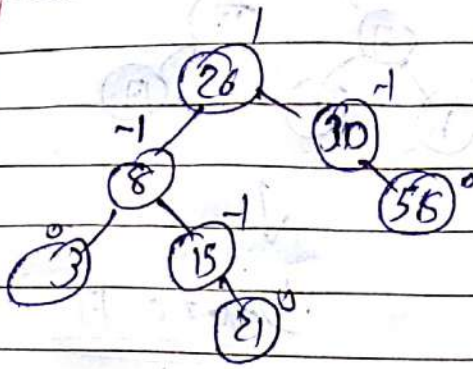
Delete 12

4, 7, 11, 12, 13, 14, 17, 53

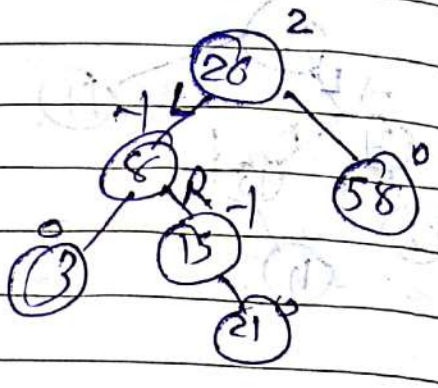


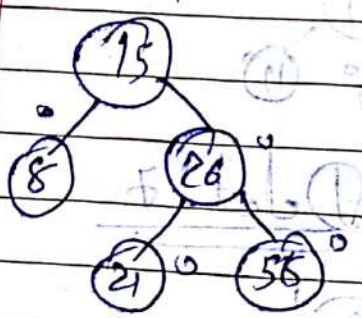


Delete 40

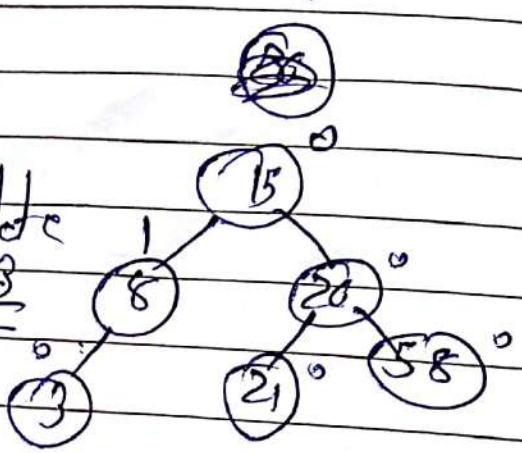


Delete 30 ⇒

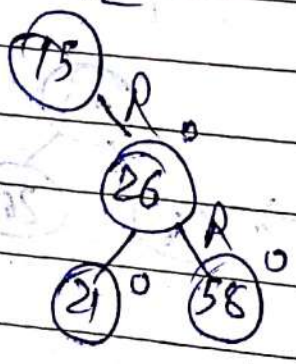




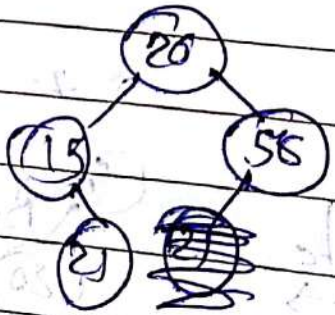
Delete 3 ⇐



Delete 8:



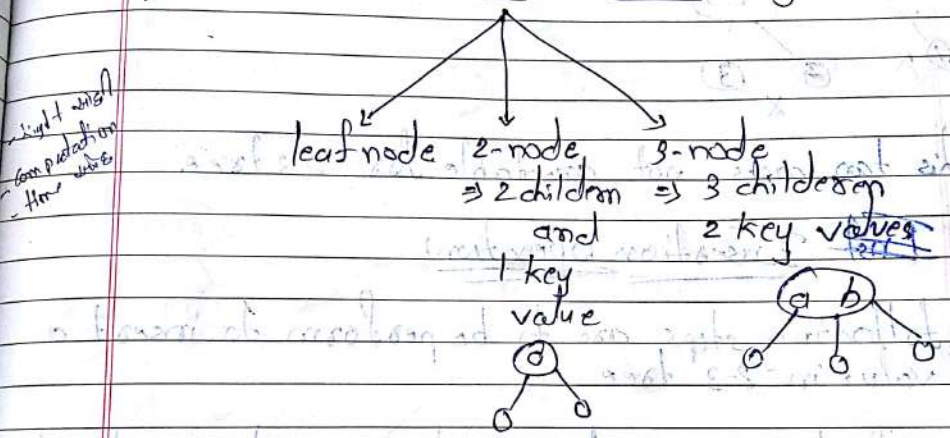
⇒



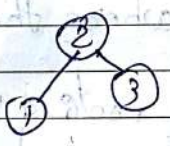
* 2-3 tree:

⇒ A 2-3 tree is a tree in which each internal node has either 2 or 3 children and all leaf nodes are at same level.

