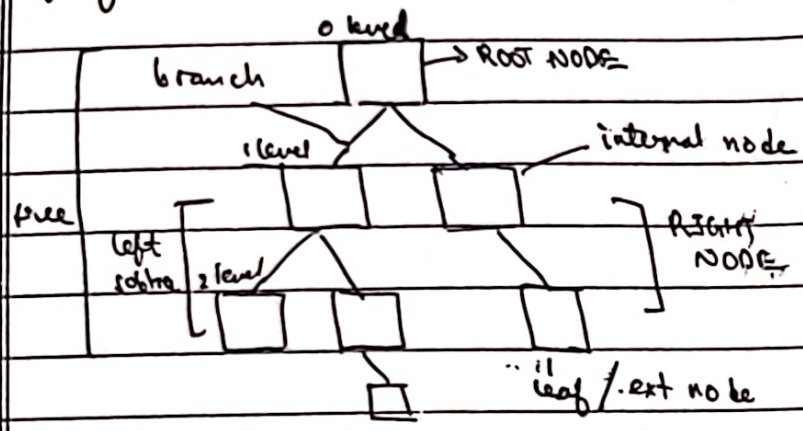
Saathi

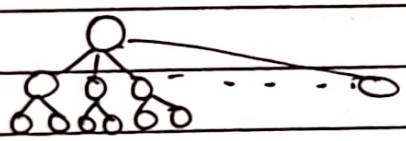
Date __/__/____

UNIT - III TREE

eg of tree → (binary tree)



eg of forest →

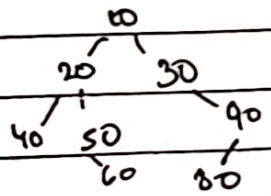


METHODS OF TRAVERSING

1- PRE-ORDER
2- IN ORDER
3- POST-ORDER

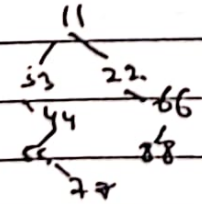
Q- preorder traverse -

(i)



root node 10, 20, 40, 50, 60, 30, 90, 80

(ii)



root node 11, 33, 44, 55, 77, 22, 66, 88

Page No.

Date ___ / ___ / _____

Saathi

Q- Inorder traversal - → necessary to make tree from nodes : give info about left & right subtrees

left subtree — root node — right subtree

i) 40, 20, 50, 60, 10, 30, 80, 90

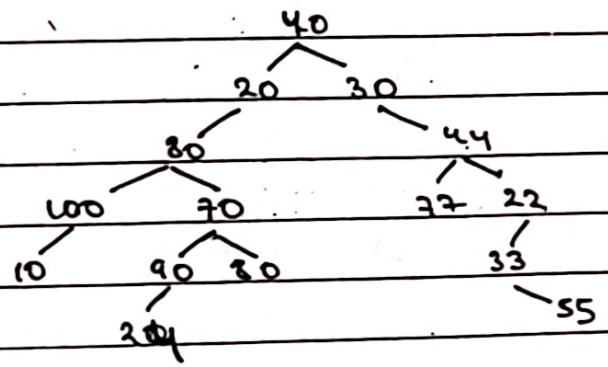
ii) 33, 55, 77, 44, 11, 22, 88, 66

Q- Post order traversal -

i) 40, 60, 50, 20, 80, 90, 80, [10] → root node

ii) 77, 55, 44, 33, 88, 66, 22, [11]

Q-



=> PRT → 40, 20, 80, 100, 10, 70, 90, 24, 80, 30, 44, 77, 22, 33, 55

IN → 100, 10, 80, 24, 90, 70, 80, 20, 40, 30, 77, 44, 33, 55, 22

POST → 10, 100, 24, 90, 80, 70, 80, 20, 77, 55, 33, 22, 44, 30, 40

Page No. ____

2/10/19

Date __/__/____

Saathi

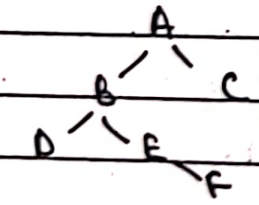
TREE - It is a non-linear data structure. It is mainly used to represent a data with hierarchical representation with records.

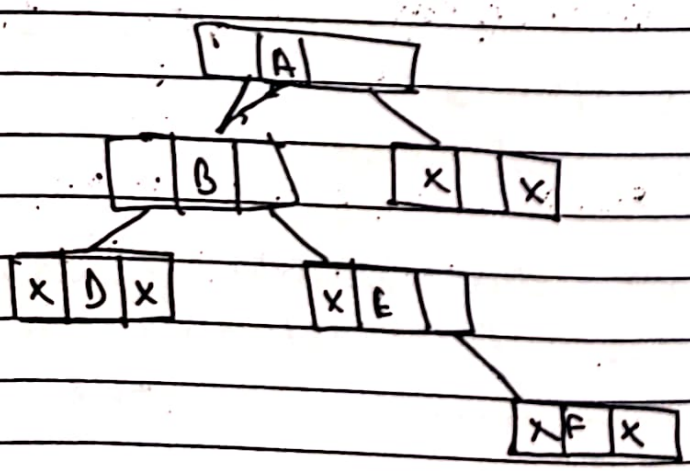
BINARY TREE - (T) - It is defined as a finite set of elements called nodes,

(i) let 'T' be null / empty tree.

(ii) 'T' contains a distinguish node 'R' called root node of 'T' & remaining node of 'T' from an order pair of disjoint trees T_1 & T_2.

→ linked list -





Page No.

(left to right fill)

Date ___/___/_____

Saathi

COMPLETE BINARY TREE - It has at most 2 children, one can show the level 'R' of 'T' can have at most '2^R' nodes.

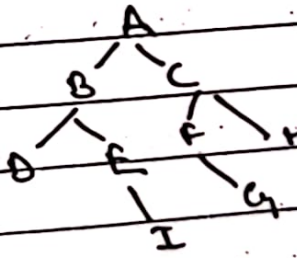
→ The tree 'T' is said to complete in all levels except possibly the last node possible levels.

→ In complete binary tree left & right children of node 'K' are '2K' & '2K+1' and parent of 'K' is the node 'K/2'

→ The depth of complete binary node

$$D_n = \lfloor \log_2 n + 1 \rfloor$$, n → no of nodes

Height $$= \lfloor \log_2 n \rfloor$$



		L	R
Starts 1	A	2	3
2	B	4	5
3	C	6	8
4	D	x	x
5	E	x	9
6.	F	x	7
7.	G	x	x
8	H	x	x
9	I	x	x

Page No

Applications → mathematical eqn, binary opt.

Saathi

Date __/__/____

EXTENDED BINARY TREE- A binary tree is said to be 2-tree or extended binary tree if each node 'n' has either 0 or 2 children. In such cases, the node with 2 children are called internal node & node with 0 children is called external node.

ALGO FOR PREORDER TRAVERSING →

PREORDER (INFO, LEFT, RIGHT, ROOT)

STEP 1- Initialy push NULL onto stack & initialise.
set TOP = 1, stack[1] = NULL
& PTR = ROOT

STEP 2- Repeat STEP 3 to 4 while PTR ≠ NULL
Apply process to INFO[PTR]

STEP 3- If RIGHT[PTR] ≠ NULL then
TOP = TOP + 1 &
STACK[TOP] = RIGHT[PTR]
[end if structure]

STEP 4- If LEFT[PTR] ≠ NULL then
set PTR = LEFT[PTR]
else

Page No.

Date ___ / ___ / _____

Saathi

Set PTR = STACK[TOP] & TOP = TOP-1

[end if structure]

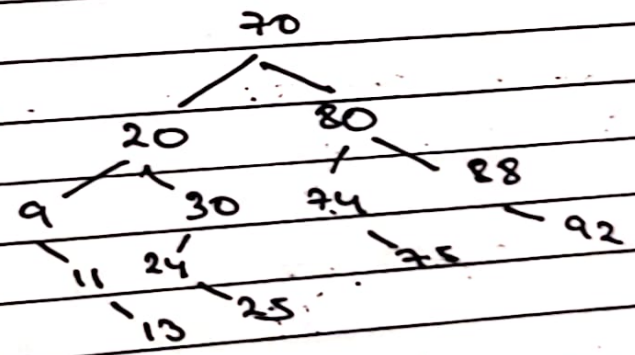
STEP 5- Exit

~~ALGO FOR~~

BINARY SEARCH TREE (BST)

L <= N, R > N

70, 80, 20, 30, 9, 88, 74, 92, 11, 13, 24, 75, 25



Page No. [ ]

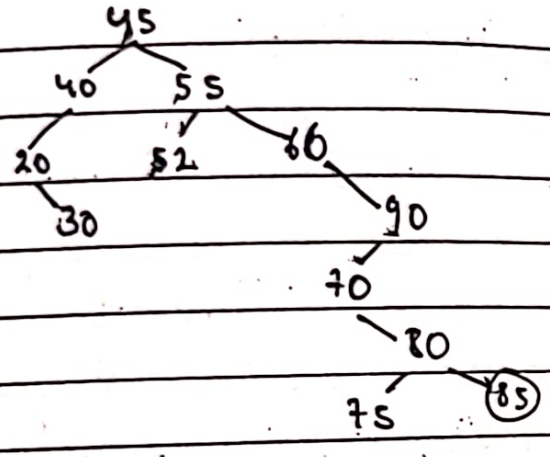
Saathi

Date __/__/____

10/10/19

BINARY SEARCH TREE

45 55 66 40 20 30 52 90 70 80 75



insert - 85

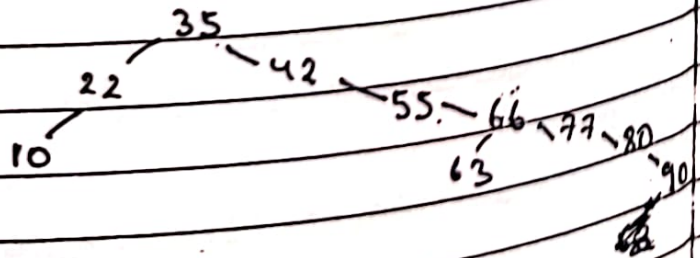
Q- 35 42 55 66 77 80 22 10 90 63

(i) insert → 40
25

(ii) delete → 55
77
66

(iii) insert → 70
20

⇒



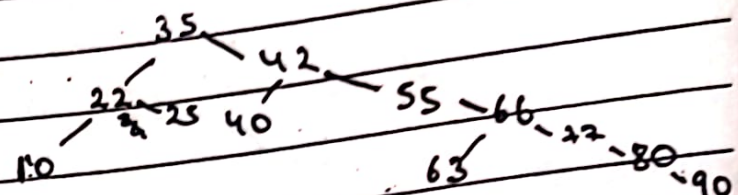
Page No.

