

DATA

STRUCTURE

(8)

Data Structure

References

1. The C-Programming Language
by
Kernighan & Ritchie 2nd Edition
2. Data structure in c by
Tenenbourn
3. Google

Syllabus

- 1- Array
- 2- Stack
- 3- Queue
- 4- Linked List
- 5- Tree (BST, AVL, B-Tree, B⁺-Tree)
- 6- C-Concepts (Pointers)

DATA

STRUCTURE

DELTA	Page No. _____		
	Date: <table border="1"><tr><td></td><td></td><td></td></tr></table>		

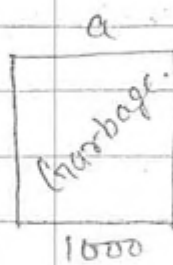
Arrays

#

Declaration

`int a ;` Creating memory for the variable `a` which will takes 2-bytes.

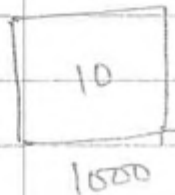
(Declaration is not for user it is for compiler)



#

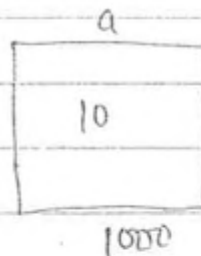
Initialization

`a a = 10 ;`



#

`int a = 10 ;`



#

`printf( "%d", a )`

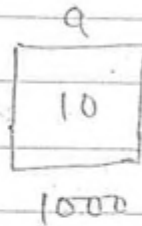
⇓

variable-name indicates values.

O/p : 10.

#

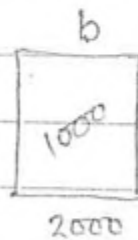
int a = 10 ; / int a;
a = 10;



// Normal variable
initialization

#

int *b



It is the compiler
purpose.
not for user.

#

b = &a

// Pointer variable
initialization.

#

printf ("%d", a) ⇒ 10

printf ("%s", *b) ⇒ 10

↓
*(1000)
↓
10

#

& ⇒ Reference operator.

* ⇒ de-reference operator.

#

int a = 10

a immediate addressing mode $\Rightarrow$ (1 memory Access)

10 $\Rightarrow$ Value. a = 10

1000

int *b

b = &a

b Direct addressing mode $\Rightarrow$ (2 memory access)

1000 $\Rightarrow$ Address.

2000

*b = 10

int **c

c = &b

c Indirect addressing mode $\Rightarrow$ (3 memory access)

2000

3000

**c = 10

Array

int a = 10 ;

a

10

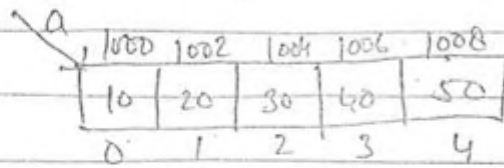
1000

int a[5] ;

a

1000	1002	1004	1006	1008
0	0	0	3	4

`int a[5] = {10, 20, 30, 40, 50}`



here `a` $\rightarrow$ base address (0th element address)

* The main advantage of array is Random access becoz of Pointer Concepts.

`a[3] = 40`

`* (a+3)`

$\Downarrow$

`* (a+6)`

`* (1000+6)`

`* (1006)`

$\Downarrow$

40

`* (a+0) = 10`

`a[0] = 10`

for compiler

$$\begin{aligned} \# \quad & \boxed{*(a+4)} = *(a+8) \\ & \text{or} \quad = *(1000+8) \\ & \boxed{a[4]} = *(1008) \\ & = 50 \end{aligned}$$

for user

address)

$$\# \quad a[i] = *(a+i) = *(i+a) = i[a]$$

in address

$$\begin{aligned} \# \quad & a[1] / *(a+1) = *(1000+2) \\ & = *(1002) \\ & = 20 \end{aligned}$$

$$\begin{aligned} a[4] &= *(a+4) \\ &= *(4+a) \\ &= 4[a] \end{aligned}$$

One Dimensional Array.

`int a[5] = {10, 20, 30, 40, 50}`

a	1000	1002	1004	1006	1008
	10	20	30	40	50
	0	1	2	3	4

$$\begin{aligned}
 \text{Loc}(a_{1,4}) &= 1000 + [(1-1) \times 4 + (4-1)] \times 2 \\
 &= 1000 + [3] \times 2 \\
 &= \underline{\underline{1006}}
 \end{aligned}$$

* Consider the following array declaration.

$a[0 \text{ --- } 10, 0 \text{ --- } 10]$

Base address = 1000

$C = 10$, RMD, then.

Find the location of $(a_{8,7})$.

no of row = $10 - 0 + 1 = 11$

no of column = $10 - 0 + 1 = 11$

Solⁿ

$$\begin{aligned}
 \text{Loc}(a_{8,7}) &= 1000 + [(8-0) \times 11 + (7-0)] \times 10 \\
 &= 1000 + [88 + 7] \times 10 \\
 &= 1000 + 950 \\
 &= \underline{\underline{1950}}
 \end{aligned}$$

* Consider the an array $A[-5 \text{ --- } +5, -5 \text{ --- } +5]$

BA = 0, $C = 10$, RMD

Find location of $(a_{0,0})$?

no of row = $5 + 5 + 1 = 11$

column = $5 + 5 + 1 = 11$

Solⁿ

$$\begin{aligned}
 \text{Loc}(a_{0,0}) &= 0 + [(0+5) \times 11 + (0+5)] \times 10 \\
 &= 0 + [60] \times 10 \\
 &= \underline{\underline{600}}
 \end{aligned}$$

If $Loc(a_{3,5}) = ?$

$$\begin{aligned}
 Loc(a_{3,5}) &= 0 + [(3+5) \times 11 + (5+5)] \times 10 \\
 &= 0 + [22 + 10] \times 10 \\
 &= 320
 \end{aligned}$$

Column-Major-Order

int a[1---4, 1---4]

a_{11}	a_{12}	a_{13}	a_{14}
a_{21}	a_{22}	a_{23}	a_{24}
a_{31}	a_{32}	a_{33}	a_{34}
a_{41}	a_{42}	a_{43}	a_{44}

5---15]

1000	02	04	06	08	10	12	14	16	18	20	22	24	26	28	30
a_{11}	a_{12}	a_{13}	a_{14}	a_{12}	a_{22}	a_{32}	a_{42}	a_{13}	a_{23}	a_{33}	a_{43}	a_{14}	a_{24}	a_{34}	a_{44}
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

eg

$$\begin{aligned}
 Loc(a_{3,4}) &= 1000 + [(4-1) \times 4 + (3-1)] \times 2 \\
 &= 1000 + 10 = 1010
 \end{aligned}$$

Q9

$$\begin{aligned}
 \text{Loc}(a_{4,3}) &= 1000 + [(3-1) \times 4 + (4-1)] \times 2 \\
 &= 1000 + 22 \\
 &= \underline{\underline{1022}}
 \end{aligned}$$

Q10

$$\begin{aligned}
 \text{Loc}(a_{2,1}) &= 1000 + [(1-1) \times 4 + (2-1)] \times 2 \\
 &= 1000 + 2 \\
 &= \underline{\underline{1002}}
 \end{aligned}$$

Consider the array A which is declared as follows:

$$a[\dots +5, -5 \dots +25]$$

$$bA = 0, \quad c = 5, \quad cmo$$

find location of $a[2, 25]$?

Soln

$$\text{no of rows} = +5 + 5 + 1 = 11$$

$$\text{columns} = 25 + 5 + 1 = 31$$

$$\begin{aligned}
 \text{Loc}(a_{2,25}) &= 0 + [(25+5) \times 11 + (2+5)] \times 5 \\
 &= 0 + [339] \times 5 \\
 &= \underline{\underline{1685}}
 \end{aligned}$$

$$\begin{aligned}
 \text{Loc}(a_{-4, -1}) &= 0 + [(-1+5) \times 11 + (-4+5)] \times 5 \\
 &= 0 + [44 + 1] \times 5 \\
 &= \underline{\underline{225}}
 \end{aligned}$$

Formula

Column Major Order

if $[lb_1 \dots ub_1, lb_2 \dots ub_2]$

BA, c, nr, nc, cmo then

find $\text{Loc}(i, j)$?

Where

BA = Base Address

c = Size of element

nr = no of row.

nc = " " columns.

CMO = Column Major Order.

$$nr = ub_1 - lb_1 + 1$$

$$nc = ub_2 - lb_2 + 1$$

$$\text{Loc}(a_{ij}) = BA + [(j - lb_2) \times nr + (i - lb_1)] \times c$$

Row Major Order

$$A [lb_1, \dots, lb_1, lb_2, \dots, lb_2]$$

BA, C, RMD

$$nr = ub_1 - lb_1 + 1$$

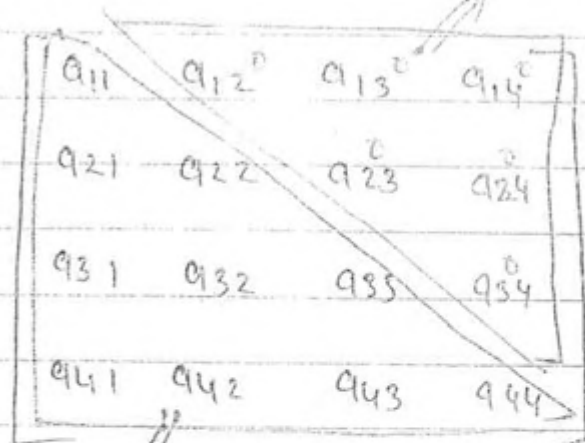
$$nc = ub_2 - lb_2 + 1$$

$$LOC(a_{ij}) = BA + [(i - lb_1) \times nc + (j - lb_2)] \times C$$

LOWER TRIANGULAR MATRIX [LTM]

(Only Square Matrices)

int a[1---4, 1---4]



7/11/2020

A is said to be LTM

iff

$$A[i, j] = 0 \text{ if } i < j$$

$$A[i, j] = \text{non zero if } i \geq j$$

~~b_2~~ ~~xc~~ ~~XX~~

LTM

ed

~~zero~~

~~a_{14}~~
 ~~a_{24}~~
 ~~a_{34}~~
 ~~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}

Row Major Order

	02	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}
c	1	2	3	4	5	6	7	8	9

$$LOC(a_{4,3}) = 1000 + \left[\underset{\substack{\text{if} \\ \text{natural no of} \\ \text{sum}}}{(4-1) + (3-1)} \right] * 2$$

$$= 1000 + \left[\frac{(4-1)(4+1)}{2} + 2 \right] * 2$$

$$= 1000 + 16$$

$$= \underline{\underline{1016}}$$

Column - Major - Order

a ₁₁	0	0	0
a ₂₁	a ₂₂	0	0
a ₃₁	a ₃₂	a ₃₃	a ₃₄
a ₄₁	a ₄₂	a ₄₃	a ₄₄

1000	02	04	06	08	10	12	14	16	18
a ₁₁	a ₂₁	a ₃₁	a ₄₁	a ₂₂	a ₃₂	a ₄₂	a ₃₃	a ₄₃	a ₄₄
0	1	2	3	4	5	6	7	8	9

$$LOC(a_{4,3}) = 1000 + \left[\frac{4 \times 5}{2} - \frac{2 \times 3}{2} + (4-3) \right] \times 2$$

$$= 1000 + [0] \times 2$$

$$= \underline{\underline{1006}}$$

$$LOC(a_{44}) = 1000 + \left[\frac{4 \times 5}{2} - \frac{(4-4+1)(4-4+1)}{2} + (4-4) \right] \times 2$$

$$= 1000 + 18$$

$$= \underline{\underline{1018}}$$

$$\begin{aligned}
 LOC(a_{4,1}) &= 1000 + \left[\frac{4 \times 5}{2} - \frac{(4-1+1)(4-1+1+1)}{2} + (4-1) \right] \times 2 \\
 &= 1000 + [10 - 10 + 3] \times 2 \\
 &= \underline{\underline{1006}}
 \end{aligned}$$

#

$$A \left[-25 \dots +25, -25 \dots +25 \right]$$

CMD, LTM

$$BA = 0 \quad nR = 51$$

$$e = 1 \quad nL = 51$$

Find ?

$$LOC(a_{00}) = 0 + \left[\frac{51 \times 52}{2} - \frac{(25-0+1)(25+1+1)}{2} + (0-0) \right] \times 2$$

=

UPPER TRIANGULAR MATRIX

A matrix A is said to be upper triangular matrix
iff.

$$A[i, j] = 0 \text{ if } i > j$$

a_{11}	a_{12}	a_{13}	a_{14}
a_{21}	a_{22}	a_{23}	a_{24}
a_{31}	a_{32}	a_{33}	a_{34}
a_{41}	a_{42}	a_{43}	a_{44}

zeros

non zero

Column major order

1000	02	04	06	08	10	12	14	16	18
a_{11}	a_{12}	a_{22}	a_{13}	a_{23}	a_{33}	a_{14}	a_{24}	a_{34}	a_{44}
0	1	2	3	4	5	6	7	8	9

matrix no sum

$$LOC(a_{3,4}) = 1000 + \left[\frac{(4-1)(4-1)}{2} + (3-1) \right] \times 2$$

$$= 1000 + 16$$

$$= 1016$$

$$\begin{aligned}
 \text{LOC}(a_{2,3}) &= 1000 + \left[\frac{(3-1)(3-1+1)}{2} + (2-1) \right] * 2 \\
 &= 1000 + 8 \\
 &= \underline{\underline{1008}}
 \end{aligned}$$

Formula

$$A[lb_1 \dots ub_1, lb_2 \dots ub_2]$$

BA, c, nr, cr, UTM, CMO, RMO

Find location of $a(i,j) = ?$.

$$\text{LOC}(a_{i,j}) = BA + \left[\frac{(j-lb_2)(j-lb_2+1)}{2} + (i-lb_1) \right] * c$$

Column Major order

$$nr = ub_1 - lb_1 + 1$$

$$nc = ub_2 - lb_2 + 1$$

$$\begin{aligned}
 \text{LOC}(a_{i,j}) &= BA + \left[\frac{nr(nr+1)}{2} - \frac{(ub_1-i+1)(ub_1-i+1)}{2} \right. \\
 &\quad \left. + (j-lb_2) \right] * c
 \end{aligned}$$

Row-Major Order

UPPER TRIANGULAR MATRIX

Row Major Order

a_{11}	a_{12}	a_{13}	a_{14}
a_{21}	a_{22}	a_{23}	a_{24}
a_{31}	a_{32}	a_{33}	a_{34}
a_{41}	a_{42}	a_{43}	a_{44}

1000	02	04	06	08	10	12	14	16	18
a_{11}	a_{12}	a_{13}	a_{14}	a_{22}	a_{23}	a_{24}	a_{33}	a_{34}	a_{44}
0	1	2	3	4	5	6	7	8	9

$$\begin{aligned}
 LOC(a_{34}) &= BA \cdot 1000 + \left[\frac{4 \times 5}{2} - \frac{(4-3+1)(4-3+1+1)}{2} + (4-3) \right] \times 2 \\
 &= 1000 + [10 - 3 + 1] \times 2 \\
 &= 1000 + 14 \\
 &= 1014
 \end{aligned}$$

$$\begin{aligned}
 LOC(a_{14}) &= 1000 + \left[\frac{4 \times 5}{2} - \frac{(4-1+1)(4-1+1+1)}{2} + (4-1) \right] \times 2 \\
 &= 1000 + [10 - 10 + 3] \times 2 \\
 &= 1006
 \end{aligned}$$

Q. 1. A one dimensional Array A contains $A[1 \dots 75]$. Each element is a string and takes 3 bit. The Array starts at location 1120. Then what is the location of $A(4,9)$.

- a) 1126 b) 1164
 c) 1264 d) 1169

Q. 2. Consider the following declaration of two dimensional array in C: `char a[100][100]`. Assume that array is stored row major address. What is location of $A[40][50]$.

- a) 4040 b) 4050
 c) 5040 d) 5050

$$+1) + (4-1) \times 2$$

$A[1 \dots 75]$

$c = 3$

BA = 1120

$$LOC(A_{49}) = 1120 + (49-1) \times 3$$

$$= 1264$$

$$(4-1) \times 2$$

Soln 2

$$LOC(a_{40,50}) = 0 + [(40-0) \times 100 + (50-0)] \times 1$$

$$= 4050$$

Q3 Assume that a LTM $A[0 \dots (n-1), 0 \dots (n-1)]$ is stored in linear Array $B[0 \dots \frac{1}{2}n(n+1)-1]$ in row by row order.

For $n=100$; If $A[00]$ is stored in $B[0]$ where is $A[90,00]$ is stored in B ?

- ☒ a) 4175 b) 0
☐ c) 4165 d) 4160

Sol

$$A[0 \dots (n-1), 0 \dots (n-1)]$$

$$B[0 \dots \frac{1}{2}n(n+1)-1]$$

$$B[0 \dots \frac{1}{2}n(n+1)-1]$$

$$B[0 \dots -9]$$

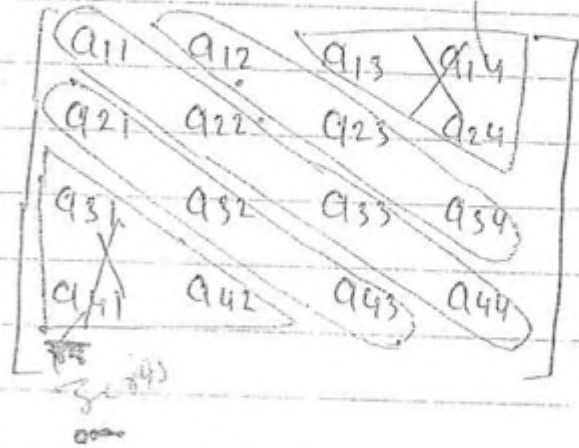
$$C=1, BA=0 \text{ RMO, LTM}$$

$$\begin{aligned}
 LOC(A_{90,00}) &= 0 + \left[\frac{(90-0)(90-0+1)}{2} + (00-0) \right] K1 \\
 &= 4175
 \end{aligned}$$

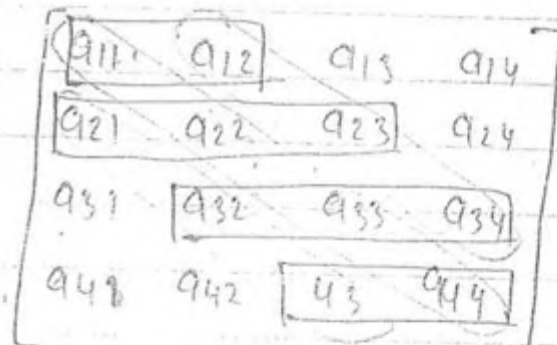
Tridiagonal - Matrix

$$4 \times 4 = 16 \quad | \quad 3 \times 3 = 9$$

$$4 + 3 + 3 = 10 \quad | \quad 3 \times 3 = 9$$



Row-Major - Order



0	1	2	3	4	5	6	7	8	9
a11	a12	a21	a22	a23	a32	a33	a34	a43	a44

$$LOC(a_{4,3}) = 1000 + [3(4-1) - 1 + 3 - (4-1)] \times 2$$

$$= 1000 + [16]$$

$$= 1016$$

$$\begin{aligned}
 LOC(a_{3,2}) &= 1000 + [3(3-1) - 1 + 2(3-1)] \times 2 \\
 &= 1000 + [5 + 2] \times 2 \\
 &= 1010
 \end{aligned}$$

Formula

$$LOC(a_{i,j}) = BA + [3(i-lb_i) - 1 + (J-(j-1))] \times AC$$

Column Major Order

a_{11}	a_{12}	a_{13}	a_{14}
a_{21}	a_{22}	a_{23}	a_{24}
a_{31}	a_{32}	a_{33}	a_{34}
a_{41}	a_{42}	a_{43}	a_{44}

1000	02	04	06	08	10	12	14	16	18
a_{11}	a_{21}	a_{12}	a_{22}	a_{32}	a_{23}	a_{33}	a_{43}	a_{34}	a_{44}
0	1	2	3	4	5	6	7	8	9

$$\begin{aligned}
 LOC(a_{3,4}) &= 1000 + [3(4-1) - 1 + 3-(4-1)] \times 2 \\
 &= 1000 + 16 \\
 &= 1016
 \end{aligned}$$

Formula

$$LOC(a_{i,j}) = BA + [3(j - lb_2) - 1 + i - (j - 1)] \times C$$

Diagonal by Diagonal

a_{11}	a_{12}	a_{13}	a_{14}
a_{21}	a_{22}	a_{23}	a_{24}
a_{31}	a_{32}	a_{33}	a_{34}
a_{41}	a_{42}	a_{43}	a_{44}

Lower			Middle				Upper		
a_{100}	a_2	a_{11}	a_6	a_8	a_{10}	a_{12}	a_{14}	a_{16}	a_{18}
a_{21}	a_{32}	a_{43}	a_{11}	a_{22}	a_{33}	a_{44}	a_{12}	a_{23}	a_{34}
0	1	2	3	4	5	6	7	8	9

$$LOC(a_{i,j})$$

$$\Downarrow$$

$$if (i == JH)$$

$$LOC(a_{i,j}) = BA + [i - 2] \times C \Rightarrow \text{lower}$$

LOC ($a_{i,j}$)

↓

if ($j == J$)

$$\boxed{\text{LOC}(a_{i,j}) = BA + [N-1 + (j-1)] * C} \text{ - Middle}$$

LOC ($a_{i,j}$)

↓

if ($j == J-1$)

$$\boxed{\text{LOC}(a_{i,j}) = BA + [N-1 + N + (j-1)] * C} \text{ - ~~Lower~~}$$

↓
Upper

Z - Matrix

a_{11}	a_{12}	a_{13}	a_{14}
a_{21}	a_{22}	a_{23}	a_{24}
a_{31}	a_{32}	a_{33}	a_{34}
a_{41}	a_{42}	a_{43}	a_{44}

Store 1st row, Last Row and middle.

1st Row				Last Row				middle	
a ₁₁	a ₁₂	a ₁₃	a ₁₄	a ₄₁	a ₄₂	a ₄₃	a ₄₄	a ₂₃	a ₃₂
0	1	2	3	4	5	6	7	8	9

Middle

$$LOC(a_{i,j})$$

↓

if ($i = lb_1$)

$$LOC(a_{i,j}) = BA + \left(\frac{lb_2}{j - lb_1} \right) * C \Rightarrow \text{1st Row}$$

$$LOC(a_{i,j})$$

↓

if ($i = ub_1$)

$$LOC(a_{i,j}) = BA + [N + (j - ub_1)] * C \Rightarrow \text{Last Row}$$

$$LOC(a_{i,j})$$

↓

if ($i \neq lb_1$ & $i \neq ub_1$ & $i + j = N + 1$)

$$LOC(a_{i,j}) = BA + [N + N + (j - 2)] * C \Rightarrow \text{middle}$$

Toeplitz - Matrix

a_{11}	a_{12}	a_{13}	a_{14}
a_{21}	a_{22}	a_{23}	a_{24}
a_{31}	a_{32}	a_{33}	a_{34}
a_{41}	a_{42}	a_{43}	a_{44}

Matrix A is said to be Toeplitz-Matrix iff.

$$A[i, j] = A[i-1, j-1] \text{ for all}$$

$$i > 1 \text{ \& \& } j > 1$$



$$4 \times 4 \Rightarrow n^2$$

$$4 + 4 - 1 = 7 \Rightarrow 2N - 1$$

10	20	30	40
50	10	20	30
60	50	10	20
70	60	50	10

⇒ Store 1st row & remaining elements of 1st column.

	1 st row				1 st column (Remaining)			
	00	02	04	06	08	10	12	14
0	a_{11}	a_{12}	a_{13}	a_{14}	a_{21}	a_{31}	a_{41}	
1								
2								
3								
4								
5								
6								

$$LOC(a_{i,j})$$

⇓

$$if(j \leq J)$$

$$LOC(a_{i,j}) = BA + (j-i) * C \quad \text{1st Row}$$

$$LOC(a_{i,j})$$

⇓

$$if(i > I)$$

$$LOC(a_{i,j}) = BA + N + j(J+1) * C \quad \left. \begin{array}{l} \text{1st Column} \\ \text{(Rowing)} \end{array} \right\}$$

#

	00	02	04	06	08
10	20	30	40	50	
0	1	2	3	4	

$$LOC(a_{[3]}) = BA + (3-0) * 2$$

$$= 1000 + 6$$

$$= * (1000)$$

$$= 46$$

offset
no need to
calculate

#

10	20	30	40	50
3	4	5	6	7

Offset (Should be calculated)

$$\begin{aligned}
 *(\text{LOC}(a[6])) &= BA + (6-3) * 2 \\
 &= 1000 + 6 \\
 &= *(1006) \\
 &= 40
 \end{aligned}$$

Note :- Why Array always start from 0, why not 1.

If Array is starting from 0 no need offset value. If array is starting other than 0 we have find offset value.

int a[5] = { 10, 20, 30, 40, 50 } ;

a	02	04	06	08
1000				
10	20	30	40	50
0	1	2	3	4

$$\begin{aligned}
 &*(a+1) = a[1] \\
 &\quad \downarrow \quad \quad \downarrow \\
 &*(1000+2) \quad *(a+1) \\
 &\quad \downarrow \quad \quad \downarrow
 \end{aligned}$$

be
calculated)

$\times (1002)$
 $\Downarrow$
20

$\times (1+a)$
 $\Downarrow$
1[4]

at X, $[a = a + 1]$
 $\Downarrow$ $\Downarrow$
L-Value R-Value

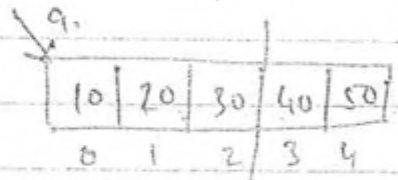
$\Downarrow$
L-value can't be changed.

in D, why

need
of other
w.

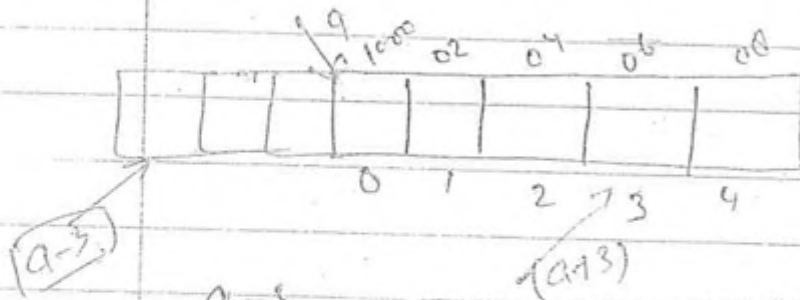
$\int a[5] = \{10, 20, 30, 40, 50\};$ ✓
 $\int a[] = \{10, 20, 30, 40, 50\};$ ✓
 $\Downarrow$ optional

✓ $\int a[5] = \{10, 20, 30, 40, 50\}$



$a[0]$
 $a[1]$
 $a[2]$
 $a[3] =$ Garbage, becoz total element you store.

$\text{int } a[5] = \{10, 20, 30, 40, 50\};$



$a-3$
↓
before 3 elements.

$a+3$
↓
we will skip 3 elements.
and we will use 4th
elements.
↓
after 3 elements

$\text{int } a[3, 2] = \{10, 20, 30, 40, 50, 60\};$

$\text{int } a[] [] = \{10, 20, 30, 40, 50, 60\};$
options: 1 6
2 3
3 2
6 1

✓ $\text{int } a[2][3][2] = \{10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120\};$

$\text{int } a[][][] = \{10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120\}$
 optional Mandatory Mandatory

✓ In multidimensional array (M) we have to specify (m-1) value.

