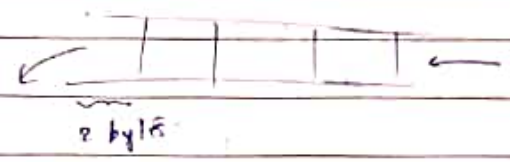
Data Structures

(i) stack → LIFO

a + b → infix
a b + → postfix
+ ab → prefix

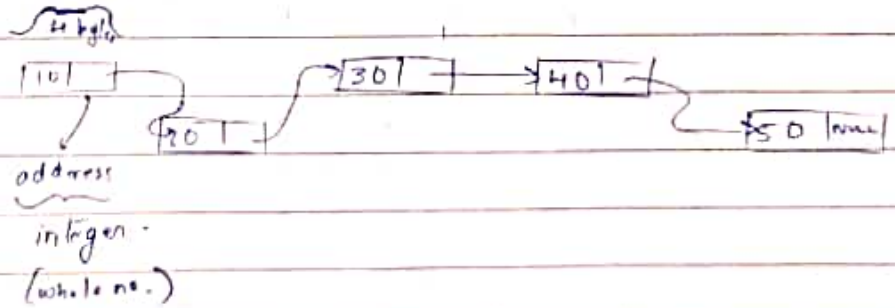
(ii) Queue → FIFO



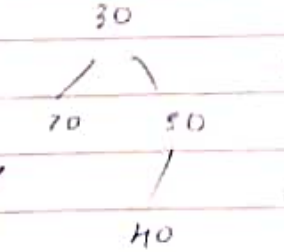
allocating memory sequentially.

(Scheduling operations)

(iii) Linked List → singly (SLL)
Doubly (DLL)
circular (CLL)
XOR (XOR LL)
Multilevel (ML LL)

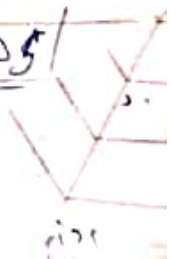


(iv) Binary tree



balanced

non-linear DS



→ Arrays (Homogeneous)

100

int a[3] = { 1, 2, 4 } ;

- *(a+1) → 1+1 = 2
- *(a+1) → 2
- a+1 → 101 (address)

(base address + index * element size) = address of element.

2d - array < row major rep / column " "

100

R_0	x	[0][0]	
		[0][1]	
R_1		1	0
		1	1
R_2		2	0
		2	1

int x[3][2] ;

array → $a[r_1][r_2]$

element → $a[i_1][i_2]$

base address + $(i_1 \times r_2 + i_2)$ × element size

$i_1 \times r_2$ → skip rows to reach ith row

i_2 → go to element in that row.

3d → base + $(i_1(r_2 \times r_3) + i_2 \times r_3 + i_3)$ × e size

nd → base + $(i_1(r_2 \times r_3 \times \cdots r_n) +$

$i_2(r_3 \times \cdots r_n) +$

⋮

$i_{n-1} \times r_n + i_n)$ × e size.

50 bytes

→ Sparse matrix (1/3rd of elements → non-zero)

5×5

	0	1	2	3	4
0	0	0	0	1	0
1	0	1	2	0	0
2	0	0	0	0	0
3	5	1	0	0	0
4	10	0	0	0	0

50 bytes

(I) Array

Storing sparse matrix

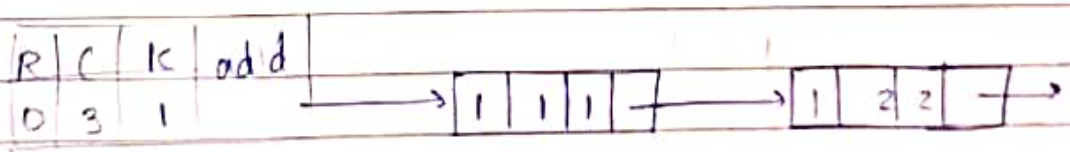
R	C	key
0	3	1
1	1	1
1	2	2
3	0	5
3	1	1
4	0	10

12

36 bytes

(elements × 3)

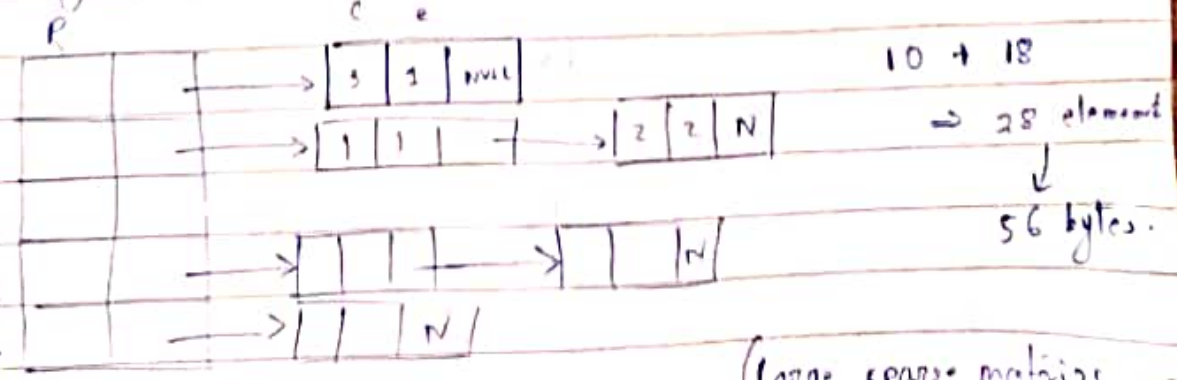
(I) in terms of linked list.



6 × 4 = 24

48 bytes

III list of list



10 + 18

⇒ 28 elements

↓

56 bytes.

(large sparse matrix

→ Stack :

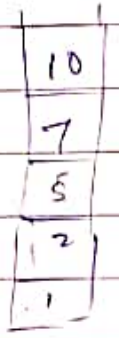
push / pop → in → top [stacktop]

is empty → prevent underflow
is full → prevent overflow

basic

search → for an element
clear
copy

```
struct stack {
 int elements [5] ;
 int top ;
} ;
```




```
struct stack s ;
struct stack *p ;

#s.elements [0]   (access elements)
  s.top = -1

void push ( stack s1 , int x) {

      s.elements [ 1+s.top] = x ;
}
```

↓

but

↓

include overflow.

so, now,

```
#define SIZE 5

struct stack
{
    int elements [SIZE];
    int top;
} s;

struct stack s;
s.top = -1

void push (stack s1, int x)
{
    if (s.top == SIZE -1)
        overflow
    s.elements [ ++s.top ] = x;
}
```

→ SORTING

→ selection sort.

it selects the smallest element in each pass and moves it to the sorted position.

first element is checked with every other, whenever it encounters smaller, switch.

then go next element.

Time complexity - $O(n^2 - n) \equiv O(n)$

ϕ → represents best case

→ Insertion sort.

Longest element will be identified in each pass, and it is moved to the sorted position

→ Applications of stack

(i) Decimal to binary

```
while (n > 0)
{
    d = n % 2 ;
    printf ("%d", d);
    n = n / 2 ;
}
```

(ii) ([{ (validity checker)

{ A + (B + [C * D]) + ([E + F] / D) * G } ;

(A) no. of left = no. of right.

```
for (int i=0 ; i < ~~str~~ n ; i++) {
    if (str[i] == '(' || str[i] == '[' || str[i] == '{')
        stack.push [str[i]] ;
    if ( str[i] == ')' || str[i] == ']' || str[i] == '}')
        a = stack.pop()
        if a
            :
            :
            use switchcase.
```

(iii) infix to postfix:

infix a+b
prefix +ab
postfix ab+

(a) final → fully parenthesize expression.

(b) innermost → postfix

(c) remove brackets

eg: (A + (B*C))
↓
(A + (BC*))
↓
(A (BC*)+)
↓
ABC*+

eg: A + B*D) + (C-F) / F

= (A + (B*D) + ((C-E) / F))

= (A + (BD*) + ((CE-) F/))

= A(BD*)(CE-)F/)+*

= ABD*+CE-F/+

eg: (((A * (B/C)) + ([E + (F * G)] - E)))

(ABC/*) + (EFG*+) E-)

: ABC/*EFG*+E-+

No fucking BODMAS

→ Display elements, if encounter an operator push to stack [if stack top [precendence] < current op [precedence]], push, else :
pop, push.

also all stor elements below.

(iii) infix to postfix

infix a+b
prefix +ab
postfix ab+

(a) first → fully parenthesize expression.

(b) innermost → postfix

(c) remove brackets

eg: (A + (B*C))
↓
(A + (BC*))
↓
(A (BC*)+)
↓
ABC*+

Eg: A + B*D) + (C-E) / F

= (A + (B*D) + ((C-E) / F))

= (A + (BD*) + ((CE-) F/))

= A(BD*)(CE-) F/) +

= ABD* + CE - F/+

eg: (((A * (B / C)) + ([E + (F * G)] - E)))

(ABC/*) + (EFG*+)E-)

= ABC/*EFG*+E-+

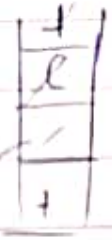
DMAS

No fueling

→ Display elements, if encounter an operator push to stack [if stack top [precendence] < current op [precedence] , push, else : pop, push.

also all star elements below.

Eg: A ^ B * C - D + E / F / (G + H)



don't care abt precedence.

Eg: ((A + B) * C - (D - E)) ^ (F + G)

AB + C * DE - - FG + ^

Eg: (A + B) * C + D + E * F - G

AB + C * D + EF * + G -

postfix → infix (stack pop push like we did in ECS)

now,
infix → prefix (right to left)

> HG + F / E / D + C B A ^ * - → then reverse

- * ^ A B C + D / E / F + G H

(ii) GF + ED − C BA + * − ^

≡ A − * + AB C − DE + FG

(iii) GF E * − D + C BA + * +

≡ + * + AB C + D − * EF G

Q-(iv) ((A − B) + C * (D + E)) − (F + G)

- AB − C DE + * + FG + − → [postfix]
- GF + ED + C * BA − + −

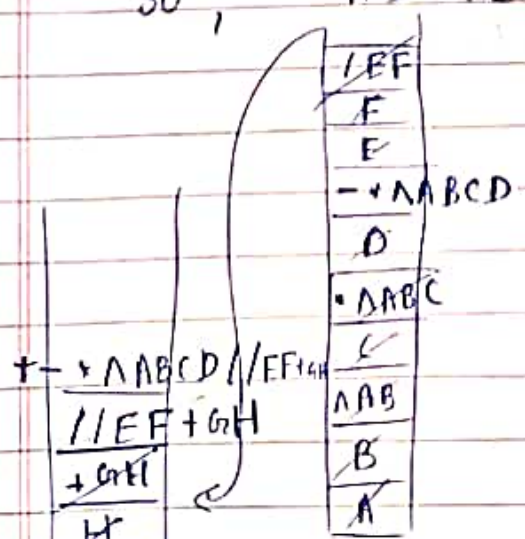
− + − AB * C + DE + FG → [prefix]

for prefix to infix → start the stack operation from back (in reverse

now, postfix to prefix (Left to right

push operands, when encounting operator, pop twice. then push [operator, top2, top] back in stack.

so, AB ^ C * D − EF / GH + / +



Infix to postfix (L→R)

curr op <= Top → POP

A*B +CD − *

−CD
D
C
+AB
B
A

*+AB−CD

→ Evaluation

2
4

postfix → 24−2* (L→R)

prefix → 2−4 (R→L)

→ Prefix to postfix (R→L)

operand → push

operator → Top1 Top2 operation

→ postfix to infix (L→R)

A^B*C−D+E/F/(G+H)

post: AB^C*D−EF/*GH+/+

pre : HG+FE//DCBA^*−+

rev

+−*^AB CD // E F + GH

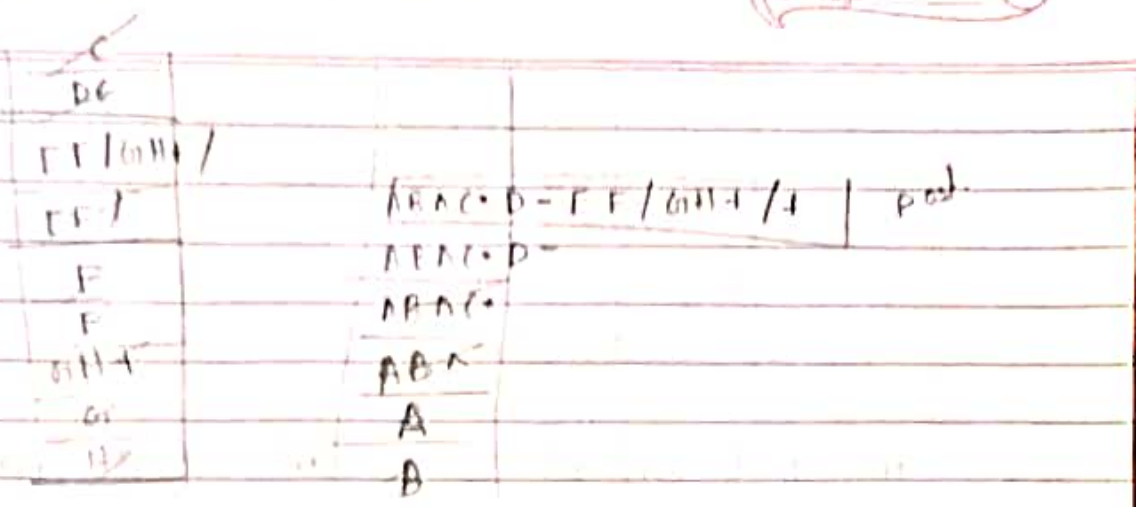
+GH
/FF
−*^ABCD
D
*^ABC
C
^AB
B
A

+−*^AB CD // EF + GH

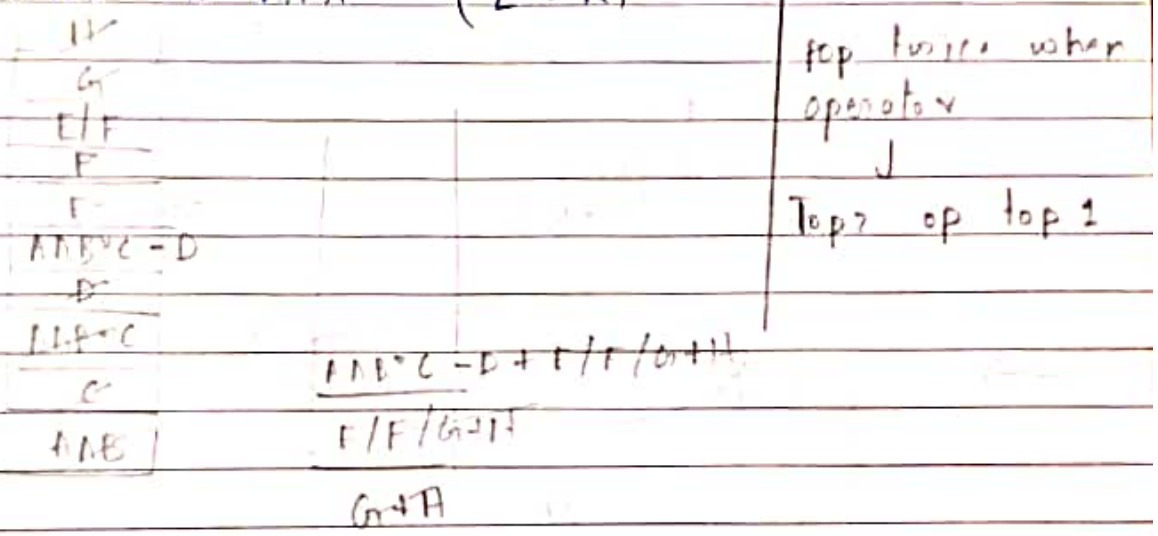
// FF+GH

Infix to prefix (R → L)

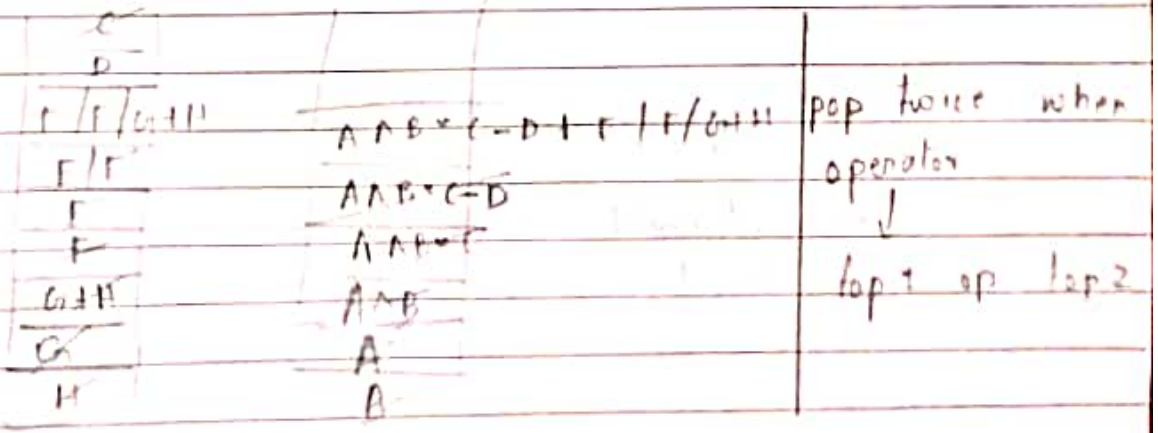
(com op >= Top ⇒ push



→ postfix to infix (L → R)



→ prefix to infix (R → L)

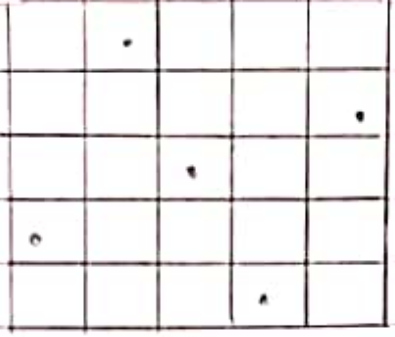


→ N-Queen Prob.

n×n → board

↓

n queens to be placed!

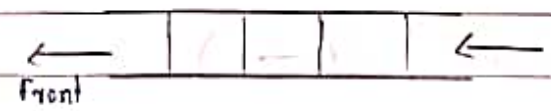


refer to Backtracking

```
int fact (int n)
{
    if (n == 0)
        return 1;
    x = n-1;
    y = fact (x);
    return (n * y)
```

→ QUEUE

LILO / FIFO




```
struct queue
{
 int element [SIZE];
 int front, rear;
}
```

```
q.front = 0
q.rear = -1
x = q.element [q.front++]      pop
q.rear = SIZE-1
    q.element [++q.rear] = x    insert
```

(i) Circular queue → vacant spaces can be occupied.

(ii) I/P restricted queue → Insert in one direc", deque in both dir
(enque)

(iii) III^{rly} o/p " " → dequeue → 1, enque → 2.

(iv) Double ended queue

Priority queue → Ascending → element with less priority will be removed first.
→ Descending → opp.

higher.

2	3	1
10	20	30

we can arrange elements acc to priority, while inserting.

→ Circular Queue

INSERT if ((front == 0 && rear = SIZE-1) || front = rear+1)
overflow.
else {
if (rear == -1)
insert
front = rear = 0.
else if (rear == SIZE-1)
insert
rear = 0;
else :
insert
rear ++ ;
q.element [q.rear] = x;
}

DELETION

```
if (front == -1)
        underflow
else{
    x = element [front]
    if (front == rear)
        front = rear = -1;
    else if (front == SIZE - 1)
        front = 0;
    else:
        front ++;
}
```

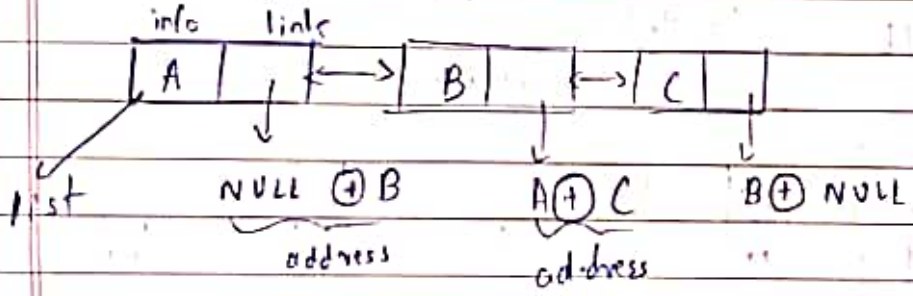
LINKED LIST

* singly linked list
  doubly linked list
  Circular " "
  } just last node.

CDLL

```
struct node
{
    int info;
    node *next;
};
```

XOR - linked list.



$A \oplus B = B \oplus A$

$A \oplus (B \oplus C) = (A \oplus B) \oplus C$

$A \oplus A = 0$, $A \oplus 0 = A$

$(A \oplus B) \oplus B = A$

```
node *curr = list
node *prev = NULL;
node *next;
```

traversing from right to left.

```
while (curr != NULL) {
    printf (curr -> info)
    next = XOR (prev, curr -> link);
    prev = curr;
    curr = next;
}
```

bitwise

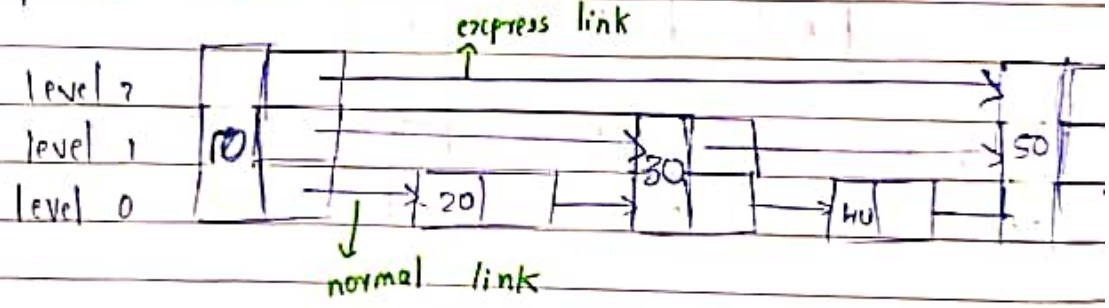
for Right to left ,

```
node * curr = list;
node * next = NULL
node * prev

while (curr != NULL)
{
    ...

    prev = XOR (next, curr → link);
    next = curr;
    curr = prev;
}
```

→ Skip LL (sorted.



start level 2 → check next, if ~~not~~ in range

↓

lower level

↓

check next element

↓

if not in range

↓

move next element

if in range

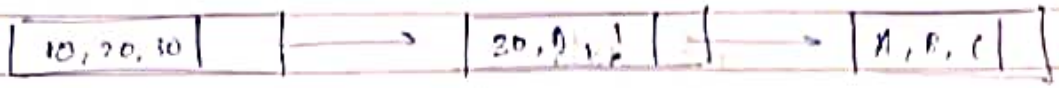
↓

lower level

↓

chek next element.

→ Unrolled LL



→ self-adjusting LL

↓

after searching an element in LL, bring it to front

↳ for LL to be searched a lot.

(i) move to front

(ii) counter → have a counter associated with each, ↑ when accessed, sort (on counter) in desc. order

(iii) transpose → accessed element swaps with previous list.