* In deletion, inorder ↑ then successor to be selected.

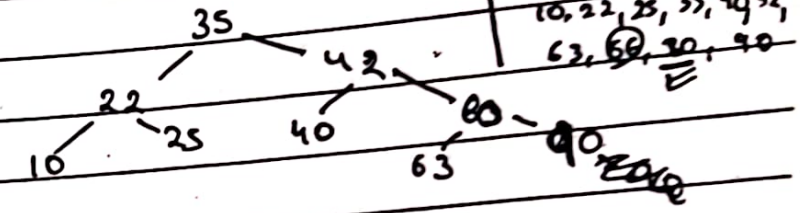
Saathi

Date ___/___/_____

(i)

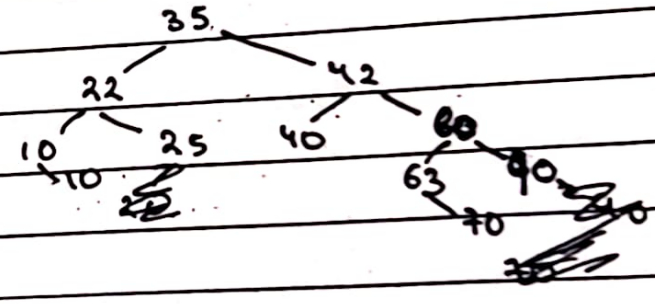


(ii)



inorder →
10, 22, 25, 35, 40, 42, 63, 80, 90

(iii)



Q- 60 65 55 50 67 45 25 70 10 90
77 35 9

i) insert → 69
29
39

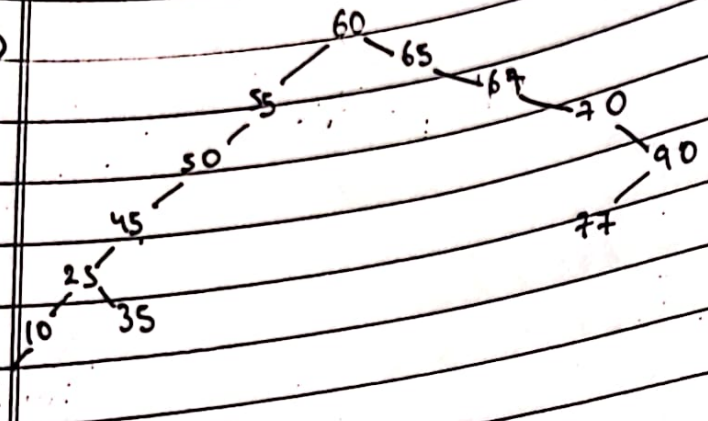
ii) delete → 67
35
70

iii) Insert → 72
62
23

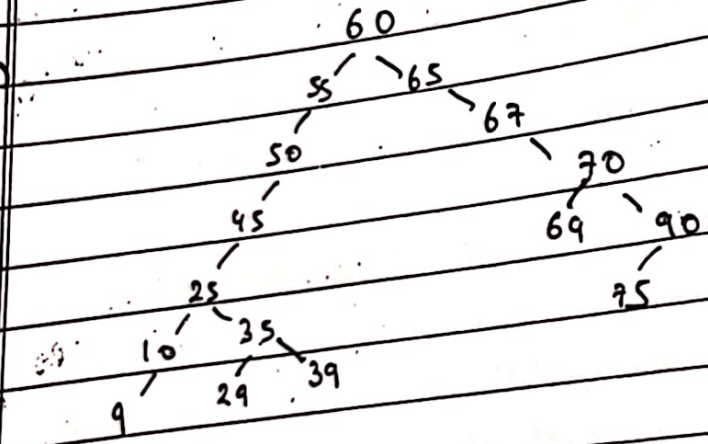
Page No.

Date __/__/____

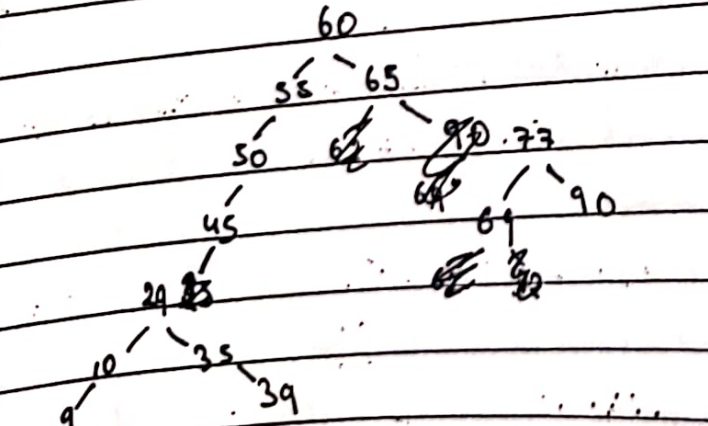
=>



(i)



(ii)



→ 9 10 25 29 35 39 45 50 ... // del 25

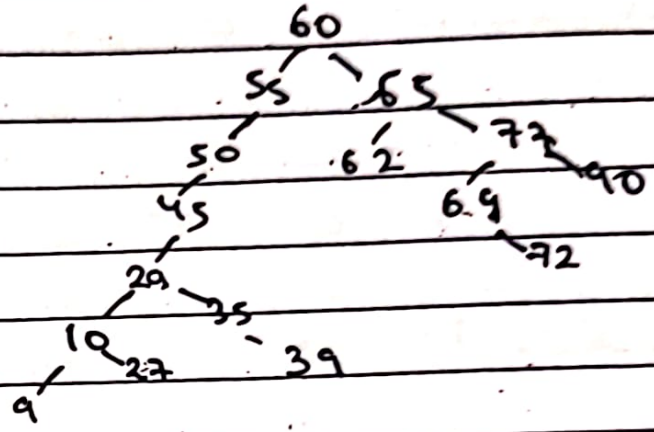
→ 9 10 29 35 39 45 50 55 60 65 69 70 77 \

Page No.

Saathi

Date ___ / ___ / ____

iii)



ALGO FOR INORDER

INORDER (INFO, LEFT, RIGHT, PTR)

STEP 1- [Push NULL onto stack]
set TOP = 1, stack[1] = NULL, PTR = ROOT

STEP 2- Repeat while PTR ≠ NULL
a) set TOP = TOP + 1 & stack[TOP] = PTR
b) set PTR = LEFT[PTR]
[end of loop]

STEP 3- set PTR = stack[TOP] & TOP = TOP - 1

STEP 4- Repeat STEP 5 to 7 while PTR ≠ NULL

STEP 5- Apply process to INFO[PTR]

Page No. ____

Saathi

Date __/__/____

STEP 6- If RIGHT[PTR] ≠ NULL
a) Set PTR = RIGHT[PTR]
b) Set PTR = RIGHT[PTR]
c) Go to step 3

STEP 7- Set PTR = Stack[TOP]
TOP = TOP - 1

STEP 8- Exit

ALGO FOR INSERTION IN BST

FIND(INFO, LEFT, RIGHT, ROOT, VALUE, LOC, PAR)

(i) LOC = NULL & PAR = NULL, which indicate tree is empty.

(ii) LOC ≠ NULL & PAR = NULL, which indicate that the value is the root of T.

(iii) If LOC = NULL & PAR ≠ NULL, value does not exist in tree.

STEP 1- if ROOT = NULL then set LOC = NULL & PAR = NULL

STEP 2- if value = INFO[ROOT]

Page No.

Date ___ / ___ / _____

Saathi

then set LOC = ROOT & PAR = NULL

STEP 3- if VALUE < INFO[PTR]
set PTR = LEFT[ROOT] & SAVE = ROOT
else
set PTR = RIGHT[ROOT] & SAVE = ROOT
[end if]

STEP 4- Repeat S5 & 6 while PTR ≠ NULL

STEP 5- If VALUE = INFO[PTR] then
set LOC = PTR & PAR = SAVE

STEP 6- If VALUE < INFO[PTR] then set
SAVE = PTR & PTR = LEFT[PTR]
else
set SAVE = PTR & PTR = RIGHT[PTR]
[end if structure]

STEP 7- set LOC = NULL & PATH = SAVE

STEP 8- Exit

Page No.