$\text{inta}[4][5][6];$

4 array and each array size is (5x6).

$nr = Ub_2 - lb_2 + 1$

$nc = Ub_3 - lb_3 + 1$

$na = Ub_4 - lb_4 + 1$

Size of array = $nr \times nc$

$a[0] \rightarrow 5 \times 6$

$a[1] \rightarrow 5 \times 6$

$a[2] \rightarrow 5 \times 6$

$a[3] \rightarrow 5 \times 6$

$\text{int } a[4][5][6]$

$a[lb_1 : Ub_1, lb_2 : Ub_2, lb_3 : Ub_3]$

Row Major Order

$$LOC(a_{i,j,k}) = BA + [(i - lb_1)(nr \times nc) + (j - lb_2) \times nc + (k - lb_3)] \times C$$

Column Major Order

$$Loc(a_{i,j,k}) = BA + [(j-lb_1)(nr \times nc) + (k-lb_3)nr + (i-lb_2)] \times C$$

Q. What does 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);
}
```

```
printf("i=%d j=%d", i, j);
```

```
return(0);
```

```
}
```

a) 2, 2

b) 2, 1

c) 0, 1

d) 0, 2

Static Variable

Main ()

```

{
1 static int var = 5;
2 printf("%d", var--);
3 if (var)
4 main();
}

```

Post increment

$x = 4$

$y = x++ + x++$

Print (x, y):

$\downarrow$ $\downarrow$
 6 8

$x = 486$

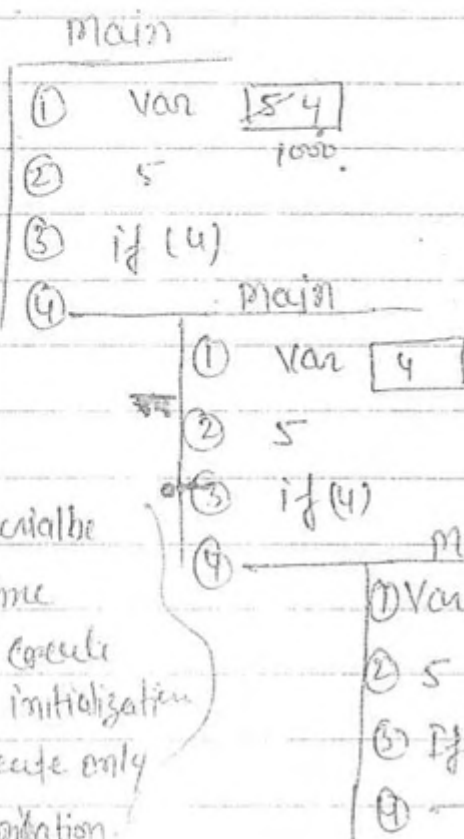
Pre Increment

$y = ++x + ++x$

Print (x, y):

$\downarrow$ $\downarrow$
 6 11

⇒ If static is not (without static)

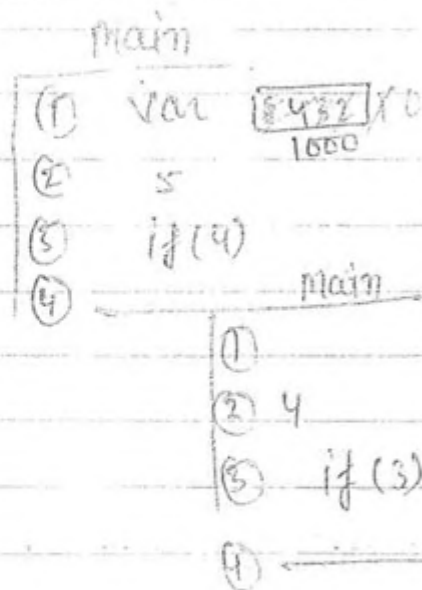


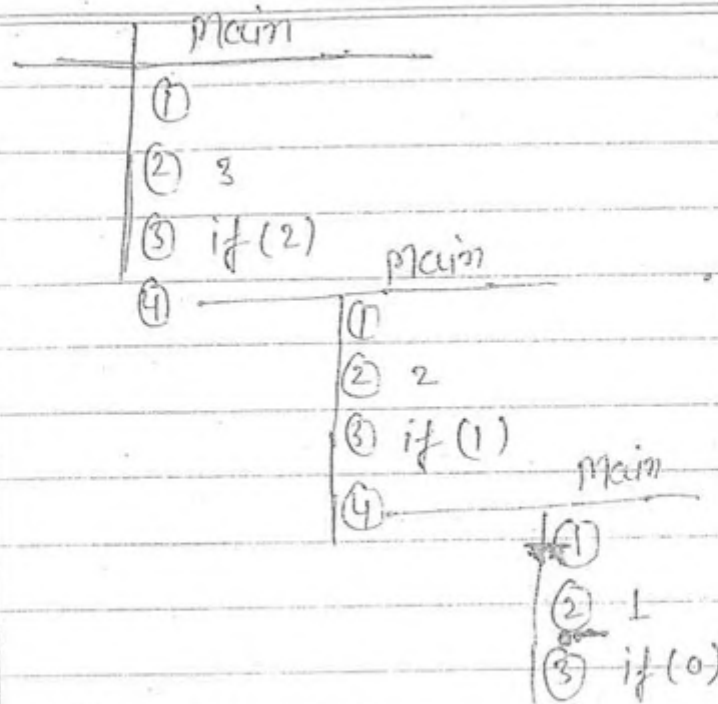
⇒ Error msg
Stack is overflow

Static variable
only one time
memory create
and also initialization
⇒ Memory create only
time of compilation.

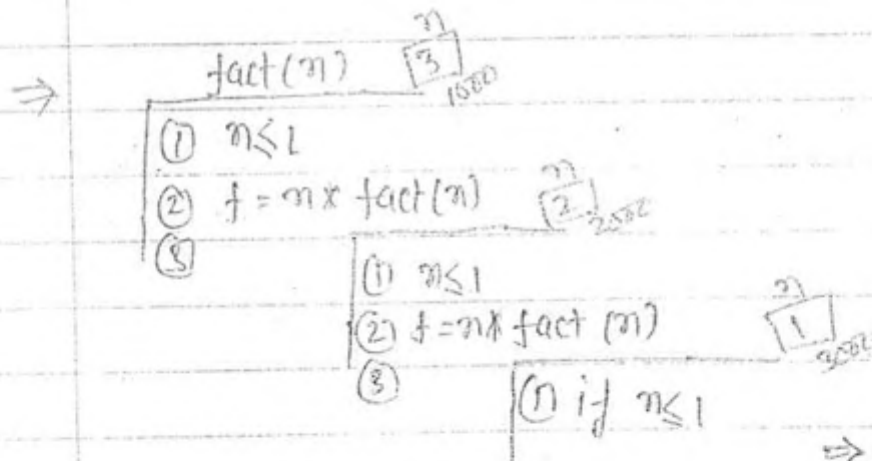
O/P ⇒ 5555 - -

⇒ If static is available (with static)





O/P $\Rightarrow$ 5, 4, 3, 2, 1.



$\Rightarrow$ O/P $\Rightarrow$ 6

Note

- ① For static variable memory will create only one becoz memory created at compile time.
- ② for static variable initialization will be done only once.

Ques Consider the following C Program.

```

int y;
y = func(x);
return (y);
}

```

```

fung (int n)
{
    static int y = 10;

```

```

    y = y + 1;

```

```

    return (n + y);
}

```

$5 + 11$
 $5 + 12$
 $5 + 13$
 $5 + 14$

O/P =

a) 42, 74

b) 80, 43

c) 43, 80

d) 43, 43.

Solⁿ

Consider the following program.

```

int incr (int i)
{

```

```

    static int count = 0;

```

```

    count = count + i;

```

```

    return (count);
}

```

Maine)

```

{
    int i, j;
    for (i = 0, i <= 4; i++)
    {
        j = incr(i);
    }
}

```

The value of j at the end of above program

a) 10

b) 4

c) 6

d) 7

```
# main()
{
    int i=0, J=1, K=-1, L=2, m;
    m = ( (J+1 + 2 * J+1 + 2 * L+1) ) / 2;
    printf (" %d %d %d %d %d ", J, J, K, L, m );
}
```

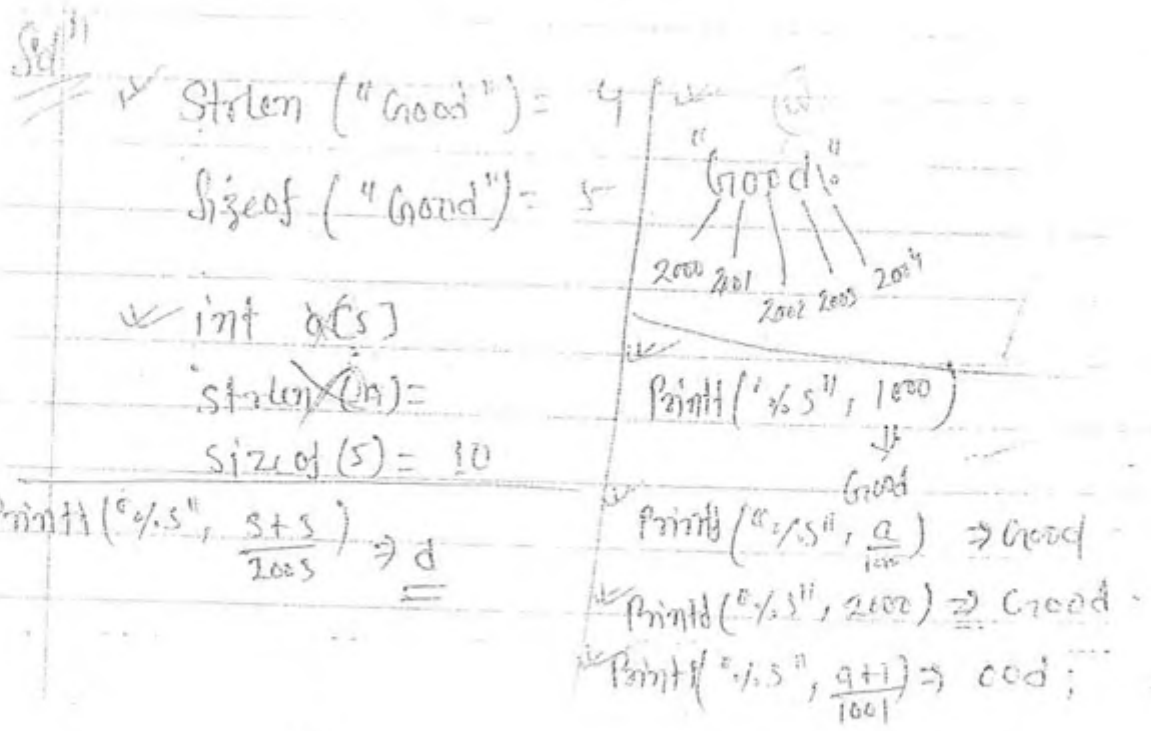
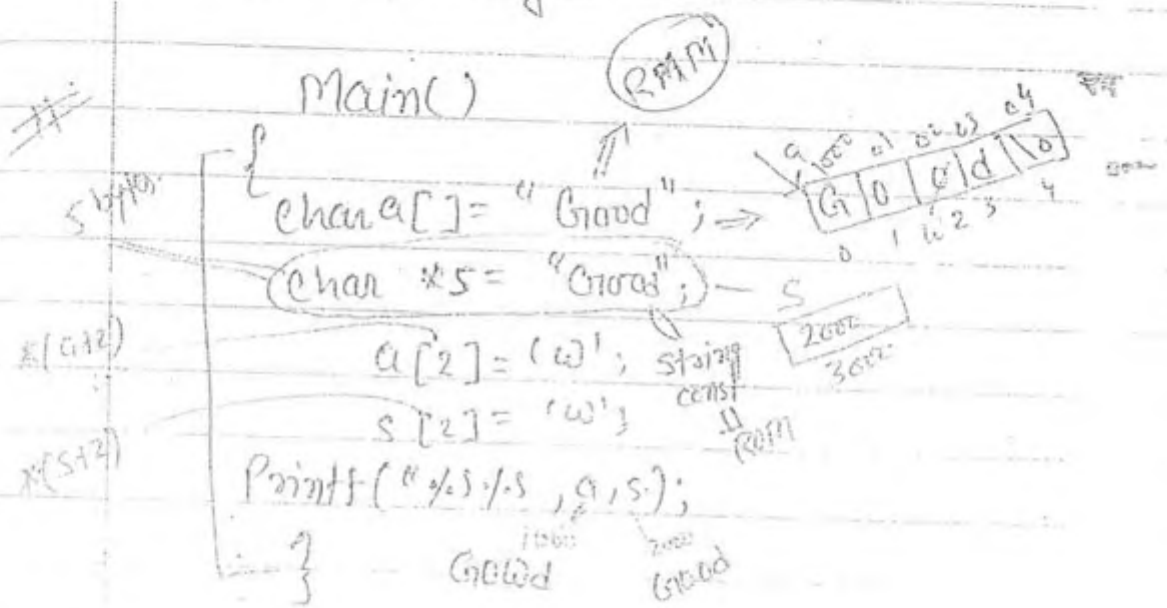
```
# void main()
{
    int i=0;
    for (i=0; i<20; i++)
    {
        switch (i)
        {
            case 0: i+=5;
            case 1: i+=2;
            case 8: i+=5;
            default: i+=4;
        }
        break;
    }
}
```

```
printf (" %d %d ", i);
```

o/p = 15, 21

Note:- In switch statement after every case there should be a break.

- * After default stmt. no need of break.
- * We can give case no as int const or char.
- * In switch stmt we can keep default stmt whenever we want. But it will execute when no one matching.



↙ printf ("%c", a) ;

↓

G

↙ printf ("%c", s[3]) ;

↓

d

↙ printf ("%c", a[3]) ;

↓
d

↙ printf ("%d", a)

↙ printf ("%d", *a)

↓

ASCII value

of G

↙ printf ("%c", a[2]) ;

↓

a[2] ✓

↙ printf ("%c", s[2])

↓

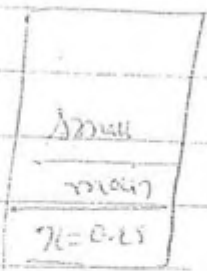
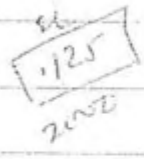
s[2]

called
follow

Show

Procedure small ()

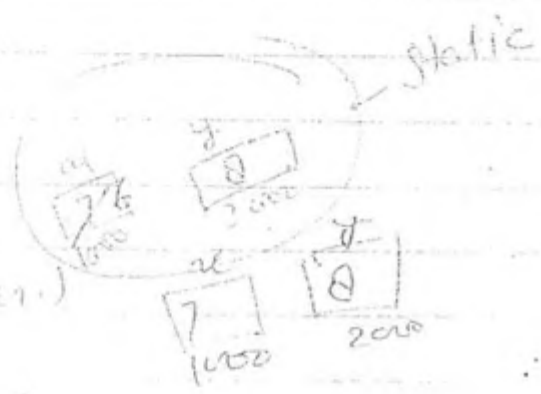
```
begin
  var u: real;
  u = 0.125;
  Show();
end;
```



```
begin
  u = 0.25;
  Show();
  small();
end;
```

Static: 0.25, 0.25

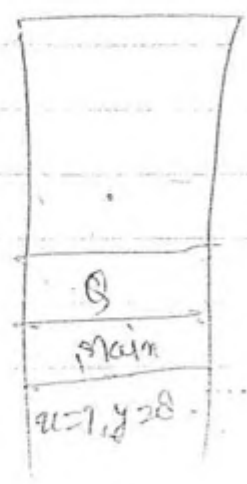
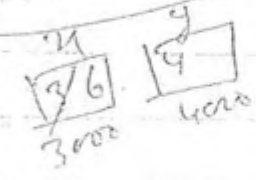
Dynamic: 0.25,



```
var u, y: integer;
Procedure P (var n: integer);
begin
  u = (n+4)/(n-5);
end
```

Procedure Q ()

```
begin
  var u, y: integer;
  u = 3, y = 4;
  P(y);
  write(u, y);
end;
```



```

begin
  x=7, y=8;
  PU
  while (x);
end.
  
```

Static : 3, 6.
Dynamic: 6, 7.

P

Var a, b: integer;
Procedure P()

a
5
1000

b
10
2000

```

begin
  a=5, b=10;
end
  
```

Procedure Q()

```

begin
  Var q, b: integer;
  q;
end
  
```

a
5
10

b
10
20

```

begin
  a=1, b=2;
  q;
  write a, b;
end;
  
```

Static: 5, 10
Dynamic: 1, 2

Q Consider the following C Program.

Var r: integer;

Procedure Two()

begin

write(r)

end;

procedure one()

begin

var r: integer;

r = 5;

Two;

end

begin

r = 2

Two;

One;

Two;

end;

Static: 2, 2, 4

Dynamic: 2, 5, 2

Parameter Passing Techniques.

- ① Call by value.
- ② Call by reference.
- ③ Call by copy-restore (or) copy in-copy out.
(or) value return
(or) value swap.
- ④ Call by Name.
- ⑤ Call by need.
- ⑥ Call by Text.

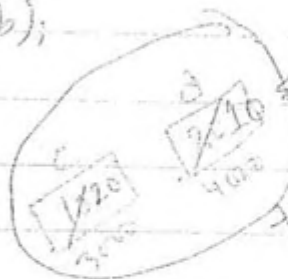
Call by Value: (Call by copy) {change in copy}

Main()

```
{
    int a=10, b=20;
    printf("%d %d", a, b);
    swap(a, b);
    printf("%d %d", a, b);
}
```

swap(int c, int d)

```
{
    int t;
    t=c;
    c=d;
    d=t;
}
```



Changes are taken place only here.

a/b - 10, 20 (without swap)
- 20, 20 (swap)

Soln

Print array (a, i, j)

```

{
    if (i == j)
    {
        printf("%d", a[i]);
        return;
    }
    else
    {
        printf("%d", a[i]);
        print array (a, i+1, j);
    }
}

```

1 5
2 5
3 5
4 5
5 5

Q

Write a recursive C-Program to print the given array of n elements in reverse order.

Sol

10 | 20 | 30 | 40 | 50

10 | 20 | 30 | 40 | 50

10 | 20 | 30 | 40, 50

10 | 20 | 30, 40, 50

10 | 20 | 30 | 40 | 50

10, 20, 30, 40, 50

Application of Stack

- (i) Recursion
- (ii) Infix to postfix conversion.
- (iii) Prefix to Postfix
- (iv) Fibonacci Series:
- (v) Towers of Hanoi.

Recursion

Q Write a recursive C-Program to print the given array of n-elements ($n=10$).

Solⁿ

10	20	30	40	50
----	----	----	----	----

1 2 3 4 5

10	20	30	40	50
----	----	----	----	----

2 3 4 5

10, 20	30	40	50
--------	----	----	----

3 4 5

10, 20, 30	40	50
------------	----	----

4 5

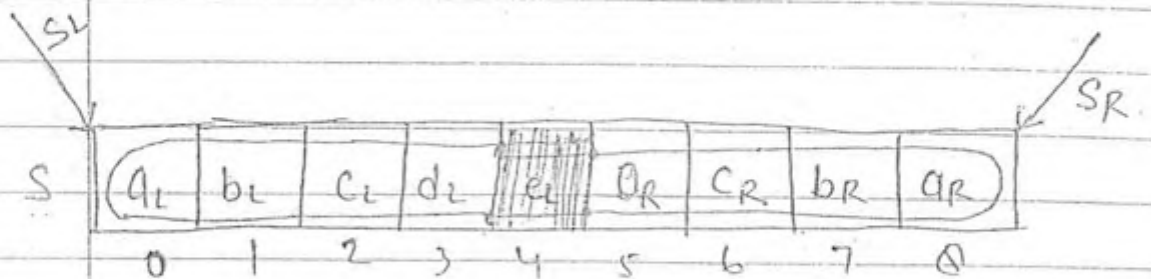
10, 20, 30, 40	50
----------------	----

5

10, 20, 30, 40, 50

Efficient way of implementing multiple stack.

$M=9$



$$T_L = -X, 0, 1, 2, 3, 4$$

$$T_R = 8, 7, 6, 5$$

$$T_L++$$

$$T_R > T_L$$

$$T_R--$$

$$S[T_L] = u$$

$$S[T_R] = v$$

~~///~~ In order to implement multiple stack efficiently ~~way~~ we are taking two stacks in which are growing in opposite order what will be the full condition.

$$a) T_L + T_R = m \times \Rightarrow T_L > R_R \times$$

$$b) T_L = T_R = \frac{m}{2} \times \text{(overlapping not possible)}$$

$$c) T_R = T_L - 1 \times \text{(because } T_L \text{ is greater)}$$

$$d) T_L = T_R + 1 \checkmark$$

POP(S, M, N, T_i)

```

{
    int y;
    if (Ti == i * (M/N) - 1)
    {
        printf("Stack is underflow");
        exit(1);
    }
    else
    {
        y = S[Ti];
        Ti--;
        return (y);
    }
}

```

Note →

Above two procedure (Push & Pop) implementing stack in single Array but it is not the efficient way implementing becoz 1 stack is full and remaining all other stack empty, it is giving error message saying that stack is overflow.

Name of array: S
 size of array: M
 No. of stack: N

DELTA	Page No. _____
Date: _____	_____

Push (S, M, N, T_i, x)

```

{
  if (  $T_i == (i+1)(\frac{M}{N}) - 1$  )
  {
    printf("stack is overflow");
    exit(1);
  }
  else
  {
     $T_i++$ ;
     $S[T_i] = x$ ;
  }
}

```

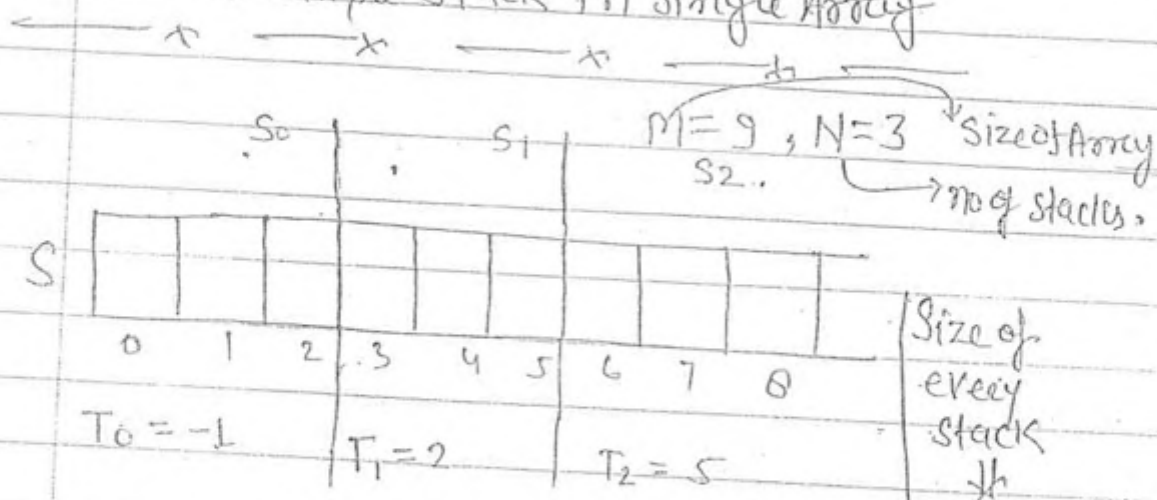
This code true for all the stack other than last stack. For the last stack push code assume as single stack push code.

POP (S, M, N, P_o)

```

{
  if (  $T_i == 1(\frac{m}{n}) - 1$  )
  else {
     $y = S[T_o]$ 
     $T_o--$ 
    return (y)
  }
}

```

Initial top of the stack.

$$0^{th} = T_0 = 0 \left(\frac{9}{3} \right) - 1 = -1$$

$$1^{st} = T_1 = 1 \left(\frac{9}{3} \right) - 1 = 2$$

$$2^{nd} = T_2 = 2 \left(\frac{9}{3} \right) - 1 = 5$$

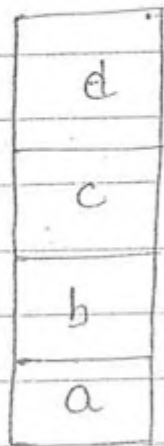
$$i^{th} = T_i = i \left(\frac{m}{n} \right) - 1 =$$

Push (S, M, N, T_0, u)

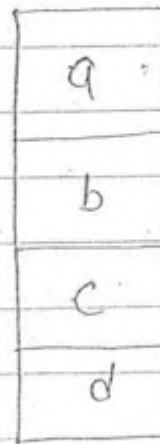
```

{
     $T_0 = (T_0 + 1) \left( \frac{m}{n} \right) - 1$ 
    {
         $T_0++$ ;
         $S[T_0] = u$ 
    }
}
    
```

Implementing Queue with Stack.



S1



S2

Queue:

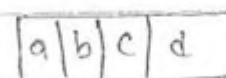
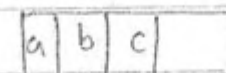
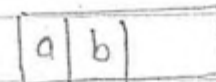
Insertion

Push all the element into stack S1

Deletion

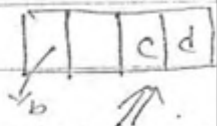
① Pop all the element from S1 and push into S2

② Pop elements from S2



Queue

Insertion ✓
Deletion ✓



↪ a

#

Main()

```

{
    char p[20];
    char s = "String";
    int length = strlen(s);
    for (i=0; i < length, i++)
    {
        p[i] = s[length-i];
    }
    printf("%s", p);
}

```

s
2000
3000

length
6
4000

s[6-0]

s[6]

*(s+6)

2000+6

2006

111111

s[6-1]

s[5]

*(s+5)

2005

11111

no output

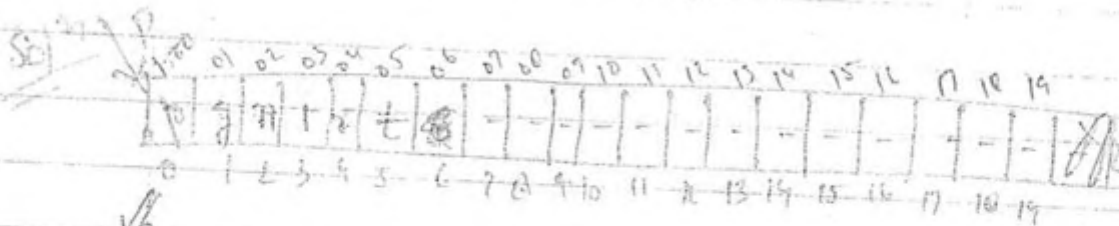
a) string

b) prints

c) no output

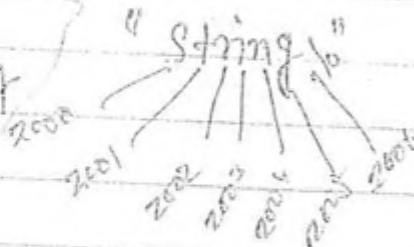
d) string

e) print



null char.

Prints start don't print anything after null



~~*~~ Main) ^{5 bytes}

{
Char a[2] = "Good"

Char *s = "Bad";

a[2] = 'w';

s[2] = 'w';

printf("a %s", a, s);

"Bad"
3000 3001 3002 3003

s
3000 4000 → 2 bytes

7 bytes

Note

- (1) Contents of the array can be change but contents of the string const. can't be change.

a[2] ✓

s[2] [result will be undefined]

(ii)

Base address of the array can't be change but s can be change.

a = 3000 ✗

s = 3000 ✓

Scope of a variable (free variable)

- ① Static Scoping (Compile time) - Global variable.
- ② Dynamic Scoping (Running time)

eg.

Main()

```
{
  int u=10;
  printf("%d", u);
}
```

O/P → 10

int u=100;

Main()

```
{
  int u=10;
  f();
}
```

f()

```
{
  int u=20;
  f1(u);
}
```

f1()

```
{
  f2();
}
```

f2(u)

{

printf("%d", u);

}

Control Stack

```
f2()
f1()
f()
Main
Global
```

Dynamic Scoping

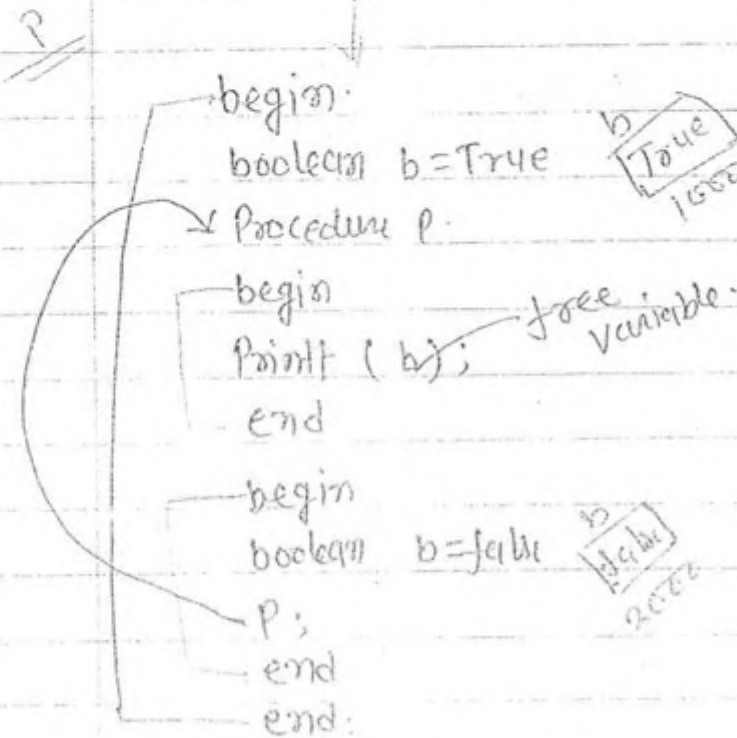
Free Variable

* In older language no one can't call the main but modern language can call the main like C.

DELTA	Page No. _____		
	Date: <table border="1"><tr><td> </td><td> </td><td> </td></tr></table>		

Dynamic Scoping.

Most recently called function. C language follow the dynamic scoping.



Static :- True.

Dynamic :- false.

Q Consider the following Program.

