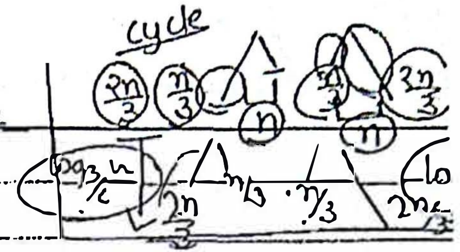
Made Easy Class Toppers Latest Notes Data Structure(Volume 1)
Copyright©Theorypoint.com

Data Structure

Hashing



1. It is a searching Technique.
2. Worst case, TC = O(1)

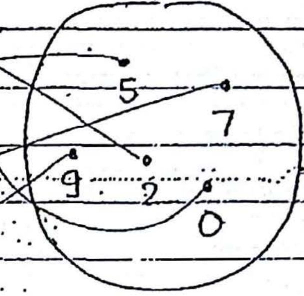
Array means continous m/m

Direct Address Table (DAT) –

x can be accessed directly HT[x].
⇓ to search x, ⇒ O(1) time is required
(No need to search at any place just retrieve the value)

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

size of table m = 10



Hash Table (HT)

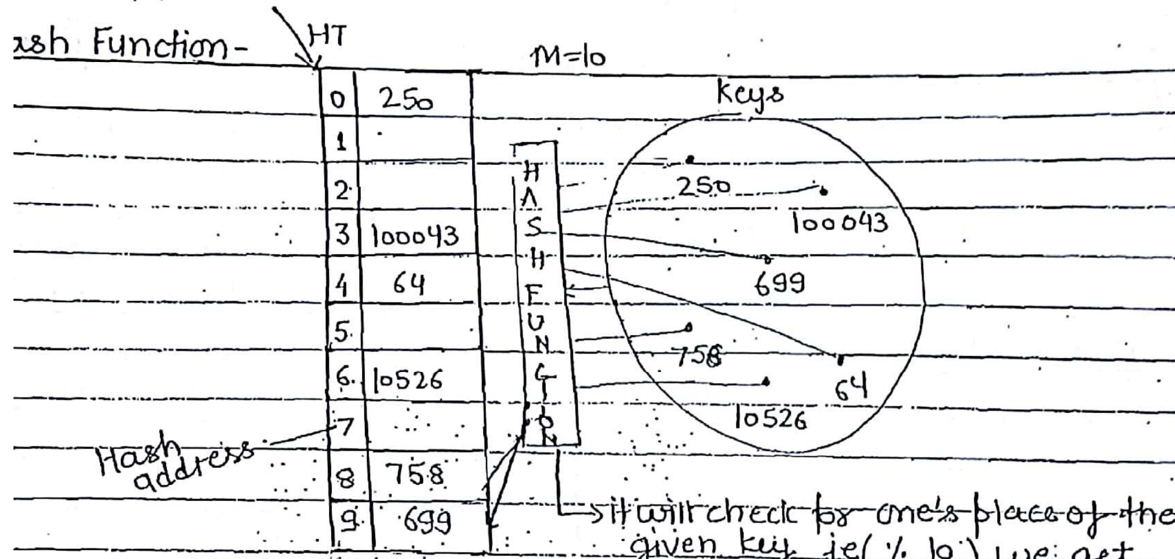
1. In direct address table (DAT), key itself is the address without any calculation.
2. Worst case searching time is O(1) [BC, AC, WC].
3. The largest key no of slot is required ie if you want to store 100000 you must have (100,0001) slots so greatest disadvantage. so for storage of 6 keys, 100001 slots are required.
4. the disadvantage with DAT is, eventhough no of keys are very less but one of the key may be very large(2^{1000}) then the required hash table is of size 2^{1000}.

To eliminate above drawback we are moving to hash functions.

- If we have duplicate data then no need to store again & again for searching

(If (-)ve keys were there)?

Hash Function -



→ it will check for one's place of the given key ie (% 10) we get the (0-9) key just store it as 699, 699 % 10 ⇒ 9

to retrieve 758
again % 10 = 8 (758) is stored at 8

∴ Again time complexity = O(1)

Hash function take more time compared to Hash table due to function calculation.

Hash function is better as it also save the space.

∴ n if n < 10 → slot = 0 to 9

n < 20 slot - 0 to 19

Collision - If two keys mapped to same hash address by hash function then it is called collision.

ie Ex - 699 759

Both get collided at 9 pos^n

%11 (is decided by us)

122545

Hash function is better if no. of collision is as min. as pos:

∴ Types of Hash function-

1. Division Modulo Method-
2. Digit Extraction Method
3. Mid Square Method
4. Folding Method-
   (i) Fold Boundary Method.
   (ii) Fold Shifting Method.

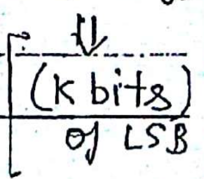
1. Division Modulo Method-

ex1- size of table m = 1000 (0 to 999)
key = 1 2 3 4 5 6 7 8 9
∴ Acc to Division Modulo Method
Hash function (key) = key Modulo m
= 123456789 % 1000
= 789

0	
:	
789	123456789

ex2- size of table m = 8 (2^3) (take LSB 3 bit here)
key = 10100101011100101
∴ Hf(key) = 10100101011100101 mod 2^3
= 101

∴ if $m = 2^k$
then Hf(key)
⇓
(k bits) of LSB



this Method donot bother about the whole data to how to store just take last k^{th} bits-

Ex 1010101111111111111 mod 2^4 = 1111

Ex 3 $m = 2^k$

key = 1010100101011111011101 mod 2^k = LSB k bits

if k = 8

∴ Hf (key) = 11011101

- if M is in power of 2, Hash function of key will be LSB only that lead to more collision.
- So donot choose M as a power of 2.

Ex 1010100101011111011101 k = 7

1010101111111111011101 ∴ LSB

Hf (key1) = 1011101

Hf (key2) = 1011101

1. So, do not pick M as exact power of 2 because if $M = 2^k$ then Hash function of key = LSB k bits always.
   but if M is not power of 2 then less chances of collision

2. Pick the M value which is a prime no but it cannot be close to power of 2

Ques- Which M value can be choosen for better result

(a) 61
(b) 523
(c) 729 ✓
(d) 1024

729 is choosen as it is not power of 2 & not close to power of 2

Steps - (1) Assume all are prime no
(2) choose one which is not close to power of 2.

[encoding - decoding
store - retrieve.

2. Digit Extraction Method- M = 1000

key = 789123456

hashfunction(key) = 7 8 9 1 2 3 4 5 6
(positions 1 2 3 4 5 6 7 8 9)

extract some digits

= 715. extract digit-(1,4,8)

0	
1	
:	
715	789123456
:	

the digits to be extracted are define in question. (you cannot extract 4 digi as size of stack is from 0-999

: Here no consideration is done to storage time.

- When you are extracting 3 digit, may lead to one digit or two digit address (if one of them become zero)

• Collision may also occur here as if

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

but it is not producing that much less collision i.e no of collision is more here.

∴ Digit Extraction Method produces more collision as compa to Division Modulo Method (as for Division modulo, you requ $M = 2^k$) but here only some bits have to be changed.

• We extract 3 digit address as 3 digit addresses >> 2 digit add

3. Mid-Square Method- M=1000
key = 84925

Step 1-
1. Square the given key
2. check Middle of obtained key (take 3 bit from mid as it will have majority of 3-bit addresses.

Making center counter here will be bit difficult, so it lead to less counter & less collision because itself key a larger no produces a more larging no.

Hash function = $(84925)^2$
= 7212(2,2,5)5625

225 255 (as no exact mid possible.
store at any one address

225	7212184925

To Retrieve- (1) Square the key
(2) take the mid, goto mid & extract the key.

Counter is not easy because ~~of large~~ each & every no (digit) participate in the calculation of square

Folding Method-

M= 1000 (0-999)

key= 123456789

(i) Fold Boundary Method

- fold the boundary (take k bit from the key from boundary)

k ≥ no of digit which have more address

∴ for 3 bits - more address

(123) 456 (789)

123
789

913

If it lead to (19)(2)

912 + 1 = 913 or 191 + 2 = 193

folding boundary again & again

∴ key stored

912	123456789

(123)(45)

123
45

168

Counter example = (123) 789 (789)

(913) again collision

counter is bit tougher than bit extraction as 6 digit participate.

(ii) Fold Shifting Method

- Fold it according to fold boundary but fold every bit

(123)(456)(789)

123
456
789
---
(13)(6)(8)

136
8

144

144 - 12345678[illegible]

∴ fold shift is better than fold boundary as it require participation of 9 bit i.e all the bits.

- Counter is also possible.

Ex 789 456 123

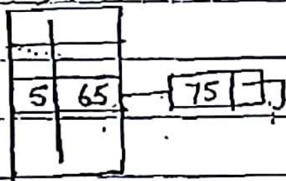
since addn satisfies commutative prop

Collision Resolution Technique-

1: chaining
2: open addressing

chaining

• outside (add linked list)



(If more than m keys are to be kept then chaining is used)

Open Addressing

• when only m keys are to kept
• inside (go to other slots) which are empty
• Linear probing if 75 check for 5 fill 6→ fill and continous checking is done when no empty space

• ~~Quad~~ Quadratic probing whe random jumps are made known as quadratic probing
• Double hashing when two values are directly jump

Chaining Process -

m = 10 (0-9)

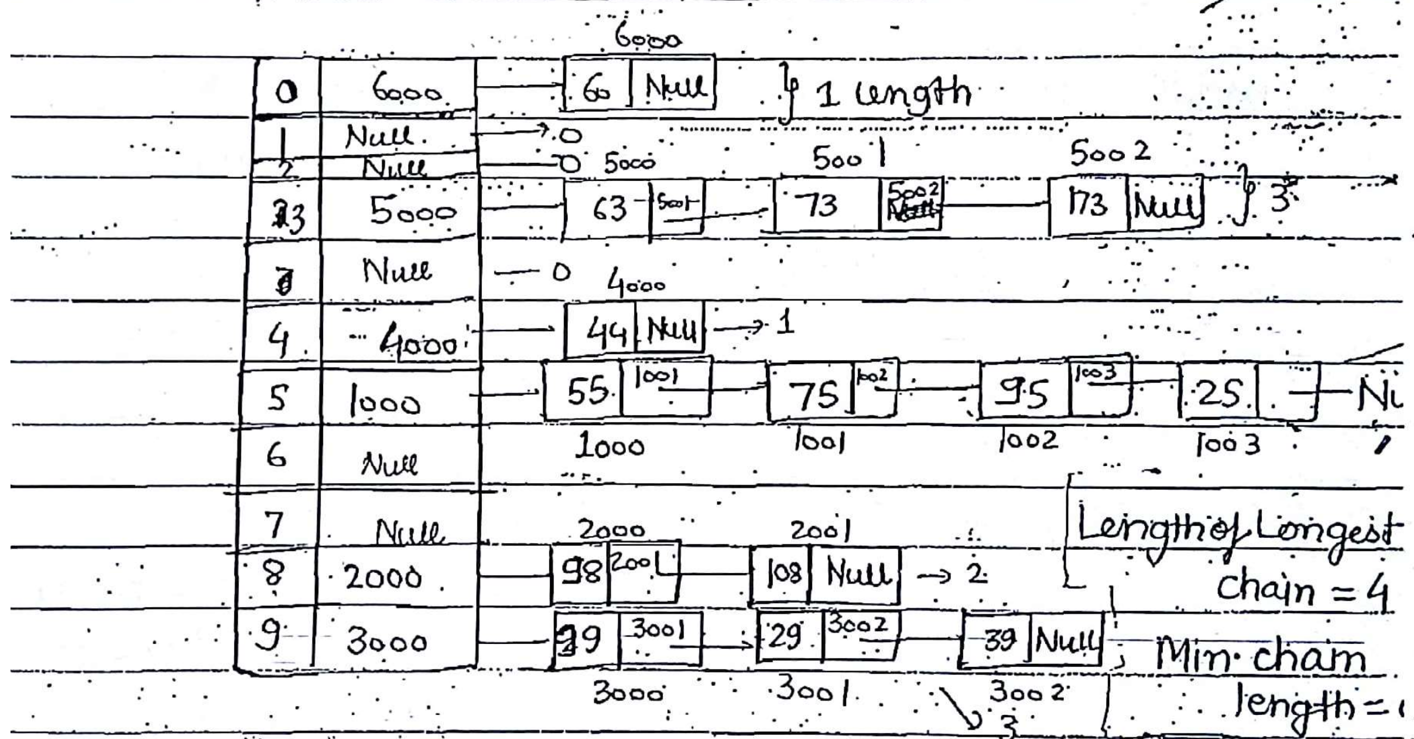
key = (55, 98, 63, 75, 99, 73, 60, 95, 44, 29, 39, 73, 25, 108)

hf(key) = key mod m

Collision Resolution Technique = chaining

0	60	
1		
2		
3	63	→ 73
4	44	
5	55	→ 75 → 95 → 25
6		
7		
8	98	→ 108
9	99	→ 29 → 39

{Slot = 10, key = 14} → ? (Not possi...



Here searching time increases.

∴ in Worst case, TC = O(n) as ~~same~~ if all element occur to 1 slot.

but if you be using better hashing function then it can be reduced as O(1).

~~It is also~~

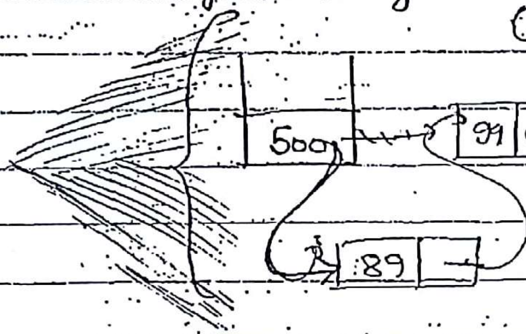
drawbacks

1. In chaining, space is wasted in the form of linked list even the space available inside.

2. The length of the longest chain possible is n. that lead to Worst case searching time = O(n).
   Best Case = O(1)
   Avg. Case = O(n/2) ≈ O(n)

3. the greatest advantage with the chaining is infinite no. of collision can be handled by it i.e infinite no. of keys can be stored because of linked list.

4. Insertion of one key will take O(1) time (BC, WC, AC)
   (as insertion is done in linked list only



n elements.
insertion will take O(1) time.

5. Deletion time will ~~take~~ be - Best case = O(1)
   Worst case = O(n) (as it may be at las[t] so you have to visi[t] each node)

If address of node which has to be deleted is given then
   time - O(1) (BC)
   O(n) - (WC)
   ↳ [since you cannot go back. so you have traversed by start]

is to be deleted & its address is provided with do
t then time to delete = O(1)
= O(1)

~~done~~ in existence as in 10 slot you can store 100
ell.

ddressing-

probing- M = 10 (0-9)
Hashing) Keys = (55, 99, 60, 75, 89, 42, 58, 69, 25)
hash function (key) = k mod m
C.R.T = Linear probing
ing (key, i) = (hf(key) + i) Mod m — i → attempt no
∀ i = {0, 1, 2, ... M-1}

0	60
1	89
2	42
3	69
4	
5	55
6	75
7	25
8	58
9	99

LP(55, 0) = (hf(55) + 0) mod 10 — 0 here signifies 1st attempt
= 5 mod 10
= 5 → (0)

LP(99, 0) = 9 — (0)
LP(60, 0) = 0 — (0)
LP(75, 0) = 5 (fail collision
LP(75, 1) = 6 (pass) ⇒ (1)
LP(89, 0) = 9 (fail)
LP(89, 1) = 0 (fail) ⇒ 2
LP(89, 2) = (pass)
LP(42, 0) = 2 — (0)
LP(58, 0) = 8 — (0)
LP(69, 0) = 9 (fail)
LP(69, 1) = 0 (fail)
LP(69, 2) = 1 (fail)
LP(69, 3) = 2 (fail)
LP(69, 4) = 3 (pass) (4)

LP(25, 0) = 5 (fail)
LP(25, 1) = 6 (fail) (2)
LP(25, 2) = 7 (pass)

Total collision = 9

You can store only M elements in M-slots.

To Retreive- same for 75 → goto 5 if there Yes
else (+1)

~~Quadratic~~

Double Hashing- two hash functions - pri

Quadratic hashing (probing)

Quadratic Probing- M = 10

Key (25, 39, 46, 55, 89, 23, 68, 70, 94, 56, 78)

Hf (key) = key mod m) $c_1 = c_2 = 1$

— CRT = QP

$QP(key, i) = (Hf(key) + c_1 \cdot i + c_2 i^2) \bmod m$

quadratic pattern

$\forall\, i = \{0, \ldots M-1\}$

0	70
1	89
2	~~23~~ ~~78~~
3	23
4	94
5	25
6	46
7	55
8	68
9	39

QP(25,0) = 5 (0) collision.

QP(39,0) = 9 (0)

QP(46,0) = 6 (0)

QP(55,0) = 5 fail

QP(55,1) = 7 (pass) (1)

QP(89,0) = 9 fail (1)

QP(89,1) = 1 Pass

QP(23,0) = 3 (0)

QP(68,0) = 8 (0)

QP(70,0) = 0 (0)

QP(94,0) = 4 (0)

Total collision = 2

If we choose li
probe = 4 colli
i.e collision de

QP(56,0) = 6

Q(56,1) = 6+2 = 8

Q(56,2) = 6+6 = 1

~~Q(56,3) =~~

$QP(78,0) = 8$ (fail) $QP(78,3) = $ fail 0
$QP(78,1) = 0$ (fail) $QP(78,4) = 8$ fail
$QP(78,2) = 14$ (fail) $QP(78,5) = 8$ fail

$Q(78,6) = 0$ (fail) $Q(78,7) = 4$ (fail) } 10 collis
$Q(78,8) =$ ~~2 pass~~ fail ~~8~~ $Q(78,9) =$ fail

∴ Total collision = 12

In worst case, m-1 to 0 is covered ∴ O(m) you have to check.

∴ **Worst case = O(m)**

• Quadratic probing is faster than linear probing but it will generate inaccurate result (as gap may be there but you are not able to keep the element).

If Linear Probing is used, if an empty slot is present, element will surely get stored.

In Linear Probing, Worst case searching time = O(m)
Best case = O(1)
Average case = O(m)

Double Hashing:- $M = 10$

keys = (25, 39, 46, 55, 89, 23, 68, 70, 94, 78)

$hf_1(key) = key \bmod m$

$hf_2(key) = 1 + (key \bmod (m-2))$

CRT = Double hashing.

$DH(key, i) = (hf_1(key) + i\,hf_2(key))$ $\forall\ i = (0 \text{ to } M-1)$

0	70
1	89
2	78
3	55
4	94
5	25
6	46
7	23
8	68
9	39

DH(25,0) = 5

DH(39,0) = 9 + 0·( )) = 9

DH(46,0) = 6 + 0( ) = 6

DH(55,0) = 5 + 0 = 5 fail

(1) DH(55, 1) = 5 + 1·hf₂ 55 = 5 + (1 + 55 mod 8
= 5 + 8 = 13 mod 10 = 3

D(89, 0) = 9 (fail)

(1) D(89, 1) = 9 + (1 + 89 mod 8 = 11 mod 10
= 1

D(23, 0) = 3 (fail)

D(23, 1) = 3 + (1 + 23 mod 8
= 11 mod 10 = 1 (fail)

(3) D(23, 2) = 3 + 2(8) = 19 = (9)

~~D(23,3) = 7~~

D(68, 0) = 8 (pass (0)

D(70, 0) = 0 (pass) (0)

D(94, 0) = 4 (pass) (0)

D(78, 0) = 8 (fail)

(2) D(78, 1) = 8 + 1(1 + 6) = 15 mod 10 =

D(78, 2) = 8 + 2(7) = 22 mod 10
= 2 (pass)

Total collision = 7

Problem 1 Apply all 3 open addressing strategies-

m=10

key= 25, 15, 90, 75, 69, 59, 44, 58, 85

(Linear probing)

0	90
1	59
2	85
3	
4	44
5	25
6	15
7	75
8	58
9	69

Total collision = 1+2+2
= 5

- (in linear probing, order of occurrence in sequence matters.)
  if 25 came first, it occupy the first pos^n as 5.

Quadratic probing

0	90
1	75
2	
3	85
4	44
5	25
6	
7	15
8	58
9	69

T collision = 1+2+1 + 10 ... = 23

3+1

- if you want to find any element and if gap occur then element is not present in table.

1. We are not wasting space outside in the form of linked l
2. Searching time = O(1) for Best case
   O(m) for worst case
3. Insertion time = O(1) for Best case
   O(m) for worst case
4. Deletion time = O(1) for Best case
   O(m) for worst case
5. Whenever delete a particular element, place $ otherwise at a time of searching gap occurs & it may lead to mis-searching.

~~if your table~~

5- If you delete one element, it will create trouble to other elements but we can manage with help of $ symbol if more dollar symbol than reHash again.

Quadratic Probing:-

1. No space is wasted outside in form of linked list.
2. Search time ⇒ O(1) = BC
   O(m) = WC
3) insertion time ⇒ O(1) = BC
   O(m) = WC
3) deletion time ⇒ O(1) = BC
   O(m) = WC
4. deletion of one element will trouble to other element but can manage with $, if more deletion, than rehash again.

Double Hashing:-

0	90
1	69
2	~~16~~
3	15
4	44 ~~60~~
5	25
6	
7	59
8	~~75~~ 58
9	75 ~~16~~ ~~69~~

DH(25,0) = 5
DH(15,0) = 5 fail
DH(15,1) = 5+8 = 13 = 3
DH(90,0) = 90
DH(75,0) = 5 fail
DH(75,1) = 5+11

EX3 – $m = 10$

keys = 53, 60, ~~61~~, 82, 91, 80, 90

$hf(k) = k \bmod m$

CRT = linear probing

0	60
1	91
2	82
3	53
4	80
5	90
6	
7	
8	
9	

primary clustering

$LP(80, 0) = 0+0 = 0$
$0+1 = 1$
$0+2 = 2$
$0+3 = 3$
$0+4 = 4$

$LP(90, 0) = 0+0$
$0+1$
$0+2$
$0+3$
$0+4$
$0+5$

Here both start with same addr follow same path unneccessarily.

1 person, 2 person, mth person

$$Avg - (1 + 2 + 3 + \cdots + m) = \frac{m(m+1)/2}{m} = \frac{m+1}{2}$$

Primary Clustering: If two keys same initial hash address they both will follow same path unneccessarily in linear fashion because of this avg. time increases, this problem is known as primary Clustering.

- Linear probing is suffering with primary clustering.

- Avg Case = $1 + 2 + 3 + \cdots + m$

$$= \frac{m(m+1)}{2} \Big/ m = \frac{m+1}{2}$$ as it suffers with the problem primary Clustering

as 2 will suffer with problem of 1.

y using Quadratic Probing

0	60
1	91
2	82
3	53
4	
5	
6	80
7	
8	90
9	

QP(80,0) = 0 (fail)
QP(80,1) = 0 + 1 + 1 (fail)
QP(80,2) = $0 + 2 + 2^2$ = 6 (true)

QP(90,0) = 0 (fail)
QP(90,1) = 0+1+1 fail)
QP(90,2) = $0 + 2 + 2^2$ = 6 (fail)
QP(90,3) = $0 + 3 + 3^2$ = 12 = 2 (fail)
QP(90,4) = 0 + 20 = 0 (fail)
and so on QP(90,5) = 0 (fail)
QP(90,6) = 2 (fail)
QP(90,7) = 8 (true)

Here ~~primary~~ clustering occurs as what path is followed by 1 element will definitely followed by 2 element but here Avg. case ~~deg~~ decrease due to large jump
Here clustering is known as secondary clustering.

Secondary Clustering- If two key contain same starting hash address, they both will follow same path unneccessarily in quadratic manner, because of this reason avg. searching time increases (it is less than linear search probing time), this problem is known as secondary clustering

Quadratic probing is suffering with secondary clustering

Avg case = O(m) but actually it is less than $\left(\frac{m+1}{2}\right)$
(Asymptotically same but mathematically less)

Double Hashing:

80

0	60
1	91
2	82
3	53
4	80
5	
6	90
7	
8	~~80~~
9	~~80~~

DH(80,0) = 0
DH(80,1) = 1
DH(80,2) = 2
DH(80,3) = 3
DH(80,4) = 4

DH(90,0) = 0
DH(90,1) = 3
DH(90,2) = 0 + 6

Here path is diff with initially address (same).

Here diff path is carried out with same initial address. no path will be same.

DH(100,0) = 0.
DH(100,0) = 0 + 5 = 5 (True)

$$Avg = \frac{3 + 5 + 7 + \cdots + 2}{m} = \frac{m}{m} = O(1)$$

(as a particular slot is covered by single element only, (it might be possible that there must be some cases which lead to redundancy there in avg case, we have to consider for all the given case) so on an avg it is said that it is repeating one time).

In Double Hashing: Eventhough two key contain same starting hash address, they both will follow diff path because second level hash fn will give diff value.

because of this reason 99% of redundancy is eliminated.

• Avg case = 3 + 5 + 7 + . . . + 2

(3 → slot covered by I element; 5 → slot covered ~~by II element~~)

∴ on an avg

$= \frac{m}{m} \Rightarrow \frac{cm}{m} = O(1)$ (as only few key will be repeating only

let 4 key repeated

∴ total $= \frac{4m}{m}$

$= O(4)$)

Avg. case = O(1)

∴ In double hashing, secondary clustering problem is present littlebit (1%)

Note - Using Perfect Hashing, you will get Worst case Searching time = O(1).

in hashing if collision occur to a, b, c at slot 5, they will go to 5 which have another hs which is best hash f^n and a hash table to store those three element then collision is removed perfectly.

0	
1	
2	
3	
4	
5	hs
6	
7	
8	
9	

(slot 5 hs →)

0	1	2	3	4	5
	a	b			c

∴ Total table = m+1

Total f^n = M+1

∴ **Worst case = O(1)**

space = $O(m^2)$

You cannot decide of size so use m size

else use linked list

so space increases but time decreases

m-slots n-keys

m slots - will store n keys

1-slots will store $\frac{n}{m}$ keys

Load factor - no of keys getting stored in 1 slot

Load factor (α) = $\frac{n}{m}$ keys per slot

Note 1 - the expected no of probes (attempt) in an unsuccessful search of an open addressing technique is

$$\left[\frac{1}{1-\alpha}\right]$$

Note 2 - the expected no of probes in Successful search of an open addressing technique is = $\left[\frac{1}{\alpha} \log_e \frac{1}{1-\alpha}\right]$

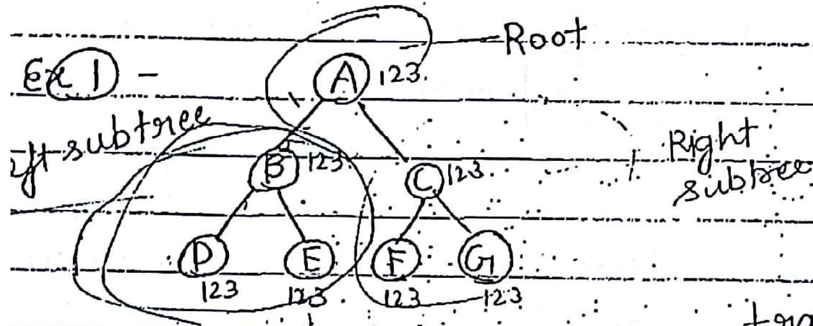
(Coreman)

Tree and Graph Traversals (2 Marks)

Tree Traversals -

1. Preorder (Root LST RST)
2. Postorder (LST RST Root)
3. Inorder (LST Root RST)

Ex 1 -

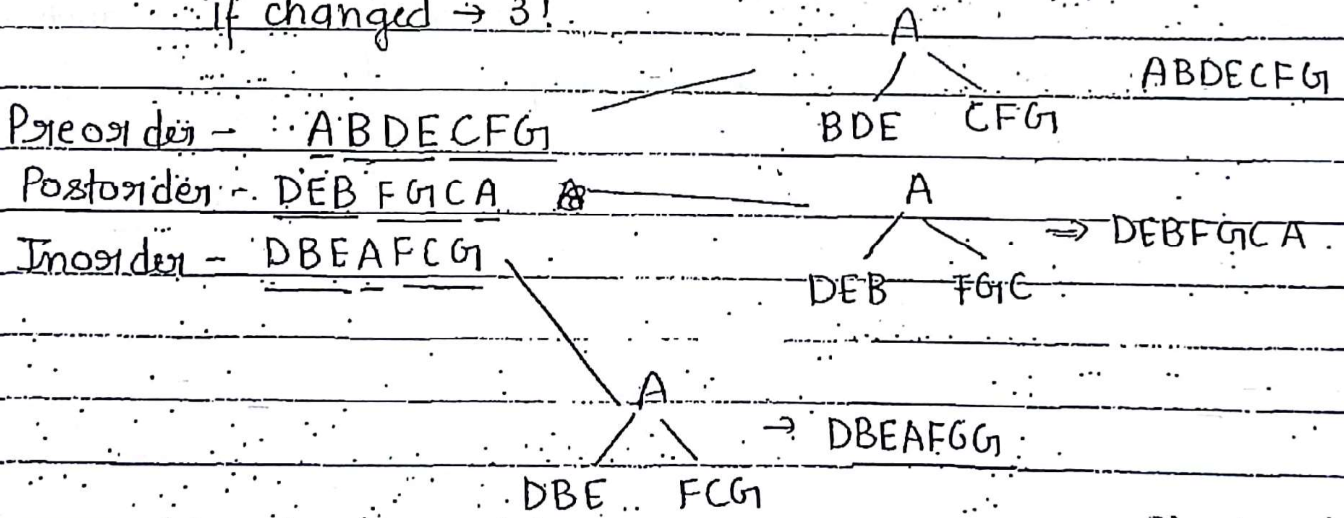


tree is best solved by recursion as of having almost sym quality.

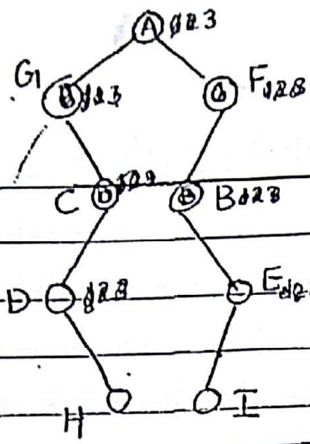
traversal - visiting each node one t in given tree.

Left & Right tree cannot changed i.e left will always come before Right subtree

if changed → 3!



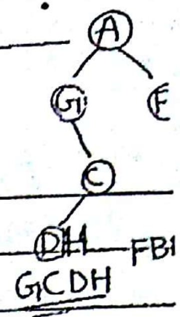
Ex2 -



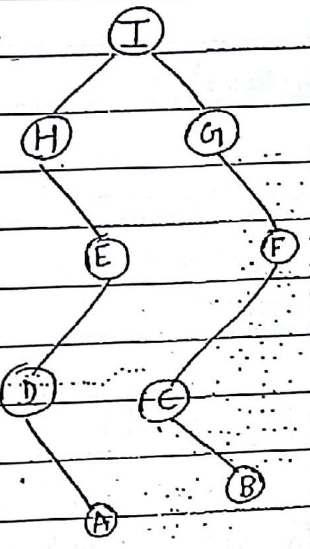
Preorder - AGCDHFBEI

Inorder - GDHC ABIEF

~~Postorder HDCGIEBFA~~



ex3 -



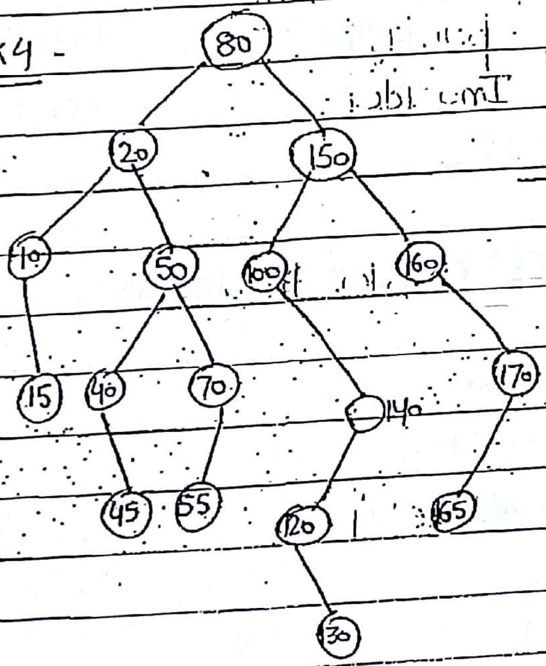
Preorder - IHEDAGFCB

Inorder - HDAEIGCBF

Postorder - ADEHBCFGI

To sort an array
- Create BST
- do inorder traversa

Ex4 -



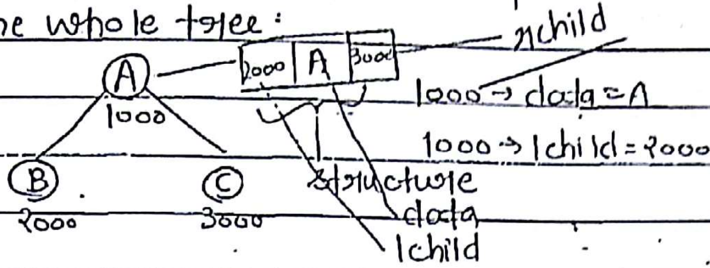
Inorder - 10 15 20 40 45 50 55 70 80 100 120 130 140 150 160 165 170

Preorder - 80 20 10 15 50 40 45 70 55 150 100 140 120 130 160 170 165

Postorder - 15 10 45 40 55 70 50 20 130 120 100 165 170 160 150 80

Note -
BST after inorder traversal will generate Ascending order

- Whenever a tree is given, a root Address is provided which can be used to traverse the whole tree:




```
Preorder(root)
{
    printf(root → data);              → O(1)
    preorder(root → lchild);          → T(n/2)
    preorder(root → rchild);          → T(n/2)
}
```

root = 1000
preorder(1000)
A
preorder(1000 → lchild)
preorder(2000)
and so on.

```
postorder(root)
{
    postorder(root → lchild);
    postorder(root → rchild);
    printf(root → data);
}
```

```
Inorder(root)
{
    Inorder(root → lchild);
    printf(root → data);
    Inorder(root → rchild);
}
```

for preorder of a binary tree

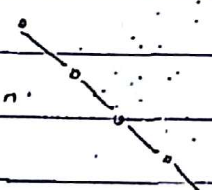
$T(n) = 2T(n/2) + O(1)$ (by master theorem)

$= O(n)$ (Best case)

Worst partition –

$T(n) = T(n-1) + T(0) + O(1)$

$= O(n)$ (Worst case)



Avg case $= O(n)$

for preorder, postorder, Inorder

WC, BC, AC same as each node has to visit once

Reccurence relation can also tell about stack space
If $T(n/2) + \log n$
$T(n-1) = n$

Note -

* Preorder Inorder Postorder will take O(n) time on n-node binary tree (WC, AC, BC), because each node has to be visited once.

~~Stack Space~~ ~~Space complexity~~ = O(logn) for balanced tree (Best case)
O(n) for unbalanced tree (Worst case)

Ques- if a BST is given, time to get sorted order - O(n)

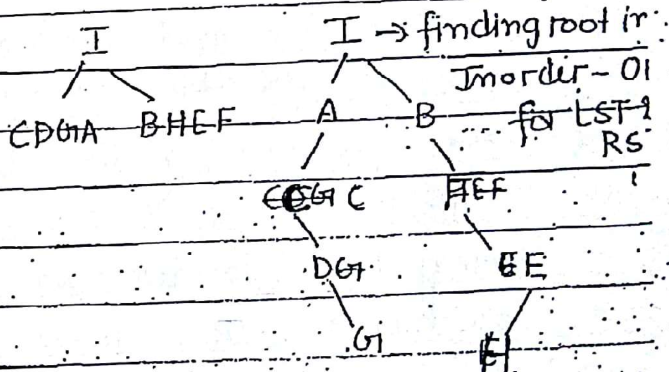
Ques- if a BST is given time to sum element 20 to 50
→ O(n) - Worst case
O(1) - Best case (start from 20 when 20 came & add till 50)

Ques- Consider the following binary tree data-

Preorder: IACDGBFEH

Inorder: CDGAIBHEF

Postorder- GDCAHEFBI



Preorder can give root as well as postorder can give it.

Inorder traversal cannot judge for root because trees are not balanced always

Inorder is used to give Left subtree & right subtree

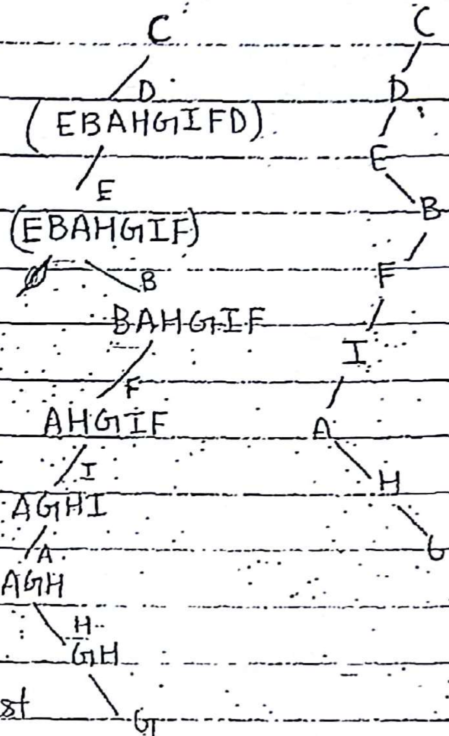
∴ Total complexity = $n \times O(n)$ [(for LST & RST) at 1 key]
$= O(n^2)$

Ques- Consider the following binary Tree data-

Postorder - GHAIFBEDC

Inorder EBAHGIFDC

preorder - CDEBFIAHG



(X = sometime possible)

To find out directly any order
- find out root
- find its posⁿ in inorder case
- the digit or no of element in inorder try to find them in postorder just reverse the order and vice versa.

— a shortcut Not applicable everywhere.

Ex- Here postorder & Inorder given, C its possition in inorder rightmost so C has empty right ST. then go to postorder get the element in LST of C in Inorder. take them & reverse them to obtain post order.

Note :- To construct Binary tree for the given preorder inorder or postorder inorder will take $O(n^2)$ time (n times linear search to find LST & RST in given inorder)

$(n^2 + n + ...)$

- n^2 → to find LST & RST
- n → to find root n time

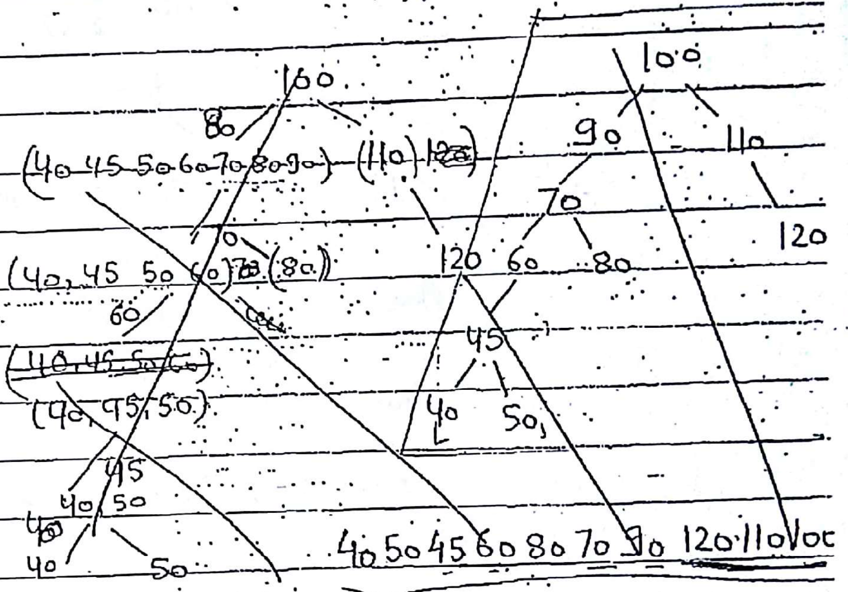
We cannot apply binary search as given sequence is not sorted but if BST it will take nlogn ti

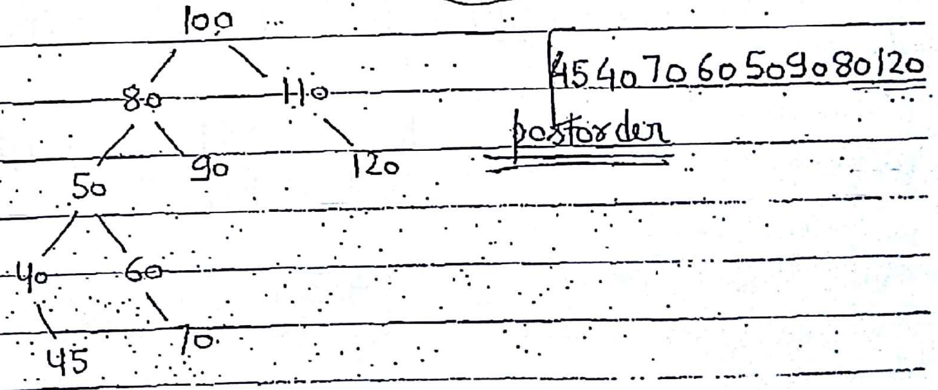
Ques - Consider the following BST data-

Pre 100 80 50 40 45 60 70 90 110 120

Post: ~~90 70 60 45 40 50 80 120 110 100~~

In- (40 45 50 60 70 80 90 100 110 120 (A.O)





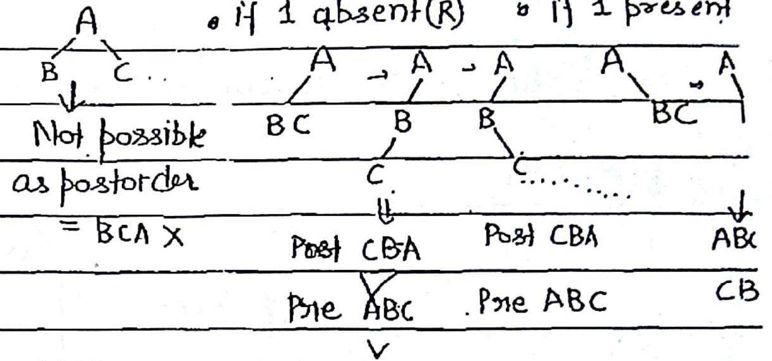
Note - for the given preorder inorder for postorder inorder to construct equivalent BT require (nlogn) time if inorder is already sorted

preorder: ABC
NLR

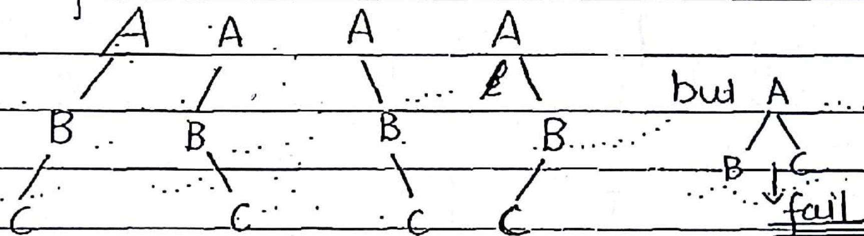
postorder: CBA
RLN

root = A

• Let both RST & LST present



So possible trees are



Note

If Preorder Inorder given - unique binary tree pos / Binary tree Unique BT
postorder Inorder given - Unique Binary tree

if preorder postorder given - more than one binary tree p
→ Binary tree posn but not u

To construct unique binary tree, Inorder is required (compulsory)

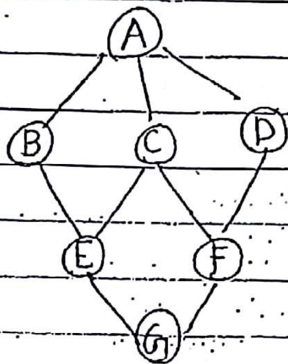
Q: When pre, post is given, manual checking is done To each node two possibilities either going to left or right so total possibilities for n node = $O(2^n)$
↳ upper bound as root will have only one possibility.