→ Queue, stack using LL : (we can release memory after deletion)

→ Code (SLL)

↓

↳ Insertion → beg, end, insert before, after

↳ same for deletion

↳ Display

↳ Search

↳ Sort

↳ Merge


```
struct node
{
    int data;
    struct node *next;
}*list;

list = NULL;

insatbeg (int x) {
    node *temp;                    → declaring pointer var
    temp = (node*) malloc(sizeof(node)) → initialized
           typecasting.  ↘ returns pointer.
    temp → data = x
    temp → next = NULL;
    if (list == NULL)
        list = temp;
    else {
        temp → next = list;
        list = temp;
    }
}

insatend (int x) {
    node *end = (node*) malloc (sizeof(node));
    node *temp = (node*) malloc (sizeof(node));

    end = list;
    while (end → next != NULL) {
        end = end → next;
    }
    temp → data = x;
    temp → next = NULL;
    if (list == NULL)
        list = temp;
```

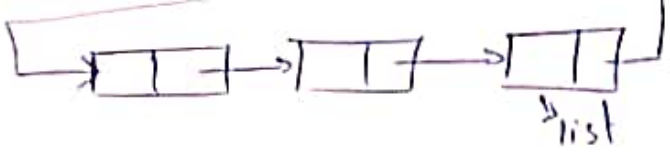
```
    else {
        end -> next = temp ;
    }
}

ins bef num ( int x , int d ) {
    node *temp , *l = list , *prev ;
    temp = (node *) malloc ( sizeof (node)) ;
    prev =
    temp -> data = x ;
    temp -> next = NULL ;

    if (list == NULL)    list = temp ;
    else {
        while (l -> data != d && l      != NULL)
            prev = l ;
            l = l -> next ;
        }
        if (l -     == NULL) printf ("NOTHEY") ;
    }
}

ins aft num ( . .
    .
}

del at beg ( )
{ node *t = list ;
  if (list == NULL)
        empty
  else
      t = l -> next ;      list = list -> next
      list = t ;           free (t)
      free (t)
```



→ CLL

```
ins al beg (int x) {
    node *temp = (node*) malloc (sizeof (node));
    temp → data = x ;
    temp → next = NULL ;
    if (list == NULL) {
        list = temp ;
        list → next = list ;
    }
    else {
        temp → next = list → next
        list → next = temp ;
    }

del al end () {
    node *temp = list ;
    while ( t → next != list )
        t = t → next

if (list == NULL)
    error
else if (list → next == list)
    list = NULL ;
    free (t)
t → next = list → next
list = t ;
free (t) ;
free (list) ;
list = t ;
```

→

→ DLL

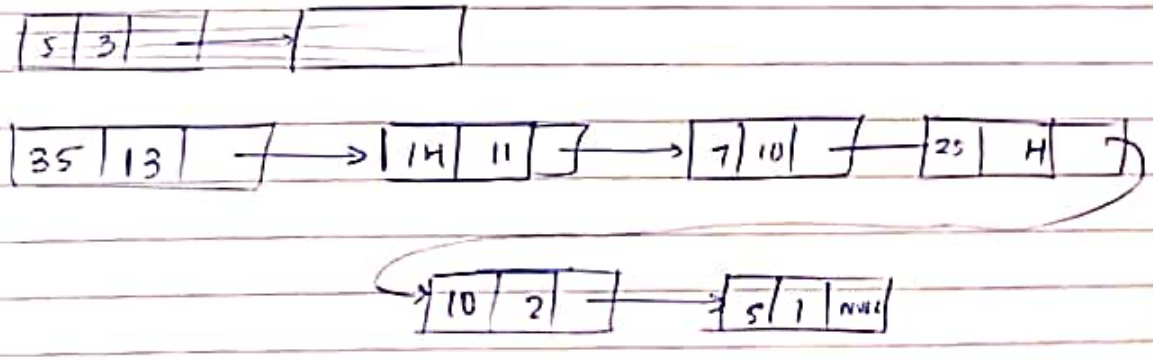
```
insertatend (int x) {
        node *l = list
        node *temp = ...
        temp -> data = x
        temp -> next = NULL
        temp -> prev = NULL.
        if (list == NULL)
                list = temp ;

        else {
                while (t -> next != NULL)
                        t = t -> next ;
                l -> next = temp ;
                temp -> prev = t ;
```

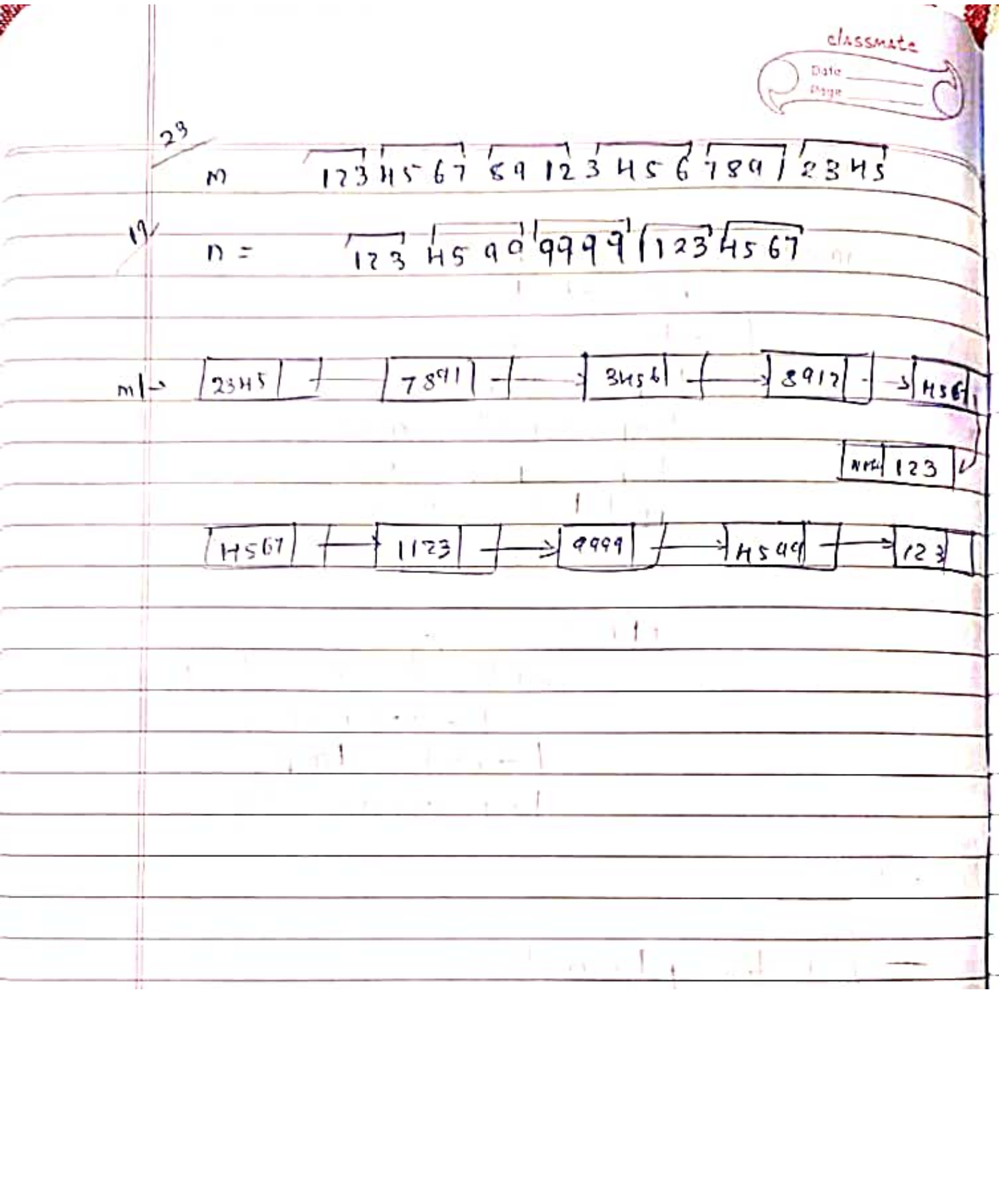
→ Add two polynomials

P : $5x^3 + 2x + 1$

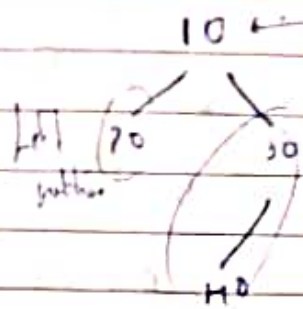
Q : $7x^{10} + 5x$



Now traverse, compare, if same power, add coeff, delete.


23
m 123 45 67 89 123 45 6 789 2345
19
n = 123 45 99 9999 123 45 67
m1→ 2345 7891 3456 8912 45 67
NULL 123
45 67 1123 9999 45 99 123

BINARY TREE



10, 30 is ancestor for 40. → child
40 is descendant for 10
↳ right descendant.

Level → root → level 0.
max level of any

Every leaf node has ~~non empty~~ let only two entries.

→ Complete binary tree.
Complete binary tree of depth d is a strictly binary tree all the leaves are at level d.

→ Almost complete binary tree.
(i) A binary tree if depth of any node n at level < d-1 has 2 children.

(ii) Any node n in a tree, the right descendent at level D, n must have a left son and left descendant of n is either a leaf at level D or it has two sides.

info(p)
parent (r)
left (p)

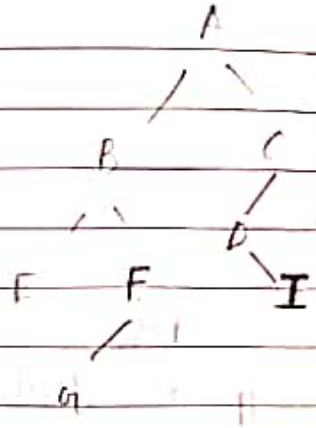
Isleft (p) → parent (p) if p == left/ parent

(L R r)

→ Preorder, inorder, postforder. → L r R

↓

Root Left right (R L r)



(i) Inorder

E B G F A D I C

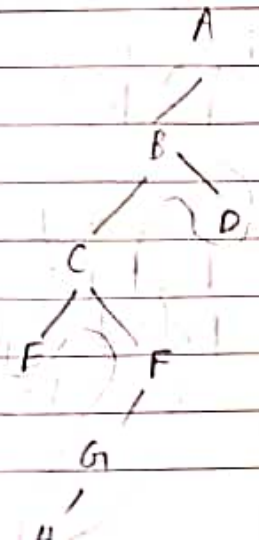
(ii) Preorder

A B E F G C D I → draw line, encounter first

(iii) Post order → except → encountered last

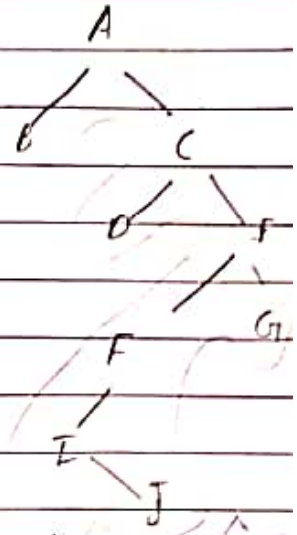
E G F B E I D C A

Q. Inorder → E C H G F B D A
Pre " → A B C E F G H D
Post " → E H G F C D B A



Inorder :

Q. B A D C I J F E G



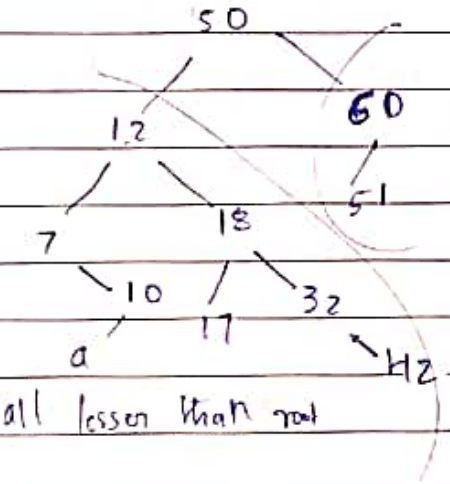
Pre : A B C D E F I J G

Post : B D J I F G E C A

→ Checking duplicates

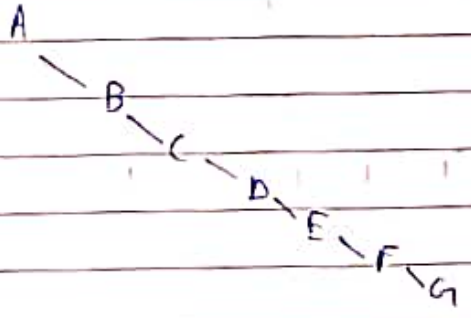
50, 12, 7, 18, 32, 60, 17, 51, 42, 10, 17, 60, 9

Q First → node
less → left subtree
>= → right "

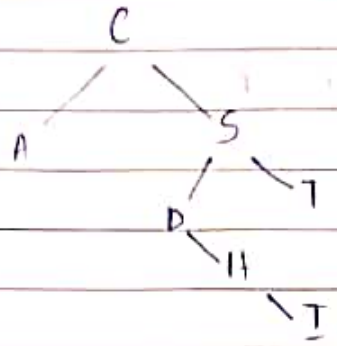


Binary search tree

Q. A B C D E F G



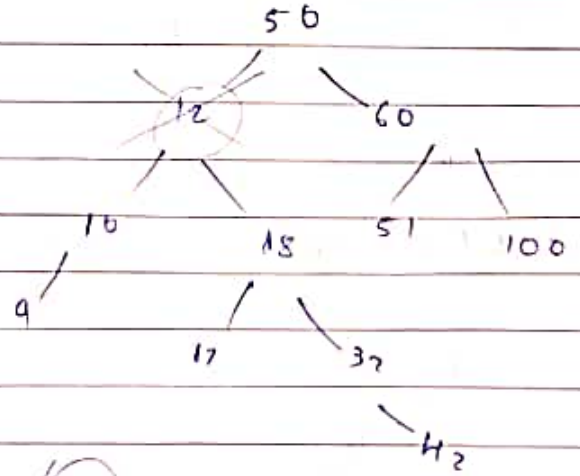
9 C A S T D H I



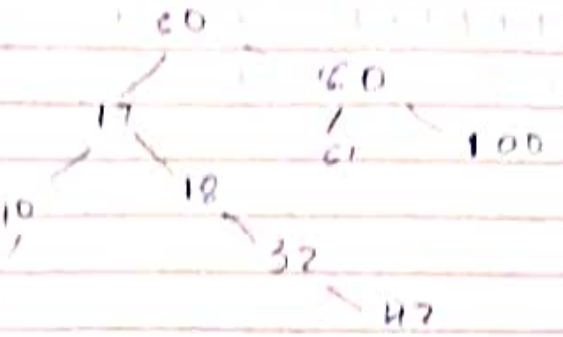
→ Deleting node.

(i) Leaf
(ii) node with 1 child
(iii) " " 2 child

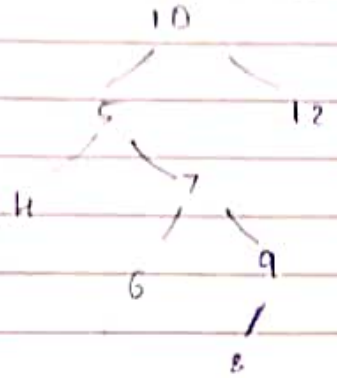
→ (i) go to node and delete.
(ii) deleted node is occupied by child subtree
(iii) node can be replaced by inorder successor or predes



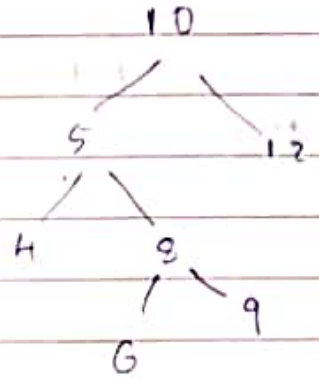
9 10 12 17 18 32 42 50 51 60 100



Q.



4 5 6 7 8 9 10 12



→ given A B C

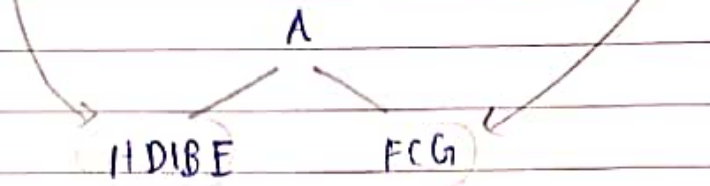
[In & Post
In & Pre

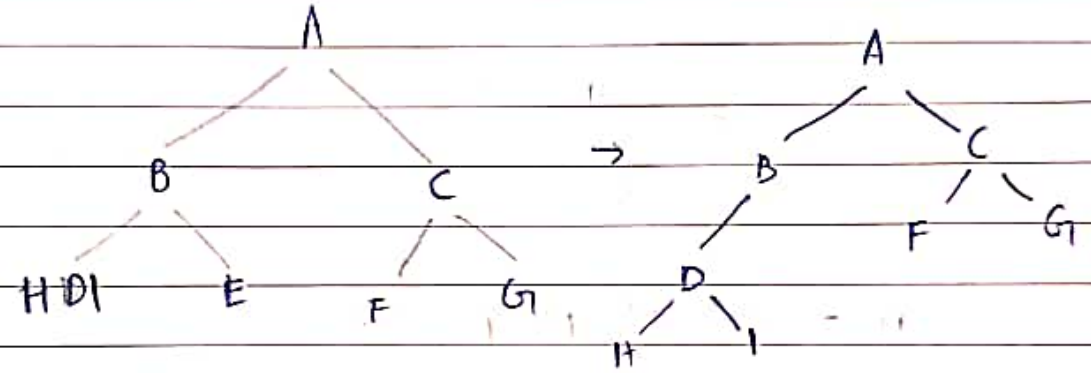
SBT → pre & post.

Q. Post : H I D F B J G C A

In : H D I B E A F C G

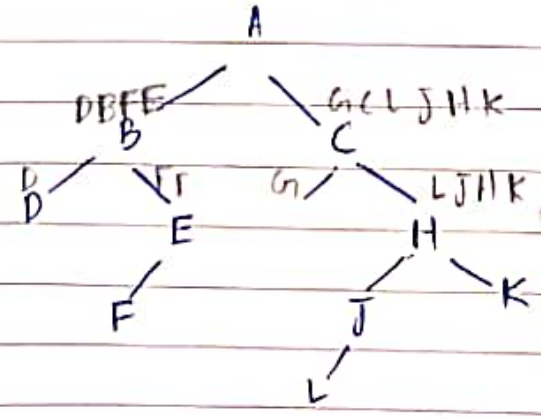
left right





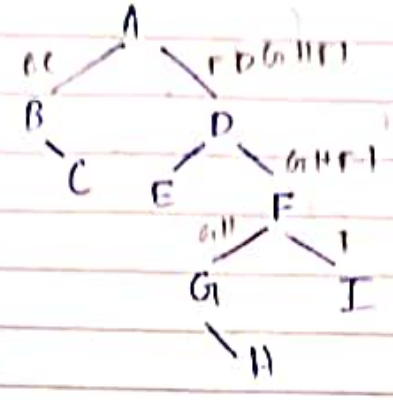
Q. In : D B F E A G C L J H K

Post : D F E B G L J K H C A



Q. Pre : A B C D E F G H I

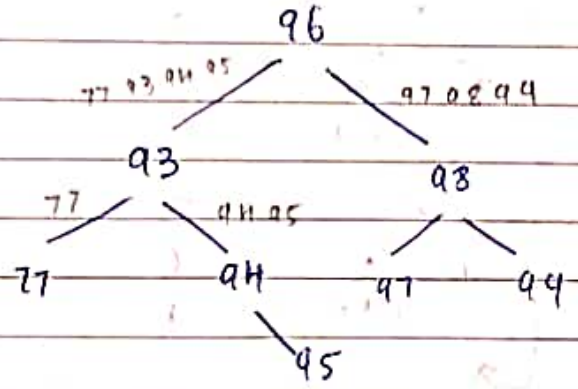
In : B C A E D G H F I



In : 77 93 94 95 96 97 98 99

Pre : 96 93 77 94 95 98 97 99

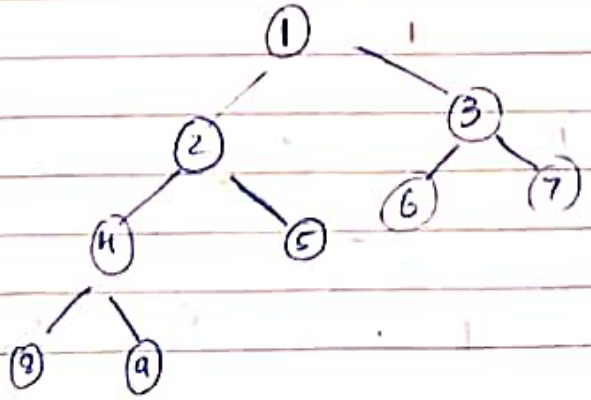
Rls



Preo : 1, 2, 4, 8, 9, 5, 3, 6, 7 root left right

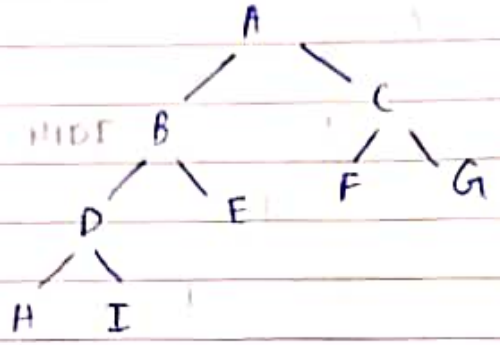
Posl : 8, 9, 4, 5, 2, 6, 7, 3, 1 left right root