Saathi

Date ___ / ___ / _____

INSERTBST(INFO, LEFT, RIGHT, ROOT, AVAIL, VALUE, LOC)

STEP 1 - Call FIND(INFO, LEFT, RIGHT, ROOT, VALUE, LOC, PAR)

STEP 2 - if LOC ≠ NULL

STEP 3 - if Avail = NULL then write "overflow"

STEP 4 - set ~~Avail~~ = New = Avail &
Avail = LEFT[Avail], INFO[New] = Value

STEP 5 - New = LOC
set ~~Left~~, LEFT[New] = NULL &
RIGHT[NEW] = NULL

STEP 6 - [add value to tree]
set ROOT = New
else if value < info[ROOT] then
set LEFT[PAR] = New
else
set RIGHT[PAR] = New

STEP 7 - Exit

Page No.

Date ___/___/_____

Saathi

ALGO FOR DELETION IN BST :

CASE_AB (INFO, ROOT, LEFT, RIGHT, LOC, PAR)

STEP 1- if LEFT[LOC] = NULL & RIGHT[LOC] = NULL
set child = NULL
else if LEFT[LOC] ≠ NULL
set child = LEFT[LOC]
else
set child = RIGHT[LOC]
[end if structure]

STEP 2- if PAR ≠ NULL
if LOC = LEFT[PAR] then
set LEFT[PAR] = child
else
set RIGHT[PAR] = child
~~else [end if]~~ ~~else if~~ end if
else :
ROOT = child

STEP 3- Return

Page No.

Date __/__/____

Saathi

CASE C (INFO, LEFT, RIGHT, ROOT, LOC, PAR)

STEP 1:

a) Set PTR = RIGHT[LOC] & SAVE = LOC

b) Repeat while LEFT[PTR] ≠ NULL

Set SAVE = PTR & PTR = LEFT[PTR]

[end of loop]

c) Set SUC = PTR & PARSUC = SAVE

STEP 2 - [Delete Inorder succesor using CASE A]

Call CASE A [INFO, LEFT, RIGHT, LOC, ROOT, PAR]

STEP 3 - a) if PAR ≠ NULL then

if LOC ≠ NULL then

set LEFT[PAR] = SUC

else

set RIGHT[PAR] = SUC

else

set ROOT = SUC

b) Set LEFT[SUC] = LEFT[LOC] &

RIGHT[SUC] = RIGHT[LOC]

STEP 4 - Return

Page No.

* max 3 node will be rotated

Saathi

Date ___ / ___ / _____

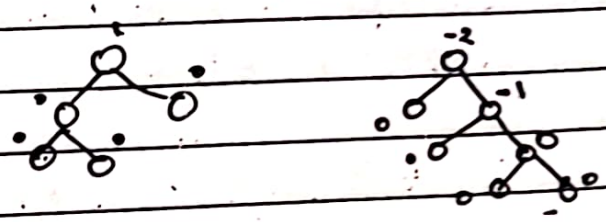
AVL : (BST) (HEIGHT BALANCED TREE)

AVL - balanced binary search tree
(AVL → Adelson Velskii & Landis)

Balanced factor (BF) = $|h_L - h_R|$ $\quad BF \leq 1$

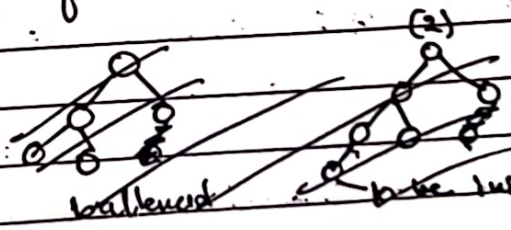
BF = 0, 1, -1 → balanced
→ not balanced

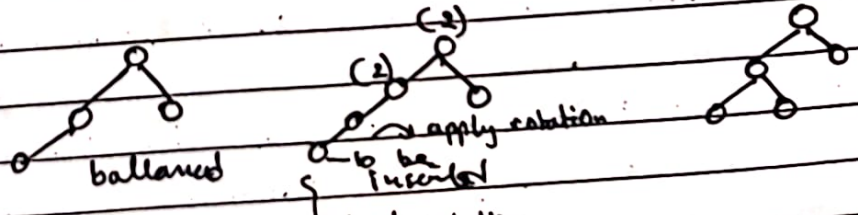
eg



rotations → LL → clockwise dir
→ RR → anti-clockwise dir
→ LR
→ RL

eg of LL →





start rotation from leaf node
if still not balanced get to next node

Page No. ____

Date __/__/____

Saathi

The AVL tree is a BST that has an additional balanced condition. This balanced cond must be maintained in AVL tree.

An empty binary tree B is an AVL tree if the non-empty binary tree with BL & BR are its left & right subtree then tree is AVL tree iff -

(i) BL & BR are AVL trees

(ii) $h_L - h_r \leq 1$ // ballance factor

h_L = height of left sub-tree, h_r = height of right sub-tree

- LL rotation means inserted node is in the left subtree of left subtree of node.
- RR rotation - inserted node is in the right subtree of right subtree of node.
- LR - Inserted node is in the right sub-tree of left subtree of node.

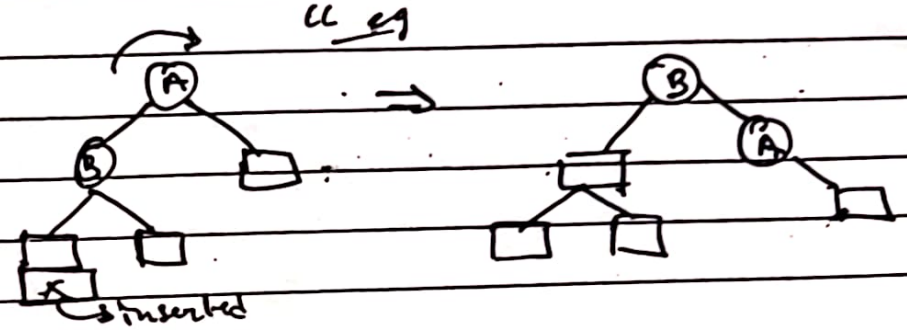
Page No.

Saathi

Date ___ / ___ / _____

- RL - inserted node is in the left subtree of right subtree of node.

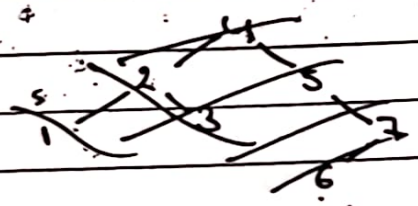
eg



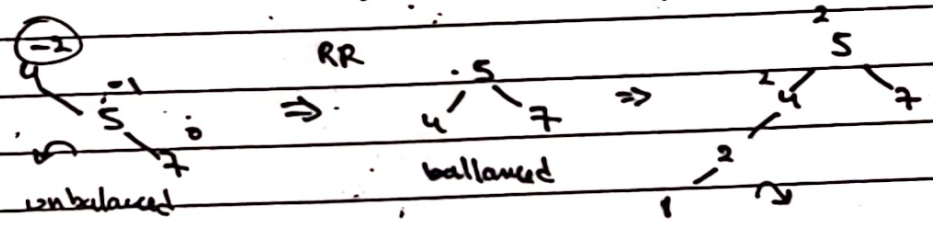
\# LL, RR are single rotation
LR, RL are double rotation