```
Var x: real;
Procedure show();
begin
  Print(x);
end;
```

show

$$1^{st} = 2 \times 0(u+y) + y$$

$$2^{nd} = 2 \times 1(u+y) + y$$

$$3^{rd} = 2 \times 2(u+y) + y$$

$$4^{th} = 2 \times 3(u+y) + y$$

$$5^{th} = 2 \times 4(u+y) + y$$

$$n = 2 \times (n-1)(u+y) + y$$

$$Total = ny + 2(u+y)[0+1+---(n-1)]$$

$$= ny + 2(u+y) \frac{(n-1)(n-1+1)}{2}$$

$$= ny + 2(u+y) n(n-1)$$

$$= n[y + (u+y)(n-1)]$$

$$Avg = \frac{n[y + (u+y)(n-1)]}{n}$$

$$= y + n - 1 + y - u - y$$

$$= \boxed{n(u+y) - u}$$

if $n=2$ we take only

c and b.

$$\begin{array}{r} \text{the } 3 \\ + 19 \\ \hline 22 \end{array}$$

$\Rightarrow 11$

thus $n=2$

$$u=5$$

$$y=3$$

##

$$u=2$$

$$y=4$$

$$n=5$$

$$e=4$$

$$d = 4, 2, 4, 2, 4$$

$$c = 4, 2, 4, 2, 4, 2, 4, 2, 4$$

$$b = 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4$$

4		4
2	e	2
4		4
2	d	2
4		4
	c	2
		4
	b	2
		4
	a	2

$$a = 4, 2, 4, 2, 4, 2, 4, 2, 4$$

$$2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4$$

$$e \Rightarrow 4$$

$$d \Rightarrow 16$$

$$c \Rightarrow 28$$

$$b = 40$$

$$a = 52$$

$$\underline{140}$$

$$S = 140$$

$$1 \Rightarrow \frac{140}{5} = 28$$

$$5(6) - 2$$

$$= 28$$

#P

Let S be stack of size n , starting with empty stack. Suppose we insert 1^{st} to n^{th} natural numbers in sequence and then we perform n -pop operations. Assume that push and pop operations takes x seconds each and y seconds. Elapse betⁿ the end of one such stack operation and the start of the next operation. For $m \in \mathbb{N}$ define the stack life of m as the time elapsed from the end of push(m) to the start of the pop operation that removes m from S . The avg stack life of an element of the stack is.

a) $n(x+y)$

b) $3y + 2x$

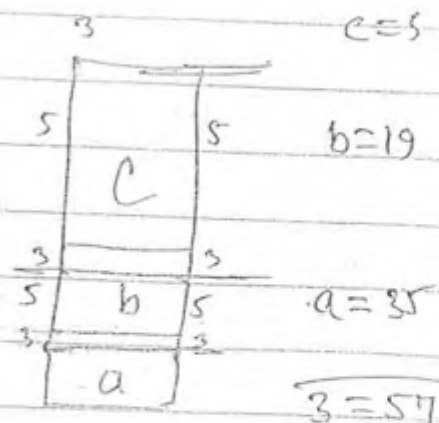
c) $n(x+y) - x$

d) $y - 2x$

Solⁿ

Push (or) Pop = 5 } 2.

$$\left. \begin{array}{l} \text{Push} - \text{Push} = 5 \\ \text{Push} - \text{Pop} = 5 \\ \text{Pop} - \text{Pop} = 5 \end{array} \right\} y.$$



$$\begin{array}{l} n=3 \\ x=5 \\ y=3 \end{array}$$

19

int Pop (s, N, Top)

```

{
    int y;
    if ( Top == -1 )
    {
        Printf ( "Stack is underflow" )
        exit (1);
    }
    else
    {
        {
            y = s[Top]
            Top--;
        }
        y = s[Top--]; ✓
        y = s[--Top]; X
        return (y)
    }
}

```

Is Empty() →

0(1) →

Observation (PUSH)

observation (POP)

- | | |
|-------------------------------|-------------------------------------|
| (i) element x to be passed. | - No need to Pass x |
| (ii) Increment top. | - Delete |
| (iii) Insert the element. | - Decrement |
| (iv) no need to return y . | - need of returning element y ... |

Name of Stack
Size of Stack

element to be inserted.

DELTA	Page No.

Void Push (S, N, Top, u)

```

{
    if (Top == N-1)
    {
        Printf ("Stack is overflow");
        exit (1);
    }
    else
    {
        Top++;
        S[Top] = u;
    }
}

```

Isfull();

abnormal termination

$\theta(1)$
constant time

$S[++Top] = u$ ✓
 $S[Top++] = u$ X

Pop (Top)

```

{
    if (Top == -1)
    {
        empty;
    }
    else
    {
        y = S[Top];
        Top--;
        return (y);
    }
}

```

S		N=6
5		
4		
3	d	
2	c	
1	b	
0	a	

- (i)
- (ii)
- (iii)
- (iv)

STACK

Defination :- One side open, another side closed.
:- Last in first out [LIFO].
:- Top is a variable which contain position of the newly inserted element.

Operations :- ADT of Stack (what opⁿ can perform)

- (i) Push (stack, integer) = stack.
- (ii) POP (Stack) = integer.
- (iii) Isempty (stack) = Boolean;
- (iv) Istull (stack) = Boolean.

Push()

When stack is empty

Top = -1 (initially)

Push(a)

```

{
  {
    Top++;
    S[Top] = a;
  }
}

```

Top = N-1.

S N=5

4	e
3	d
2	c
1	b
0	a

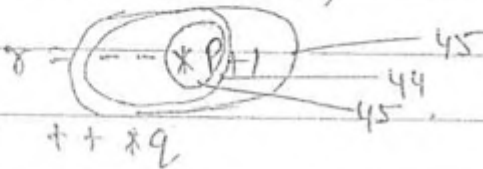
P int main()

```
{
    int m=44;
    int *p = &m;
```

int &r = m; (r address is same as m address)

int n = (*p)++; means r is alias to m.

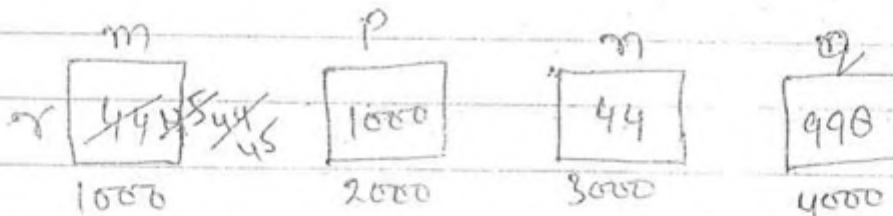
```
int *q = p-1;
```



```
printf("m=%d, n=%d, r=%d", m, n, r);
```

45 44 45

Solⁿ



Q/P 2.

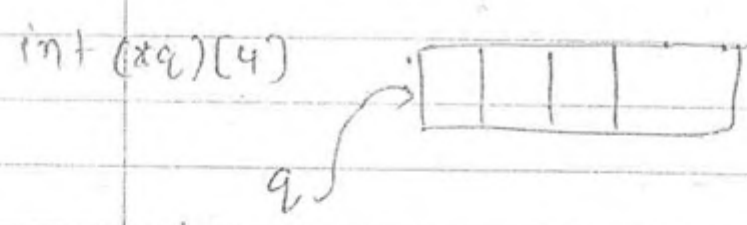
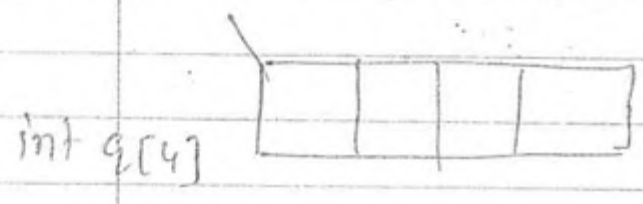
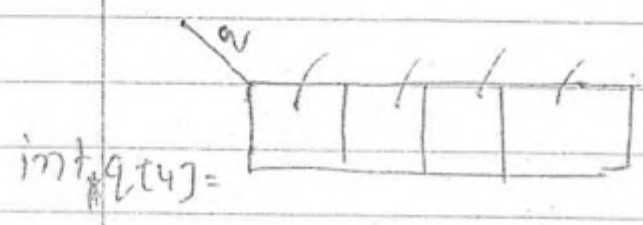
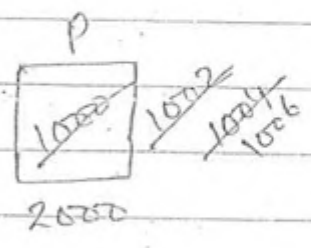
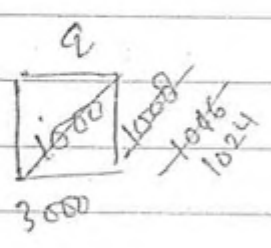
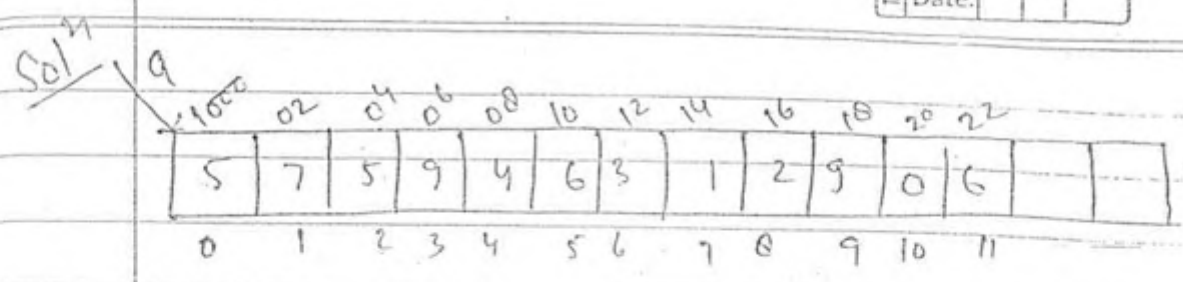
a) 44, 46, 45

b) 45, 44, 45

c) 46, 44, 46

d) 46, 45, 46

2, 9, 0, 6?



O/P	q+1	p	q.
i = 0	1000	1000	1000
i = 1	1002	1002	1000
i = 2	1016	1004	1016
i = 3	1024	1006	1024

main()

{ int a[3][3] = { 5, 7, 5, 9, 4, 6, 3, 1, 2, 9, 0, 6 };

int *p;

int (*q) [4];

p = a;

q = a;

for (i = 0; i < 3; i++)

{
Printt (" %d %d %d ", a[i], p[i], q[i])

p++;

q++;

}

a

	00	01	02	03
00	5	7	5	9
01	4	6	3	1
02	2	9	0	6

a[0] = 1000

a[1] = 1008

a[2] = 1016

Sol

in

int

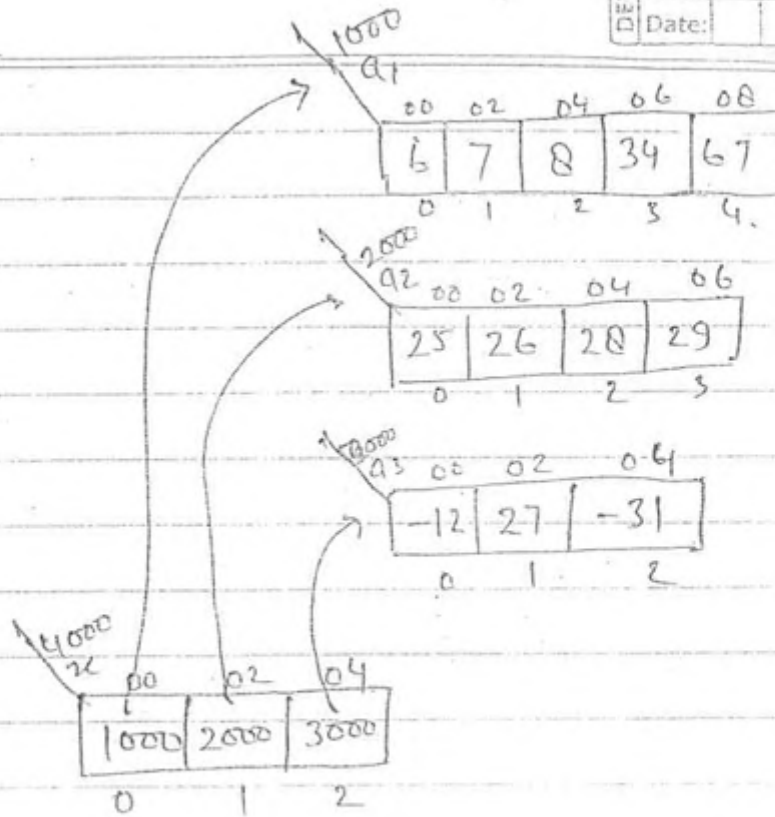
int

0

i = 1

i = 2

i = 3



$$*(*(a+0)+2) = a[0][2]$$

`fun (int *a)` 4000

a
4002
Save

$\Rightarrow a[0][2] = *(a+0)+2$

$\xrightarrow{4000}$

$\xrightarrow{1000+4}$

$\xrightarrow{1004}$

$\Rightarrow *(a[2])$

$\xrightarrow{*(a+2)}$

$\xrightarrow{*(4000+4)}$

$\xrightarrow{*3000}$

$= -12$

$\Rightarrow *++a[0]$

$\xrightarrow{*(a+0)}$

$\xrightarrow{(4000+0)}$

$\xrightarrow{4000}$

$\xrightarrow{++(1000)}$

$\Rightarrow *++a[0]$

$\xrightarrow{4002[0]}$

$\xrightarrow{*(4002+0)}$

$\xrightarrow{*4002}$

$\xrightarrow{*(4002)}$

#

int a1[] = {6, 7, 8, 34, 67}

int a2[] = {25, 26, 28, 29}

int a3[] = {-12, 27, -51}

int x[] = {a1, a2, a3}

main()

{
 fun(x);
}

fun(int x)

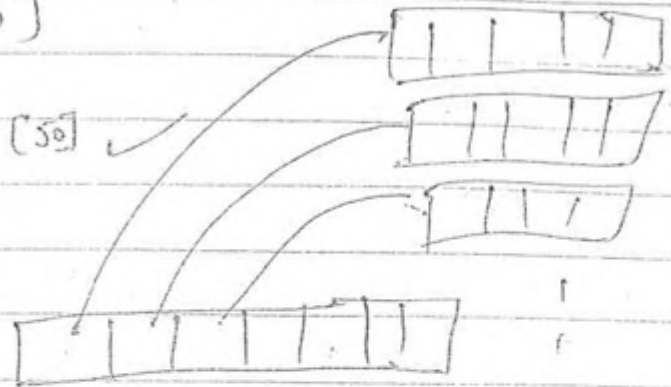
{
 printf("%d", a[0][2]); $\Rightarrow 8$
 printf("%d", x[a[2]]); $\Rightarrow -12$
 printf("%d", x + a[0]); $\Rightarrow 7$
 printf("%d", x[1+a][0]); $\Rightarrow 25$
}

9
4000
5000

~~int * p[10]~~

int p[10][50]

Solⁿ

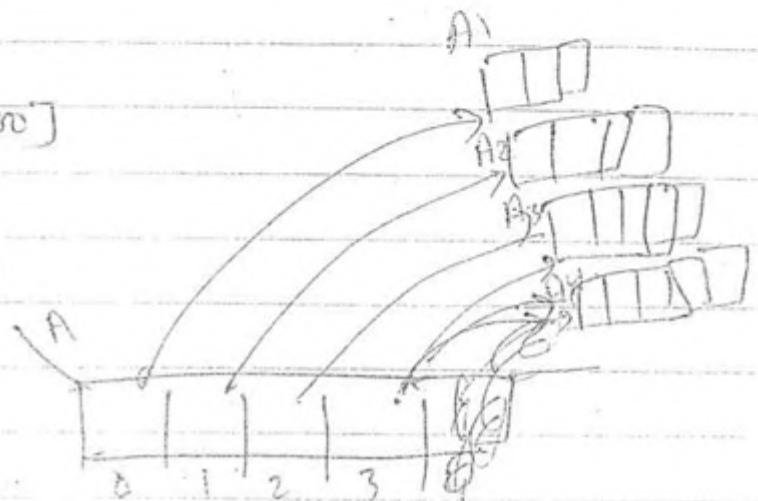


~~int~~

A[5][100]

↓

* A[5]



int A1[3], A2[4], A3[5], A4[6],

int * A[5] = {A1, A2, A3, A4, A5}.

$$x(p+0)$$

$$x(p[0] + 0)$$

$$\downarrow$$

$$1$$

$$*(p+i) = p[i]$$

Soln

i	j	$*(*(p+i) + j)$
0	0	1
	1	2
	2	3
1	0	4
	1	5
	2	6
2	0	7
	1	8
	2	9

In this Problem
P is 1D-Array
but it ~~give~~ ^{become} like
2-D Array
becoz
it gives
array.

Which of the following is true for the following declaration.

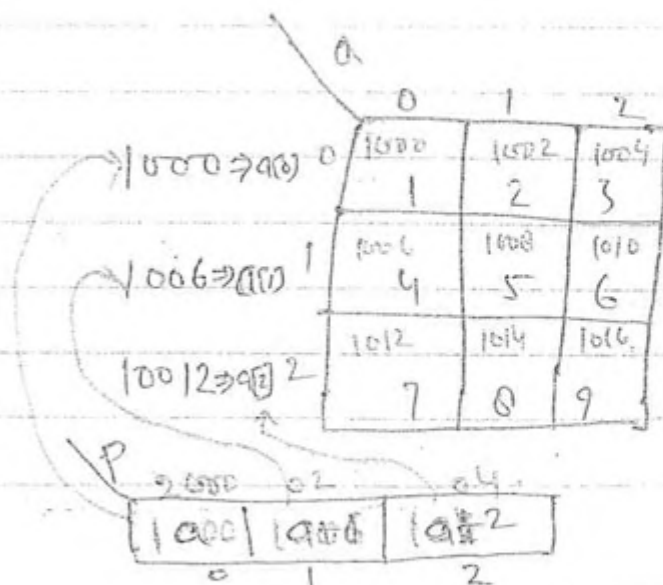
int A[10][10], *B[10]

- a) B[5] = address ✓
- b) B[5][5] = 10 ✓
- c) A[5] = address ✗
- d) A[5][6] = 10 ✓

n | ptr
 $n+1 \Rightarrow 1006$ | $ptr+1 \Rightarrow 1002$

```
#P Main()
{
    int a[3][3] = {1, 2, 3, 4, 5, 6, 7, 8, 9};
    int *p[3] = {a, a+1, a+2};
    for (i=0; i<=2; i++)
    {
        for (j=0; j<=2; j++)
        {
            Printf("%d", * (p[i]+j));
        }
    }
}
```

O/P $\Rightarrow$ 1, 2, 3, 4, 5,
6, 7, 8, 9.



#

main

{

int n[3][3] = {2, 4, 3, 6, 8, 5, 5, 5, 1};

int * ptr;

ptr = n;

printf("%d", n[2]); $\Rightarrow 1012$

printf("%d", ptr[2]); $\Rightarrow 3$

printf("%d", *(ptr+2)); $\Rightarrow 3$

Soln

		n			
		0	1	2	
1000	0	2	4	3	$\Rightarrow n[0] \Rightarrow 1000$
1006	1	6	8	5	$\Rightarrow n[1] \Rightarrow 1006$
1012	2	3	5	1	$\Rightarrow n[2] \Rightarrow 1012$

ptr

1000

* (ptr + 2)

1000 + 2

* (1004)

3

$$\frac{(a[4] + 1) + 1}{1016}$$

1016

1020

$$* \left(\frac{(a[1] + 1) + 1}{1016} \right)$$

1016

$$\frac{a[2] + 1}{1024 + 4} \rightarrow (1028)$$

$$\# * (* (a + 0) + 1) + 1$$

$$\# * [a + 0] = * a$$

$$* * a = a[0][0]$$

$$* (* (a + 0) + 0)$$

$$* (a[0] + 0)$$

In 3-D Array first * means array second * means rows and 3rd * (star) means column is selected.

3-D Array

Diagram illustrating a 3-D array structure with indices a , $a[0]$, and $a[1]$.

	$a[0]$	$a[1]$
$a[0]$ (1000)	2	4
$a[1]$ (1004)	7	8
$a[2]$ (1008)	3	4
$a[3]$ (1012)	2	2
$a[4]$ (1016)	2	3
$a[5]$ (1020)	3	4

$$a = 1000$$

$$a+1 = 1012$$

$$a[1] = 1012 \text{ means it is selected.}$$

not selected

$$(a+1)+1 \Rightarrow 1024$$

$$a[1]+1 \Rightarrow 1016$$

$$\frac{* (a[0] + 1) + 1}{20 + 1} \cdot a[j][j]$$

$$21$$

b[j]

$$* (b + j)$$

$$* (a[j] + j)$$

$$* (* (a + j) + j)$$

$$* (a[0] + 0) \Rightarrow a[0][0]$$

$$* * a[0][0]$$

Main ()
{

int a[2][3][2] = { 2, 4, 7, 8, 3, 4, 22, 3, 5, 4 }.

printf ("%u %u %u %d", ¹⁰⁰⁰a, ¹⁰⁰⁰*a, ¹⁰⁰⁰***a, ²***a);

printf ("%u %u %u %d", ¹⁰⁰²a+1, ¹⁰¹²* (a+1), ¹⁰¹²* (* (a+1)),
²* (* (* (a+1))));

printf ("%u %u %d", ¹⁰⁰⁴*a+1, ¹⁰⁰²* *a+1, ³* * *a+1)
}

$a+1$ $* (a+1)$
 $\Downarrow$ $\Downarrow$
 1006 1006
 both are same but ~~$x(a+1)$~~ $x(a+1)$
 means that is selected.

$(a+1)+1$ $\overset{1006}{* (a+1)+1}$
 $\Downarrow$ $\Downarrow$
 $a+2$ 1008
 $\Downarrow$
 1012

In 2D Array two store are compulsory.
 where 1st $x(\text{store})$ indicate row address.

$(a+2)+2$ $\Downarrow$ $1012+12$ $\Rightarrow 1024$	$* (a+2)+1$ $\underline{1012+2}$ 1014	$* (x(a+2)+1)+1$ $\underline{1012+2}$ $\underline{80+1}$ 81
--	---	--

a
 $a+0$
 $x(a+0) \equiv a[0]$

} 1000

* Belⁿ two pointers addition, division and multiplied is not possible becoz there is no meaning.

p

Main()

{

int a[3][3];

Print(" %d ", ((a == *a) && (*a == a[0])));

}

o/p = 1

a
1000

	0	1	2
0	00	02	04
	10	20	30
1	06	08	10
	40	50	60
	12	14	16
2	70	80	90

⇒ a[0] ⇒ 1000

⇒ a[1] ⇒ 1006

⇒ a[2] ⇒ 1012

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

a = 1000

a+0 = 1000

a+1 = 1006

a+2 = 1012

means In 2D array 0 rows skip.

" " " " 1 row skip.

a[0][0] = 10

a[a][0] ⇒ *(a+0)[0]

((a+0)+0)

a+1

Note

④ for unary operator associativity is from right to left.

$++$ & x both are having same associativity is right to left.

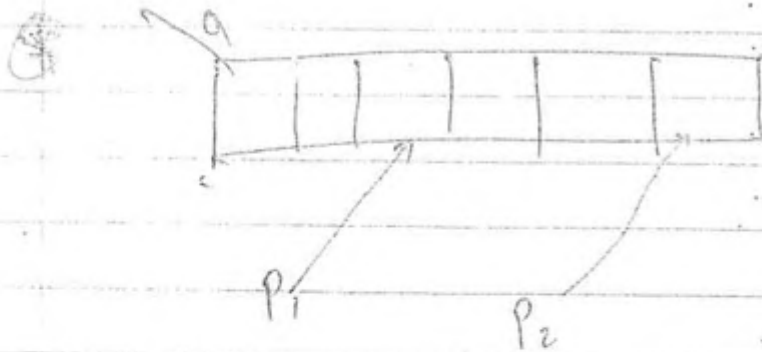
$x \text{ } ++$ in this $++$ is completed first.

Note

⑤ For the pointer variable we can add integer constant.

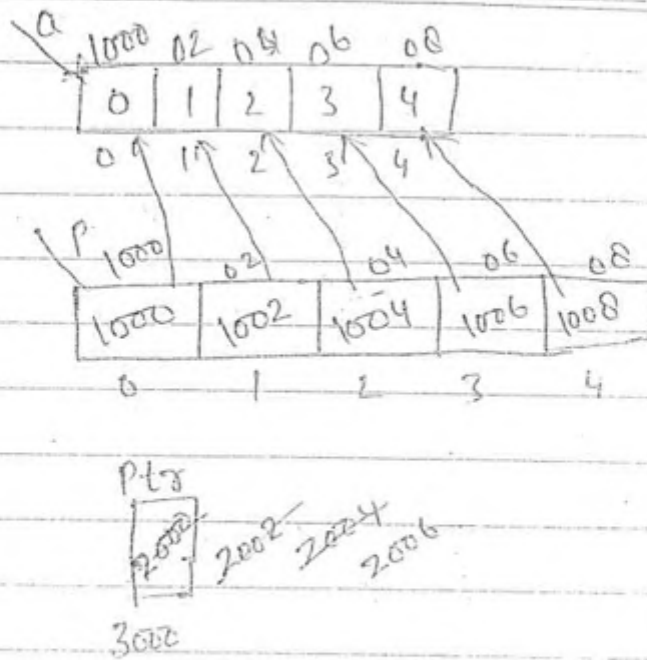
$p = p + i$ indicates p will point to after i elements from current place.

⑥ From the pointer variable we can subtract integer constant.



$$p_2 - p_1 \quad (p_2 \geq p_1)$$

Subtraction betⁿ two pointers is allowed, but the condition is both with pointer will same array.



#P

Main