To find out cycle initially flag=0 then if visited flag=1 if again counter flag 1 then it is a cycle.

Graph Traversal

1. BFT (Breadth First Traversal)
2. DFT (Depth First Traversal)

Total Graph Traversal time = $O(V+E)$
⇓ visiting each vertex & each edge



We donot consider edge in tree & edge is less than vertices

Tree = $(V+E)$
$V+V-1$
$= 2V-1 = O(V)$

Breadth first traversal - ABCDEFG
Depth " " ABEGCFD

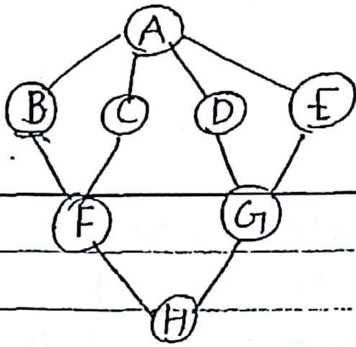
Breadth first Traversal - • Tree traversal are unique but graph traversal not.

• Visited is required as graph contains cycle and there is no mean of visiting a vertex again & again.

• Queue is used in BFT as we need FIFO structure

while loop is covering vertex & for loop is caring for edges.

Ex



Initially

Visited = | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | (A B C D E F G H)

Queue empty — no of vertices

Starting with A

Visited = | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | Q: | A | | | | |

x = A (delete)

w = B C D E

Visited = | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | (A B C D E F G H) Q: | | B | C | D | E |

B deleted

w = A F — A not added again, F added

Visited = | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | (G H) Q: | | | C | D | E | F |

C → deleted w = A.

Visited = | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | Q: | | | | D | E | F |

D → deleted w = A

Visited = | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | (G) Q: | | | | E | F | G |

E → deleted w = A,

Visited = | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | Q: | | | | | F | G |

F → deleted w = B, .

Visited = | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | Q: | | | | | | G | H |

Q: | | | | | | | H |

Q: | | | | | | | |

empty

loop end

```
BFT(V)
{
  Visited[V]=1
  add(V,Q);
  while (Q is not empty)
  {
    x=delete(Q)
    print(Q)
    for all w adjacent to x
    {
      if(Visited[w]=0)
        Visited[w]=1.
        add(w,Q)
    }
  }
}
```

O/P → A B C D E F G H

(order of deletion of element)

Note –

- To implement BFT, we use queue data structure.
- Total complexity = $O(V+E)$ (time) (BC, WC, AC)
  [if adj matrix is used $TC = O(V^2)$]

Space → i/P + extra

↓ $(V+E)$ ↓↓ $(V+V)$ — if graph is adjacency list

$(V+V)$ → queue, visited array

$= (V+E+2V)$

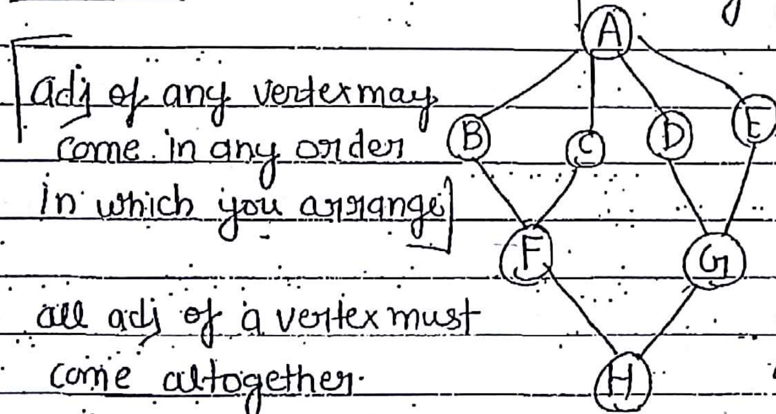
$= 3V+E$

$= O(V+E)$

[$= O(V^2)$ if adj. matrix is used]

- BFT is also known as Level order traversal i.e level by level printing of nodes. (means adjacency list)
  A → next level (adj of A)

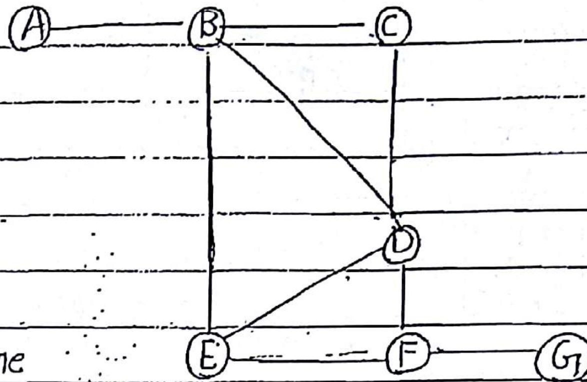
Problem 2 – Consider the following graph–



Find correct BFT's from the following.

1. ABCDEFGH (correct)
2. CAFBDEHG (correct)
3. ADCBEGFH (Correct)
4. EAGCDFBH (false)
5. ABCDEGHF (Not correct)

Ques - consider the following graph.



Whether these are correct BFTs or not

(a) EBDFCAG (correct)

(b) GFCEDBA (incorrect)

(c) CBDAEFG (correct)

(d) DBCFEAG (Correct)

Application of BFT -

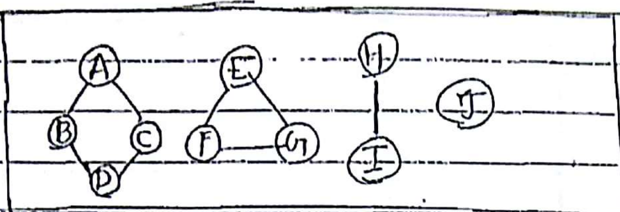
Ques - if a graph is given, how to check whether it is connect or not?

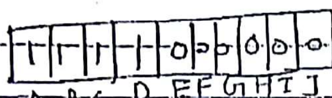
- ~~Draw~~ Apply BFT
- Check visited array
- if any 0 in visited array
- then graph disconnected. O(V+E)+V

↓ to check for arr

if a linked list is given, to check whether it is connected?
a linked list is also a graph but directed so apply the same procedure.

∴ We can verify given graph is connected or not.





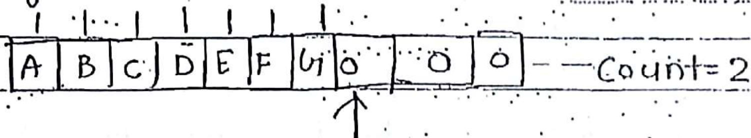
starting from A

→ no of connected component

∴ count = 1 (initially zero)

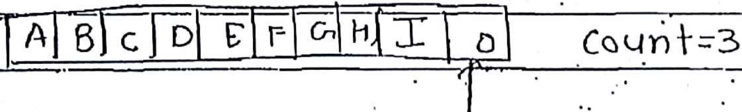
- When you get I zero, just stop & again apply the DFT again and so on, you can find connected connected components can be find out.

again



Count = 2

again



Count = 3

Count = 4.

∴ No of connected component = No of time BFT applied

∴ TC to find connected comp = $O(V+E)$

∴ Using BFT we can find out connected components in the given graph.

- if a null graph $G(V, E)$ - no of connected component = $|V|$

3. to check for cycle in a given graph
- Apply BFT
→ If visited [i] = 1 & it encounter again then there is a cycle.

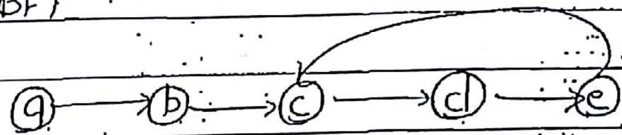
Time complexity = O(V+E) (as BFT work)

4. Whenever kruskal is going on edges are added to MST we have to check for occurence of cycle too as no cycle must occur, then by side BFT also working to check for visited [i] → it take constant time (for 1 edge)

$T(n) = E \log E + (V+E)$
↓ for BFT cycle check

3. Using BFT, we can check given graph contain cycle or not

If a linked list contain cycle, it can be checked easily by applying BFT

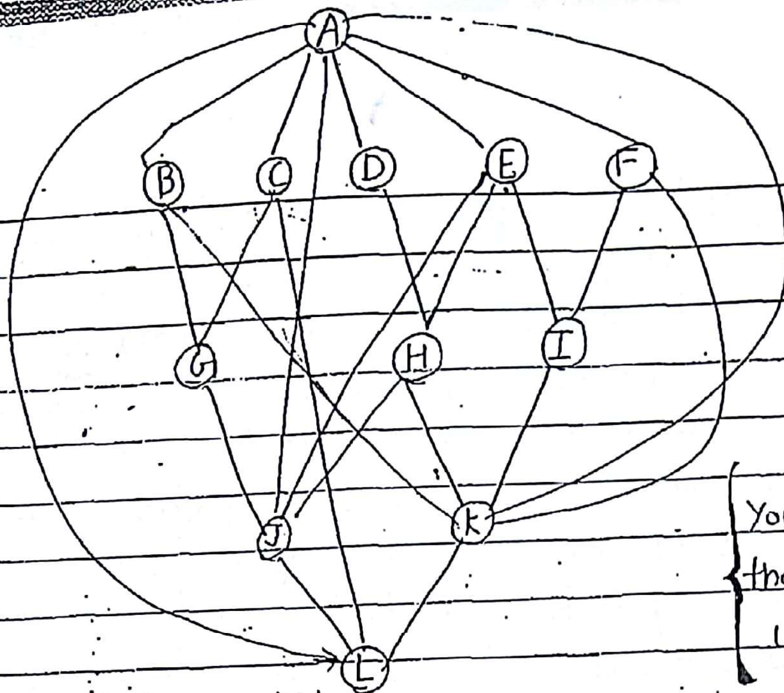


A B C D E C
↑ again occur ∴ cycle exists.

4. If an unweighted graph is given & single source ~~search~~ shortest path is ~~not~~ to be finded, then BFT is the efficient one algo

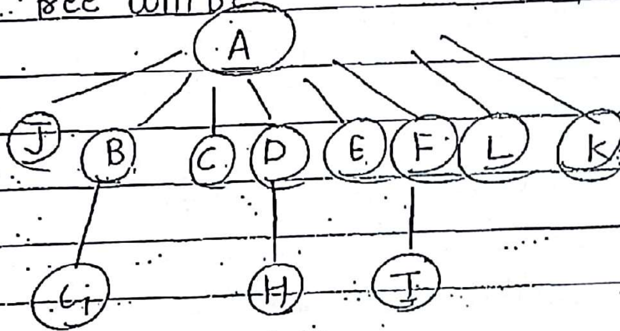
~~as it will find the sh~~

∴ Using BFT, we can find out shortest path from given source to every vertex in given unweighted graph or the graph having equal edge weight (+ve)



You can easily find out the shortest path using O(V+E) time.

the BFT tree will be



unweighted graph - default weight = 1

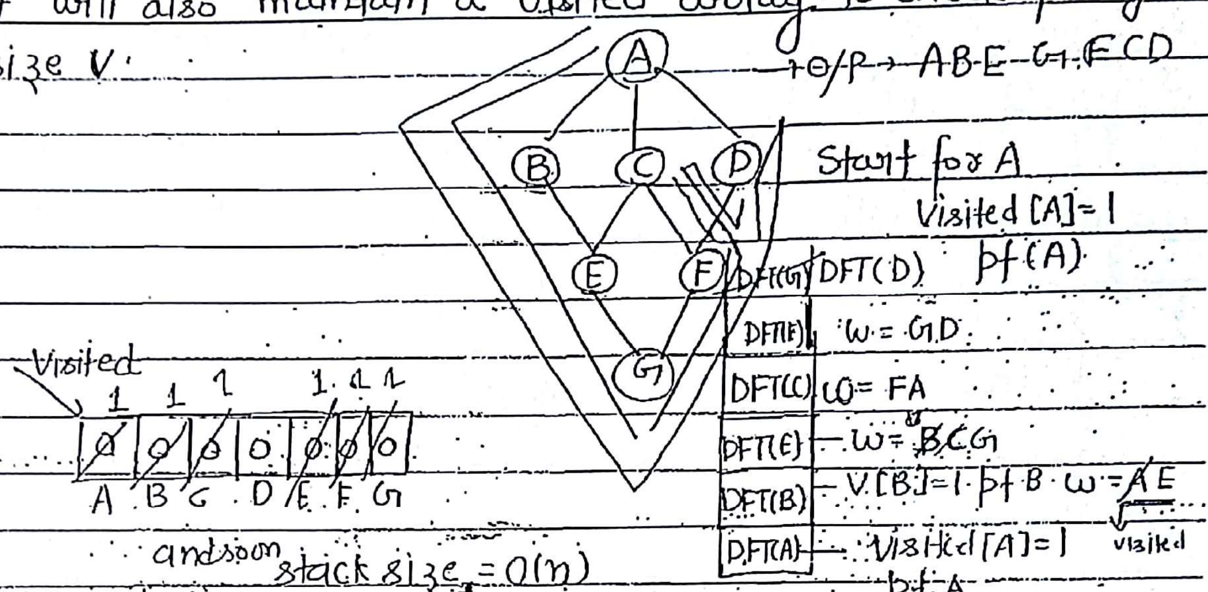
Ques- if a unweighted graph is given; the shortest path to be calculated from given source which data structure is used?

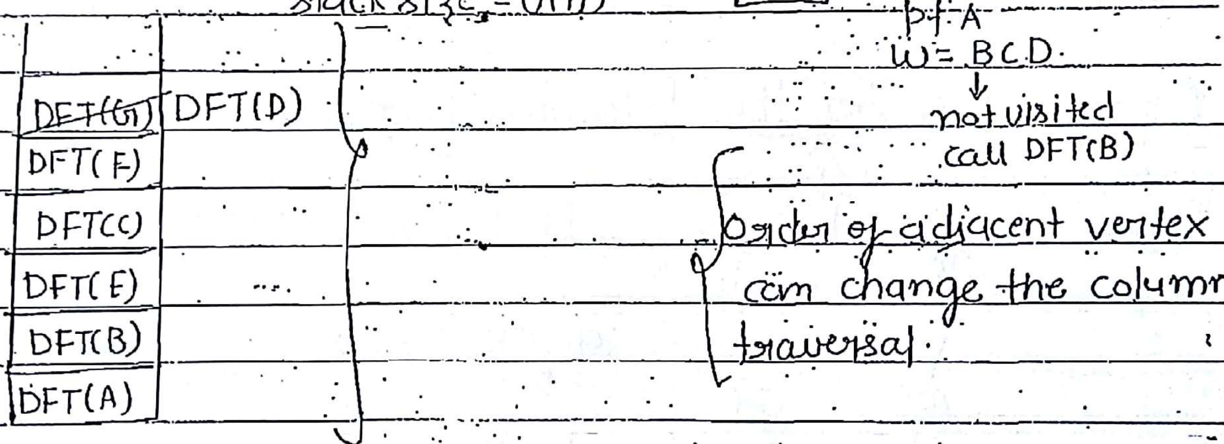
Queue.

5. Using BFT we can verify a given graph is Bipartite graph?

Depth First Traversal

- It uses stack as recursion occur here.
- it will also maintain a visited array to check for cycle. size V.





Order of adjacent vertex can change the column traversal.

1. To implement DFD we are using stack.
2. Time Complexity = no of functional calls * work done at each func call

   = V + E (as if work is summed up)

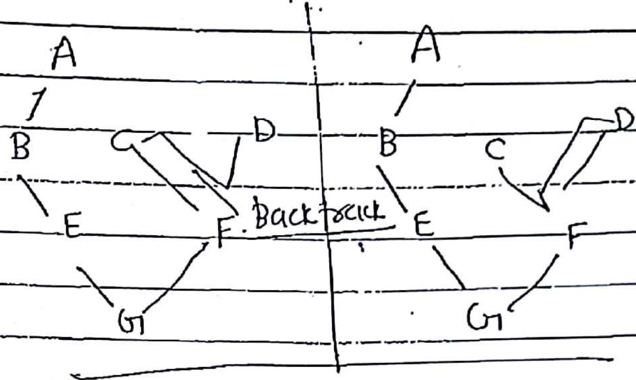
   = O(V+E)

3. Space Complexity = i/P + extra

   = V+2E ↓ stack, Visit = O(V+E)

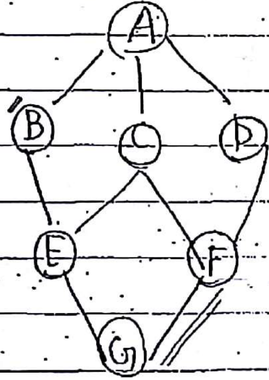
   V+2E V V ↓ for adjacency list

DFT(V)
{
1. visited[V] = 1
2. pf(V);
3. for all w adj to V
{ 1. if (w is not visited)
2. DFT(w);
}
}



{ In case of Adjacency Matrix = $O(V^2)$
(space as well as time complexity) }

Problem - Consider the following graph -



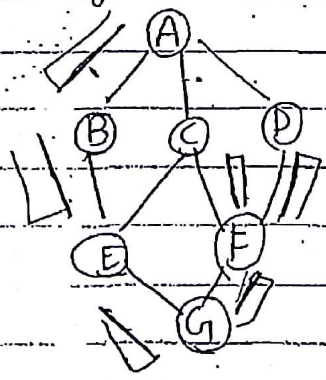
Find Correct DFT's

[You can take any of the adjacent of a particular element.]

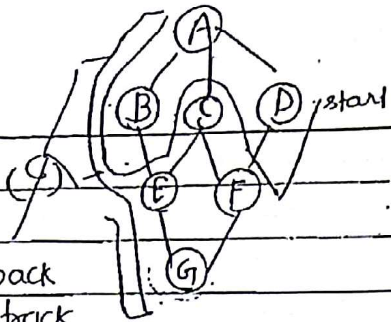
(a) ABEGFCD

No of back track

(b) ABECFDG
↓
backtrack occurs here



{Total Backtrack = 6}



DFCEBAG (correct)

(d) CFDABEG (No Backtrack) (Stack Size = n) — Correct

[max. stack size = O(n)] ↓ when no backtrack occurs

(e) GIECFABD (Incorrect) correct - GIECFDAB

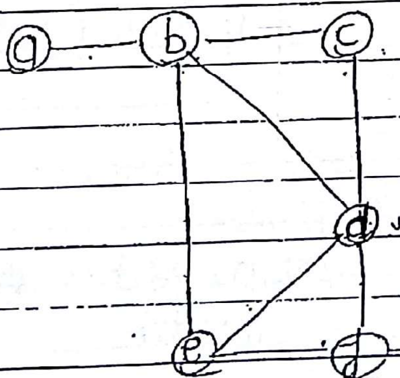
(f) ACFGIEBD (correct)

(h) BACFDEG → (fail) Incorrect) correct - BACFDGE
because at D when possibilities ended we back track to F & F has G to be done but it is first completing G.

Backtracking is done when all possibilities ended.
Backtracking is nothing but poping.

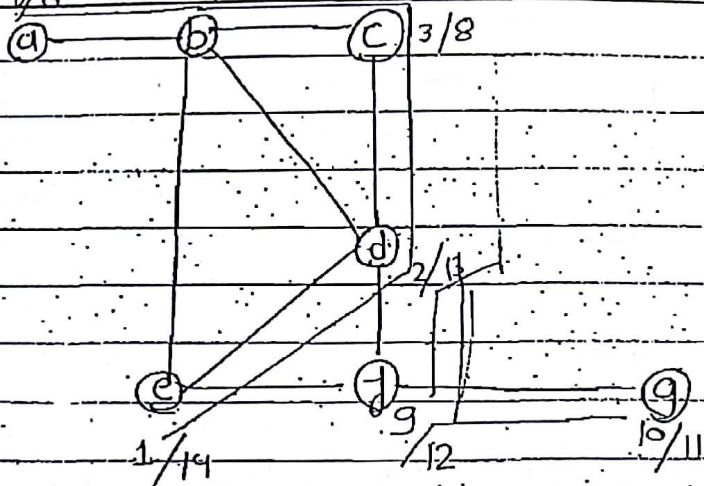
In DFT, if I print D after B, what is the relation b/w then
No relation because of backtracking any thing possible in DFT but BFT have adjacency.

Ques - Consider the following graph -



Check whether they are DFT or ~~BFT~~ or not.

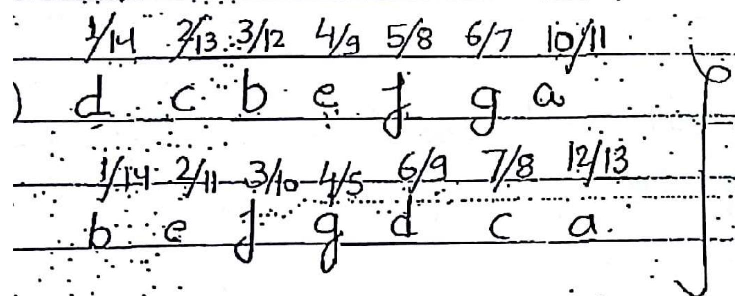
1) edcbafg (DFT)
2) fdebagc (incorrect)
3 abdfgce (incorrect)
4) dcbefga (Correct)
5) befgdca (DFT)



e d c b a f g

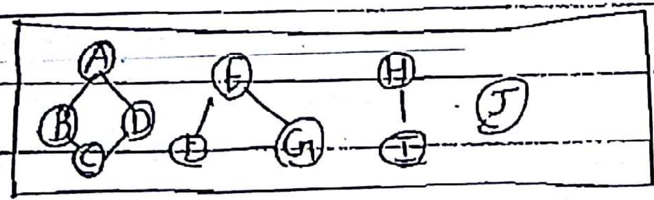
how many element where the diff b/w push & pop time is 1 (Immediate back track)

∴ Total element = 2 (a, g)

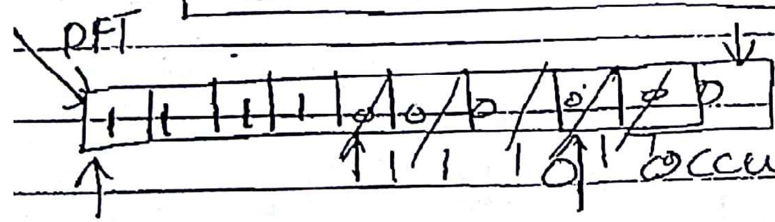


...lications of DFT–

– will check where graph is connected or not.



if start for A



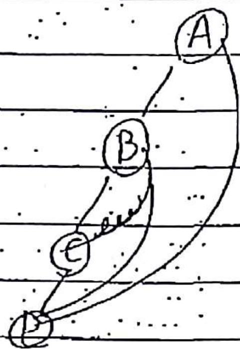
occur in visited array.

O(V+E).

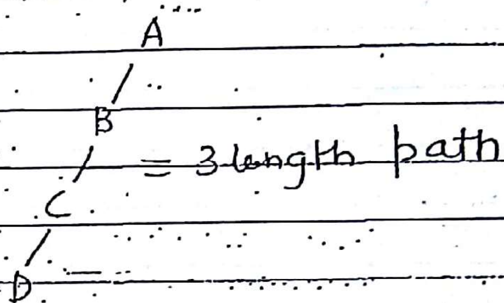
2. We can verify given graph contain cycle or not
O(V+E)

3. We can find out no of connected components
count = no of time DFT applied.

(4)



Here shortest path = (1)
but if we take

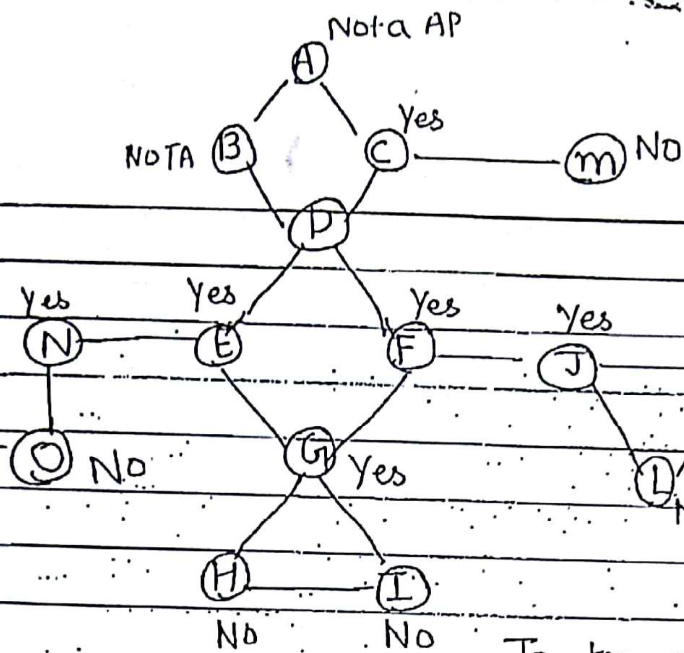


So, DFT cannot (may or may not) work finding single source shortest path for given unweighted graph.

5. [Using DFT, We can check given directed graph is strongly connected or not.?]

6. Using DFT we can verify a vertex is articulation point or not.

Articulation point - if the deletion of an ~~edge~~ ~~leat~~ vertex ~~ale~~ along with its edge lead to disconnection of graph known as articulation point.



To known whether a ~~Graph have~~ vertex is any articulation point -

1) Apply DFT on given graph
2) Delete that vertex & edge.
3) Again apply DFT
4) If Visited contain 1 for all except that delete vertex then that is an articulation point

Total Complexity = $O(V+E) + O(V+E)$
$= O(V+E)$

Bipartite Graph using BFT

first time visited in one set
second time another set
(level by level storing of elements

Strongly Connected using DFT

For finding strongly connected component:
you can use the concept of strongly connected graph
Articulation point.

{ No of articulation point = No of strongly connected component + 1 }

Total Complexity = $E(V+E + V+E)$
→ for articulation point

$= EV + E^2$
$= O(E^2)$

Programming-1

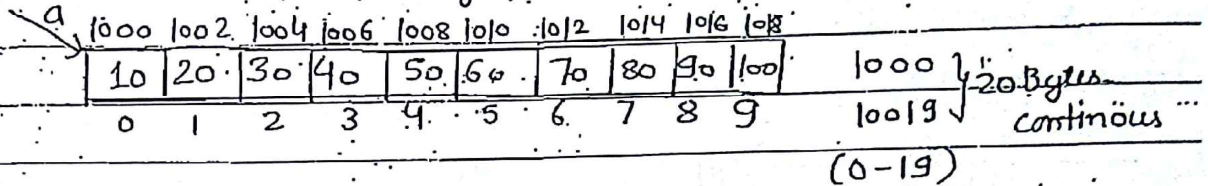
{ before 5 ele from 0 = (5-0) = 5 }

Array- 1-D Array

Ex-1

int a[10] = {10, 20, 30, 40, ..., 100} a will store base address.

↳ declaration (memory creation)



1000 } 20 Bytes continous
10019
(0-19)

1 to 7 → how many elements = 7-1+1 = 7

-5 -4 -3 -2 -1 0 1 2 3 4   4-(-5)+1 = 10 elements

2-(-2)+1 = 5 elements

5 element occupy -1010 & 101

print a = 1000 (base address)

Loc[a[5]] = base address + (size of data type) * (5-0)
= 1010

* 1010 - to access that particular M/M location)

Loc[a[9]] = base address + size of int * (9-0)
= 1000 + 2×9 = 1018

(only 9-0 bco you dont wc to cross 9th element)

Ex2-

a[55 .... 550]   Total element = 550-55+1 = 496

Base address = 990

Size of an element = 10 byte (C)   → before 450 from 55

Loc[a[450]] = Base address + C * [450-55]
= 990 + 10 * 395
= 3950 + 990 = 4940

• array must be continous no matter whether indices are (+)ve or (-)ve.

Ques a[-55 to 55] BA = 1000 C = 5

$Loc[A[5]] = BA + [5-(-55)] \times 5$
$= 1000 + 60 * 5 = 1000 + 300$
$= 1300$

ex4 - a[lb ... ub] BA = Base address
lower bound, upper bound
size = C
total element = ub - lb + 1

$$Loc[a[i]] = BA + C*[i - lb]$$

array index when started from 0, no need to do subtract at time to calculate location but when it start with 1, a subtract has to be done. So less time is required to calculate the value when starting from 0.

Loc[a[1 ... 10] $Loc[a[5]] = BA + (5-1)*C$ → offset calculation
a[0, 10] $Loc[a[5]] = BA + (5-0)*C$

• Array index is, by default, will start from 0 only because no need to calculate offset value

2-D Arrays- int a[4][5] (row, column)

~~(0-3)~~ ~~(0-4)~~ → ~~(5-1)+1 = 5 column~~

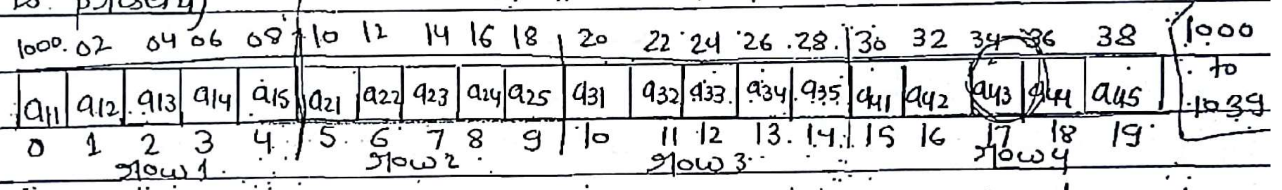
int a[1..4][1..5] =

	1	2	3	4	5
1	a_{11}	a_{12}	a_{13}	a_{14}	a_{15}
2	a_{21}	a_{22}	a_{23}	a_{24}	a_{25}
3	a_{31}	a_{32}	a_{33}	a_{34}	a_{35}
4	a_{41}	a_{42}	a_{43}	a_{44}	a_{45}
	~~a_{51}~~	~~a_{52}~~	~~a_{53}~~	~~a_{54}~~	~~a_{55}~~

total row (4-1)+1 = 4 Row

Total element = 20 element

In m/m 40 bytes for continous m/m (no separate row & column is present)



Row Major order- When elements are getting stored row by row

Ex-

C follows Row Major Order

$Loc[a[4][3]] = 1000 + 2(5*(4-1) + (3-1))$ (visit row, visit col)

17 + 2 = 21

$= BA + C[Column*(x - lb_1) + (y - lb_2)]$

$= BA + C[(ub_2 - lb_2 + 1)(x - lb_1) + (y - lb_2)]$

$Loc\ a[2][5] = BA + c[5*[2-1] + (5-1)]$

$= 1000 + 2[5+4]$

$= 1018$

$Loc\ a[i][j] = BA + C[(ub_2 - lb_2 + 1)(i - lb_1) + (j - lb_2)]$

$lb_1\ ub_1\ lb_2\ ub_2$

$(ub_2 - lb_2 + 1)(i - lb_1)$ → visit correct row.

$(j - lb_2)$ → visiting column

Ex-2 ($\nearrow$ 726 rows, $\nearrow$ 71 column)

$a[25 \ldots 750][80 \ldots 150]$ $c = 10$ byte

Row Major order $BA = 1000$

1) $Loc(a[550][140]) = BA + c[(150-80+1) * (550-25) + (140-80)]$

$= 1000 + 10[71(525) + 60]$

$= 1000 + 10[37335$

$= 373350 + 1000$

$= 374350$

525
× 71
525
3675x
37275

Ex 3: ($\nearrow$ 51, $\nearrow$ 101) $a[-25 \ldots +25][-50 \ldots 50]$, $BA = 0$ $c = 1$ Byte   RMO

$Loc\ a[20][30] = BA + c[(50+50+1)(20+25) + (30+50)$

$= 0 + 1(101 * 45 + 80)$

$= 4545 + 80$

$= 4625$

Ex 4 – $a[lb_1 \ldots ub_1, lb_2 \ldots ub_2]$

$\Downarrow$ $ub_1 - lb_1 + 1$ $\Downarrow$ $ub_2 - lb_2 + 1$

a[47][3]

BA, C

Row Major order $a[i][j] = BA + c[(ub_2 - lb_2 + 1)(i - lb_1) + (j - lb_2)]$ (nc (no of column))

Column Major order $a[i][j] = BA + c[(ub_1 - lb_1 + 1)(j - lb_2) + (i - lb_1)]$ (no of rows)

26-Mar

Column Major Order $a[1 \ldots 4][1 \ldots 5]$

a)	1	2	3	4	5
1	a_{11}	a_{12}	a_{13}	a_{14}	a_{15}
2	a_{21}	a_{22}	a_{23}	a_{24}	a_{25}
3	a_{31}	a_{32}	a_{33}	a_{34}	a_{35}
4	a_{41}	a_{42}	a_{43}	a_{44}	a_{45}

BA = 1000 C=1o

1000	02	04	06	08	10	12	14	16	18	20	22	24	26	28	30	32	34	36	38
a_{11}	a_{21}	a_{31}	a_{41}	a_{12}	a_{22}	a_{32}	a_{42}	a_{13}	a_{23}	a_{33}	a_{43}	a_{14}	a_{24}	a_{34}	a_{44}	a_{15}	a_{25}	a_{35}	a_{45}
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
column 1				Column 2				column 3				column 4				column 5			

Loc[a[4][3]] = BA + C[(3-1) * (no. of Rows) + (4-1)]

= 1000 + ~~10~~ 2(2 x 4 + 3)

= ~~1110~~ 1022

Loc a[2][5] = 1000 + ~~10~~ 2[(5-1) * no of row + (2-1)]

= 1000 + 2 [4 * 4 + 1]

= 1034

both are same (Row Major, Column Major)

if no order is mentioned, go for Row Major (by default)

Ex2- a[-75 ... 125][80 . 150] BA=0 C= 10 byte

CMO

Loc [a[15][130]] = 0 + 10 [(130 - 80) * (125+75+1) + (15+75)]

= 10(50 * 201 + 90)

= 10(10050 + 90) = ~~100640~~

= 101400

$nr = ub_1 - lb_1 + 1$

Ex3 – $a[lb_1 \ldots ub_1][lb_2 \ldots ub_2]$ BA, C, CMO

$nc = ub_2 - lb_2 + 1$

$Loc\ [a[i][j]] = BA + c[(j - lb_2) * nr + (i - lb_1)]$

3-D Arrays – collection of 2-D arrays

$a[3 \ldots 9][15 \ldots 35][8 \ldots 50]$

$9-3+1 = 7$; $35-15+1 = 21$ (no of rows) ; $50-8+1 = 43$ (no of column)

7

$\Rightarrow$ 7 table of size 21×43

$= 7$ [2-D array of size 21×43]

BA, Base Address = 1000   C = 10 , RMO

$Loc\ [a[7][30][40]] = BA + c[(7-3)(nc * nr) + (30-15) * nc + (40-8)]$

$= 1000 + 10(4 * 21 * 43 + 15 * 43 + 32)$

$= 43890$

CMO

$Loc'[a[7][30][40]] = BA + 10[(7-3) * nc * nr * (40-8) * nr + (30-15)]$

$= ~~BA + 10~~$

$= 1000 + 10(4 * 21 * 43 + 8 * 21 * 15)$

$= 1000 + 10(3612 + 168 + 15)$

$= 1000 + 10(3795$

$= 38950$

43
84
172
344
3612

~~Loc~~

Ex ~~a[lb1, ub2~~ $a[lb_1 \ldots ub_1, lb_2 \ldots ub_2, lb_3 \ldots ub_3]$

BA, C

$nr = ub_2 - lb_2 + 1$

$nc = ub_3 - lb_3 + 1$

no of 2D = $ub_1 - lb_1 + 1$

(R.M.O.)

$loc[a[i][j][k]] = BA + C\,[(i - lb_1) * nc * nr + (j - lb_2) * nc + (k - l$

CMO

$Loc[a[i][j][k]] = BA + c\,[(i - lb_1) * nc * nr + (k - lb_3) * nr + (j - lb$

Lower Triangular Matrix -

	1	2	3	4
1	a_{11}	a_{12}	a_{13}	a_{14}
2	a_{21}	a_{22}	a_{23}	a_{34}
3	a_{31}	a_{32}	a_{33}	a_{34}
4	a_{41}	a_{42}	a_{43}	a_{44}

→

a_{11}	0	0	0
a_{21}	a_{22}	0	0
a_{31}	a_{32}	a_{33}	0
a_{41}	a_{42}	a_{43}	a_{44}

⇓ 16 element

6 = zero

$(n/2)$ approx - zero

⇓ to save spa

Why we should store zero

a_{ij} belong to upper part if $\{j > i\}$

} return 0 at that time

Here store in row Major order -

row - 1 will have 1 element.

row - 2 will have 2 element.

ow Major Order-

1000	1002	04	06	08	10	12	14	16	18
a_{11}	a_{21}	a_{22}	a_{31}	a_{32}	a_{33}	a_{41}	a_{42}	a_{43}	a_{44}
0	1	2	3	4	5	6	7	8	9
row1	row2		row 3			row 2			

visit row — visit row colm

Loc(a[4][3]) = BA + c[~~(4+3 3+2+1)~~ ~~(4-1)~~ (3+2+1) + (3-1)]

(sum of 3 natural no.. (4-1) nat no)

= BA + c[sum of 3 natural no + (3-1)]

= 1000 + 2 [6 + 2] = 1016

[All triangular Matrix are square Matrix]

a[25 ... 85, 25 ... 85] (61, 61) (starting row no & colum no is also same)

BA = 1000

C = 10 Byte RMO LTM (Lower Triangular Matrix)

Loc[a[60][55]] = BA + c [sum of (60-25) natural no + (55-25)]

= ~~1000 +~~ ~~(60-25+1)~~

= BA + c [sum of (60-25) natural no + (55-25)]

$= 1000 + c \left[\frac{(60-25)(60-25+1)}{2} + 30\right]$

= 7600

formula

Loc[a[lb1 ... ub1, lb2 ... ub2]] = BA + c RMO, LTM

$$Loc[a[i][j]] = BA + c\left[\frac{(i-lb_1)(i-lb_1+1)}{2} + (j-lb_2)\right]$$

$a[4][3] = BA + c$ [sum of 3 natural no + (total rows − (3−1))]

$= 1000 + 2[6 + 2]$

$= 1016.$

• $a[25 \ldots 85, 25, 85]$ BA = 1000 C = 10 Byte

CMO LTM

Rows = 61 column =

Loc $a[60, 55] = BA + c$ [sum of (55−25) natural no + [no of rows − (55−1)]

$= 1000 + 10[35 \times 18 + 61 - [55 - 25 - 1)]$

$= 1000 + 10[630 + 27$

$= 7570.$

CMO $a[i][j] = BA + c$ [sum of $(j - lb_1)$ natural no + (no of rows − (j − lb

~~extra~~

~~fibonacci heap~~

Loc[a[4][3]) = 1000 + [$\frac{4\times5}{2}$ − $\frac{2\times3}{2}$ + (4−3)] * 2 (1000 = BA)

$\frac{4\times5}{2}$ ↓ crossing all column

$\frac{2\times3}{2}$ ↓ crossing unneccesary column

(4−3) → Visiting 4 throw

↳ 3 row and 3 column

= 1000+16 = 1016

Loc(a[4][1]) = BA + c [sum of 4 nat no − sum of (4 − 1+1) no + (4 − 1)]

it is used to cross all column

sum of 4 nat no ⇓ crossing all column

sum of (4 − 1+1) no ⇓ crossing unneccessary crossed column

(4 − 1) ⇓ Visiting row

= 1000 + 2 [10 − 10 + 3]

= 1006

ex2 − A[25.... 100, 25, 100]   BA = 1000   C = 10B   LTM   CMO

(76 rows, 76 columns)

Loc(A[80][45]) = BA + c [sum of 76 natural no − sum of (100−45+1) + (80 − 45)]

= ~~1000 + 10 [$\frac{76\times77}{2}$ + $\frac{31\times32}{2}$ + 10]~~ ⇓ Visiting 80th row

= ~~1000 + 10 [38×77 + 31×16 + 10]~~

= ~~1000 + 10 [2926 +~~

38 × 77: 266, 266×, 2926

31 × 16: 26

= 1000 + 10 [$\frac{76\times77}{2}$ − $\frac{56\times57}{2}$ + 35]

= 151469

When you are at 15 row col you are at 15 ~~co~~ row.