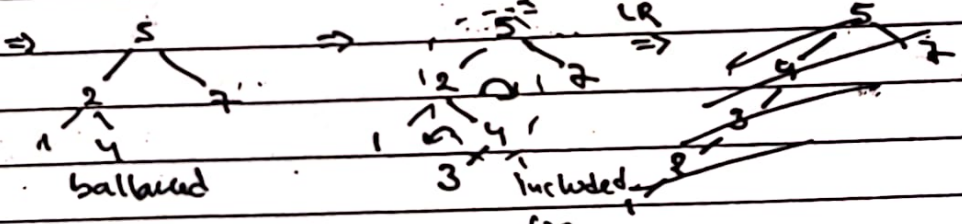
Q- 4 5 7 2 1 3 6

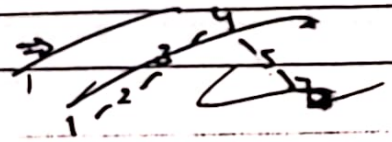
⇒



⇒



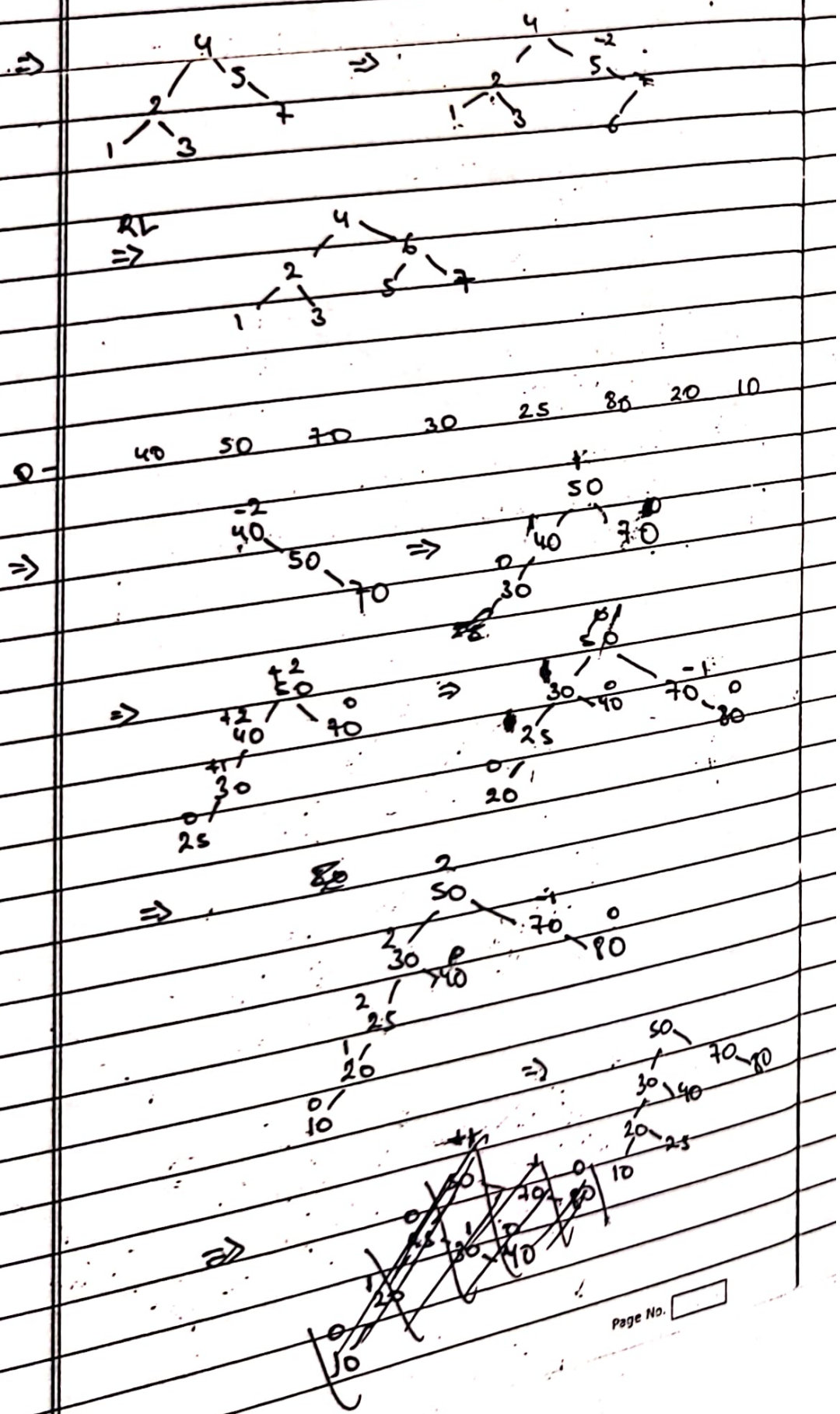




Page No. ____

Saathi

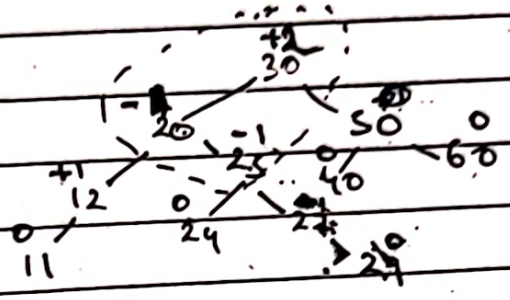
Date __/__/____

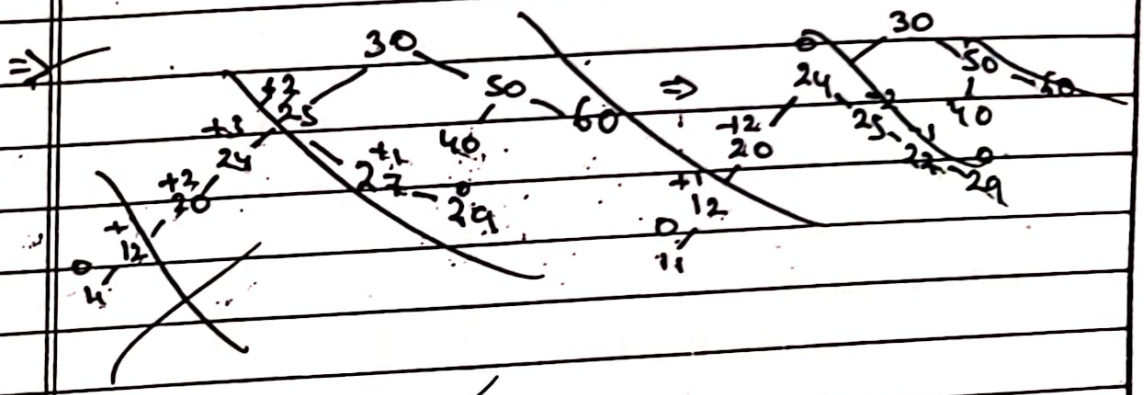


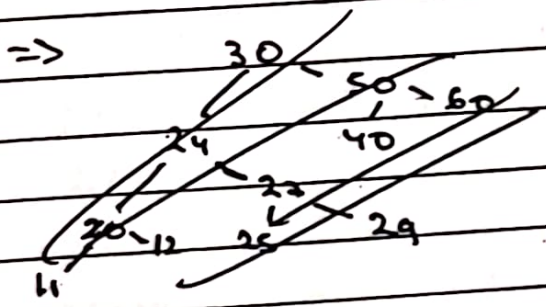
Saathi

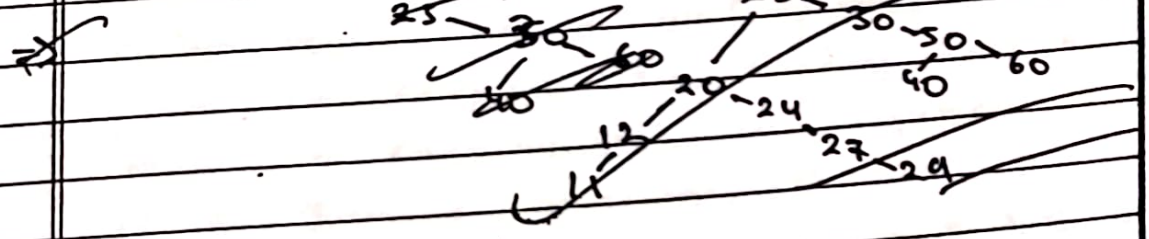
Date ___/___/_____

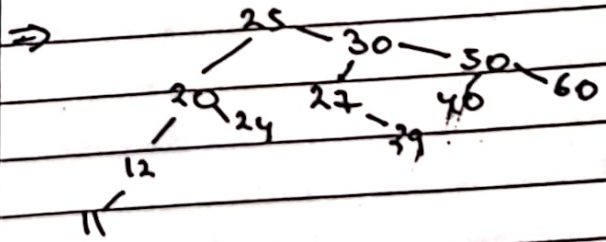
Q.







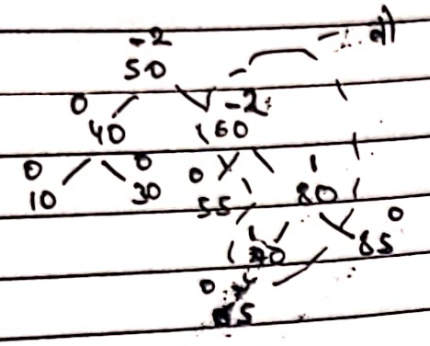




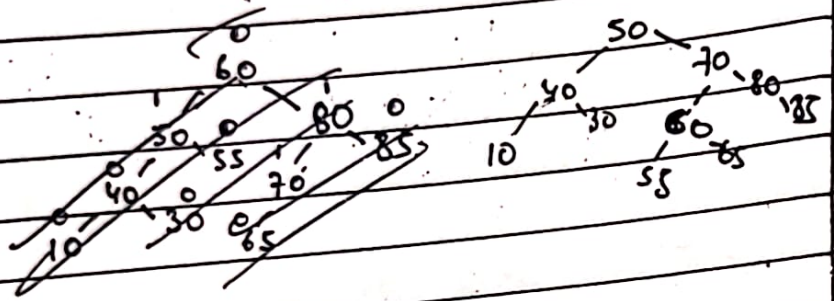
Page No. ☐

Saathi

Date ___/___/_____

Q - 

... वो 3 node identify होंगे जिसकी वजह से unbalanced हुआ

⇒ 

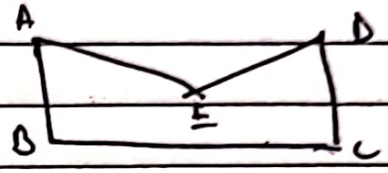
Page No. ☐

* In case of directed graph arrows will be present to represent the direction.

Date ___ / ___ / _____

UNIT - IV - GRAPH

eg →



// undirectional graph

⇒ AB = BA

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

- DFS → depth first search // stack used
  BFS → breadth first search // queue used

- The general idea behind the BFS beginning at a starting node A first we examine the starting node A then we examine all the neighbours of A. Then we examine all the neighbour's of the neighbour of A & so no. We need to keep track of the neighbours of a node & we need to guarentee that no node is processed more than once. This is accomplished by using a queue to hold nodes that are waiting to be processed & change the status of the particular node.

Page No.

Date __/__/____

Saathi

Status ← Ready / waiting / processed

Ready → Read/Traverse
↓

waiting -

processed -

ALGO FOR BFS

STEP 1- Initialise all nodes to the ready state.

STEP 2- Put the starting node A in Queue and change the status to the waiting state.

STEP 3- Repeat STEP 4 & 5 until queue is empty.

STEP 4- Remove the front node N of queue. Process N by change the status of N to the processed state.

STEP 5- Add to the rear of Queue all the neighbours of N that are in the ready state, and change their status to the waiting state.