```

{
    int arr[] = {0, 1, 2, 3, 4};
    int *p[] = {0, a+1, a+2, a+3, a+4};

    int **ptr = p;

    ptr++;

    printf("%d %d %d", ptr-p, *ptr-a, **ptr);
    *ptr++;

    printf("%d %d %d", ptr-p, *ptr-a, **ptr);
    *(++ptr);

    printf("%d %d %d", ptr-p, *ptr-a, **ptr);
    ++(*ptr);

    printf("%d %d %d", ptr-p, *ptr-a, **ptr);
}
  
```

Diagram illustrating pointer arithmetic and memory addresses for the code above:

- For `ptr-p`:
 - Initial: $\frac{1}{2} \frac{2008-2000}{2008-2000}$
 - After `ptr++`: $\frac{2}{4} \frac{2004-2000}{2004-2000}$
 - After `*(++ptr)`: $\frac{3}{6} \frac{2006-2000}{2006-2000}$
 - After `++(*ptr)`: $\frac{4}{8} \frac{2008-2000}{2008-2000}$
- For `*ptr-a`:
 - Initial: $\frac{1}{2} \frac{1002-1000}{1002-1000}$
 - After `ptr++`: $\frac{2}{4} \frac{1004-1000}{1004-1000}$
 - After `*(++ptr)`: $\frac{3}{6} \frac{1006-1000}{1006-1000}$
 - After `++(*ptr)`: $\frac{4}{8} \frac{1008-1000}{1008-1000}$
- For `**ptr`:
 - Initial: $\frac{1}{2} \frac{1002-1000}{1002-1000}$
 - After `ptr++`: $\frac{2}{4} \frac{1004-1000}{1004-1000}$
 - After `*(++ptr)`: $\frac{3}{6} \frac{1006-1000}{1006-1000}$
 - After `++(*ptr)`: $\frac{4}{8} \frac{1008-1000}{1008-1000}$

#

Main ()

```

{
    char a[] = "Visual C++";
    char *b = "Visual C++";
    printf ("%d %d", sizeof(a), sizeof(b));
    printf ("%d %d", sizeof(*a), sizeof(*b));
}

```

Diagram illustrating memory sizes:

- `a` (array) is 12 bytes (11 characters + 1 null terminator).
- `b` (pointer) is 4 bytes.
- `*a` (character) is 1 byte.
- `*b` (character) is 1 byte.

#P

Main ()

```

{
    int a[] = {0, 1, 2, 3, 4};
    int xP[] = {a, a+1, a+2, a+3, a+4};
    int **ptr = P;
    printf ("%u %d", a, *a);
    printf ("%u %u %d", P, xP, **P);
    printf ("%u %u %d", ptr, *ptr, **ptr);
}

```

Diagram illustrating memory addresses and values:

- `a` (array) starts at address 2500, contains values 0, 1, 2, 3, 4.
- `xP` (array of pointers) starts at address 1000, contains addresses 2500, 2504, 2508, 2512, 2516.
- `ptr` (pointer to pointer) starts at address 0, contains address 1000.

Date:

Main ()

2

2

Char * str [] = { "Frogs", "Dogs", "Nuts", "Oie",
"croak", "break" }

Print (" %d %d", Size of (str), Size of (str[0]));

3.

$$\begin{array}{r} \downarrow \\ 6 \times 2 \\ \downarrow \\ 12 \text{ B.} \end{array}$$

100
11
2 B.



```
int a; 
```

$$171 - 9[5] =$$

10	20	30	40	50
0	1	2	3	4

size of $(\mathcal{A})^2 = 10$

if float
then

Size of $(9) = 20$

$$\ln p_a(s) =$$

Size of $(a) = 10$

if float
fn

Size of (4) = 10

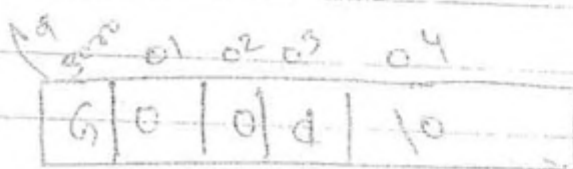
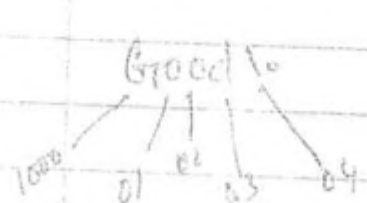
#P

Main()

```
{
  char *p = "Good";
  char a[] = "Good";
```

P
1000
2000

```
  printf("%d %d %d", sizeof(p), sizeof(*p),  
        strlen(p));  
  printf("%d %d", sizeof(a), strlen(a));  
}
```

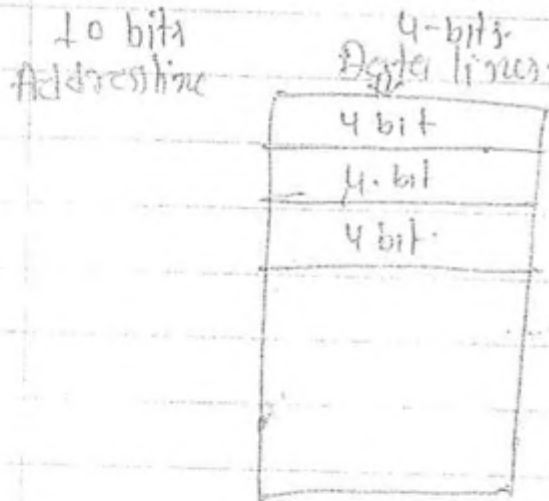


strlen("Good") = 4
strlen(p) = 4

int a = 10 a
10
 1000

int *b
 b = &a b
1000
 2000

char sum = 'c'; sum
c
 1000
 char *b;
 b = &sum b
1000
 2000



$\Rightarrow 2^{10} \times 4 = 4 \text{ KB}$

my node *p; 16 B.
 Size of (p) = 2 B
 Size of (*p) = 16 B

void *p not specified
 Size of (p) = 2 B
 Size of (*p) = 16 B
 / error

Call by need

* integer $i = 100$, $J = 5$;

Procedure (integer n)

Point f. ^{its} $(2+5) \Rightarrow 110$
$$l = 2\pi,$$
$$\underline{J = 20 ;}$$

Printf (u); $\Rightarrow$ 105

मा'त ()

$p(2.15)$

3

$\begin{array}{c} \text{I} \\ \text{200} \\ \text{100} \end{array}$
 $\begin{array}{c} \text{J} \\ \text{200} \end{array}$

exactly calling
same
but it will evaluate
the expression
only once.

Recently modified
thing will not be
effected.

main ()

Assume:

Address line = 16 bit

Chem BP;

Print: (" %d %d", Size of (P), Size of (xP))

Proof x b

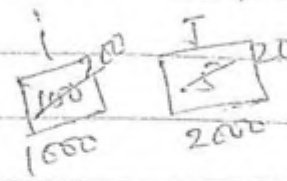
Size of (b) = 28

$$\sin^2 \theta \left(\frac{x}{a} \right) = \frac{4}{5}$$

Dr. A. B. G. H.

Consider the following Program. Call by name
Passing Parameters

integer i=100, j=5
Procedure (integer x)
begin



 i+j
 Printf (x+j); $\Rightarrow 110$
 i = 200;
 j = 20;
 Printf (x); $\Rightarrow 220$
end;

int i=100	i=100
110	j=100
120	110
	105

Main()

Call by Name :- 110, 220

{
 P(i+j)
}

integer i=100, j=5
Procedure (integer x)
begin



 i+j
 Print (x+j) = 110
 i=200;
 j=20;
 Printf (x); $\Rightarrow 220$
end;

Main()

{
 P(i+j)
}

$$a = 1000, n = 6$$

$$\begin{array}{l} \textcircled{1} \\ \textcircled{2} \end{array} \begin{array}{c} a \\ 12 + (1002, 5) \end{array}$$

$$\begin{array}{l} \textcircled{1} \\ \textcircled{2} \\ \textcircled{3} \end{array} \begin{array}{c} a \quad n \\ 7 - f(1004, 4) \end{array}$$

$$\begin{array}{l} \textcircled{1} \\ \textcircled{2} \\ \textcircled{3} \end{array} \begin{array}{c} a \quad n \\ 13 - f(1006, 3) \end{array}$$

$$\begin{array}{l} \textcircled{1} \\ \textcircled{2} \end{array} \begin{array}{c} a \quad n \\ 4 + f(1008, 2) \end{array}$$

$$\begin{array}{l} \textcircled{1} \\ \textcircled{2} \\ \textcircled{3} \end{array} \begin{array}{c} 11 - f(1010, 1) \end{array}$$

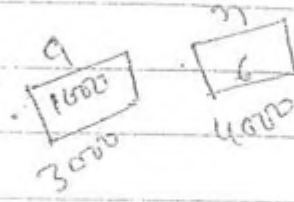
$$\begin{array}{l} \textcircled{1} \\ \textcircled{2} \end{array} \begin{array}{c} 6 + (1012, 0) \end{array}$$

ing C

What is the value printed by the following C. Program

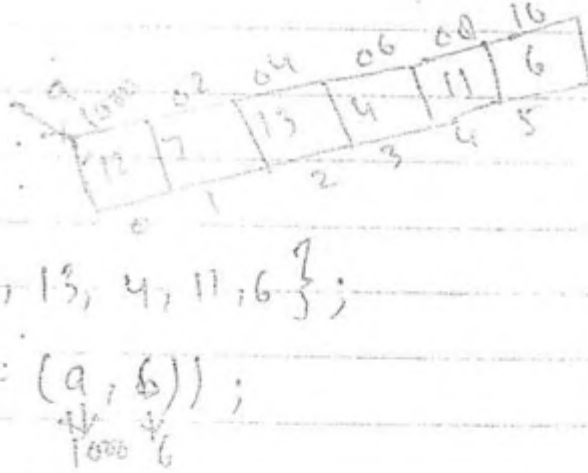
```
#include (stdio.h)

int f (int *a, int n)
{
    ① if ( n <= 0 ) return 0;
    else
    ② if ( *a % 2 == 0 )
        return *a + f(a+1, n-1);
    else
    ③ return *a - f(a+1, n-1)
}
```



main()

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

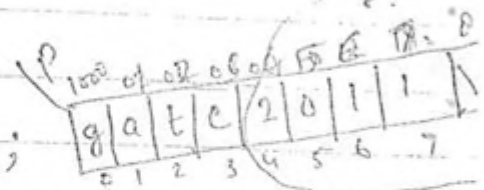


- a) -9 b) 5
c) 15 d) 19

Q. Consider the following C program.

Main

```
{
    char P[] = "gate2011";
    printf("%s", P + P[3] - P[1]);
}
```



- a) gate2011 b) 2011
c) e2011 d) 011

$$P + P[3] - P[1]$$

$$1000 + e - a$$

$$1000 - 1001 = -1$$

$$1000 + 4$$

$$1004$$

Q. Consider the following C program.

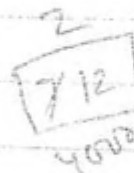
int x;

main

```
{
    x = 5;
    P(x);
    printf("%d", x);
}
```

void P(int z)

```
{
    z += x;
    printf("%d", z);
}
```



Void P(int *y)

{

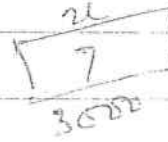
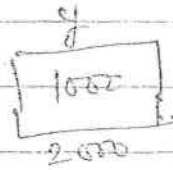
int x = *y + 2;

Q(x);

*y = x - 1;

printf("%d", x);

}



O/p - ?

a) 12, 7, 5

b) 12, 7, 6

c) 14, 6, 6

d) 7, 6, 6

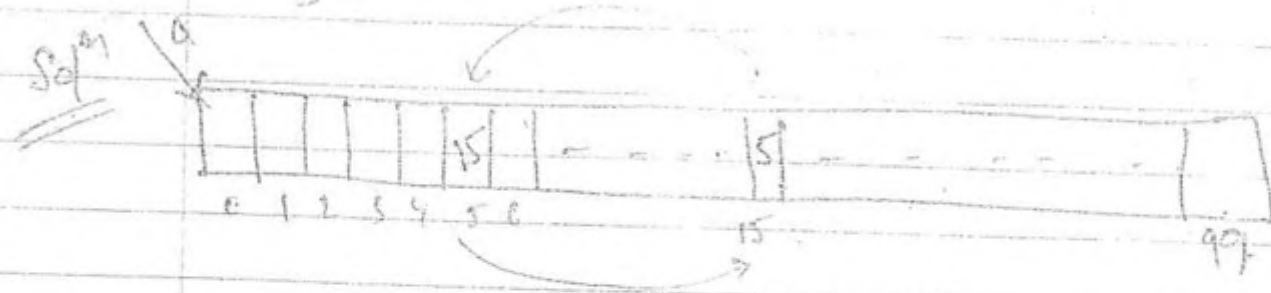
#P Main () Call by name

```
{
    int i=5, A[100];
    A[i]=15;
    printf("%d %d", i, A[i]);
    swap(i, A[i]);
    printf("%d %d", i, A[i]);
}
```



Swap (int c, int d)
 i A[i]

```
{
    int t;
    t=c;   ⇒ t=i;
    c=d;   ⇒ i=A[i];
    d=t;   ⇒ A[i]=t;
}
```



Call by name
o/p = 5, 15
 15, 5

Strict evolution

Call by value -
Call by reference -
Copy - restore -

Lazy evolution (non strict)

Call by Name -
Call by need -
Call by Text -

Note - In Lazy evolution category we will pause the expression without execution which will lead to evaluating the same expression many times.

In the above program c(m) evaluated 5 times,

```
# swap (int c, int d)
{
    c[m], m
```

d = d + 5 ⇒ m = m + 5

d = d + 100 ⇒ m = m + 100

}

In this
0 time
evaluated

Name Conflict :-

When existing local variable name is as same as formal parameter then that problem is known as name conflict.

Existing Local variable name is replace by other name.

```

//
main ()
{
    int c[10] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
    int m = 2;
    printf ("%d %d", c[m], m);
    swap (c[m], m);
    printf ("%d %d", c[m], m);
}

swap (int c, int d)
{
    // c[m]    m
    c = c + d  => c[m] = c[m] + m
    d = c - d  => m = c[m] - m
    c = c - d  => c[m] = c[m] - m
}

// 22
// 22
    
```

evaluated first

P

int a=100
main()



```
{
    int a=10, b=20;
    printf("%d %d", a, b);
    swap(a, b);
    printf("%d %d", a, b);
}
```



```
{
    int t, a=5000;
    t=c; a=t;
    c=d; a=b;
    d=t; b=t;
}
```



Name conflict

↓
new local variable will
be proceed further
means existing local
variable will be removed

C/P ⇒ 10, 20
20, 10

Q //

int a=100
Main()

a
100
1000

int a=10, b=20

a
10
2000

b
20
10

printf("%d %d", a, b);

swap(a, b);

printf("%d %d", a, b);

swap(int a, int b)

int t;

t
10

t = a; $\Rightarrow$ t = 10

a=20
b=10

a = b; $\Rightarrow$ a = 10

b = t; $\Rightarrow$ b = 10

a++

a=21, b=10

O/P

Value	Name
10, 20	10, 20
10, 10	21, 10

Call by Name

main()

int a=10, b=20;

printf("%d %d", a, b);

Swap(a, b)

printf("%d %d", a, b);

}

Swap(int a, int b)

{
int t;

t = a; $\Rightarrow$ t = 10;

a = b; $\Rightarrow$ a = 20;

b = t; $\Rightarrow$ b = 10;

}

O/P $\Rightarrow$ 10, 20 (without swapping)

$\Rightarrow$ 20, 10 (after swapping)

Q

What is the value printed by the following C Program,

```
int f(int x, int xpy, int xppz)
{
    int y, z;
    xppz = 1;
    z = xppz;
    xpy = z;
    y = xpy;
    x = 3;
    return (x + y + z);
}
```

x	xpy	xppz
4	1000	2000
4000	5000	6000

y	z
4	5
7000	8000

void main()

```
{
    int c, b, a;
    c = 4, b = 2 * c, a = 2 * b;
    printf("%d", f(c, b, a));
}
```

c	b	a
4	8	16
1000	1000	2000

4 1000 2000

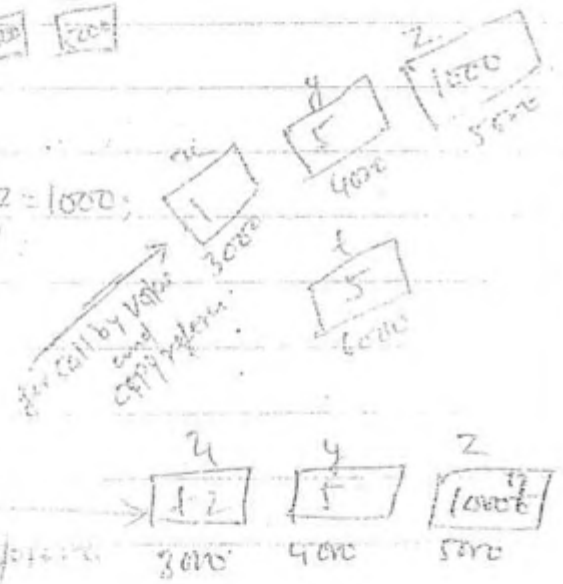
- a) 10 b) 19
c) 21 d) 22

Note--

Call by copy will give different answer comparing with call by reference. If the parameter passed to the funⁿ follows aliasing.

Procedure C(int a, int b, int c)
begin (int *a, int *b, int *c)
*a = 2
*c = *a + b;
end.

begin
int x=1, y=5, z=1000;
P(x, x*y, z);
Print(x, y);
end.



e/p - Call by value :- 1, 5
Copy-reference :- 2, 5
Reference :- 2, 5

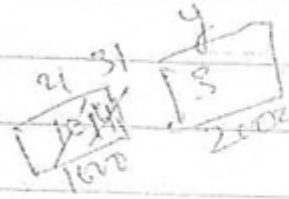
Q. Consider the following C Program.

Program P1 (C)

```

{
    int x = 10, y = 3;
    fun1 (x, y);
    Print (x, y);
}

fun1 (int *x, int *y, int *z)
{
    *x = *y + 4;
    *z = *x + *y + *z;
}
    
```



Aliasing
 two variables
 having same
 address



Call by Value: 10, 3
 Copy Restore: 14/27, 3
 Reference: 31, 3

aliasing

Start execution

9

main()

{

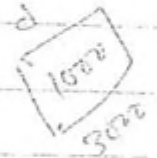
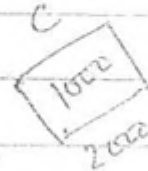
int a=5



Test (a, a)

printf (a);

Test (int xc, int xd)

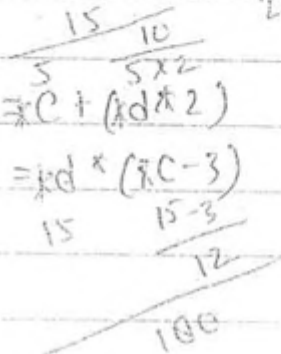


{

c = d * 2; $\Rightarrow c = c + (d * 2)$

d = c - 3; $\Rightarrow d = d * (c - 3)$

}



Q Consider the following Program.

```

Main()
{
    int a=5;
    Test (&a, a);
    Printf(a);
}

Test (int c, int d)
{
    c = d * 2;  ⇒ c = c + (d * 2)
    d = c - 3;  ⇒ d = d * (c - 3)
}
    
```

a
5
1000

c
15
2000

d
60
3000

Q
Call by Value : 5
Copy - make
Call by Reference

15/60
1800

d = 4
a[++i] = ++i
a[5] = 6

i = 4
j = ++i
a[++i] = 5

or
a[6] = 5
global copy

aligning

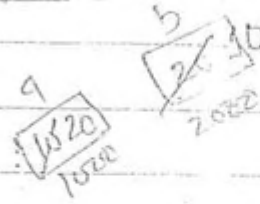
i = 4
j = ++i
a[j] = ++i
a[5] = 6

Side Effect

3. Call by Copy - Restore (Indirectly modify)

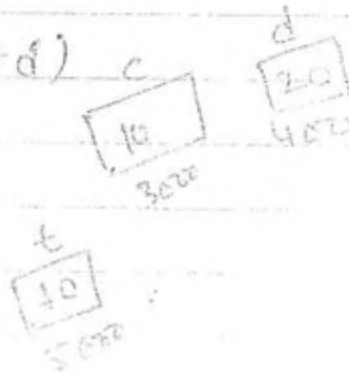
Main ()

```
{
    int a=10, b=20;
    printf ("%d %d", a, b);
    swap (a, b);
    printf ("%d %d", a, b);
}
```



swap (int c, int d)

```
{
    int t;
    t = c;
    c = d;
    d = t;
}
```



O/P - 10, 20
20, 10

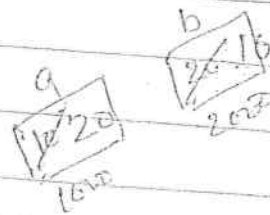
Q

Main()

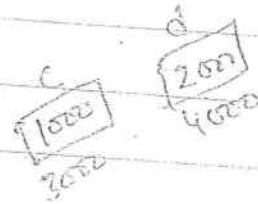
```

{
    int a=10, b=20;
    printf("%d %d", a, b);
    swap(a, b);
    printf("%d %d", a, b);
}

```



swap(int *c, int *d)



```

{
    exchange(c, d);
}

```

exchange(int *e, int *f)



```

{
    int t;
    t = *e;
    *e = *f;
    *f = t;
}

```



C/P → 10, 20
20, 10

* C language by default follows call by value

C II supports both call by value
call by reference;

Page No.	
Date:	

Swap without temp variable.

$$a = 10, b = 20$$

$$\begin{array}{r} 30 \\ 20 \end{array} \quad \begin{array}{r} 10 \\ 20 \end{array}$$

$$a = a + b$$

$$b = a - b$$

$$a = a - b$$

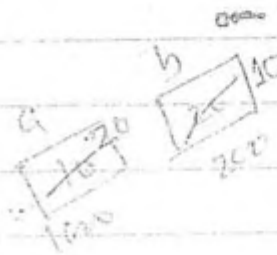
Call by Reference (Directly changes in original)

Main ()

```
{
  int a = 10, b = 20;
  printf("%d %d", a, b);
  swap(a, b);
  printf("%d %d", a, b);
}
```

Swap (int *a, int *b)

```
{
  int t;
  t = *a;
  *a = *b;
  *b = t;
}
```



O/P - 10, 20
20, 10

in copy

only
args are
passed only
re.

without swap
swap!

#P

Write a recursive C-Program to find the max^m element of the given array.

Ans

10	20	30	40	50
1	2	3	4	5

10, 20, 30, 40, 50
i j

Print max (a, i, j)

```

{
    if (i == j)
    {
        printf ("%d" a[i])
        return;
    }
    else
    {
        Print max
        if (a[i] < a[j])
            Print max (a, i+1, j)
        else
            Print max (a, i, j-1)
    }
}

```


eg

50	10	170	110	60
1	2	3	4	5
i	i	$i=j$	j	j

Types of Recursion

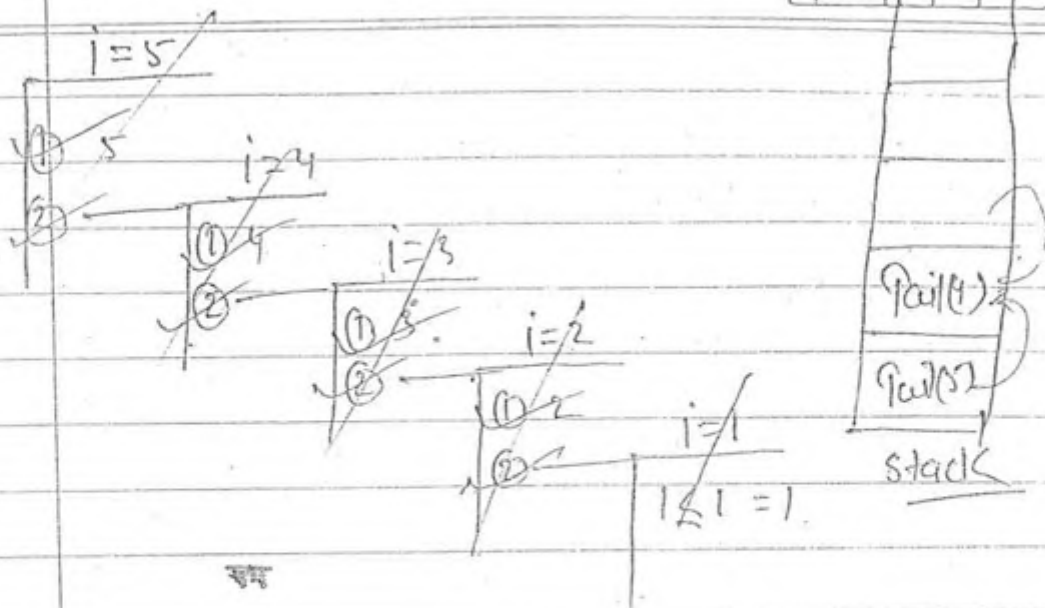
- (1) Tail-Recursion.
- (2) Non-Tail-Recursion.
- (3) Nested Recursion
- (4) Indirect Recursion.

Tail-Recursion

```

Tail (int i)           i=5
{
    if (i <= 1) return
    else
    {
        ① printf ("%d", i);
        ② Tail (i-1);
    }
}
    
```

c/p = 5, 4, 3, 2, 1.



⇒ In the given recursive program if recursive call is there at the end of the program then that program is called Tail recursive Program.

⇒ Tail recursive Program is not advisable becoz unnecessarily we are wasting stack space.

⇒ The advantage with the Tail recursive Program is we can write easily non-recursive Program ^{equivalent} for the given tail recursive program with the help of loop.

Non-Tail Recursive

Non Tail (int i)