~~Loc E AE5~~

$A[-50 \ldots 50, -50 \ldots +50]$ BA = 0 C = 1Byte LTM, CM

$Loc(A[25][15]) = 0 + 1$ [sum of 101 nat no − sum of [50−15+ nat no + (25 − 15)]

$= \frac{101 \times 102}{2} + 10 - \frac{36 \times 37}{2}$ ↑ since you at 15 row

(as 15 column will start from ~~15 row at all~~ element before 15 are: 2010

$= 5151 - 666$

$= 4495$

Ex4 $A[lb_1 \ldots ub_1][lb_2 \ldots ub_2]$ BA, C, LTM, CMD

~~Loc E~~

$Loc[A[i][j]] = BA + C$ [sum of $(ub_2 - lb_2 + 1)$ nat no − sum of $(ub_2$ nat no $+ (i - j)$]

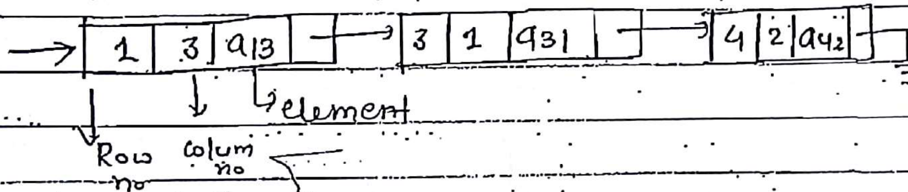
⇓ sum of all column

Sparse Matrix - very less element in array are 1 and other are zero & there is no relation in them

0	0	a_{13}	0
0	0	0	0
a_{31}	0	0	0
0	a_{42}	0	0

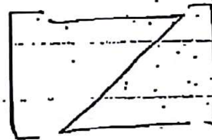
⇒ so need to store in array.

So we used linked list for its storage because if you use array lot of space is required.



Upper triangular Matrix

Z Matrix - all matrix except Z are zero



to exe... | oneo

to

Programming

Scope of a variable
(i) Static Scoping
(ii) Dynamic Scoping

(data type followed by name of variab... – declarati...)

Ex 1 -

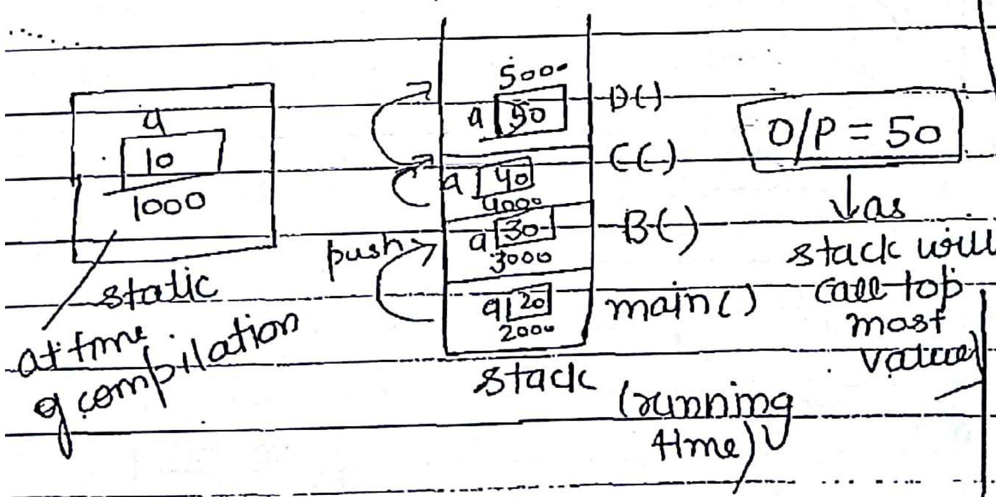
```
int a = 10;
main()
{
    int a = 20;
    B();
}
B()
{
int a = 30;
D();
}

C()
{
int a = 40;
D();
}
D()
{
int a = 50;
pf(a);
}
```

for all f^n call, a single stack is u...

(both static & dynamic as no scope problem) 50

M/M
- stack (for local f^n variable)
- static (at time of compilation)
- heap (dynamic m/m)

Whenever a function is ... memory to local vari... is created & its runn... completed, m/m is deactivated but ... global variable, m... created at time of compilation. the m/m is created is static area)



the space taken by an any f^n in stack is equal to local variable

[static by default - C]

if both local & global data are not there, no mean (Garbage)

free variable - if a f^n is using a variable which is not declared in it.

as if D( )
{
pf(a); [no a is declared but used]
}

(A variable which is not initialized cannot be a free variable)

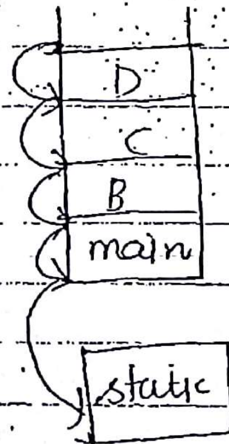
then
O/P = 10 (static)
40 (dynamic)

Whenever there is a free variable, compiler will identify it at time of compilation so, compiler will initialize it by global variable & this problem is known as scope problem & this problem if resolved at time of compilation, known as static scoping because at that time only global variable was initiali

i.e. compiler can't solve problem at time of compi

but if compiler has restricted to solve scope problem at the time of compilation that it is not initialized by global data it will be resolved at run time if 'a' in not in D then go to its calling function & use from there and so on. if no calling function has a, then use global variable 'a'.

- (Dynamic Scope will generate closer result as selecting the value from the closer f^n)



static scoping - problem solve (easier) at compile tim

dynamic scoping at run time (difficult)

because [In dynamic scoping Worst case - all f^n have to be poped to che

- (Dynamic is difficult to implement)

static scoping - by default

C, C++, J

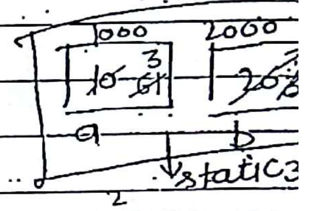
static variable get m/m first & but deallocated at l... i.e. after ... complete ...

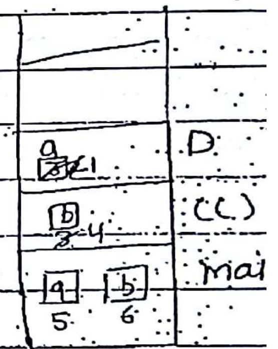
If a f^n as D()
{ pf(a) and so on then static scoping - 10
} dynamic - 40

change done at compile time - static change
Change done at run time - dynamic change (in dynamic changes are m... to previous ...)

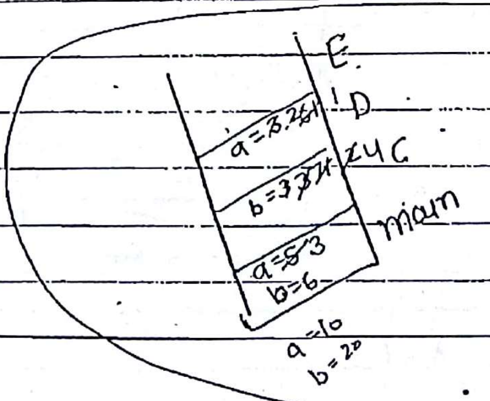
Ex2

```
int a=10, b=20;
main()
{
   int a=5, b=6;
   pf(a,b);
   C();
   pf(a,b);
}

C()
{ int b=3;
   pf(a,b);
   D(b);
   pf(a,b);
   a=3, b=4;
}

D(int a)
{
   pf(a,b);
   a=2;
   b=3;
   E();
   pf(a,b);
   a=1
   b=2;
}

E()
{
   pf(a,b);
   a=61
   b=71
}
```





Static -
5, 6 M
10, 3 C
3, 20 D
10, 3 E
2, 71 D
61, 3 C
5, 6 M

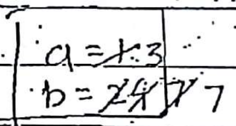


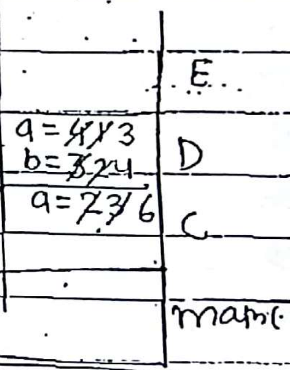
Dynamic -

5	6	5, 6
5	3	5, 3
3	3	~~3, 3~~
3	3	2, 3
3	3	61, 71
5	3	~~5, 2~~
5	6	3, 6

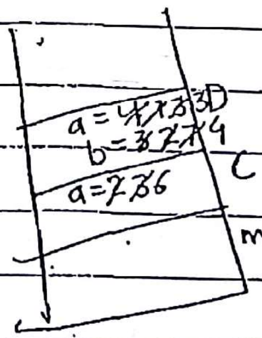
Q-3
```
int a=1, b=2;
main()
{
    pf(a,b);
    C(b);
    pf(a,b);
}
C(int a)
{
    a=3, b=4;
    pf(a,b);
    D(b,a)
    pf(a,b);
    a=6;
    b=7;
}
D(int a, int b)
{
    pf(a,b)
    a=1, b=2
    E();
    pf(a,b);
    a=3 ; b=4;
}
E()
{
    pf(a,b);
    a=3 ; b=7;
    pf(a,b);
```

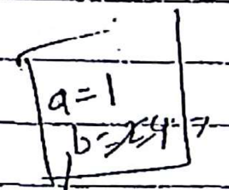
Static Scope -
1,2
3,4
4,3
1,4
3,7
1,2
3,7
3,7





Dynamic
1,2
3,4
4,3
1,2
3,7
3,7
3,4
1,7





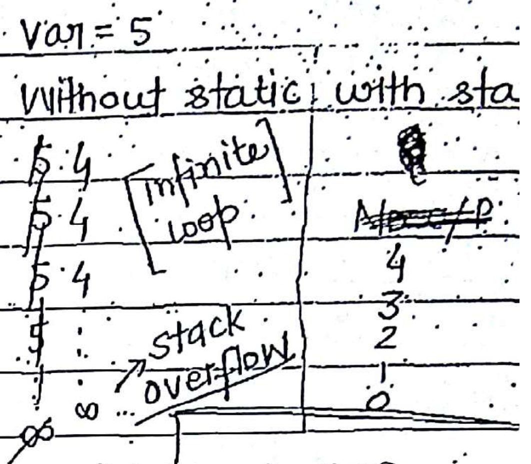
local variable ...
dynamic m/m → heap area

in char - '\0'
~~int '0'~~
float '0.0'
pointer - 'Null'

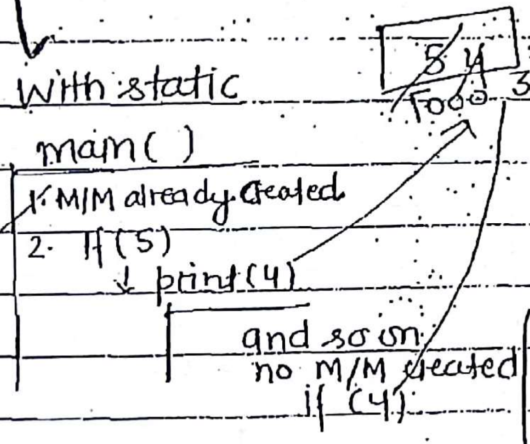
Static Variable-

ex 1

```
main()
{
    static int var = 5;
    ~~printf("%d", var--);~~
    if(var--)  5
    {
        pf("%d", var)
        main();
    }
}
```



- When local variable are declared with static; M/M will be created at Compile time.

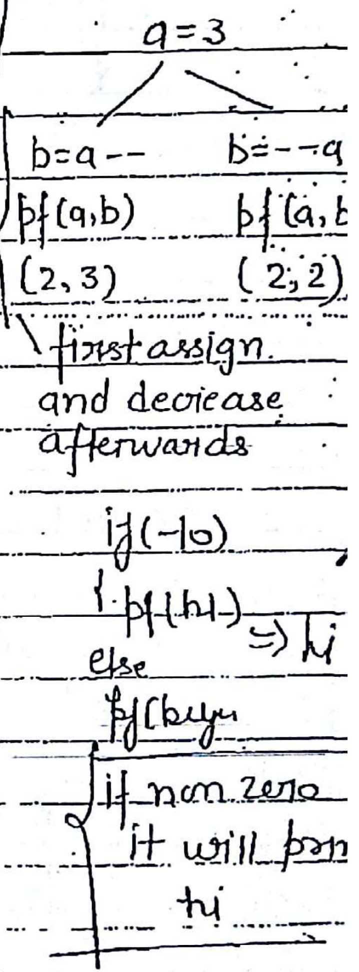


- by default, global are static as m/m created at compile time.

O/P - 4 3 2 1 0
as M/M is created only once.

at end of program
var = -1

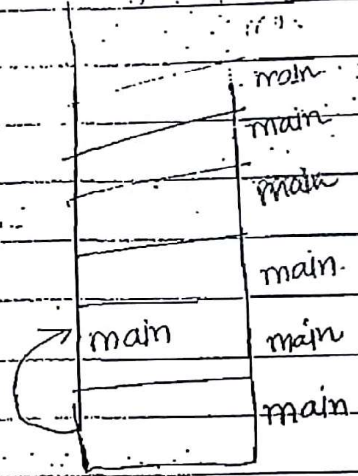
- M/M created i.e static is only one time as compilation is done one time only.



static by default = 0

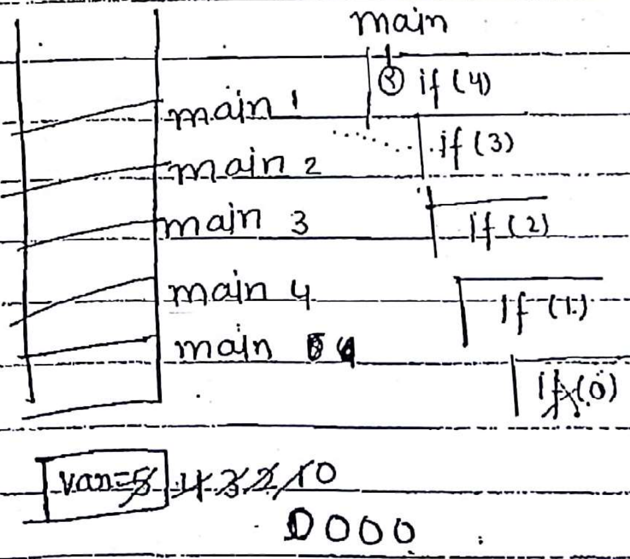
Note 1 - 1. Static Variable M/M will create only once because m/m created at compile time.
2. for static variable, initialization will be done only once.
3. by default static variable will be initialized to zero.

Ex2 -
```
main( )
{
    static int var=5;
    if (var)
    {
        main();
        pf( var--);
    }
}
```



↳ infinite loop (stack over flow)

```
main ()
{
  static int var = 5
  if (--var)
  {
     main();
     pf(var)
  }
}
```



O/P - 0000

```
main()
{
  static int var = 5;
  if(var--)
  {
     main();
     pf(var);
  }
}
```

O/P - -1, -1, -1, -1, -1

```
main()
{
  static int var;
  if(--var)
  {
     main();
     pf(var);
  }
}
```

⇒ O/P → infinite loop

as var = 0
↓
--0 = (-1)
if(-1) ⇒ t

```
main()
{
  static int var;
  if(var--)
  { main();
    pf(var); }
}
```

⇒ O/P - No O/P

```
main()
{
  static int var = 5;
  if(--var)
  {
     pf(var);
     main();
  }
}
```

O/P - 4, 3, 2, 1

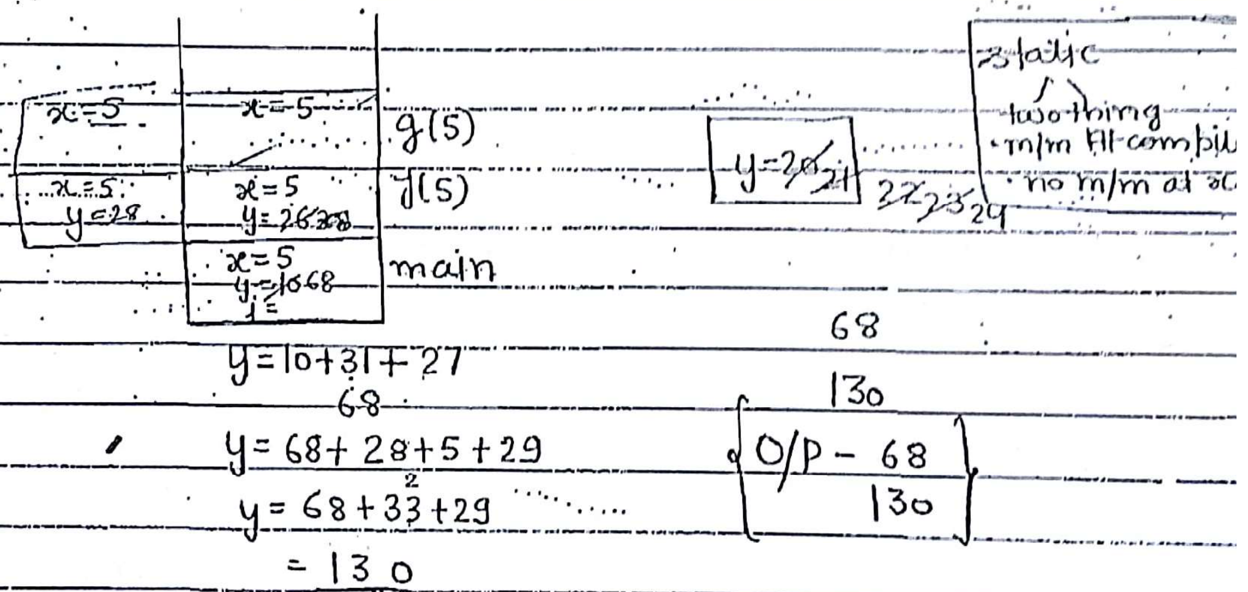
if(var--)
O/P → 4, 3, 2, 1, 0

= (right to left)

static

Ex-4

```
main()
{
  int x = 5, y = 10; int i;
  for (i = 1; i <= 2; i++)
  {
     y = y + f(x) + g(x);
     pf(y);
  }
};

f(int x)
{
  int y
  y = g(x);
  return(x+y);
}

g(int x)
{ static int y = 20;
  y++;
  return(x+y);
}
```



y = 10 + 31 + 27
= 68

y = 68 + 28 + 5 + 29

y = 68 + 33 + 29

= 130

O/P - 68
130

When main() executes completely, then static variable are deallocated its life span is to whole program.

- with same name variable in diff functio
  you can create static m/m.

auto - M/M is created at run time in stack. f^n are by default 'auto'.

Global variable is by default static then no need of putting it.

~~Extern~~ -

```
int a=10;          — global static variable
main()
{
  static int a = 20;   → local static variable [a]
  pf(a);               only main can use it
}
```

Extern - local variable with extern will not take M/M but point to the M/M outside but that variable is local variable but not free variable.

```
int a=10            a [10] 1000
main( )
{
  extern int a; (no m/m point to global m/m)
  pf(a);             O/P=10
}
```

if no extern = O/P = Garbage Value
if static   O/P = 0
if auto   O/P - Garbage value

if extern is there & no global M/M with name a, then it will give error.

ex2 - main()

```
main()
{
    extern int a;      // 1 local variable
    pf(a);
}
```

extern get executed at run time
↓ no memory due to extern
↓ but no m/m outside

then error occur. 'Undefined Symbol a'

ex3 :- main()

```
main()
{
    extern int a;   //no m/m created
    pf(hi);         O/P = hi
}
```

{No error as value of a is never checked}

1 local variable

ex4 -

```
int a = 10;
main()
{
    extern int a;
    pf(a);
    a=20;
    pf(a);
}
```

a: 10 → 20

O/P = 10, 20

```
extern int a;
main()
{
    extern int a;
    pf(a);
    a=20;
    pf(a);
}
```

O/P ⇒ error
Undefined symbol 'a'

ex5 -

```
extern int a=10;
main()
{ extern int a;
    pf(a);
}
```

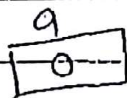
a: 10

O/P = 10

extern variable globally with initialization creates m/m

but extern variable local with initialization give error.

ex5
```
int a;
main()
{
  int a=5;
  int a=6;
  pf(a);
}
```
O/P – error at compile time (semantic error) as no two variable in same fn with same name.

```
int a;
main()
{
  extern int a;
  extern int a;
  pf(a);
}
```


No of variable – 3 (1 global, 2 local)

O/P = 0
as both will point to single m/m so no need to go for error

extern allow us to declare same variable many time bec M/M is not creating.

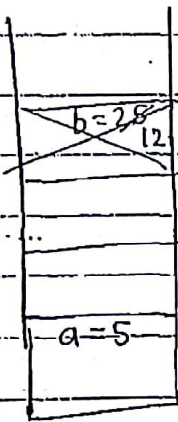
register int a – same as auto but M/M is created in register which make execution faster.

goto:
If you want to get out of the program from stack level 5, use exit

```
int a=1,b=2;
main()
{
    int a=5;
    pf(a,b);
    C();
    pf(a,b);
    D();
    pf(a,b);
}

C()
{
    static int b=7
    pf(a,b);
    D();
    pf(a,b);
}
a=8; b=9

D()
{
    extern int b
    pf(a,b)
    a=4
    b=97
    E()
    pf(a,b);
}

E()
{
    auto int b=25;
    pf(a,b)
    a=11
    b=12
    pf(a,b)
```

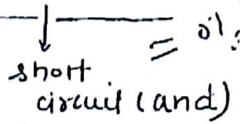
static

a = 1 4 11 8 11
D b = 2 97 97
C b = 7 9
b=7



Static

5,2	5,2
1,7	1,7
1,7	1,2
4,25	4,25
11,12	11,12
11,97	11,97
11,97	11,97
5,2	8,9,97
	8,97
	4,25
	11,12
	11,97
	5,97



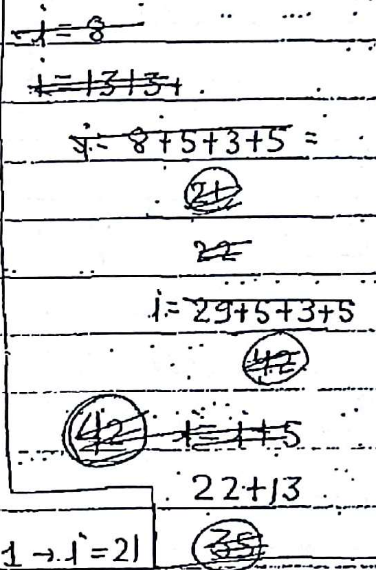
Use Brain
be concious

Int with C and E

Ques- main()

```
{
    int i;
    for(i=1; i<25; i++).
    {
        switch(i)
        {
            case '1': i+=7;
            case '2': i+=5;
            case 3 : i+=3;
            default: i+=5;
            break;
        }
        pf(i);
    }
}
```

Note- In switch statement, After every case there is requirement of break otherwise it will execute remain cases also.
After default, there is no requirement of break as it is the last stmt.



35

i=21 i=1 → i=21

O/P = 21, 27 | O/P-21, 27 | i=22 i=(27)

Ques - main()

```
{ int i=-1, j=-1, k=-1, l=2, m;
    m=((i++ && j++ && k++) || (l++));
    pf(i, j, k, l, m);
```

1 (T && T) && (T) -1 || T (T)

0 0 0 2 1 T

(0 0 0 2 1)

() - left to right associativity

-1 ... -∞, 1 ... ∞ ⇓ True only zero are false

l=2 will remain 2 as no change is done to l due to '1 bracket is (1)
(1 or -) = 1
so no need to check (l

```
main()
{
    int i=-1, j=-1, k=0, l=2, m;
    m = ((i++ && j++ && k++) || (l++))
    pf (i, j, k, l, m)
}
```

m = i++ ·· j++ & k++ then check l++

(i++ → 1, j++ → 1, k++ → 0; together → 0; l++ → 1)

m = 1

O/P = 0, 0, 1, 3, 1

```
if int i=0; j=-1; k=-1; l=2, m;
    m = ((i++ && j++ && k++) || (l++));
    pf (i, j, k, l, m)
```

i → 0 no need to check others (j & k)

check for l

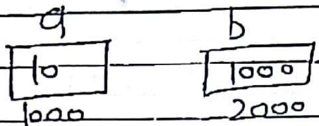
O/P = 1, -1, -1, 3, 1

Pointers- int a = 10 - a 2 Byte M/M is allocated to a to store integer data.

int b = &a - 2B M/M is allocated to b to store address integer data which may go wrong as address may be more than 2B.

int *b = &a - a 8 byte M/M is allocated to b to store add of integer data

⇓ ~~OS ⇒ will take it as declaration to store integer~~ address



whenever you wanna store address use *.

pf (a, b) ⇒ (10, 1000)

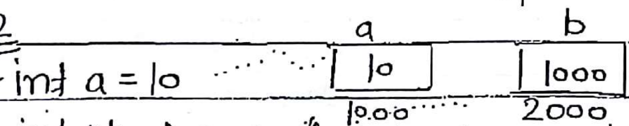
~~pf (a, *b) ⇒ 10, 10~~

⇓ User ⇒ value ~~at~~ point by value stored at

int *b = a may get wrong as a > b then error

- int b = 'a' Yes
- int b = a Yes
- int b = 0.0 No (4 byte cannot be mapped to 2 byte)
  - ⇓ to do this type casting is used.

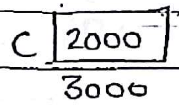
Ex-2



int a = 10
int *b = &a → (for operating system declarations are done.)
int **c = &b (when b is also storing address of another variable)

⇓ 8 byte

2 Byte



double pointer is used when value getting stored also points to another address

** - address of a single pointer.

c → double pointer.

pf (a, b, c) = 10, 1000, 2000

↳ directly ~~we~~ point value stored in a, b, c

pf (a) = 10 (immediate addressing mode)

pf (*b) = 10 (Direct address mode)

pf (*c) = 1000 (indirect address mode)

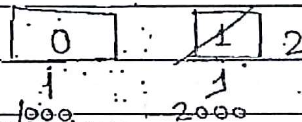
pf (**c) = 10 (Indirect address mode)

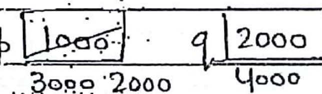
single pointer - *
double ptr - **

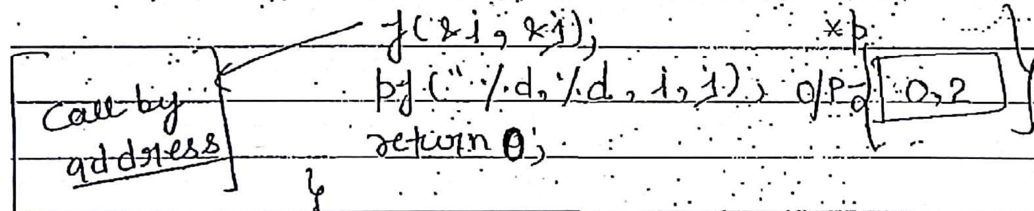
- if b is used with double (**) & c with (***) It will generate an error message
the retrieval of data must be done same way of storage

Problem 1:- Consider the following C program -

```
# include<stdio.h>
void f (int *p, int *q)
{
    p=q;
    *p = 2;
}
int i = 0, j = 1;
int main()
{
    f(&i, &j);
    pf("%d, %d, i, j);
    return 0;
}
```







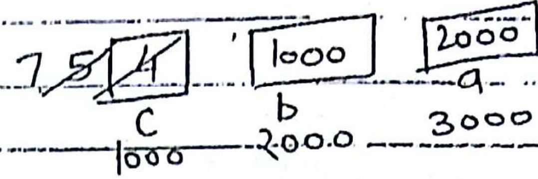
Call by Value- when values are needed for the processing

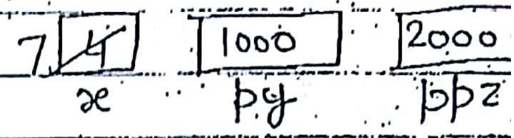
Call by reference- when actual originals are to be changed while processing.

(preprocessor will take the ~~preprocessor code~~ definition of f^n to program)

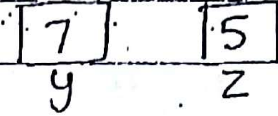
{

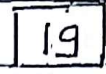
```
int c, *b, **a;
   c=4;
   b=&c
   a=&b
   pf("%d", f(c, b, a))
}
```





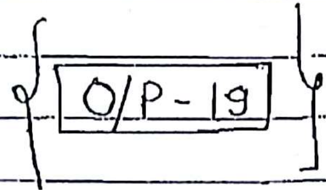

```
f(int x, int * py, int **ppz)
{
   int y, z;
   **ppz += 1;
   z = **ppz
   *py += 2
   y = *py;
   x += 3
   return(x+y+z);
}
```





19

Steps-
1. check for static variable.
2. check for static & dynam[ic] scope
3. go on



O/P - 19

$**ppz = **ppz + 1$
↓
2000
$*2000$
↓
$*1000$
↓
$4 + 1$
$= 5$

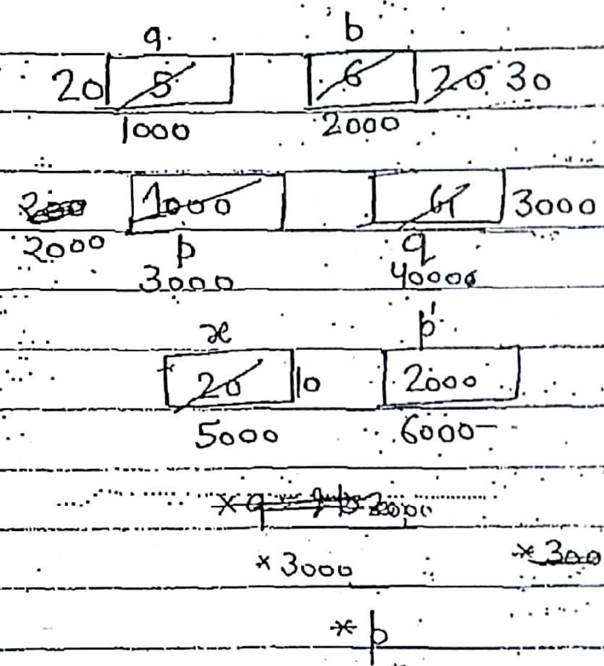
$*py = *py + 2$
$= *1000 + 2$
$= 5 + 2 = 7$

(* has priority ove[r] +) as unary ope[rator] has priority mo[re] than Binary op[erator] as to complete unary 1 is enough

float pointer - M/M pointed by that pointer has float va

Ques - ~~For~~ Consider the following C program

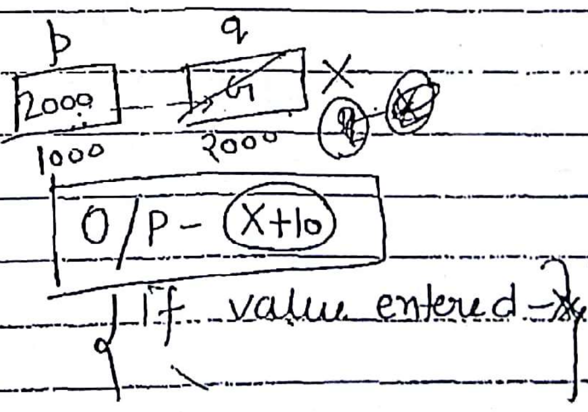
```
f(int x, int *p)
{
    *p = x;
    x = 10;
}
main()
{
    int a=5, b=6;
    int *p = &a, **q;
    *p = 20 , q=&p;
    f(a, &b)
      *q = &b
        *p = 30;
    pf(a, b);
}
```



O/P - (20, 30)

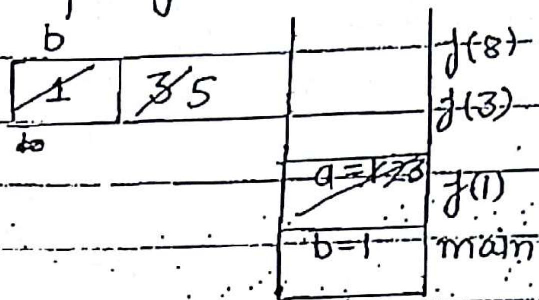
Ques - main()

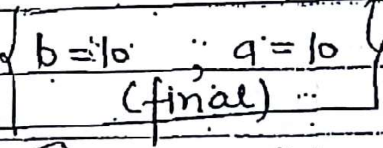
```
{
    int * p;
    int q;
    p = &q;
    scanf("%d", P);
    pf("%d", q+10);
}
```



Ques - Consider the following C program-

```
main()
{ int b=1;
    b=f(b);
    b=f(b);
    b=f(1+f(b));
}

f(int a)
{
    pf("%d", a++);
      return (++a);
}
```





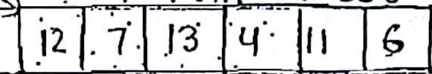
{ b=10 ; a=10 } (final)

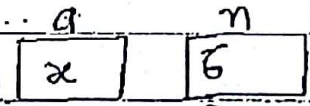
it a is array it will keep address

Ques - main()

by default int is the with main, no need write it but if don't return anything, error will occur.

```
main()
{
  int a[]= { 12, 7, 13, 4, 11, 6 };
  pf("%d", f(a, 6));
  return(0);
}

f(int *a, int n)
{
    if (n<=0) return 0;
    else
      if( *a % 2 ==0) return *a + f(a+1, n-1);
    else
      *a - f(a+1, n-1);
}
```





x+2 as you h...

12 % 2 = 0

then 12 + f(x+1, ...

x+1, 5

12 + 12 + f(x+2, ...

12 + 12 + 13 + 13 ...

12 + 7 - (13 - (4 + 11 - (6)))

= 19 - (13 - (4+5))

• Base Address of an array cannot be changed. it will give error

| a = a+3 initially a = 1000
| a ≠ 1006 a cannot be changed.

1006

initially a = x (base address)

f(x, 6)

↓

| x | | 6 |
a n

n ≠ 0

∴ *a = 12 % 2 = 0

12 + f(a+1, 5)

12 + (7 − (f(a+2, 4))

12 + (7 − (13 − (f(a+3, 3)))

↓

12 + (7 − (13 − (4 + f(a+4, 2)))

↓

12 + 7 − (13 − (4 + 11 − (f(a+5, 1)))

↓

12 + 7 − (13 − (4 + 11 − (6 + f(a+6, 0)))

12 + 7 − (13 − (4 + 11 − 6)

12 + 7 − (13 − 9)

⇒ 12 + 7 − 4

= (15)

| O/P = 15 |

a → 1000

a+1 ⇒ 1002 (skip 1 el.

a+3 ⇒ 1000 + 3×2
(skip 3 ele

a[i] = *(a+i)
= *(i+a)
= i[a]

a+i = skip i elements
a−i = go back i elements from

a[i][j]
↓
b[j] = *(b+j)
= *(a[i]+j)
= *(*(i+a)+j)
= *(i[a]+j)

for integer pointer a = 1000
a+1 = 1002

for float pointer
a+1 = 1004

for char pointer
a+1 = 1001

27-Mar

Ques - main()

{

char P[] = "gate2011";

pf ("%s", P + P[3] - P[1])

}

gate2011 (it is a string, end with null)

~~undefined~~ double course

~~1000~~ 0 1 2 3

P + P[3] - P[1]

= P + ~~*(P+3)~~ e - ~~2~~ ~~*(P+1)~~

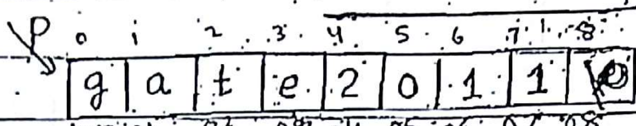
= 1000 + e - a   ~~e2011~~

= 1000 + 101 - 97 = 1004 (afte...

1004 w...

~~1000~~ 1000 + 1003 - 1001 be...

to print...



If don't want to have null

{'g', 'a', 't', 'e', '2', '0', '1', '1'}

P + P[3]

O/P = 2011

printf ("%d, p) - 1000

printf ("%s", P+2) ⇒ te2011
(1000 + 2 = 1002)

printf ("%s", "gate2011") ⇒ ga...

printf ("%s", P) ⇒ gate2011

↳ it will print string from address stored in ~~array~~ P u... Null character

printf ("%s", P+5) ⇒ 011
(1000 + 5 = 1005)

~~P+P+3~~

printf ("%c", *(P+3)) ⇒ e (only one char)
(1003)
(for printing one value put *).

~~P+*(P+3)~~ ~~*(P...~~

~~P+*1003~~ - * 100...

printf ("%d", *(P+1)) ⇒ 97 (ASCII value of a)
(1000 → a)

pf ("%c", P+2) ⇒ error as you cannot print string by %c)

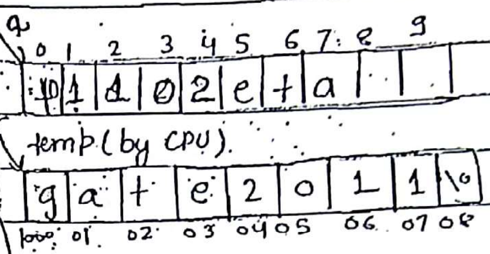
pf ("%s", *(P+5) ⇒ error as string cannot accept char

Note - If a char value from given array has to be printed, pointers are used i.e a[i] else base address is used i.e directly a.

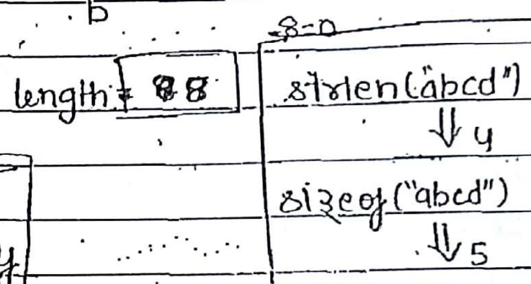
Every array is a pointer

Ques 2 - main()

```
{
    char a[] = "gotohell";   (struck out)
    char a[10];
    char *b = "gate2011";
    int i, length = strlen(b);
    for (i=0; i<length; i++)
    {
        a[i] = b[length - i];
    }
    pf("%s", a);
}
```



length = 8



O/P = No output as starting char itself is null character

a & temp & b are stored in diff. location but they are link.

but a & a[0] are strongly connected they cannot be stored separately

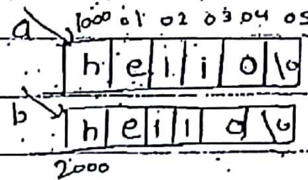
pf("%s", a+1)
⇒ 1102eta G G ... ∞ (upto time any null char occur).

Note - Here, a = a+5 (Wrong. Base address can't be changed)
⇓ (base address constt.)
a[5] = 'b' - possible
b = b+5 (possible)
*(b+5) = 'a' (Wrong. we cannot change string here)
⇓ these are known as string constant format.
It will not give error msg but no change occur.

Note -

Array Base Address cannot be changed but array content can be changed. String Constant base address can be cha but contents cannot be changed.

Ques. main()
{
char a[] = "hello";
char b[] = "hello";
if (a==b)
pf(hi);
else
pf(bi);



O/P → bi (as 1000 ≠ 2000)

if (*a == *b)
*a = h ; *b = h
⇒ O/P ⇒ hi

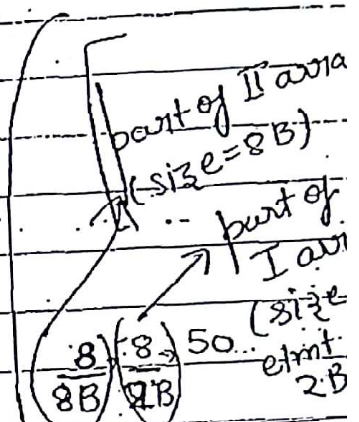
• main()
{ char a[] = "hello";
char b[] = "hello";
if (a=b) pf (hi); error as you are trying to char
else pf (bi); base address of array.

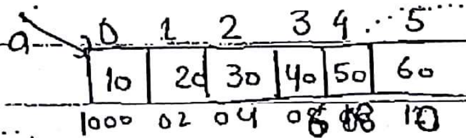
if (*a = *b) ⇒ print hi

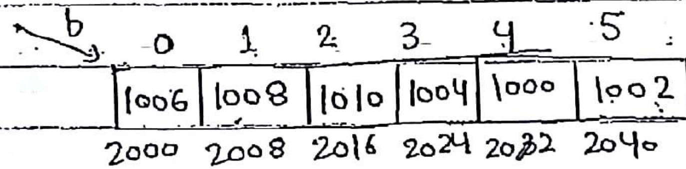
%U - Unsigned integer (used to print address)

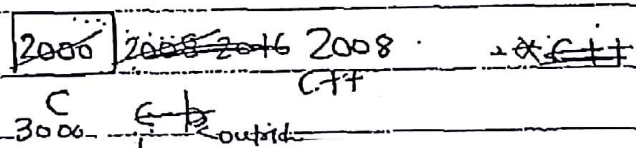
Ques -

```
main()
{
  int a[]= {10, 20, 30, 40, 50, 60};
  int *b[]= {a+3, a+4, a+5, a+2, a, a+1}
  int **c = b;
  *c++;
  pf ("%U %U %d", c-b , *c-a , **c)
  *c++;
  pf ("%U %U %d", c-b, *c-a, **c);
  ++ * ++c;
  pf ("%U%U%d", c-b, *c-a, **c);
  *(++(++c));
  pf ("%U %U %d", c-b, *c-a, **c);
}
```









① c - b = 2008 - 2000 = 8/8B = 1 i.e it will be dividing the no. of bit it take to store one value i.e it will give no of element present b/w two pointer

*c - a = *c = 1008 - 1000 = 8/2B = 4 (as size of array is 2B)

**c = 50

② ↓ *c++ = 1008

Whenever two pointer are subtracted it will give no. of elements b/w those two ptrs including first one & excluding last.