Q-

जिसको path से बाहर निकाला उसे queue में include करेंगे

Date __/__/____

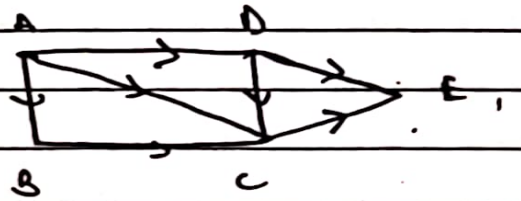
Saathi

[end of step 3 loop]

STEP 6 - Exit

BFS

eg→



Ready state → A, B, C, D, E

Queue - A

Source - A

queue - B C D

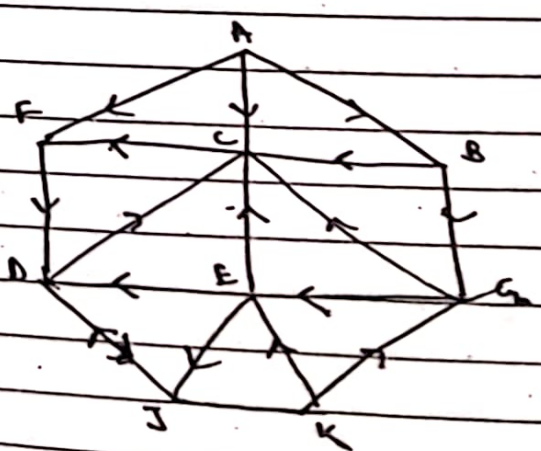
Source - A B C

queue - D E

source - A B C D

queue - E

source - A B C D E



Source A.

Page No.

Saathi

Date __/__/____

⇒ by using adjency list

A	F, C, B
B	C, G
C	F
D	C
E	J, D, C
F	D
G	E, C
J	K, D
K	E, G

// F, C, B in any order does not matter

Source A

Queue F C B — F को हटाओ

Source A F

Queue C B D — F का E, C हटाओ

Source A F C

Queue B D F

Source A F C B

Queue D F G — already present in source

Source A F C B D

Queue F G

Source A F C B D K

Que G

Source A F C B D K G

Que E

Page No.

Saathi

Date __/__/____

Source A F C B D E G E

Queue I X X

-S A F C B D E G E J

Q K .

S A F C B D E G E J K

DFS (DEPTH FIRST SEARCH)

STEP 1 - Initialize all nodes to the ready state.

STEP 2 - Push the starting node A onto stack & change the status to the waiting state.

STEP 3 - Repeat STEP 4 & 5 until stack is empty.

STEP 4 - Pop the top node of stack. Process N & change its status to the processed state.

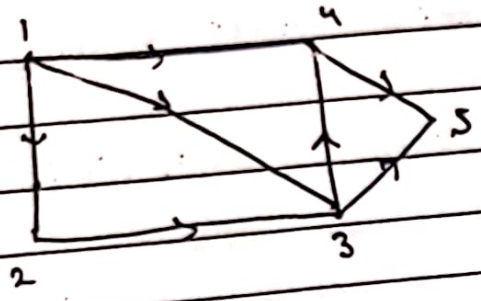
STEP 5 - Push onto stack all the neighbours of N that are still in the ready state & change this state to waiting state.

STEP 6 - Exit.

Page No.

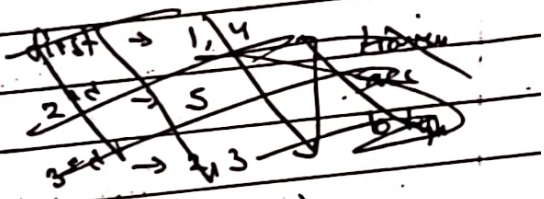
Date __/__/____

* Q-



⇒

1	2, 3, 4
2	3
3	4, 5
4	5
5	—



Stack - 1

source → 1

str. 2, 3, 4 → top → निकाल जाएगा

source 1, 4 ← 4 th column में

stack - 2, 3, 5

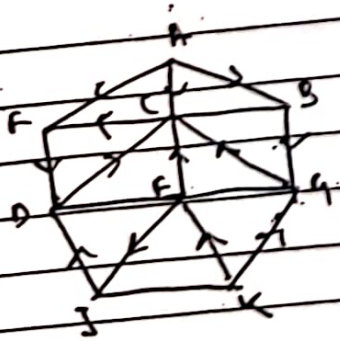
source - 1, 4, 5

Stack → 2, 3

Soore - 1, 4, 5, 3

1, 4, 5, 3, 2

Q-



Page No.

Date ___ / ___ / _____

stack - A

⇒ S. - A

stack - F, C, B

S - A, B

stack - F, C, G

S - A, B, G

stack - F, C, E, ~~E~~

S - A, B, G, E

stack - F, C, I, D

S - A, B, G, E, D

stack - F, C, I, ~~K~~

S - A, B, G, E, F, D

stack - F, C

S - A, B, G, E, D, I

stack - F, C, K

S - A, B, G, E, D, I, K

stack - F, C, ~~K~~ E

S - ~~F, C,~~ A, B, G, E, D, I, E

stack - F, C

S - A, B, G, D, I, E, C

stack - F

S - A, B, G, D, I, E, C, F

⇒ stack	A	S	A BG E
S	A	st	F C I D
st	F C B	S	A B G E D
S	A B	st	F C I
st	F C G	S	A B G E D I
S	A B G	st	F C K
st	F C E	S	A B G E D I K C F

Date __/__/____

Saathi

MINIMUM SPANNING TREE (KRUSKAL'S ALGO)

STEP 1- Initially construct a seperate tree for each node in a graph.

STEP 2- Edges are placed in a priority queue, we take edges in ascending order.

(We can use a heap for the priority queue)

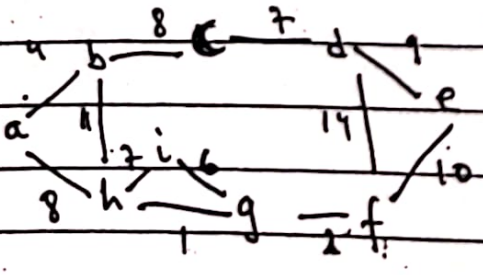
STEP 3- Until we have added n-1 edges-

a) Extract the cheapest edge from the queue.

b) If it form a cycle reject it, else add it to the forest.

STEP 4- Whenever we insert an edge, 2 trees will be joined, every step will have joined 2 trees in the ~~forest~~ forest together so that at the end there will be only 1 tree.

eg→



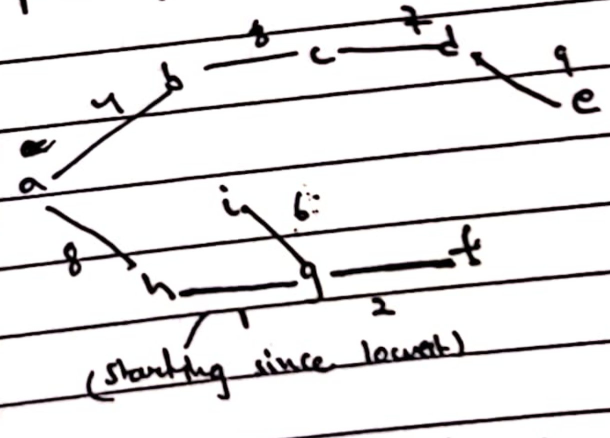
Page No.

Date ___ / ___ / ______

=> ~~Kgst~~ adjency list // arranging in ascending order

h, g	1
g, f	2
~~g, b~~ a, b	4
i, g	6
i, h	7
c, d	7
b, c	8
a, h	8
d, e	9
e, f	10
b, h	11
d, f	14

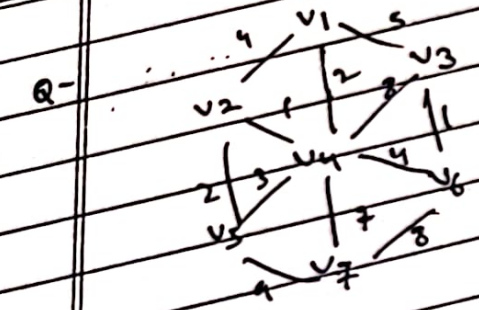
(i) plotting all nodes without edges



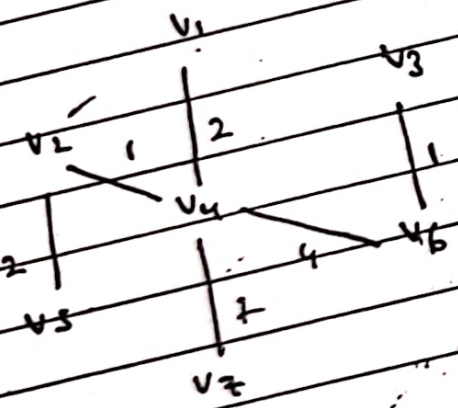
(i, h), (e, f), (d, f), (b, h) are removed

Date ___/___/_____

Q-



⇒



$v_2 v_4$	1
$v_3 v_6$	1
$v_1 v_4$	2
$v_2 v_5$	2
$v_4 v_5$	3
$v_2 v_1$	4
$v_4 v_6$	4
$v_1 v_3$	5
$v_4 v_3$	7
$v_4 v_3$	8
$v_7 v_6$	8
$v_6 v_7$	9

Page No.

Date __/__/____

Saathi

imp + num imp

UNIT - II

[20-25 marks min]

BINARY SEARCH

Q- 5 8 15 25 30 40 55

(i) search 30

⇒ BEG = 0, END = 6, MID = 3

~~A[3] = 30~~ A[M] < 30

BEG = MID + 1 = 4, END = 6, MID = 5

A[M] > 30

BEG = 4, END = MID - 1 = 4, MID = 4

A[M] = 30 → 4//

Q- 4 15 20 35 45 55 65 75

search 20

⇒ B = 0, E = 7, MID = 3

A[M] > 30

B = 0, E = M - 1 = 2, M = 1

A[M] < 20

B = M + 1 = 2, E = 2, M = 2

A[M] = = 20 → 2//

Page No.