```
{
  if (i <= 1) return;
  else
```

```
{
```

```
  NonTail (i-1)
```

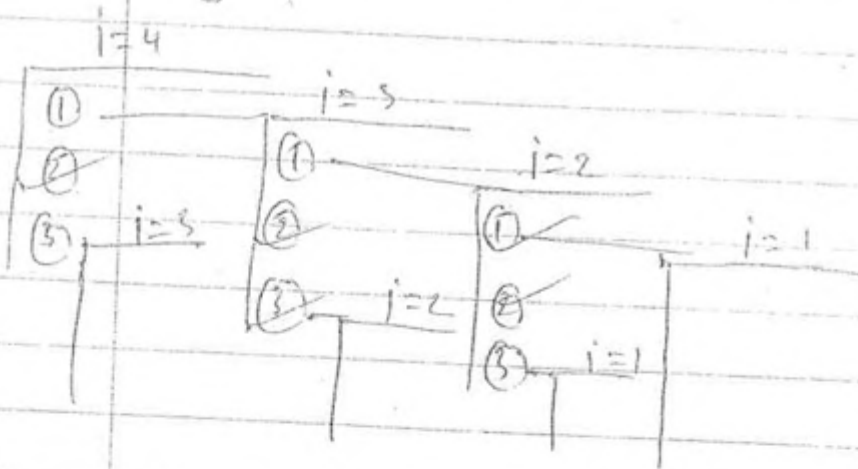
```
  printf ("%d ", i)
```

```
  NonTail (i-1)
```

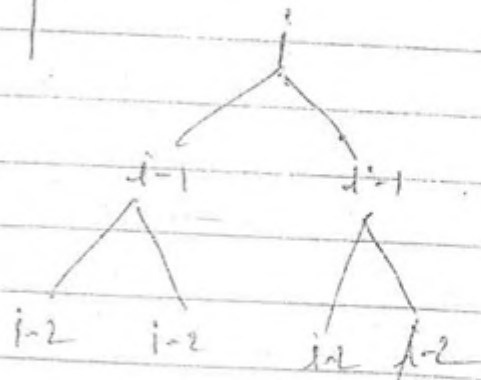
```
}
```

```
}
```

i = 4



O/P = 2, 3, 2, 4, 2, 3, 2



⇒ In the given recursive program after recursive call some stmt. are there then that recursion is called Non-Tail Recursion.

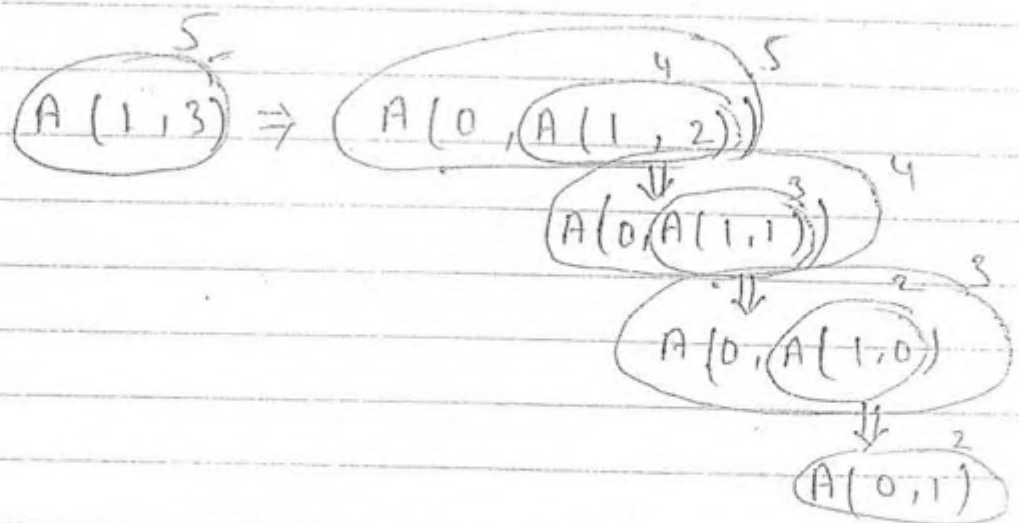
Note

we are utilizing the space efficiently and but it is difficult to write equivalent to write with equivalent non-recursive program.

Nested Recursion

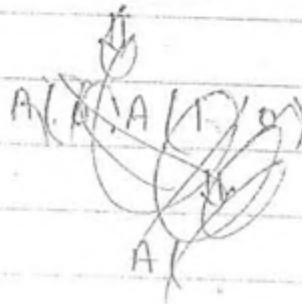
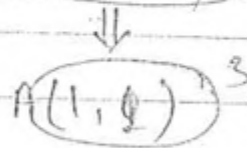
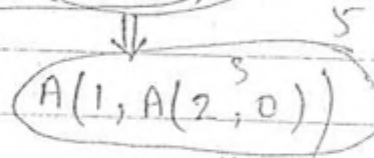
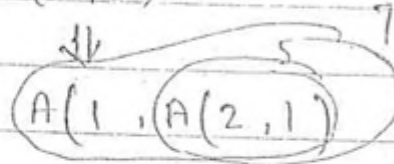
Ackermann's Recursion Relation.

$$A(m, n) = \begin{cases} n+1 & \text{if } m=0 \\ A(m-1, 1) & \text{if } n=0 \\ A(m-1, A(m, n-1)) & \text{otherwise} \end{cases}$$

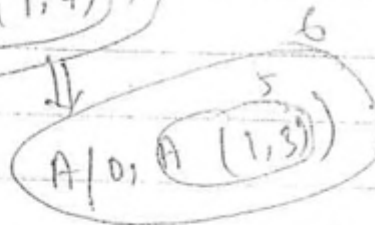
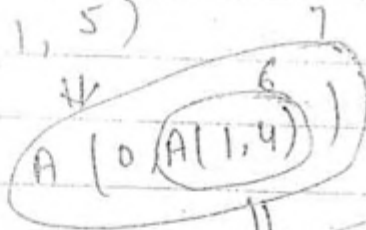


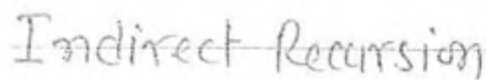
Find $\Rightarrow 7$

$A(2, 2)$



$A(1, 5)$



$$A(2, 0)$$


let
I = 3

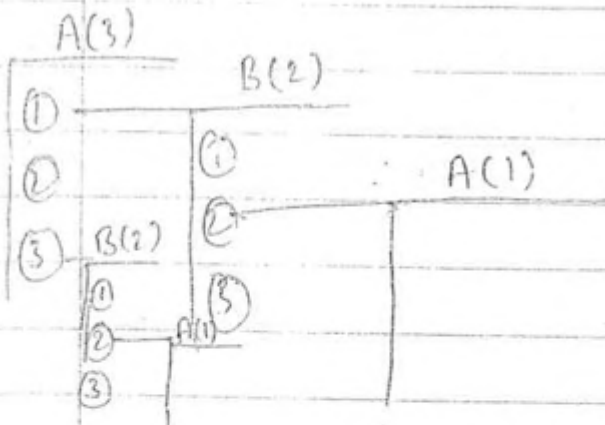
© Wiki Engineering


```

Border (int i)
{
    if (i <= 1) return
    else
    {
        ① Printt ("%d", i)
        ② Aorder (i-1)
        ③ Printt ("%d", i);
    }
}

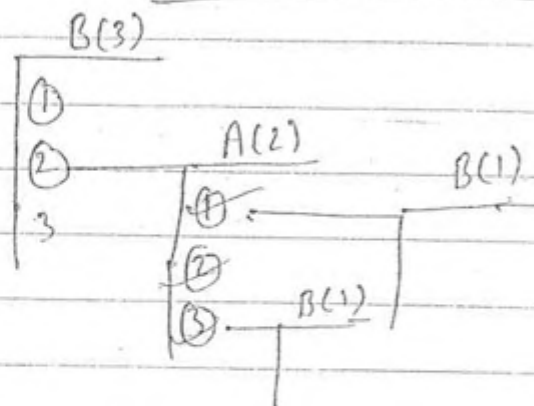
```

i/p : Aorder (3)



O/p → 2 2 3 2 2

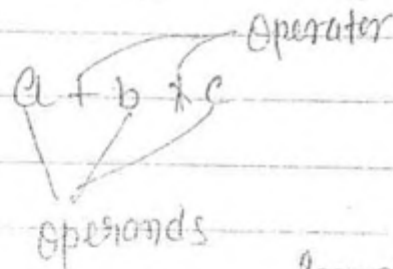
i/p: Border (3)



O/P :- 323

Infix to Postfix Conversion

Ex



Compare operator and operands
operands have higher priority.

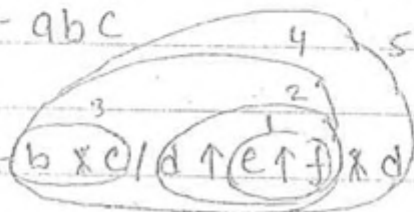
	Symbol	priority	Associativity
①	+ , -	1	L-R
②	* , /	2	L-R
③	↑	3	R-L

✓ Infix :- $a + b * c$
 Postfix :- $a b c * +$

Prefix :- $+ a * b c$

✓ Infix :- $a + b - c$
 Postfix :- $a b + c -$
 Prefix :- $- + a b c$

✓ Infix :- $a + (b * c) / (d * (e + f)) * d - c + b$



Postfix :- $a \underbrace{b c *}_3 \underbrace{d \underbrace{e f +}_1}_2 / d * + c - b +$

4

Prefix :- $+ - + a \underbrace{* b c}_3 \underbrace{d * (e + f)}_2 d c b$

4

✓ Infix :- $a + b / c * d - e \uparrow f * g + h * i \uparrow a \uparrow b * c / d - e$

Postfix :- $a + b / c * d - e \uparrow f * g + h * i \uparrow t_1 * c / d - e$

$a + (b / c) * d - (e \uparrow f) * g + h * (i \uparrow t_1) * c / d - e$

$a + t_5 - t_6 + t_9 - e$

Index:- $a + (b/c) \times d - e \uparrow f \times g + h \times i \uparrow a \uparrow b \times c/d - e$

Postfix:- $abc/d \times e f \uparrow g \times h - i a b \uparrow \uparrow \times c \times d / + e -$

Prefix $- + - + \times / bcd \times \uparrow efg / \times \times h \uparrow i \uparrow abcde.$

Note

Infix: $a + b * c$

Postfix: $abc * +$

Prefix: $+ a * bc$

Operands
order same but
operator order
changing.

Postfix

Infix

Prefix

$234 * +$

$2 + 3 * 4$

$+ 2 * 34$

$2 + 12$

11

$2 \ 12 \ +$

14

Many

14 one.

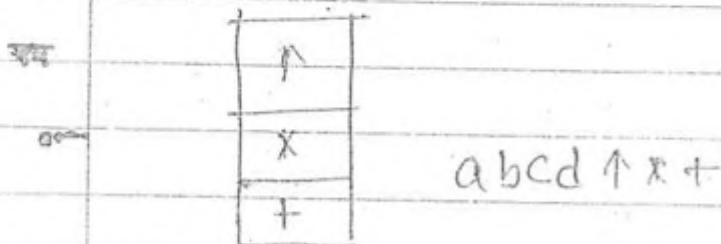
Many

Note:-

1. In order to evaluate infix expression many scans are required becoz we don't know where is higher priority order.
2. In order to evaluate Prefix expression many scans are required becoz we don't know where is higher priority order.

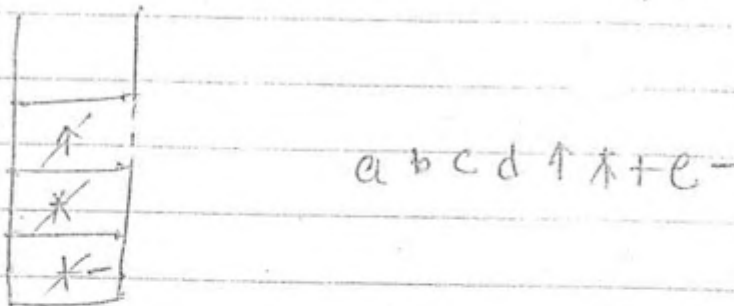
38 In order to evaluate postfix expression one scan is required. becoz higher priority will come at first.

$a + b * c \uparrow d$



#	Top lower	next Higher $\Rightarrow$ Push(next)
---	--------------	---

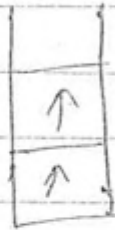
* $a + b * c \uparrow d - e$



#	Higher	lower $\Rightarrow$ Pop(Top)
	L-R	$\Rightarrow$ Pop(Top)
	R-L	$\Rightarrow$ Push(next)

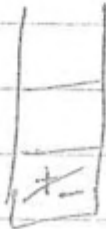
one
nty

* $a \uparrow b \uparrow c$



$abc \uparrow \uparrow$

* $a + b - c$



$ab + -$

Infix :- ~~$a + b * c / d \uparrow e \uparrow f * d - c + b$~~



Postfix

$abc * d e f \uparrow \uparrow / d * + c - b +$

#

Infix : $-a \times b / c \times d - e \uparrow f \times g + h \times i \uparrow a \uparrow b \times c / d - e$

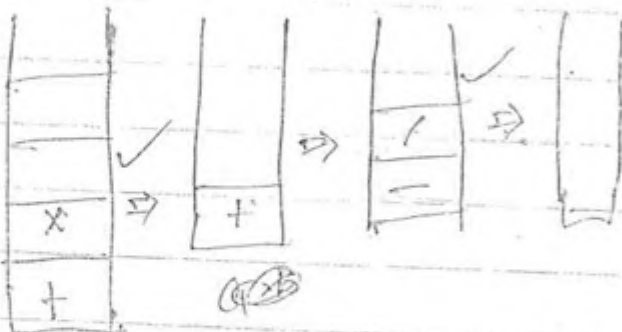
a/	
+	

$abc / d \times + e f \uparrow g \times - h i a b \uparrow \times c \times$

$d / + e -$

$\Rightarrow$ While converting Infix expression into Postfix expression we are pushing operator on the stack. which is known as. operator stack.

P What is the max^m size of operator stack during the conversion of infix operation $a + b \times c - d / e$ into postfix.



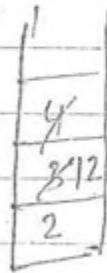
$abc \times + de$

max 2

Postfix Evaluation

1/p : $2 + 3 * 4$ op

Post : ~~(2+3)*4~~ +



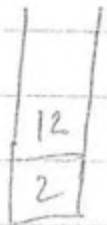
⇒

${}^4\text{op}_2 \Rightarrow \text{Pop}()$

${}^3\text{op}_1 \Rightarrow \text{Pop}()$

$r = \text{op}_1 \quad \text{op} \quad \text{op}_2$

Push(r)



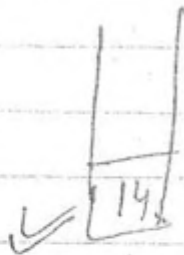
⇒

${}^{12}\text{op}_2 = \text{Pop}()$

${}^2\text{op}_1 = \text{Pop}()$

$r = \text{op}_1 \quad \text{op} \quad \text{op}_2$

Push(r)



~~Postfix~~

Postfix Evaluation

```

{
  while (character is not null)
  {
    if (character is operand)
      Push(character)
    else
    {
      OP2 = Pop()
      OP1 = Pop()
      r = OP1 character OP2
      Push(r)
    }
  }
  Print f("%d", Pop());
}

```



$OP_2 = 12$
 $OP_1 = 2$
 $r = 2 + 12$
 Push(r)

① (i)
 (ii)

②

Solⁿ

Q If -, & and \$ are used Subtraction, multiple and exponential.

- (1) (i) - is higher, than * and the \$ (lowest)
(ii) All are L-R associative.

i) P: - $3-2 \times 4 \$ 1 \times 2 \$ 3$

a) 512 b) 80 c) 51 d) ~~4096~~

(2) all are R-L.

$2 \times 2 - 1 \$ 1 \$ 4 - 2$

a) 256 b) ~~2~~ c) 25 d) 2

Solⁿ

*
7*

$$32 - 4 \times 12 \times \$ 3 \$$$

24
24

$$(16)^3 = (2^4)^3 = 2^{12} = 4096$$

-
+
*

$$= 221 - * 142 - \$ \$$$

P Evaluate the following expression.

$$1 \times 2 \wedge 3 \times 4 \wedge 5 \times 6$$

- a) 32^{30} b) 16^{30} c) ~~49/51~~ d) 173458.

Solⁿ



$$1 \times 2 \wedge 3 \times 4 \wedge 5 \times 6 \times$$

P Assume that the operators $\times$, $-$ and $+$ are left to right associative and $\wedge$ is R-L associative. The order of the priority ^{from} high to low is $(\wedge, \times, +, -)$

The Postfix expression corresponding to the following Infix expression.

$$a + b \times c - d \wedge e \wedge f$$

a) ~~$abc \times + def \wedge \wedge -$~~

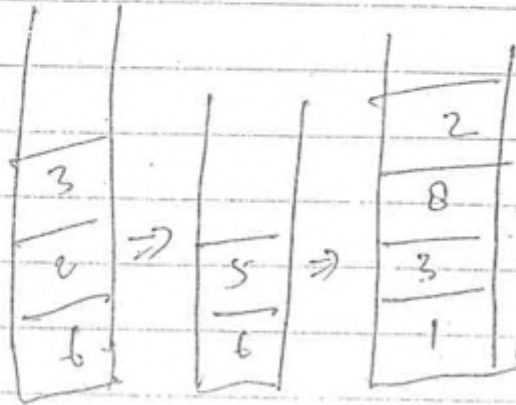
b) $abc \times + def \wedge \wedge -$

c) $ab + c \times b - e \wedge f \wedge d$

Q What is the max^m size of the operand stack while evaluating the postfix expression.

$-623 + -302 / + *$

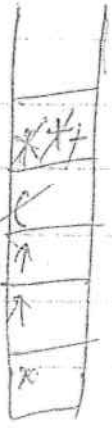
Soln



④

Prefix to Postfix Conversion.

✓ $a \times b \wedge c \uparrow (d \wedge e + b - c)$



g b c d e x b + - c -

①

②

③

④

⑤

✓ $c \wedge (a + b) \times d$



c a b + d x ^

⑥

Prefix

Postfix

① $a \text{ (minimum) } \longrightarrow a$

② $+ a b \longrightarrow a b +$

③ $* a * b c \longrightarrow a b c **$

④ $+ x a b / c d \longrightarrow a b / a b * c d / +$

⑤ $+ (- * / a b c d) \uparrow e f$ $\longrightarrow$ $+ (- * / a b c d) \uparrow e f$

$\Downarrow$

$a b / c * d - e f \uparrow +$

⑥ $x = \uparrow a b c d - \uparrow / x a b c d + a b$

$\Downarrow$

$a b \uparrow c / d - a b * c / d \uparrow a b + - *$

$* (/ \uparrow a b c d - \uparrow / * a b c d + a b)$

Prefix

* $a + bc$

↓

Postfix: $- a bc +$

Infix: $- a \times (b + c)$

* $a[b] = "abcde";$
0 1 2 3 4

~~str~~ $S_1 = "ab"$

Substr ($a, 1, 3, temp$) $\rightarrow S_2 = "cd"$
bcd

Strcat (S_1, S_2)

1 = Is operand (Symbol)
 0

PretoPost (Prefix, Postfix) $x + ab - cd \setminus o$
+ a b - c d \setminus o
0 1 2 3 4 5 6 7

$\{$
4
 length = strlen (Prefix)

if (length == 1)

$\{$
OP
 Postfix [0] = Prefix[0];

Postfix[1] = '\0';

return (Postfix);

$\}$

else

$OP[x] = Prefix[x];$

$OP[1] = "1";$

$Substr(Prefix, 1, length-1, temp)$

$m = find(temp)$

$Substr(Prefix, 1, m, opnd_1)$

$Pre to Post(opnd_1, Prefix_1)$

$Substr(Prefix, m+1, length-1, temp)$

$n = find(temp)$

$Substr(Prefix, m+1, m+n, opnd_2)$

$Pre to Post(opnd_2, Postfix_2)$

$strcat(Postfix_1, Postfix_2)$

$strcat(Postfix_1, OP[x]) \Rightarrow ab+$

$strcpy(Postfix_1, Postfix_2)$

$ab+cd- \quad ab+cd-*$

Red

Red color is
a example
and black
color is also
a another
example.

$ab-cd$
 012345
 abc
 012

$find(str)$
 $length = strlen(str)$
 $if(str[0] == operand)$

return 1;

else

$Substr(str, 1, length-1, temp)$
 $ab-cd$

$m = find(temp)$
 $ab-cd$

$Substr(str, m+1, length-1, temp)$
 $b-cd$

$find(temp)$
 $b-cd$

return(m+n+1)

Fibonacci Series.

n	0	1	2	3	4	5	6	7	8	9	10
fib(n)	0	1	1	2	3	5	8	13	21	34	55

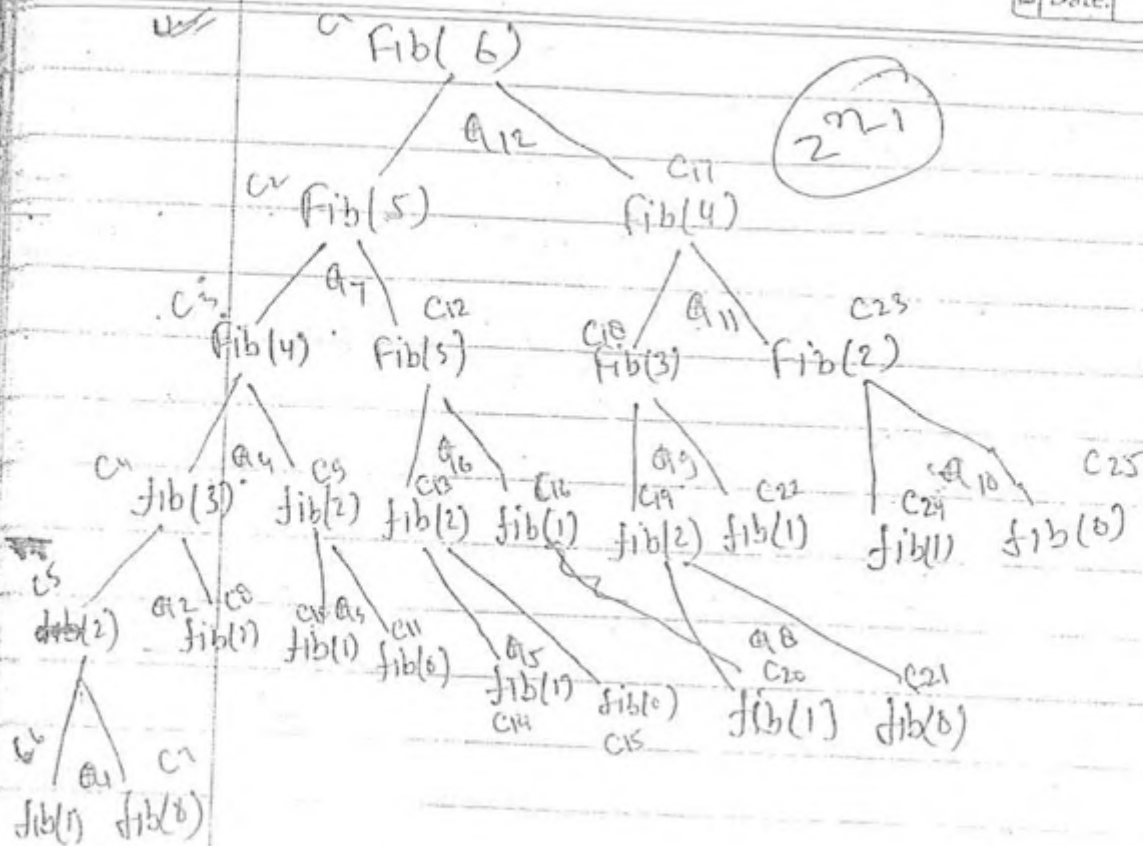
$$fib(n) = \begin{cases} 0 & \text{if } n=0 \\ 1 & \text{if } n=1 \\ fib(n-1) + fib(n-2) & \text{otherwise} \end{cases}$$

Fib (int n)