2) ~~*c++~~ ~~c++;~~ ~~2008~~ 2016 (c)

= *c

c - b = 2016 - 2000 = 16 / 8B (size of array elmt is 8B)

= 2B

*c - a = 1010 - 100 = 10/2B (size = 2B of a)

= 5

*(*c) = ~~*c = 100~~ → 2016 → 1010 → 60

(2, 5, 60)

3) ++*++c; ~~2016~~ 2024 (c)

(*2024)

*c: ~~200~~ ~~1008++~~ 1010

② *c++ → 2008 — 2008 (c)

Note - *, ++ both are unary operators with same priority but associativity is right to left.

3) *c++ ~~2008~~ 2016 (c)

c - b = 2016 - 200

= 16/8B (size of array elmt is 8B) = 2B

*c - a = 1010 - 1000 = 10/2B = 5

**c = 60

(2, 5, 60)

3) ++*++c = ~~2016~~ 2024

2016 → 2024 → 1004 ~~1010~~ 1006

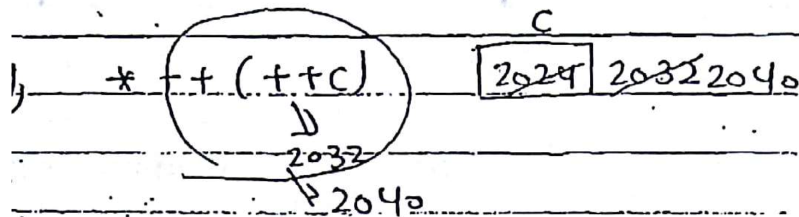
b:

00	08	16	24	32	40
1006	1008	1010	1016	1000	1002

c - b = 24/8 = 3

*c - a = 1006 - 1000 = 6/2 = 3

**c = 40



$c - b = 40/8 = 5$

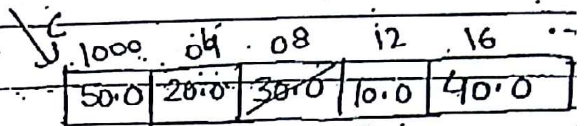
$*c - a = 1002 - 1000 = 2/2 = 1$ (5, 1, 20)

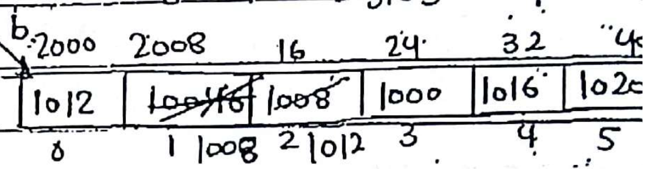
$**c = 20$

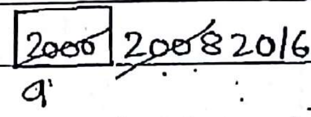
O/P →
1, 4, 50
2, 5, 60
3, 3, 40
5, 1, 20

e1
- We can add integer constant to pointer variable.
- Ex- P - pointer, P+i → skipping i slots/element includi[ng] P (in forward direction).
  P - pointer, P-i → skipping i slots/element including P (in backward direction)
- We cannot add float constant to pointer variable.
- We can subtract integer constant from pointer variabl[e]
- We cannot subtract float constant from pointer varia[ble]
- We can subtract two pointers pointing to same array
  $P_1 - P_2$ = No of element b/w P_1 & P_2 excluding (P_1) & inclu[ding] (P_2) (P_1 is greater than P_2)
- To subtract two pointers –
  - $P_1 > P_2$
  - P_1, P_2 both should point to same array.
- Subtraction of two pointer give no. of element b/w the[m]
- Pointer division & ~~subtract~~ addition has no use
  (We can Not perform add^n, multiplication & division bec[ause] they dont have any meaning)

```
main()
{
  float c[] = {50, 20, 30, 10, 40, 60}
  float *b[] = {c+3, c+1, c+2, c, c+4, c+5}.
  float **a = b;
      *a++;
  pf(a-b, *a-c, **a);
      ++*a;
  pf(a-b, *a-c, **a);
      ++**a;
  pf(a-b, *a-c, **a);
      ++*++a;
  pf(a-b, *a-c, **a);
}
```





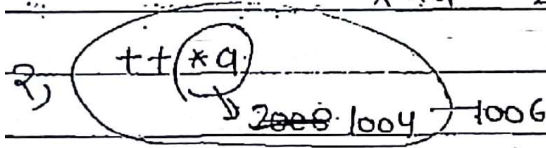


1, 1, 20.0

1) *a++ $a-b = 8/8 = 1$

$*a-c = 1004-1000 = 1$

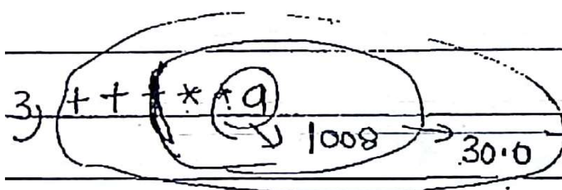
$**a = 20.0$

2) 

$a-b = 8/8 = 1$ 1, 2, 30.0

$*a-c = 1008-1000 = 2$

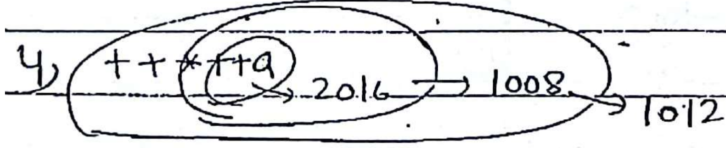
$**a = 30.0$

3) 

$a-b = 8/8 = 1$ 1, 2, 31.0

$*a-c = 1008-1000 = 2$

$**a = 31$

4) 

$a-b = 16/8 = 2$ 2, 3, 10.0

$*a-c = 12/4 = 3$

$**a = 10.6$

```
main()
{
  char *a[]={"papa","stupid","gotohell","break","chacha","chachi"
  char **b[]={a+2,a+3,a+4,a,a+1,a+5}
  char ***c = b;
  *c++;
  pf("%s", **++c);          chacha
  pf("%s", *++*c+2);        achi
  pf("%c", *(**++c+2));     p
}
```

p a p a \0
1000 01 02 03 04

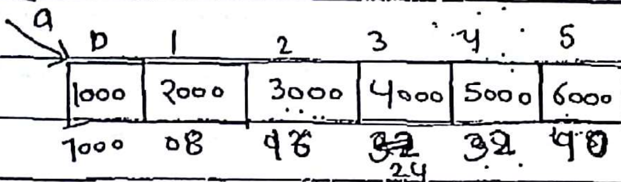
s t u p i d \0
2000 01 02 03 04 05

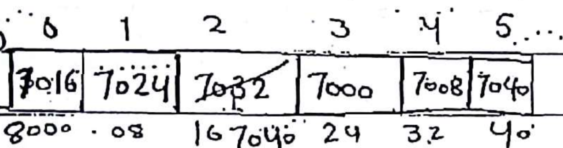
g o t o h e l l \0
3000 01 02 03 04 05 06 07 08

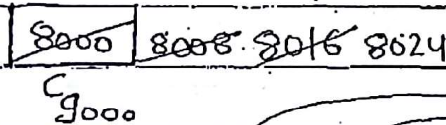
b r e a k \0
4000 01 02 03 04 05

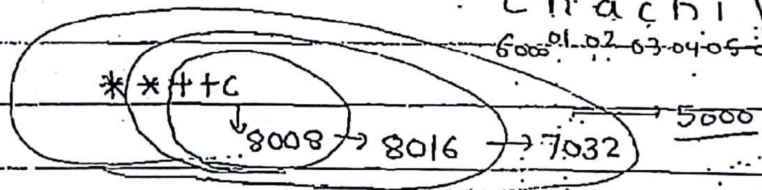
c h a c h a \0
5000 01 02 03 04 05 06

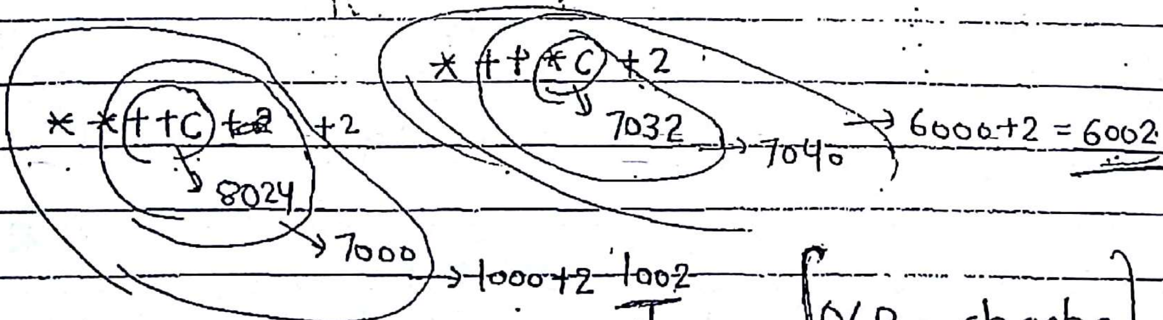
c h a c h i \0
6000 01 02 03 04 05 06











O/P - chacha
achi
p

Ques

```
main()
{
    int a[5][3] = {10,20,30,........,140,150}
    pf ("%d", (a == a[0] && (*a = *(a+0))));
    pf ("%u", *(a+2)+2);
    pf ("%u", *a+2);
    pf ("%u", *((a+1)+2)+2 );
}
```

1011 = 11 (unsigned)
1011 = -3 (signed)

a	Row 1			Row 2			Row 3			Row 4					
	10	20	30	40	50	60	70	80	90	100	110	120	130	140	150
	1000	02	04	06	08	10	12	14	16	18	20	22	24	26	28

a = 1000
a[0] = *(a+0) = 10

3) *a + 2
10 + 2 = 12

4. 1002 + 2 1004
32

It can also be represented as

* a = *(a+0)
0

(2) *(a+2) + 2
* 1004 + 2 = 32

	0	1	2
0	10 (1000)	20 (1002)	30 (04)
1	40 (06)	50 (08)	60 (10)
2	70 (12)	80 (14)	90 (16)
3	100 (18)	110 (20)	120 (22)
4	130 (24)	140 (26)	150 (28)

Note - In 2-D Array
a+1 ⇒ skipping 1-D array
if a = 1000
a+1 = 1000 + no of column in row = 1000 + 3×2
= 1006 (base address of 1st r

*(a+1) = ~~10~~ 1st row is selected = 1006
*(a+1)+2 → skipping 2 element in 1 Row
= 1006 + 2×2 = 1010

*(a+1)+2 = 1010
{ *(*(a+1)+2) = 60 }

a → 1000 (base address of whole array)
a + i ⇒ skipping i rows i.e no of column × i

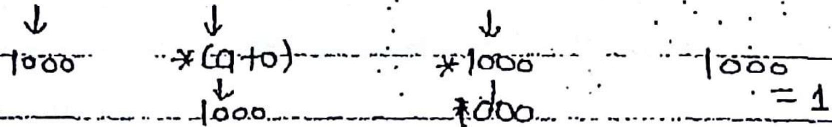
a+2 ⇒ 1012 → (2 row skipped)
a+2+2 ⇒ 1024 (Again 2 row skipped)

a[4][3][5]
↓
a+2 ⇒ skipping 2 2-D table
= 1000 + 2 * size of data * size of table
= 1000 + 2 * 2 * 15
= 1060.

*(a+2) ⇒ 2nd table selected
*(a+2) + 2 ⇒ skipping 2 row in second table
((a+2)+2) ⇒ selecting 2nd row in table (selected)
(*(*(a+2)+2)) + 2 ⇒ skipping 2 element in 2nd row of second table
((*(a+2)+2) = value

(n-D vector array require n pointer to access value)

(i) (a == a[0]) && (*a == *(a+0))



a → 1000; a[0] → *(a+0) → 1000; *a → *1000 → 1000; *(a+0) → 1000 = 1

(2) *(a+2)+2
*(a+2) → 1000 → skipping two row → 1000 + 2×3×2
⇒ 1012
⇒ skipping two element
1012 + 2×2
= 1016

3) $*a+2$

$a = 1000$ $*a = 1000$ $*a+2 = 1000 + 2\times2$

↳ 0th row selected $= 1004$

3) $*(*(a+1)+2)+2$

↓

$a = 1000 \Rightarrow a+1 = 1000 + 3\times2 = 1006 + 2$

↳ skip 1 row ↳ again skip 2 row

$= 1006 + 2\times3\times2$

$= 1018$

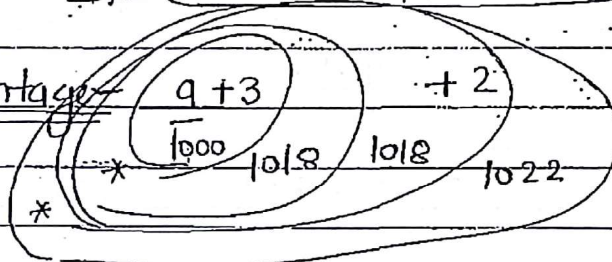
$*1018 \Rightarrow$ 3 row selected

$+2$ (skip two element in 3 row)

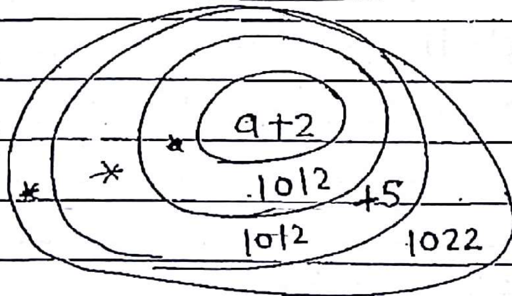
$\Rightarrow 1018 + 2\times2 = 1022$

O/P - 1, 1016, 1004, 1022

DisAdvantage-



$\Rightarrow 120$ a[3][2]



120 $\Rightarrow$ a[2][5]

it is not actually possible but occuring here as there is no row & column present in the array.

Ques main()

```
{                BA = 1000
float a[5][2][3] = {10, 20, 30, ... 300}.
pf("%u", *(a+3) +5);
pf("%u", *(*a+5) +2);
pf("%u", *a[3] +5);
pf("%u", *(**a +2) +5);
}
```

i) a = 1000 (3 2D skipped)

a+3 = a + 3 * (2×3×4) (size of 2D ↓ size of element)

= 1000 + 72

= 1072

~~*(1072)~~ = *(a+3) = 1072

*(a+3) + 5 = 1072 + 5×[2 3 × 4]

(↓ skip 5 1-D; size of 1-D; size of element)

= 1072 + 60 = 1132

(ii) *(*a+5)+2

a = 1000 (table selected) as ((a+0) = *a)

*a + 5 = 1000 + 5×(2×3×4) (row selected)

= ~~1120~~ 1060

*(*a+5) = 1060

*(*a+5)+2 = 1060 + 2×4 = column

= 1068

(iii) *a[3]+5

a[3] = *(a+3)

= *(1000 + 3×2×3×4)

= *(1072) = 1072 (table selected)

*a[3] = 1072 row selected

*a[3]+5 = 1072 + 5×4 = 1092

(iv) $*(**a+2)+5)$

$*a$ = table selected = 1000

$**a$ = row selected = 1000

$**a+2 = 1000 + 2\times4 = 1008$

$*(a**a+2)$ = ~~1008~~ 30

$*(**a+2)+5 = 35$

O/P = 1132, ~~1168~~ 1068, 1092, 35

(v) $*(*(*a+7)+5)$ (column) ele

$*a$ → 1000 (row)

$1000+7\times3\times4$

1084

$1084+5\times4 = 1104$

Structure & Unions

Structure-
- User defined data type
- a variable is defined

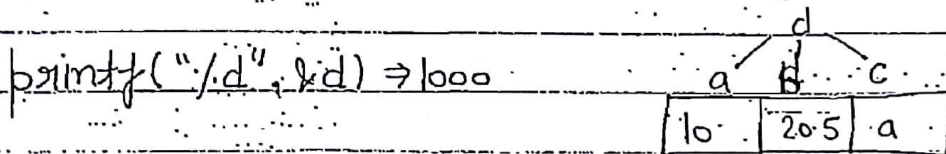
(upto the time no variable is declared no M/M is allocated)

```
struct node
{
  int a;
  float b;
  char c;
};
```

⇒ a data type in compiler is defined of 7 bytes
⇒ name of datatype = struct node
⇒ size of data type = 7 Byte

struct node d; - declared a variable of type struct node

printf("%d", &d) ⇒ 1000



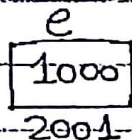
d = {10, 20.5, 'a'};

printf("%d", d.a) ⇒ 10
printf("%f", d.b) ⇒ 20.5
printf("%c", d.c) ⇒ a
printf("%d", d) ⇒ by default 1st value will come ⇒ 10

(.) operator is used to access the element inside the structure.

struct node *e; e a variable, i.e pointer variable of 8 by store address of struct node type data

*e = &d;



pf("%d", e) ⇒ 1000

pf("%d, (*e).c) ⇒ 'a'
pf("%d", (*e).a) ⇒ 10

$(*e).a \Rightarrow *e \Rightarrow$ *1000 will point to d.
↓
1000
.a will select a field.

$(*e).a \cong e \rightarrow a$

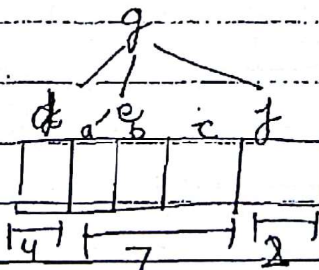
e = e+1 ⇒ if e = 1000
e = e+1 = 1000+1 = 1007
↑
size of 1 variable of type struct node.

printf("%d", &(d.b)).
⇓ 1002

Ex.2 -
```
struct s1
{
    int a;
    float b;   } 7Byte
    char c;
}
```
⇒ Here data type is created only, no allocation to memory is done so, in structure, you cannot initialize the data.

```
struct s2
{
    float d;      ⇒ 4
    struct s1 e;  ⇒ 7
    int f;        ⇒ 2    } 13 Byte
};
```
* You can use a particular structure as a Member in another structure.

```
//struct s2
{   float d;        ⇒ 4B
    struct s1 *e;   ⇒ 8B    } 14 Byte
    int f;          ⇒ 2B
};
struct s2 g;
g = {10.25, {20, 20.5, 'a'}, 10};
```



g is able to watch & access 3 variable d, e, f only

pf("%f", g) ⇒ 10.25 (by default d is printed)
pf("%f" g.d) ⇒ 10.25
~~pf("%d" g.f) ⇒ 6~~

pf("%d", g.e.a) = 20
↳ to access element inside e

struct s2 *h   size = 8 byte
~~*~~h = &g   will store the address of struct s2 data

1000
h

pf("%d", h) ⇒ 1000
~~pf("%d", &h~~
pf("%f", *h.d) ⇒ 10.25
pf("%f", *h.d) = 10.25

{*, (), . ⇒ removed by → }

pf("%d", *h.~~d~~.a) = 20
⇓
pf("%d", (h→e).a) = 20

pf("%f", (*h.d)) = pf("%f", (h→d))
pf("%f", *h) ⇒ 10.25
h = h+1 ⇒ if h = 1000
h+1 = 1000 + 13 = 1013
↑ increasing from a slot

- declare first,
- donot declare more than one,
- if operation to be performed, In two variable, they both have to be of same type.

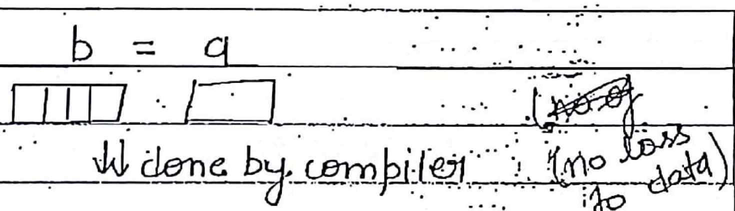
Typecasting - Data

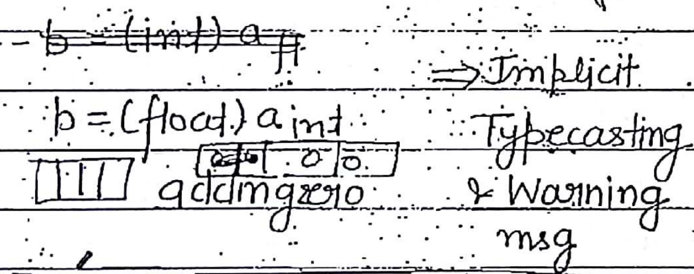
int a; float b; float c;

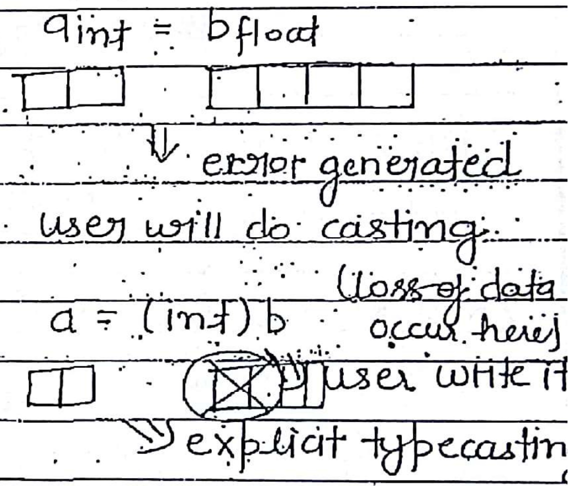
Typecasting splits into:

- Implicit (done by compiler) (when a data of smaller size is getting stored in greater size then compiler will do it)
- Explicit (done by user) → when there is chance of loasing the data.

Typecasting is done because to any operation to be performed, data type of variable must be same).



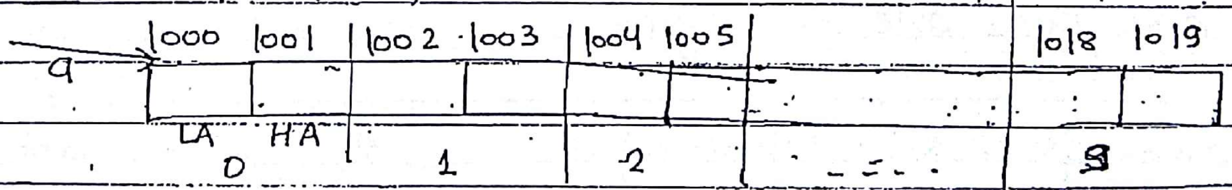




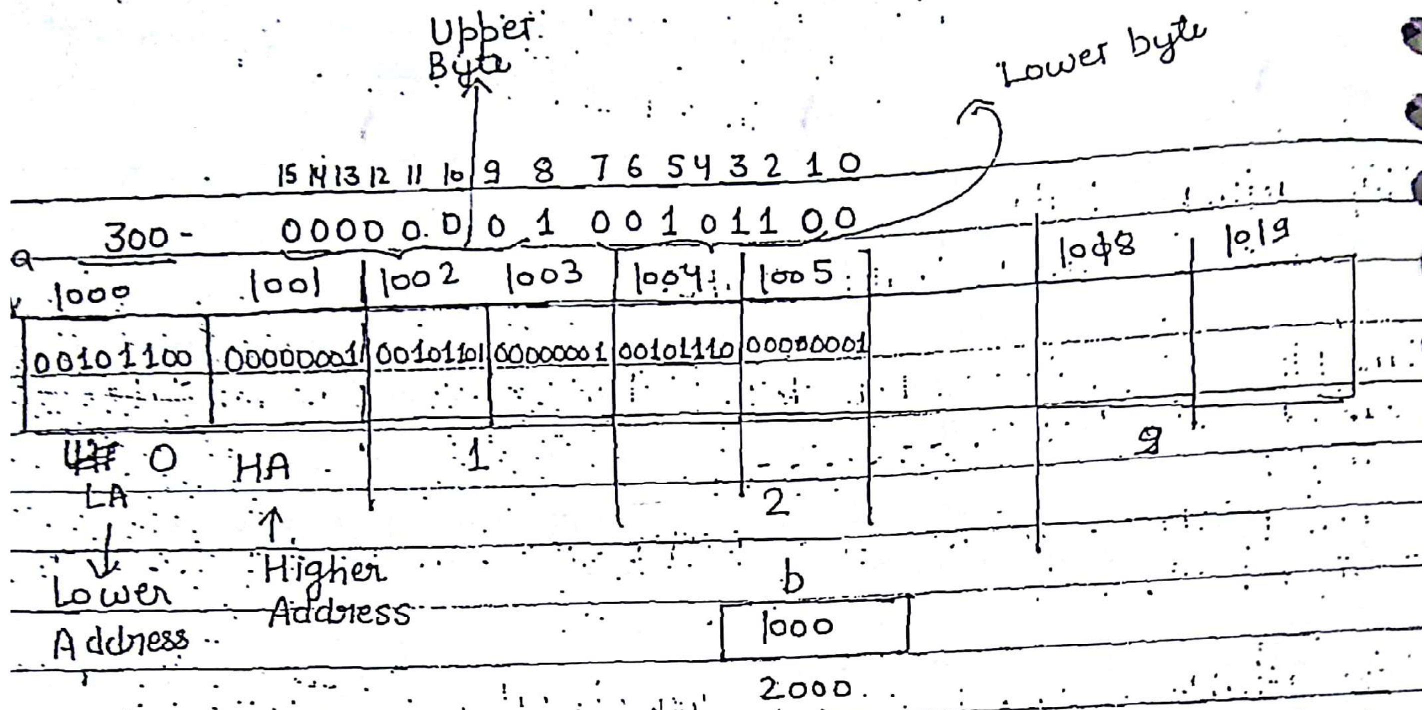
Typecasting in Pointers -

300 ~~111~~ 100101100

int a[10] = {300, 301, ... 309}



Lower byte get stored in lower address & High Byte i.e MSBs get stored in Upper By[te]

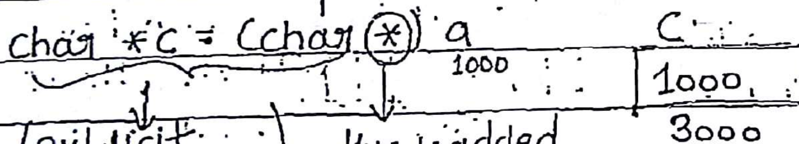


int *b; b is a variable which will store address of integer data.

size of b = 8Byte

int *b = a (no error no typecasting required as b was expecting integer type address)

char *c = a (problem) as it was expecting character pointer.

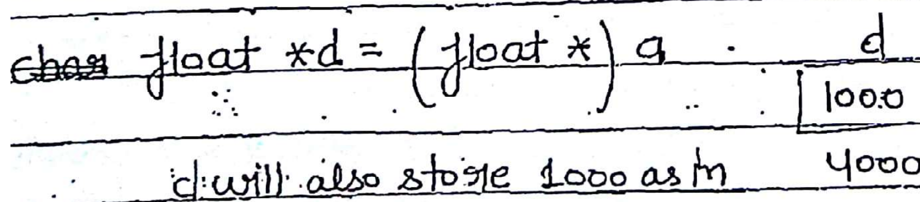


as you have to change for typecasting has to be done for pointer.

the value stored in c will be 1000 as it is in a beca

a & c both are of 8 bytes.

~~char~~ float *d = (float *) a



d will also store 1000 as in a as they both have same size but if float + 1 = 1004

* is kept with float as we have to do typecasting for pointer

void - it will allow any type of data so no conflict and no need of type casting · but typecasting is require at time of retrieval ·

void *e = a   e [1000]

if print b ⇒ 1000
*b ⇒ since it is an integer pointer it will read 2 Byte of data
= 300
if b+1 = 1000 + 2 = 1002
↑ integer size

if c = 1000
*c = it is an char pointer it will read 1 byte
= 44
c+1 = 1000 + 1 = 1001
size of char

d = 1000
*d = it will read 4 Byte
(1003 1002 1001 1000) bits
d+1 = 1000 + 4 = 1004

e = 1000
*e = error as it is of void type, it donot have any info to read bit.

If you want to use e, then type casting of e must be done

```
int*j = (int*)e;
```

ues - a file is provided, how to read it char by char.
- open the file, it will return pointer to start of file (fp)
- type cast it to char ~~by~~

  char *c = (char *) fp

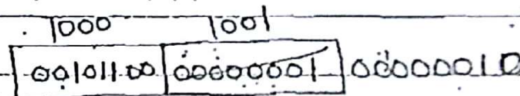
  then do c=c+1

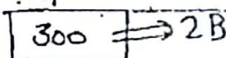
ues - to read file bit by bit typecast to Bool value.

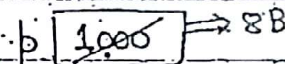
ote - when typecasting of pointers is done, it seems as they have same address but the method of accessing data get changed here.

roblem -

```
main()
{
    int i = 300;
    char *p = (char *) &i;
    p++;
    *p = 2;
    pf("%d", i);
}
```







p++ (due to char it will be increased 1 byte.)

~~*p = *p+2~~

~~i = 812~~

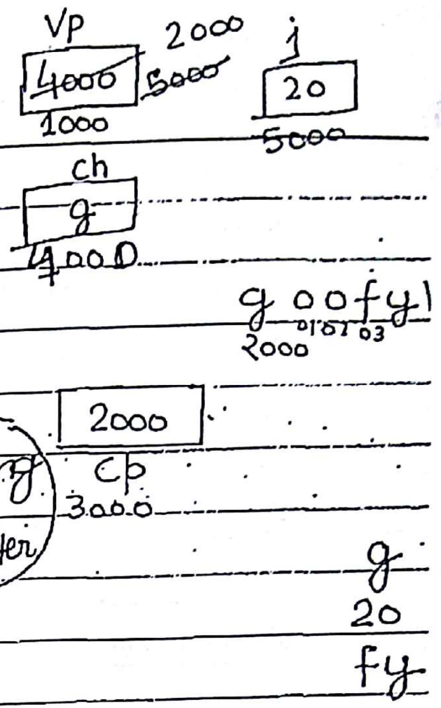
O/P - i = 512+44 = 556

*p = 2

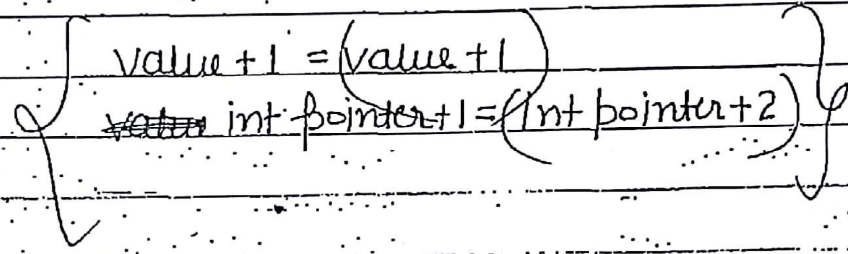
int a=20	int a=20
int **c =(int **)a	int *b=(int *)a = 20
c = 20	b = 20
but ++c=28 ~~as it is~~ pointing to ptr as it is of 8Byte	as 2 byte data get mapped in 8-byte ++a = 21 ++b = 22(as it will treated like

Ques Consider the following C program

```
main()
{ void *vp;
  char ch='g';
  char *cp="goofy";
  int j=20;
  vp=&ch;
  pf("%c", *(char *)vp);
  vp=&j;
  pf("%d", *(int *)vp);
  vp=cp;
  pf("%s", (char *)vp+3);
}
```



O/P - g, 20, fy   O/P - g20fy



Use of typecasting - Body scanning in hospital.

↳ Deeptilli wy Shekhchilli :P :P

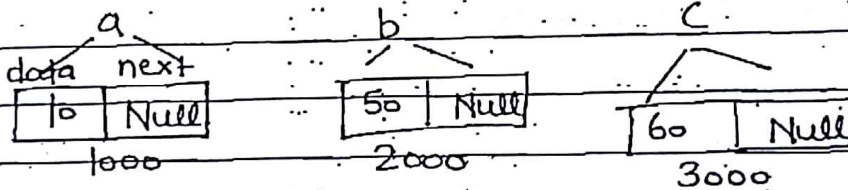
Linked List- List- collection or group of elements
Linked list- when many lists are linked.
• Self Referential data structure

1) struct node
{
int data; ⇒ 2B
struct node *next; ⇒ 8B
};

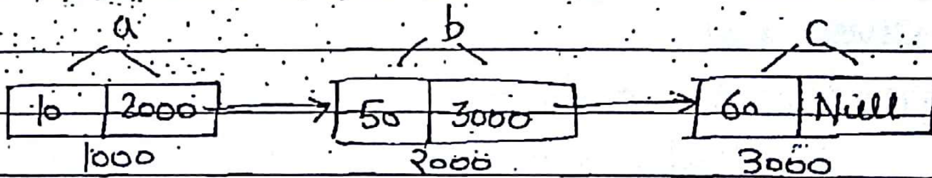
// it will create a data type of 10B with name struct node.
(storing address of struct node type data)

2. struct node
{
int data;
struct node next; you cannot use it, it will give error as undefined datatype.
};

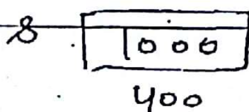
3. struct node a = {10, Null} ⇒ allocation of memory while declaration
struct node b = {50, Null}
struct node c = {60, Null}



4. linking is done at this stage
a.next = &b
b.next = &c



5. through linked list, from starting node it is ~~easy~~ possible to access each node.
struct node *s = &a



pf(s) ⇒ ~~1000~~
50

EXCELLENT

Page No.
Date : / /

5. pf ("%d", (*s).data) ⇒ 10
⇓ it can also be written as
pf ("%d", s→data) ⇒ 10.
pf ("%d", (*s).next) = 2000
or
s→next

: When linked list is created, s is actually storing base address i.e. starting address of list then return s & you can access the whole list.

6. return(s);

```
int* q()     // it is a function which donot take any i/p
{                but return pointer ie integer address
}
```

So for creating Linked list, s will be a pointer of struct node type.

```
struct node * Create_LL()
{
    return(s);
}
```

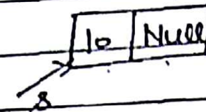
s ⇒ 1000
s →next ⇒ 2000
++s ⇒ 1010

Problem 1 - Write a C program to print every node data in given list.

```
void Print(struct node *s)
{   printf("%d", s->data);
    While(s->next != NUL  do n
    While(s->next != NULL) {
    {
```

```
void PRINT(struct node *s)
{
   // if(s==NULL)          (No need to
   { print("No data");     write if
     return;               condition
   }                       but it is not
                           wrong)
   while (s != NULL)
   {
      printf("%d", s->data);
      s = s->next;
   }
}
```

Time Complexity - O(n)
(BC, WC, AC)



if (s->next == NULL)
LL contain 1 element

if (s == NULL)
LL is empty → Null

if s = NULL
(*s will be segmentation error)

as s = NULL
* NULL - No memory there.
known as segmentation error

Ques - Write a C program to return address of second last node in the given linked list.

```
struct node * address(struct node *s)
{  if(s==NULL) return NULL & else if(s->next==NULL)
                                          return NULL;
   while (s->next != NULL)
   {   p = s
       s = s->next;
   }
   return p;
}
```

Time Complexity - O(n)
(BC, WC, AC)

```
{
   if(s==NULL)
   return NULL
   else if (s->next==NULL)
         return NULL;
   while (s->next != NULL)
       p = s;
       s = s->next;
   return (p);
}
```

Copyright©Theorypoint.com

Topological Sorting

call by

Activation Record (Do More+More)

VAL:

Variable declaration in Pascal

Aliasing

Anagram ADT

go to basics like ADT, Aliasing etc.

Page No.
Date : / /

-Apr

Ques- WAP in C to return the address of a next which contain data as X.

```
(struct node*) address(struct node *s)
{
    if(s->data == while (s->next != N
    if(s==NULL)
      return NULL
    if(s->data==x)
        return s;
    if(s==NULL) return NULL;
    while(s->next != NULL)
    if(s->next==NULL)
        if(s->data==x)
            return s;
        else return -1;
    while (s->next != NULL)
    {
        if(s->data == x)
            return s;
        else
            s=s->next;
    }
}
```

⇒ Time complexity - O(n) WC
O(1) - BC

OR

```
(struct node*) address(struct node *s)
{
    if(s==NULL) return s;
    else
    while (s!=NULL)
    { if(s->data == x)
```

{Merge Sort on Linked list ⇒ n log n}

Page No.
Date: / /

{Min. time Complexity with linked list = O(n)}

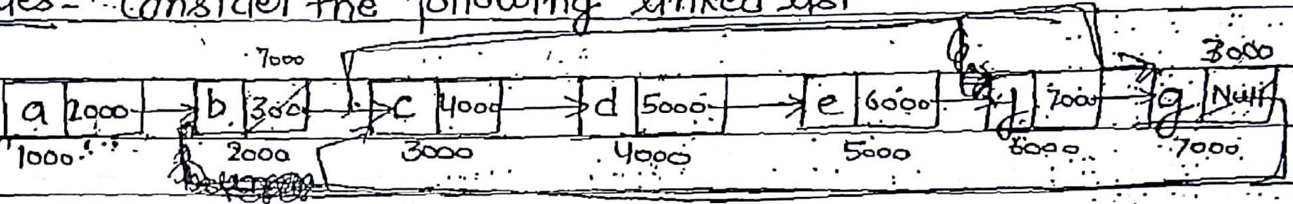
(struct node *) find_x (struct node *s)
{
    while (s != NULL && s→data != x)
        s = s→next;
    if (s == NULL) return NULL
    else return s.
}

(It will return the posⁿ of x present in list)

⇒ {Time Complexity – O(n)} WC

*** (triple pointer)

Ques:- Consider the following linked list-



(i) struct node *p;
(ii) p = s→next→next→next;
(iii) p→next→next→next→next = s→next→next;
(iv) s→next→next = p→next→next→next
(v) pf ((p→next→next→next→next→next→next→next→data

5000 6000 7000 3000 4000 5000 6000

s = 1000   ~~s→next→((*s).next).next.next~~   *(*((*s).next).next).next

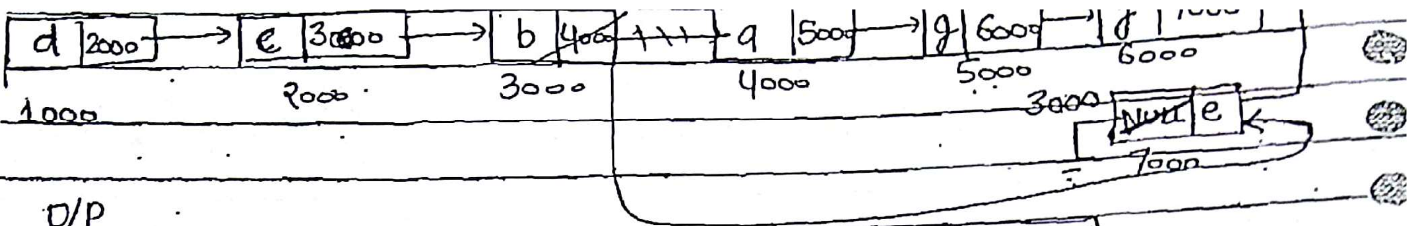
(ii) p = 4000

(iii) *(*(*((*p).next).next).next).next

(iii)

(b)(f)

O/P – f



O/P

i) struct node *p;

ii) p = s→next→next→next→next;

iii) s→next→next→next = p→next→next;

iv) ~~p =~~

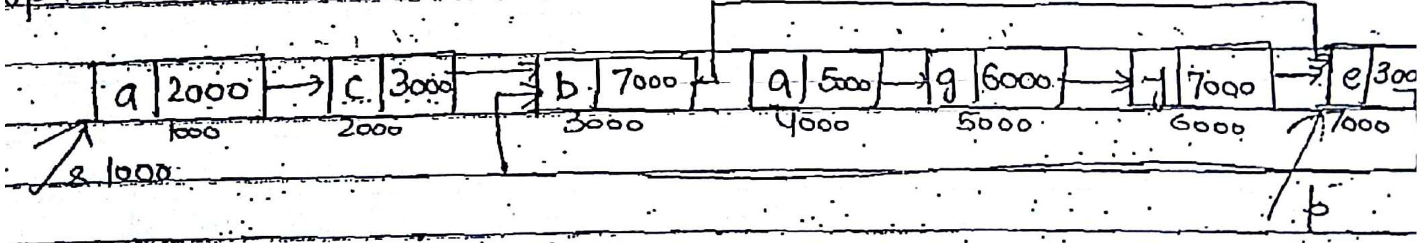
iv) p = s→next→next→next

v) s→next→next→next→next = s→next→next

vi) pf(s→next→next→next→next→next→next→next→data)

O/P - e

Updated list



Page No.
Date : / /

Ques - WAP in C to add a node with data x at the end of given linked list.

```
(struct node *) insert(struct node *s, int x)
{   struct node p, *q;  q=s;
    if(s==NULL)
    {  p->data = x;
       p->next=NULL;
       s=&p;
       return s;
    }
    While(s->next != NULL)
        s=s->next;
    p->data=x
    p->next=NULL;
    s->next=&p;
    return q;
}
```

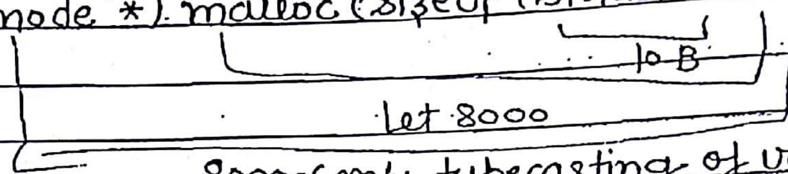
(WC, AC, BC)
⇒ Time complexity - O(n)
But linking take - O(1)

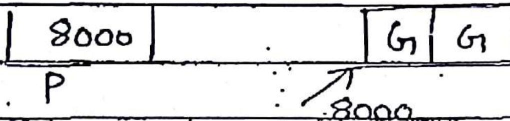
malloc(10B) - M/M of 10 B get created in Heap area

malloc (size of(struct node))
⇓ Here
allocation of memory is done in Heap Area of size struct node of void type. i.e any type of data can be stored.
So it is a void m/m so typecasting is done here.
it will return address of that Memory.
= (struct node *) (malloc(size of (struct node)))
↓ struct node type since malloc will return add.
so * is used at type casting.
Let malloc return 8000
8000+1 = 8010

~~st~~ struct node *P;

P= (struct node *) malloc (sizeof (struct node))





∴ printf (P) = 8000

~~*P =~~ (*P).data = x;

(*P).next ⇒ NULL;

Dynamic M/M allocation- M/M creates at run time & remain forever until ~~use~~ u free it

Dynamic Time M/M - stack (when f^n over M/M to local variable also deallocated)
- heap (m/M will remain after execution complete).

malloc - to allocate M/M dynamically initial value are Garbag

free - to free the created Memory; ie deallocation of M/M.
Here, deallocation of M/M is in user hand.

calloc → allocation of M/M by Clearing i.e M/M get created (c) (alloc)
by initialization of default value to zero.