SELECTION SORT

avg → $O(n^2)$ worst → $O(n^2)$

Eg →

77 33 44 11 88 22 66 55

11 33 44 77 88 22 66 55

11 22 44 77 88 33 66 55

11 22 33 77 88 44 66 55

11 22 33 44 88 77 66 55

11 22 33 44 55 77 66 88

11 22 33 44 55 66 77 88

11 22 33 44 55 66 77 88

ALGO

MIN(A, K, N, LOC)

STEP 1 - Set MIN = A[K] & LOC = K

STEP 2 - Repeat for J = K+1, K+2 ... N
if min > A[J] then
set MIN = A[J], LOC = A[J], LOC = J

STEP 3 - Return

SELECTION (A, N)

STEP 1 - Repeat STEP 2 & 3 for K = 1 ... N-1

Date __/__/____

STEP 2- Call MIN(A, K, N, LOC)
interchange A[K] & A[LOC]

set Temp = A[K], A[K] = A[LOC]
A[LOC] = temp

STEP 3- Exit

INSERTION SORT ALGO $O(n^2)$, $O(n^2)$

eg→ 77 33 44 11 88 22 66
(-∞) A[0] A[1] A[2] A[3] . . . A[7]

⇒ -∞ 33 77 44 . . . . . .
⇒ ∞ 33 44 77 11 88 22 66
⇒ -∞ 11 33 44 77 88 22 66
⇒ -∞ 11 22 33 44 77 88 66
⇒ -∞ 11 22 33 44 66 77 88

INSERTION (A, N)

STEP 1- set A[0] = -∞

STEP 2- Repeat steps 3 to 5 for K = 2, 3 . . . N

STEP 3- set temp = A[K] & PTR = K-1

STEP 4- Repeat while temp < A[PTR]

Page No.

Date ___/___/_____

Saathi

a) set A[PTR+1] = A[PTR]

b) set PTR = PTR - 1.

[end loop]

STEP5- set A[PTR+1] = Temp

STEP6- Exit

MERGE SORT

eg → 66 33 40 22 55 88 60 11 80 20 50 44 77 30 → divide array into 2 parts.

divide

⇒ 66 33 40 22 55 88 60 11 80 20 50 44 77 30

⇒ 66 33 40 22 55 88 60 | 11 80 20 50 44 77 30

⇒ 66 33 | 40 22 | 55 88 | 60 | 11 80 | 20 50 | 44 77 | 30

⇒ 33 66 | 22 40 | 55 88 | 60 | 11 80 | 20 50 | 44 77 | 30

conquer

⇒ 22 33 40 66 | 55 60 88 | 11 20 50 80 | 30 44 77

⇒ 22 33 40 55 60 66 88 | 11 20 30 44 50 77 80

Page No.

=> 11 20 22 30 33 40 44 50 55 66
77 80 88

(66) 33 40 22 55 88 60 (11) 80 20
50 44 77 30, apply quick sort.

11 33 40 22 55 88 60 (66) 80 20
50 44 77 30

=> 11 33 40 22 55 (66) 60 88 80 20
50 44 77 30

=> 11 33 40 22 55 20 60 88 80 (66)
50 44 77 30

=> 11 33 40 22 55 20 60 (66) 80 88
50 44 77 30

=> 11 33 40 22 55 20 60 30
80 88 50 44 77 (66)

=> 11 33 40 22 55 20 60 30
80 (66) 88 50 44 77

=> 11 33 40 22 55 20 60 30
80 44 88 50 (66) 77

=> 11 33 40 22 55 20 60 30 80
44 (66) 50 88 77

Page No.

max heap → data in descending order

min → ascending

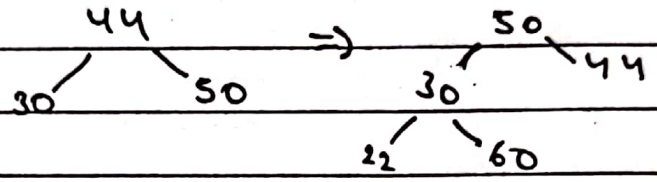
Saathi

Date __/__/____

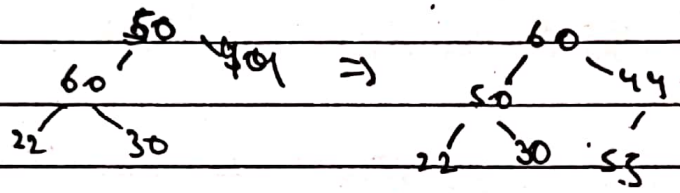
HEAP SORT $O(n \log_2 n)$ (both avg & worst)

eg → 44 30 50 22 60 55 77 55

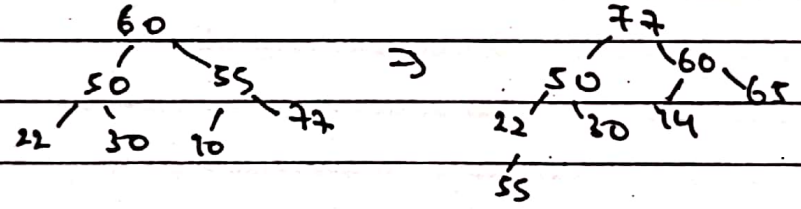
=>



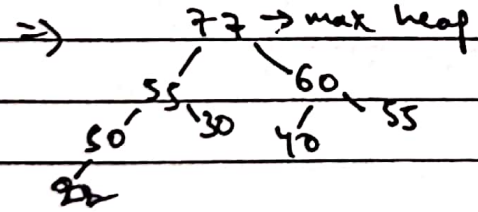
=>



=>



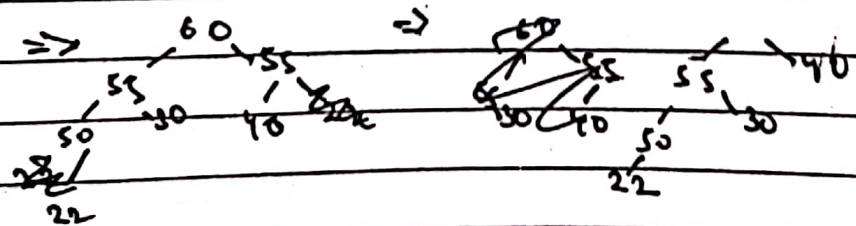
=>



=> 77 60 55 55 50 40 30 22

↑ totally root

=>

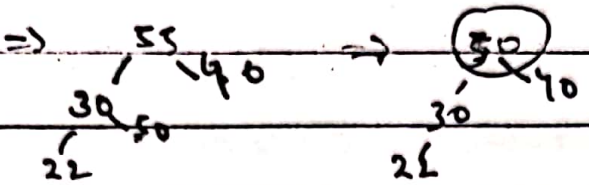


Page No.