```

{
    int f;
    if (n==0) return 0;
    else
    if (n==1) return 1;
    else
    {
        f = fib(n-1) + fib(n-2);
        return f;
    }
}

```



Complete binary tree

$$2^n - 1$$

* The funⁿ calling sequence in any programming language is pre order traversal.

P In fibonaci of 10,000 after how many funcⁿ call 1st Addition taken place.

Solⁿ after 1001^{fun} call 1st addition will be done.

NOTE

In fibonoci of n after $n+1$ function called after 1st addition will be done it

Q In fibonoci of n is there any funⁿ call after last addition ?

Solⁿ no - becoz after last addition parent is available. (means final result availb)

Q In fibonoci of n is there any addition after last funⁿ call.

Solⁿ Yes. becoz after C₂₅ there are three addition. a_{10}, a_{11}, a_{12} .

Q To Evaluate fibonic of n value How many funⁿ will be called.

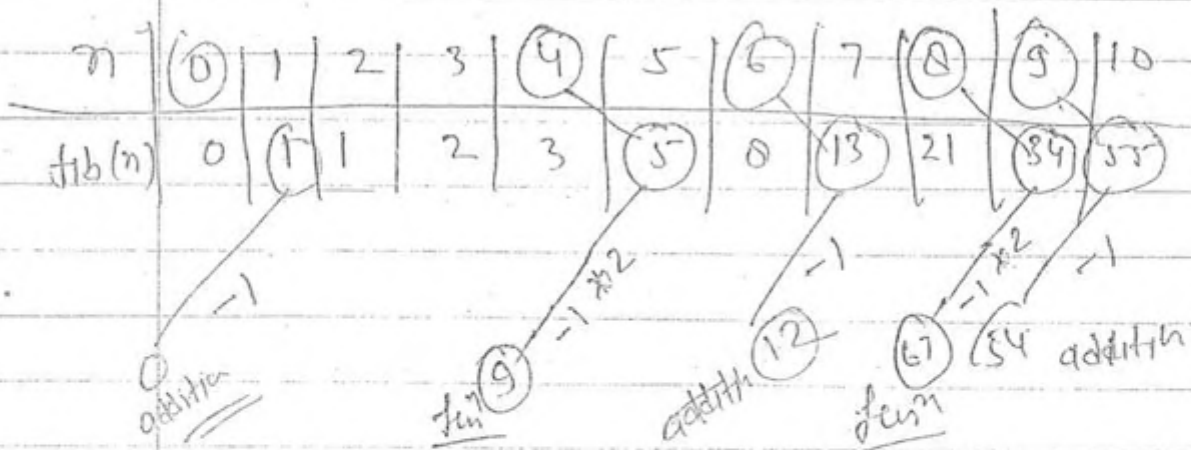
$f(0)=1, f(1)=1, fib(2)=3, fib(3)=5, fib(4)=9$
 $f(5)=15, f(6)=25, f(7)=41, f(8)=67, fib(9)=109$
 $f(10)=177$.

Q In fibononic of n how many additions will be performed.

$f(0)=0, f(1)=0, fib(2)=1, fib(3)=2, fib(4)=4$

$f(5)=7, f(6)=12, f(7)=20, f(8)=35$

$f(9)=54, f(10)=88$



Note The no of addition required to find fibnoci of n is.

$$\text{addition} = \text{fib}(n+1) - 1$$

value

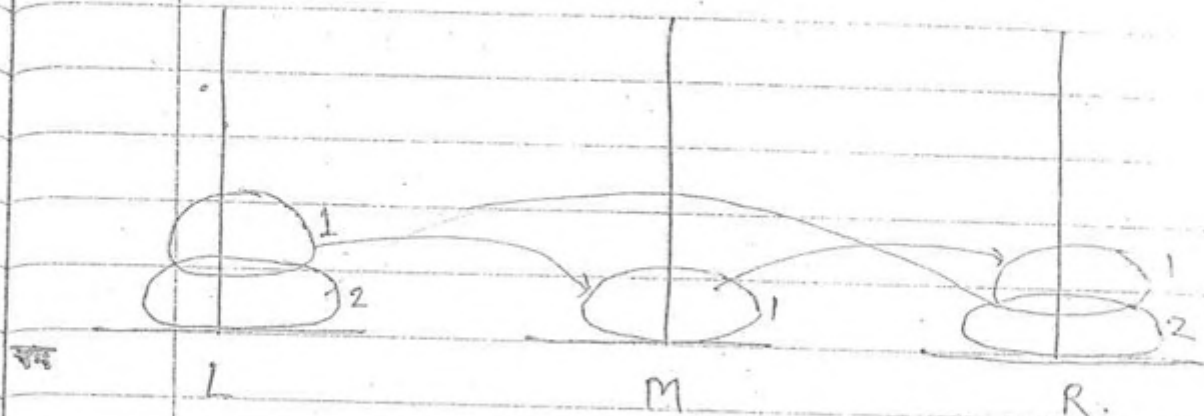
The no funⁿ calls in fibonic of n is.

$$\text{function} = 2 \times \text{fib}(n+1) - 1$$

value.

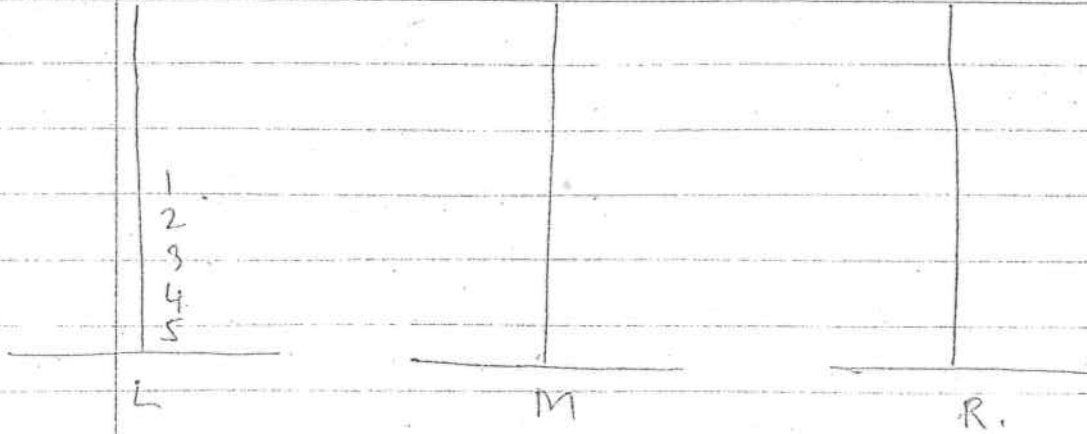
- ①
- ②
- ③

Towers of Hanoi



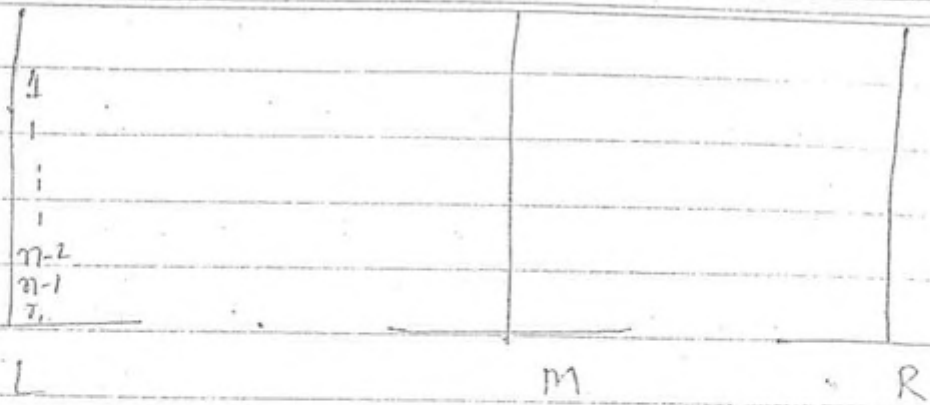
- ① At a time only one disk.
- ② On large size disk smaller size disk allowed.
- ③ All the disks will be there in 3-towers Only.





#

#



$TOH(n, L, M, R)$

- ```

{
 ① $TOH(n-1, L, R, M)$ - left to right middle using right.
 ② $MOVE(L - R)$
 ③ $TOH(n-1, M, L, R)$
}

```

#  $TOH(n, L, M, R)$

```

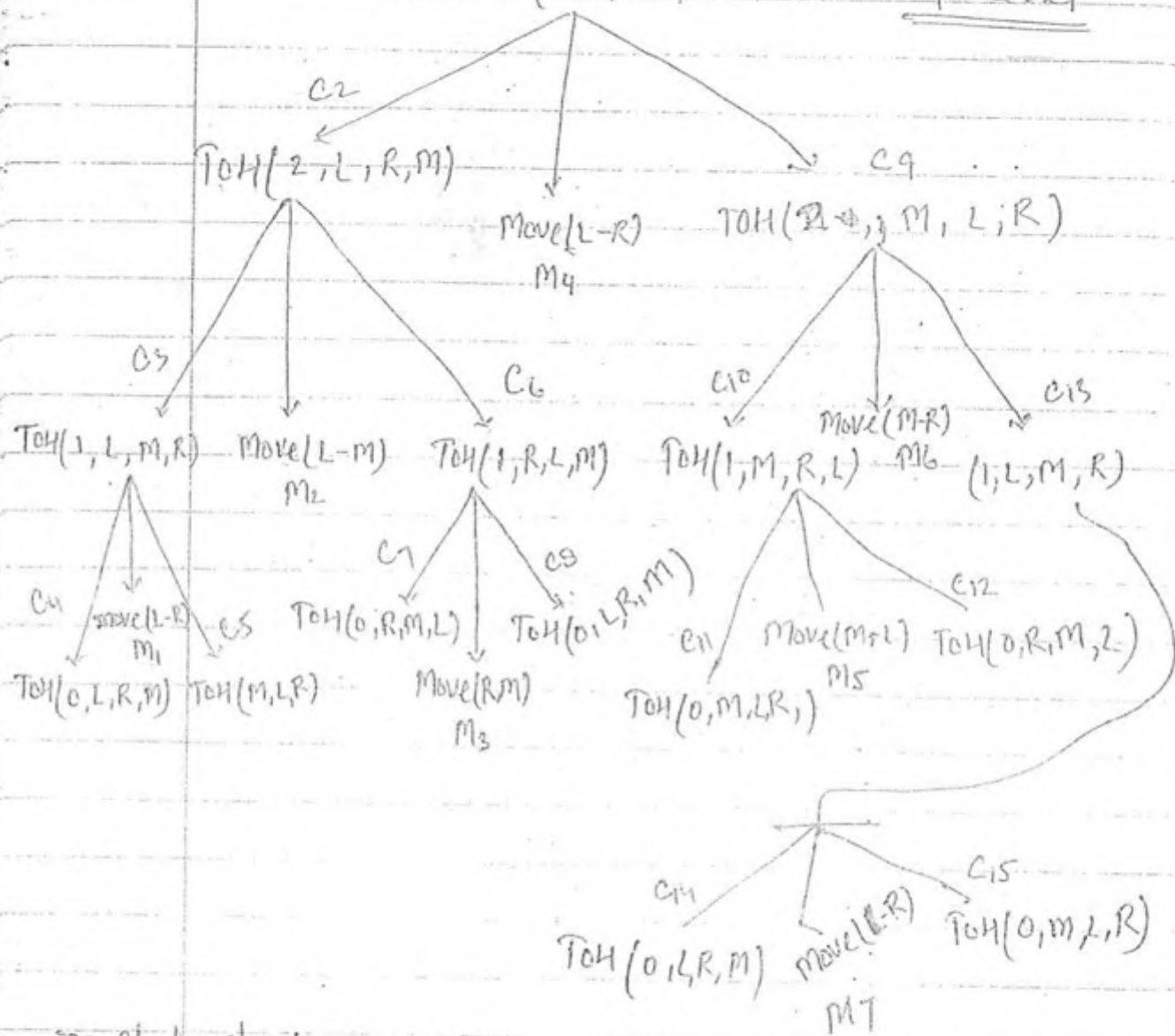
{
 if $(n == 0)$ return;
 else
 {
 ① $TOH(n-1, L, R, M)$
 ② $MOVE(L - R);$
 ③ $TOH(n-1, M, L, R);$
 }
}

```

#

$n = 3$

$C_1$   
TOH(3, L, M, R) 4-level



no of level = 4

no of nodes =  $2^n - 1$

no of function calling =  $2^n - 1 = 15$

no of move = 7

① (L-R)

② (L-M)

③ R-M

④ R-L

⑤ M-L

⑥ M-R

⑦ L-R

P Is there any fun<sup>n</sup> call after last move.

Sol<sup>n</sup> Yes after m7 there is C15.

P Is there any move after last fun<sup>n</sup> call.

Sol<sup>n</sup> No, after C15 there is no move.

P After How many fun<sup>n</sup> call first move will be taken place.

Sol<sup>n</sup>  $(n+1)$

P After How many fun<sup>n</sup> call there is a move from (m-R)

Sol<sup>n</sup> after 12 fun<sup>n</sup> call.

P How many fun<sup>n</sup> will be call in TOH n.

Sol<sup>n</sup>  $TOH(0) = 1$      $TOH(1) = 3$      $TOH(2) = 7$

$TOH(3) = 15$      $TOH(4) = 31$

$$T_{OH}(0) = 1 = (2^{0+1} - 1)$$

$$T_{OH}(1) = 3 = (2^{1+1} - 1)$$

$$T_{OH}(2) = 7 = (2^{2+1} - 1)$$

$$T_{OH}(3) = 15 = (2^{3+1} - 1)$$

$$T_{OH}(4) = 31 = (2^{4+1} - 1)$$

Note

The number of  $\text{fun}^n$  calls in  $T_{OH}(n)$

$$\boxed{2^{n+1} - 1}$$

Q

How many move are there in  $T_{OH}(n)$ .

$$T_{OH}(0) = 0 =$$

$$T_{OH}(1) = 1 = (2^1 - 1)$$

$$T_{OH}(2) = 3 = (2^2 - 1)$$

$$T_{OH}(3) = 7 = (2^3 - 1)$$

$$T_{OH}(4) = 15 = (2^4 - 1)$$

~~$$T_{OH}(n) = 2^{n+1} - 1$$~~

$$\boxed{T_{OH}(n) = 2^n - 1}$$



## Function Returning Pointer

```
int * fun(int a, int b)
```

```
{
 int x, y;
 x = 10, y = 20;
 x = x + y;
 x = x * a;
 y = x + y / b;
 return (y);
}
```

fun returning integer pointer.

## Function Pointer

```
Main()
```

```
{
 int display() <=> int a
 int (*fp)() <=> int *b
 fp = display; <=> b = &a
 (*fp)();
 //
 display
 //
 int display()
 {
 printf("Hi");
 }
}
```

fp is a fun<sup>n</sup> which contains fun<sup>n</sup> address.

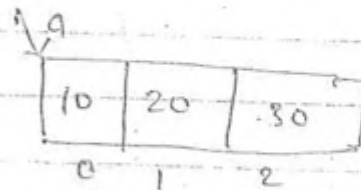
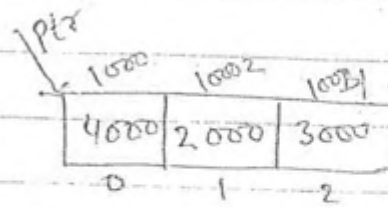
fun<sup>n</sup> name indicates fun<sup>n</sup> address.

Q. Consider the following Program.

```

Main()
{
 int (*Ptr [3])();
 Ptr[0] = aaa;
 Ptr[1] = bbb;
 Ptr[2] = ccc;
 (*Ptr[2])();
}

```



```

4000
aaa()
{
 Printf("aaa");
}

```

```

2000
bbb()
{
 Printf("bbb");
}

```

```

3000
ccc()
{
 Printf("ccc");
}

```

O/p :- ccc

`int a`  $\Rightarrow$  1 integer value

`int a[5]`  $\Rightarrow$  array integer value,

`int *a`  $\Rightarrow$  variable pointer.

`int *a[5]`  $\Rightarrow$  Array of variable pointer.

`int (*a)()`  $\Rightarrow$  1 fun<sup>n</sup> pointer

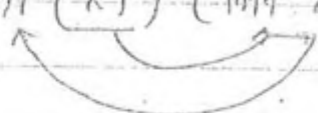
`int (*a[5])()`  $\Rightarrow$  Array of fun pointer.

`int (*a)[10]`

a is a pointer pointing to array of 10 elements.

Q What does the following C. Statement declaration?

`int (*f)(int *x)`



f is a pointer to a fun<sup>n</sup> which will take integer pointer as input which will be return integer as output.

```

f
 |
 v
{
 (int *)
 {
 return (22);
 }
}

```

P

What does the following C statement declare.

`void (*abc) (int, void (*def) ())`

Sol<sup>n</sup>

abc is a pointer to a fun<sup>n</sup> which will take two parameter as input

- ① integer.
- ② def is a ptr to a fun<sup>n</sup> with return void.
- ③ which will return void.

P

What does the following C-statement declare?

`char * (* (*a[5]) ()) ()`

Sol<sup>n</sup>

- ① array of five pointers point into a fun<sup>n</sup> which will return a pointer. which will. pointing to a fun<sup>n</sup>, which will return char pointer.

Q. What does the following C statement declare:

$(\text{char} * (*)) (\text{XPTR}[N]) ();$

Soln

Array of  $n$  pointers to fun<sup>n</sup> which will return char pointer.

Q. Match the following statement.

Group - I

- |                                                                           |                     |
|---------------------------------------------------------------------------|---------------------|
| (1) (c) A pointer to an array of 8 floats.                                | (a) float x(xf) ()  |
| (2) (d) A pointer to an array of 8 pointer to float.                      | (b) float (*f) ()   |
| (3) (b) A pointer to a fun <sup>n</sup> that returns float.               | (e) float (*a) [8]  |
| (4) (a) A pointer to a fun <sup>n</sup> that return a pointer to a float. | (d) float x(xa) [8] |
-

8

Consider the following C-program,

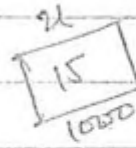
```

int main()
{
 int u = 15;
 printf("%d", fun(5, &u));
}

fun(int n, int *fp)
{
 int t, f;
 if (n <= 1)
 {
 *fp = 1;
 return 1;
 }

 t = fun(n-1, fp);
 f = t + *fp;
 *fp = t;
 return f;
}

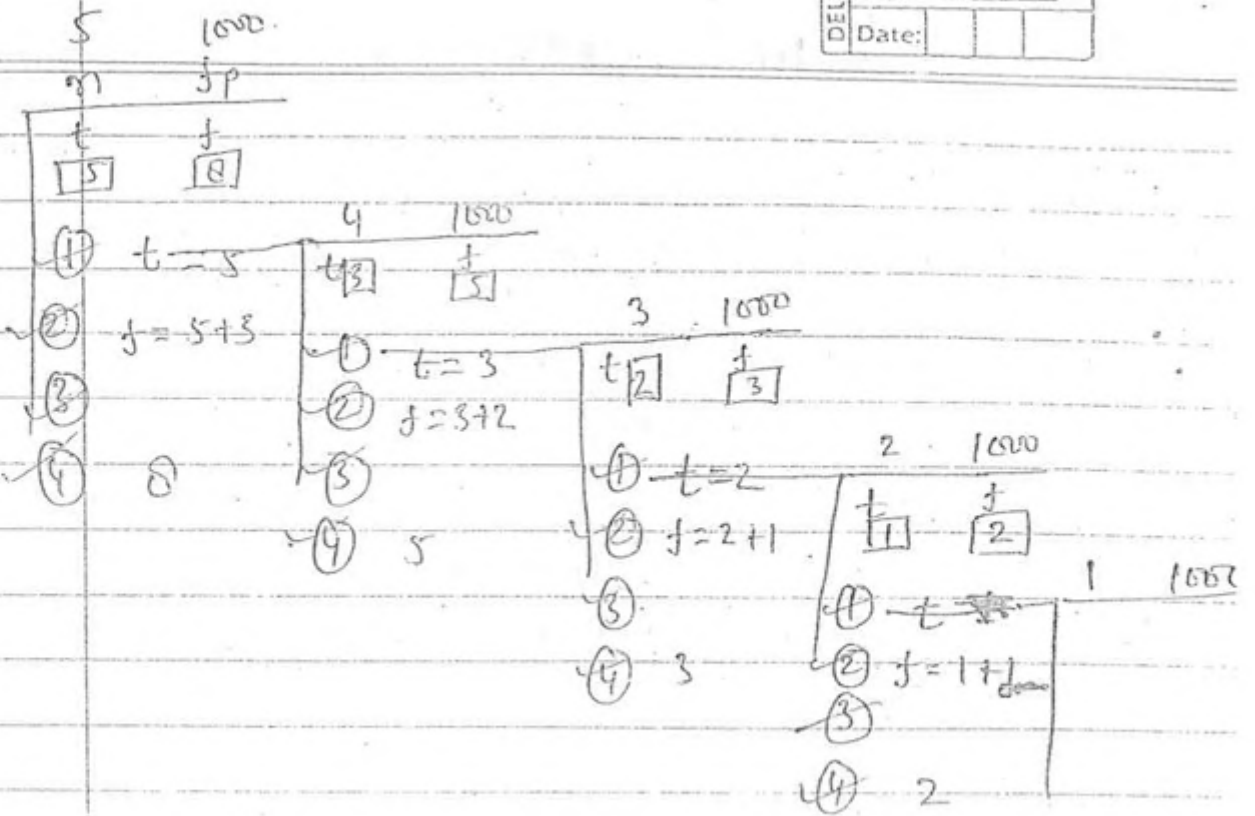
```



The value printed is.

- a) 6    b) 0    c) 14    d) 15







## Structure

↳ User defined  
Data types.

Struct node  
{  
  int a;  
  float b;  
}

*Creating a data type of struct.*

Struct node c  
  int a → 2  
  float b → 4.

*6 byte.*

⇒

Struct node

{  
  int a ;  
  float b ;  
}

*int a = 10  
float b = 10*

*not allocated*

Main ( )

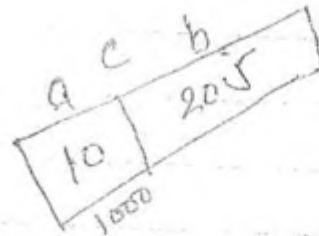
{

Struct node c = { 10, 20.5 };

printf ("%d" &a);

*select operator (.)*

}



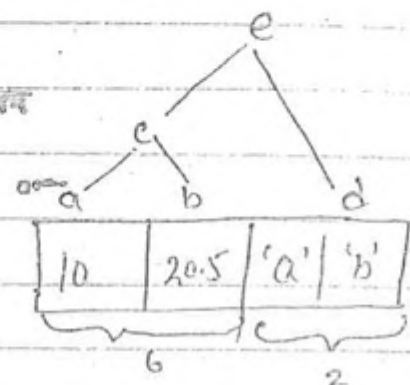
⇒ Struct node ⇒ 6

```
{
 int a; ⇒ 2
 float b; ⇒ 4
};
```

6 bytes

Struct node 1 ⇒ 8

```
{
 struct node c; ⇒ 6
 char d[2]; ⇒ 2
}
```



main()

```
{
 struct node 1 e = {10, 20.5, {'a', 'b'}}
}
```

```
printf("%d", e.c.a);
 e.c.b
 e.d[0]
 e.d[1]
}
```

⇒ Struct node

```
{
 int a;
 float b;
}
```

int a[20];

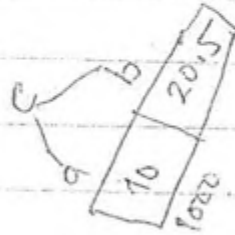
array of 20 int variable.

Struct node d[20];

array of 20 struct node variable.

Main()

```
{
```



① struct node c = { 10, 20.5 };

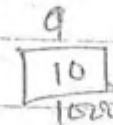
② struct node \*d;

③ d = &c

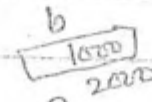
printf (" %d",

```
);
```

① int a = 10



② int \*b



③ b = &a

④

⑤ \*b

d = 1000

\*d = a, b

(\*d).a = 10

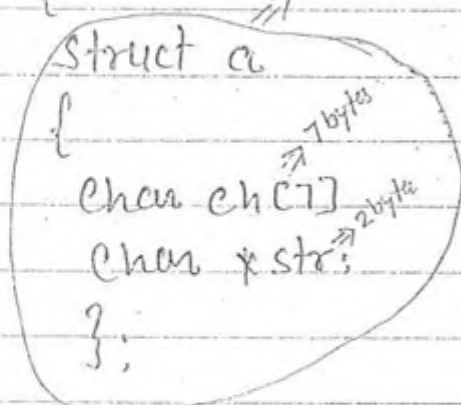
(\*d).b = 20.5

(\*d).a ⇒ d → a

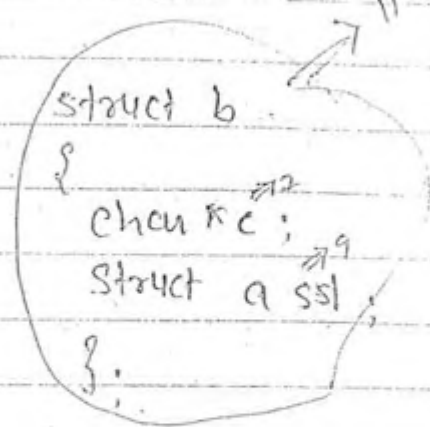
(\*d).b ⇒ d → b

o/p ⇒

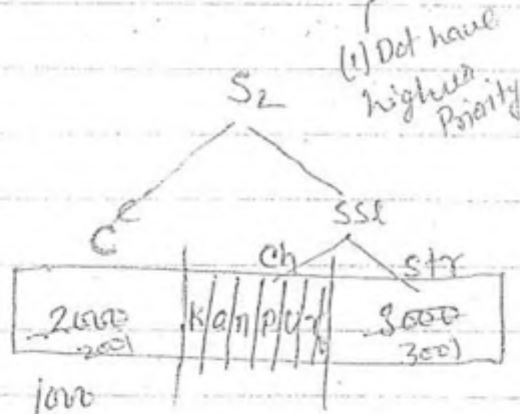
Main C)   
 {   
     Struct a   
     {   
       char ch[7]   
       char \*str;   
     }   
 }



struct b   
 {   
     char \*c;   
     struct a ssl;   
 }

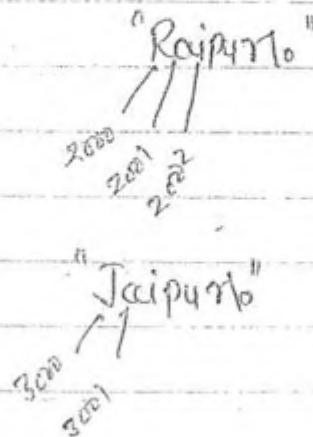


struct b s2 = { "Raipur", "Kampur", "Jaipur" };   
 printf [ "%s %s", s2.c, s2.ssl.str ];   
 printf [ "%s %s", +s2.c, +s2.ssl.str ];



Raipur

O/P → Raipur, Jaipur.   
        kaupur, aipur



##

Consider the following C. Program.

Main()

Struct s1

{  
Char \*str;

int i;

Struct s1 \*ptr;  
}

6 bytes

2 bytes

2 bytes

2 bytes

Struct s1 a[] = {  
{"Nagpur", 1, a+1},  
{"Raipur", 2, a+2},  
{"Kampur", 3, a}

2000

2000

2000

Struct s1 \*pa = a;

int j;

for (j=0; j<2; j++)

{  
Printf ("%d", a[j].i);

Printf ("%s", a[j].str);  
}

2000

o/p → 0 ; 1 2

agpur; aipur anpur

| str  | i  | ptr  | str  | i | ptr  | str  | i | ptr  |
|------|----|------|------|---|------|------|---|------|
| 2000 | 10 | 1006 | 3000 | 2 | 1012 | 4000 | 3 | 1000 |
| 0    |    |      | 1    |   |      | 2    |   |      |

P

Main ( )

```

{
 struct s1
 {
 char x2;
 int i;
 struct s1 *p;
 }

```

6 bytes.

①

$++(ptr \rightarrow 2)$

②

$a[(++ptr) \rightarrow i].z$

③

$a[--(ptr \rightarrow p \rightarrow i)].z$

```

struct s1 a[] = {
 {"Nagpur", 1, a+1},
 {"Raipur", 2, a+2},
 {"Kanpur", 3, a}
};

```

Struct s1 \* ptr = a

~~→ Nagpur.~~

Printf("%s", ++(ptr → 2)) ⇒ agpur

Printf("%s", a[(++ptr) → i].z) ⇒ Kanpur

Printf("%s", a[--(ptr → p → i)].z); ⇒ Kanpur.

| str  | i | p    | str  | i | p    | str  | i | p    |
|------|---|------|------|---|------|------|---|------|
| 2000 | 1 | 1006 | 3000 | 2 | 1012 | 4000 | 3 | 1000 |
| 0    |   |      | 1    |   |      | 2    |   |      |



Q

typedef struct P

{ char \*a, \*b;

} t;

void f1(t s);

void f2(t \*p);

main()

{  
t s = {"a", "b"};

printf("%s %s", s.a, s.b);

f1(s);

printf("%s %s", s.a, s.b);

f2(&s);

};

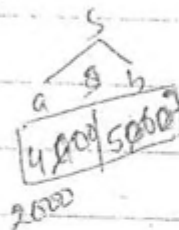
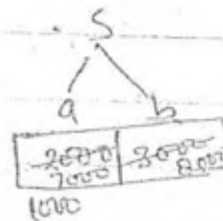
void f1(t s)

{  
s.a = "u";  
s.b = "v";

printf("%s %s", s.a, s.b);

return;

}





Void f2 (t \*p)

{ p → a = "v" <sup>7000</sup>

p → b = "w" <sup>8000</sup>;

printf ("%s %s", p → a, p → b);

}

<sup>↓</sup> v <sup>↓</sup> w

O/P:-

a) a b

u v

v w

v w

b) a b

u v

a b

v w

c) a b

u v

u v

v w

d) a b

u v

v w

u v

3-#

Struct Test

```
{
 int i;
 char *c;
};
```

```
Struct test st [7] = {
 { 5, "become" },
 { 4, "better" },
 { 6, "Jungle" },
 { 0, "ancest" },
 { 7, "brother" }
};
```

Annotations: 2000, 3000, 4000, 5000, 6000

main()