Page No.
Date : / /

Ques - WAP in C to insert a node with data x before a node with data y in given linked list.

```
(struct node *) insert(struct node *s, int x)
{
    struct node *p, *q, *r ; r=NULL;
    p=(struct node *)malloc(size of(struct node))      // creation of node
    p->data=x;
    p->next=NULL;
①  if(s==NULL)
        return;
    q=s
③  while(s!=NULL)
    {
        if(s->data==y) break;
        else {
            r=s;
            s=s->next;
        }
    }
    r->next=p;
    p->next=s;
    return q;
}
④ if(s==NULL)
    return (no Y found)
}
```

```
② if (s->data==y)
   { r=p;
     p->next=s;
     return r;
   }
   else
```


EXCELLENT

Page No.
Date.: / /

```
① struct node *P, *q;
   P=(struct node*) malloc (size of (struct node));
   *P. data= x;
   *q. next= Null
2. if (s==NULL) return NULL.        // list empty
3. if (s->data ==Y)          // for node Y at 1st place
   { P->next=s;
      s= p;
      return s;
   }
4. q= NULL
5. while(s->data != y && S->next != NULL)
   { q=s;
      s=s->next;
   }
   if(s->data ==y)
   {
      q->next=p;            {Time complexity- O(n)}
      p->next=s
   }
   else pf(no y)
}
```

EXCELLENT

(Deletion means freeing of ...)

Page No.
Date : / /

s→next
free(s)·

Ques. WAP in C to delete a node at the end of given linked list.

```
(struct node *) delete_at_end ( struct node *s)
{  struct node *q;  q=NULL;  s1=s;
   if(s==NULL)
      return;
   while( s1->next != NULL)
   {
       q=s1;
       s1=s1->next;
   }
   free(s1); s1=NULL      (to free dangling pointer)
   q->next = NULL;
   return s;
}
```

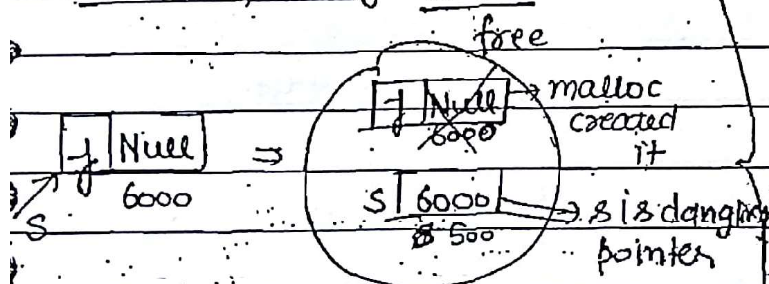
```
if(s->next==NULL)
{ free(s);
} return NULL;
```

If you dont wanna use q, use (s->next->next)

then you can do like-

```
q=s
if(s==NULL)
    return
if (s->next==NULL)
{ free s;
  return NULL;
}
while(s->next->next != NULL)
{
   s=s->next;
}
p=s->next;
free(p); p=NULL
s->next=NULL;
return q;
```

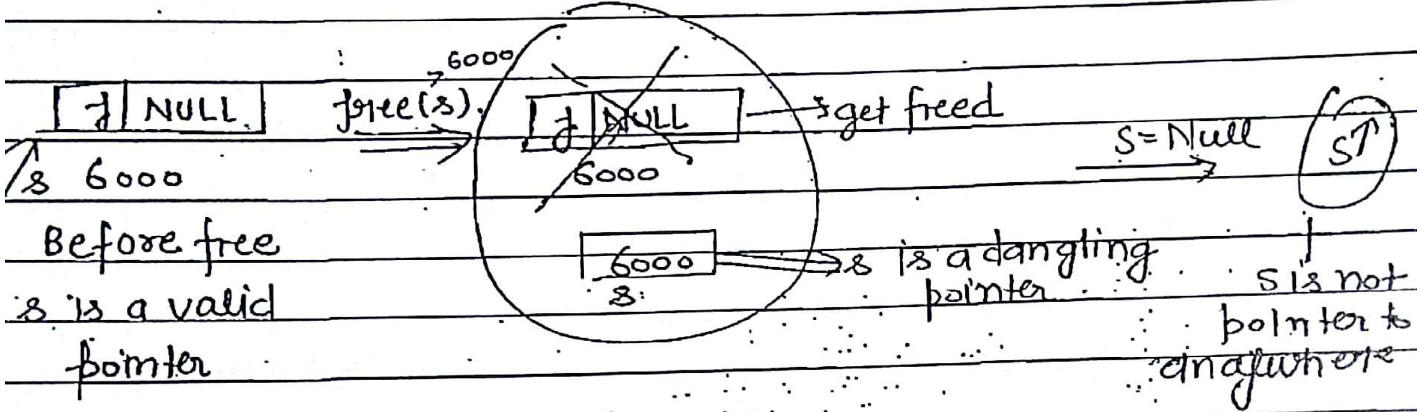
Time Complexity - O(n).



free(s) - It will delete the m/m location 6000
but s still have data 6000
It is pointing to 6000 from where M/M is freed, but s is still pointing. s is known as dangling pointer.

Dangling pointer - A pointer pointing to M/M location which is not there ie if is freed.

so use s = NULL to free it.



When M/M is freed & still used it will give Garbage value.
i.e.

before free - print (s, s→data, s→next) ⇒ 6000, f, NULL
after free - print(s, s→data, s→next) ⇒ 6000, G, G
s = NULL - print (s, s→data, s→next) ⇒ NULL, segmentation error.

Ques - WAP in C to delete a node which contain data x in the given linked list.

```
(struct node *) deletex(struct node *s, int x)
{ struct node *p, *q
  if (s==NULL)
     return;
  if (s->data==x)
  { free(s);
    s=NULL;
    return NULL;
  }
```

Page No..
Date : / /

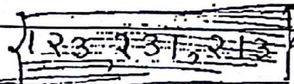
```
q=s
while( q->data != x && q->next != NULL)
{
    p=q;
    q=q->next;
}
p->next=q->next;
free(q);
q=NULL //freeing dangling pointer
if(s==NULL)
{
    Nox there so return;
    return s
}
```

(You cannot change the order here because AND operator works on short circuit property)

{Time Complexity - O(n)}

Ques - WAP in C to move last node of linked list to front of linked list

```
(struct node *) movelast to first(struct node *s)
{
    struct node *p, *q; q=s;
    if(s==NULL)
        return NULL
    if (s->next==NULL)
        return NULL;
    while (s->next !=NULL)
    {   p=s;
        s=s->next;
    }
    ① p->next=NULL;
    ② s->next=q;
    ③ q=s;
    return q;
}
```



{change of order here will change the code}

EXCELLENT

In array - base add. & no of ele is given.
in linked, - only start add. is given.

Ques- WAP in C to find middle of given linked list.

```
(struct node *) middle (struct node *s)
{
    if (s == NULL) return -1;
    if (s -> next == NULL) return -1;
    ~~if (s -> next~~
    while ( s1 != NULL)      } O(n)
    { count++;
      s1 = s1 -> next;
    }
    count = [count/2];
                                    while ( count != 1)
    for ( i=1; i < count; i++)
    {                                         O(n/2)
        s = s -> next;
    }
    return (s);
}
```

$\Rightarrow$ Total complexity $= O(n) + O(n/2) = O(\frac{3n}{2}) \simeq O(n)$

| Last element - stack |
| first element - Queue |

$\frac{3n}{2} + \frac{n}{2} + \frac{n}{4}$

$\frac{3n}{2} + \frac{3n}{4} + \frac{3n}{2^3} + \ldots$

$3n \left[\frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} + \ldots \right]$

$3n \left[\frac{1}{2} \ldots + (\frac{1}{2})^{k} \right] O(\frac{3n}{2})$

Algorithm 2 -

p	q
1	1
2	3
3	5
4	7
5	9
6	11
:	
50	100
:	:
n/2	n

[p is always incremented by 1.
q is always incremented by 2
when q reaches at last of the list, p reaches mid of the list.]

to check condition for q, skip two element is required.
[q→next→next != NULL]

- If q has to be incremented, q must have two elements after it.
- if q has one element after it it will not incremented.
- if q has zero element after it, it will not incremented.
- either both will ~~inp~~ incremented or no one get incremented.

Algorithm - mid(S)

```
1) p = q = S
2) while( q != NULL && q->next != NULL && q->next
                                       ->next != NULL)
   { p = p->next;
     q = q->next->next;
   }
   if(q == NULL)
       return NULL
   else if (q->next == NULL)
           q = q->next (to be at end)
   return p ; // mid of list
   return q; // last element
```

O(n/2) (while loop)

cond^n (if / else if)

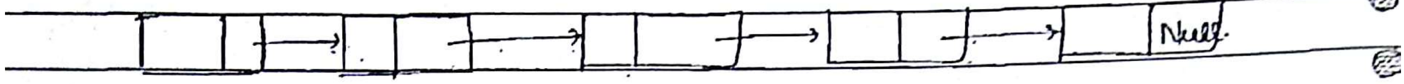
Time complexity = O(n)

it return ~~ceiling~~ floor value of mid if n is even

EXCELLENT

Page No.
Date : / /

Ques - WAP in C to perform Binary Search in linked list



• List must be sorted.

```
Binary Search(s, x)  ── 1 element [ if (s->next == NULL)
                                    { if (s->data == x)
                                          return s
                                      else return -1 ]
   else  p = q = s
      while(q != NULL && q->next != NULL && q->next->next != NULL)
      {
            r = p;
            p = p->next;                    } O(n)
            q = q->next->next;
      }
      { If (q != NULL)
            return;
        else if (q->next == NULL)           { TC = O(n log n) (WC)
                  q = q->next;                BC = O(n)
                                            { TC = O(n)
                                              BC = O(n)
      if (p->data == x)
            return p
      else (p->data > x)
            ~~Binary Search~~ r->next = NULL;
               Binary Search (s, x);        T(n/2)
      else
            Binary Search (p->next, x);     T(n/2)
```

Page No.
Date : / /

~~To find mid~~

```
BS(S, x)
{
  if (s->next == NULL)                 => O(1)
  { if (s->data == x)
        return s;
    else
        return Null;
  }
  else
  { m = mid(s)                         => O(n)
    if (m->data == x) return m         => O(1)
    else if (m->data > x)
      {   m->next = NULL
          BS(S, x);                    // T(n/2)
      }
    else
      { BS(m->next, x);                T(n/2)
      }
  }
}
```

$$T(n) = \begin{cases} O(1) & ; \text{ if } n=1 \\ n + O(1) + O(1) + T(n/2) & ; n>1 \end{cases}$$

Time complexity

$T(n) = T\left(\frac{n}{2}\right) + n$

$= O(n)$

// Use Brain Never ~~cons~~ go on your consideration

space complexity $= O(\log n)$

EXCELLENT

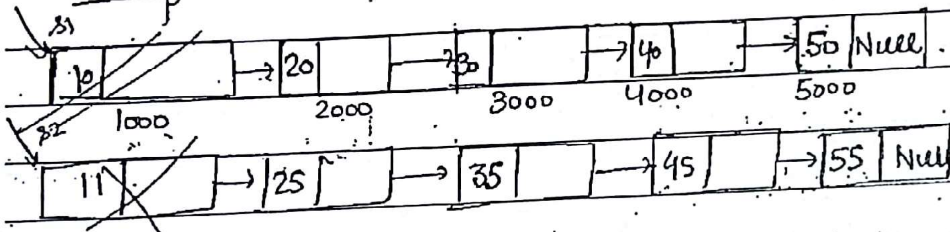
(Binary search is applicable to all ... structures)

Note- On Linked list, Binary Search is possible but it is not efficient because it will take time complexity of O(n) (BC, WC, AC)
{ Linear Search also O(n) }.

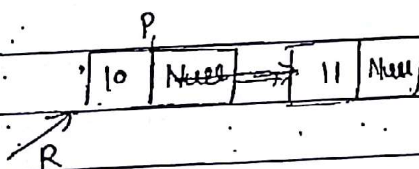
Ques - Write a C program to perform Merge Sort on given linked list.

struct node*) Merge_Sort_LL(struct node *s)
{

Merge -



if (s1→data < s2→data)
p = s1. ; s1 = s1→next . p→next = Null ; R = p
else p = s2 s2 = s2→next p→next = Null ; R = p.
& soon but R = p R→next = p;



⇓ O(n) (since n/2 + n/2 moves are here)

- [Inplace Algorithm since same m/m is used again & again.
No Need of Creating M/M.

Initially compare the elmt in list which one is smaller store in p, increment the list & store p in R i.e R is pointing P & again element getting added to R.

Time = O(n) as Total moves = O(n)
↳ Total comparison = O(n)

Note - [Merging two sorted ~~subarray~~ linked list each of size m, n will take O(m+n) time (BC, WC, AC).] · It is an inplace algorithm but in arrays, it is outplace.

Page No.
Date : / /

```
(struct node*) Merge_Sort (struct node *s)
{
    if (s -> next == NULL)          --> O(1)
    {
        return s;
    }
    else {
        m = mid(s)                  --> O(n)
        s1 = Merge Sort (m -> next);   --> T(n/2)
        m -> next = null;
        s2 = Merge Sort (s);           --> T(n/2)
        s = Merge (s1, s2)             --> O(n)
```

$$T(n) = \begin{cases} O(1) & ; \text{ if } n == 1 \\ 2T\left(\frac{n}{2}\right) + O(n) + O(n) + O(1) & ; \text{ if } n > 1 \end{cases}$$

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n) + O(n)$$

$$= 2T\left(\frac{n}{2}\right) + 2n$$

$$= 2n \log n$$

$$= O(n \log n) \quad (\text{Inplace})$$

[It increases the time to solve ~~due to~~ because random access not possible]

EXCELLENT

Page No.
Date : / /

Ques. Consider the following C program-

```
f(struct node *p)
{
    return (((p == NULL) || (p->next == NULL) ||
            ((p->data <= p->next->data) && f(p->next)));
}
```

the above f^n on given linked list p always return 1 then linked list p should be-

1. p contain 0 node ✓
2. p contain 1 element ✓
3. the data in p is in increasing order/ascending element ✓
4. the data in p is in descending order.

Option a & b are contained in c.

Ans - (C)

```
return ((p == NULL) || (p->next == NULL) || ((p->data <= p->next->data)
        || f(p->next)))
```

⟹ any sequence always return 1

Time Complexity = O(n) (WC)
= O(1) (BC)

Ques - WAP in C to perform Selection Sort on given linked list.

```
(struct node *) Selection_Sort(struct node *s)
{
    while if (s->next == NULL)
        return s;
    while (s != NULL)
```

EXCELLENT

Page No.
Date: / /

```
    count++;
    s=s->next;
  }
  ~~for(i=1; i<count; i++)~~
  while(count !=0)
  {  p=s ~~->data~~
```

ModifiedSelection Sort(s)   (Modified Selection Sort)

```
{
   for(p=s; p!=NULL; p=p->next)
   {
      for(q=p->next; q!=NULL; q=q->next)
      {
         if(q->data < p->data)
           swap(p->data, q->data)
      }
   }
}
```

Time Complexity $= n + n-1 + n-2 + \cdots 1$
$= O(n^2)$

Swaps $= O(n^2)$ (Worst case)

But in case of array, Swaps were $O(n)$ → in actual ($n-1$)

Ques- WAP in C to find loop in given linked list.

address part

a 2000	→ b 3000	→ c 4000	→ d 5000	→ e 6000	→ f 7000	→ g 8000	→ h 9000	→ i 10000	→ j 11000	→ k 6000
1000	2000	3000	4000	5000	6000	7000	8000	9000	10000	11000

EXCELLENT

Page No.
Date: / /

Algorithm-1.
b. Assume each node has three fields of linked list (data, flag next). & flag field is 0 for all model initially.

. Visit each node & check flag field
if (flag==0)
flag=1
else
pf(loop is found).

Here a extra field is added it seems to be added but actually when LL is created for project initially five fields are ~~required~~ there so that when required, project can be expanded.

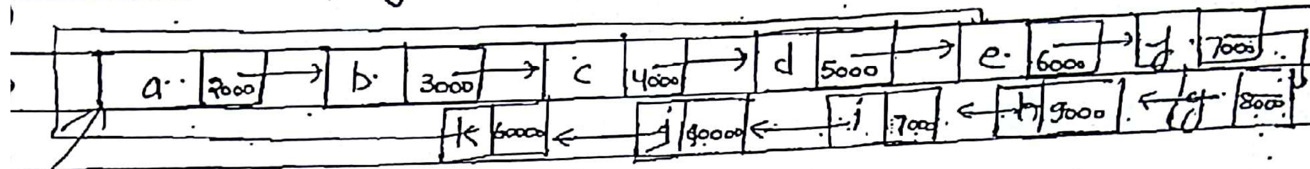
if no extra M/M is there, we can use resize f^n to resize the structure.
To initialize the flag with 0, we can use calloc memory.

{Time Complexity = O(n)}

Note
TC of kruskal Algorithm when edges weight ~~cons~~ are already sorted. (O(E) + (E+V)
↳ by ~~cycle~~ DFS. cycle also checked
O(E+V)

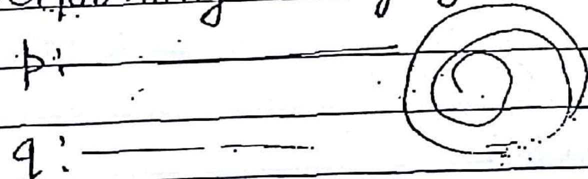
Algorithm 2 - Apply BFS or DFS

Algorithm3- p & q will start from same place p will go slower than q. if p & q never meet, then no cycle otherwise no cycle-



p   q   p increment by 1
1000 — 1000   q by 2
2000 ← 3000
3000 ← 5000
4000 ← 7000
5000 → 9000
6000 ← 11000
7000 — 7000 (cycle is present)

- You can increment q by any value, it is possible that at end it is performing many cycles



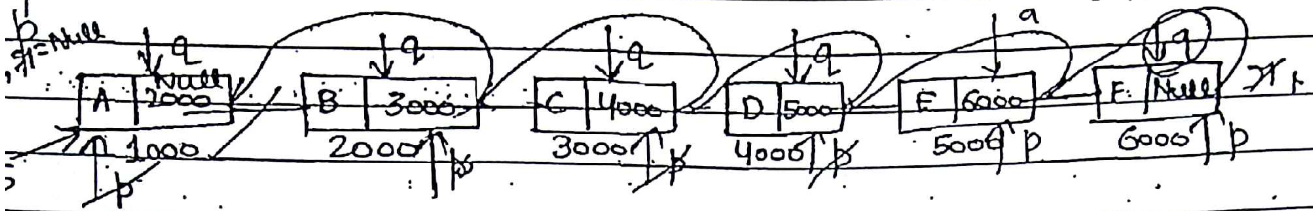
1. p = q = s
2. p by 1
   q by 2
3. if (p == q)
   pf(cycle)
   else
   goto step 2.

Time Complexity = O(n)
Space Complexity = O(1)
Time Complexity is due to p only.

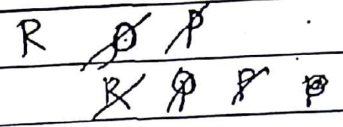
EXCELLENT

Page No.
Date: / /

Ques – WAP in C to reverse a linked list.

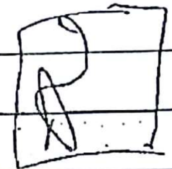



```
(struct node *) Reverse (struct node *s)
{
    p=null;
    r=s->next;
    while (r != NULL)
    {   s->next=p;
        p=s; s=r;
        r=r->next;
    }
```




```
p=s
q=r=Null
while (p != next)
{
    r=q;
    q=p;
    p=p->next
    q->next=R;
}
s=q;
return s
```

andsoon

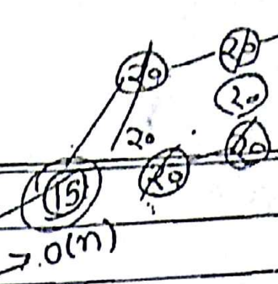


~~Time Compt~~

{Time Complexity – O(n)}
(WC, AC, BC)

To print data of linked list in reverse order (singly linked list)
TC = O(n) + O(n)
↑ to reverse ↑ to print

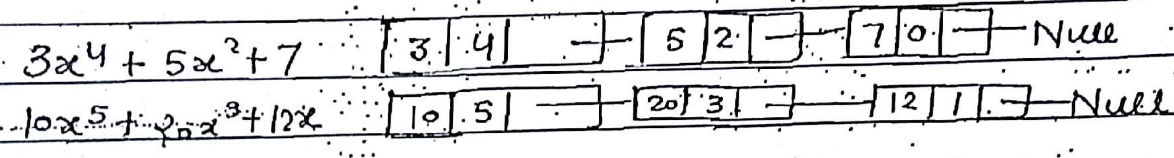
To print data of doubly linked list in Reverse order
= TC = O(n)



HomeWork-

1. WAP in C to perform concatenation, Union & Intersection b/w two linked list → O(nlogn)
2. Cardinality algorithm
3. Membership algorithm
(2, 3) → O(n)

4. Write a P in C to implement Linked list using array
5. Write a P in C to implement stack using linked list.
6. Write a C program to implement Queue using LL.



to add two polynomial-

```
if L1→pow == L2·pow
    l3·coff = L1→coff + L2→coff
    l3→pow = L1·pow
else
if L1→pow > L2→pow
    l3→coff = L1→coff
    l3→pow = L1→pow
else
    l3→coff = L2→coff
    l3→pow = L2→pow
    L = L→next;
```

Time Complexity = O(m+n) (when no add)
↓ (WC)
O(m) best case (when all power getting added to other list.)
↳ due to move.

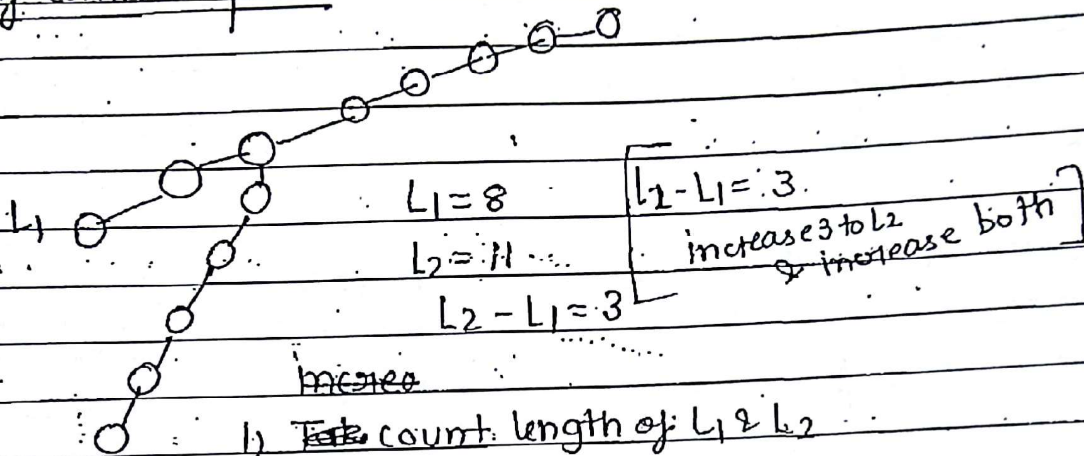
WAP in C to perform polynomial addition & multiplication using linked list → O(m·n)
↳ each value get multiplied to other

EXCELLENT

Page No.
Date : / /

~~Drawback of singly linked list-~~

finding common point-



1) count length of L_1 & L_2
2) find $L_1 - L_2 = c$
3) find whether L_1 or L_2 greater
4) initially increase greater than c time
5) then increase both at a point of time they will meet
7) start count from point of meeting

Complexity- $O(m) + O(n) + O(n-m)$

- $O(m)$ ↓ to count no of elmt
- $O(n)$ ⇓ to count no of elmt
- $O(n-m)$ ⇓ (to calculate length)

Drawback of singly linked list-

1.) We cannot go back because every node contain only one pointer & that pointer also contain next node address.

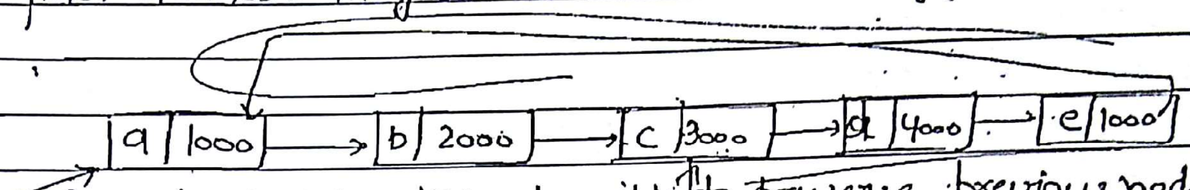
2. In singly linked list, last node next part is always null i.e., we are not utilising it properly.

To eliminate above two drawbacks, we are moving toward circular singly linked list.

EXCELLENT

Circular Singly linked list-

- In single linked list, last node next part, if we restore first address they it became circular singly linked list.



due to this, it is possible to traverse previous node as well.

- We can go back by visiting last node and them first node & so on. but it will take O(n) time.

in doubly linked list, visiting last node is O(1) but it may take some space.

So Here, to check end of ~~loop~~ list-

```
p=s
while (p->next != s)
    p=p->next;
```

p will finally point at last node

to add a node at last of LL

q (node created) ——> O(1)

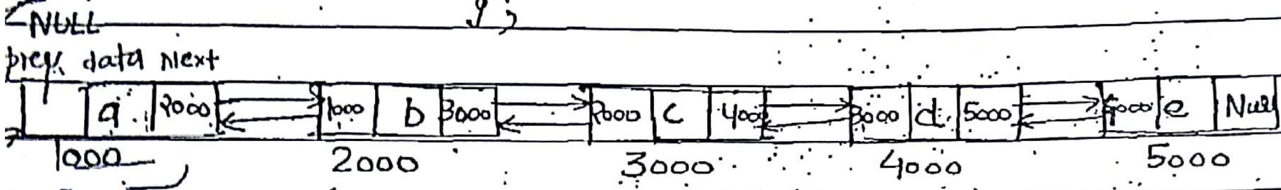
```
p=s;
while (p->next != s)     } // goto last node O(n)
    p=p->next;
p->next=q;               } linking node
q->next=s;               } O(1)
```

→ {TC= O(n)}

Date : / /

An operation in Singly linked list which is independent of length of list = Insertion at front.

Doubly linked list- struct node
{
~~int data;~~
struct node *prev; int data;
struct node *next;
};



o print data in reverse • go to last
o print by s = s → prev

first node = s → prev = NULL
Last Node = s → next = NULL

Ques.) - WAP in C to insert a node with data x at end of DLL.

```
struct node *q = (struct node *) malloc (sizeof (struct node));
q => data = x
q -> prev = q -> next = NULL
if (s == NULL)
    return NULL
while (s -> next != NULL)
    s = s -> next;            } O(n)
s -> next = q;
q -> prev = s;
```

TC = O(n)

EXCEL

Page No.
Date: / /

Ques WAP in C to insert a Node with data x before a node with data y in DLL.

```
struct node *p;
p=(struct node *) malloc (sizeof(struct node));
p->data=x;
p->prev=p->next=NULL;
while(s->data!=y && s->next!=NULL)
    s=s->next;
if(s->data==y){
(s->prev)->next = p;
 p->prev = s->prev;
 p->next = s;
 s->prev = p;
}
```

order is important

connect outside people first

```
p->prev= s->prev
p->next=s
s->pre = p
p->pre->next = p
```

order matters at 1st & 4th stmt (1,3)

Ques

WAC to delete a node at last in DLL.

```
① while(s->next !=NULL)
    s= s->next
② (s->prev)->next = NULL
③ free(s);
④ s=NULL;
```

Ques - Delete a node in middle

```
while (s1!=NULL)
{ c++;
} s1=s1->next;
c=⌈c/2⌉;
while(c!=1)
{ s=s->next;
```


EXCELLENT

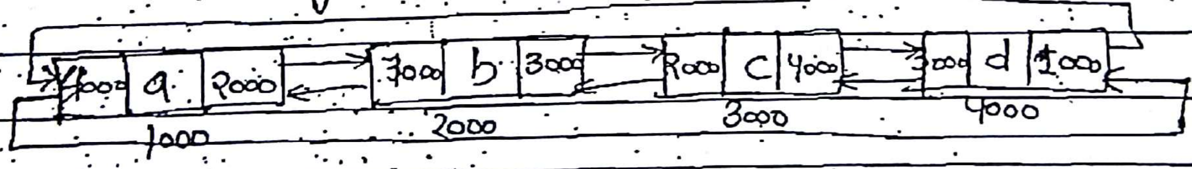
(s->prev)->next = s->next;
~~s->next s->prev~~
(s->next)->prev = s->prev;
free(s);
S = NULL;
}

Ques- WAP in C to delete a node with data x in DLL.

```
(struct node*) deletex (struct node*s)
{
    while (s->data != x & s->next != null)
        s = s->next;
    if (s->data == x);
    {  (s->prev)->next = s->next;   } only Modification
       (s->next)->prev = s->prev    }
       free(s);
       S = NULL
    }
    else if (s->next == NULL)
        No Node with data x
}
```

order is important

Circular Doubly linked list-



	CLL	DLL	CDLL
3-4	O(1)	O(1)	O(1)
1-3	O(n)	O(1)	O(1)
ton	O(n)	O(n)	O(n)
1 to 1	O(1)	O(n)	O(1)

No random access possible as to visit 10th node you have to traverse 10 node.

Page No.
Date : / /

In Circular DLL, O(1) time to add node at last.

p = s → prev
p → next = q
q → prev = p   q - new node.
q → next = s
s → prev = q

add a node at start = O(1)
: middle
Delete ~~all~~ a node at start

In Circular DLL - O(1) time to concatenate two ~~le~~ list.
(s1, s2).

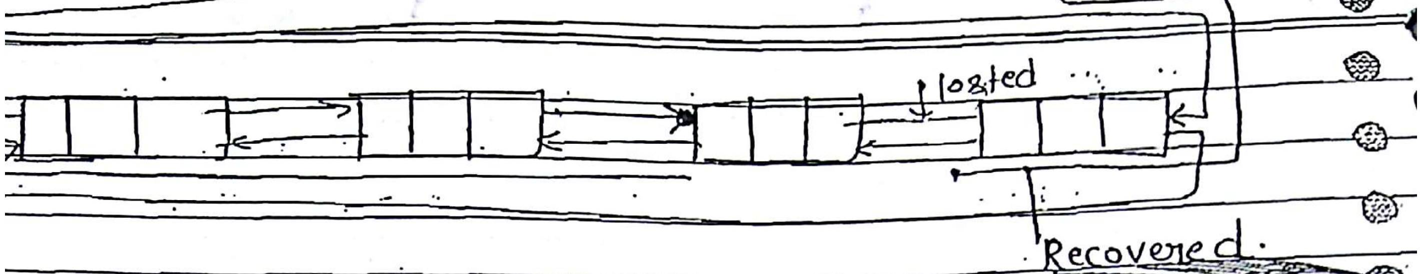
p = s1 → prev;
p → next = s2;
~~s2 → prev = p~~
q = s2 → prev;
s2 → prev = p;
s1 → prev = q;

concatenate DLL - O(m)

In circular linked list - O(m+n)
because two list has to be traversed

In CDLL - if a link get lost you can use another link to recover.

The greatest advantage with CDLL or DLL is if you lost one pointer, with help of another pointer, we can get back. [there is a possibility

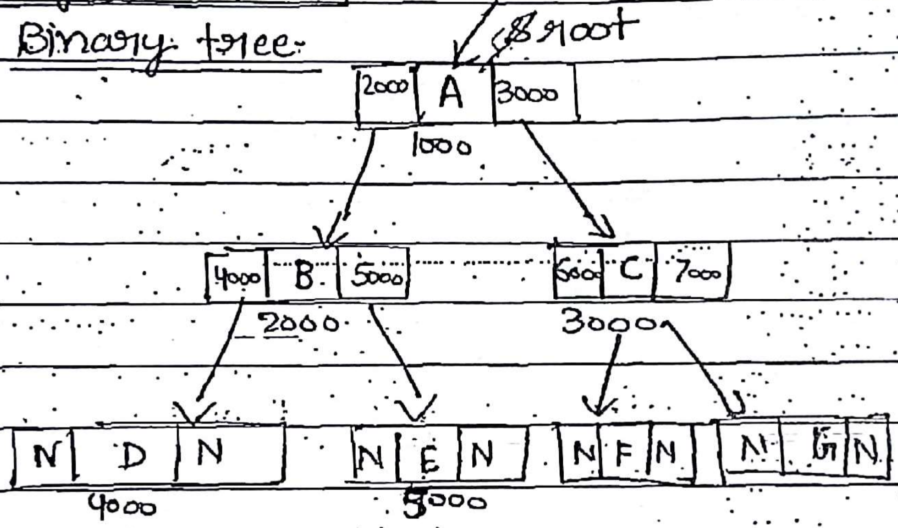


There is no possibility of recovery of link in single linked list.

but if you losted both pointers of a node in list, you cannot recover even with CDLL.

Application of ~~Binary~~ DLL-

1) Binary tree-



In DLL, a small change is done in DLL, it does not point back to previous node but it point to another node

Each node is pointing to two node, acting as child.

prev → left pointer

next → right ptr

When a tree is given, address of root is given i.e starting node is given.

```
Struct BTnode
{
    structBTnode * lc;
    int data;
    struct BTnode *rc;
};
```

print(root) - 1000
root → lc = 2000
root → rc = 3000

if (root == NULL) tree is empty.
if (root→lc == NULL && root→lc == NULL) a single node.
if (node→lc != NULL || node→rc != NULL) ⇒ Internal node
If (node→lc == NULL && node→rc == NULL) ⇒ leaf node.

Height = 2. (path length from root node to leaf node)
||
No of level - 1

Ques - To findout left ~~mode~~ most node of given tree.

```
while (root→lc != NULL)
    root → root → lc.
```

. Here random access not possible, so traverse is ~~also~~ only done from root.
Use linked list for binary tree, when it is not almost complete binary tree or complete binary tree because it will save space.
No formula to access parent the node.
Use Array for BT where BT is ~~also~~ almost or complete binary tree.
if CBT or ACBT, we used array, else use linked list only.

When No Mention about BT- use linked list

Page No.
Date: / /

Tree-Recursion

Q- WAP in C to count total no of leaf node in given binary tree

```
int leaf(struct node *root)     m=0 initial
{   if(root==NULL)
        return(0);
    if(root->left==NULL && root->rc==NULL)
        return 1;
    else
        m=m+ leaf(root->left)+ leaf(root->right);
}   return m;
```

$T(n) = 2T(\frac{n}{2})+1 = O(n)$

(BC, WC, AC) = O(n) because total tree has to be visited once.

~~int internal~~

(Unbalanced tree)

Worst case- $T(n)=T(n-1)+c$
$= O(n)$

Termination condⁿ for ~~leaf no tree~~ = leaf nodes

Stack space = O(log n) BC
O(n) WC

Ques WAP to count non leaf node in given tree-

```
        m=0;
int non-leaf(struct BTnode *root)
{
        if (root==NULL)
            return(0).
        else if(root->left==NULL && root->rc==NULL)
            return 0;
        else
        {
            m = 1 + non-leaf(root->lc) + nonleaf(root->rc);
        }
        return m;
}
```

Time Complexity = $2T(\frac{n}{2}) + 1$ (Best case)

$= O(n)$ (Balanced tree

Stack space = O(log n) (Balanced tree)

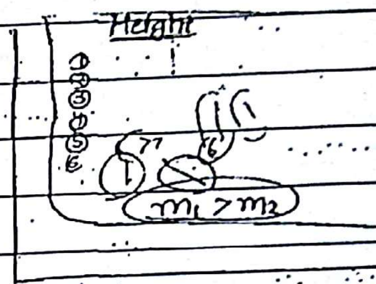
Page No.
Date : / /

Worst case - when tree is an unbalanced tree

$$TC; \quad T(n) = T(n-1) + c$$
$$= c.n$$
$$= O(n)$$

Space of stack = O(n) (due to n levels)

(For any tree, Recursion is used & leaf node serves as termination bcoz you cannot move further)



Ques. WAP to find total no of nodes

```
total node{
  if (root == NULL) return 0
  else
     m = 1 + total_node(root->lc) + totalnode(root->rc);
}
```

or

```
{ if (root == NULL) return 0
  if (root->lc == NULL && root->lc == NULL)
     return 1
  else
     m = 1 + totalnode(root->lc) totalnode(root->rc);
}
```

EXCELLENT

```
1. if (root == Null) return 0
   if (root -> lc == NULL && root -> rc == NULL)
           return 1;
   else
   { xL = CNLN(root -> lc)
     xR = CNLN(root -> rc)
     C = 1+xL                 => return leftmost path visited
     return(c);                                      count.
   }
```

{ if c = xL -> leaf node in leftmost path
  c = 1+xL & return(0) -> non-leaf node in leftmost path }

Ques- WAP in C to count height of a given BT

```
int height_of_BT(struct node * root)
{
        if (root == NULL)
            return 0;
        if (root -> lc == NULL && root -> rc == NULL)
            return 0;
        else
        {
            hl = 1 + height(root -> lc);
               hr = 1 + height(root -> rc);
         if (hl > hr)
            return(hl);
          else
            return(hr);
        }
```

[See the code ...]

Page No.
Date: / /

Height of a Binary Tree = the length of the longest path from root to leaf node.

TC → $T(n) = 2T\left(\frac{n}{2}\right) + K$ (Balanced Tree)

$= O(n)$

$T(n) = T(n-1) + C + T(1)$ (UnBalanced Tree)

$= O(n)$

(BC, AC, WC)

Stack Space – $O(\log n)$ (Best case Balanced tree)
$O(n)$ (Worst case unbalanced tree)

You can find height by finding no of level if you wanna then just count leaf node as well. then find no of level just count leaf node as well. finally subtract by 1.

x_L = height(root→lc)
x_R = height(root→rc)
c = 1 + max(x_L, x_R)

To count no. of levels –

```
1 if(root == Null) return 0.
② if (root → left == Null && root → right == Null)
       return 1;
③ else { xL = level (root → lc);
         xR = level (root → rc);
         C = 1 + max(xL, xR);
         return C;
  }
```

{ if leaf node return 10 → it will give (height + 10) }
if C = 15 + max(x_L, x_R)
return 10 in ② then 15 times height to 10

Use Brain

EXCELLENT

Ques Write a C Program to verify given BT is Strict BT or node.

```
bool Verify_Strict_BT(struct node *root)
{
    1. if(root == Null)
           return 1;
    2. if(root->lc == Null && root->rc == Null)
           return 1;
    3. else {
           a = Verify_Strict_BT(root->lc);
           b = Verify_Strict_BT(root->rc);
           If(a==b)
               return 1
           else
               return 0;

SBT(root)
{ 1. if(root == NULL)
         return 1;
  2. if( root->lc == NULL && root->rc == NULL)
         return 1;
  3. else
     { if( root->lc != NULL && root->rc != NULL)
         return(SBT(root->lc) && SBT(root->rc));
       else
         return 0;
```

You cannot write the boolean exp by dividing in two sub expression (boolean exp) donot divide) so time & space get wasted as if SBT(r→l) && (SBT(r→l) .
if is →o → no need of doing right traversal.

Time Complexity:- When it is strict binary tree

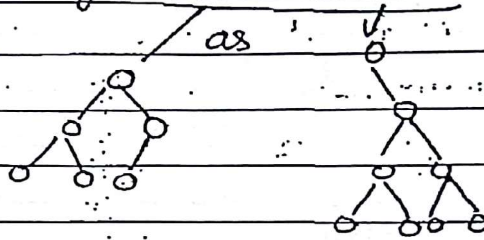
$$T(n) = 2T(\frac{n}{2}) + c$$

$$= O(n) \text{ (Worst case)}$$

When it is not a strict binary tree

it may be O(n) or O(1).

as



So Time Complexity, { BC = O(1) , WC = O(n) }

If code is written as

{ a = SBT(r→lc)
b = SBT(r→rc)
c = a && b

then TC, T(n) = O(n) (WC, AC, BC)

because no use of property of short circuit.