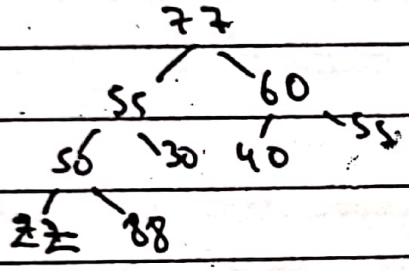
⇒ 50, 35, 44, 22, 77, 35, 60, 40

Saathi

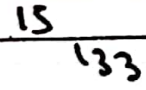
Date ___ / ___ / _____

⇒ 

(ii) Insert 88
70

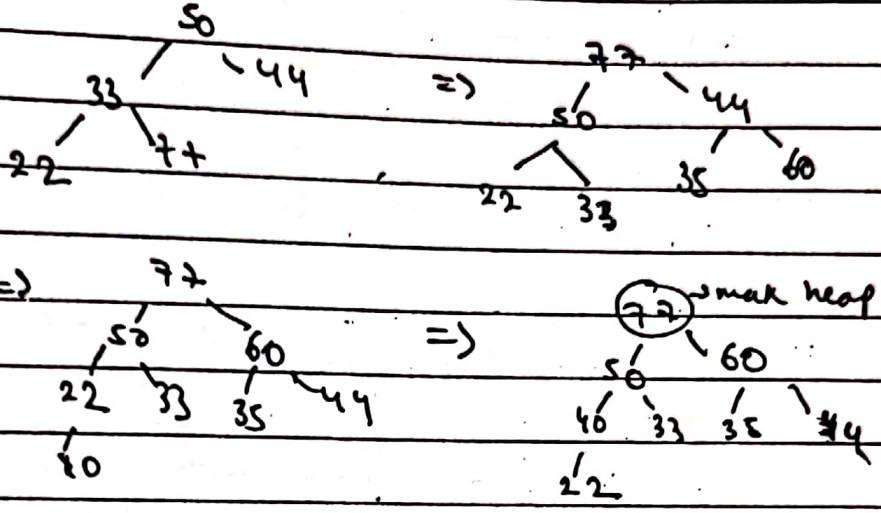
⇒ 

2- 15 33 44 22 77 35 60 40

⇒ 

3- 50, 33, 44, 22, 77, 35, 60, 40

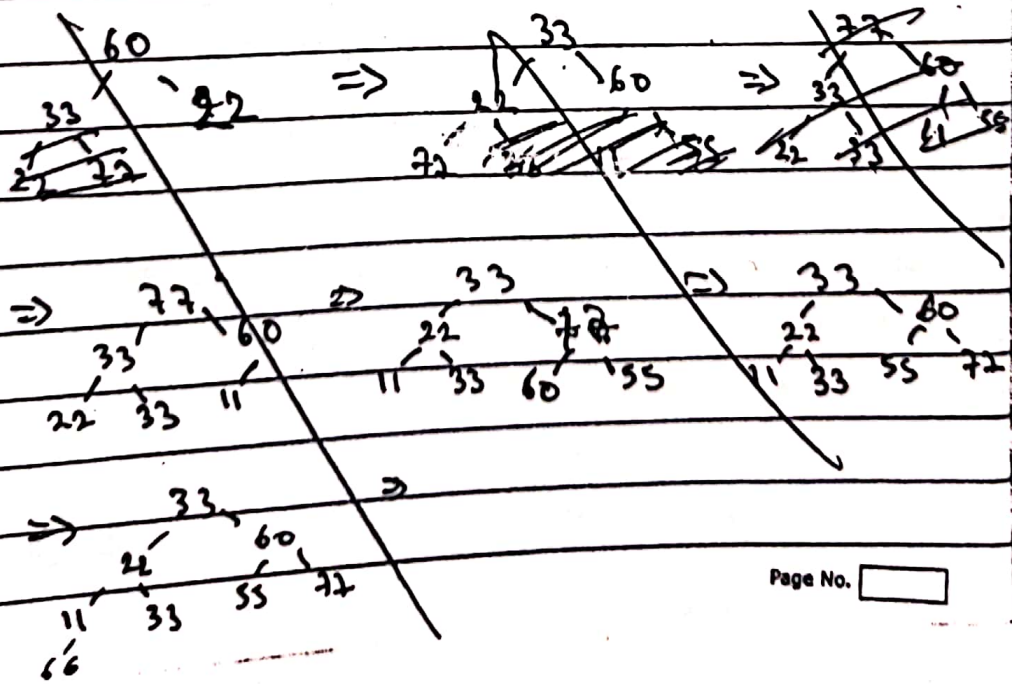
=>



=>

4- 60, 33, 22, 77, 88, 11, 55, 66, 78

=>



Page No.

Suppose H is a complete BT with n elements then H is called a heap or max heap if each node n of H has the following property -

(i) The value at n is greater than or equal to the value at each node of children of n.

(ii) for min heap → The value at n is less than or equal to any of the children of n.

INSERTING INTO A HEAP

Suppose H is a heap with n elements & value of info is given, we insert value into the heap H as follows -

(i) adjoin x value at the end of the edge so that H is still a complete BT, but not necessarily a heap.

(ii) Then let value rise to its appropriate place in H so that H is finally a heap.

* Direct mapping.

Date __/__/____

Saathi

HASHING

$h(k) = k \bmod 10$

$23 \bmod 10 = 3$

$45 \bmod 10 = 5$

$10 \bmod 10 = 0$

0	10
1	
2	
3	23
4	
5	45
6	
7	
8	
9	

1- DIVISION METHOD

$H(k) = k \bmod n$

- H: hash function
- k: key or data
- n: number of storage slots

HASH VALUE

2- MID-SQUARE METHOD

eg → 2369

↓

key = 2 (36)² 9

2 (1234) 9 (sqr)

= 2 [illegible] 4

= 21 (23) 49

↓

hash value

a b c d

(bc)² = xyz

a(xyz)^(n-1)

* div method - choose a number m larger than the no. n of keys in k. The no m is usually chosen to the prime no or a number & without small divisor since this frequently minimise the no of collisions.

Page No. **

* We hash func it is defined by $H(k) = k \bmod n$

Date __/__/____

Saathi

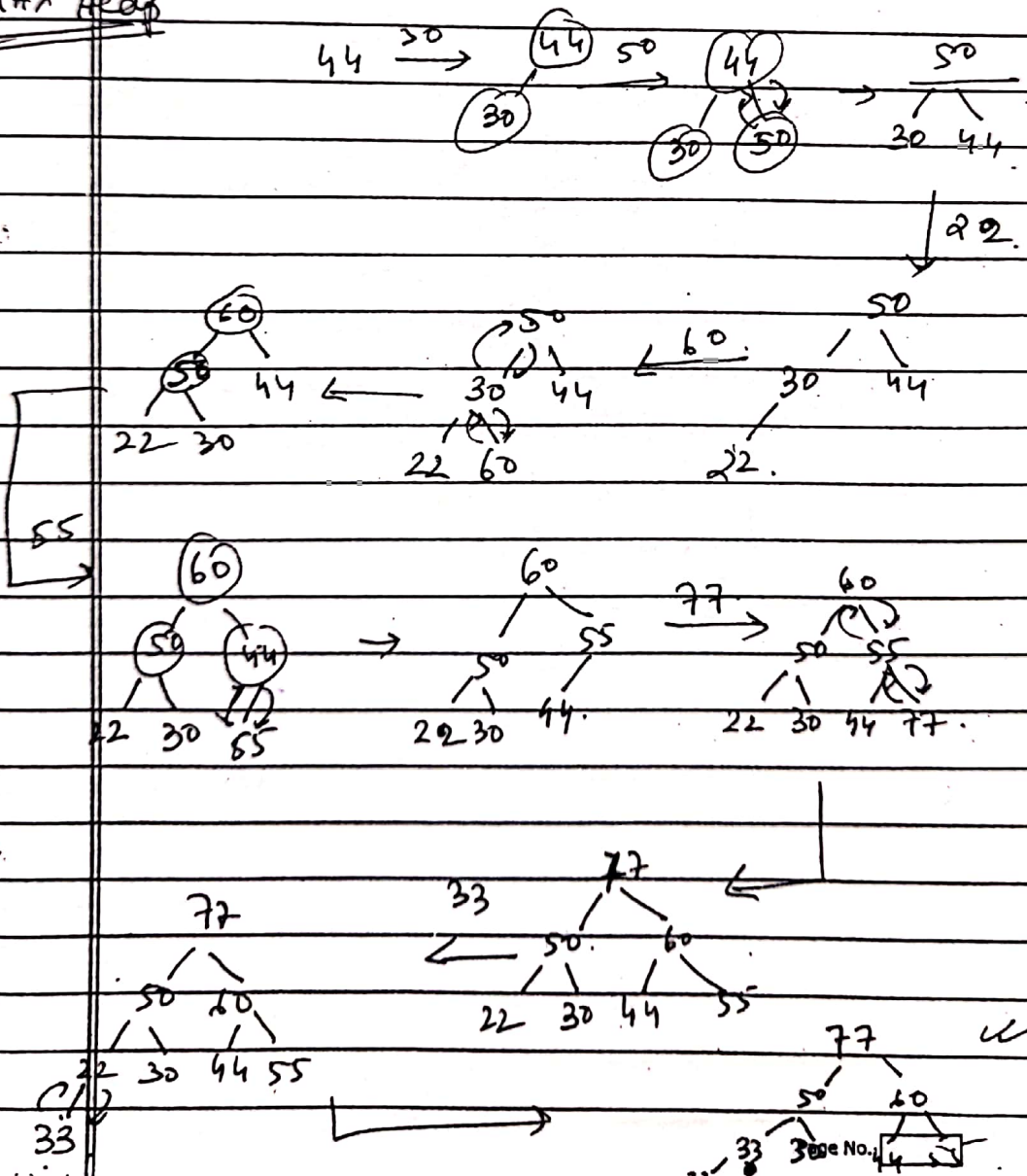
(Max/Min) Heap Sort → Heap tree + Sorting.
(Insertion) + (Deletion)

Heapify

① Root should be greater than all its child
② Left-justified tree

44, 30, 50, 22, 60, 55, 77, 33.