L Right

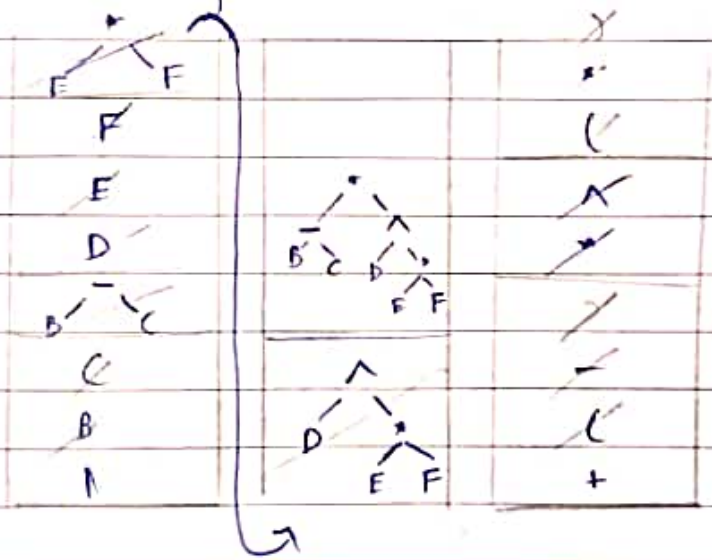


Pre: A B D H I E C F G RLY

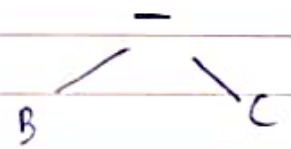
Post: H I D E B F G C A LYR

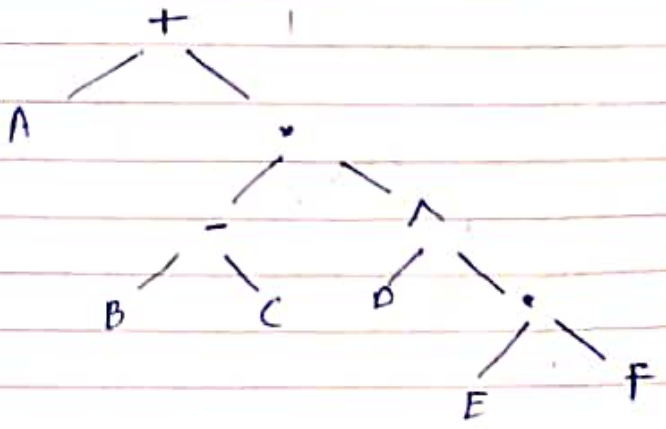


→ Representing Exp - BT

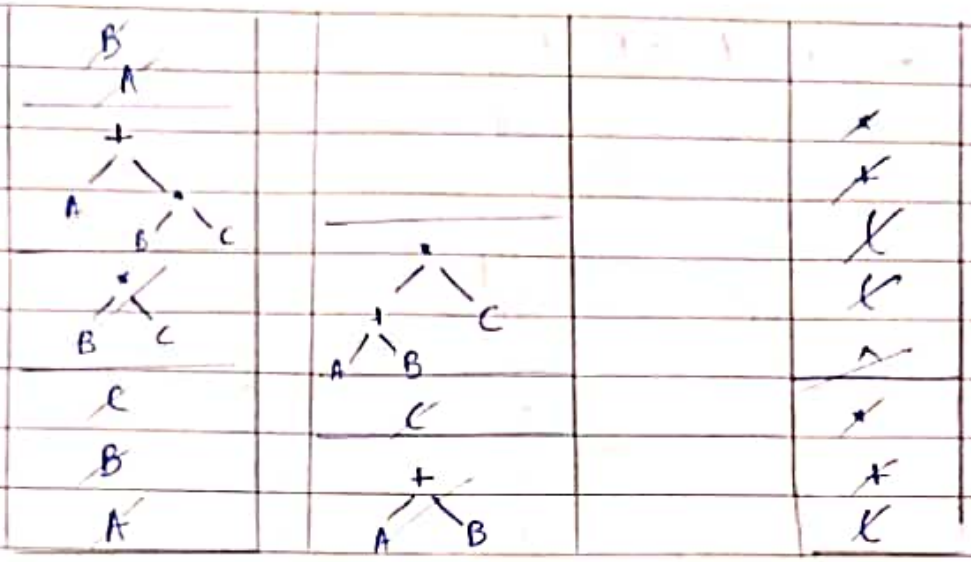


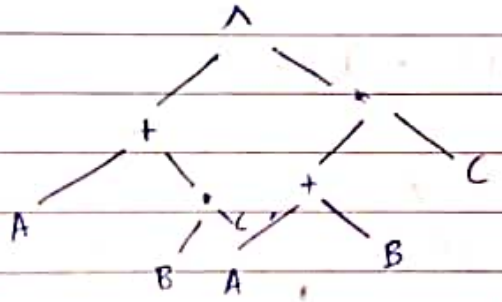
then pop operand twice



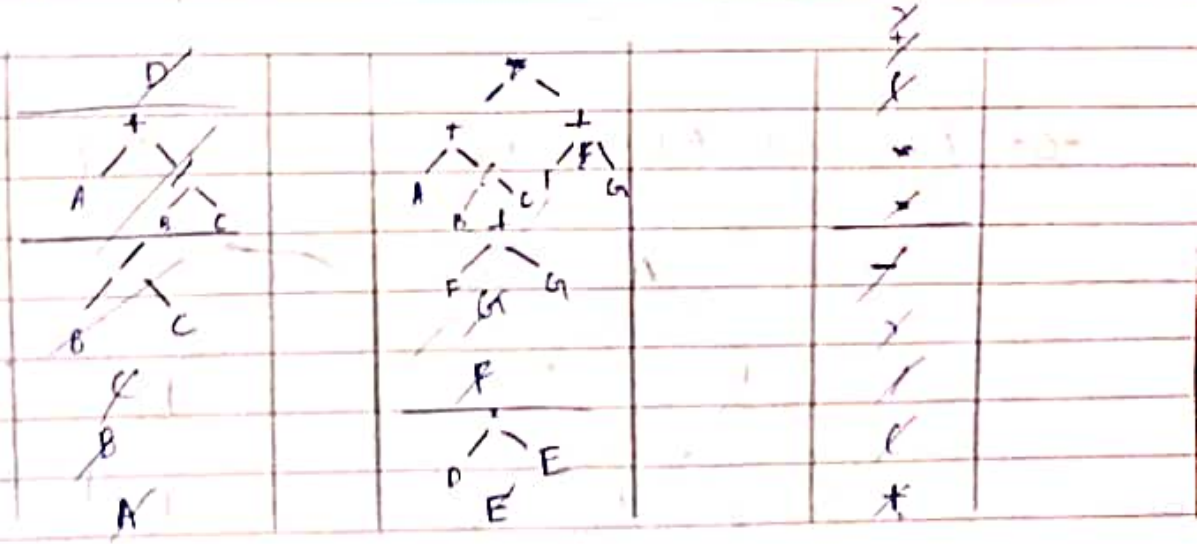


Q. $(A + B * C) \wedge ((A + B) * C)$



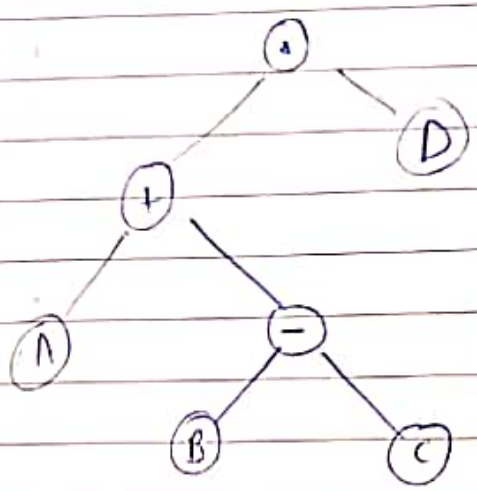


Q. $A + (B/C) - D * E * (F + G)$



→ Preorder

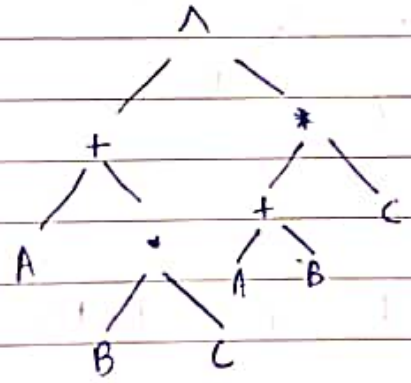
→ * + A - B C D



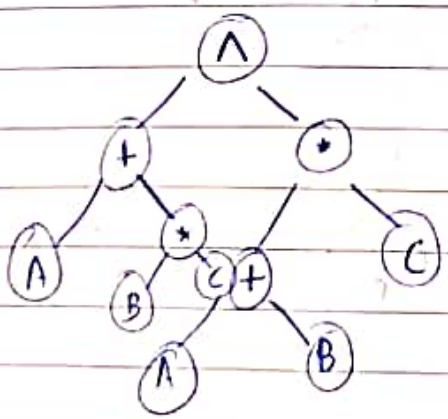
when reach operator, go left subtree

when operand backtrac to previous node until right child is not there.

Q. ^ + A * B C * + A B C



→ A B C * + A B + C * ^ Postorder



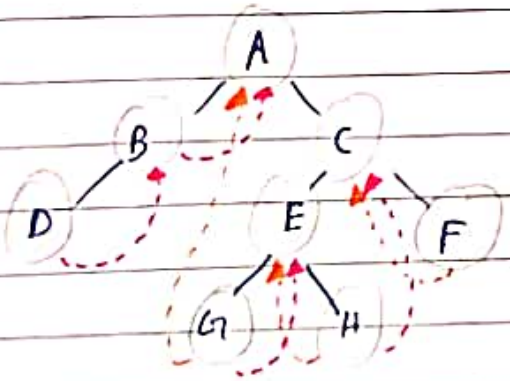
Go from right to left, go right child if operand, go both left

or mirror image of left from right & left

```
void inorder (tree *p)
{
    if (p != null)
        inorder (p -> left);
    print (p -> info);
    inorder (p -> right);
}
```

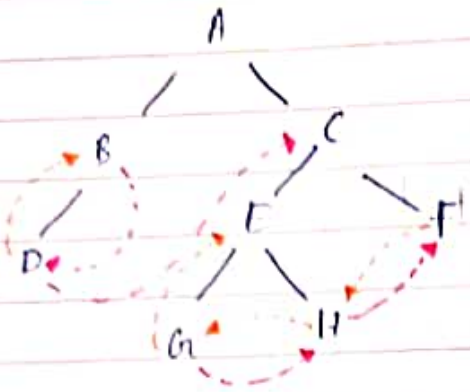
too many recursive calls.

→
```
tree *p1;
p1 = p;
do {
    while (p1 != null)
    { push (s, p1);
      p1 = p1 -> left;
    }
    if (!empty (s))
        p1 = pop (s);
        printf (p1 -> info);
        p1 = p1 -> right;
    }
} while (!empty (s) || p1 != null)
```



in-TBT { Right ■ - in Threaded
Left ■ - in Threaded

DBAGEHCF



Pre - TBT

right - pre

left - p

A B D C E G H F

→ transmitting A B A C C D A

ASCII → 7 × 7 = 49 bits

to reduce

Huffman Algorithm.

A B A C C D A

A — 3

C — 2

B - 1

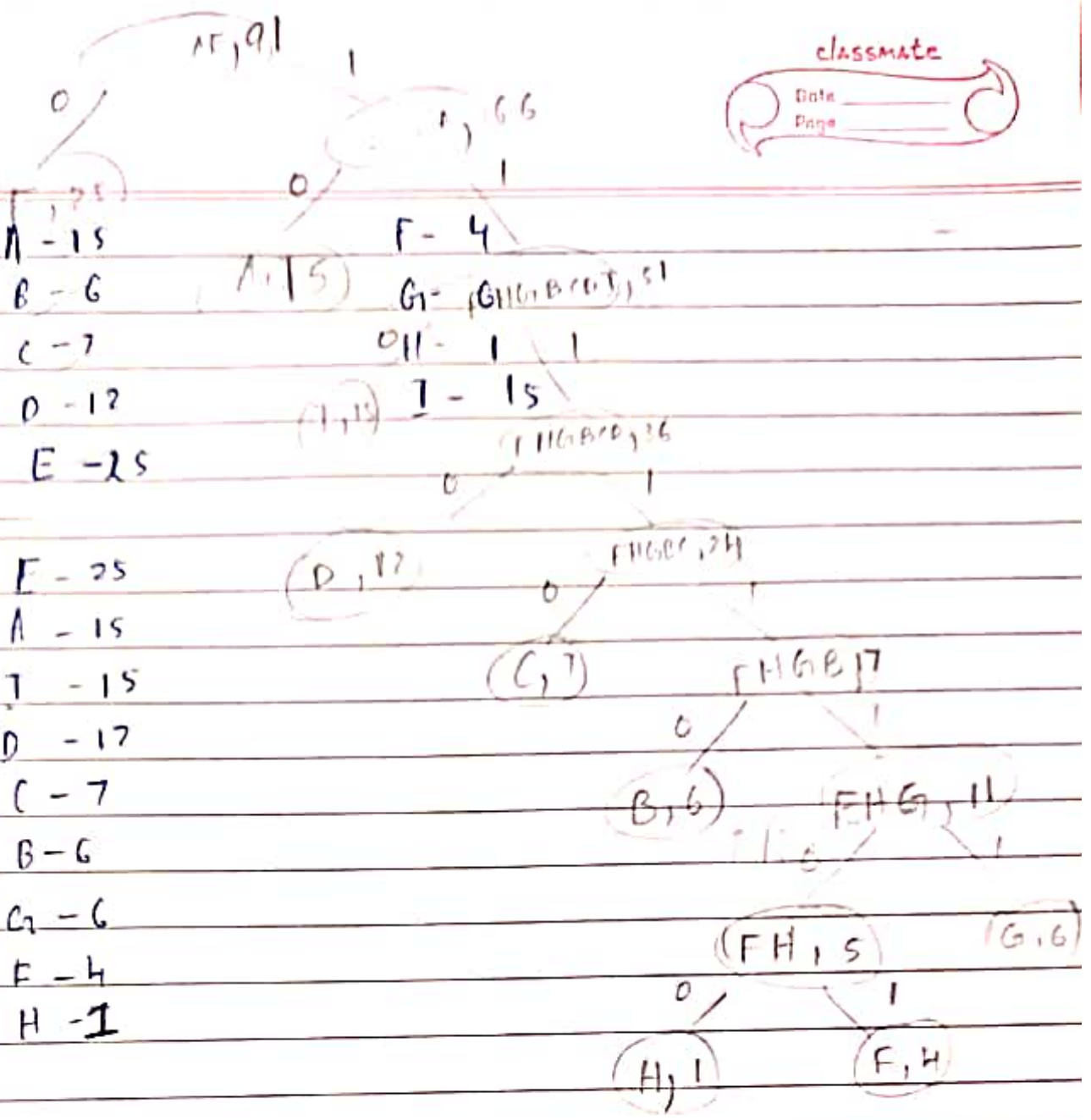
D - 1

Q.

A - 15
B - 6
C - 7
D - 17
E - 25

E - 25
A - 15
T - 15
D - 17
C - 7
B - 6
G - 6
F - 4
H - 1

F - 4
G - 6
H - 1
T - 15



→ 18
25
34
55
75
85
99
110

Binary search tree

75
34
99
55
110
12
25

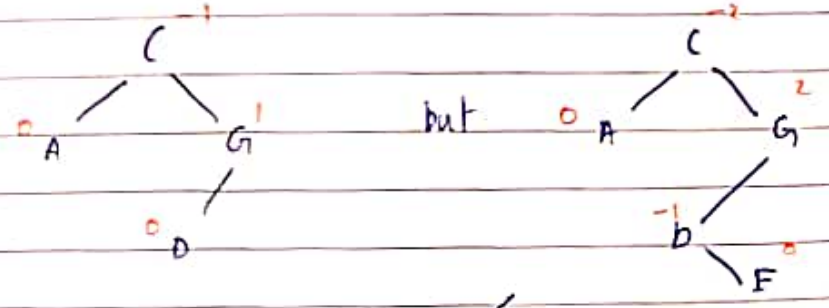
→ AVL Tree. (Addson - velshill Landis)

(i) BST

(ii) Balance factor $\{-1, 0, 1\}$

height of left subtree - h (Right subtree)

Eg:



to change this to AVL

(i) choose a node A with balance factor $\neq \{-1, 0, 1\}$ and $= \{2, -2\}$ that is nearest to the inserted node.

4 possible setups to do rotation

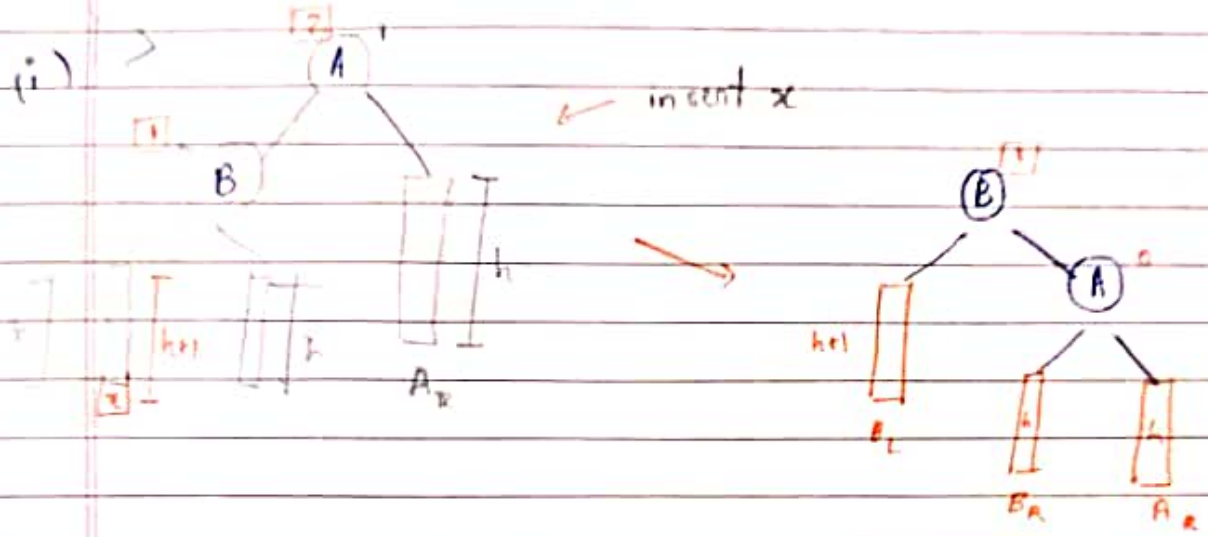
(i) LL - left subtree (ST) of LST

(ii) RR - RST of RST

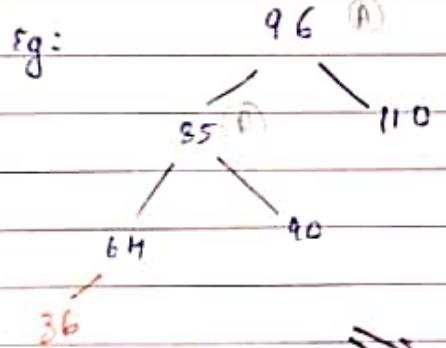
(iii) LR - RST of LST

(iv) RL - LST of RST

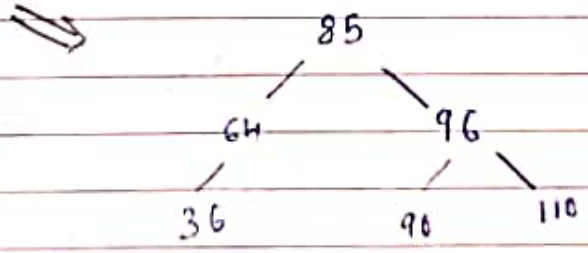
(i)



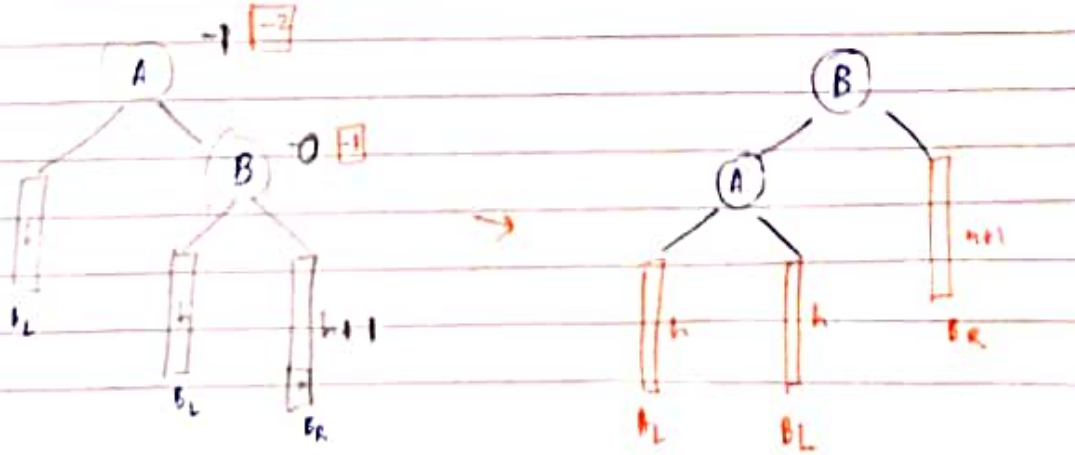
Eg:

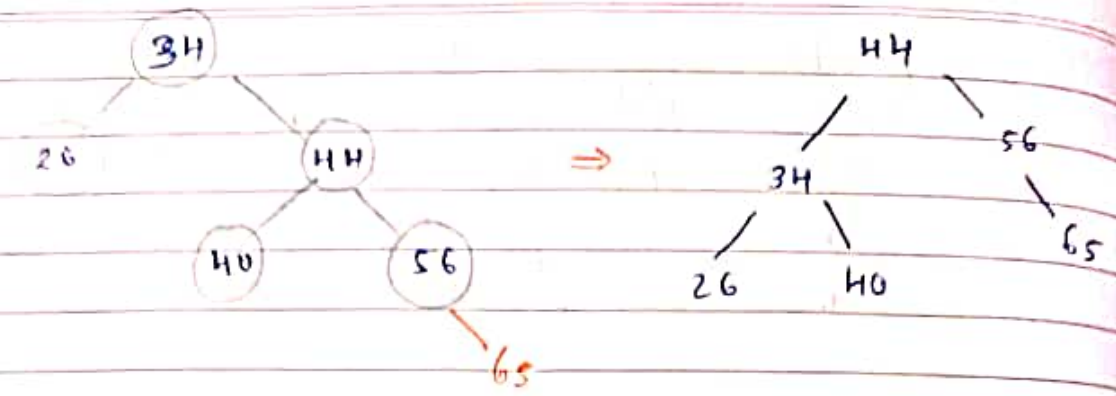


insert 36



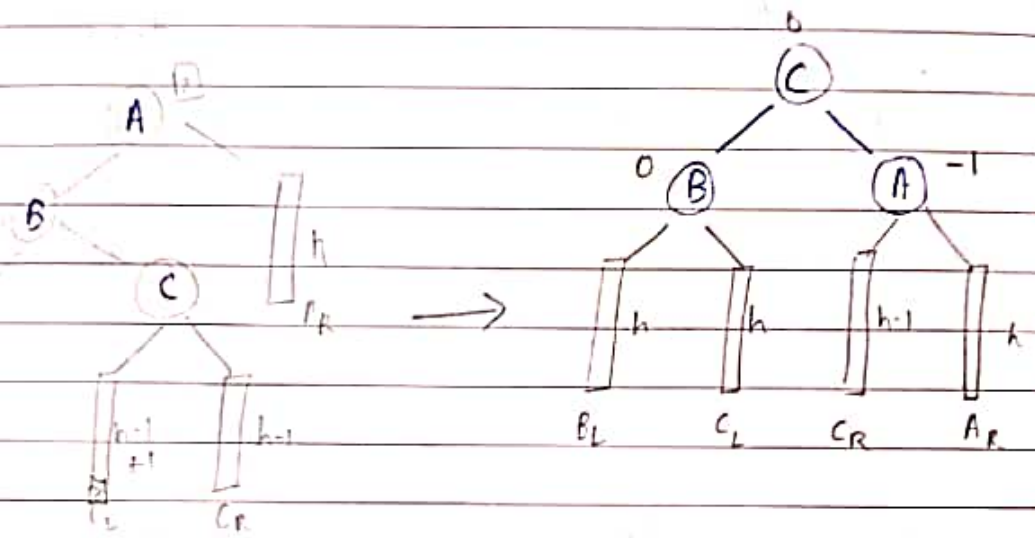
(ii)





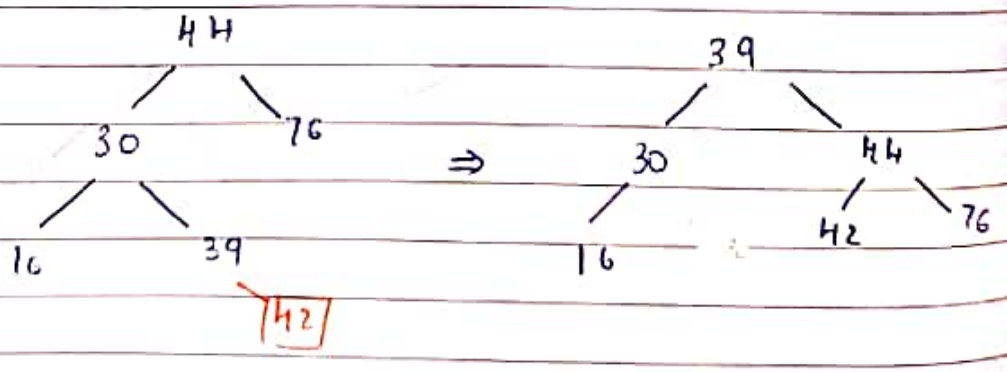
(iii)

2 - rotations

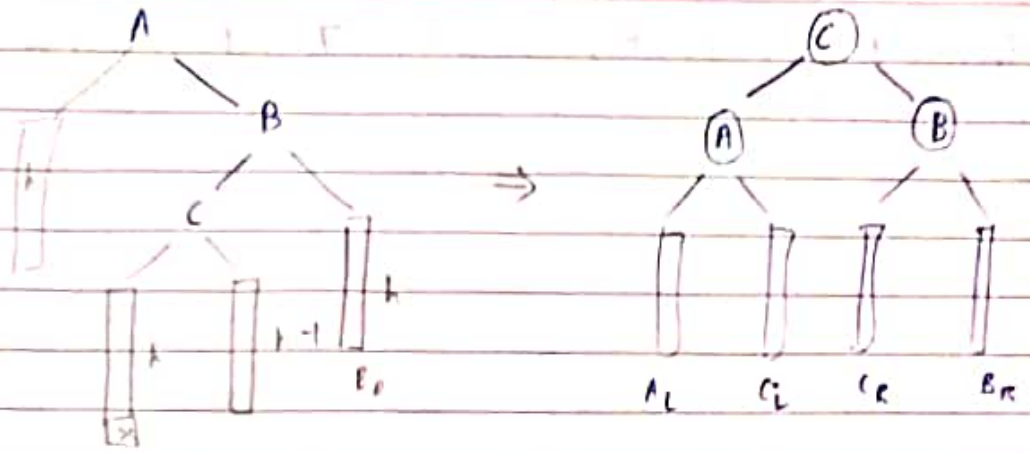


If either left or right one inserted

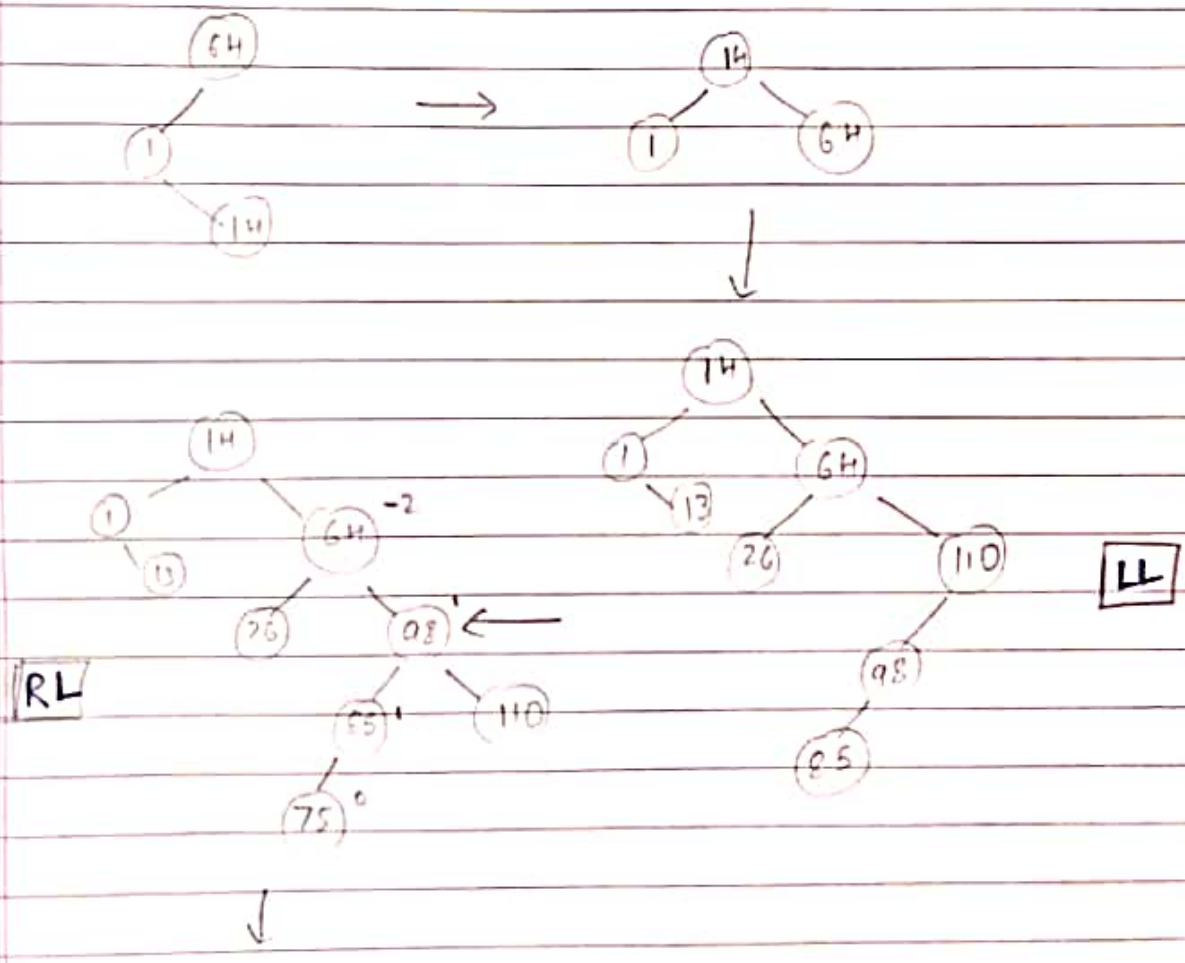
Eg:

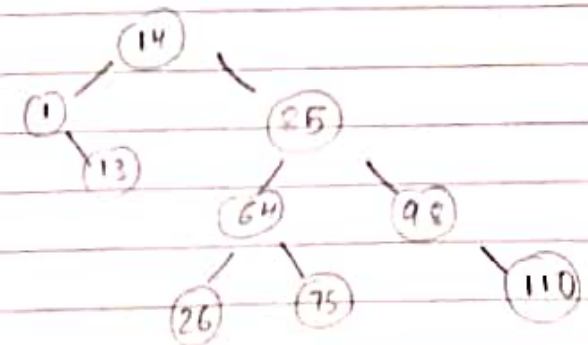


(iv)



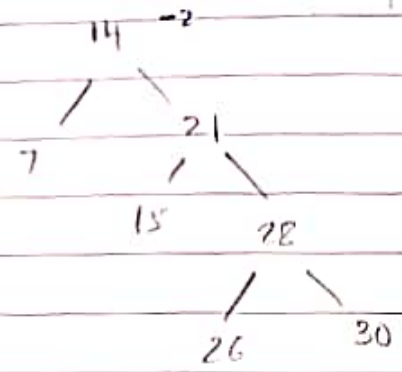
Q. 64, 1, 14, 26, 13, 110, 98, 85, 75



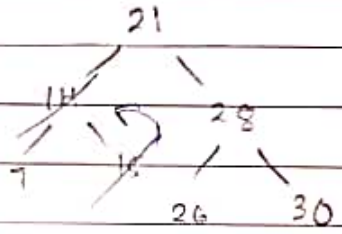


Q. 1, 2, 3, 4, 5, 6, 7, 15, 14, 13, 12, 11, 10, 9, 8.

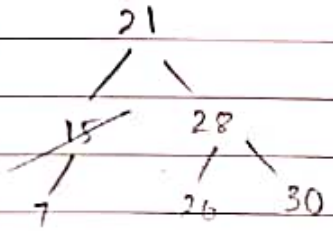
Q. 15, 25, 10, 5, 7, 3, 30, 20, 8, 15

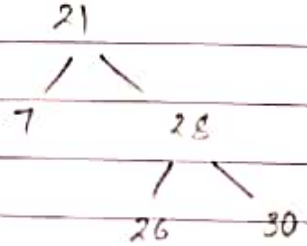


↓ RR



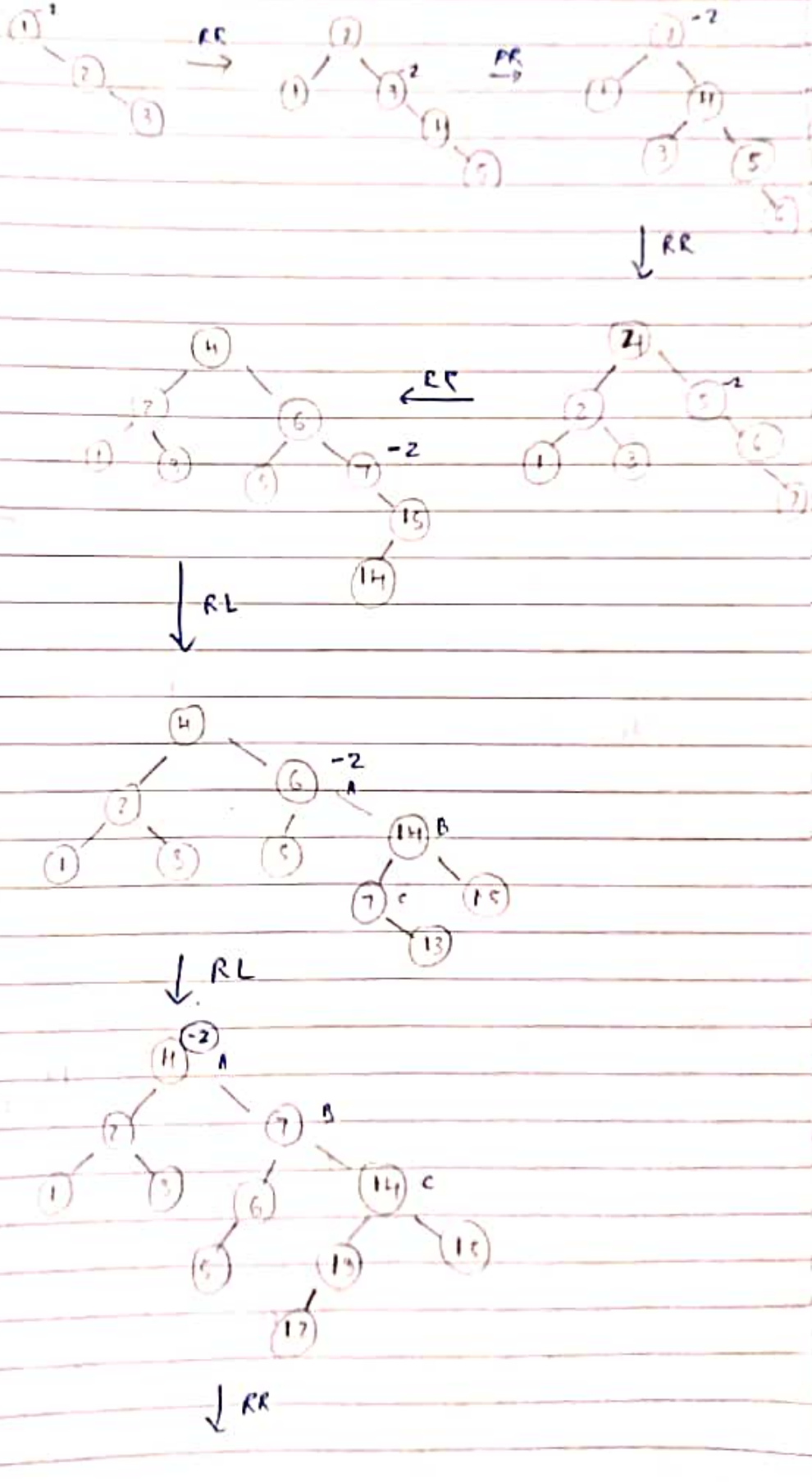
↓ Inorder successor



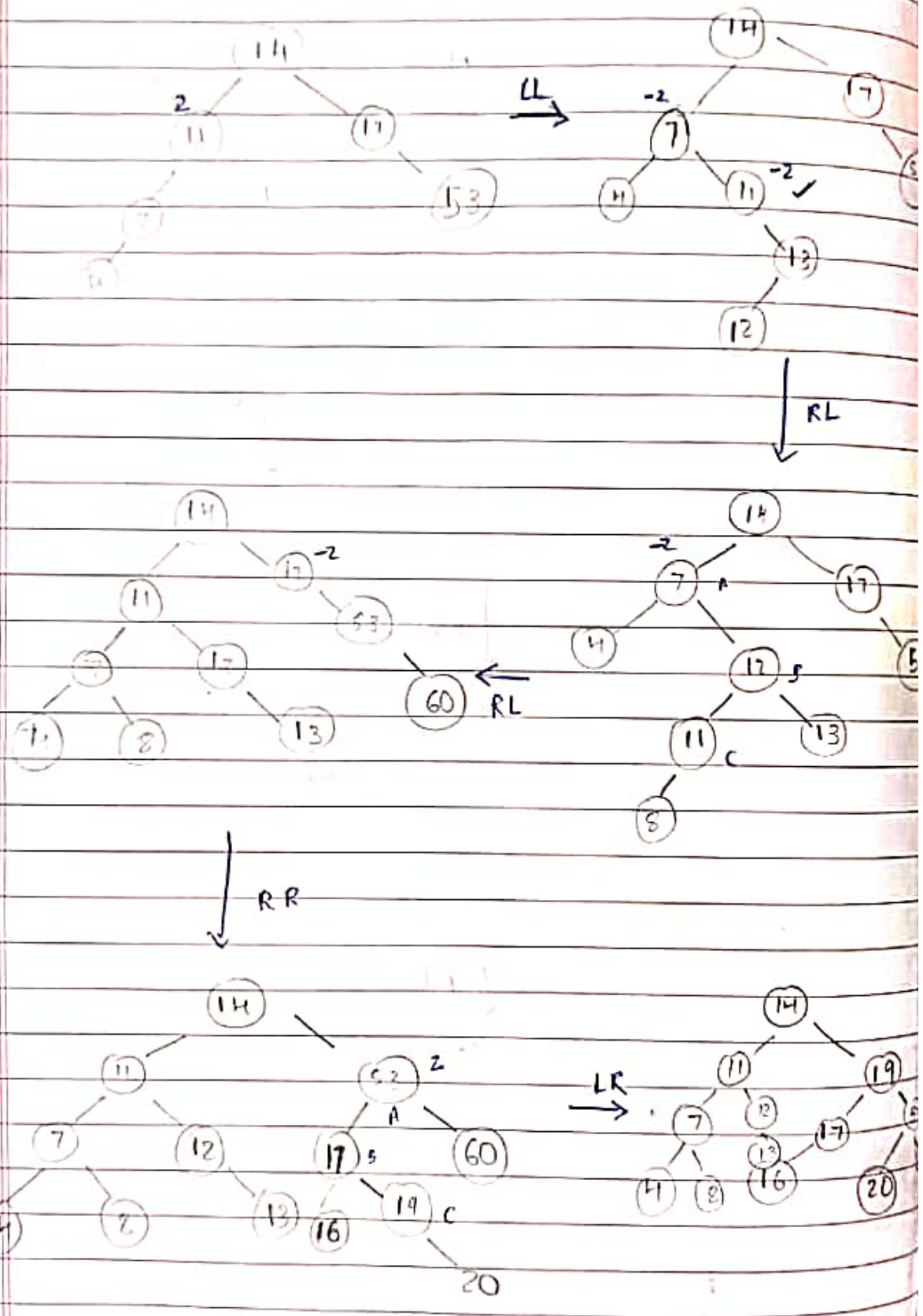


→ 1, 2, 3, 4, 5, 6, 7, 15, 14, 13, 12, 11, 10, 9, 8

→ 50, 25, 10, 5, 7, 3, 30, 20, 5, 15

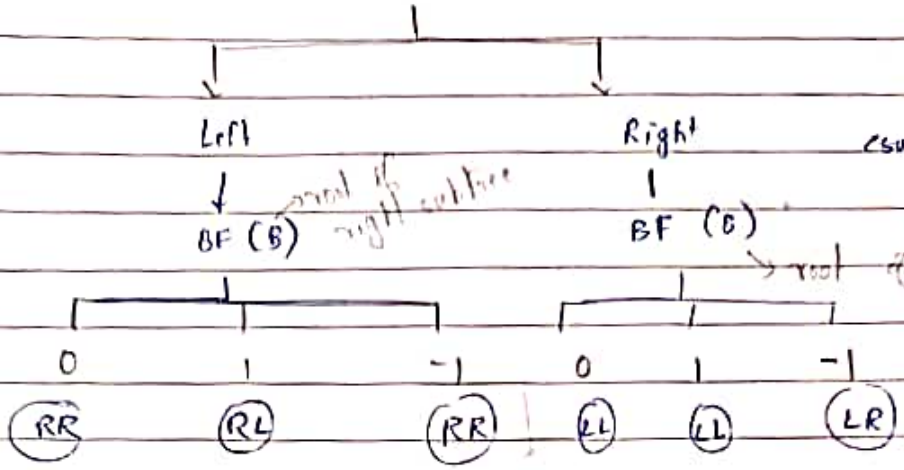


→ 14, 17, 11, 7, 53, 4, 13, 12, 8, 60, 19, 16, 20.

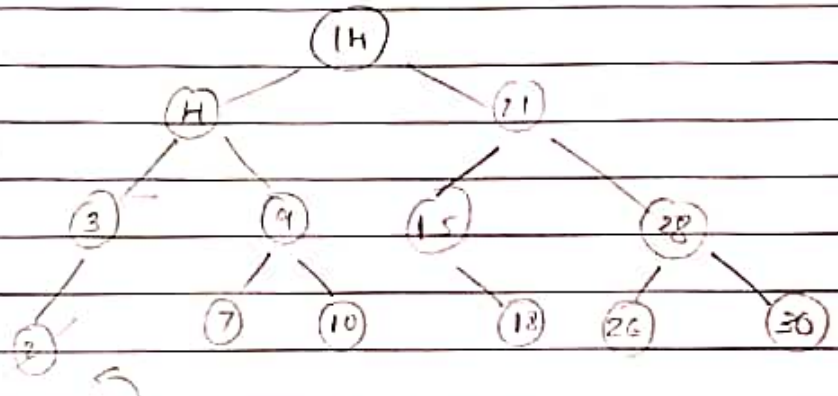


BF

→ Find : A (-2/2)

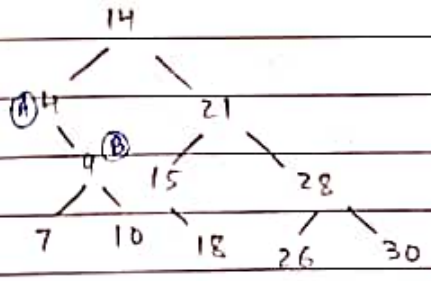


Q.

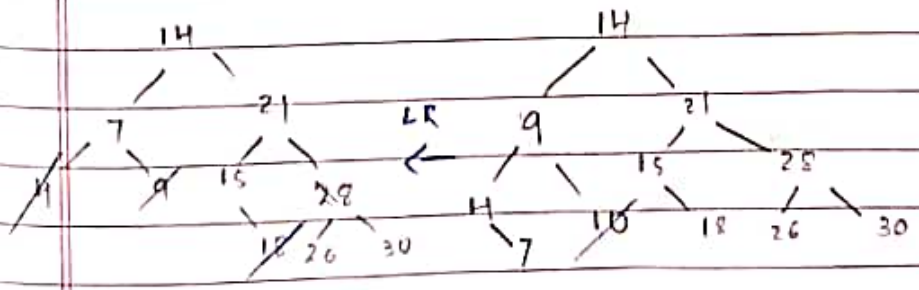


Delete → ~~2~~, ~~3~~, ~~10~~, ~~18~~, ~~4~~, ~~9~~, ~~14~~, ~~7~~, ~~15~~

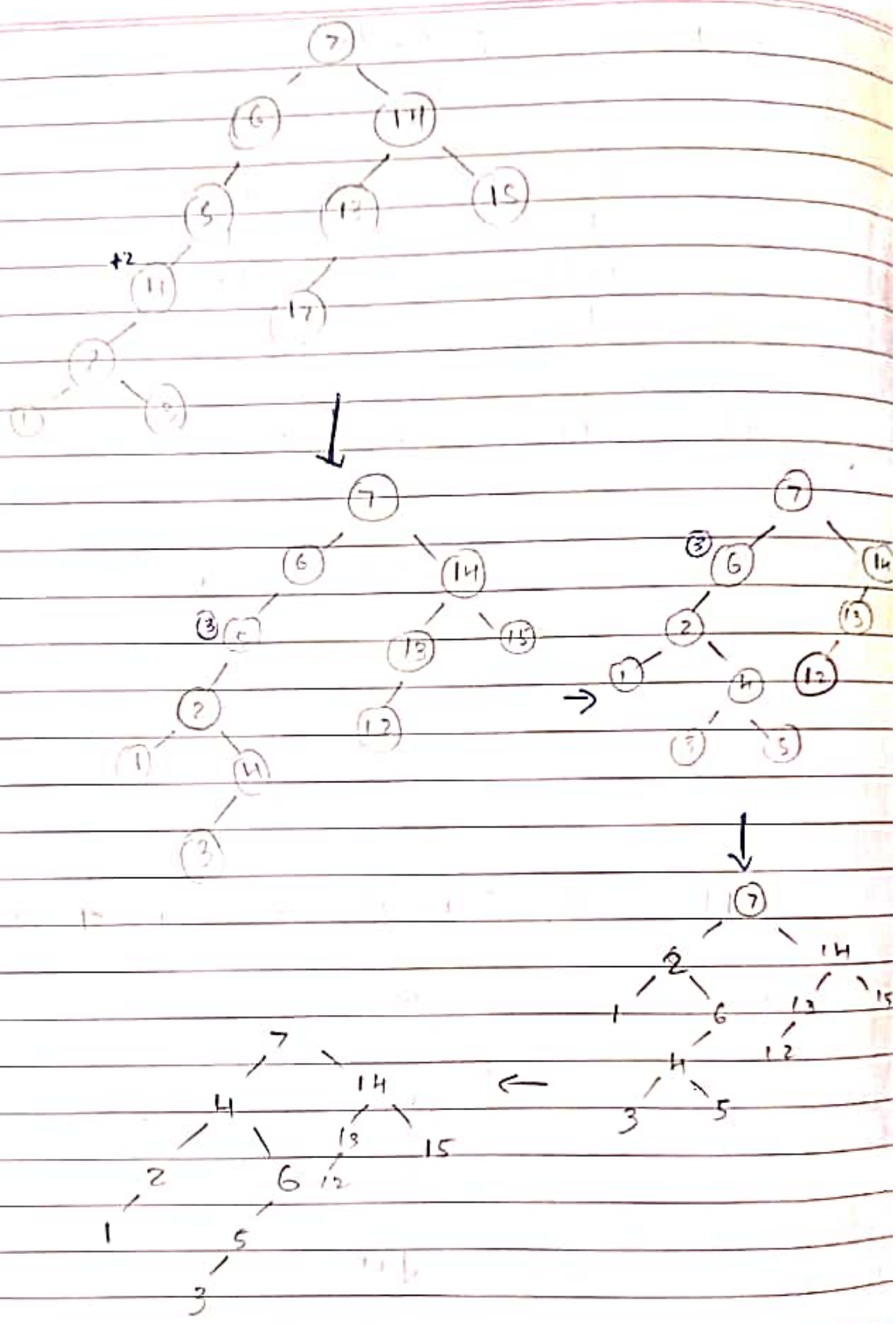
For (3)



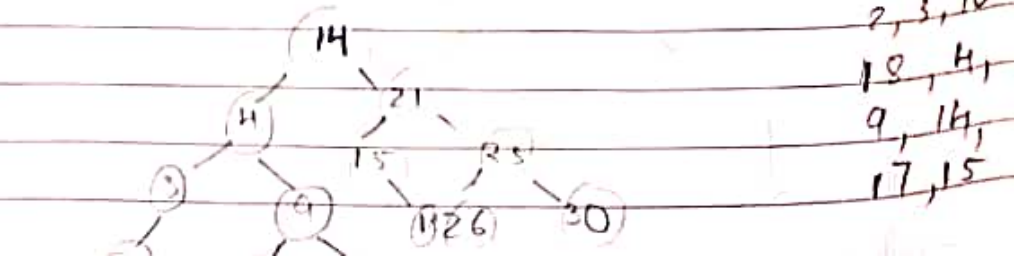
↓ RR



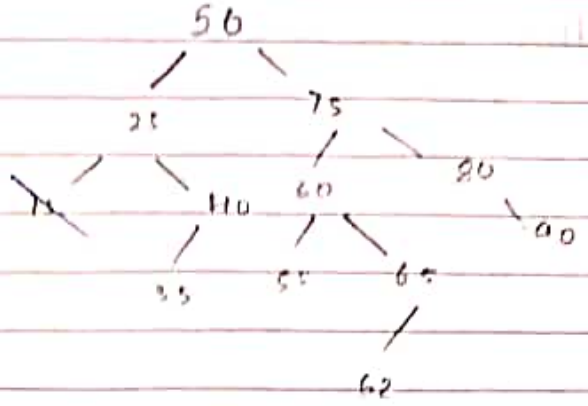
↓



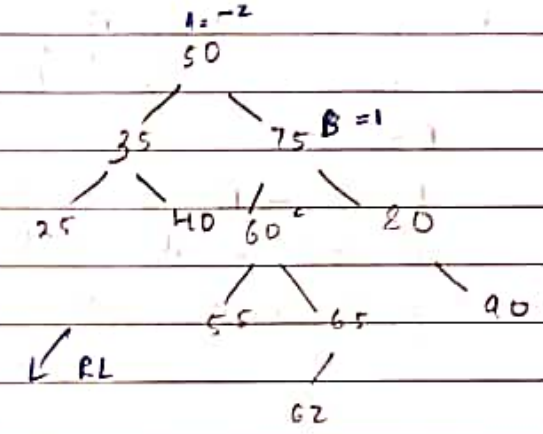
→ Deletion → use inorder successor



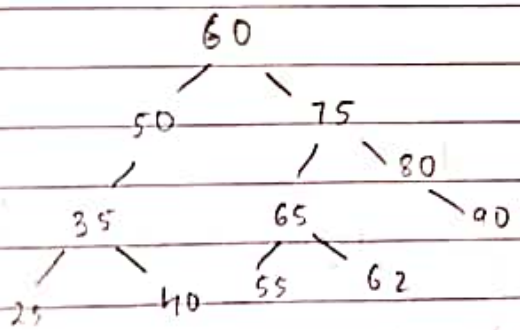
Q.


50
75
80
60
40
65
62

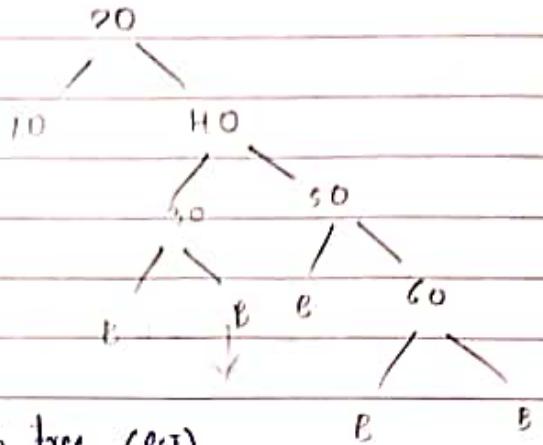
RL


50
35
75
25
40
60
20
55
65
90
62

RL


60
50
75
35
65
80
25
40
55
62
90

→ Red black tree (for insertion / deletion)



(i) Binary search tree (BST)
(ii) all nodes → colored (Black / red)
(iii) Root - B
(iv) leaves - B (external)
(v) if node - red → both descendant → Black
(vi) path from node to descendant leaves.

→ insertion

1) (i) (U) → red (assign)
if tree is empty → U → root, B

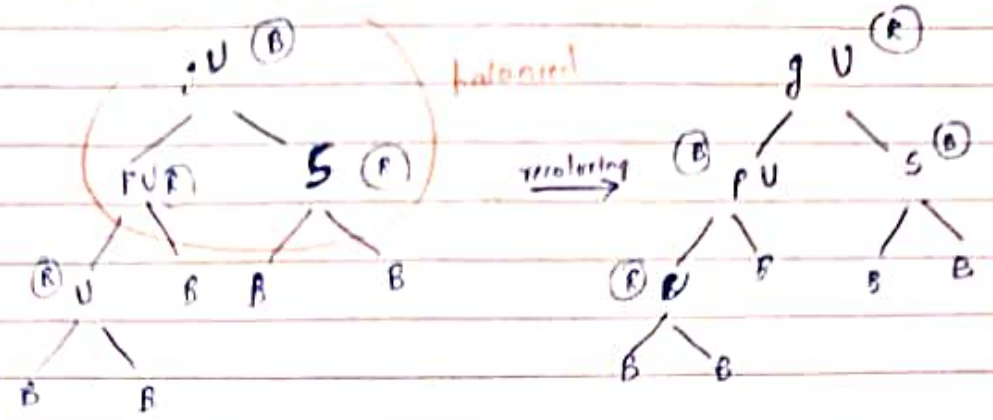
2) (ii) U → red (node)
parent (U) → pU — B / R

pU (B)
U (R) → nothing needed.