Modification:- if (roo........)
return (SBT(root→lc))

Check whether leftmost path is SBT.

WAP in C to verify given two Binary trees are equal or not

```
equal(root1, root2)
{
    if(root1 == NULL && root2 == NULL)
        return 1;
    if((root1->lc == NULL && root1->rc == NULL) && (root2->lc
        == NULL && root2->rc == NULL))
    { ~~return 1;~~
        if(root1->data == root2->data)
            return 1;
        else
            return 0;
    }
```

(wrong)

```
① if(root1 == NULL && root2 == NULL) return 1;
② if(root1 == NULL && root2 != NULL) return 0;
③ if(root2 == NULL && root1 != NULL) return 0;
④ if((root1->lc == NULL && root1->rc == NULL) &&
      (root2->lc == NULL && root2->rc == NULL))
   { if(root1->data == root2->data)
        return 1;
     else
        return 0;
   }
⑤ else
   { if(root1->data == root2->data)
     {
                               root2->lc            root2, rc
        return (equal(root1->lc) && equal(root2->rc));
     }
     else
        return 0;
   }
```

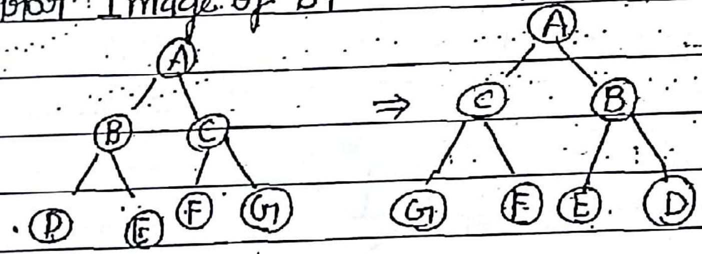
Page No.
Date: / /

Here it is not important to put fourth condⁿ, it may terminate when root is empty.

```
equal(r1, r2)
① if(r1==NULL && r2==NULL) return 0;
② if((r1==NULL && r2!=NULL) or (r1!=NULL && r2==NULL))
      return 0;
   else { if (r1->data == r2->data)
          ~~return(equal(r1->lc, ) &&~~
            return(equal(r1->lc, r2->lc) && equal(r2->lc, r1->lc))
          else
            return 0;
        }
```

TC, T(n) = O(n)

Ques- Mirror Image of BT



~~Mirror Image(~~

Write a Program in C to convert given binary tree into its mirror image. mirror(root)

```
1. if(root==NULL)
      return NULL;
2. if(root->lc==NULL && root->rc==NULL)
      return root;
3. else
   {
      ~~swap(root->lc, root->rc)~~ //swaping of nodes
      ~~l1= mirror(root->lc);~~
      ~~l2= mirror(root->rc);~~   l = root->rc
      root->rc = mirror(root->lc);
      root->lc = mirror(root->rc);
```

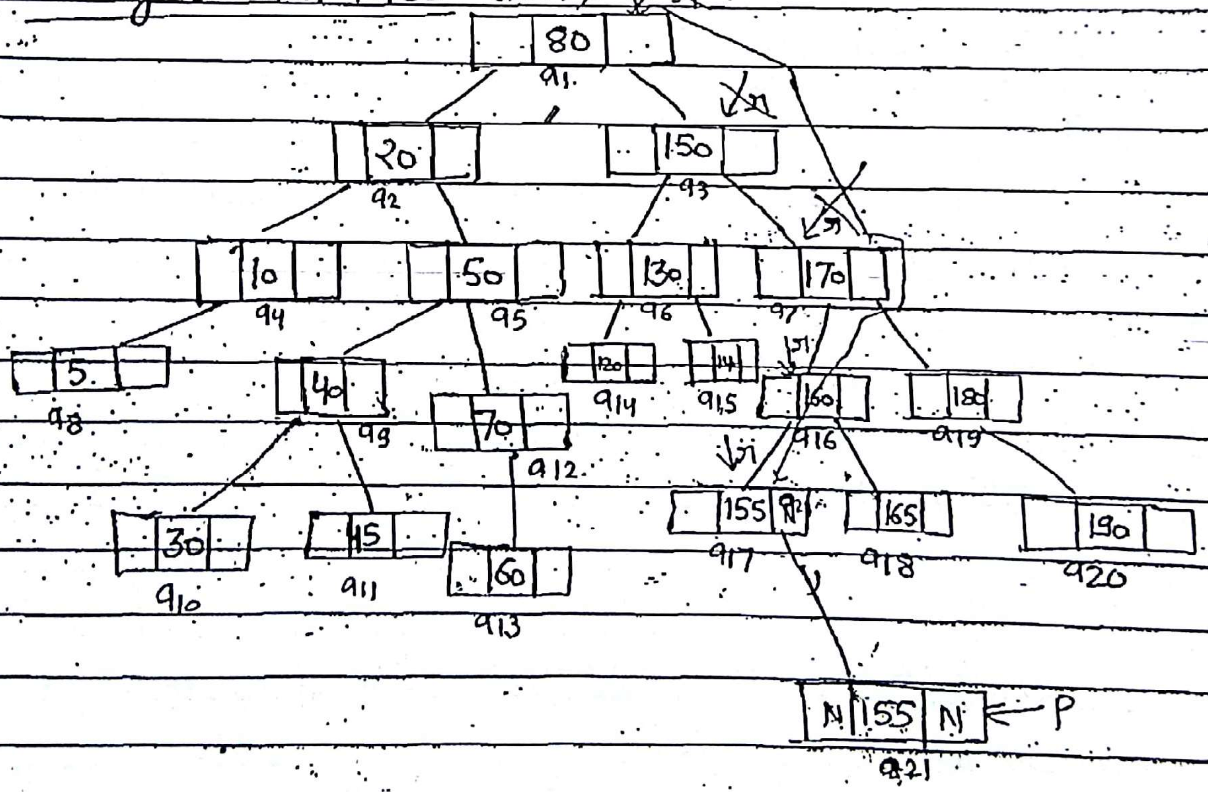
EXCELLENT

```
Mirror I (root) :
{
    if (root=NULL) return root
    if (root→lc==NULL && root→rc=NULL)
            return root;
    else
    {   t=root→rc;
        root→rc= Mirror-I(root→lc);
        root→lc= Mirror-I (t);
    }
}
```

T.C, $T(n) = O(n)$

- To convert into Mirror image
  - firstly convert the left tree store to right of root.
  - Again convert right to mirror image & store to left of root.

Binary Search tree (BST).



Page No.
Date : / /

1. BST with n nodes, Minimum level = $\log_2(n+1) \simeq \log n$
Maximum level = n

2. AVL tree with n nodes, no of level = log n (Max & Min).
(because it is a balanced BST)

3. To find smallest person in BST, goto leftmost node

```
while(r→lc != NULL)
{
    r = r→lc
}
```

[Best case - O(1)
  Worst case = O(n) (all element in left side /)]

4. To find largest element in BST, goto rightmost node

```
while(r→rc != Null)
    r = r→rc
```

[Best case = O(1) (root itself is greatest)
  Worst case = O(n) (all elmt in right side \)]

5. To find smallest person in Binary tree.
O(n) (BC, WC, AC)
[as all elmt. has to be traversed]

Maximum element - O(n)
(WC, BC, AC)

6. Smallest element in AVL tree = O($\log_2 n$) (~~Maximum~~ & minimum)
(BC, WC, AC)

O(1) - not possible as no case (\) (/)

EXCELLENT

If tree is given, if not mentiod, it is provided with LL.

[Searching an element x] –

- Binary Tree – Worst case – $O(n)$ ⇒ $2T(n/2)+1$
  - Best case – $O(1)$
- Binary Search Tree – Worst case – $O(n)$ ⇒ $T(n-1)+1$
  - Best case – $O(1)$
- AVL tree – Worst case – $O(\log n)$ ⇒ $T(\frac{n}{2})+1$
  - Best case – $O(1)$

• To say, element x is not there, (Best case)

- Binary Search Tree – BST – $O(1)$ when (Let I was searching for 48 & root = 80 & root → lc = NULL)
  - Worst case – $O(n)$
- Binary Tree – Worst case – $O(n)$
  - Best case = $O(n)$ (all elmt to be checked)
- AVL tree – Worst case = $O(\log n)$
  - Best case = $O(\log n)$

Insertion of a node –
- Initially create a Memory
- traverse to Right pos^n upto u find Null
- link node

Best case = $O(1)$

if Root – 80 4 wanna insert 1000.

Root → rc = Null.

∴ insert it.

Worst case = $O(n)$

check Root
is Root 80 158
no
greater than root
go right
if right = NULL
insert there

Algo:- 1. Create a node $\Rightarrow O(1)$
2. Traverse to Right pos^n, Find place $= O(n)$
3. Link $\Rightarrow O(1)$

$O(n)$ (WC)

In AVL trees, Insertion of a node,
Best case $= O(\log_2 n)$ (because only at last level, null will be there & insertion is done when null is there)
Worst case $= O(\log_2 n)$.

IN BST, always insertion will be done always at null place.

Searching an element in min heap $= O(n)$

It is not the case that value close to root will be placed closed to root in tree

Ques- A element to be inserted in min heap but checked first whether exist or not
\\ TC $= O(n) + O(\log n)$

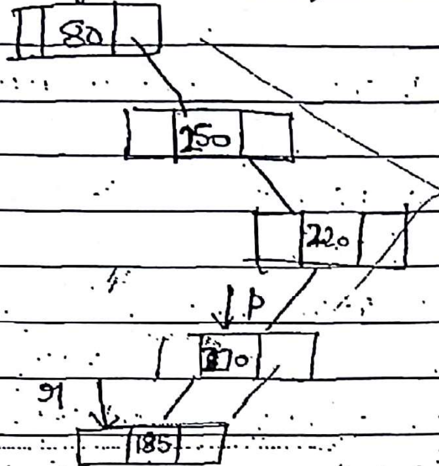
In BST TC $= O(n)$ (to search to find it is there + insert as after finding null you can directly insert it.

In AVL TC $= O(\log n)$ as (to search 'null' or to know it is there or not logn time & then insertion is directly done)

Data Structure- Insertion an element if data not there $= O(\log n)$

Deletion of a Node- • Deletion of node with 0 child

- Delete x
  - Find x
  - Store parent of x in p (going with find) (because it is not possible to come back)

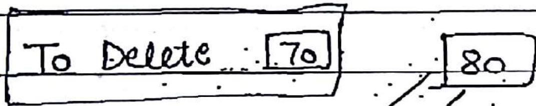


O(n) time to search
- Worst case

O(1) time to search
- Best case

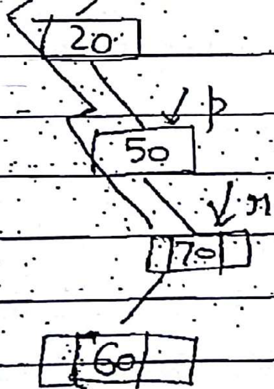
```
p->lc = null;
free(p r);
r = NULL;
```

⇒ T.C = O(n) Worst case



To Delete 70

Deletion of node with 0 child.



- to find x
- check for pos^n of child (left, right)

```
p->rc = r->lc
free(r)
r = NULL
```

T(n) = O(n) (Worst case)

• Deletion of node with two child–

predecessor of a node – greatest Elmt in left subtree.
Successor of a node – smallest elmt in right subtree.

5 10 20 predeccessor is found by going rightmost path in left subtree.
Successor by going leftmost node in right subtree.

• ~~Delete the node~~
• ~~Del Replace by~~
• Replace the node by its predeccessor or successor
• Delete the node get replaced.

A node is said to be predeccessor ~~left poin~~ if Right ptr is null; left ptr may or may not be null.

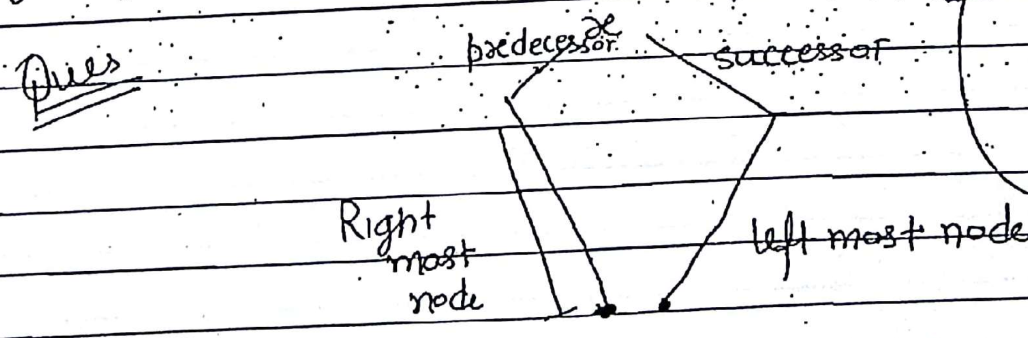
• predeccessor is always leaf (No)

Successor – left Most pointer must be null.
• Right Most pointer may or may not null.

predeccessor or successor of a node on maximum can have one child only.

Best case – find ~~max~~ = O(1)
find predeccessor = O(1)

Ques



- If finding element takes $O(n)$ then finding predecessor will take $O(1)$ time only as no of levels are n only.
- If finding element takes $O(1)$ time then finding predecessor can take $O(n)$ in worst case.

Deletion - Algorithm -

① Find element(x) ⇒ $O(n)$, $O(1)$ (WC, BC)

② (i) 0-child - delete simply by replacing Null at parent node. ⇒ $O(1)$

(ii) 1-child - delete x by connecting graph father & grand child. ⇒ $O(1)$

(iii) 2-child - (a) Find successor or predecessor
(b) replace x data by predecessor data or successor data.
(c) delete the node (ie predecessor or successor)

Total time taken for finding an element & predecessor $= O(n)$

So on total - Delete time complexity $= O(n)$ (WC)
$O(1)$ = Best case

It is never possible that finding $x = O(n)$ & finding predecessor $= O(n)$ as No of level is only $O(n)$ but not $2n$.

To Create BST with n elements, $O(n^2)$ time required (n insertion & each insertion will take n time);

EXCELLENT

(left height & right height must be equal)

AVL Trees – • Balanced binary search tree.

(Adelson, Valksy, Landis)

Balanced BST = Height Balanced BST (no of level = logn)

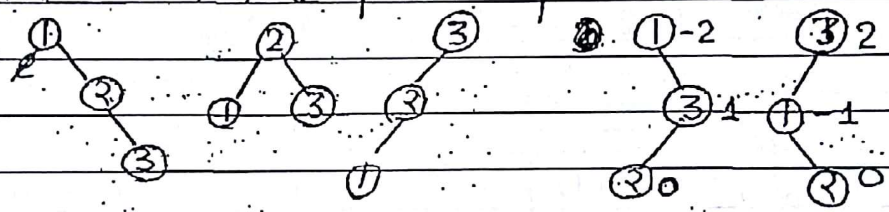
A BST Balanced is said to be height balanced if

H(LST) − H(RST) ⇒ 0, +1, −1

[Height of Left subtree] — Balanced factor

Balanced factor = H(LST) − H(RST)
= 0, −1, +1 only

Example – n=3 {1, 2, 3}. find all possible BST.

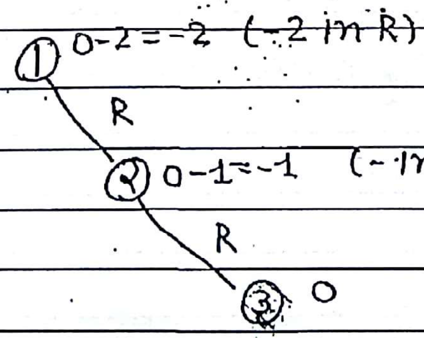


BST but not AVL — BST as well as AVL — BST but not AVL — BST but not AVL

(−) means greater height in right side

(+) greater height is in left side.

+2 ⇒ left is greater by 2

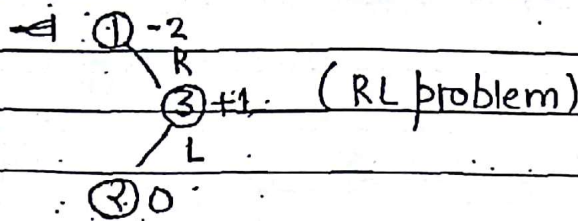


(It is RR problem because right side is incremented by 2)

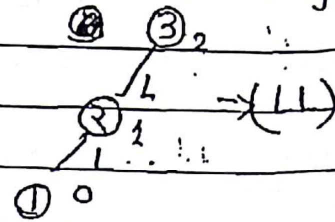
EXCELLENT

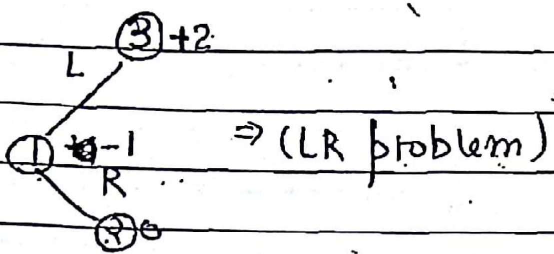
Each AVL is BST but, each BST is not AVL

start giving ~~from starting & go on to zero~~



(RL problem)

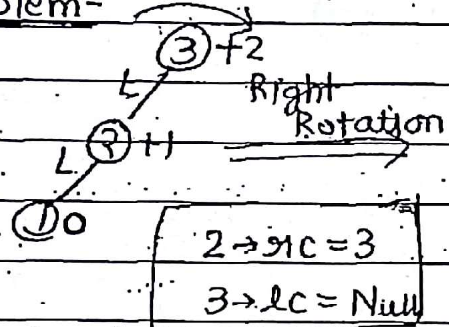


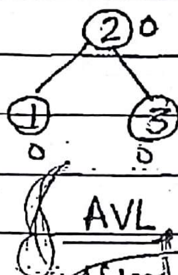


⇒ (LR problem)

If LL → Rotate Right
RR → Rotate Left

LL problem-





2 → rc = 3
3 → lc = Null

AVL

the node which have problem ie which is not balanced. Rotate it.

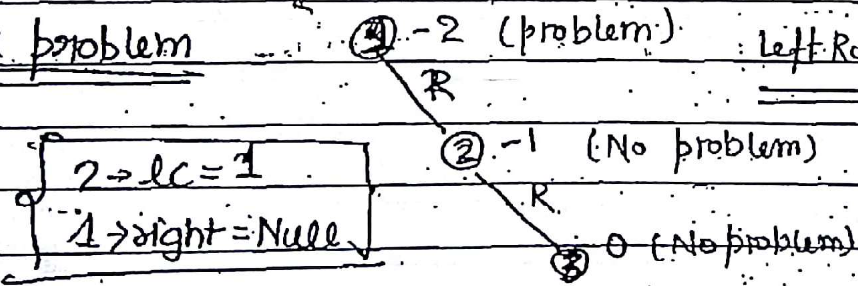
After rotation, ~~parent A & child~~ B get rotated as Parent B child A

When parent fall down in right side ⇒ Right Rotation

$TC = O(1)$

~~Left Rotation~~

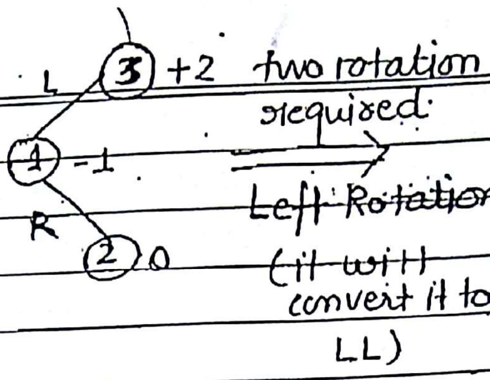
RR problem



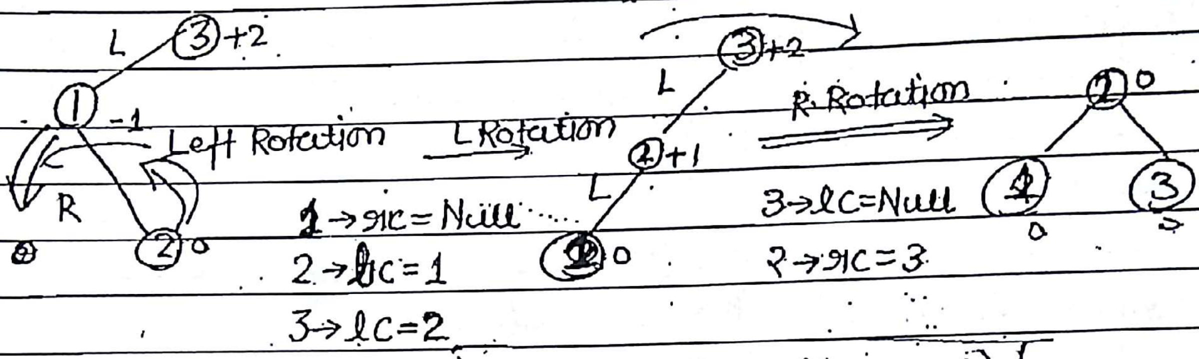
2 → lc = 1
1 → right = Null

Time Complexity = O(1)

LR problem-



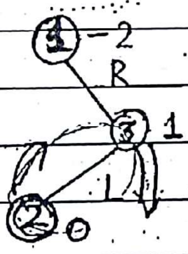
two rotation required → Left Rotation + Right Rotation
(it will convert it to LL) (then LL get solved)



| Time Complexity = O(1) |

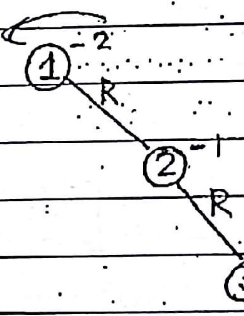
O(1)

R.L. problem



Right Rotation →

1→rc = 2
2→rc = 3
3→lc = Null



Left Rotation →

2 0
1 3
0 0

2→lc = 1
1→rc = Null

Time Complexity = O(1)

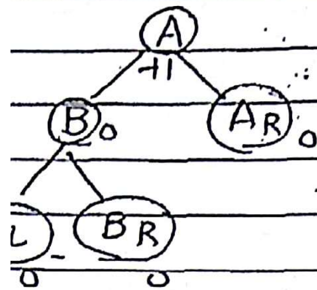
(After Rotation, parent & child get exchanged)

EXCELLENT

Page No.
Date : / /

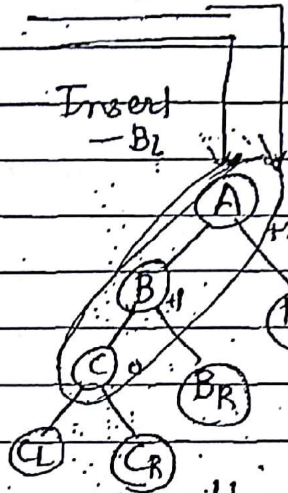
Insertion - LL Problem -

A_R, B_L & B_R are 3 AVL trees
to obtain LL problem add Node to B_L.



Add a node to AVL may or may not create a problem

Insert B_L



$O(1)+O(\log n)+O(1)$
creation of node — finding null — link

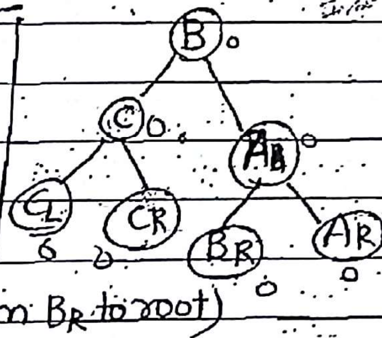
(a Tree has Balance factor +1, it is close problem ie not an tree)

check for AVL
$O(\log n)$

Rotate

Right Rotate

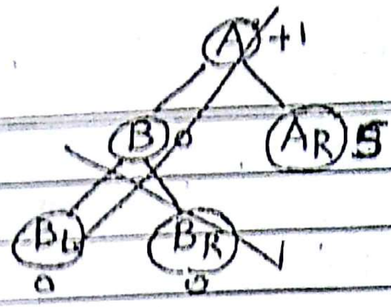
$\therefore$ Total comp = $1+\log n+1+\log n$
$= 2\log n$
$= O(\log n)$



Here B_R must follow the same path as it was showing in original LR (from B_R to root)

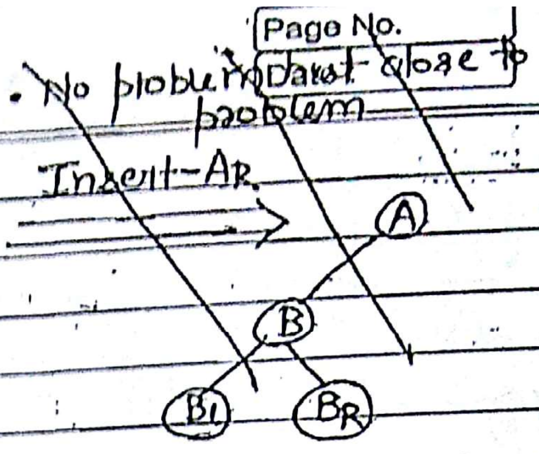
EXCELLENT

RR problem

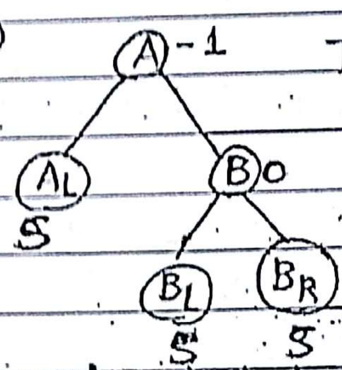


• ~~No problem but close to problem~~

~~Insert-AR~~



• No problem but close to problem.

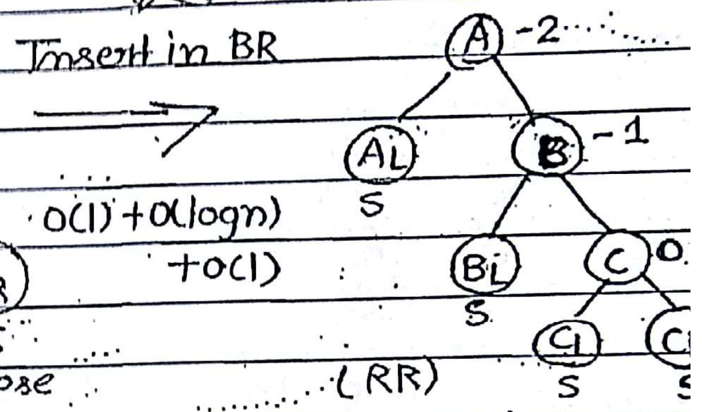


AVL but close to RR

(to get RR)

Insert in BR

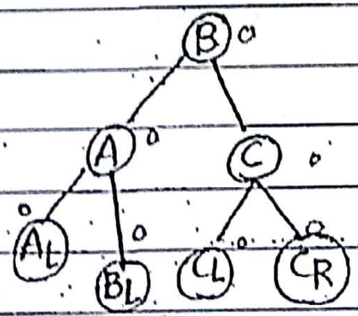
$O(1) + O(\log n) + O(1)$



(RR)

$O(\log n)$

Check for problem



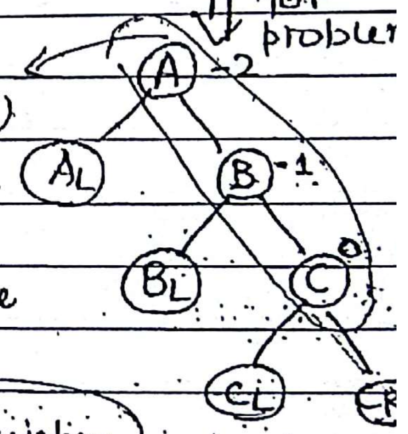
(AVL

(We came to know about RR by − sign)

⇓

RR problem

Left Rotate



B_L become victim

Time complexity $= O(1) + O(1) + O(\log n) + O(\log n)$

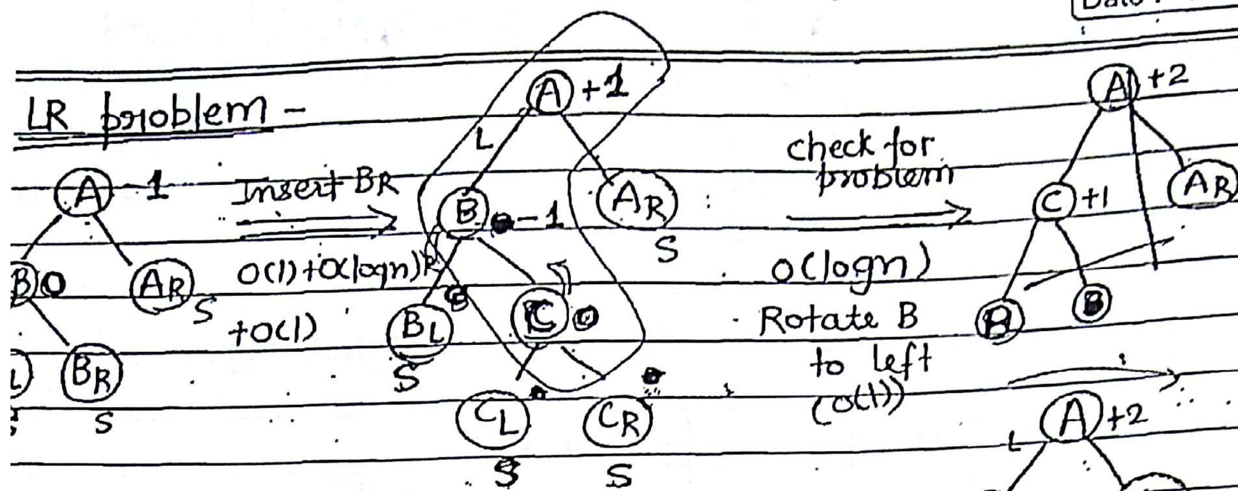
$= 2\log n$

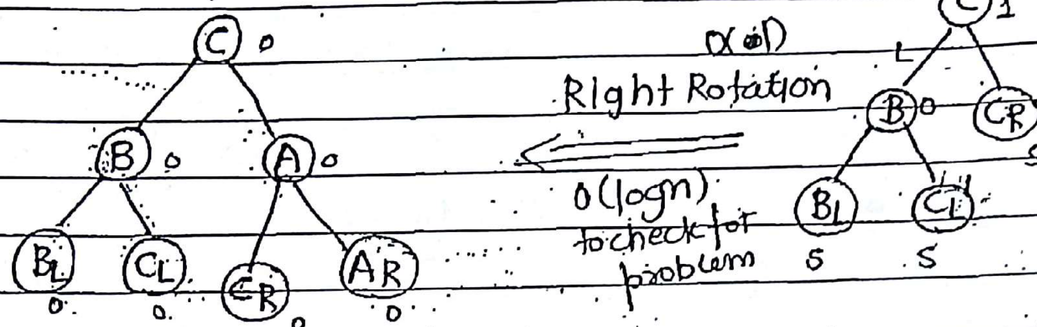
$= O(\log n)$

To attach B_L, check its behaviour 'Less than B more than A' do it in same way.

(To check for problem same path has to be taken to root from which path you reach to that node)

LR problem -





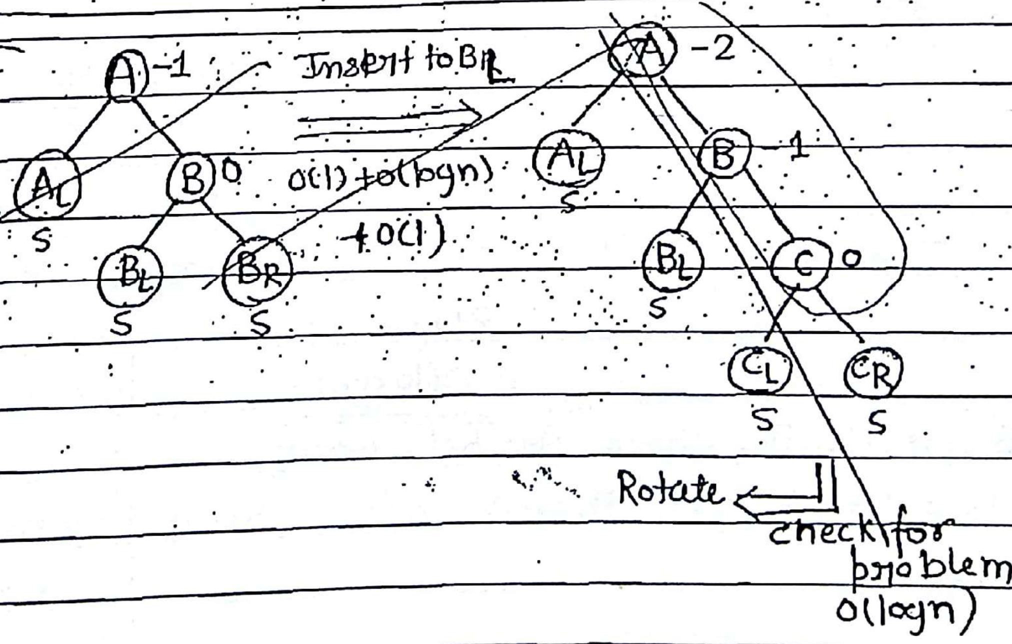
Total time = O(log n) + (log n) + log n

⇓ left Rotate ⇓ Right rotation

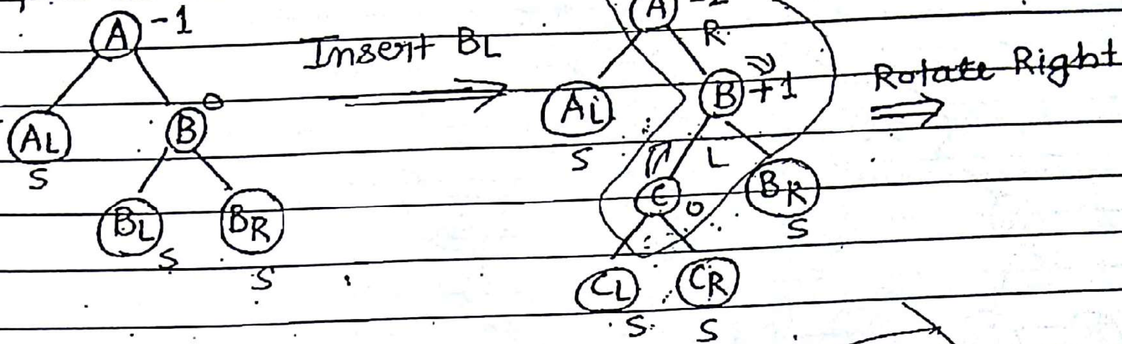
= 3 log n

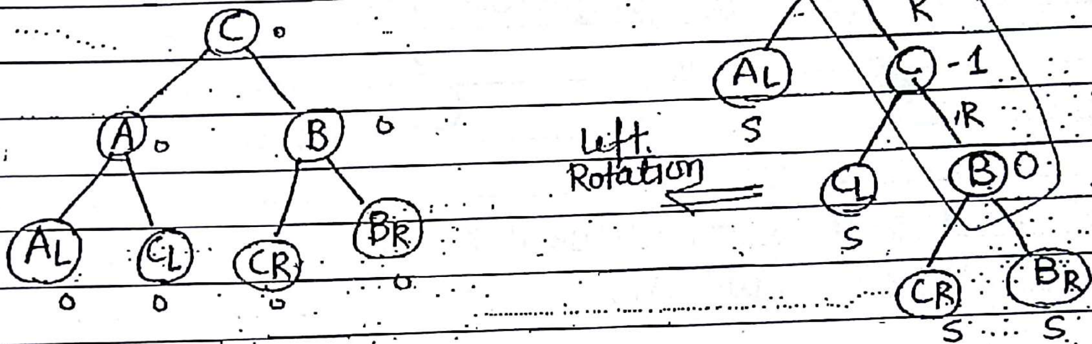
= O(log n)

L problem-



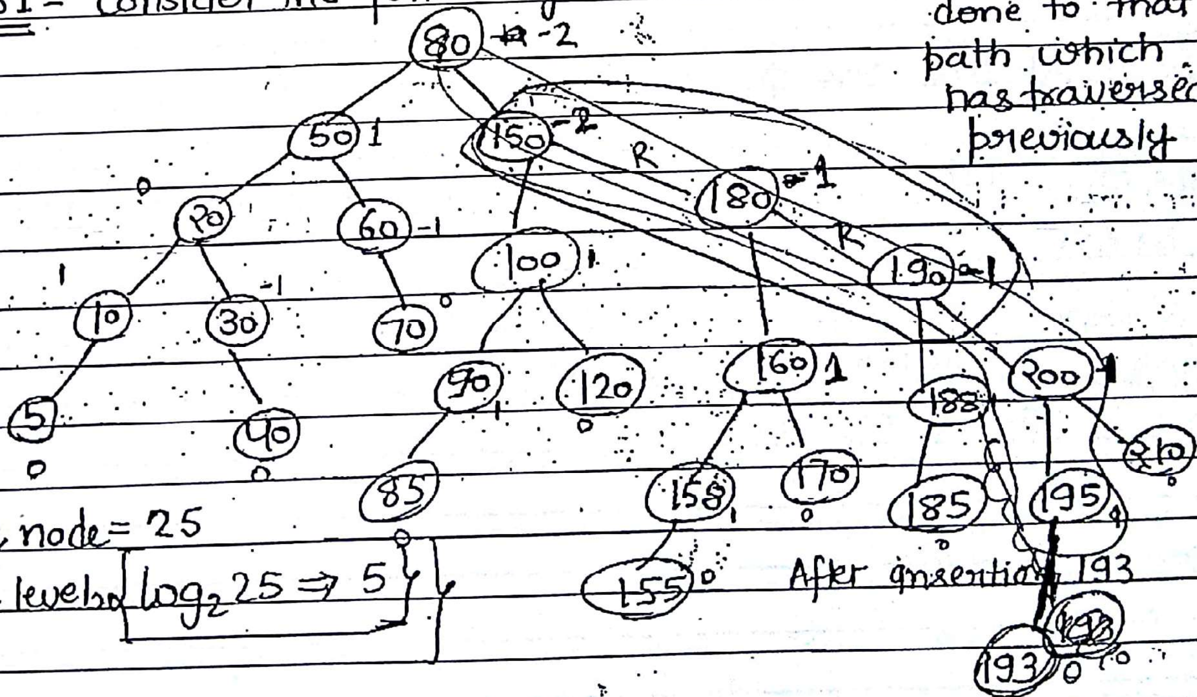
RL problem





Ques 1 - Consider the following AVL tree-

check for AVL done to that path which has traversed previously

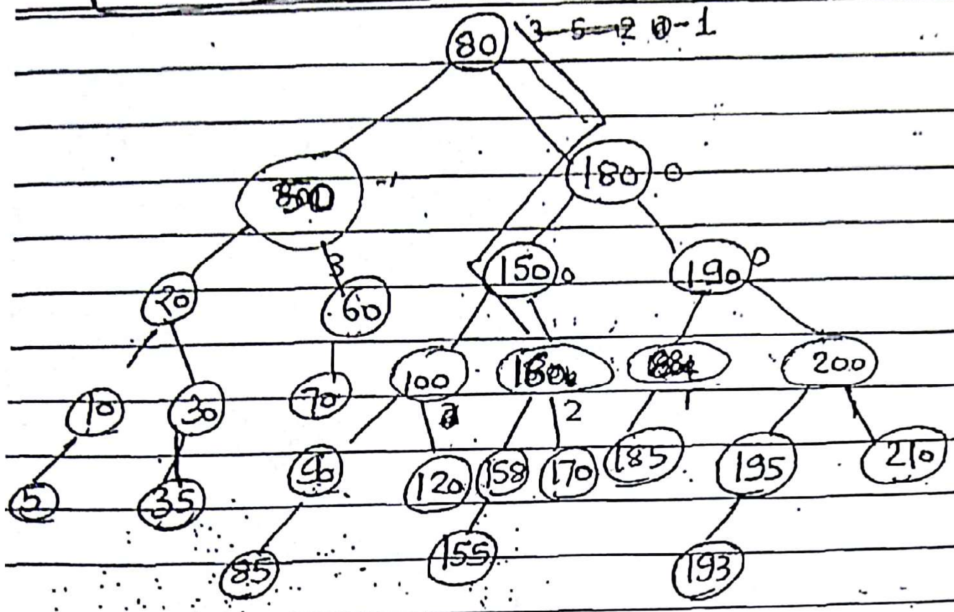


no of node = 25

{no of levels $\log_2 25 \Rightarrow 5$ }

O(1) + O(log n) + O(1) + O(log n) + O(1)
creation ⇓ find ⇓ link ⇓ check ⇓ Rotate

Page No.
Date: / /



Note – Insertion an element into AVL tree will take O(log n) time (BC, WC, AC).