```
{
 Struct Test *p = st;
 pt = 1;
 printf("%d", ++(p->i));
 printf("%c", *(p->c));
 printf("%d", p[0].i);
 printf("%d", p[0].c);
}
```

Annotations: 3000, 4000, 5000, 6000, 7000, 8000, 9000, 10000

p[0].i  
\*(p+0)  
(\*(p+0)).i

O/P :-

a) jungle, n, 0, nester

b) etter, u, 6, ~~unle~~, ungle

c) etter, K, 6, jungle.

d) etter, u, 0, nester.

|             |       |   |       |       |   |       |       |   |      |       |   |       |       |   |
|-------------|-------|---|-------|-------|---|-------|-------|---|------|-------|---|-------|-------|---|
| St<br>10000 | i     | c | 10004 | i     | c | 10008 | i     | c | 1012 | i     | c | 1016. | i     | c |
| 5           | 20000 |   | 4     | 30000 |   | 6     | 40000 |   | 0    | 50000 |   | 7     | 60000 |   |
| 0           |       |   | 1     |       |   | 2     |       |   | 3    |       |   | 4     |       |   |

struct s

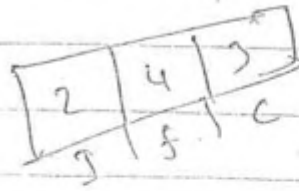
{ int a;  $\Rightarrow 2$

float b;  $\Rightarrow 4$

char c[3] = "abc"  $\Rightarrow 3$

}

Size of (struct s) = 9.



Union s

{ int a;  $\Rightarrow 2$

float b;  $\Rightarrow 4$

char c[3];  $\Rightarrow 3$

};

Size of (union s) = 4.  
(max)



Union s1

{

int a;  $\Rightarrow 2$

float b;  $\Rightarrow 4$

char c[3]  $\Rightarrow 3$

struct s d;  $\Rightarrow 9$

}

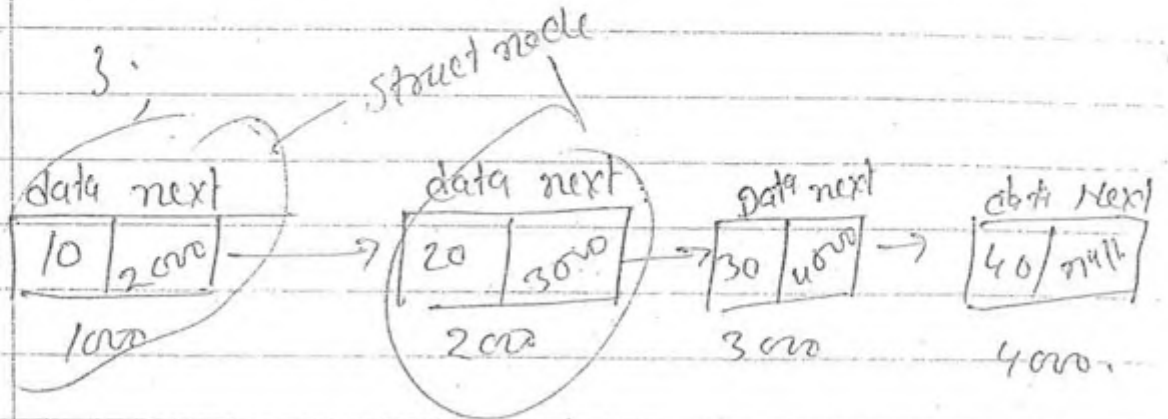
Size of (union s1) = 9.

Struct node

```
{ int data;
```

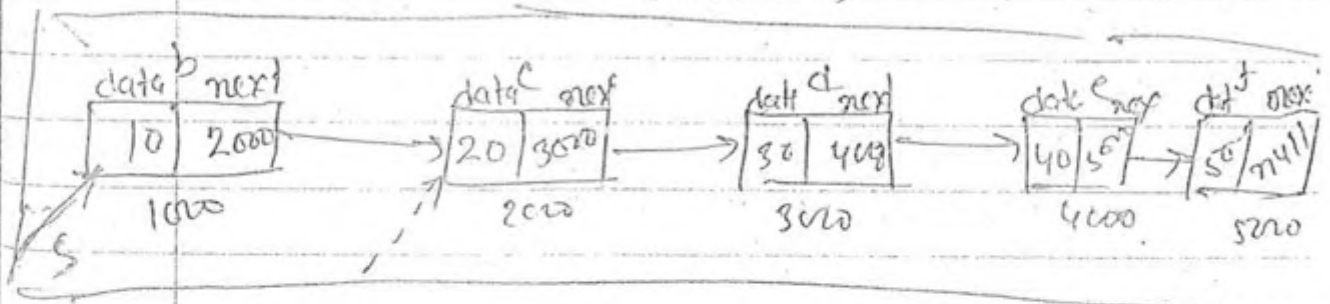
```
struct node * next;
```

};



Self referential.  
data structure.

Struct node b, c, d, e, f;



b. data = 10  
b. next = 2000

5000 X  
↓  
1000

$S = S + 1 \Rightarrow 2000$  X

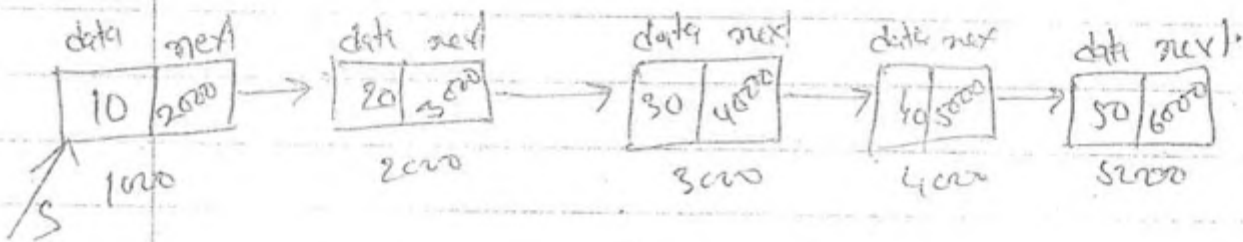
$\Downarrow$   
1004 ✓

$S = S \rightarrow \text{next} \Rightarrow 2000$

$\Downarrow$   
1004 X

P

Write a C Program to print the data in the given linked List.



S/m

Struct node

$\boxed{S \rightarrow 1000}$

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

display (struct node \*s)

```
{
 while (s != null)
 {
 printf ("%d", s->data);
 }
}
```



$s \Rightarrow s \rightarrow \text{next};$   
 $\downarrow$   
 $\downarrow$   
 $\downarrow$

O/P  $\Rightarrow 10, 20, 30, 40, 50$

Q Write a C-Program to print data value of a last node.

Soln

$s$   
 $\boxed{100}$

$s \rightarrow \text{data} \Rightarrow 100$   
 $s \rightarrow \text{next} \Rightarrow 2000$

$s \leftarrow$  Dangling pointer  
 $\boxed{\text{null}}$

$s \rightarrow \text{data}$  } Segmentation error?  
 $s \rightarrow \text{next}$  } " " )

Print last (Struct node \*s)

```

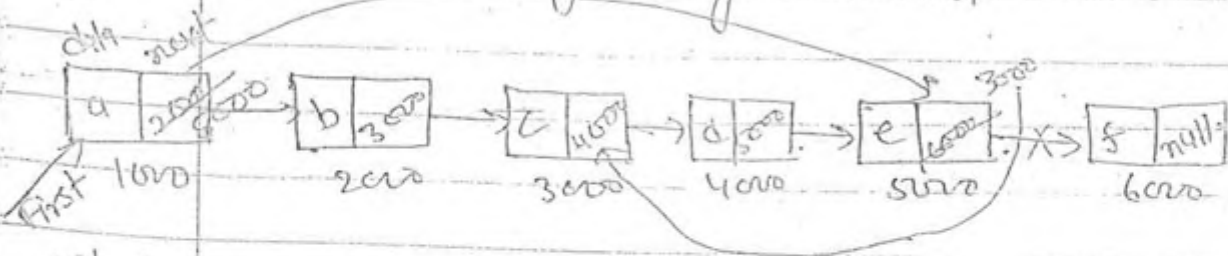
{
 if (s == null)
 return (-1);
 else
 {
 while (s->next != null)
 {
 s = s->next;
 }
 printf("%d", s->data);
 }
}

```



P#

Consider the following linked list.



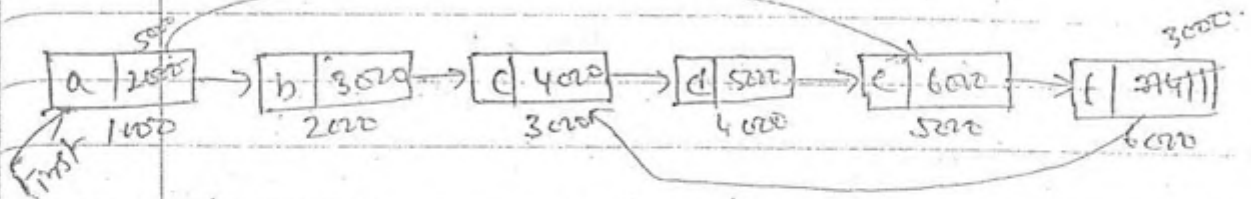
What will be the output of after following execution.

- (i) Struct node \* p;
- (ii) p = (first → next) → next → next;
- (iii) first → next = (p → next) → next;
- (iv) (p → next) → next → next = p;
- (v) printf ("%c", first → next → next → next → data);

Soln

o/p = d.

2. Consider the following linked list.



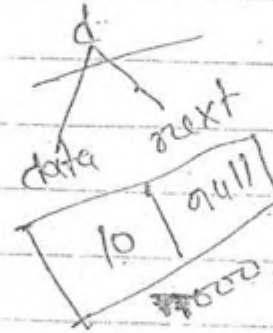
What will be the app after following execution.

- (1) struct node \* p;
- (2) ~~p~~ = first → next → next;
- (3) first → next = p → next → next;
- (4) p → next → next → next → next = p;
- (5) printf( first → next → next → next → data );

app → c

```
Main();
{
```

```
 struct node d = {10, null};
```



```
}
```

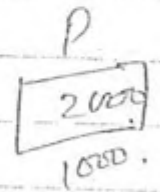
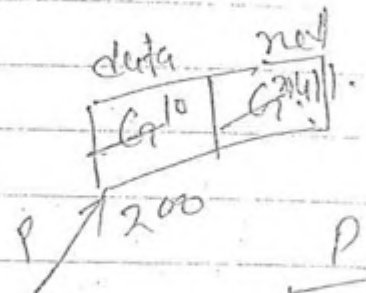
Static ← compile time memory  
errors

Stack

Heap

Dynamic memory allocation means Heap

Malloc  
memory allocation.



```

{
 struct node *P;
 P = (struct node *) malloc (size of (struct node))

```

Type casting

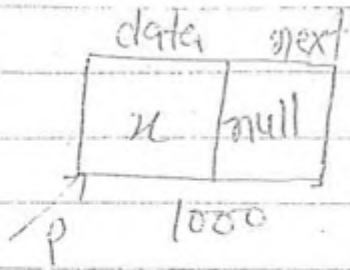
(\*P).data or P->data = 10

(\*P).next = null or P->next = null

return (P)

}

#



~~Getnode(u)~~

~~{~~  
Struct node \*p;



double a = 10.5

double \*x;  
int \*b = (int\*) &a

char \*c = (char\*) &a

float \*d = (float\*) &a

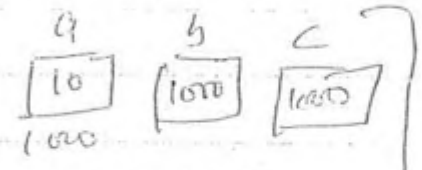
| double     | float      | char       | int        | <del>Void</del>        | Void       |
|------------|------------|------------|------------|------------------------|------------|
| ++<br>1000 | ++<br>1000 | ++<br>1000 | ++<br>1000 | <del>++<br/>1000</del> | ++<br>1000 |
| 1008       | 1008       | 1008       | 1002       | <del>1001</del>        | ?          |

int \*p  
float \*q  
~~int \*r~~ } Error msg

int \*p  
void \*q  
q = p; } no Error.

int a = 10  
b = (float) a

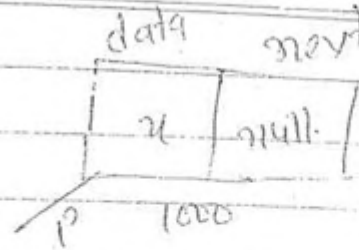
int \*c  
int a = 10  
b = (float\*) &a  
c = &a.



b++ = 1004  
c++ = 1002

~~Getnode (u)~~

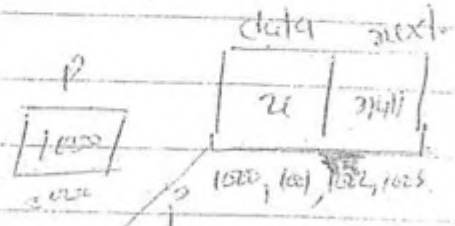
```
{
 struct node *p;
```



```
p = (struct node) malloc (sizeof (struct node));
```

```
if (p == Null)
```

```
{
 printf ("memory over");
 exit (1);
}
```



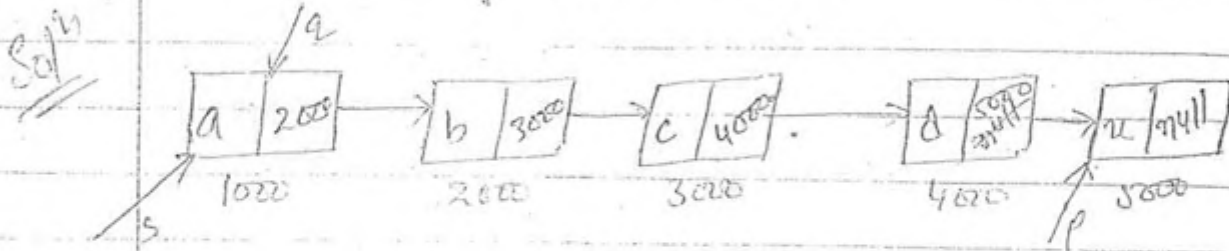
```
else
```

```
{
 p->data = u;
 p->next = null;
 return (p);
}
```

```
}
```



# WAP to add a node with data  $u$  at the end of linked List.



Struct node: Addend LI (Struct node \*s, int u)

{

Struct node xp, kq;

q = s;

p = (Struct node) malloc (sizeof (struct node))

p → data = u.

p → next = null;

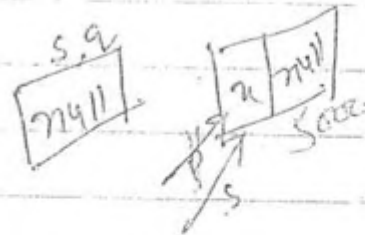
if (q == Null)

{

s = p;

return (s)

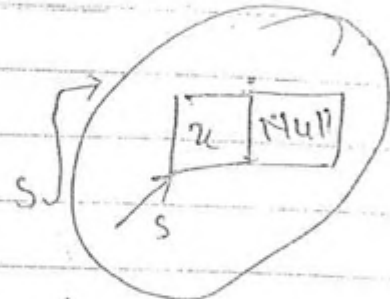
};





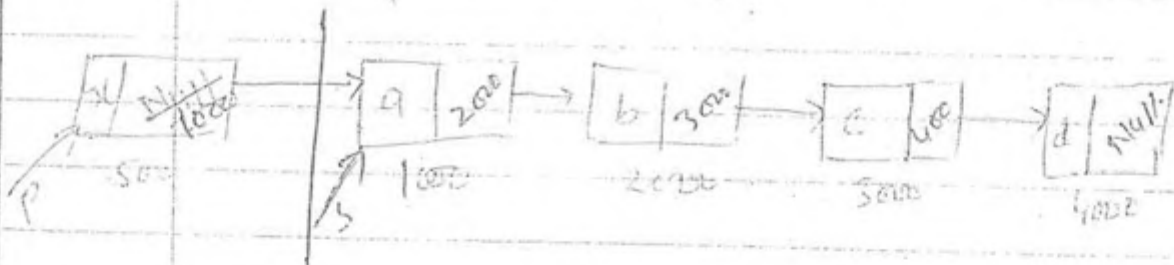
else

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



}

# WACP to add a node with data  $u$  at the start of linked list.



Add  $u$  before (L/struct nodes  $s$ , int  $u$ )

{

struct node \* p;

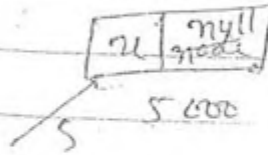
p = (struct nod) malloc (Size of (struct node));

p->data = u, p->next = null;

$$p \rightarrow \text{next} = s$$
$$S \equiv P;$$

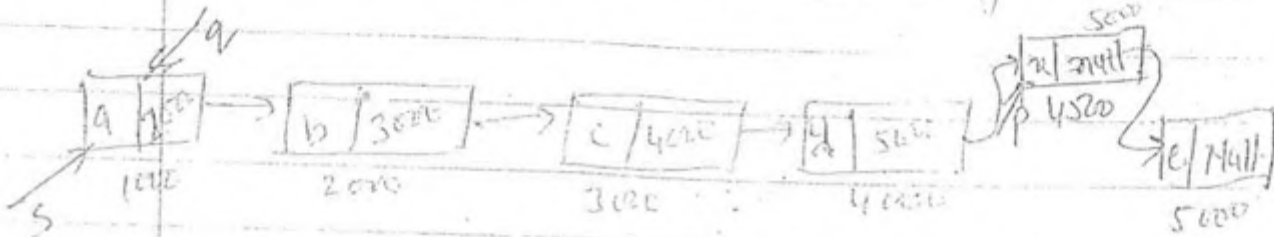
```
return CS;
```

3



井

WACP to insert a node with data  $x$  after a node with data  $y$ .



Add  $z$  after  $y$  in `(Struct node *x, int z, int y)`.

Struct node  $x_p, x_q$

$p_i = \text{head node}(u);$

$$q = 5 \div$$

if  $(s == null)$

```
Print + (" no node L:Li is empty");
exit(1);
}
```

else

if (s->data == y)

{

p->next = s->next;

s->next = p;

}

else

{ while (q->data != y && q->next != null)

q = q->next

if (q->data == y)

{

p->next = q->next;

q->next = p;

}

else

{

printf("y not present")

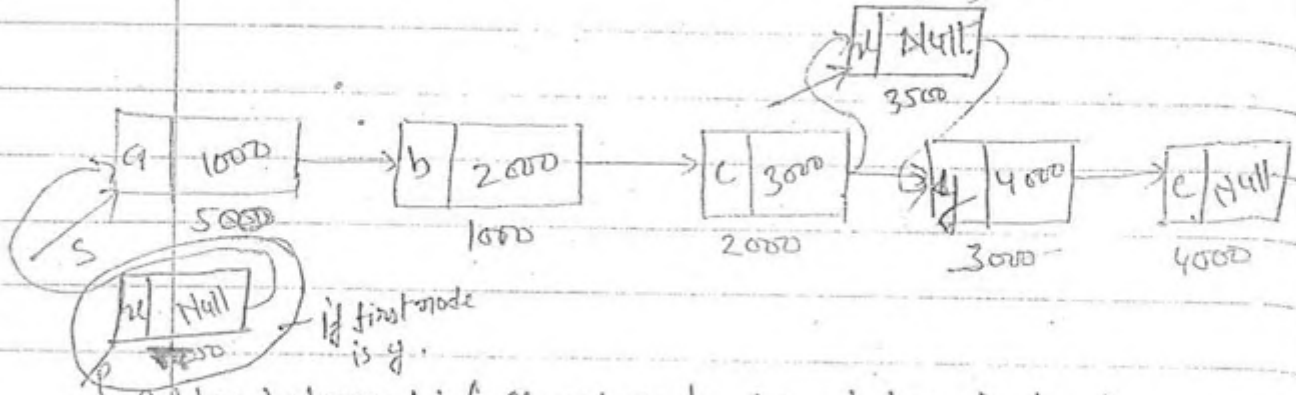
exit(1);

}

}

}

# WACP to add a node with data  $x$  before a node with data  $y$ .



Add  $x$  before  $y$  LL (Struct node  $*s$ , int  $x$ , int  $y$ )

```
{
 Struct node $*p$, $*q$, $*r$;
```

```
 $p = \text{get_node}(x);$
```

```
 $q = s; r = \text{null};$
```

```
 if ($s == \text{null}$)
```

```
 {
 Print ("LL is empty")
 exit (1);
 }
```

```
 else
```

```
 {
 if ($s \rightarrow \text{data} == y$)
 {
 $p \rightarrow \text{next} = s;$
 $s = p;$
 }
 }
```

else

{

⇒

while(  $q \rightarrow \text{data} \neq y$  &&  $q \rightarrow \text{next} \neq \text{null}$  )

{

$r = q;$

$q = q \rightarrow \text{next};$

with 2 pointers

}

if (  $q \rightarrow \text{data} == y$  )

{

$r \rightarrow \text{next} = p;$

$p = \text{next} = q;$

}

else

{

Printf("Not possible");

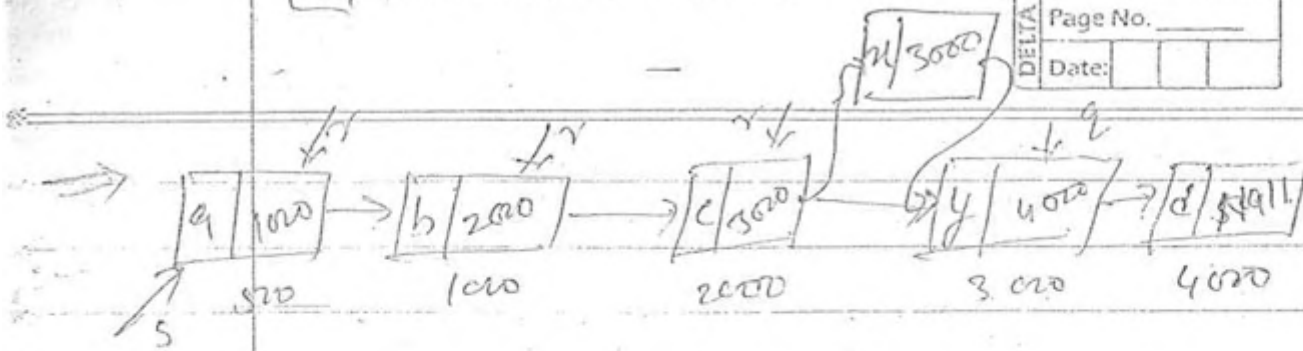
}

}

}

[More than one node]

|       |                |
|-------|----------------|
| DELTA | Page No. _____ |
|       | Date: _____    |



q = s;

while (q → next != Null && q → next → data != y)

q = q → next;

if (q → next == Null)

{

printf("y is not available");  
exit(1);

}

else

{

p → next = q → next;

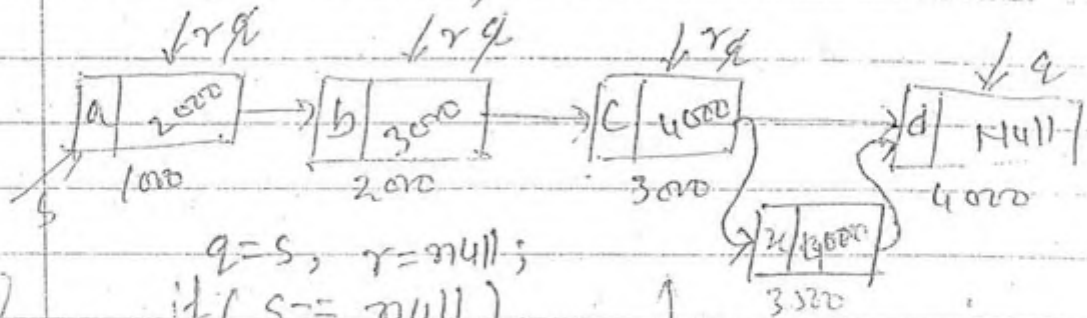
q → next = p;

}

Using Single  
Pointer



# WACP to add a node with data  $x$  before the last node;



```

(1) q = s, r = null;
 if (s == null)
 {
 printf("not possible") LL empty.
 exit(1);
 }

```

```

(2) if (s->next == null)
 {
 p->next = s;
 s = p;
 }

```

↑ 1-node

```

while (q->next != null)
{
 r = q;
 q = q->next;
}
r->next = p;
p->next = q;

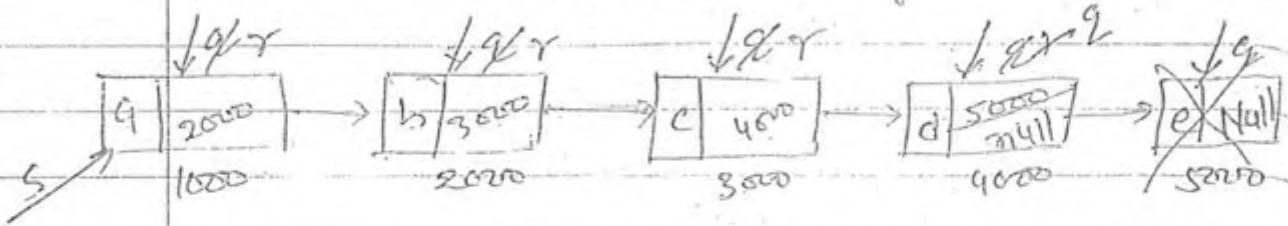
```

↑ more than 1 node.



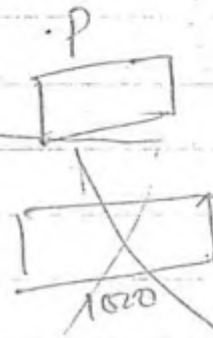
(mode)

# WACP to delete last element of Linked list.



Struct node \*p

p = ( ) malloc( )



\* no one can point where no memory is called Dangling Pointer



Delete node LL (struct node \*s)

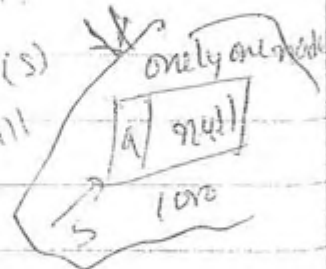
```
{
 struct node *q;
 q = s;
 if (s->next == null)
 {
 free(s);
 s = null;
 }
}
```

① empty.

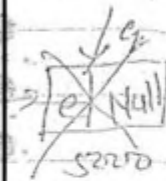
② while (q->next != null)

```
{
 r = q;
 q = q->next;
}
```

more than



list.



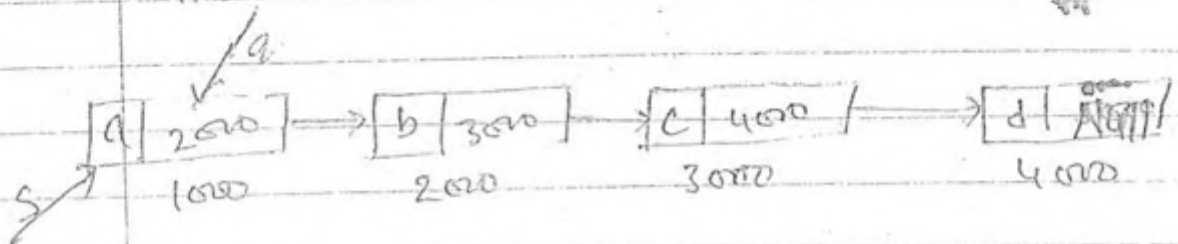
$r \rightarrow \text{next} = \text{null};$

$\text{free}(q);$

$q = \text{null};$

1-node.

P WACP to delete a node from start of linked list.



$\text{delete\_start}(\text{struct node } *s)$

{

① empty  $\updownarrow$  0-nodes.

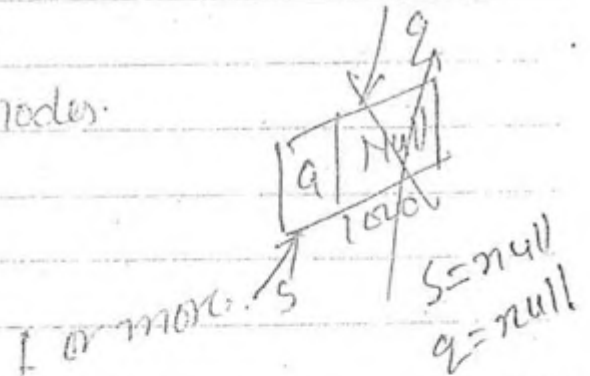
②  $q = s;$

$s = s \rightarrow \text{next};$

$\text{free}(r);$

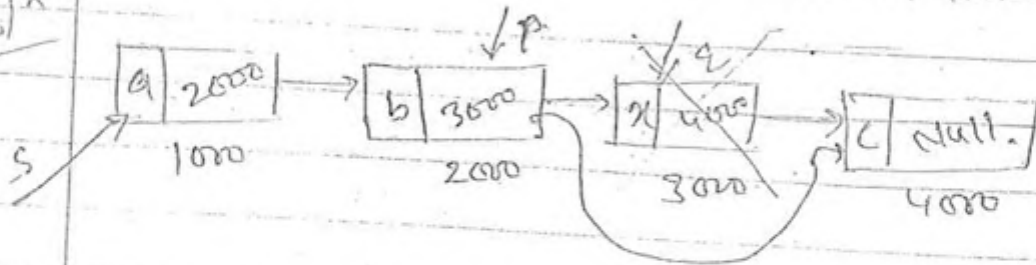
$q = \text{null};$

}



# WACP to delete a node which contains data as  $x$  from the linked list.

Sol<sup>n</sup>



Delete  $x$  LL (struct node \*S, int  $x$ )