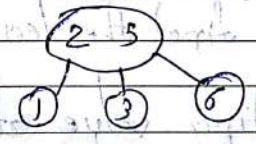
* The tree contain three different types of node:



2 Node



3 node



- A 2-3 tree is a special form of a B tree with the following properties:

(i) Each node has either one or more values

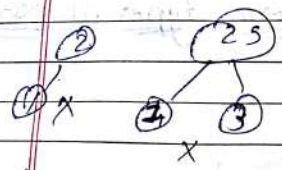
(ii) A node with one value is either a leaf node or has exactly two children.

(iii) A node with two value is either a leaf node or exactly three children

(2) All leaf node are at same level.

Advantages:

- It allows a node to have three children so tree might be shorter.
- Maintaining of 2-3 tree is simple then maintaining balanced BST.



This two cases not acceptable for 2-3 tree.

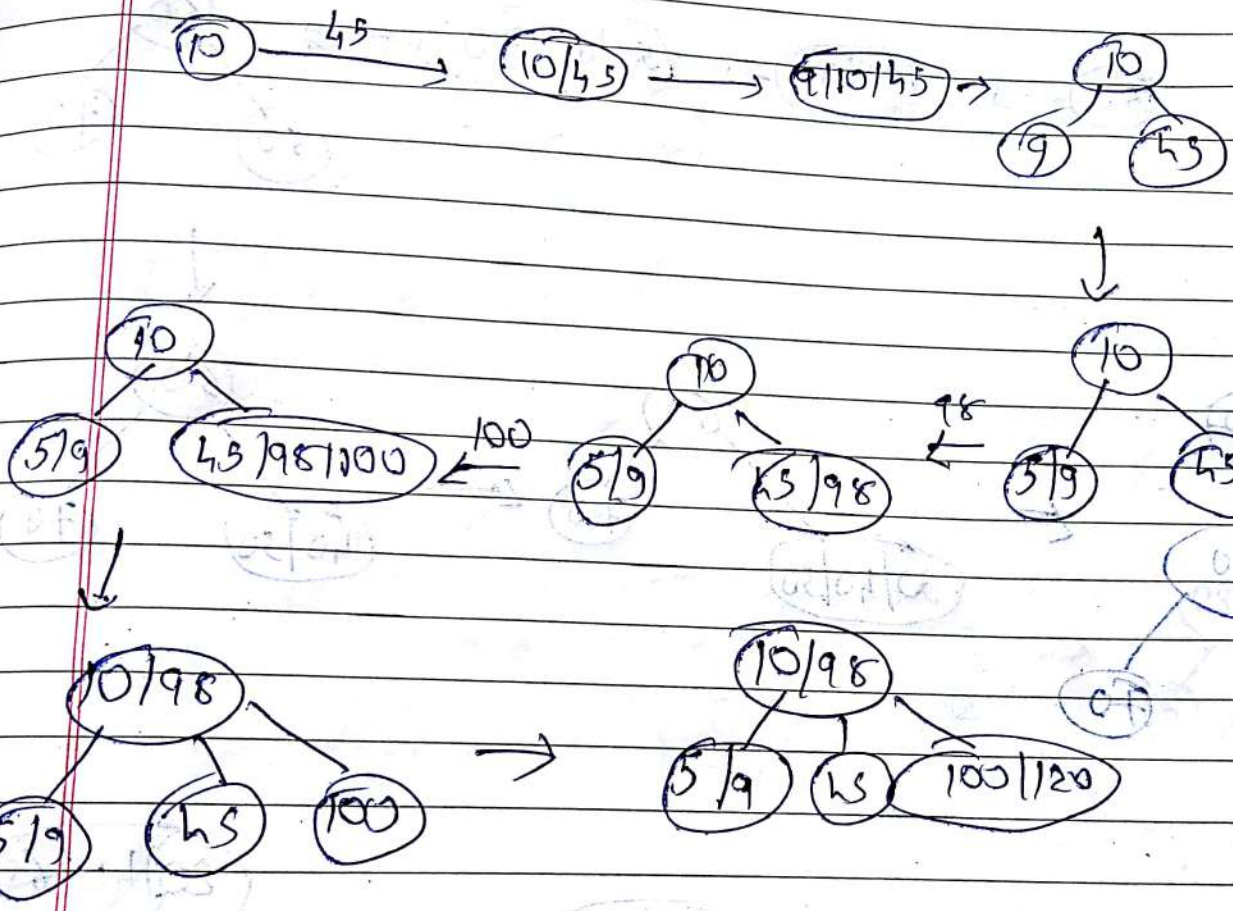
Insertion operation:

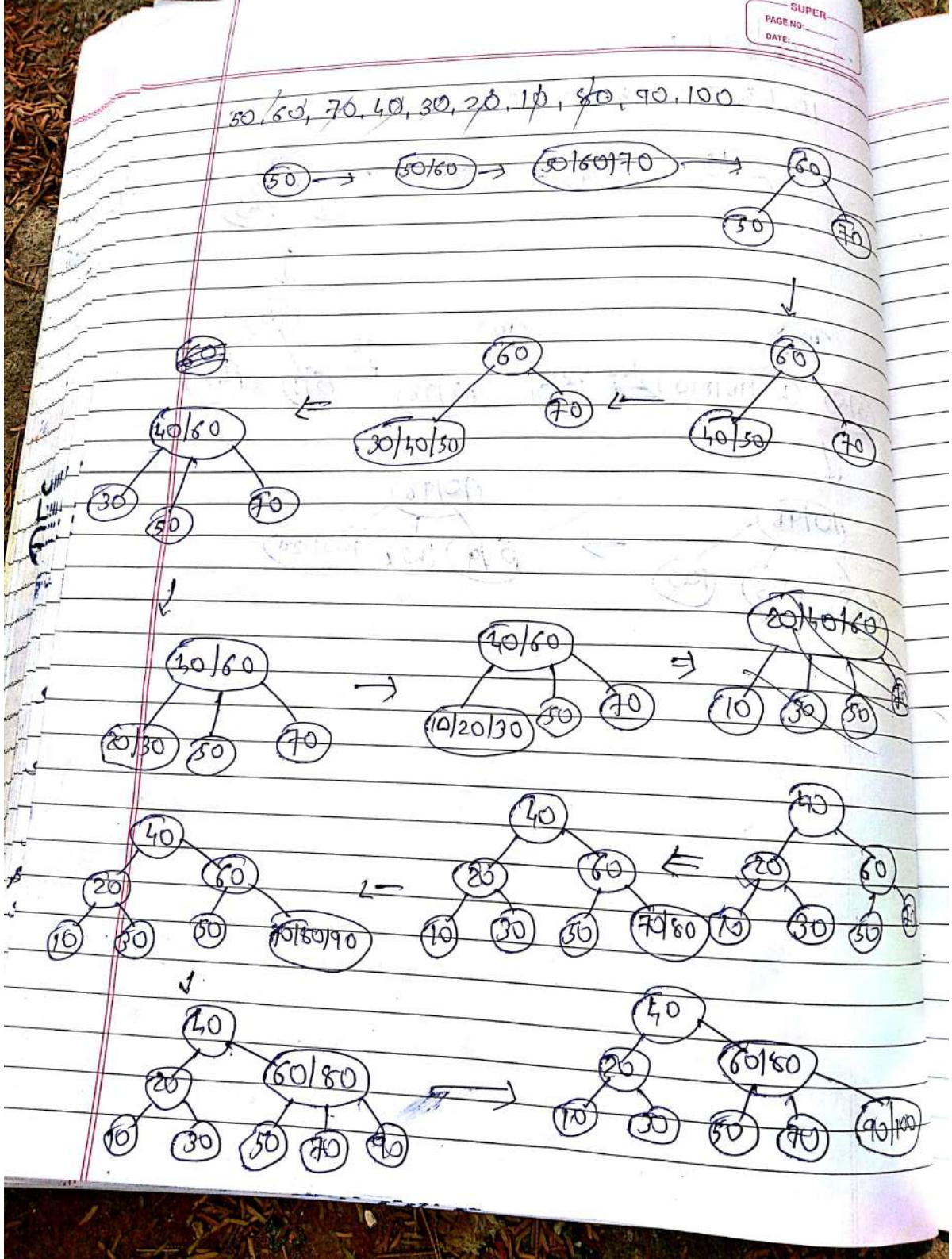
- Following steps are to be perform to insert a value in 2-3 tree

step-1 If tree is empty, create a node and put value into that node.
step-2 Otherwise find the leaf node where the value belongs
step-3 After inserting value at identify node & if node has two values then no operation to be perform
step-4 If node has a more than two values then split the node and promot medial value as a parent
step-5 If the parent has three values then same procedure should be applied and if necessary new root node should be formed.

Insertion:

10, 45, 9, 5, 98, 100, 120





58, 15, 82, 36, 23, 64, 27, 35, 45