3) (iii) → pU (R) → not balanced.
U / (R)
→ gU (grand parent of U)

3·1) (iii) (a)

~~(iv)~~ gU , S(R) → only recoloring



3·1·1) gU — root

gU → B

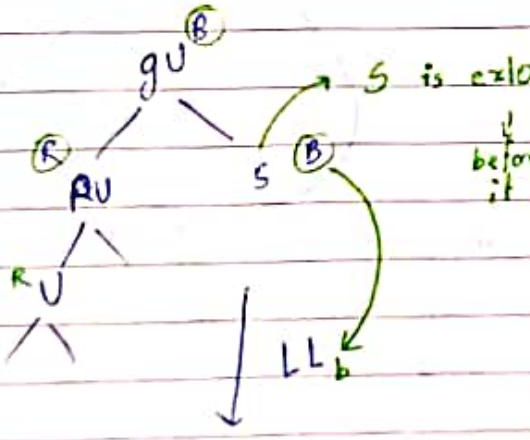
3·1·2) gU — non root

parent(h) — B completed

R → go to step 3 (where h → pU, gU → U)

(Red-Red violation)

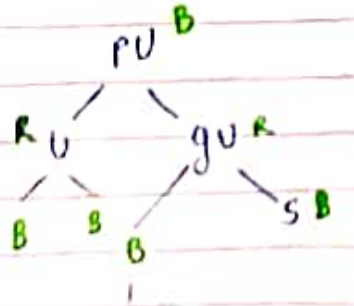
3·2)

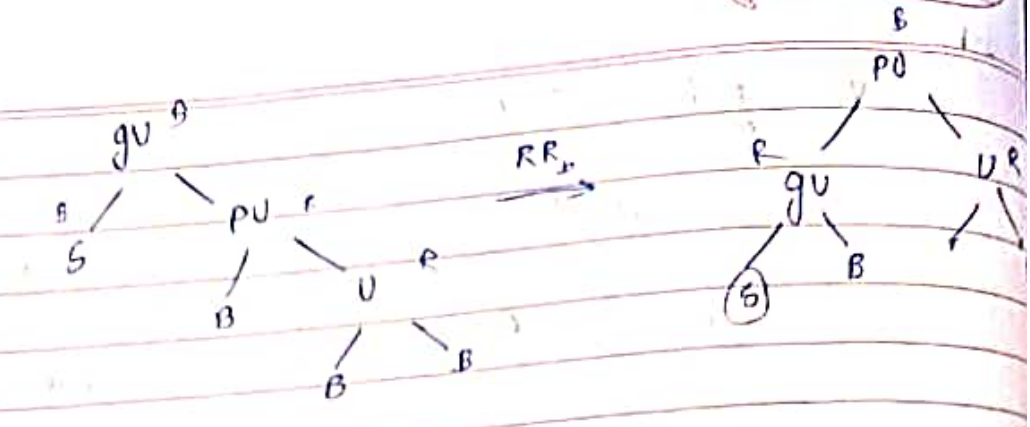


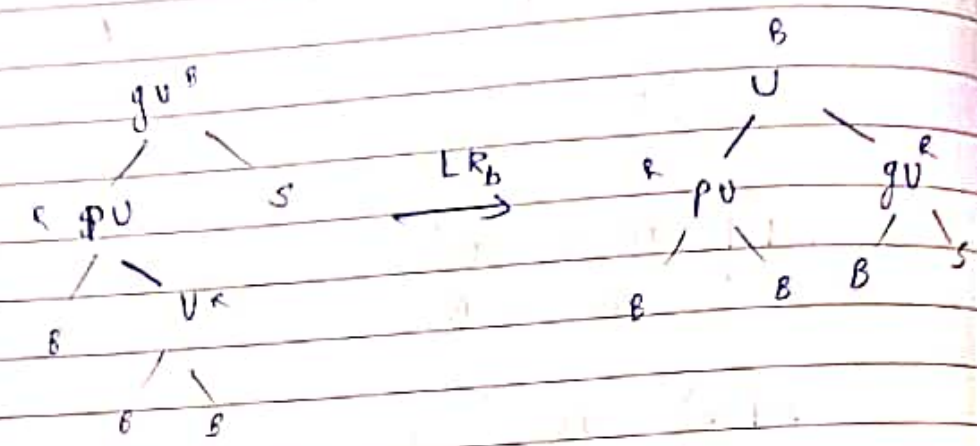
S is external

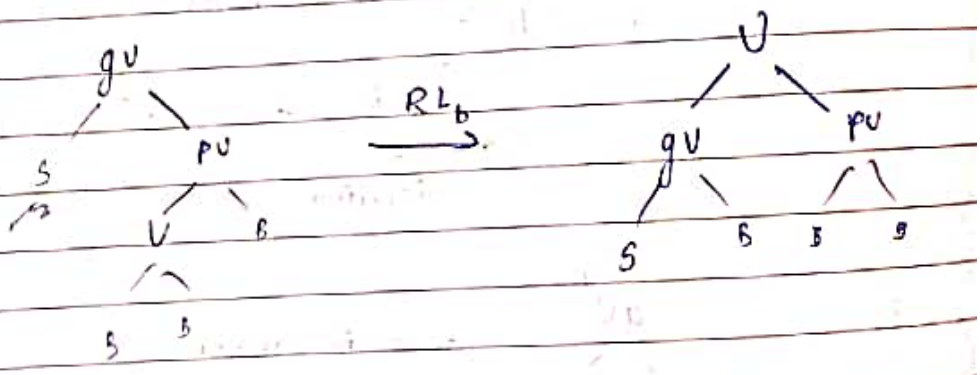
before inserting, it is already balanced

[pU — red, S → B]

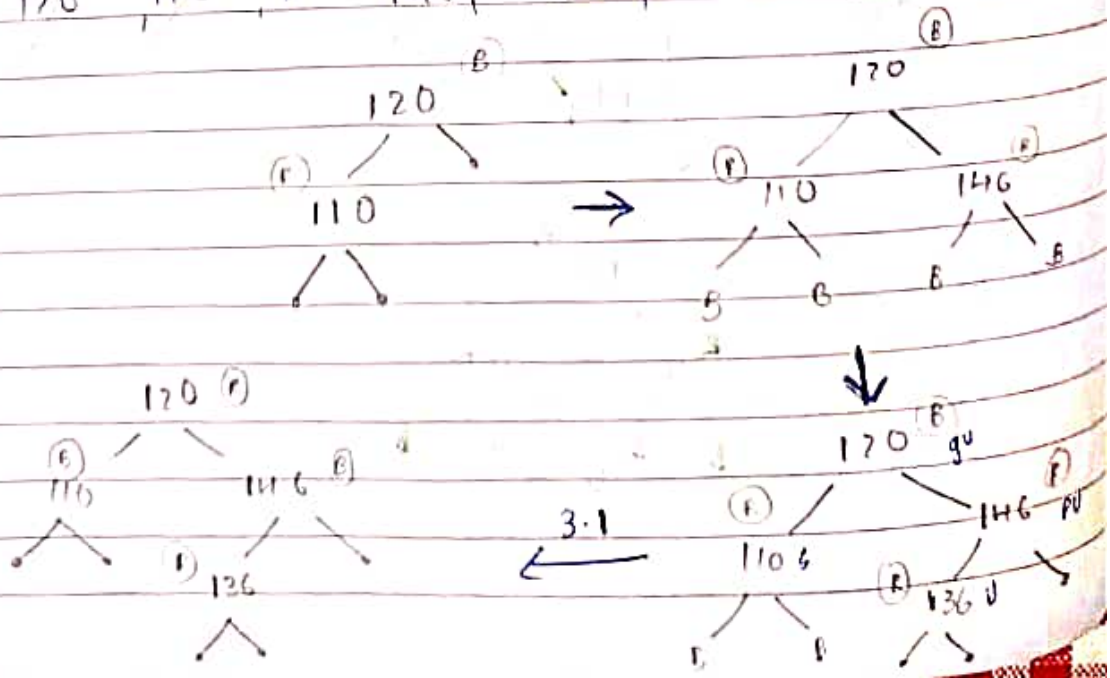


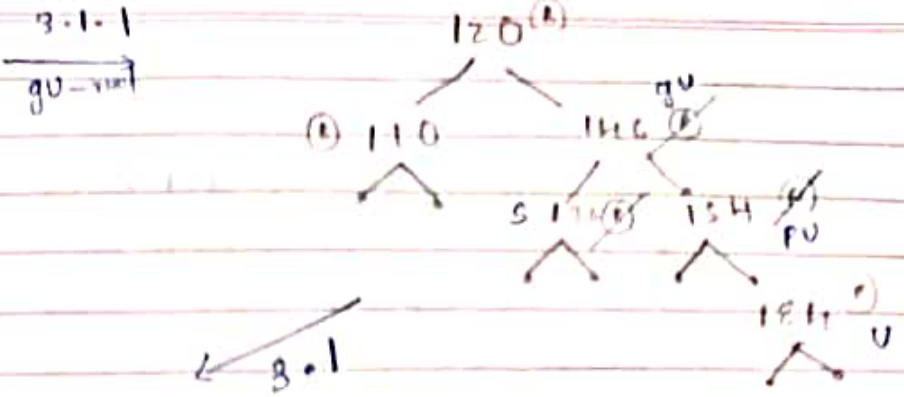


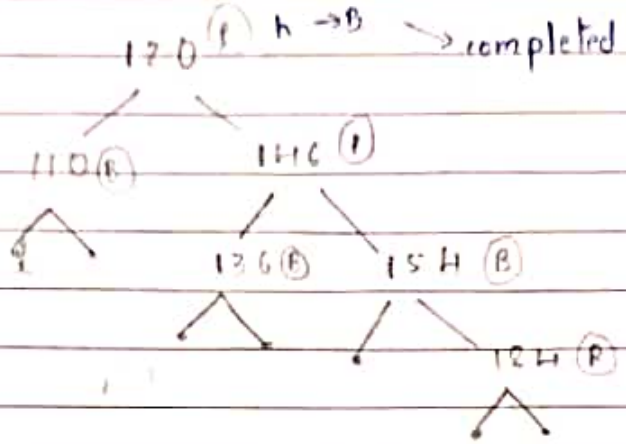




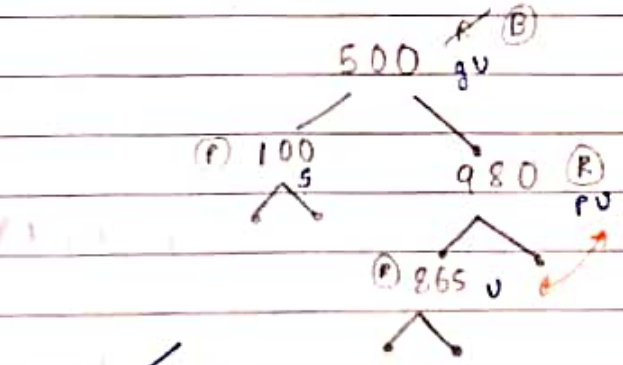
Eg: 120, 110, 14, 146, 136, 15H, 18H,

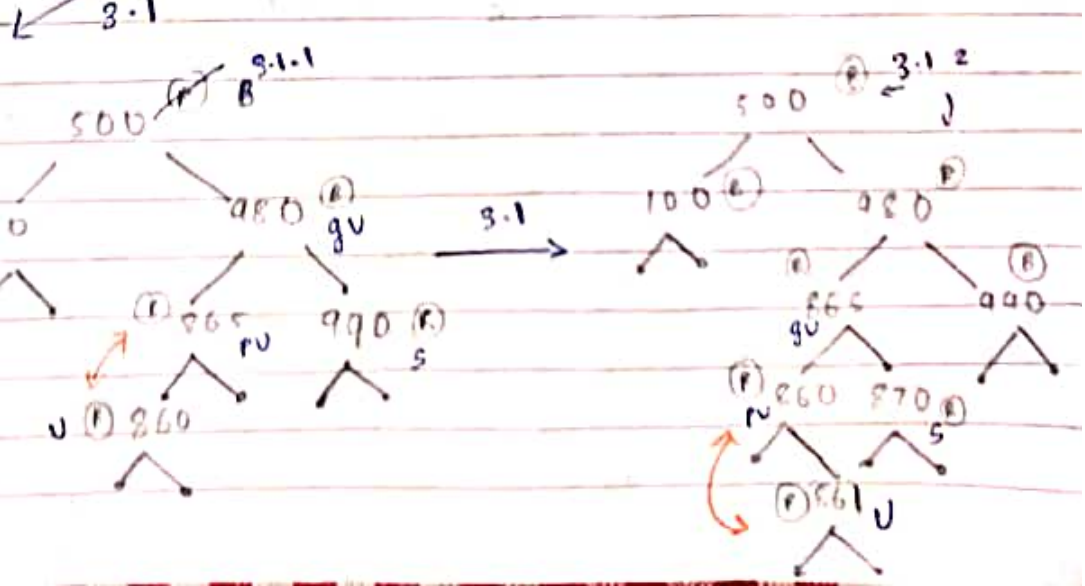


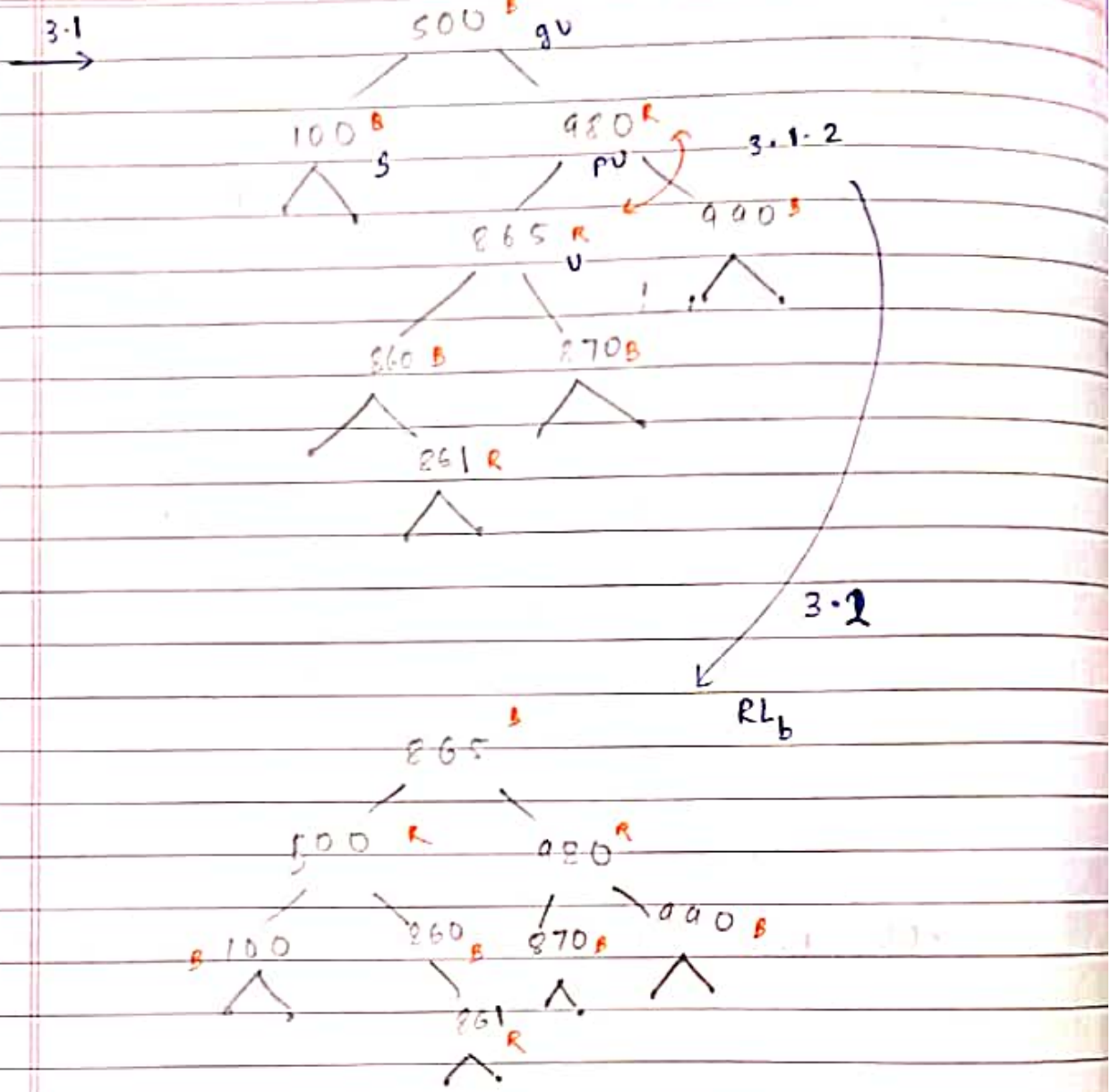




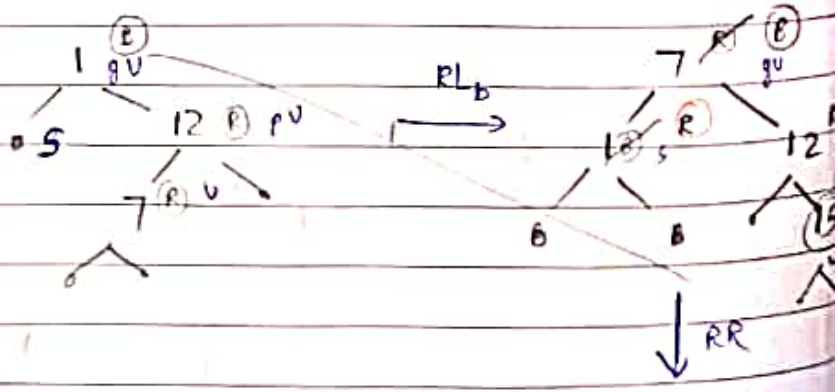
Q. 500, 100, 980, 865, 990, 860, 870, 861

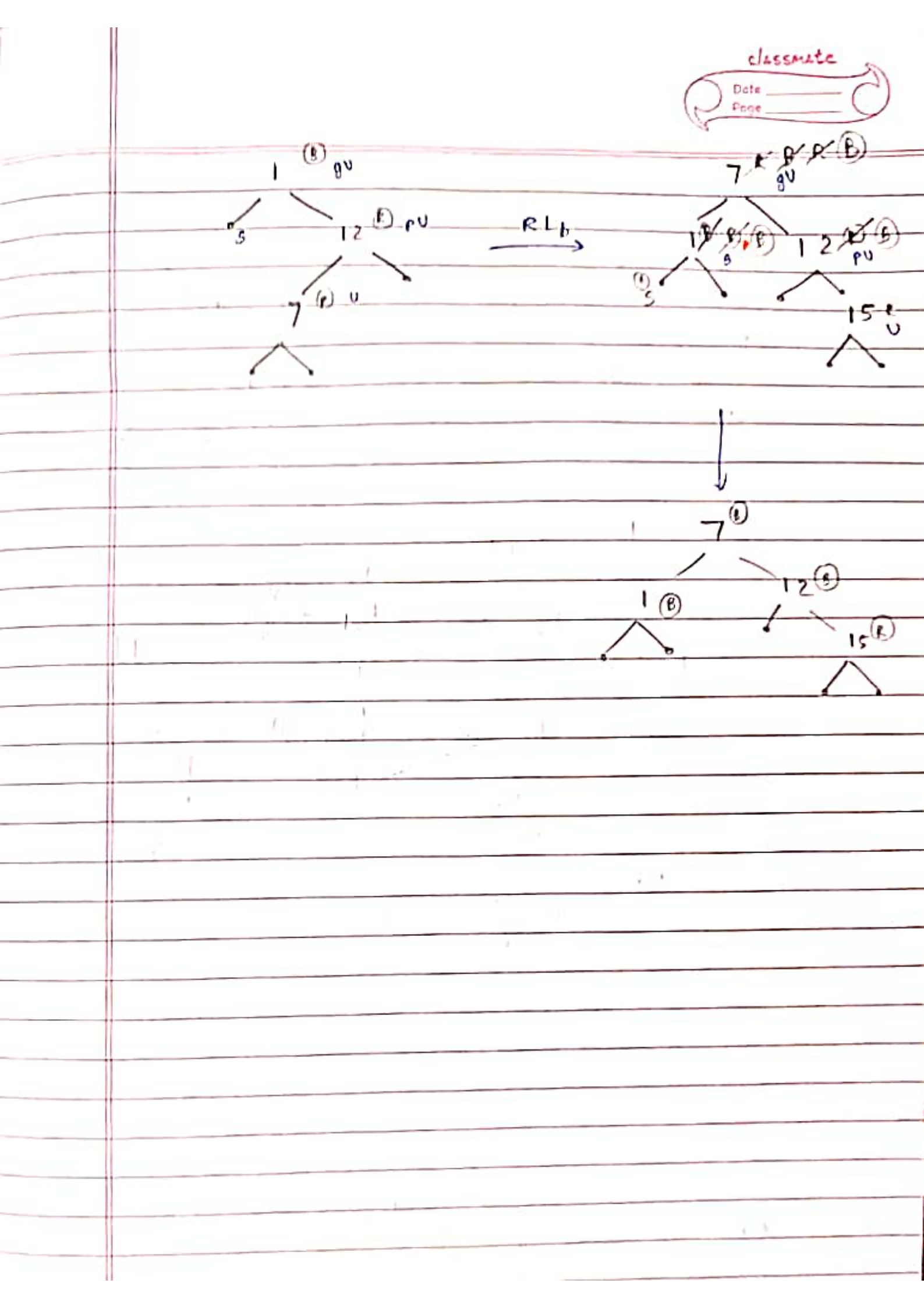






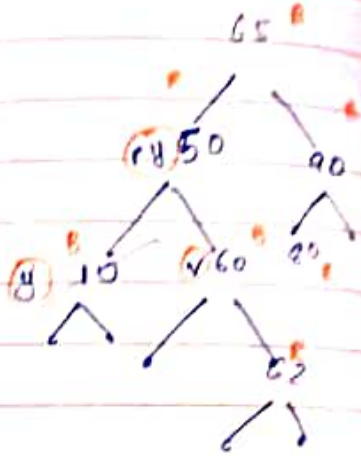
→ 1, 12, 7, 15, 18, 9, 20, 8, 13



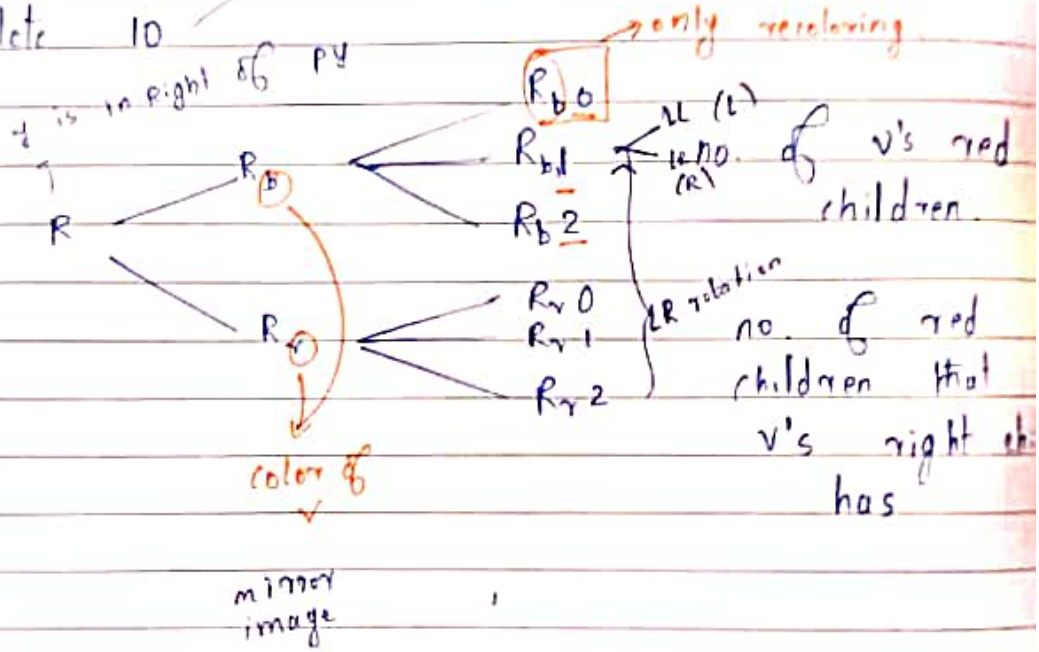

classmate
Date
Page
RL
gu
pu
u
s

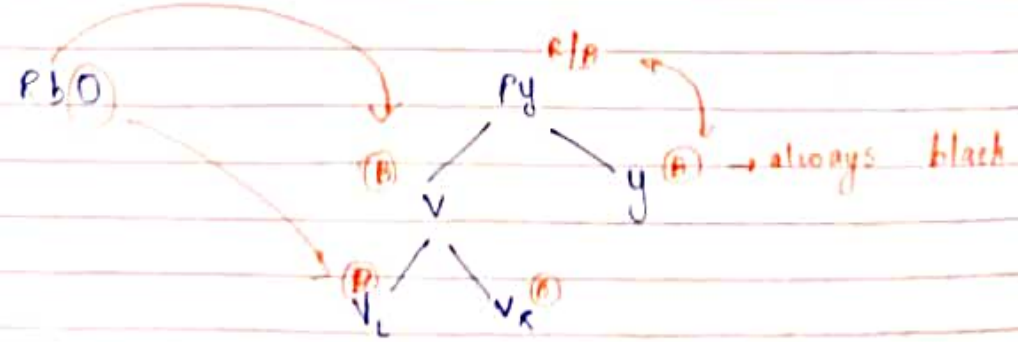
→ Deletion

if physically deleted node is red, no rotation or color change.