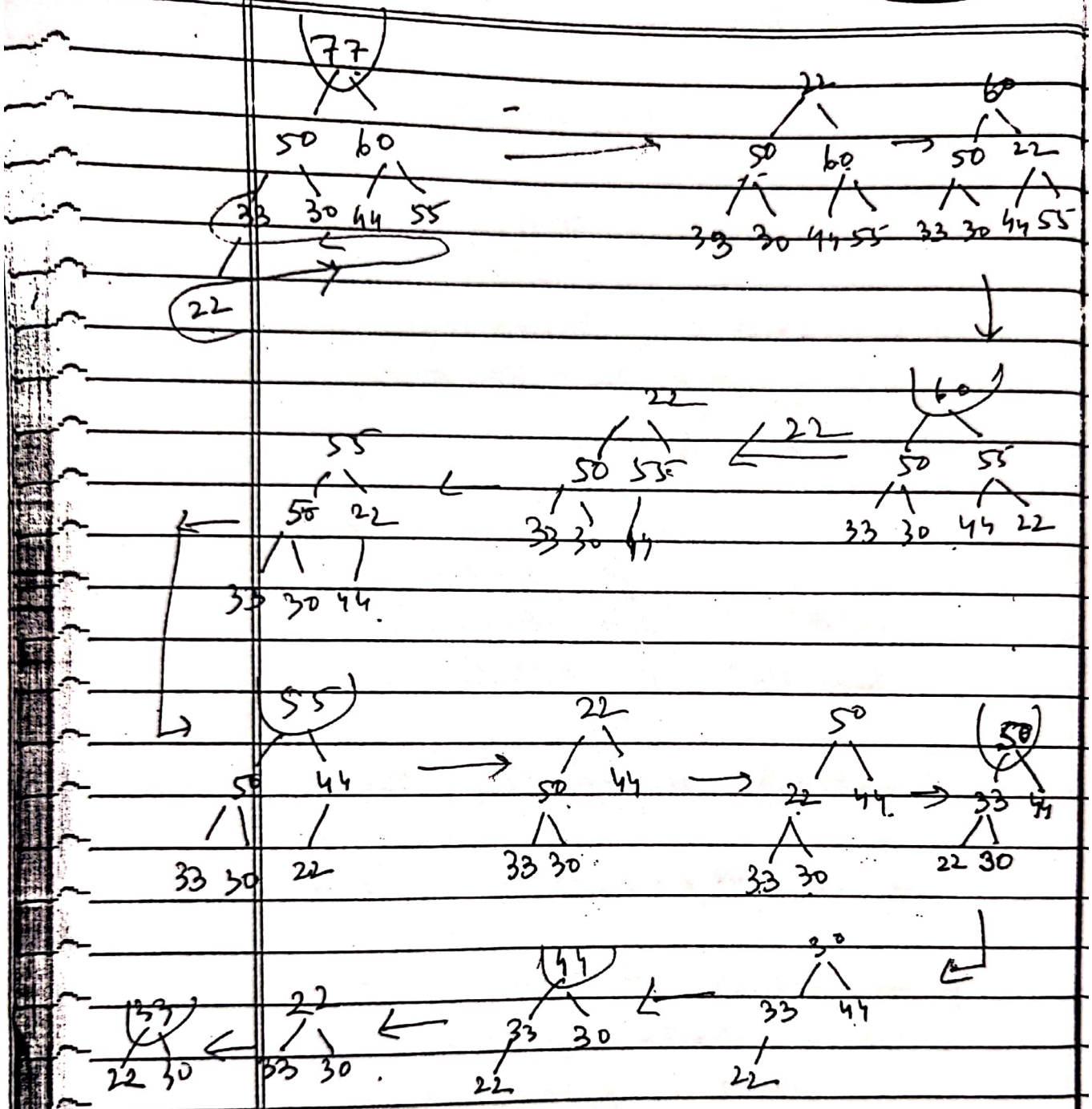
MAX Heap



Page No.

Date Deletion

Saathi



20	30	33	44	50	55	60	77

Ascending.

77	60	55.

Descending.

Page No.

Quick Sort → Divide & Conquer → Merge

① Pivot → ② Partitioning ↓ → Sorted

Date

Saathi

66 33 40 22 53 88 60 11 80 20

Pivot

Left pos.

Right

Right to Left

20 33 40 22 53 88 60 11 80 66

Left

right pos

Left to right

20 33 40 22 53 66 60 11 80 88

Left pos

Right

Right to Left

20 33 40 22 53 11 60 66 80 88

Left pos.

Right pos.

Left to right

Saathi

Date __/__/__

HASHING (CONTINUED)

3- FOLDING METHOD

eg→ key = abcd → // devide in 2 partition
hash func = ab + cd = x — hash value / जो की index आहे

eg → 1234
12 + 34 = 46
21 + 43 = 64 (ba + dc)
1+2+3+4 = 10 // also valid

COLLISION RESOLUTION TECHNIQUE

1- LINEAR PROBING : h, h+1, h+2, h+3, ...

2- DOUBLE HASHING : combination of 2 - 3 hashing methods is called double hashing methods.

x MID-SQUARE ME - The key is squared, the hash func is defined by H(K) = L where L is obtained by deleting digits from both ends of K^2.

x FOLDING ME - The key K is partitioned into a no. of parts $K_1 - K_r$ where

Page No.

each part except possibly the last has the same no of digits as the required address then the parts are added together ignoring the last carry.

$h(k) = k_1 + k_2 \ldots$

COLLISION RESOLUTION TECH.

2- DOUBLE HASHING - In double hashing, 2nd hash fune h' is used for resolving the collision, $h, h+h', (h+h')+h'$

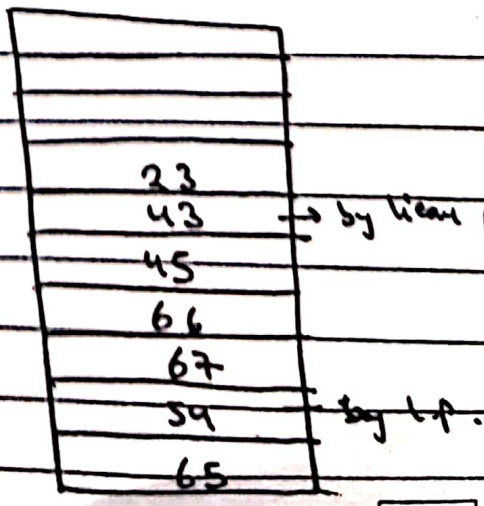
3- QUADRATIC PROBING - Suppose a record R with key k has the hash address $h(K) = h$, then searching the location with address with $h, h+1, h+4, h+9, h+16 \ldots\ldots$

Q- 23, 45, 43, 67, 66, 54, 65, store in array using div method, with linear prob.

⇒ hash fune ⇒

$H(k) = k \bmod 10$

$23 \bmod 10 = 3$

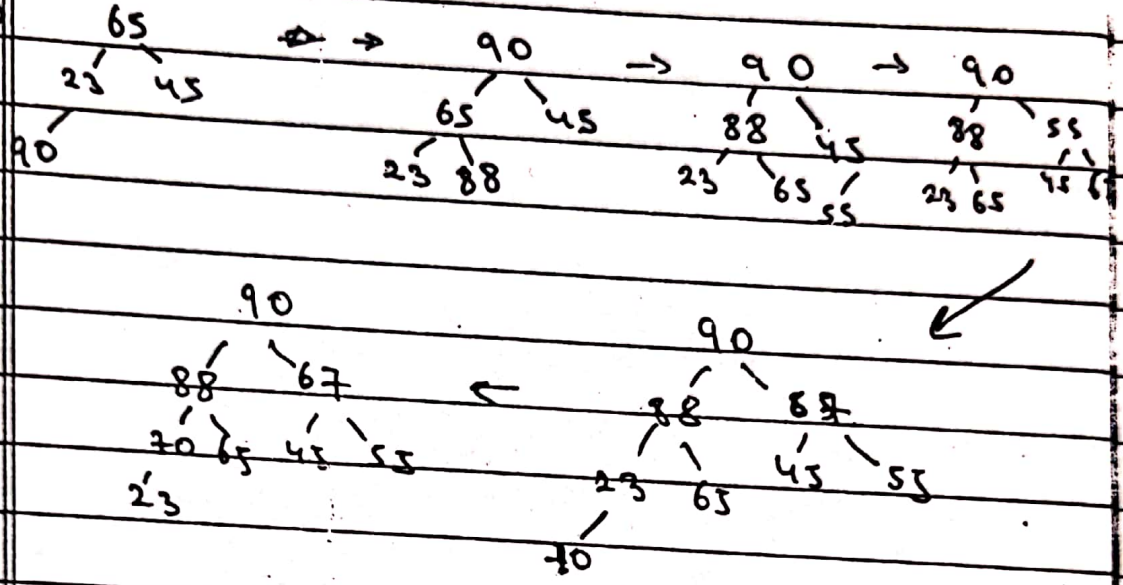


Page No.

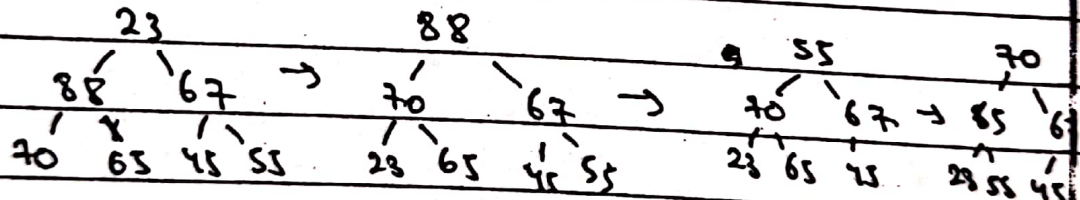
HEAP SORT DEFINITION

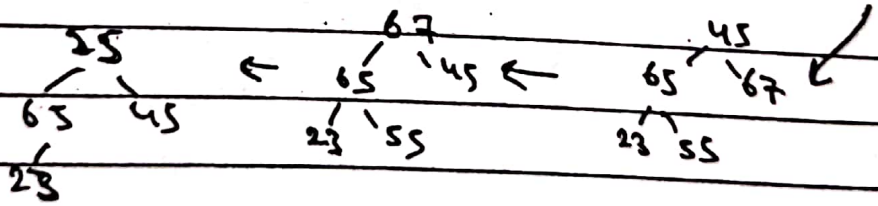
eg→ 65 23 45 90 88 55 67 70

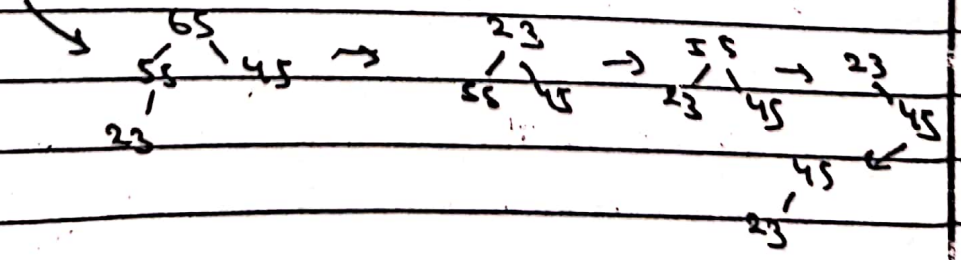
⇒



deletion







90	88	70	67	65	55	45	23

Page No.