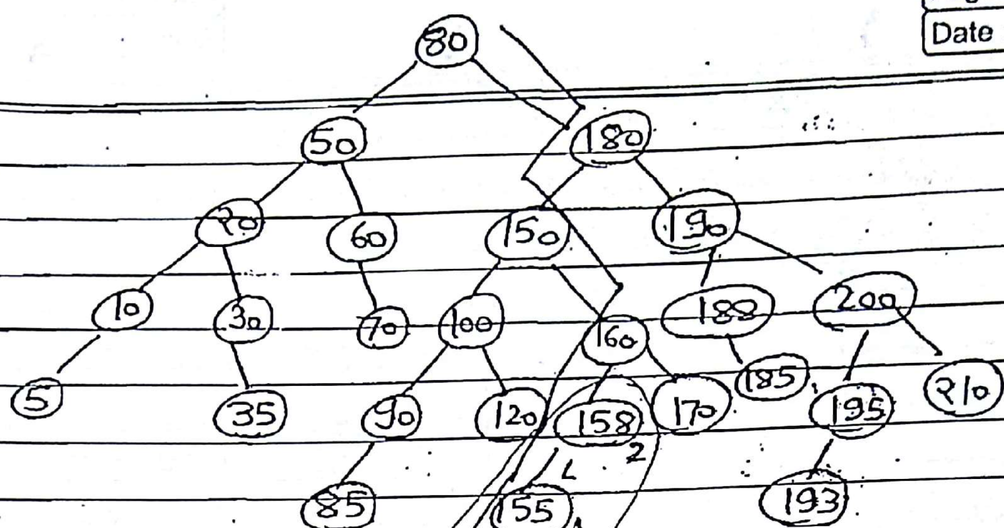
n elements to create AVL tree = O(n log n)

reating AVL tree for n elements = O(n log n)

e only diff b/w AVL tree & BST is the balanced structure n AVL

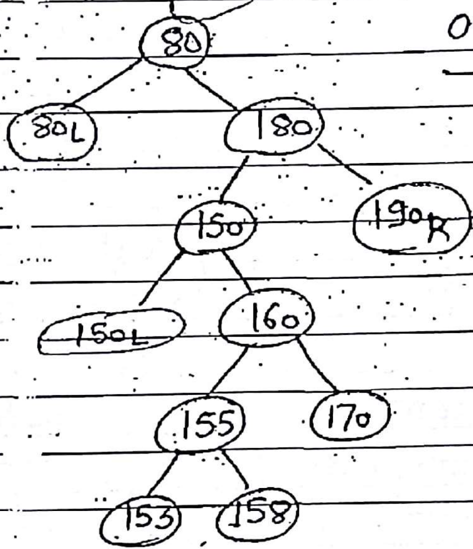
ies:– ~~Consider the following~~
Insert element 153

EXCELLENT



$\Longrightarrow$ Right-Rotate

O(1) + O(logn) + O(1) + O(logn) + O(1).

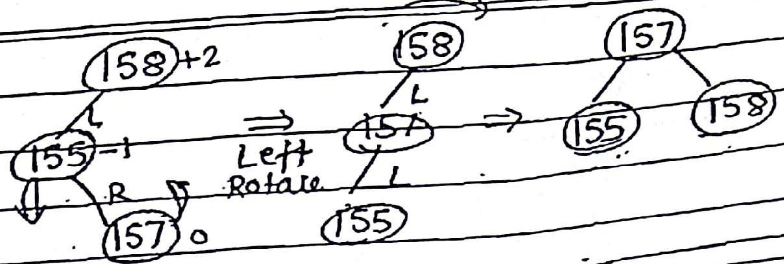


$\Longleftrightarrow$ AVL tree

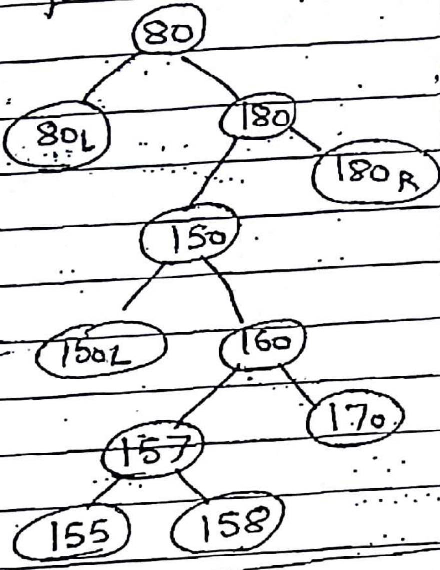
Insertion -

1. Create a node $\Longrightarrow$ O(1)
2. find its place $\Longrightarrow$ O(logn)
3. link $\Longrightarrow$ O(1)
4. check update tree. Is AVL or not, by again following the same path you traversed. $\Longrightarrow$ O(log n)(WC) O(1)(BC)
5. Rotate acc. to req. $\Downarrow$ O(1)

Insert 157



final tree



No of rotation = 2
No of problem = 1

- the person who are part of rotation will change first & then there children will change

⇓ just rectify one problem ie 1st problem in your path. No need to check after it.

T.C. comp – O(logn) in Best case
O(2logn) in worst case

Page No.
Date : / /

Dynamic Sets- the set change over time as when manipulated by algorithm.

Dictionary- a dynamic set that supports insertion, deletion, searching of elmt.

Each element in dictionary or a dynamic set is represented by an object whose attribute can be examined & manipulated if we have a pointer to object.

Satellite Data- any other attribute

operation- • Search
- Insert
- Delete
- Minimum
- Maximum
- Successor
- Predecessor

EXCELLENT

Parents of node performing rotation remain same

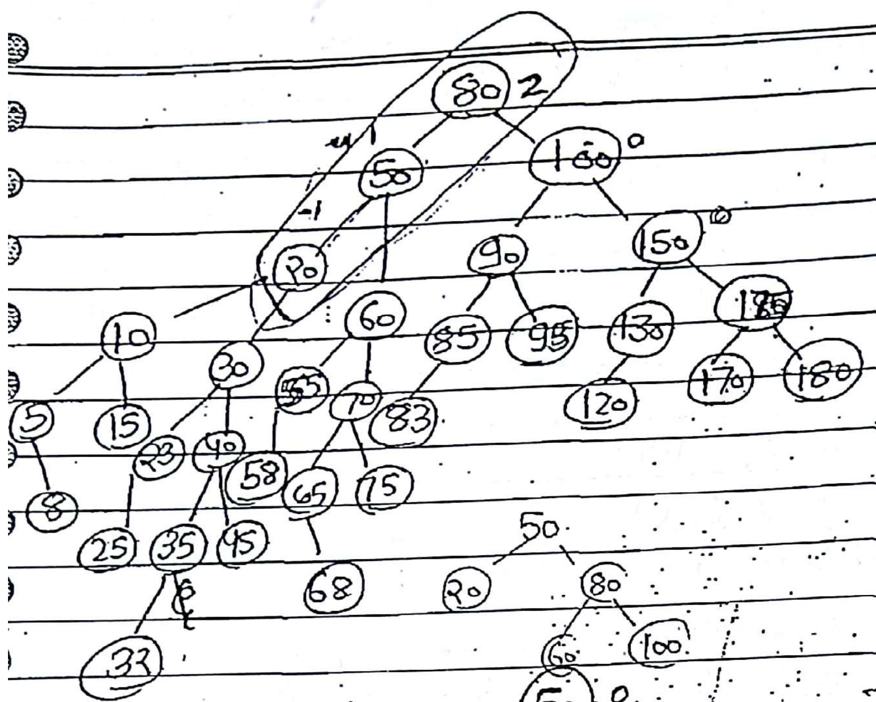
Page No.
Date: / /

Ques - Consider the following AVL tree-

Delete 160.

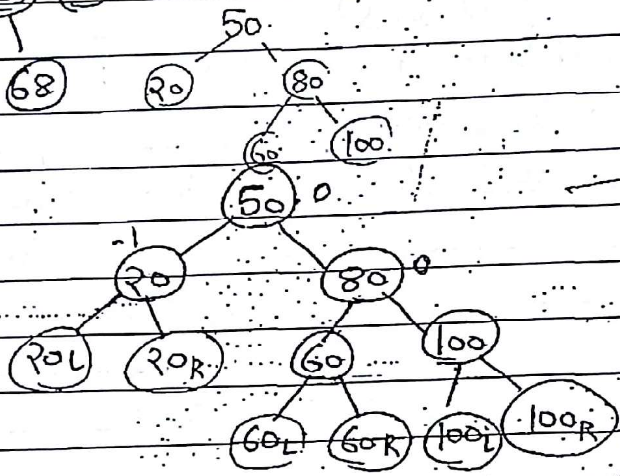
1. Find 160
2. Check for child
3. Delete
4. check for AVL from the parent of deleted node
5. check for problem type
6. Do Rotation.
7. Again check for AVL for the same path which you traversed while finding the node upto the root

Right O(1)



To find predecessor of a given node O(log m) to find that node + O(log n-m) to find predecessor

↓ Total time
O(log n) (WC)
BC (O(1))



+2 - left
-2 - Right

- Hence, it is a AVL tree

Time Complexity - ~~O(log n) + O(1) + O(log p) + O(1) + O(log m)~~) (when every node is in problem ~~p+m=~~ in entire path)

O(log n) + O(1) + O(log n) + O(log n)

↓ to find, ↑ Delete, ↓ check for AVL, ↑ Rotation

No of Rotation - 3 [LR (1) RR (2)]
↓ Left Rotation
↓ Right Rotation

(When you insert a node, height will increase or may not increase)
When you [delete] height decrease or may not decrease

At time of insertion, you have to check for AVL one time only but in delete you have to check for the whole tree

EXCELLENT

Page No.
Date : / /

find ↑ delete ↑ AVL ↑ Rotation ↑

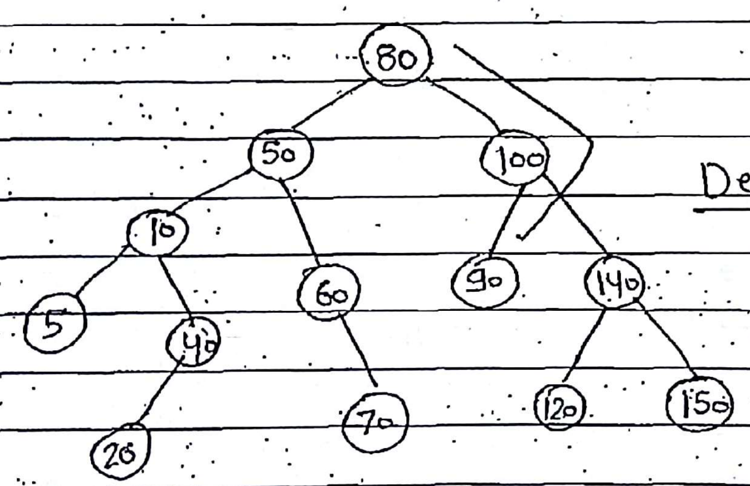
$TC = O(\log n) + O(1) + O(\log n) +$ (?)

$= O(2\log n)$ (WC, BC, AC) → (to find predecessor)
$= O(1) + O(1) + O(1) + O(1) + O(\log n) ?) O(\log n)$

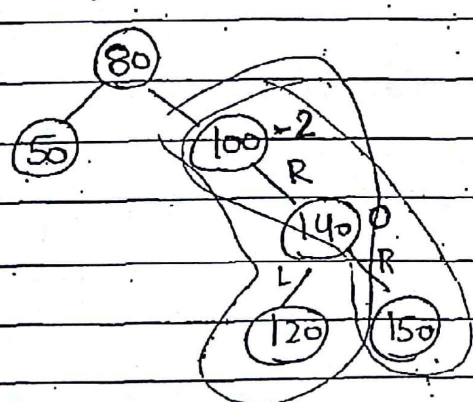
Note- In AVL tree, after insertion, in that path, max one problem can happen
Max Rotations are 2.
$O(2\log n)$ (WC)
$O(\log n)$ (BC)

In AVL tree, after deletion, in that path while backtracking max ($\log n$) can happen.
• Max. Rotation - ($2\log n$) Rotations (if each problem is LR)

Ques- Consider the following AVL tree-



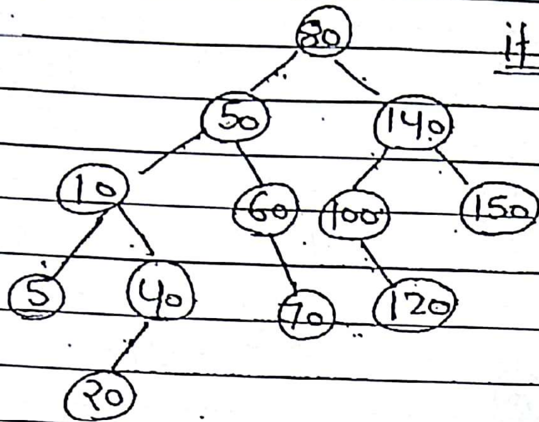
Delete 90



RL or RR
(You can choose anyone of them)
Acc. to your req.

AVL tree cannot be balanced at once
You cannot do it by using min heap.

Page No.
Date



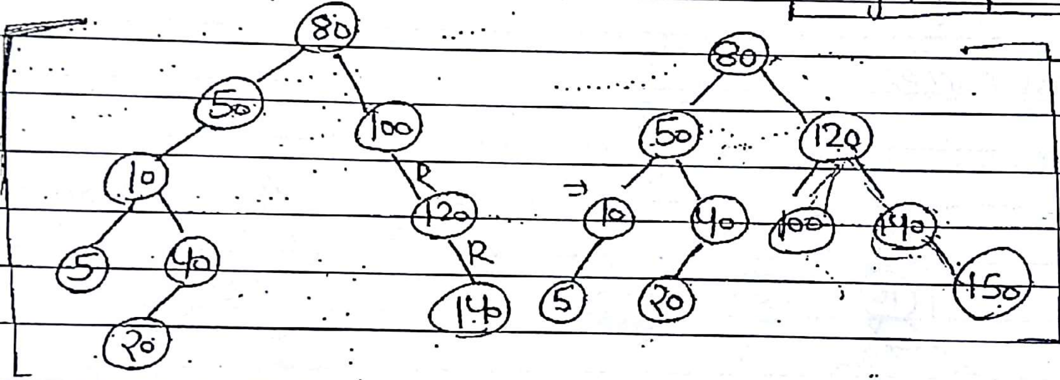
if I choose RR (because we want min. rotation

~~You cannot~~ choose ~~RL to be solved bcoz RL convert to RR & Here itself is RR. you cannot make it RR.~~

Min. no of rotations required = 1 (Left Rotation)

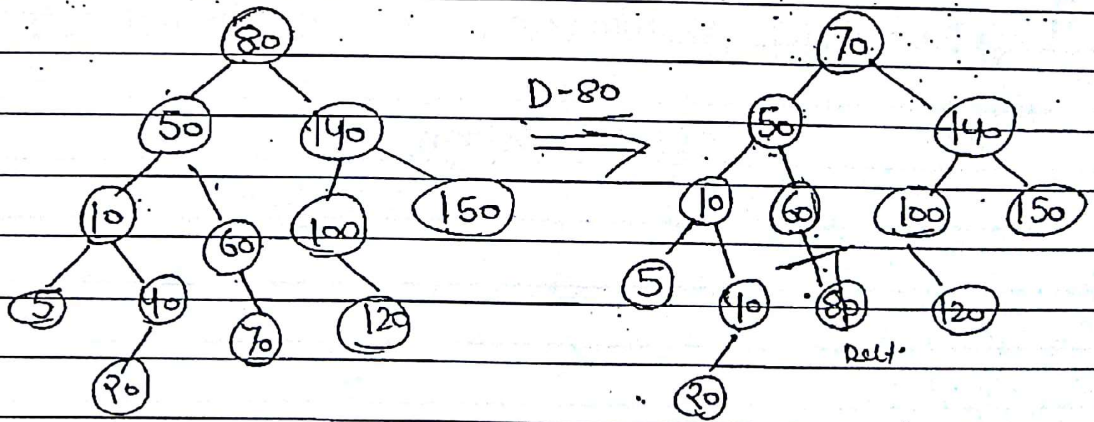
- O(1) → to find at BC
- O(1) → to delete
- ~~O(n) to find BF~~ O(logn) → to check AVL
- O(logn) to find predecessor

if I choose RL



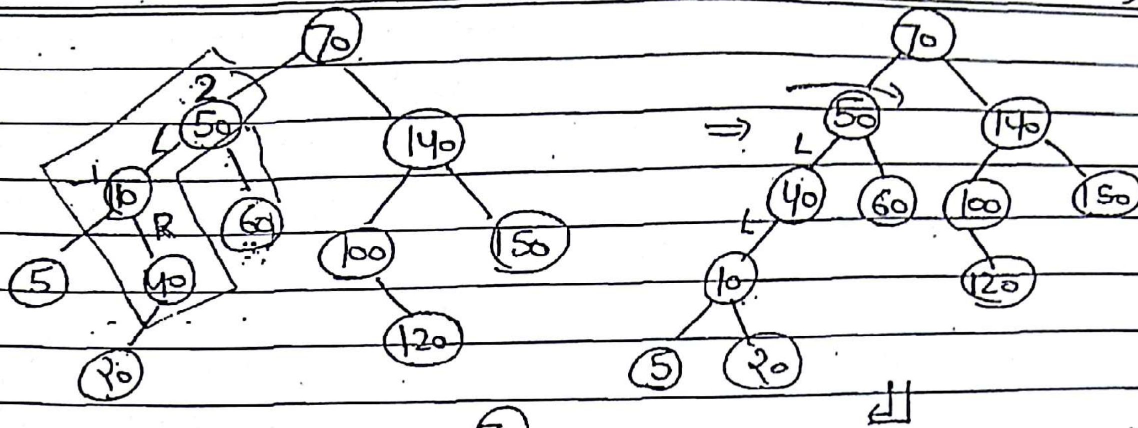
- Due to Rotation, height may be decreased ~~or may remain same but not rem~~ will never increase

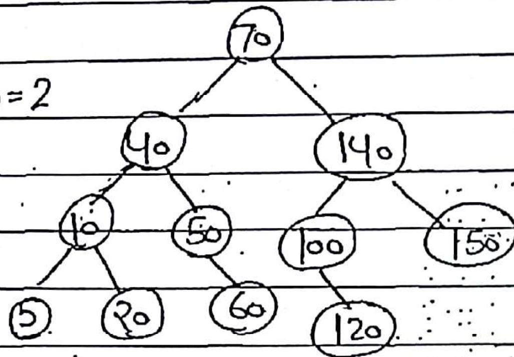
Ques -



EXCELLENT

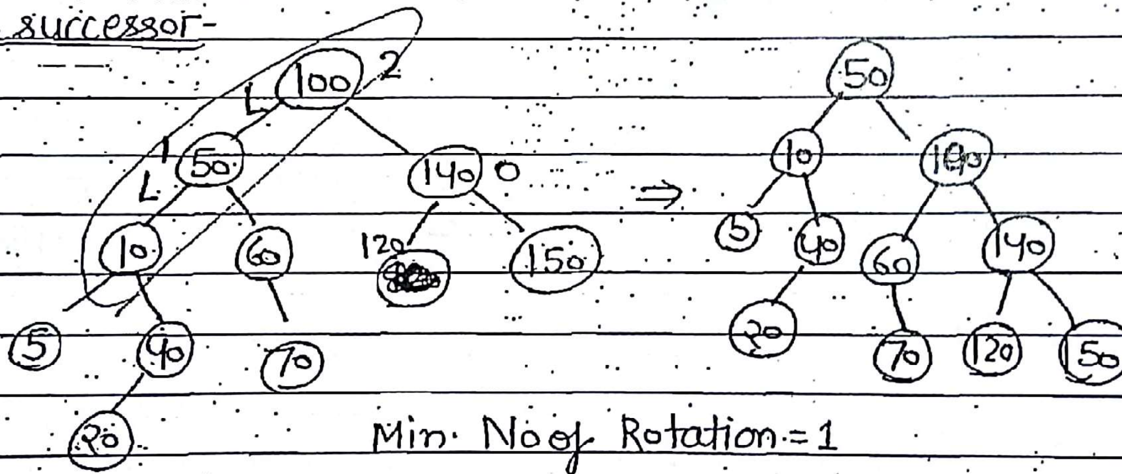
In Deletion, check for AVL = O(log n) (WC, BC, AC)





Min. No of Rotation = 2

Replace by successor-



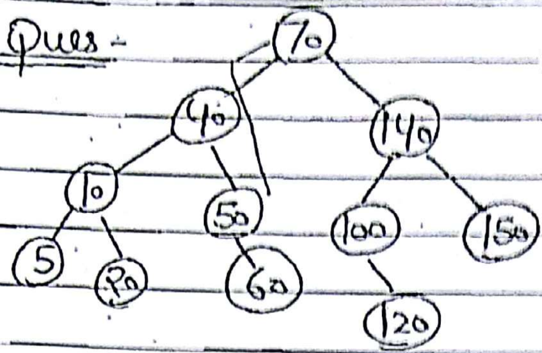
Min. No of Rotation = 1

You can Replace by predeccessor or successor, If not mentioned. If you wanna get minimum rotation, check for both

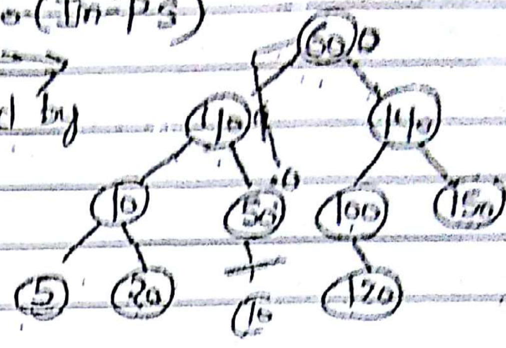
$O(\log n) + O(\log n)$

$= O(\log n)$

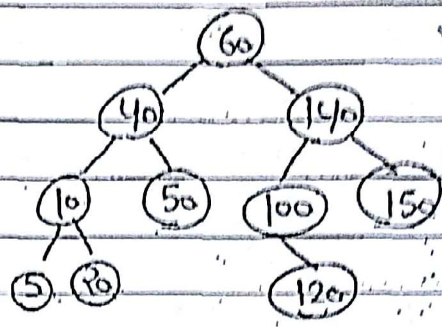
Ques -



Delete 70 (In-PS) replaced by pre



⇓ No problem



• When Deletion is done, problem may or may not occur.

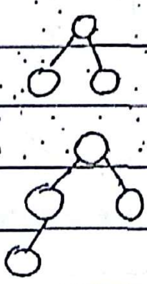
Ques - What is the max. height of an AVL tree with 20 nodes?

Max. height of BST with 20 nodes? - 19

No. of levels - 0 - o → 0

1 - o → 0

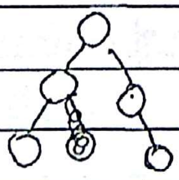
2 - → 1

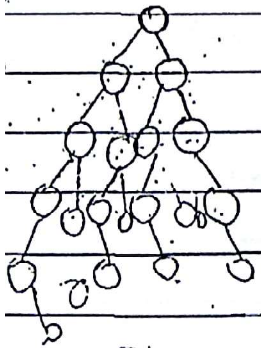


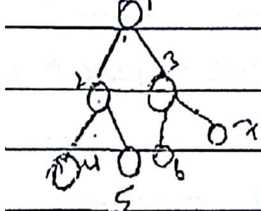
3. → 1

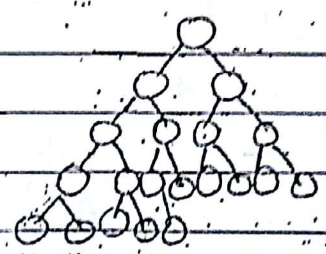
4. ⇒ 2

5. ⇒ 2







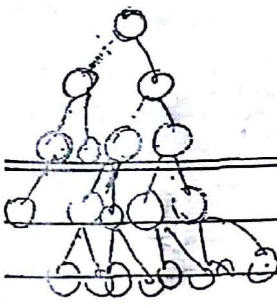


Max height → n-1

Min height = $\lfloor \log n \rfloor$

Min height = $\lceil \log n \rceil$

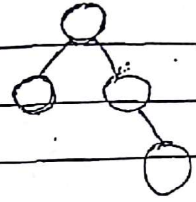
EXCELLENT



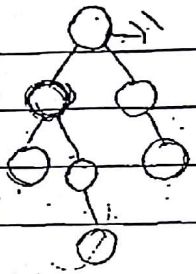
min. no of node

h = 0 (1 node)

h = 1 (2 node)

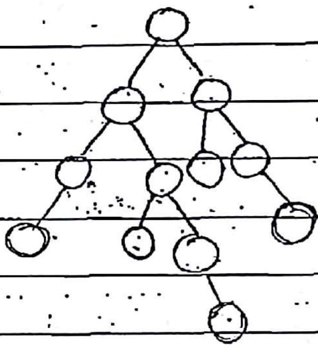


h = 2 (4 node) (2+2)

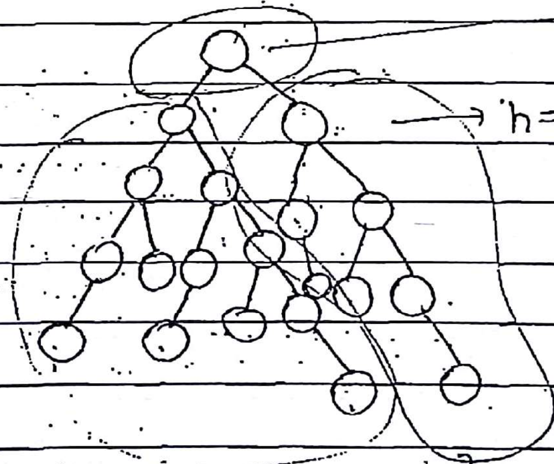


h = 3 (7 node) (4+4)

h(3) is obtained from attaching ~~two~~ h(2) tree h(1) tree



h = 4 (12 node)



h = 5 (20 node)

12+7+1 = 20

$h(n) = h(n-1) + h(n-2) + 1$

{ $h(6) = h(5) + h(4) + 1 = 33$ }

min no of nodes (H) = min no of node (H-1) + min no of node (H-2) + 1

↓
Height of tree

min no of node in Height h AVL tree,

$$MNN(H) = \begin{cases} 1 & ; \text{ if } H=0 \\ 2 & ; \text{ if } H=1 \\ MNN(H-1) + MNN(H-2) + 1 & ; \text{ if } H \geq 2 \end{cases}$$

~~$T(n) = T(n-1) + T(n-2) + 1$~~

∴ Height of AVL tree with n nodes = $\log n$

Max. height of AVL tree = $\lceil \log n \rceil$

$n=100$ $\log n = 6.$ ~~5~~ 7

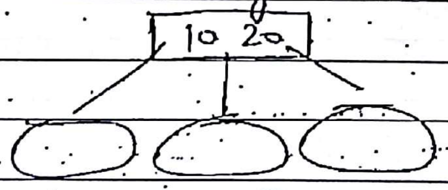
Min height of AVL tree, with n nodes = $\lfloor \log_2(n+1) - 1 \rfloor$

as $n = 2^k - 1$

no of level, $k = \log(n+1)$

height $= \lfloor \log_2(n+1) - 1 \rfloor$

B tree - if a particular node has more than one data field i.e it may have more than two children



multiway search tree

Height → $\log_3 n$ as it is three pointer field.

- B tree has lesser height than AVL tree.

if it is two data field, then it will store 3 pointers

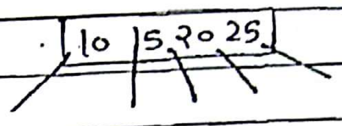
EXCELLENT

Date: / /

rder of a tree- no of children it can have.

rder of a Binary tree- no of children = 2

order = 5   No of children = 5
    data field = order - 1
    = 4.

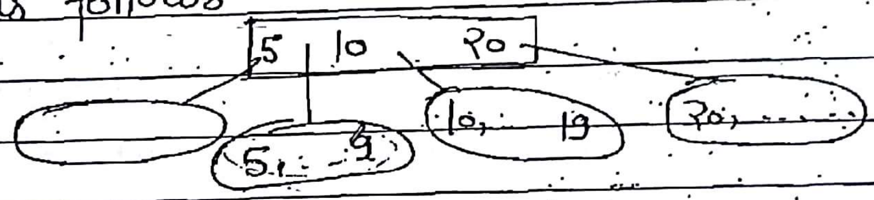


VL, B-Tree both are search tree as well as they both re balanced.

No of level in B-tree = $\lceil \log_{order} n \rceil$ (BC, WC, AC)

if order = 5 = $\log_5 n$ (No of level)

Tree- right pointers of a node will also point to No. itself as follows



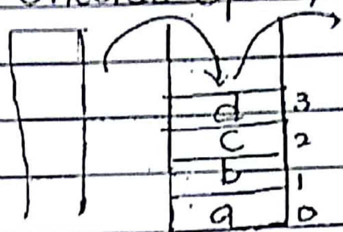
dvantage- All the Nodes will be at leaf level itself. you can perform linear search.

It is also a type of B-tree.

EXCELLENT

Stack

Definition- Oneside open, another side is closed



Top=3
↓ variable which contain posn of top most elmt in the stack.

To Insertio- Top = Top+1
To delete Top = Top-1

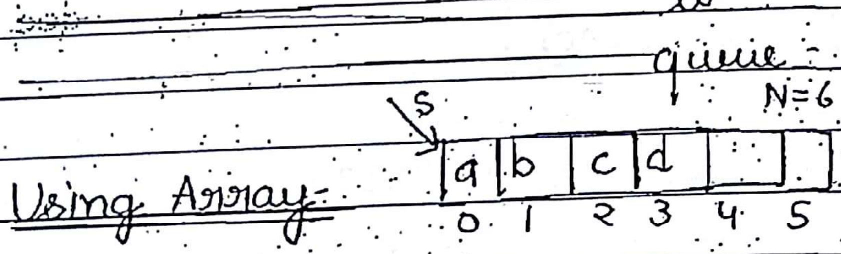
: LIFO, FILO (Last In First Out, First In Last Out)

ADT of Stack - push ()
- pop()

ADT of a particular Data Structure - the opn which are possible on a particular DS, but you donot bother for the implementation.

Implementing Stack- implementation is done using a particular data structure.

Implementation of stack using array -
ll -
queue -



Using Array-

initially, int top = -1

```
void push(char x)
{
    if(Top==N+1)
    { pf(" Stack overflow");
    } exit(1);
```


EXCELLENT

```
else
{                      (increment & then store)
    Top++;       |  s[++Top]=x   |   if s[Top++]=x  X
    s[Top]=x;
}
}

char POP( )              // returning the element which get deleted.
{  char y;
   if (Top==-1)
   {
     printf("Stack Underflow");
     exit(1);
   }
   else
   { y=s[Top];   (store & then decrement) | y=s[Top--]
     Top--;                                 y=s[--Top] X
   }
   return y;
}
```

Time - O(1) (Push, pop) (BC, WC, AC)

Implementing Multiple Stacks in Single Array-

- A single array will store multiple stacks for each stack in that array, there will be top variable

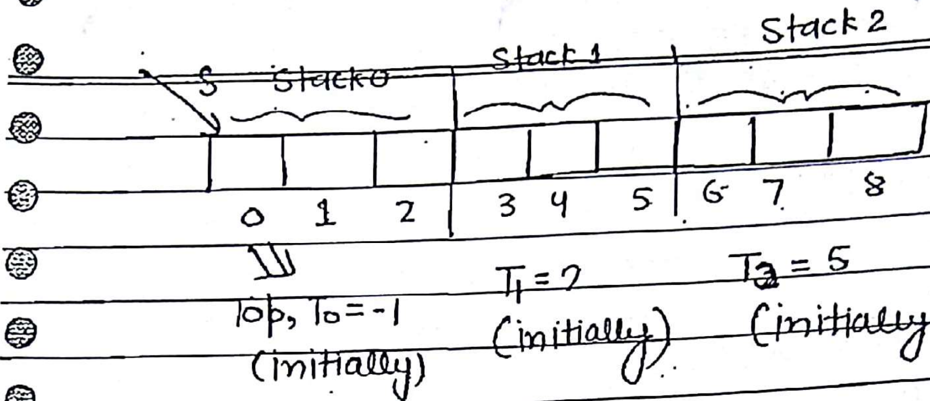
Let array size, $N=9$

No of stack $= m = 3$

$\therefore$ Stack size $= \frac{N}{m} = 3$

If $N=8$ stack size 2.6

$\therefore$ 3, 3, 2



To calculate initially top of ~~array~~ of any stack.

Initially for the whole array, it is -1.

∴ for 0th stack, Initial Top of Stack, $0 \times 3 - 1 = -1$

(0 → no of stack before that stack; 3 → no of elmt in that stack; −1 → Baseaddress (i.e the Initial top of stack for the whole array))

for 1th stack, Initial Top of Stack $= 1 \times 3 - 1 = 2$

for 2nd stack, Initial Top of Stack $= 2 \times 3 - 1 = 5$

⋮

for ith stack, Initial Top of Stack $= \left(i * \frac{N}{m} - 1\right)$

Void push(Ti, x) (insertion to stack i)

```
{
    if (Ti == ((i+1)*N/M - 1))
    { pf("Stack overflow");
        exit(1);
    }
    else{
        Ti++;
        s[Ti] = x;
```

char pop(T_i) (deletion from stack i)

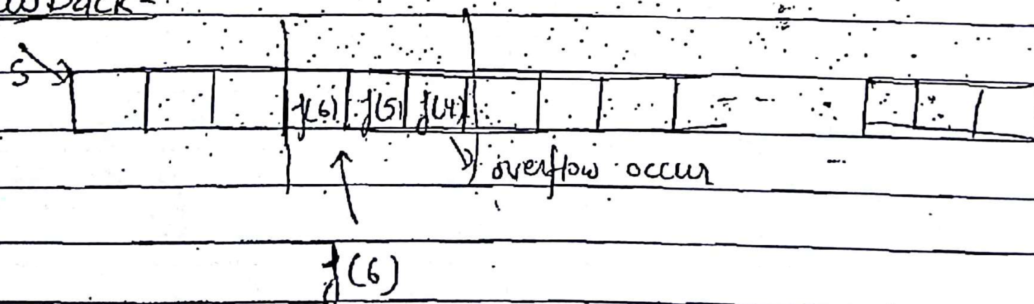
```
{
    if (Ti == i*N/M - 1)
    {
        pf("Stack Underflow");
        exit(1);
    }
    else
    {
        y = s[Ti]
        Ti--;
    }
    return y;
}
```

TC (push pop) = O(1)

Advantage of Multiple Stack in a single array-

- If there is a single stack only in a single array you can not run two recursive program at a time.
  By using MSSA concept, you can run two recursive program at a time.

Draw Back-

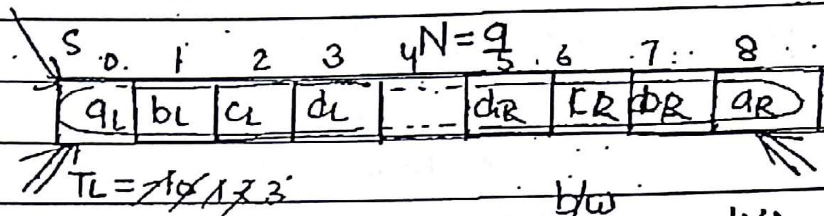


Using above program we can implement multiple stacks in a single array but it is not efficient as lot of space is empty still error msg came if one stack is full & other remaining

are empty we are getting error msg stack overflow even though lot of space available.

Implementing Multiple Stacks in single array efficiently:-

ADT- left push
left pop
Right push, Right pop



Here Memory is shared b/w two diff ds, which lead to the synchronization problem & that M/M Must be considered as critical section which must be handled by semaphore.

An array is full $T_L = T_R - 1$ or $T_R = T_L + 1$ (same for both stack)

Stack empty - if $T_L = -1$
if $T_R = N$

In this strategy, one stack will start from left & other will start from right. left will have top = -1 (initially) & Right will have top = N (initially)

For insertion, right will decrement Top
left will increment the Top.

if somewhere; $T_L = T_R$ (when overwritting of data occured)

if given, $T_L = T_R = \frac{N}{2}$, it may or may not possible.

as it might be possible that no of insertion in T_L is less than T_R.

if given $T_L == N$, it may or may not be possible
$T_R == 0$ May or may not possible

Page No.
Date: / /

If you want to implemented multiple stack efficiently, you can only implement two stacks at a time.

Stack life time of an element-

Ex 1:- n=3 (3 elmt pushed continously followed by 3 pops)

	(a, b, c)
c	
b	
a	

Let us assume a push or pop take 5 min. time.

Elapsed time- y=3 time wasted b/w two push ~~operto~~ operation

		3 ~~(time elapsed b/w push op^n of c & pop op^n of c)~~
~~pop(c)~~ 5	c	5 (push(c))
3		3
pop(b) 5	b	5 (push b)
3		3
pop(a) 5	a	5 (push a)

Life time of elmt in stack is time after push op^n of that element before pop op^n of that element.

Life tym of c = 3
Life tym of b = 19 (3, 5, 3, 5, 3)
Life tym of a = 35 (3, 5, 3, 5, 3, 5, 3, 5, 3)

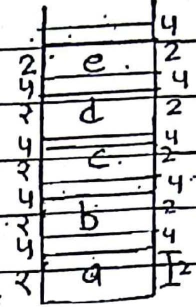
Life tym of an elmt is the smallest then that element is the last elmt of the stack.

$$\text{Avg. life time of elmt in stack} = \frac{3+19+35}{3} = 19$$

life tym of an elmt is basically ~~time~~ time for which elmt above that is present in stack.

EXCELLENT

e2- $n = 5 (a, b, c, d, e)$
$x = 2$ (push & pop time)
$y = 4$ (elapsed time)



L.T of $e = 4$
LT of d = time taken to push e & pop e & then elapsed time to pop d
$= \cancel{4,2,2,} \; 4, 2, 4, 2, 4 = 16$

LT of $c = (4,2)\,(4,2)\,(4,2)\,(4,2), 4 = \cancel{30}\ \cancel{28}\ 28$
d e e d

LT. of $b = (4,2)\,(4,2)(4,2)\,(4,2)(4,2)(4,2), 4 = \cancel{48}\ 40$
c d e e d c

LT of $a = (4,2)\,(4,2)\,(4,2)(4,2)\,(4,2)(4,2)(4,2)(4,2), 4 = 52$
b c d e e d c b

Total time $= 136 + 4$
Avg. time $= \cancel{26.12}\ \cancel{27.2} \Rightarrow 28$
$= \frac{140}{5} = 28$

$n(x+y) - x$
$30 - 2$

for $a = 4 * 2(x+y) + y$
(for 4 element) (for push & pop)

$b = 3 \times 2(x+y) + y$
$c = 2 \times 2(x+y) + y$
$d = 1 \times 2(x+y) + y$
$e = y + 0 \times 2(x+y)$

if n no

$(n-1) \times 2(x+y) + y$

Sum,
$= 2 * (x+y)\,[0 + 1 + 2 + \cdots n-1] + ny$
$= \cancel{2} * (x+y) \frac{n(n-1)}{\cancel{2}} + ny$

$= n(x+y)(n-1) + ny$
$= n[(x+y)(n-1) + y]$

Avg $= (x+y)(n-1) + y$
$= n(x+y) - x$

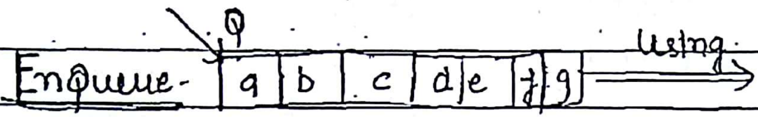
88

EXCELLENT

Page No.
Date : / /

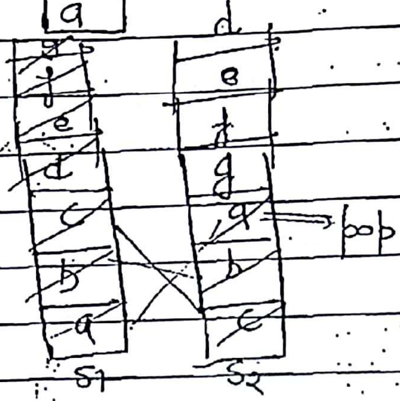
Implementing queue using stack-

Enqueue-



1 Enqueue = 1 push op^n in S_1
2 Enqueue = 2 push op^n in S_1

Dequeue- 1 Dequeue - 3 pop op^n (S_1)
+3 push op^n (S_2)
+1 pop op^n (S_1)



pop(S)
push (S_2)
pop (S_1)
push (S_2)
pop (S_1)
push (S_2)
pop (S_2)

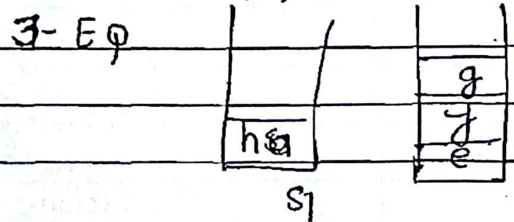
Time taken by Enqueue - O(1)

Taken taken by Dequeue (first d-Q) = O(n)
(2n+1)

but II element when Deleted = O(1) as S_2 has elmt.

3- EQ = 3 push S_1
1- DQ = 3 pop S_1 +
3 push S_2 +
1 pop S_2
1- DQ = 1 pop S_2
1- DQ = 1 - pop S_2

then again
4- EN = 4 push
1- DQ = 4 popS_1 + 4 pushS_2 + 1 pop S_2.
3- EQ

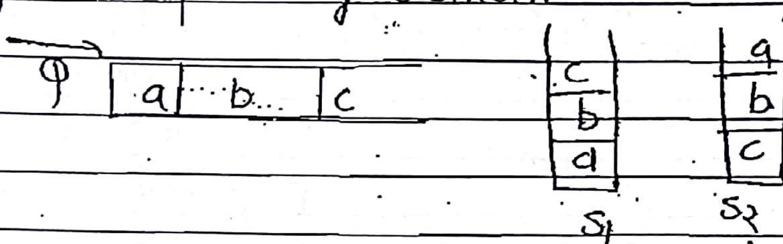


1- DQ = 1 pop - S_2
= (e)

EXCELLENT

Observation-

if you want to Enqueue the elmt. directly visit S_1 & insert the elmt.
if you want to dequeue, directly visit S_2 if S_2 is empty goto S_1 perform pop on S_1 & push it into S_2. S_2 will have original sequence of element



it will take 3 pop S_1 + 3 push S_2 + 1 pop S_2.

but if S_2 is not empty directly delete from S_2 lead to O(1)

time-

1- ENQUEUE = 1 push
1- DEQUEUE
- 1 pop (if S_2 is not empty)
- n push() + n pop + 1 pop (if S_2 is empty)
  (S_2) (S_1)

• Insertion is done in S_1.