→ delete 10





at Pb0 → it has to be imbalanced.

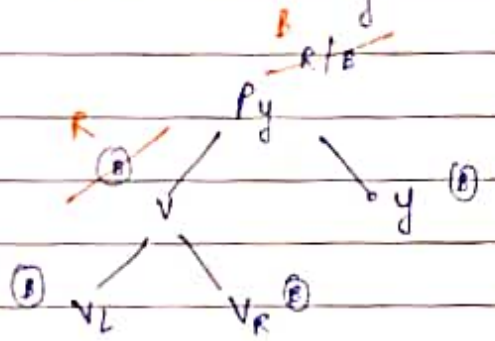
compare Left and right tree, we have one less black in right.

if y was internal → there is no imbalance

but we need imbalance

⇒ y is external

final coloring



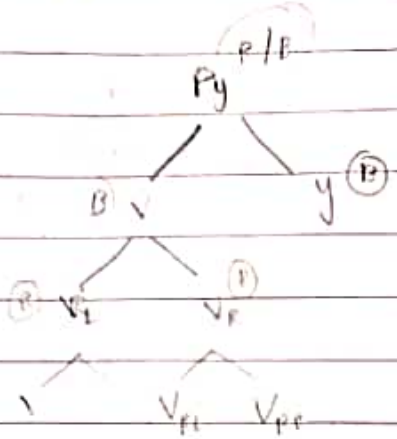
→ for Rb1 → if red child in { Left → LL rotate, Right → LR }

→ for Rb2

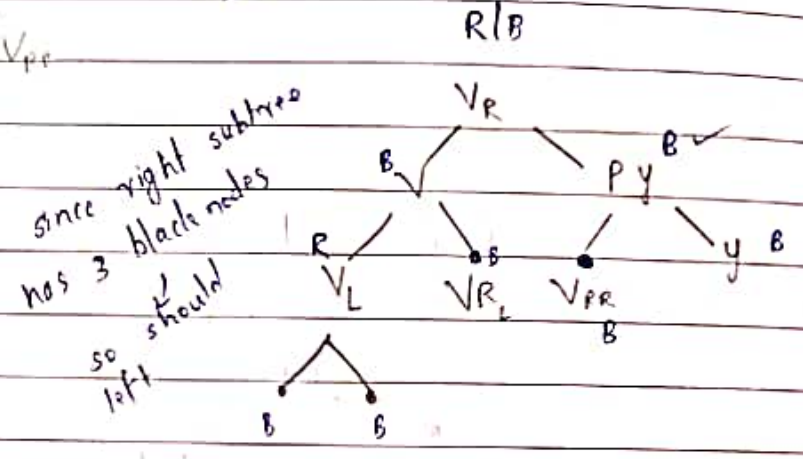
Case 1:

Case 2:

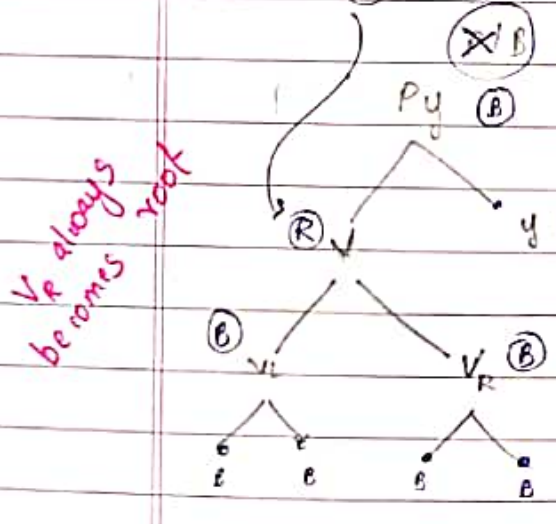
→ for Rb2



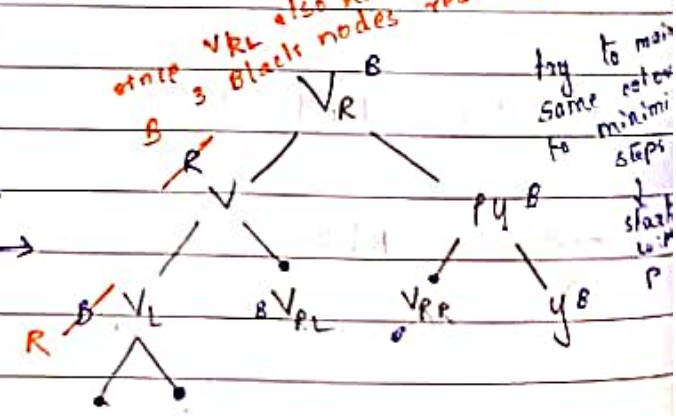
LR rotation



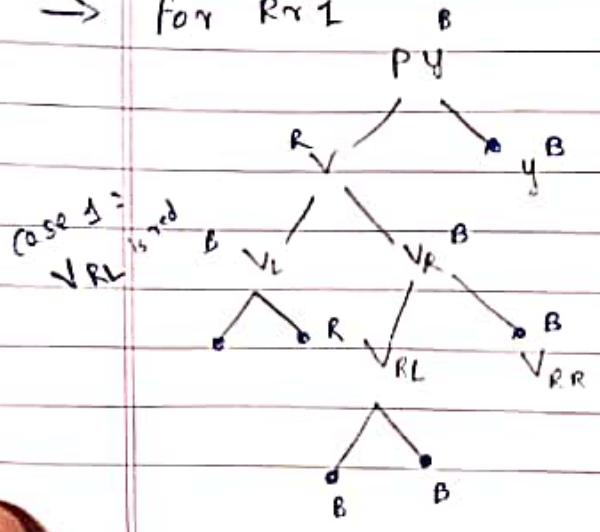
→ for RrO



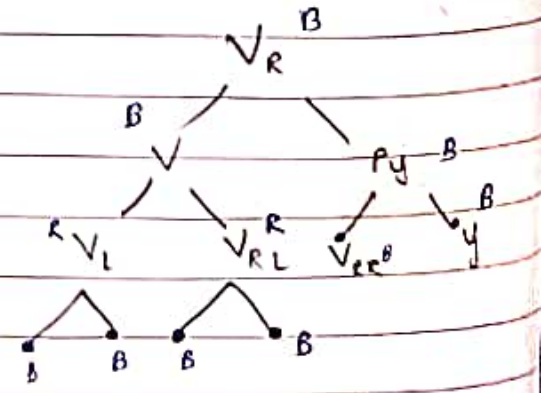
LR →

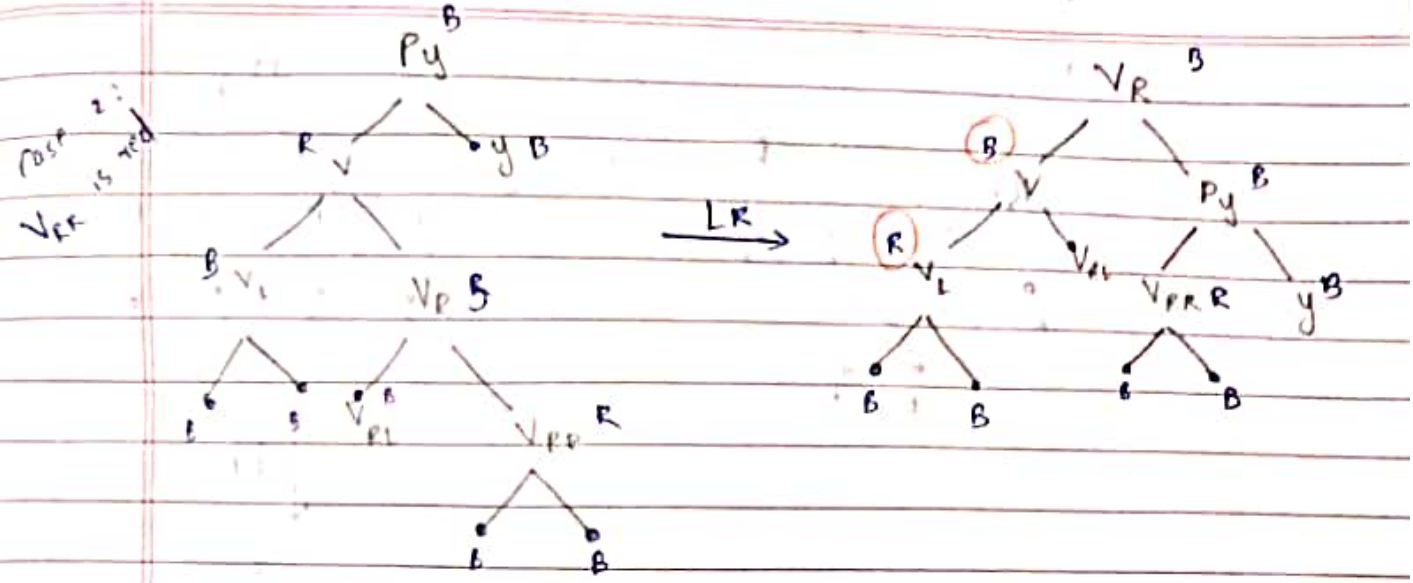


→ for Rr1

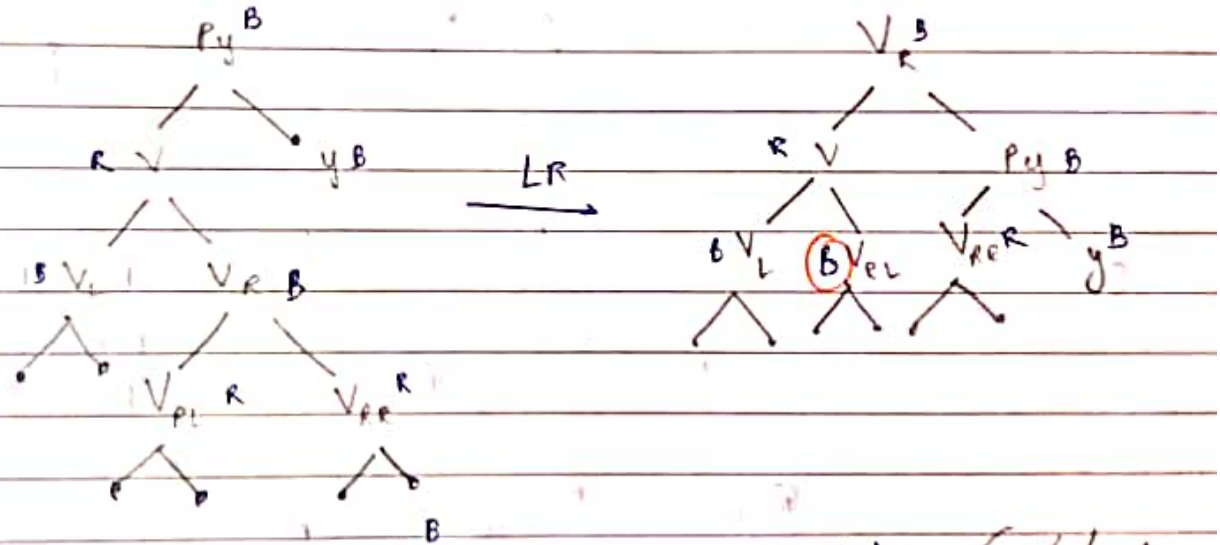


LR →



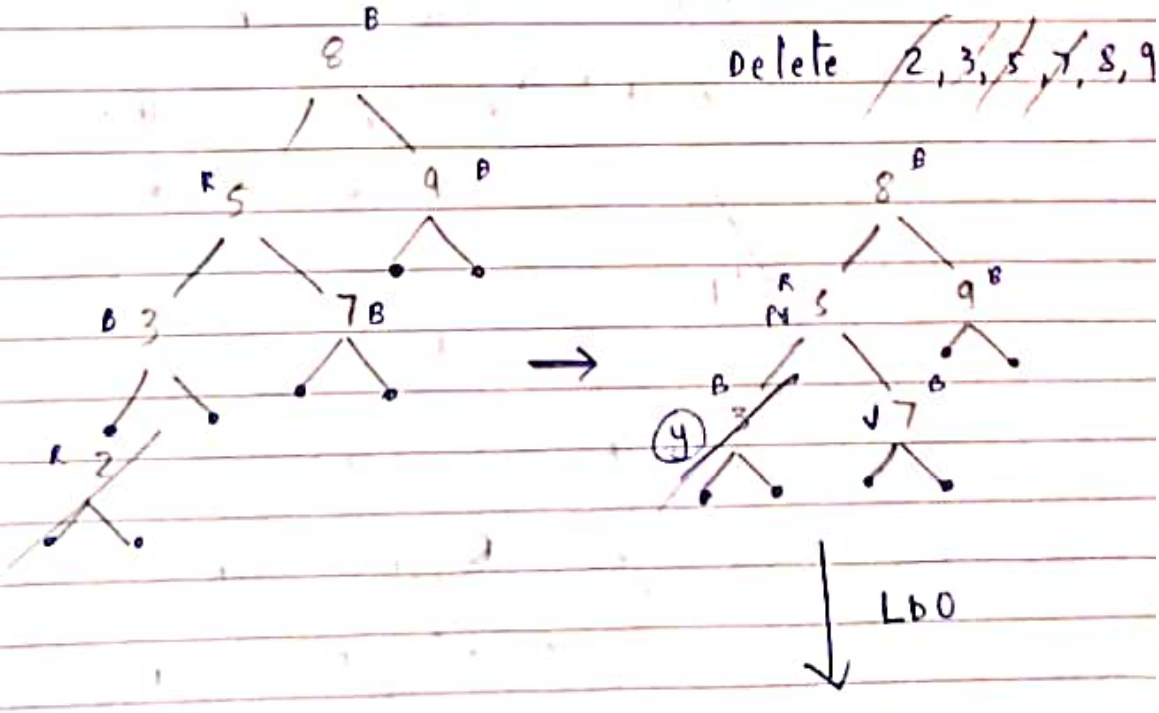


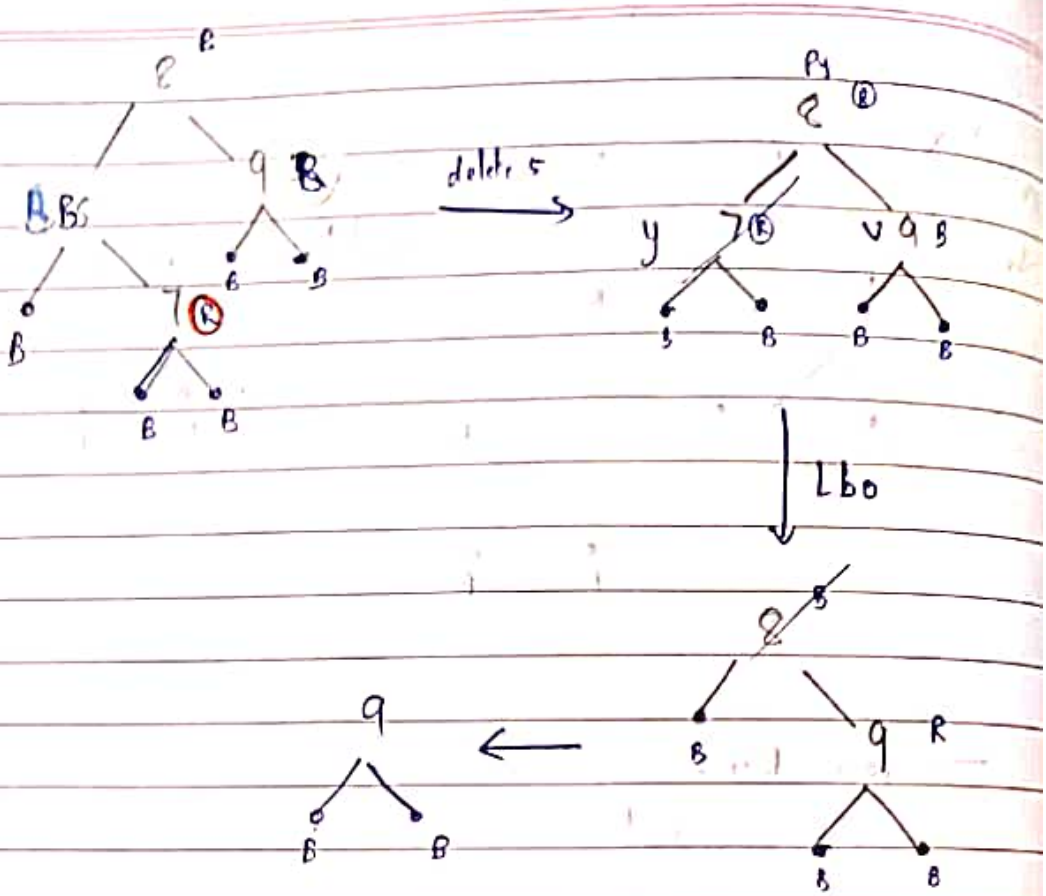
→ for Rn 2



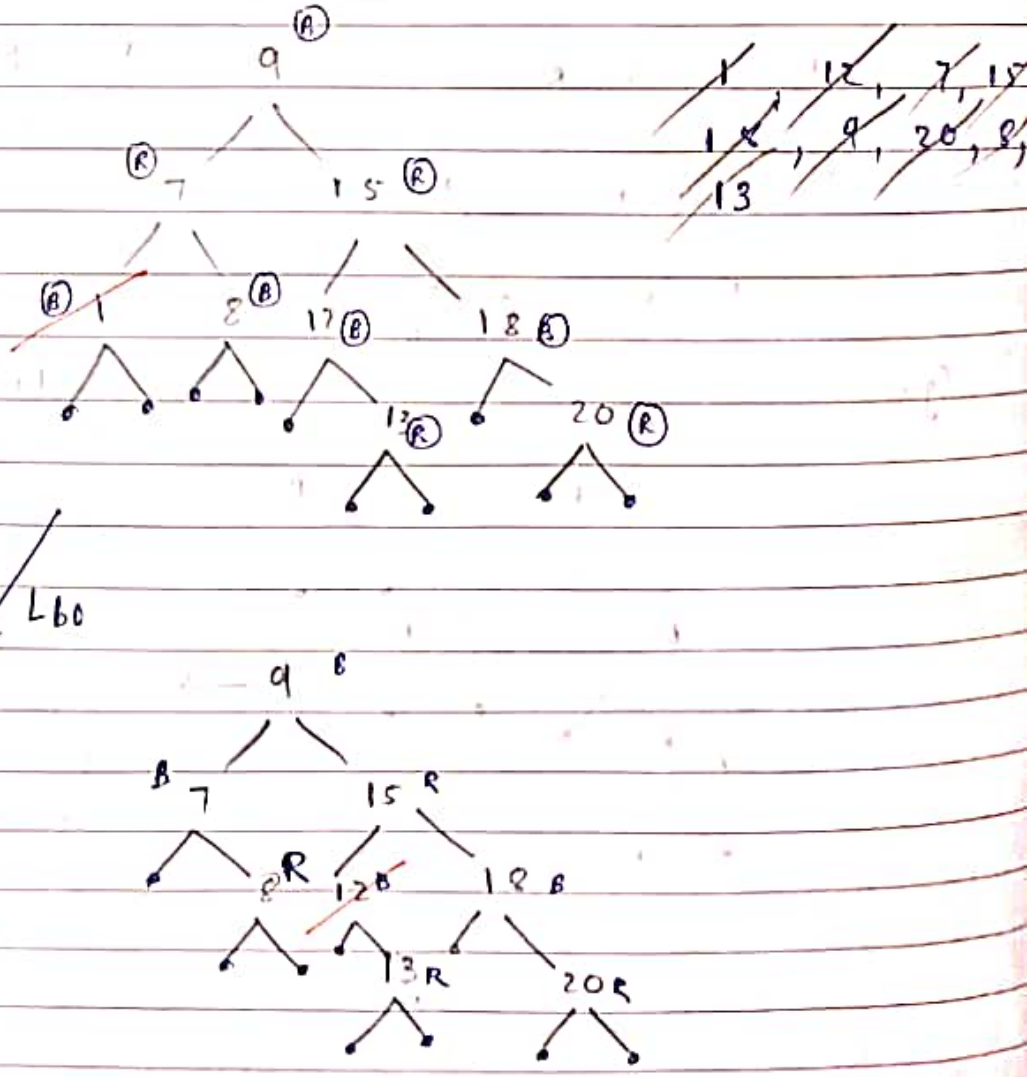
Eg:

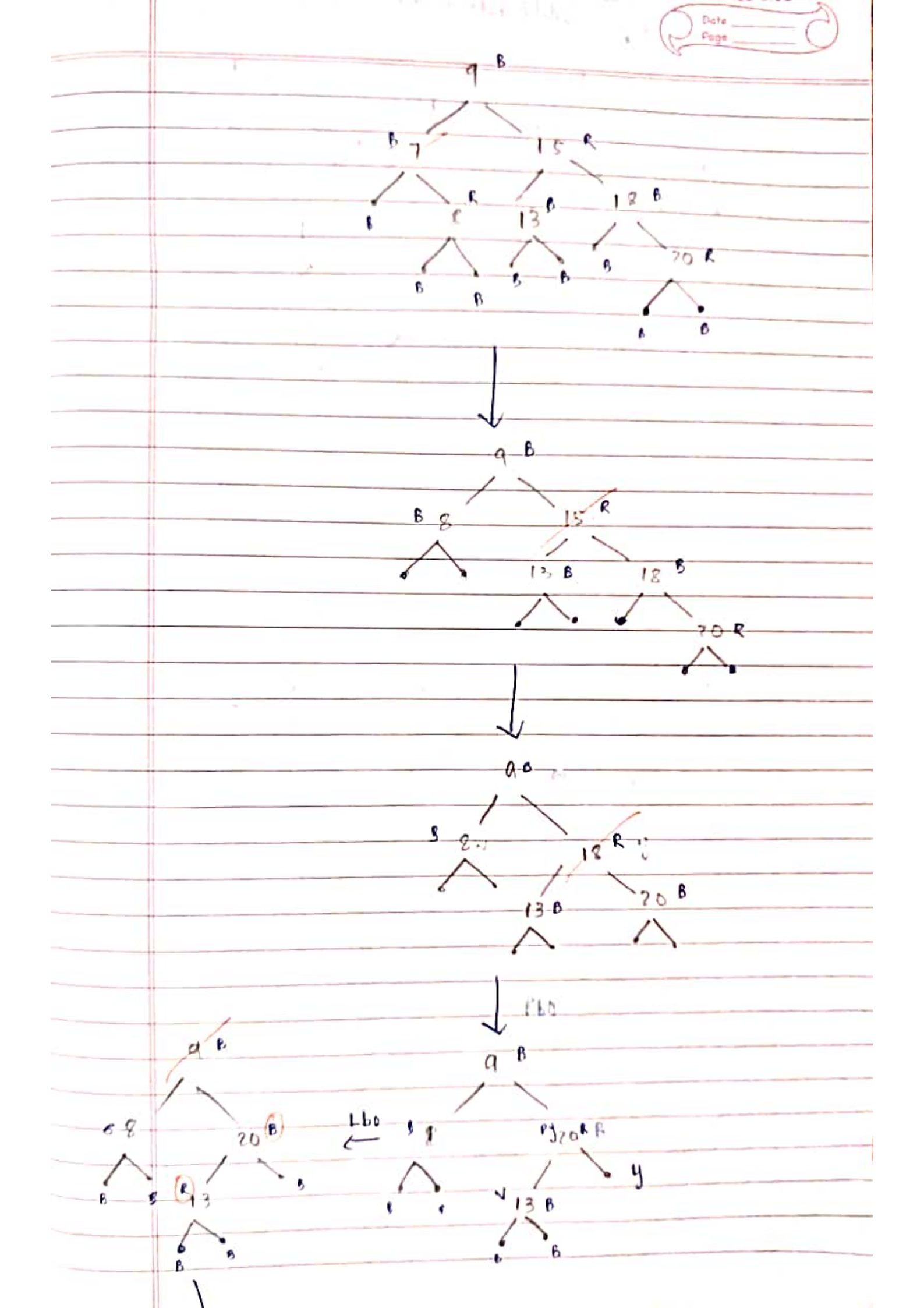
Delete 2, 3, 5, 7, 8, 9



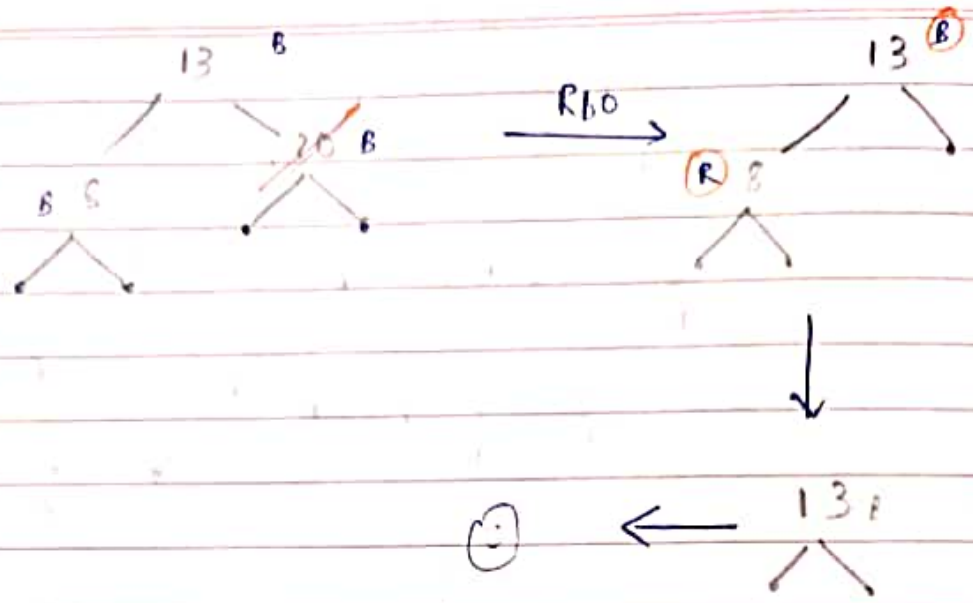


Q.



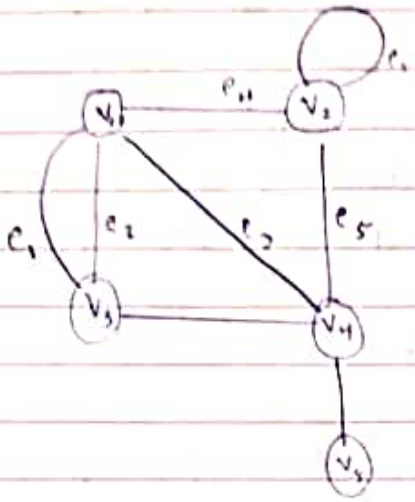

Date
Page
9 B
7 B
15 R
8 R
13 B
18 B
20 R
9 B
8 B
15 R
13 B
18 B
20 R
9 B
8 B
18 R
13 B
20 B
LLO
9 B
8 B
20 B
13 R
LLO
9 B
y
13 B

nothing → as physically deleted node is red.



GRAPHS

$G = (V, E)$, $V(G)$, $E(G)$



pendant — degree
connected — undirected
self loop — Path
11^{th} edges — circuit
weighted — Adjacent vertices edges

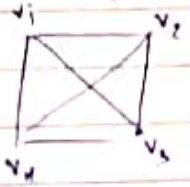
v_1 → incident edges → e_1, e_2, e_3, e_{11}

General graph – 11^{th} edge, self loop allowed
Simple " – " " " not "

walk, path, cycle → beginning
↓ → Don't repeat vertex.
edges should not be repeated (but vertex can be repeated)

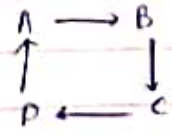
→ Complete graph → all vertices are adjacent.

Regular graph → every vertex has same degree.



(3) - regular graphs.
↓
degree.

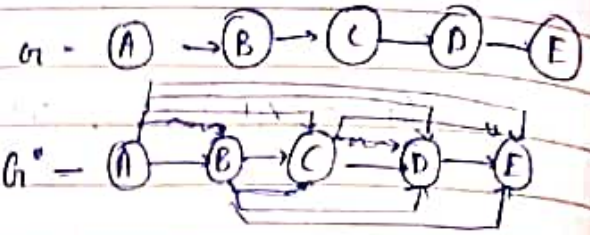
→ Digraph → directed graph.



→ Transitive closure

$G = (V, E)$

$G = (V, E^*)$



E is reachable from all vertices.

if path exists in b/w two nodes, add edges if not present directly.

→ Representation of graph

Adjacency matrix (Parallel edges are not allowed)

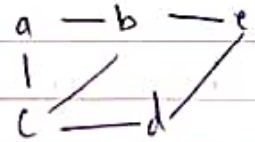
Boolean matrix

	A	B	C	D	E
A	0	1	0	0	0
B	0	0	1	0	0
C	0	0	0	1	0
D	0	0	0	0	1
E	0	0	0	0	0

- For G

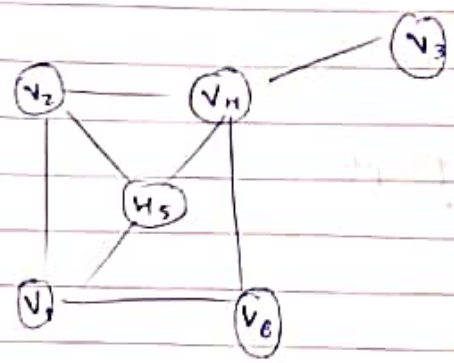
E → not possible to reach any vertex from E

Eg:



	a	b	c	d	e
a	0	1	1	0	0
b	1	0	1	0	1
c	1	1	0	1	0
d	0	0	1	0	1
e	0	1	0	1	0

Eg:



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

= X

from v_1 to v_1 3 different path of length 2

$$X^{(2)} = \begin{bmatrix} (3) & 1 & 0 & 3 & 1 & 0 \\ 1 & 3 & 1 & 1 & 2 & 2 \\ 0 & 1 & 1 & 0 & 1 & 1 \\ 3 & 1 & 0 & 4 & 1 & 0 \\ 1 & 2 & 1 & 1 & 3 & 2 \\ 0 & 2 & 1 & 0 & 2 & 2 \end{bmatrix}$$

$$B = X + X^2 + X^3 + X^4 + \ldots + X^6$$

Path matrix = Boolean matrix.

if in $B[i] \neq 0$, $P[i] = 1$

else $P[i] = 0$.

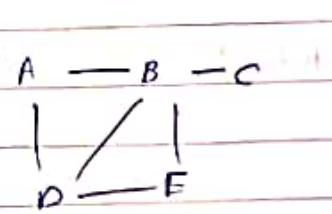
→ if general graph → no adjacency matrix → just matrix

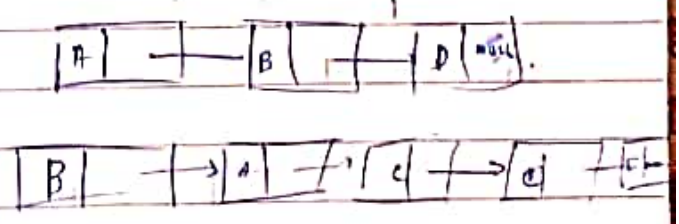
val = no. of edges b/w the nodes.

Degree = sum of (2x Diagonal element + other element)

if directed → add all in row and column for degree.

(ii) Adjacency list





no. of lines in list. = 2 x edges

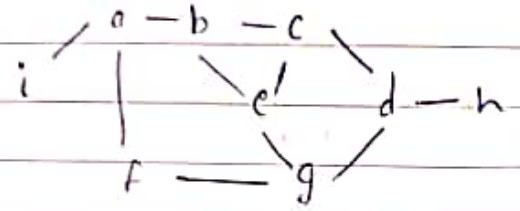
→ BFS (Queue)

→ DFS (stack)

→ Backtracking.

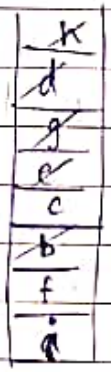
Decision making.

Q.

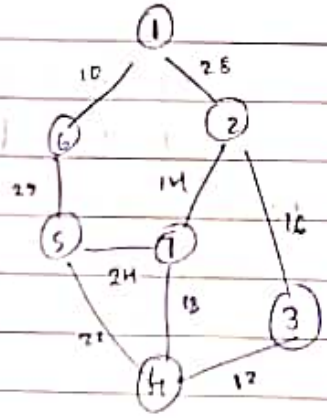


BFS: a i f b g c e d h

DFS: a b e g d h c f i



Q.



Shortest path in unweighted graph b/w 1 and 4

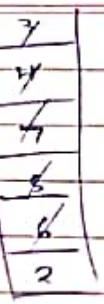
1 6 2 5 7 3 4

(parent) sol: – 1 1 6 2 2 5 → took 4 as child of 5 first.

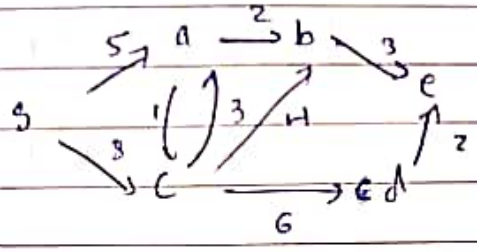
path : 1 → 6 → 5 → 4

DFS : 1 6 5 4 3 7 2

- 1 6 5 4 5 1

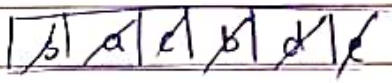


Q.



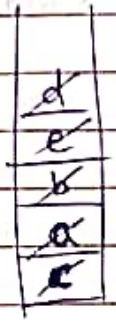
BFS : s a c b d e

- s s a c d



DFS : s a b e c d

- s a b s c

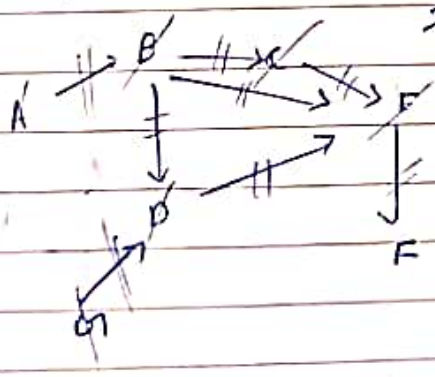


s → d

↳ s c d DFS

BFS → shortest path

→ Topological sorting → DAG (Directed Acyclic graph)



add 10 in degree vertices.

Q : A G B C D E F

A : G B C D E

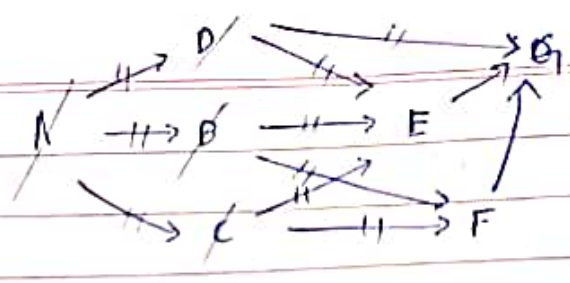
↓

remove vertex from graph (and its edges)

→ Application

(i) Linear ordering of vertices

↳ ~~four~~ vertices comes first before remaining vertices to which it has outgoing edges.

Q. 

A	D	B	C	E	F	G

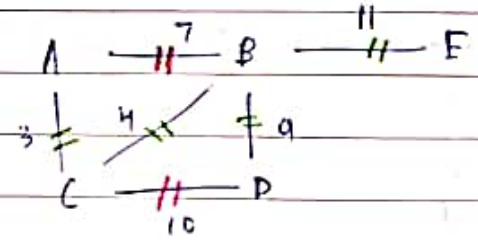
A D B C E F G

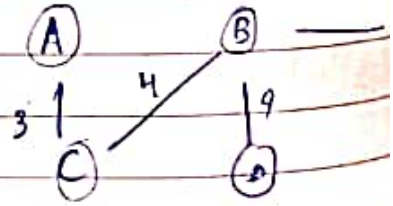
A B C D E F G

(1) Minimum Spanning Tree.

G
→ spanning tree → sub graph
→ Tree
→ all vertices connected (path exists)
min no. of edges.
→ if weight → min weight
→ directed or not doesn't matter.

(i) Prims Algorithm.



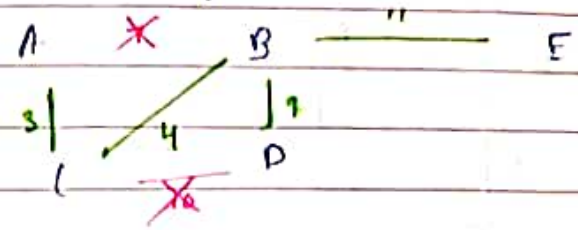


→ starting with A
→ then min weight next vertex
→ if vertex + edge in tree gives cycle, go for alternative edges.

we only consider local minimum
↓
Finally at end global optimal solution

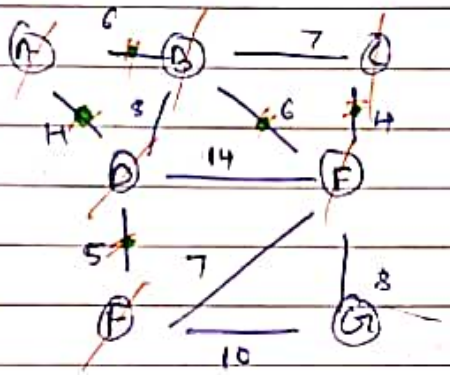
■ → including makes a cycle
■ → edge in spanning tree

(ii) Kruskal's Algorithm.

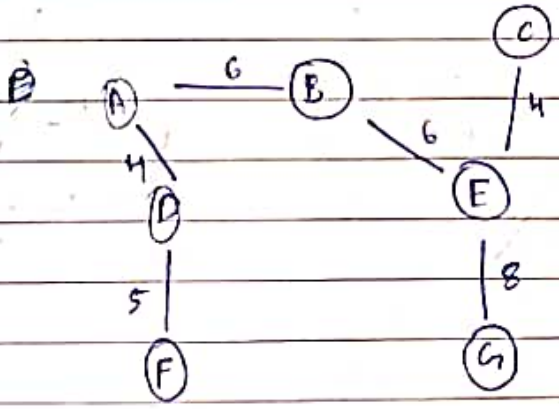


→ all vertices in spanning tree.
→ check min overall edge.
→ ignore cycle.

Q.

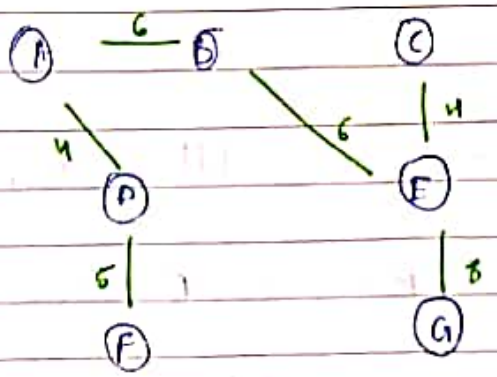


single spanning tree is not possible as same cost for different edges.



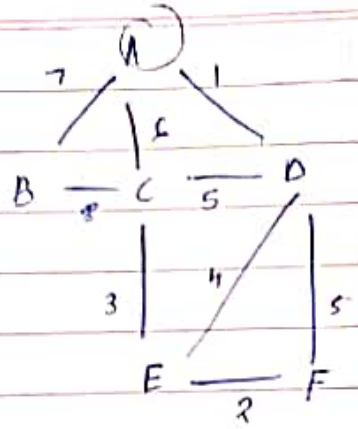
Prims

cost = 33
same for both trees (trees might be diff)



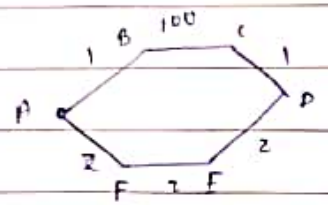
Kruskal

Q.



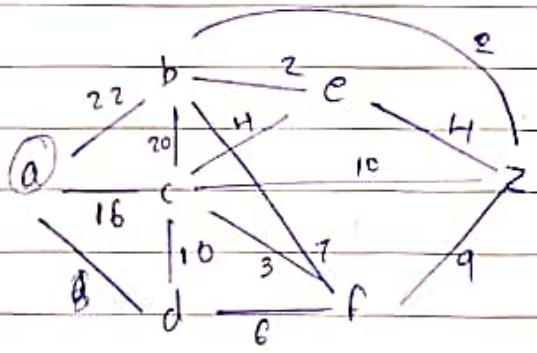
Q ✱ what will be the procedure for directed?

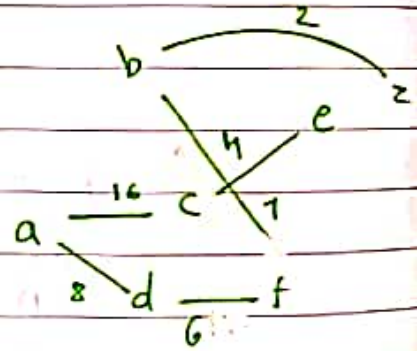
Q.



greedy approach fails here.

→ Dijkstra's Algorithm.





$P = \{a\}$ $\quad T = \{b, c, d, e, f, z\}$

$dist(a) = 0$ $\quad \infty$

Step 1:

$dist(b) = \min\{olddist(b), dist(a) + w(a,b)\}$

$= \min\{\infty, 0 + 22\} = 22$

$dist(c) = \min\{old\ dist(c), dist(a) + w(a,c)\}$

$= \min\{\infty, 0 + 16\} = 16$

draw edge b/w → From current state

dist (d) = min { ∞, 0+8 } = 8

dist (e) = min { ∞, 0+∞ } = ∞
(no direct edge)

(f) = ∞

(z) = ∞

Step 2:

P = { a, d } T = { b, c, e, f, z }

dist (b) = min { olddist (b), dist(d) + w(b,d) }
= min { 22, 8 + ∞ } = 22

dist (c) = min { 16, 8 + 10 } = 16 taking dist from prev dist ↓ go to previous state

dist (e) = min { ∞, 8+∞ } = ∞

(f) = min { ∞, 8+6 } = 14

(z) = ∞

Step 3:

P = { a, d, f } T = { b, c, e, z }

dist (b) = { 22, 14+7 } = 21

(c) = { 16, 14+3 } = 16 taking from prev dist ↓ go to previous state

(e) = { 100, 14+∞ } = ∞

(z) = { ∞, 14+9 } = 23

Step 4:

P = { a, d, f, c } T = { b, e, z }

dist (b) = { 21, 16+20 } = 21 go prev

(e) = { ∞, 16+4 } = 20

(z) = { 23, 16+10 } = 23

Step 5 :

$P = \{a, d, f, c, e\}$ $\quad T = \{b, z\}$

dist (b) = $\{21, 20 + 2\} = 21$ (go previous state)

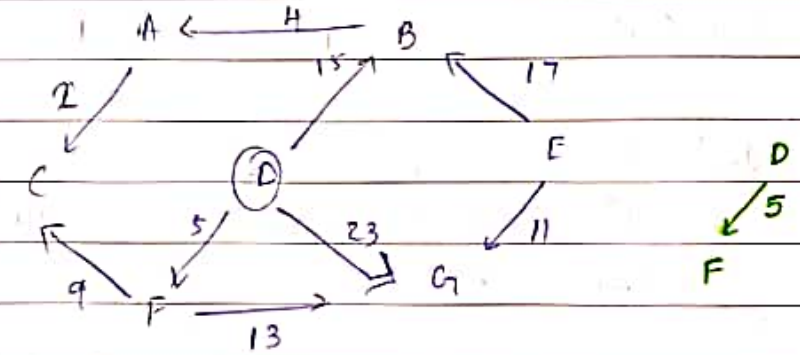
(z) = $\{23, 20 + 4\} = 23$

Step 6 :

$P = \{a, d, f, c, e, b\}$, $\quad T = \{z\}$

dist (z) = $\{23, 21 + 2\}$

Q.



Step 1 : (from edges)

$P = \{D\}$ $\quad T = \{A, B, C, E, F, G\}$

0 $\quad \infty$

dist (A) = $\{\infty, 0 + \infty\} = \infty$

(B) = $\{\infty, 0 + 15\} = 15$

(C) = $\{\infty, 0 + \infty\} = \infty$

(E) = $\{\infty, 0 + \infty\} = \infty$

(F) = 5

(G) = 23

Step 2 :

$P = \{D, F\}$ $\quad T = \{A, B, C, E, G\}$

dist (A) = $\{\infty, 5 + \infty\} = \infty$

(B) = $\{15, 5 + \infty\} = 15$

$(C) = 9+5 = 14$

$(E) = \infty$

$(G) = \{23, 5+13\} = 18$

Step 3:

$P = \{D, F, C\}$, $\bar{T} = \{A, B, E, G\}$

dist $(A) = \{\infty, 14+\infty\} = \infty$

$(B) = \{15, 14+\infty\} = 15$

$(E) = \{15,$

→ Warshall's all pair.

$A_{ij}^{(0)}$ $A_{ij}^{(1)}$ → i d. i—j or i—1—j

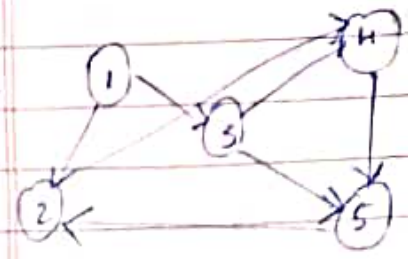
$A_{ij}^{(2)}$ i—j , i—1—j , i—2—j , i—1—2—j

$A_{ij}^{(3)}$ $\{1, 2, 3\}$

$A_{ij}^{(r)}$ $\{1, 2, 3, \ldots, r\}$

$A_{ik}^{(r)}$ $A_{kj}^{(r)}$ → $A_{ij}^{(r+1)}$

eg:



A_{ij}^{0} =

	1	2	3	4	5
1	0	1	1	0	0
2	0	0	0	1	0
3	0	0	0	1	1
4	0	0	0	0	1
5	0	1	0	0	0

$A_{11}^{(0)} * A_{ij}^{(0)}$ gate first column is 0

$A^{(1)}$ =

	1	2	3	4	5
1	0	1	1	0	0
2	0	0	0	1	0
3	0	0	0	1	1
4	0	0	0	0	1
5	0	1	0	0	0

A^2 =

	1	2	3	4	5
1	0	1	1	1	0
2	0	0	0	1	0
3	0	0	0	1	1
4	0	0	0	0	1
5	0	1	0	1	0

$A_{i2} * A_{2j}$

$A_{12} * A_{2H}$
$= A_{1H}^{(1)}$

$A_{52} < A_{2H}$
$= A_{5H}$

A^3 =

	1	2	3	4	5
1	0	1	1	1	1
2	0	0	0	1	0
3	0	0	0	1	1
4	0	0	0	0	1
5	0	1	0	1	0

$A_{13} \times A_{3H}$

$A_{13} \times A_{35} = 1$

A_5 =

	1	2	3	4	5
1	0	1	1	1	1
2	0	1	0	1	1
3	0	1	0	1	1
4	0	1	0	1	1
5	0	1	0	1	1

for weighted write ∞ for non existing edge then next following matrices just do the same but add them and replace that cell's value with the min (old value, new value)

SORTING

1) Bubble sort. (max n-1 iterations) (Largest element g... last at first)

n^2 → worst
n → best

	0	1						
	39	9	81	45	90	27	72	18
1st	9	39	45	81	27	72	18	90

swap alternates

2) Insertion sort

1st	9	39				
	9	39	81			
n^2	9	39	45	81		sorted subarray.
	9	31	45	81	90	
n	9	27	31	45	81	90

(iii) Selection sort (min element at each index comparing with all other values dom...)