EXCELLENT

Page No.
Date: / /

Ex 2

5 EQ
3 DQ
8 ENQ
7 DQ
15 ENQ
14 DQ

how many push & pop using queue by 2 stack.

a	b	c	d	e

S_2: a, b, c, d, e (struck through); f, j, k, l, m
S_1: a, b, c, d (struck through); f, g, h, i, j, k

5-EQ = 5 push-S_1
3-DQ = 5 pop-S_1
+5 push-S_2
+1-pop-S_2
1-DQ = 1-pop-S_2
1-DQ = 1-pop-S_2
8 ENQ = 8 push-S_1
1-DQ = 1 pop-S_2
1-DQ = 1-pop-S_2
1-DQ = 8 pop-S_1
+8 push-S_2
+1 pop-S_2
4-DQ = 4 pop-S_2
15-EQ = 15 push
1-DQ = 1-pop-S_2
1-DQ = 1-pop-S_2
1-DQ = 1-pop-S_2
1-DQ = 15-pop-S_1
+15 push-S_2
+1-pop S_2
10-DQ = 10 pop-S_2

Total push - 56
pop - 52

Program ENQUEUE-

```
void ENQUEUE(S1, x)
{
    push(S1, x);        TC = O(1)
}

char DEQUEUE(S1, S2, x)
{
    if(S2 = empty)
    {
        while(S1 is not empty)
        {
            x = pop(S1);
            push(S2, x);
        }
    }
    pop(S2);
```

EXCELLENT

Page No.
Date: / /

{To Implement Queue, No of Stack ...

```
else
{
    y = pop(S2);
}
return y;
}
```

TC = O(1) = {Avg. Case, Best case}
Worst Case = O(n)

HomeWork- Implement Stack using Queue. {push = O(1), pop (O(n))}

3 push = 3 EQ - Q1
1 - Pop = 2 DQ - Q1
2 - EQ = Q2
1 - DQ (Q1)
3 push = 3 EQ - Q2
to check it push the elmt where queue is not ~~elem~~ empty -
pop = O(n) (BC, AC, WC)

Applications of Stack-

1. Recursion
   - (i) Tail Recursion
   - (ii) Non Tail Recursion
   - (iii) Indirect Recursion
   - (iv) Nested Recursion
2. Infix to postfix
3. prefix to postfix
4. postfix evaluation.
5. Tower of Hanoi
6. Fibonacci Series

1. Recursion- WAP in C using Recursion to print the array of n elements.

```
print ( int a[], int n)
{
    static int i=0;
    if (i<n)
        printf(a[i])
    i++;
    print(a[], n);
}
else
    exit(0) or return.
```

EXCELLENT

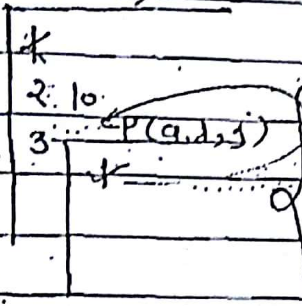
Page No.
Date : / /

```
print (a, i, j)
{
  if (i == j)
  { printf(a[i])
    return;
  }
  else
  { printf(a[i])
    print(a, i+1, j);
  }
}
```

Tail Recursion- When a f^n call poped out i.e. a f^n come back after its completion and there is no work to do, known as Tail Recursion.
i.e when last stmt of a f^n is itself a Recursion. at this stage is not required actually.

~~10 20 30 40 50~~

PA(a, i, j)



Here no work to do but still it wait in stack known as tail recursion.
Here, in actual, no stack is required but it uses so known as tail recursion.

Non-Tail Recursion- When f^n call is ~~the~~ Not the last stmt of f^n i.e some ~~function in call stmt~~ in calling f^n depends upon called f^n,

Recursion = $T(n) = T(n-1) + c$
$= O(n)$

If we have
fn
{ PA( ) → Non-tail Recursion
PA( ) → Tail Recursion — combining Non-tail Recursion
}

10-Apr

Ex for Tail Recursion- above program.

(2) The disadvantage with tail recursion is lot of stack space is wasting

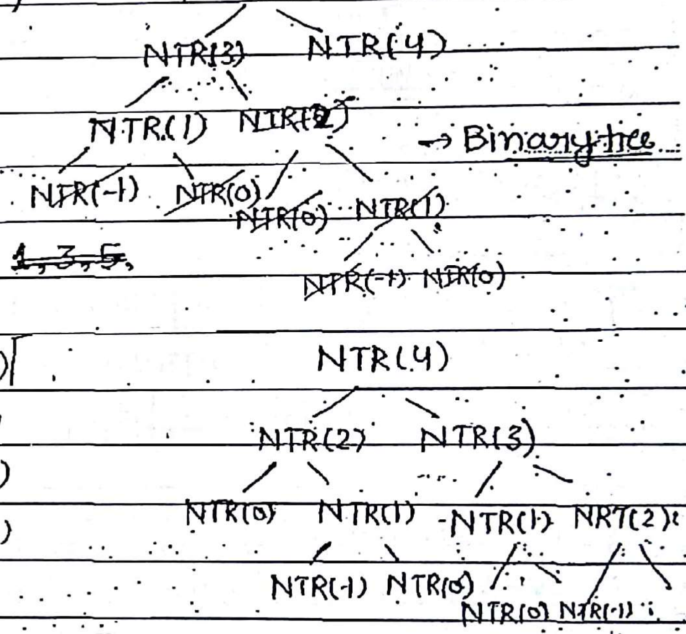
(3) The advantage with the tail recursive program is we can write equivalent non recursive program easily with the help of loops.

Non-Tail Recursion-

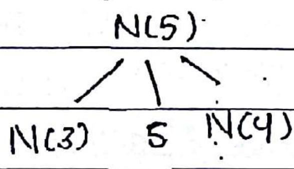
```
NTR(n)
{
  if(n<=0) return;
  else
  {
    NTR(n-2);
    pf(n);
    NTR(n-1);
  }
}
```

O/P - ~~1,3,2,1,2,1,1,3,2~~
1,3,2,1,5,2,1,4,1,3,2,1.

I/P=5



fn calling sequence - preorder.
[O/P of sequence - postorder (as parent is getting completed only when child are completed)]



→ Binary tree-
as 5 is only a printf taking constant tym.

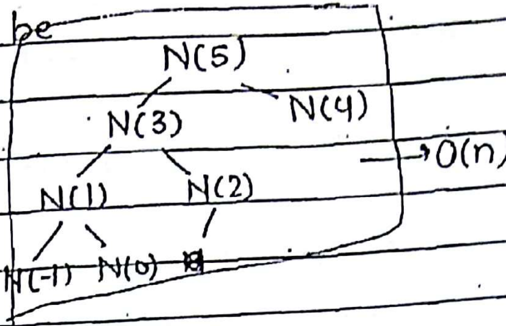
$TC = O(2^n)$

$TC = T(n-1) + T(n-2) + O(1)$

stack space = O(n)

- By using DP, TC = O(n+1) ≅ O(n) (as n+1 distinct DP calls)

New tree will be



→ O(n)

stack size = O(n)

stack table space = O(n+1)

If ask simple, TC = $O(2^n)$

no repitition of f^n call = O(n) (using DP)

NR( )
{
push(N(5))
push(N(3))
,
}

	TC	SC
with DP	n	n+n = 2n
without DP	2^n	n

- Pascal do not support recursion as it does not have stack.

- for a recursive program, if you want to write equivalent Non-Recursive program the stack is required as will be doing it by push & pop.

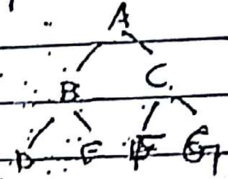
- for a recursive program, if a stack is not required for non-recursive program then that must be a tail recursion.

1. IN the Given recursive program, After the function call, there is something to do then it is called Non-tail recursion.
2. It is very difficult to write equivalent Non recursive program for given non-tail recursive program.
3. We are not wasting stack space unneccessarily in Non-Tail Recursive program.
4. For the given non tail recursive program, if you write non recursive program then you have to take stack

5. For the given tail recursive program, if you write Non-Recursive program, then stack not required.

if you wanna writes its equivalent preorder.

```
Preorder(r)
{   r
    po(r->lc)
    po(r->rc)
}

(r) {  pf(r)
       push(A);   (because after going to left, when you came back you required A->right)
       r = r->left
       pf(r)
       push(B)
       r = r->left
       push(D)
       r = r->left
       push(E)
       pop(D)
       push(E)
       pop(E)
       pop(B)
       r = A->right
       pf(r)
    }
```



[Whenever I write NR of NTR then you have to take care everything taken care by system & stack is always required.]

for Quick Sort, when worst partitioning occur

() ( )
1   n-1

QS(R)

QS(R)

there is no need to come back to sort a single element so no need to come back cas it is a tail recursion

∴ it is said for better program it require logn stack space

You can writes its equivalent non recursive program. no stack is required.

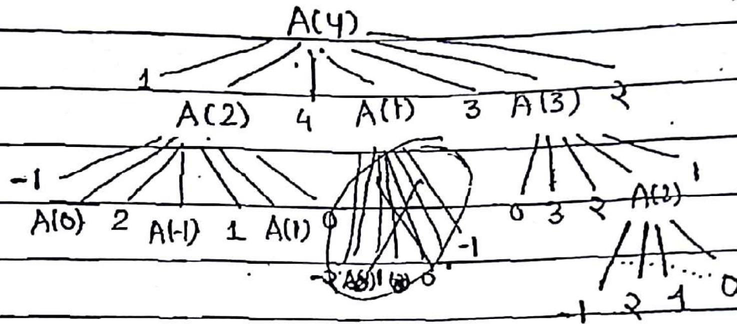
EXCELLENT

Page No.
Date : / /

(n)

i/P = 4
O/P : ~~1, 4, 3~~

```
if (n≤1) return;
else
{
    pf (n-3);
    A(n-2);
    pf (n);
    A (n-3);
    pf (n-1);
    A (n-1);
    pf (n-2);
}
```



1 -1, 2, 1, 0, 4, ~~-2, 0, 0, -1~~, 3, 0, 3, 2, -1, 2, 1, 0, 1 2

Without using DP, TC = 3^n space = n   = O(n)
with DP = O(n)   space = n+n = O(n)

Indirect Recursion Ex: A() B()

```
A()          B()
{            {
  ==           ==
  B()          A()
  ==           ==
}            }
```

A is calling B but B indirectly called A - Indirect Recursion.

Page No.
Date: / /

Ex 2

```
A(n)
{
    if(n≤1) return;
    else
    {
        B(n-2);
        pf(n);
        B(n-1);
    }
}

B(n)
{
    if(n≤1) return;
    else
    {
        pf(n-1)
        A(n-1);
        A(n-2);
        pf(n);
    }
}
```

I/P : A(4)

A(4)
- B(2)
  - 1
  - A(1)
  - A(0)
  - 2
- 4
- B(3)
  - 2
  - A(2)
    - B(0)
    - 2
    - B(1)
  - A(1)
  - 3

Here Execution tree will have diff f^n at different levels.

O/P: 1 2 4 2 2 3

No of Levels - n

∴ Complexity = $O(2^n)$ } without DP

Stack space = O(n)

with DP = n/2 for A & n/2 for B

as if for A if 4 is there then for B 4 is not there

= O(n)

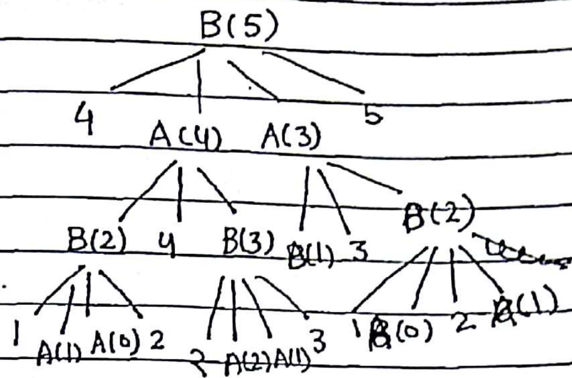
↳ n distinct f^n calls

Space = O(n+n) = 2n ≃ O(n)

[If in Indirect Recursion, if A() has 2 f^n call & B has 3 f^n call then $TC = 3^n$ as upper bound will be taken to ~~ternary~~ (three)]

Page No.
Date: / /

I/P B(5).

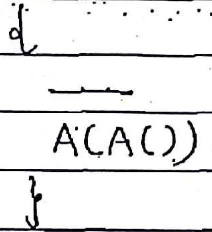


O/P - 4, 1, 2, 4, 2, 2, 3, 3, 1, 2, 5

Nested Recursion-

$$A(m,n) = \begin{cases} n+1 & ; \text{ if } m=0 \\ A(m-1, 1) & ; \text{ if } n=0 \\ A(m-1, A(m, n-1)) & ; \text{ if otherwise} \end{cases}$$

When f^n calling itself with calling itself in parameters as well as

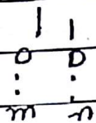


```
A()
{
  ___
  A(A())
}
```

A(1, 5) ⟶ A(0, A(1,4)) = 1 + A(1,4)
= 1 + A(0, A(1,3))
= 1 + 1 + A(1,3)
= 1 + 1 + 1 + A(1,2)
= 1 + 1 + 1 + 1 + A(1,1)
= 1 + 1 + 1 + 1 + A(0, A(1,0))
= 1 + 1 + 1 + 1 + A(0, A(0,1))
= 1 + 1 + 1 + 1 + A(0, 2)
= 1 + 1 + 1 + 1 + 3
= (7)

O/P = 7

With DP, $O(mn) \Rightarrow mn$ distinct f^n call.



check once

$A(2,5) \Rightarrow A(1, A(2,4)) \Rightarrow$ ~~A(1,~~ $= 13$

⇓

$A(1, A(2,3)) = 13$

⇓

$A(1, A(2,2)) = A(1,9) = 11$

⇓

$A(1, A(2,1)) \Rightarrow A(1,5) = A(0, A(1,5)) = A(0, A(1,4)) = 6$

⇓

$A(1, A(2,0)) = A(1,3) \Rightarrow A(0, A(1,2)) = 5$

⇓

$A(1,1)$ ⇓ $A(0, A(1,1)) = 4$

⇓

$A(0, A(1,0)) = A(0,2) = 3$

⇓

$A(0,1) = 2$

$A(0, A(1,3)) = 7$

as
- 1,0 = 2
- 1,1 = 3
- 1,3 = 5
- 1,50 = 52

(only two extra)

AckerMan Relation

(try to find out any relation b/w given expression)

Ex- $A(3,3) = A(2, A(3,2)) \Rightarrow A(2,33) = 69$

⇓

$A(2, A(3,1))$ $A(2,15) = 33$

⇓

$A(2, A(3,0))$ $A(2,6) = 15$

⇓

$A(2,1) = 5$

O/P = 69

- 2,1 = 5
- 2,2 = 7
- 2,3 = 9
- 2,4 = 11 (2×4+3)
- 2,5 = 13 (2×5+3)
- 2,6 = 15 (2×6+3)

EXCELLENT

[A (1, 0 to n) = 1*n+2
A (2 ... 0 to n) = 2xn+3
in AckerMan]

Page No.
Date : / /

Infix to Postfix- Given a+b (Inorder)
ab+ (postfix)
+ab (prefix)
(order of operand cannot changed)

Ex1 - Infix- a+b*c
Postfix- a+bc* = abc*+
Prefix- a+*bc = +a*bc

Note :- In all above 3 format, order of operands can not be changed.

Ex2 Infix- b-a+c
Postfix- ba-c+
Prefix - +-bac

[operators with same priority getting solved acc. to their ~~precedence~~ associativity]

a+b+c+d-e-f-g-h
ab+c+d+e-f-gh-

+, - > left associativity
*, /

Ex3 - Infix: d-a*b/e↑f+c↑g

↑ - highest priority with right associativity.
a↑b↑c
abc↑↑

Postfix - d-a*b/e↑f+(c↑g)

d-a*b/e↑f+cg↑
d-a*b/ef↑+cg↑
d-ab*/ef↑+cg↑
d-ab*ef↑/+cg↑
dab*ef↑/-cg↑+

Stack size- (4)
d-a*b/c↑

Prefix d-a*b/e ↑ j + ↑cg
d-a*b/↑ej + ↑cg
d-*ab/↑ej + ↑cg
~~d-/a~~
d-/*ab↑ej + ↑cg

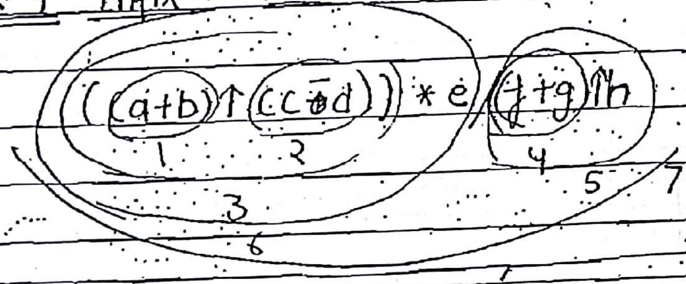
-d/*ab↑ej + ↑cg
+-d/*ab↑ej↑cg

( ) - Left Associative.

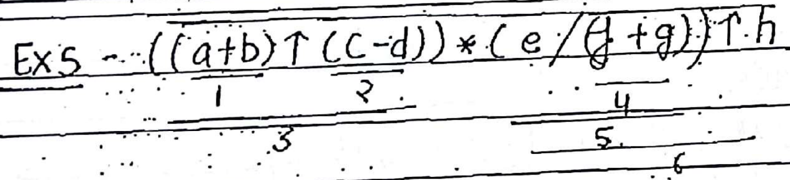
if a question donot mention about priority of operator & you are not aware, do left ~~to right~~

Count outer brackets only.

Ex-4 Infix



~~ab+~~



Postfix ab+cd-↑ ejg+/h↑*

Prefix
~~*↑+ab-cd ↑/+ej g e h~~
*↑+ab-cd ↑/e+jg h

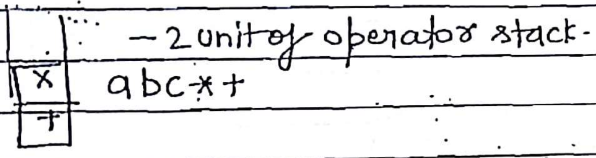
operators		priority	Associativity
+	-	(1) (Lowest)	L-R
*	/	(2) (	L-R
↑		(3)	R-L
( )		(4)	L-R

EXCELLENT

Page No.
Date : / /

Algorithm-
- if any operand
  - print operand
- if any operator
  - if has highest priority than one in stack
    - push into stack.
  - ~~else else~~
    - ~~pop the print the operator~~
  - if has ~~lesser~~ equal priority to the top of stack

a+b*c

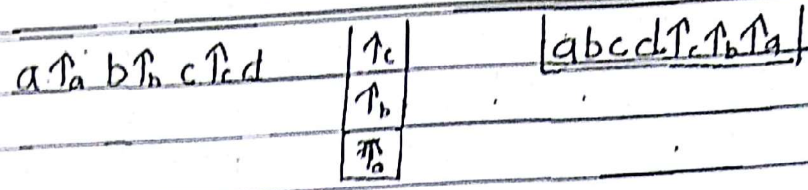


- 2 unit of operator stack.

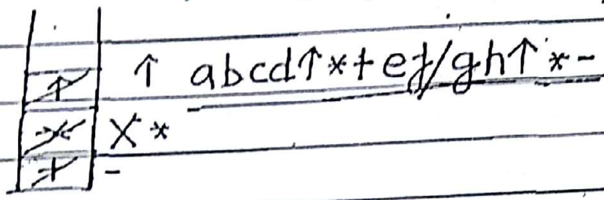
abc*+

Ex 2 - b-a+c   b   O/P ba-c+

∴ You can not put two operator with same priority into stack.
You have to pop previous one acc. to associativity
if both have same priority, pop previous one ~~as it has not~~ in (LR) associativity.
- ~~When two operator wi~~



$a+b*c\uparrow d-e/f*g\uparrow h$



In the algorithm, the pushed values are operators only so it is an operator stack.

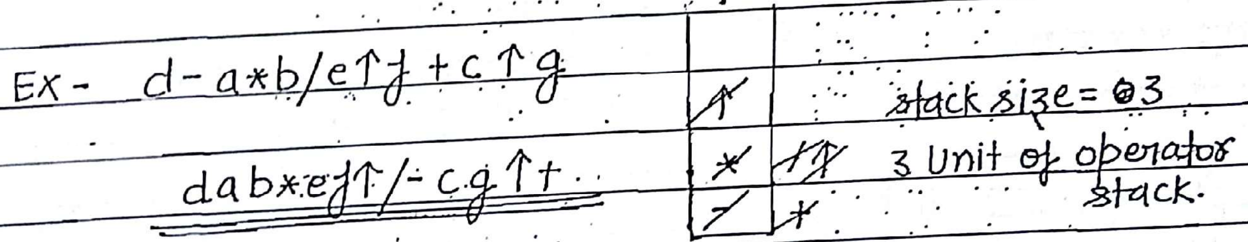
Time Complexity = $n \times O(1)$ (as push, print, pop taking O(1) time)

$= O(n)$

(n - size of expression)

Ex - $d-a*b/e\uparrow f+c\uparrow g$

dab*ef↑/-cg↑+



Note - Infix to postfix conversion uses operator stack.
To convert n-length infix exp. into equivalent postfix, O(n) time required (for every symbol constant time (as pop, push, print is done only))

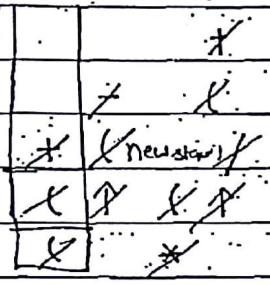
Page No.
Date : / /

When brackets are present in expression, they get pushed into stack. In stack, open bracket indicates the start of stack & closing bracket represent no one is there in stack. Whenever closing bracket came, pop until ~~closing~~ open bracket came.

((a+b)↑(c-d))*(e/(f+g))↑h

(	new stack started
(	new stack started
+	
)	stack ends pop until another open bracket

ab+ cd-↑ e fg+/ h↑*



brackets are also considered as operators

Prefix to Postfix :- to do this, you need 3 thing
operator & two operand } when you find them just circle them

1. +ab ⇒ ab+

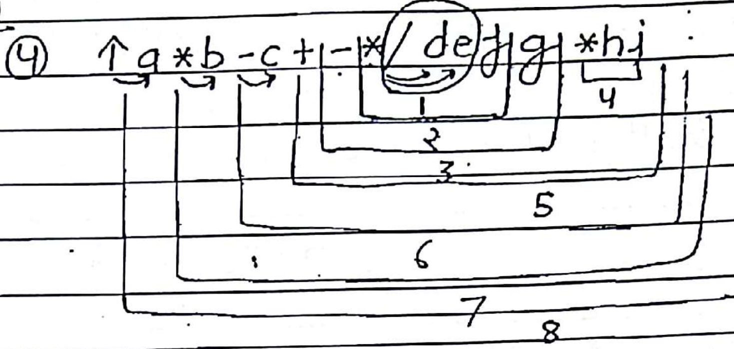
2. * +cd -ba ⇒ cd+ ba- *

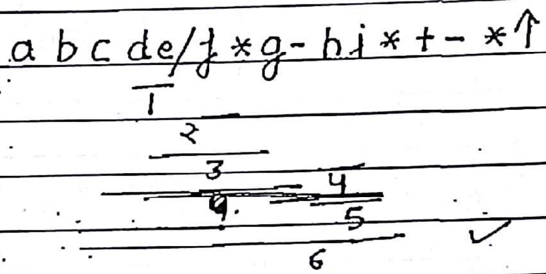
3. * ↑ / +ca B * db ↑ f g

Page No.
Date : / /

operators order cannot change.

Question

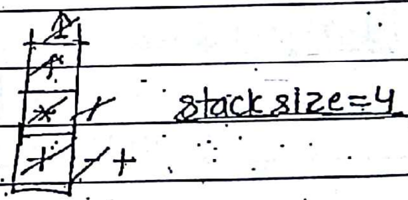




In this case, operand & operator both can be there in stack.
- Processor convert the exp. to postfix expression.

Postfix Evaluation- • Postfix expression if used to evaluate expression it take only one scan to execute the O/P as the operators are in order acc. to their precedence.
• fn pl are written in prefix.

ex - 2+3*4↑1↑5/2-10+3



Postfix

2 3 4 1 5 ↑ ↑ * 2 / + 10 - 3 +

O(n) time

When postfix Evaluation is done we use operand stack.

Operand- push into stack

an operator- pop two times
- 1st pop - operand2
- 2nd pop - operand1
- take r = op1 op op2
- push r

EXCELLENT

Page No.
Date: / /

~~top~~ for operand - push - O(1)
for operator - pop + pop + cal. rest + push = O(1)

∴ time complexity = O(n)

~~8~~	
~~1~~	~~1~~
~~4~~	~~4~~
~~3~~	~~12~~ ~~2~~ ~~16~~ ~~3~~ 1
~~2~~	~~6~~ ~~8~~ ~~-2~~

O/P = 1

for infix - scan for each operator to whole expression to evaluate

for postfix - O(n) only
↑ More suitable

Ex2 Infix: 5↑1 * 2↑1 * 3 - 7↑1 * 2

Postfix - 5 1 ↑ 2 1 ↑ * 3 * 7 1 ↑ 2 * -

~~↑~~ ~~*~~ ~~↑~~ (2-unit operator stack)

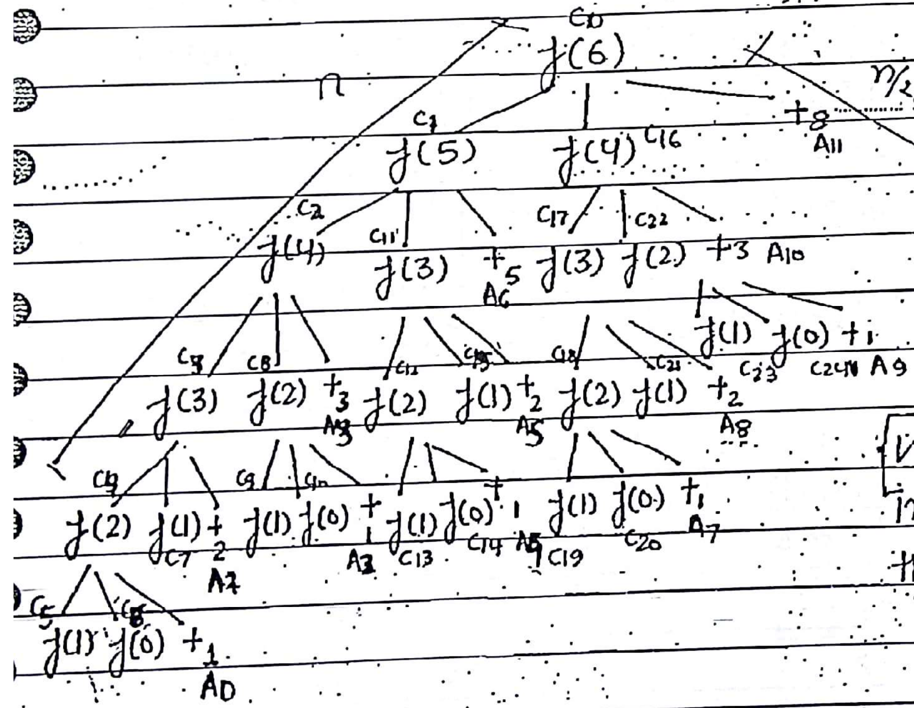
evaluation

Answer = 16

```
fibonacci(n)
{
    if(n==0) return 0;
    if (n==1) return 1;
    else (return (fib(n-1)+fib(n-2));
}
```

Time Complexity, $T(n) = \begin{cases} O(1) & ; \text{if } n=0 \text{ or } 1 \\ T(n-1)+T(n-2)+O(1) & ; \text{if } n \geq 2 \end{cases}$

No of add; $A(n) = \begin{cases} 0 & \text{if } n \leq 1 \\ A(n-1)+A(n-2)+1 & ; \text{if } n > 1 \end{cases}$



[When a particular f^n is in stack, the parent of that f^n is in stack.]

* In Every Programming language, f^n calling seq. preorder & f^n execution order postorder.

Rectify Diwy

$2^n - 1 - 2^{n/2}$

f(2) 3

Page No. 5

Date: / /

Ques1 - In fibonacci of n, is there is any f^n call after last add^n?

A - No f^n call (After All total problem computed)

Ques2 - In fibonacci of n, is there any add^n after last f^n call?

A - Yes, 3 Addition will be done [($n/2$) no of add^n]

as it will be tracing the right most path of tree whose length is $\lceil n/2 \rceil$

Ques3 - In fibonacci of n, After how f^n call first add^n taken place?

A - it will occur when whole left most path get traced.

$= n$ f^n call $+1$ (as two f^n call are there)

$= (n+1)$ (after C_7, there is A_0.)

Ques4 - In Fibonacci of 6, After how many f^n calls 9th addition takes place?

A - 22 f^n calls (After C_{22} there is A_8).

Ques.5 In Fibonacci of n, how many f^n calls are there?

$$f(n) = f(n-1) + f(n-2) + 1$$

(f^n call, f^n call, f^n call)

$f(0) = 1$

$f(1) = 1$

$f(2) = 3$

$f(3) = f(2) + f(1) + 1$

$= 5$

$f(4) = f(3) + f(2) + 1$

$= 9$

$$f(n) = \begin{cases} 1 & n \leq 1 \\ f(n-1) + f(n-2) + 1; & \text{if } n \geq 2 \end{cases}$$

for no of f^n call, $f(n) = f(n-1) + f(n-2) + 1$

EXCELLENT

Q6)- In fibonacci of n, how many add^n are there?

~~f(0)~~

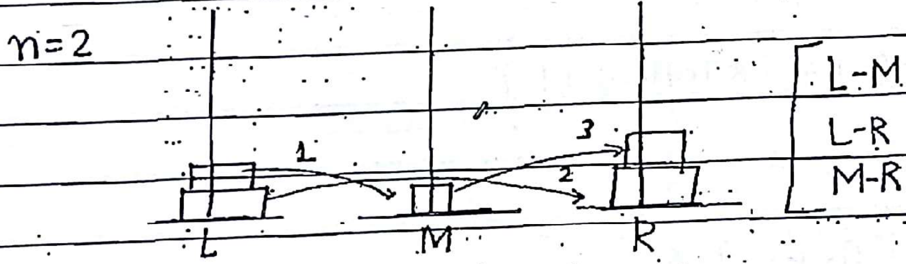
No of addition in $f(0) = 0$
$f(1) = 0$
$f(2) = 1$
$f(3) = 1+1 = 2$
$f(4) = 4$
$f(5) = 7$

$f(n) = f(n-1) + f(n-2) + 1$

No of Add^n $f_A(n) = \begin{cases} 0 ; \text{ if } n \le 1 \\ f(n-1)+f(n-2)+1 ; \text{ if } n > 1 \end{cases}$

Tower of Hanoi

n=2



L-M
L-R
M-R

n=3

L-R
L-M
R-M
] to move top 2 to Middle

L-R] to move larger ~~1 to R~~

M-L
M-R
L-R
] to move top 2 to Right

n=4

L-M
L-R
M-R
L-M
R-L
R-M
L-M
L-R
M-R
M-L
R-L

M-R
L-M
L-R
M-R

n x, y, z
↓
n-1 x, z, y
~~x, y, z~~ print(x)
~~n-1 y, x, z~~

$$TOH(n) = \begin{cases} TOH(n-1, x, z, y); \\ TOH(1, x, y, z); \\ TOH(n-1, y, x, z); \end{cases}$$

TOH(4) = {
TOH(3, L, R, M); // Move to Middle towers
MOV(L→R) // Move to Right
TOH(3, M, L, R);
}

- for n=3, solⁿ is obtained in the form of n=2
  - Move 2 to Middle tower
  - Move 3rd plate to Destination tower.
  - Now Move 2 plates from Middle tower to destination

TOH(5) = {
TOH(4, L, R, M)
MOV(L→R)
TOH(4, M, L, R)
}

Total = 31

Total Moves, TOH(n) = 2×TOH(n) + 1

Recursive Program-

(L = source, M = intermittent, R = destination)

```
TOH(n, L, M, R)
{
    if(n==0) return;
    else
    {
        TOH(n-1, L, R, M)
        MOV(L to R)
        TOH(n-1, M, L, R)
    }
}
```

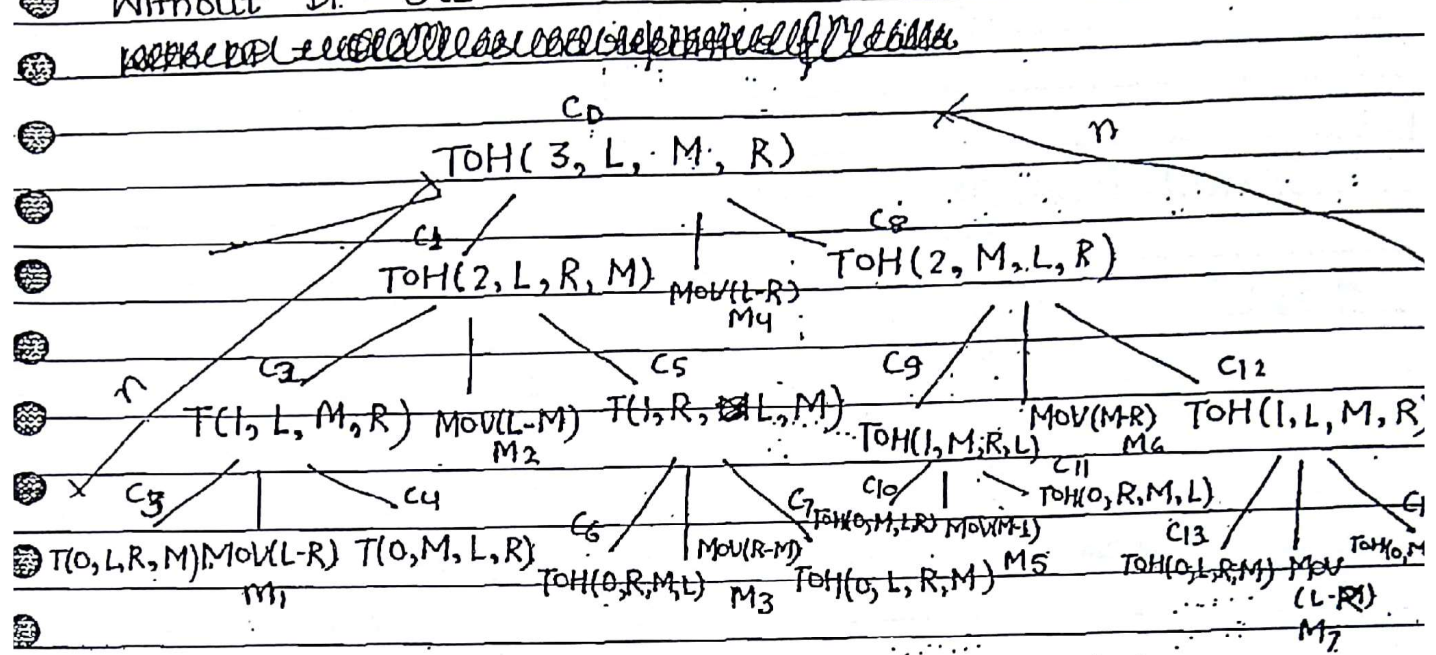
$TC = O(2^n)$
↓ complete binary tree

EXCELLENT

Date: / /

(improve count)

Without DP = $\Theta(2^n)$



stack space = $O(n)$

Ques1 - In TOH(n), is there any f^n call after Move?

Ans - 1 f^n call after M_7

(1) f^n call after last move

Ques2 - In TOH(n), is there any move after last f^n call?

Ans - No due to tail recursion.

(After C_{15}, in backtracking, no work)

Ques3 - In TOH(n), after how many f^n call 1st move takes place?

Ans - 4 f^n call before M_1

$(n+1)$ f^n call (left most path)

(After C_4, there is M_1)

Ques4 - In TOH(n) How many f^n call are there?

TOH(0) = 1 $\quad 2^{0+1}-1$

TOH(1) = 3 $\quad 2^{1+1}-1$

TOH(2) = 7 $\quad 2^{2+1}-1$

$[\text{TOH}(n) = 2^{n+1}-1]$

EXCELLENT

Page No.
Date : / /

No of f^n call, $T(n) = \begin{cases} 1 & ; \text{if } n=0 \\ 2T(n-1)+1 \end{cases}$

as $T(3) = T(2) + T(2) + 1$
(Left tree) (Right tree)
$= 7 + 7 + 1 = 15$

Ques -
IN TOH(n), How many Move are there?

TOH(0) = 0
TOH(1) = 1
TOH(2) = 3
TOH(3) = 7

Moves, TOH(n) = 2×TOH(n)+1

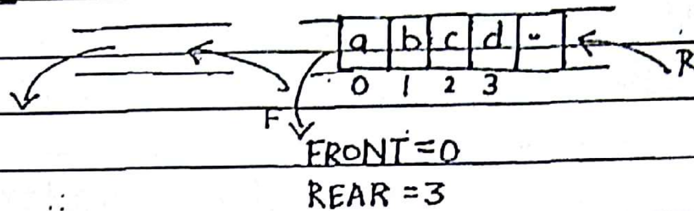
or

$TOH(n) = 2^n - 1$

Moves, $TOH(n) = \begin{cases} 0 & \text{if } n=0 \\ 1 & \text{if } n=1 \\ 2 * TOH(n)+1 & ; \text{otherwise} \end{cases}$

EXCELLENT

QUEUE open from both side, one side insertion, one time deletion
- FIFO



FRONT = 0
REAR = 3

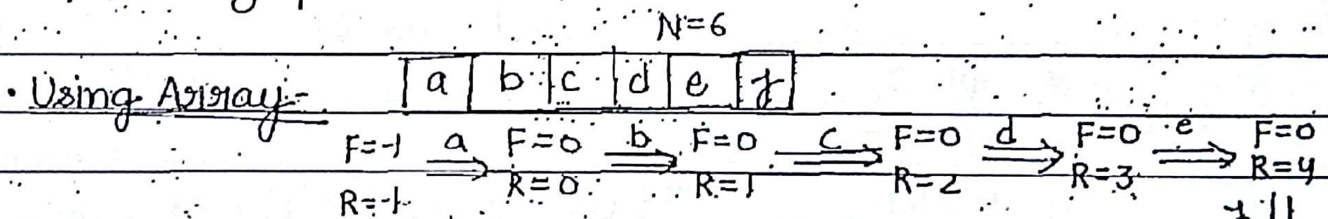
FRONT is a variable which ~~stores~~ stores the first posⁿ of the queue it help to delete the elmt.
REAR is a variable which contain posⁿ of element newly inserted. it helps to insert the elmt.

ADT of a queue - • ENQUEUE()
• DEQUEUE()

Implementing Queue -

N=6

• Using Array -




```
int QUEUE EMPTY( )
{
  if (R == -1 && F == -1)
    return 1;
  return 0;
}
```

```
int FULL( )
{
  if(R == N-1)
    return 1;
  return 0;
}
```

Date Date : / /

```
Void ENQUEUE(int x)
{
    if (R+1 == N)
      { printf("QUEUE OVERFLOW"); exit(1); }
    R = R+1;
    Q[R] = x;
    if (F == -1)
        F = 0;
}
```

{Time complexity = O(1)} (BC, AC, WC)

```
if (F == -1 && R == -1)
{ F = 0  R = 0
} else R = R+1
Q[R] = x;
```

```
int DEQUEUE()
{ int x;
    if (R == -1 && F == -1)
      { printf("QUEUE UNDERFLOW");
        exit(1);
      }
    x = Q[F];
      F = F+1;
      if (F > R)
        { F = R = -1;
        }
    return x;
}
```

{Time Complexity = O(1)} (BC, AC, WC)

(• Synchronization b/w rear & front is required here)

It can also be written as

```
{ int x;
   if (R == -1 && F == -1)
     { pf("Queue Underflow
       exit(1);
     }
   if (F == R)
     { x = Q[F];
       F = R = -1;
     }
   else
     { x = Q[F]
           F = F+1
     }
   return x;
```

{TC = O(1)}

EXCELLENT

[In Enqueue if you remove F==-1 && R==1 & uses ~~R==-1~~ both have same meaning]

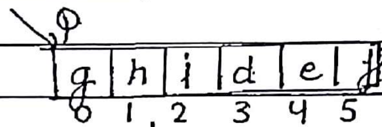
Circular Queue - Implementing queue Efficiently

~~Linear Queue is eri~~ ~~(Linear)~~

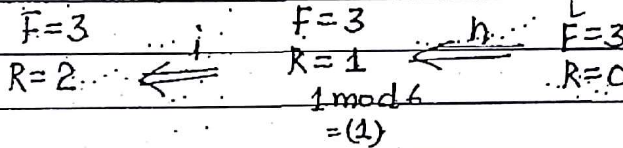
Note - Using above program, we can implement queue but It is not efficient because after rear reaching right most place, it can not go back even though space is available.

- To implement queue efficiently, we are moving toward circular queue.

Circular Queue -



F=3
R=5 → in case of circular, use { R=(R+1)Mod N }



if(F ==(R+1)Mod N)
then circular queue is full.

if (F = -1 && R=-1)
circular queue is empty

void ENQUEUE(char x)
{

```
else
    R=(R+1) Mod N;
  Q[R]=x;
}
char DEQUEUE( )
{
  char y;
  if( F==-1 && R==-1)
  { printf("Queue UnderFlow");
    exit(1);
  }
  else
  { y=Q[F];
    if(F==R)
    {
        F=-1; R=-1;
    }
    else
    {
        F=(F+1) Mod N;
    }
    return y;
}
```

$\Rightarrow TC = O(1)$

EXCELLENT