	9	39	81	45	90	27	72	18
1st	9	18	81	45	90	27	72	39
n^2	9	18	27	45				

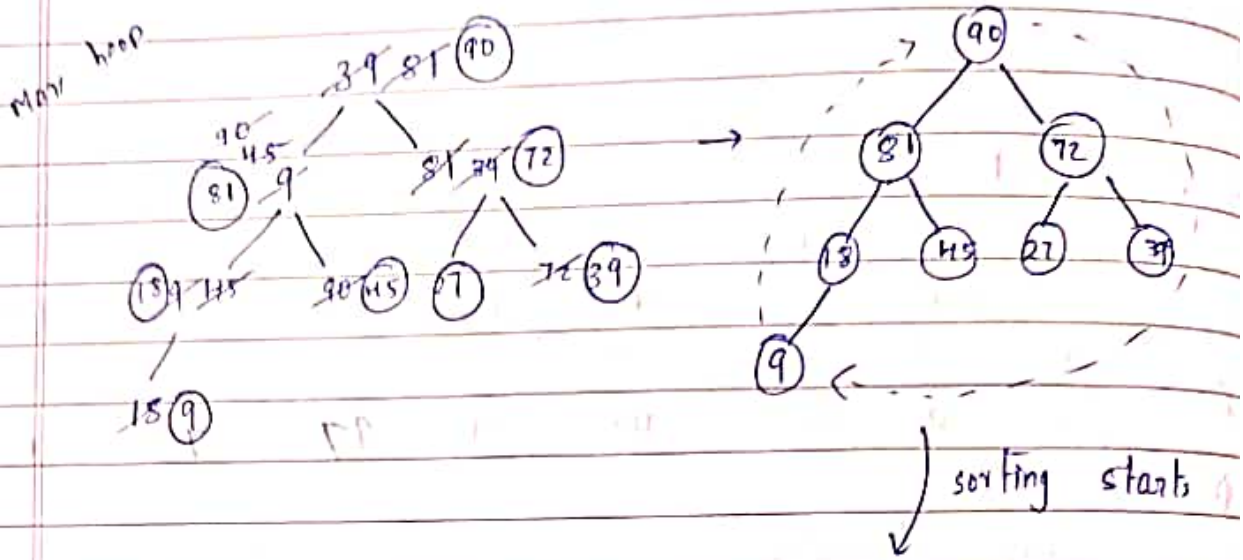
all times

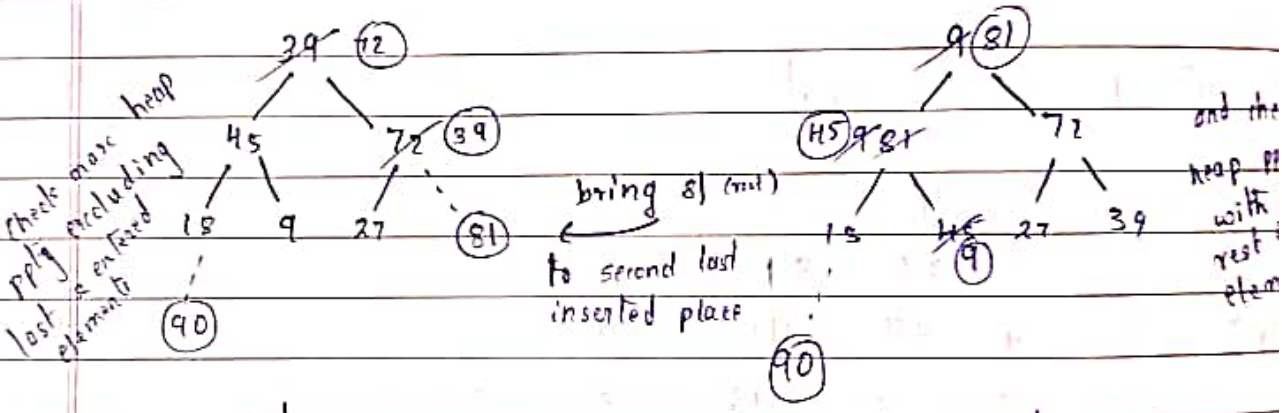
→ Max Heap tree. (Heap sort) (Large tree.)

$n \log n$ insert in level order. each time check with parent and ancestor and swap until max value is at top at all iterations.

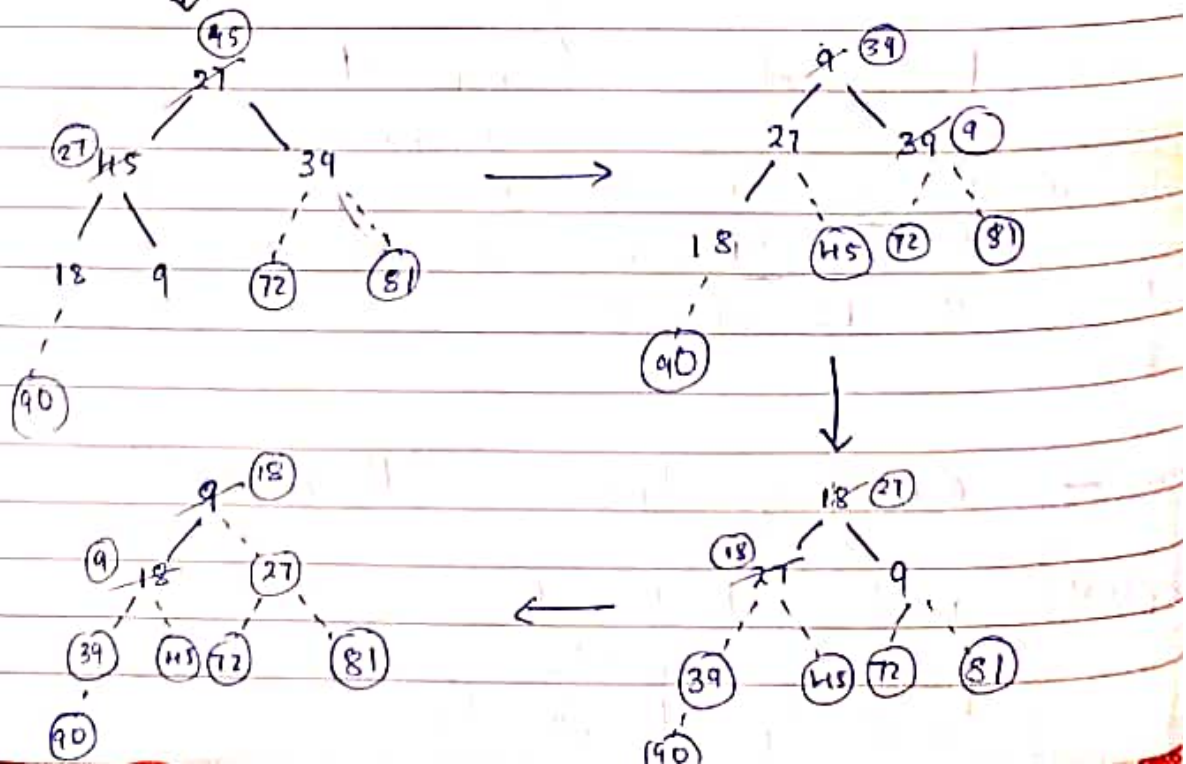
Heap sort

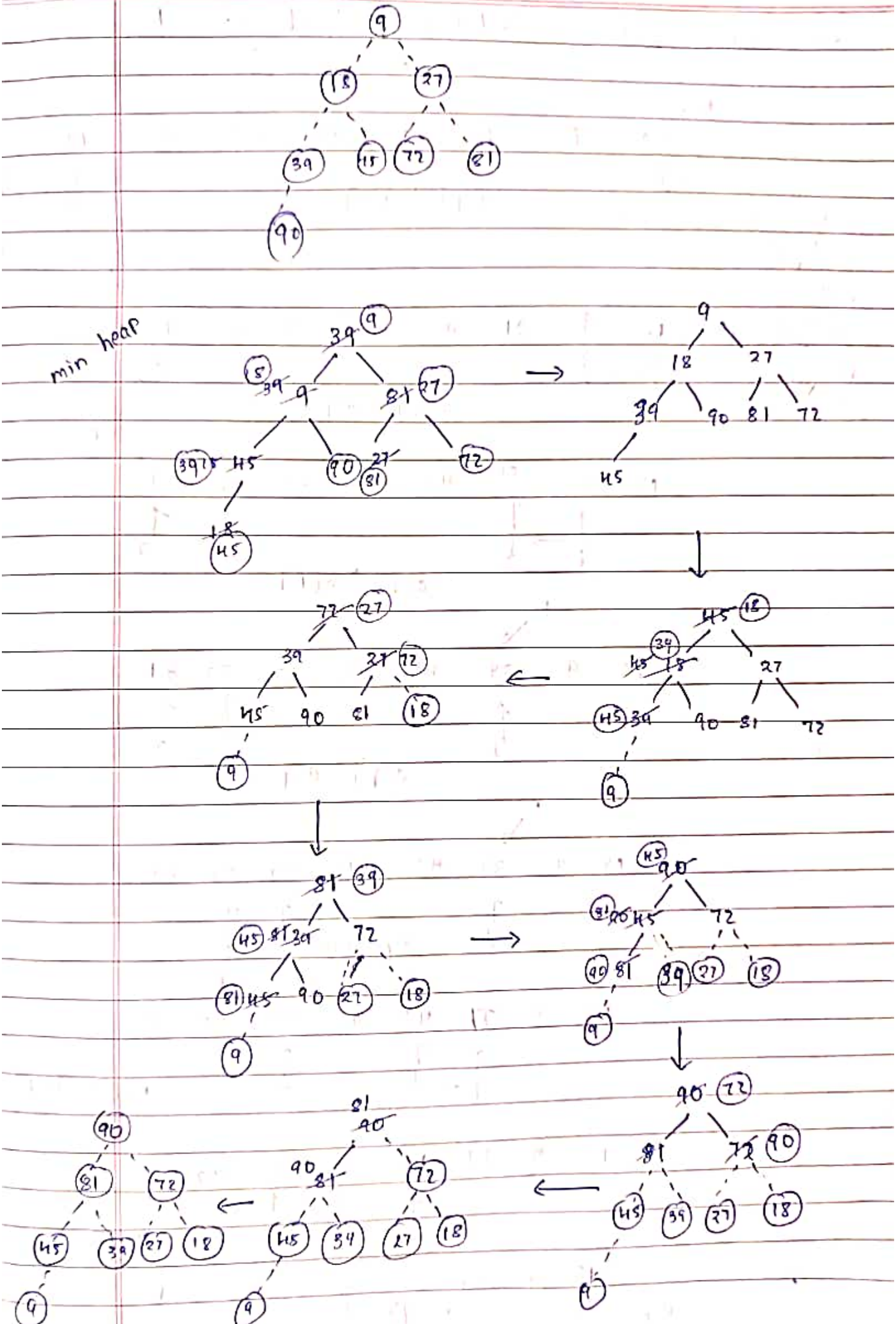
39 9 81 45 90 27 72 18





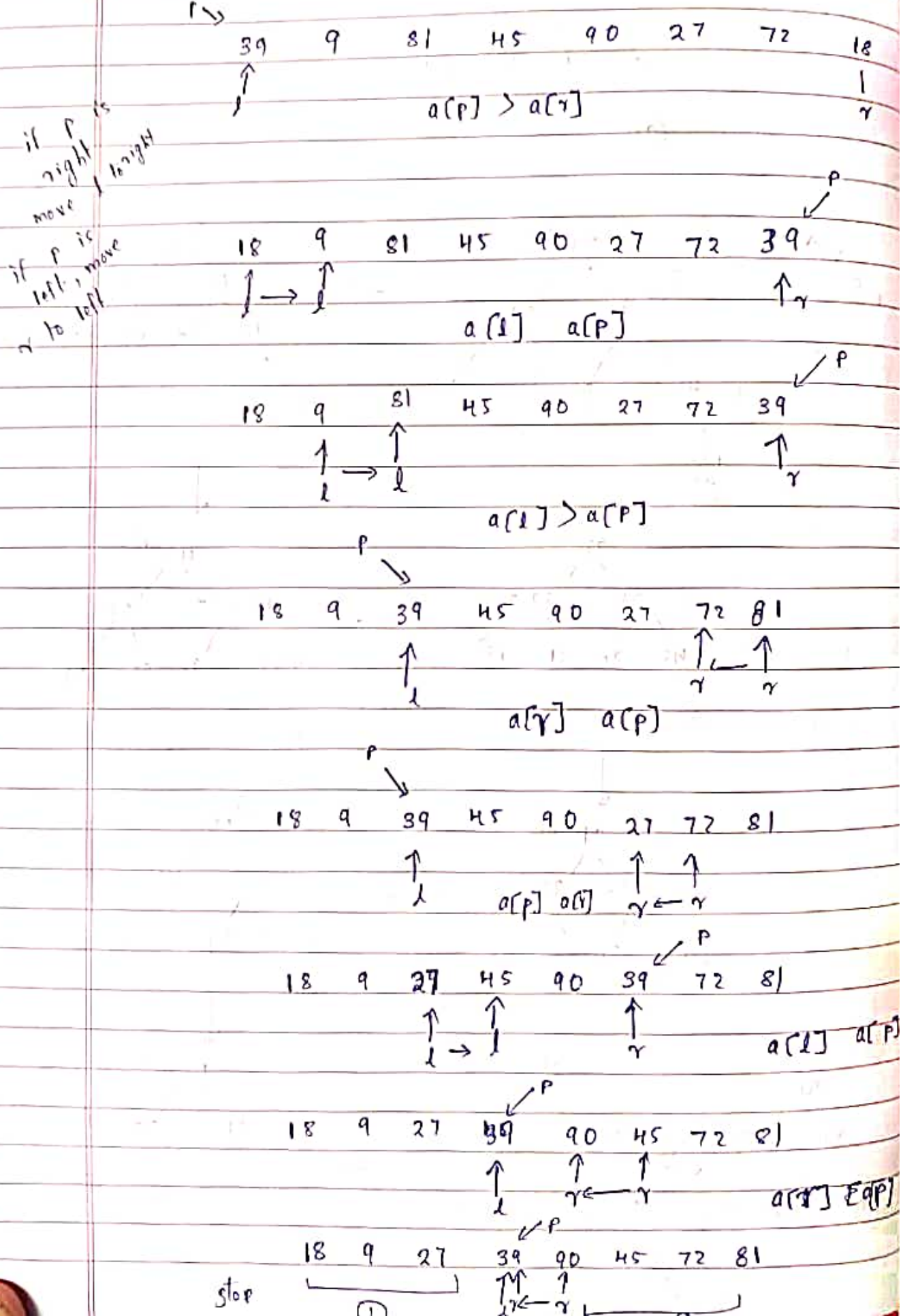
final place we last inserted.

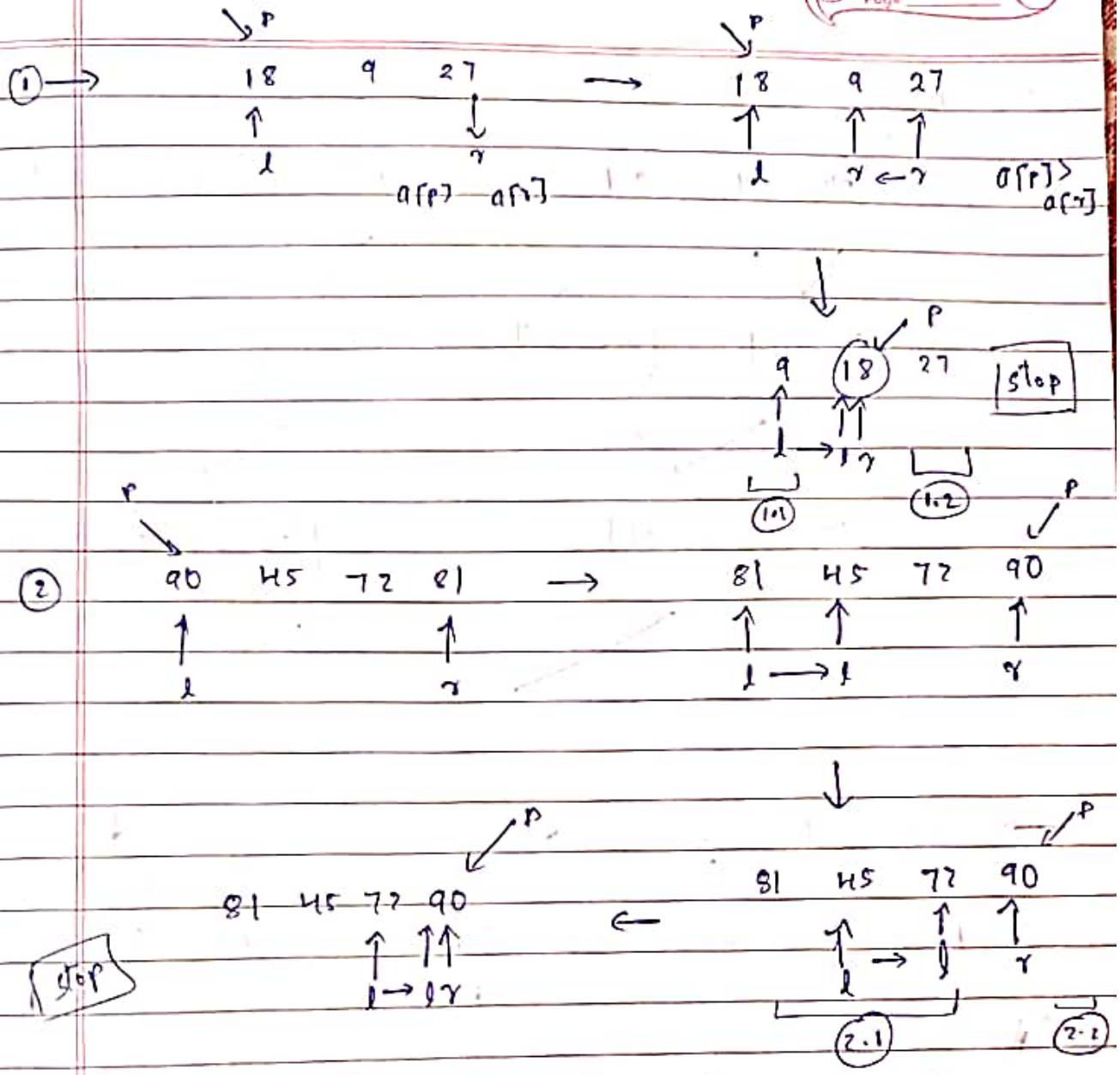



min heap

first last middle random

→ Quick sort (selecting pivot element leads to split into lists)





2.1 81 45 72

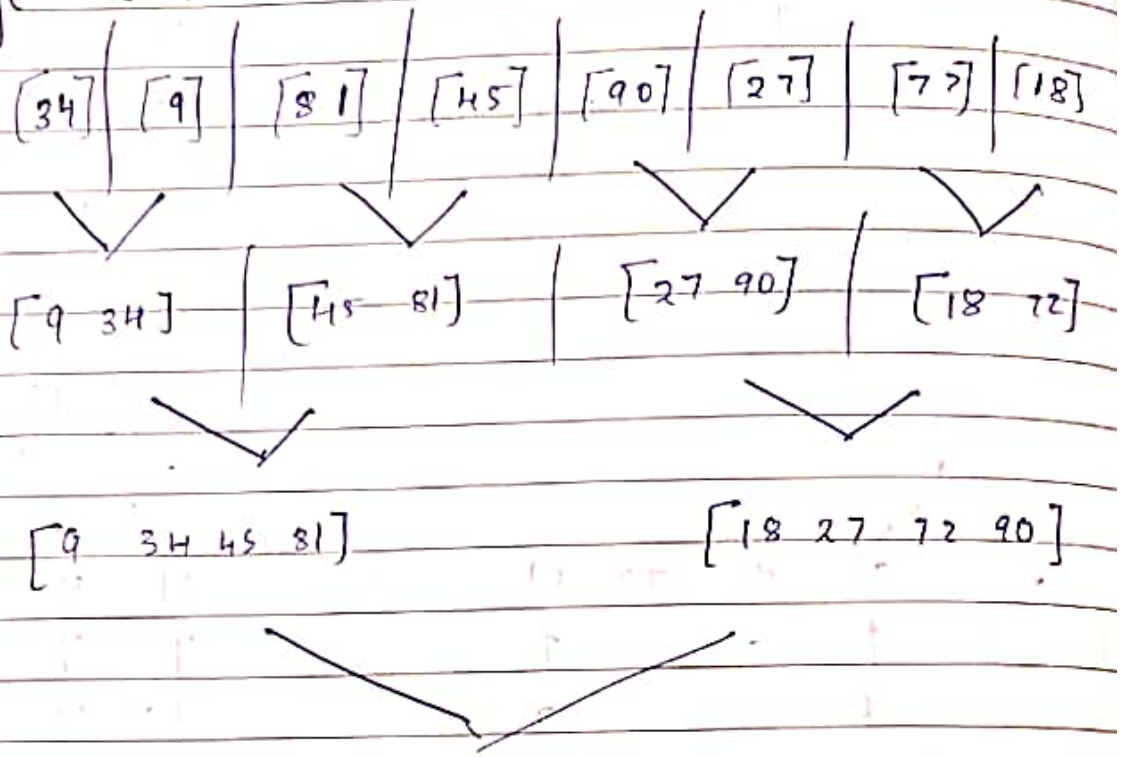
72 45 81

2.1.1

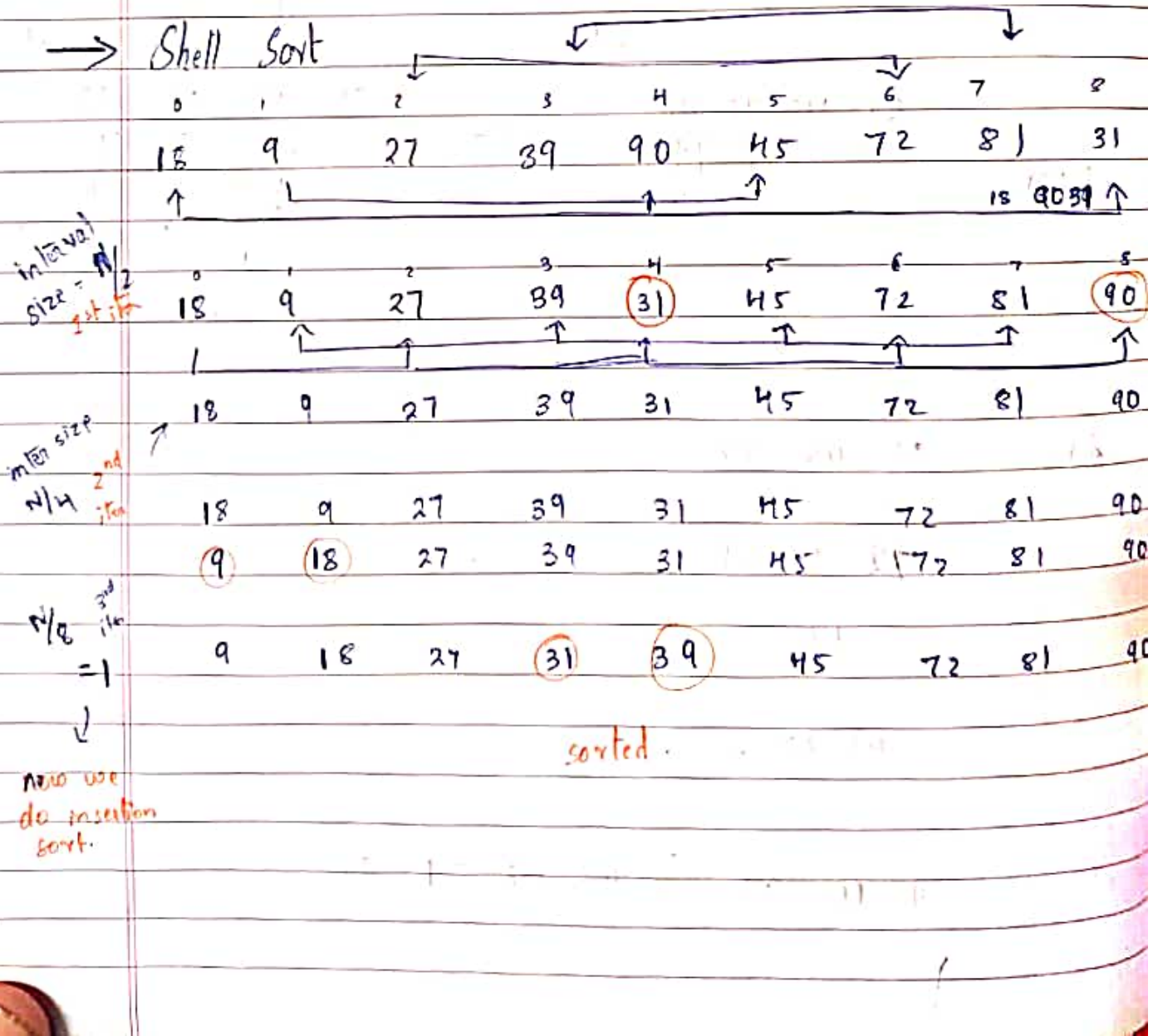
45 72 ✓

9 18 27 39 45 72 81 90

→ Merge sort



→ Shell Sort



→ Radix Sort (make every no. to max int size)

218 009 327 457 023 459 078 891

start with 1's

218	891	009	009
009	023	218	023
327	327	023	078
457	457	327	218
023	218	457	327
459	078	459	457
078	009	078	459
891	459	891	891

sorted

→ Address calculation sort.

10 files for 10 digits.