```

{
 struct node *p, *q;
 if (S == null)
 {
 empty
 }
 else
 {
 if (S->data == x)
 {
 p = S;
 S = S->next;
 free p;
 p = null;
 }
 }
}

```

else

q = s

while ( q → data != u && q → next != null )

{

p = q;

q = q → next;

}

if ( q → data == u )

{

p → next = q → next;

free(q); q = Null;

}

else

{ u is not there

}

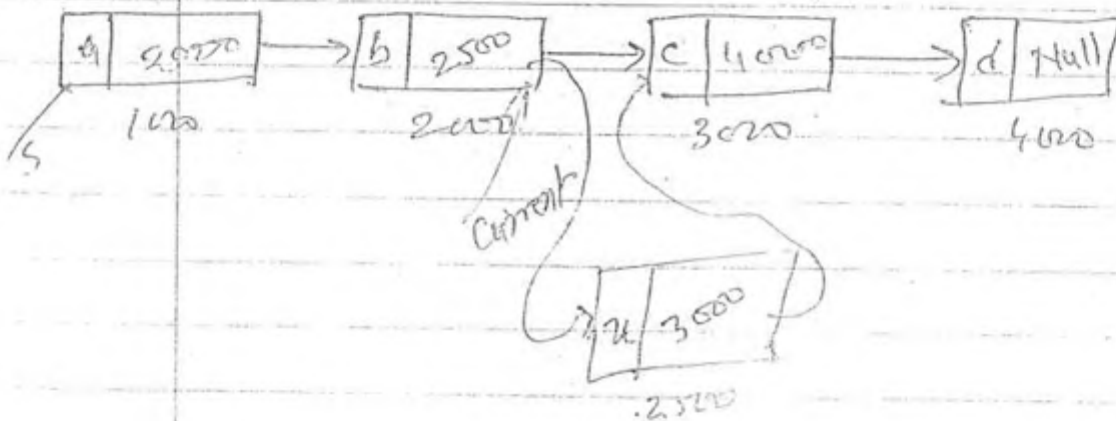
}

}

✖

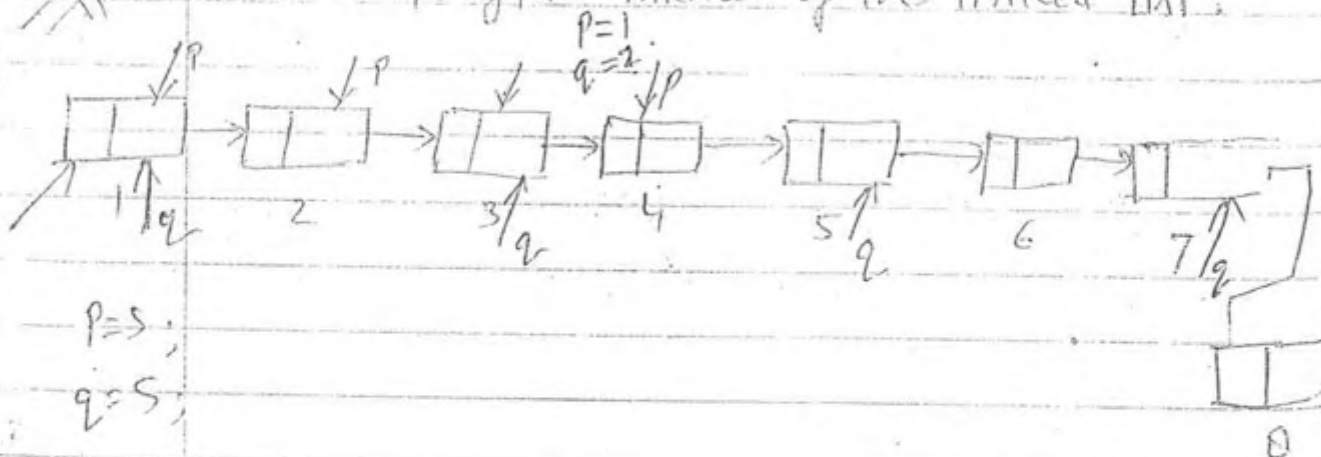
which one of the following statement insert an element  $x$  after position current.

- a)  $current = NewNode(x, current)$
- b)  $current = NewNode(x, current \rightarrow next)$
- c)  $current \rightarrow next = NewNode(x, current)$
- ~~d)  $current \rightarrow next = NewNode(x, current \rightarrow next)$~~



✖

WACP to find middle of the linked list.



Current.

```

P = q = s ;
while (q != null && q->next != null && q->next->next != null)
{
 P = P->next;
 q = q->next->next;
}
return (P)

```

# Consider the fun<sup>n</sup> f defined below.

```

struct item
{
 int data;
 struct item * next;
}

int f (struct item * P)
{
 return ((P == null) || (P->next == null) || (P->data <= P->next->data)
 || f(P->next))
}

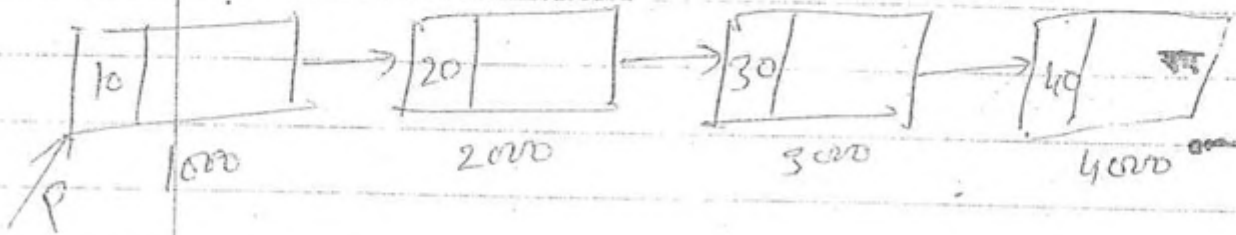
```

for a given linked list P the above function

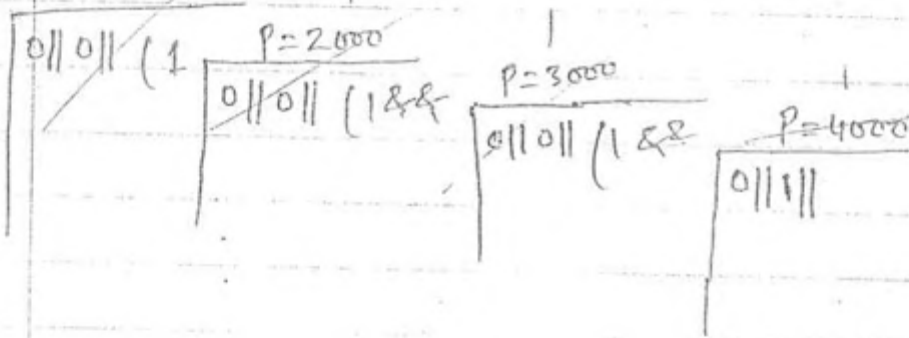


f return 1 iff.

- P is empty (or) exactly 1-element.
- ~~non-decreasing order.~~
- ~~non-increasing order.~~
- none.

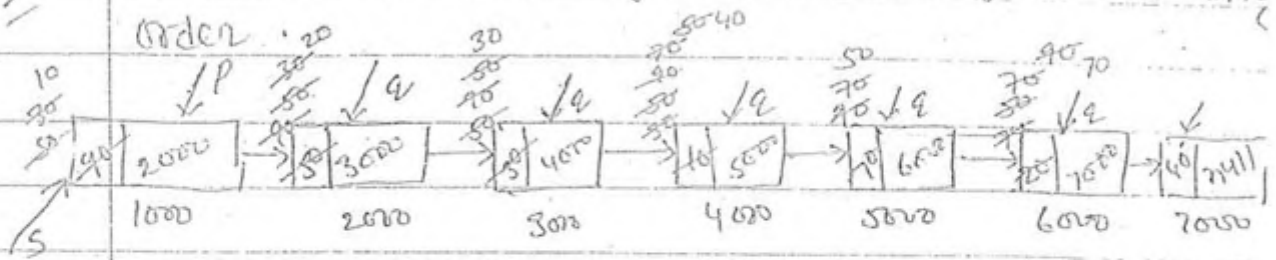


1 = P = 1000





Q WAP to sort the given linked list in ascending order.



Selection Sort  
Sort (struct node \*s)

```

{
 struct node *p, *q;
 p = s;
 while (p != null)
 {
 for (q = p->next; q != null; q = q->next)
 {
 if (p->data > q->data)
 swap (p->data, q->data);
 }
 p = p->next;
 }
}

```

Selection Sort

#8

The following C function takes single linked list of integer as a parameter and rearranges the ~~param~~ elements of the list. The ~~fun~~<sup>fun</sup> is called with the list containing the integer 1, 2, 3, 4, 5, 6, 7 in the given order. What will be the contents of the list after the function completes execution.

Sol<sup>n</sup>

Struct node

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

void rearrange(struct node \*list)

```
{
 struct node *p, *q;
```

```
 int temp;
```

```
 if (!list || !list->next) return;
```

```
 p = list; q = list->next;
```

```
 while (q)
```

```
 {
 temp = p->value;
```

```
 p->value = q->value;
```

```
 q->value = temp;
```

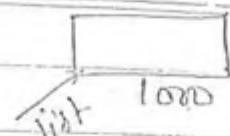
```
 p = q->next;
```

```
 q = p->next;
```

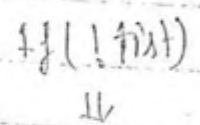
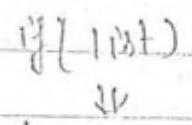
ed list  
arranges

Q/P →

a) 1, 2, 3, 4, 5, 6, 7



b) 2, 3, 4, 5, 6, 7, 1



c) 1, 3, 2, 5, 4, 7, 6

if (list != null)

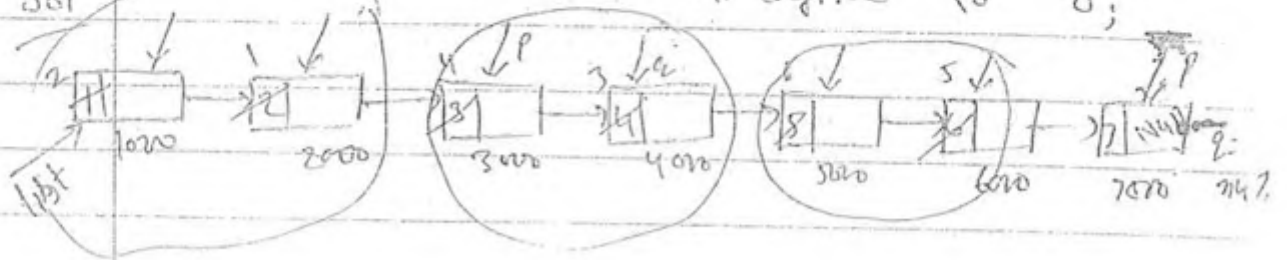
if (list == null)

d) 2, 1, 4, 3, 6, 5, 7

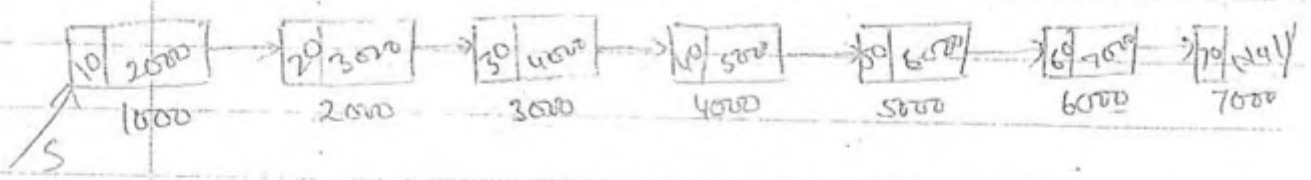
Ans.  
Execution.

Sol<sup>n</sup>

```
#define Null 0;
#define 10 0;
```



#P Q/A/P to Print the data value of node in given linked list in reverse order.



- Sol<sup>n</sup>
- ① Store into an array and Print in reverse order.
  - ② Recursion.
  - ③ Reverse the L.L and print one by one.

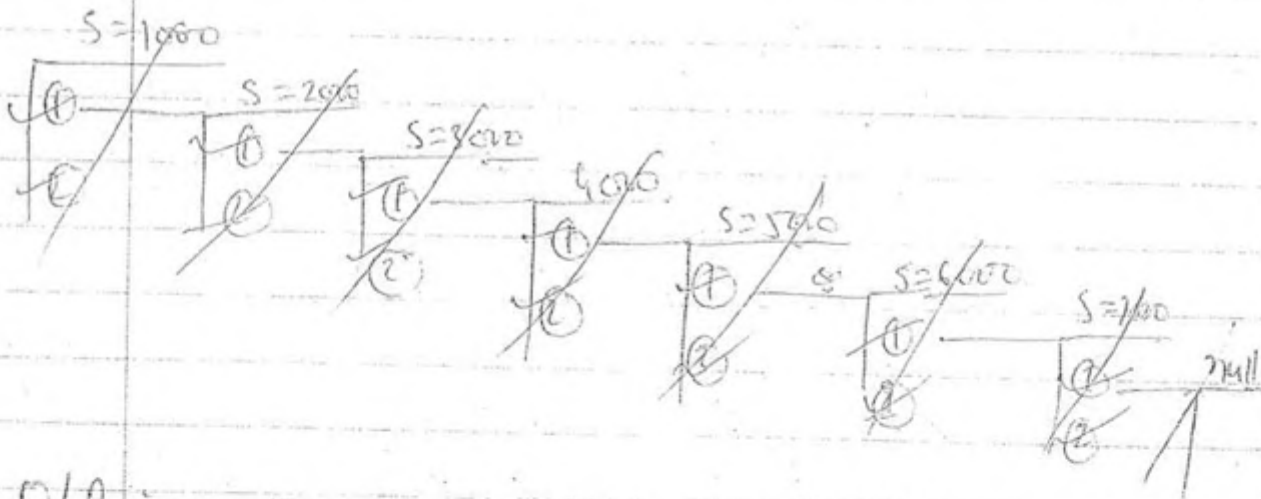
# - Recursion

Print LL reverse (struct node \*s)

```

{
 if (s == null)
 return;
 else
 {
 Print LL Reverse (s->next);
 printf ("%d", s->data);
 }
}

```

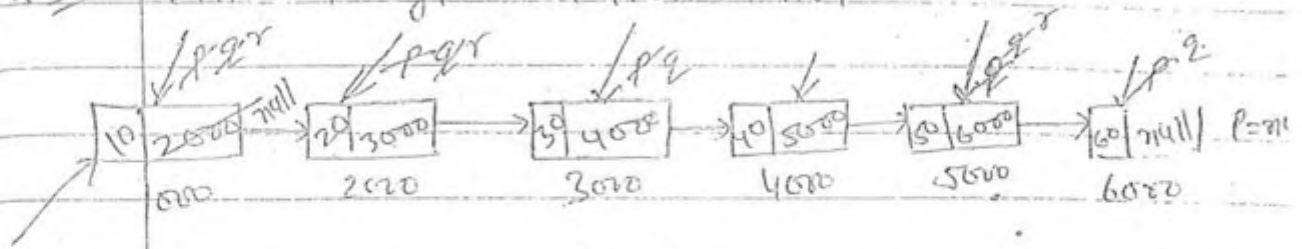


O/p :- 70 60 50 40 30 20 10

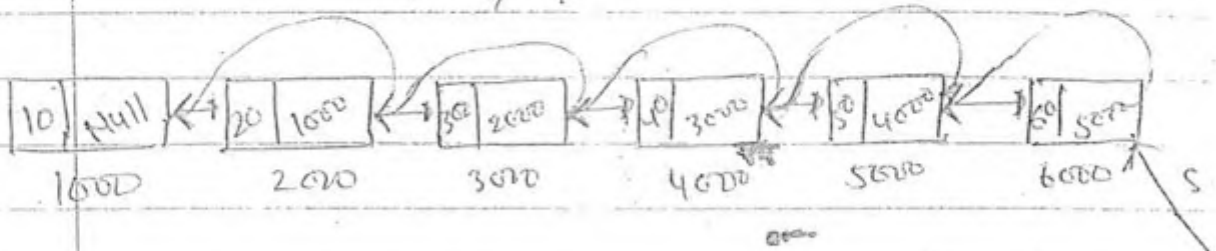
10. Given a linked list these pointers are compulsory - Current, Previous and.

|       |          |  |
|-------|----------|--|
| DELTA | Page No. |  |
|       | Date:    |  |

#P WACP to reverse the given a linked list.



↓ O/P



Reverse of LL (struct node \*s)

```
{ struct node *p, *q, *r;
```

```
 p = s;
```

```
 q = r = null;
```

```
 while (p != null)
```

```
 {
 r = q;
 q = p;
 p = p->next;
 q->next = r;
 }
```

```
 s = q;
```

```
}
```

$T.C \Rightarrow O(n)$   
 $3 \times 2 = 6 \Rightarrow O(1)$



Note: -

Reversing the given linked list require using 3-pointers.

2010  
#P

The following fun<sup>n</sup> takes a single linked list as input arguments. It modify the list by moving the last node to the front of the list and return the modify list. Some parts of the function left blank (meanwhile some where gap is given)

Sol<sup>n</sup>

typedef struct node.

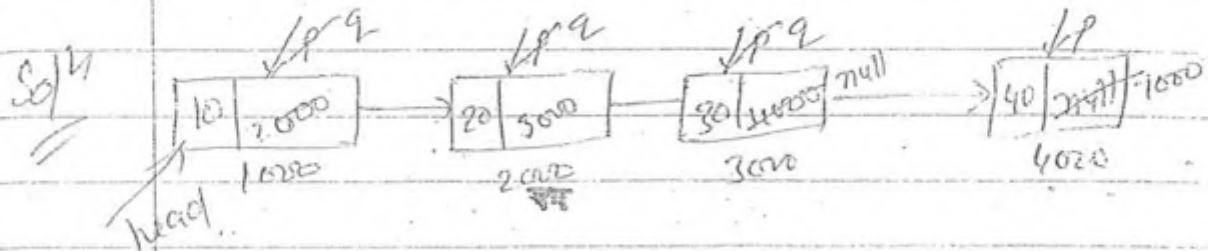
```
{
 int value;
 struct node * next;
} node;
```

node \* modified list (node \* head)

```
{
 node * p, * q;
 if ((head == null) || (head->next == null))
 return (head);
 q = null, p = head;
 while (p->next != Null)
 {
 q = p; p = p->next;
 }
}
```

#P

$q \Rightarrow \text{next} = \text{null};$   $p \Rightarrow \text{next} = \text{head};$   $\text{head} = p;$   
 $\text{return}(\text{head})$



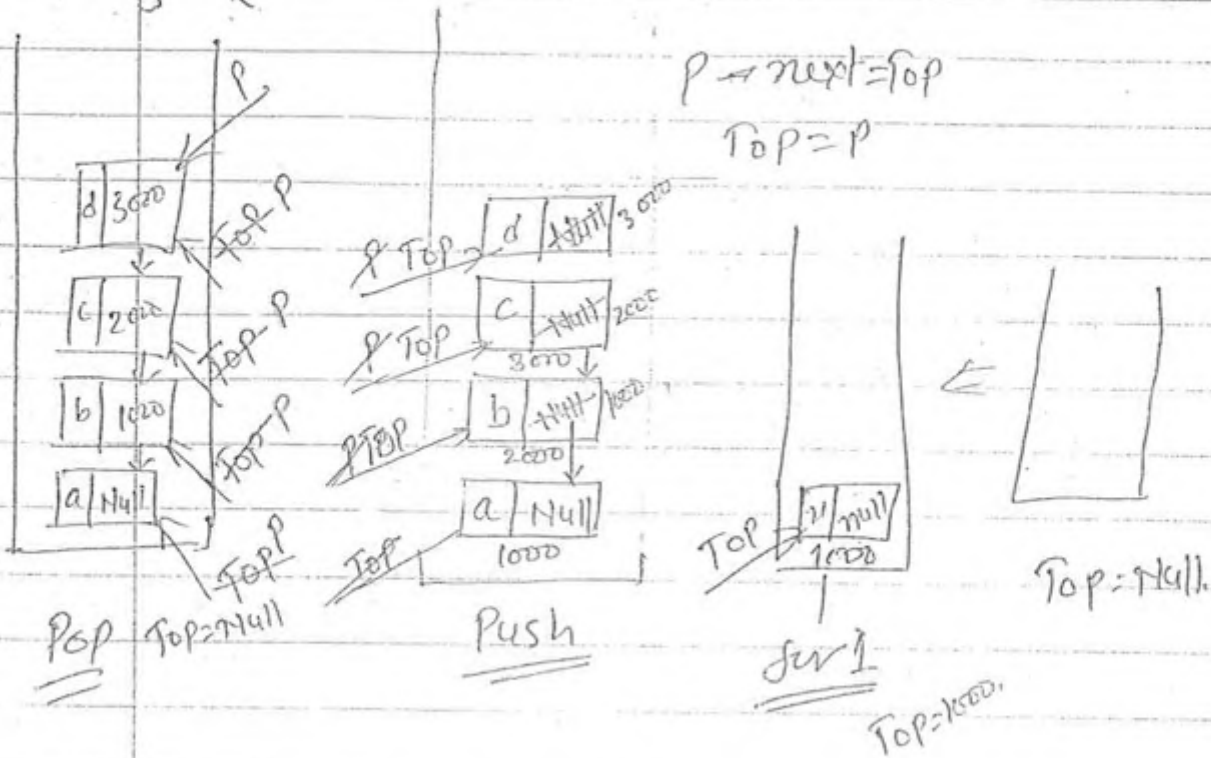
O/P -

- $q = \text{null}, p \Rightarrow \text{next} = \text{head}, \text{head} = p;$
- $q \Rightarrow \text{next} = \text{null}; \text{head} = p; p \Rightarrow \text{next} = \text{head}, \text{head} = p;$
- $\text{head} = p; p \Rightarrow \text{next} = q; q \Rightarrow \text{next} = \text{Null};$
- ~~$q \Rightarrow \text{next} = \text{null}; p \Rightarrow \text{next} = \text{head}; \text{head} = p;$~~

null)) # P WACP to implement linked stack. (ie implemented  
 push to pop operation) means last in first  
 out.



$P = \text{top}$   
 $\text{Top} = \text{Top} \rightarrow \text{next}$   
 $\text{free}(P)$   
 $P = \text{Null}$



```

Push(Struct node *Top , int u)
{
 Struct node *P ;
 P = (struct node *) malloc(Size of (Struct node))
 if (P == Null)
 {
 Printf ("Stack is overflow");
 exit (i);
 }
 P->data = u;
 P->next = Top;
 Top = P;
}

```

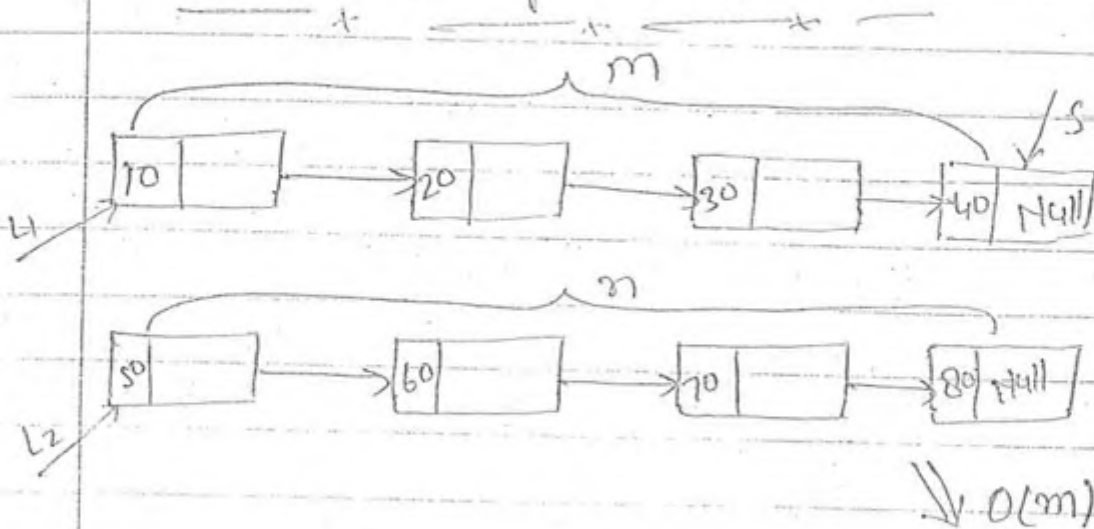
Pop ( struct node \* Top )

```

{
 struct node * p;
 int y;
 if (Top == Null)
 {
 printf ("Stack is underflow");
 exit (1);
 }
 y = Top -> data;
 p = Top; Top = Top -> next;
 free (p); p = Null;
 return (y);
}

```

## Concatenation of 2-Linked list.



if both  $L_1$  and  $L_2$   
are not null  
 $\Rightarrow O(m)$

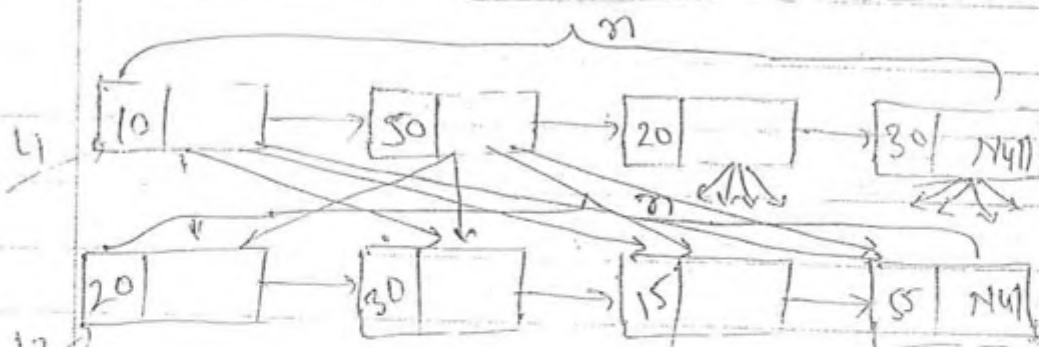
```

S = L1;
while (S->next != null) {
 S = S->next;
}
S->next = L2;
return(L1)

```

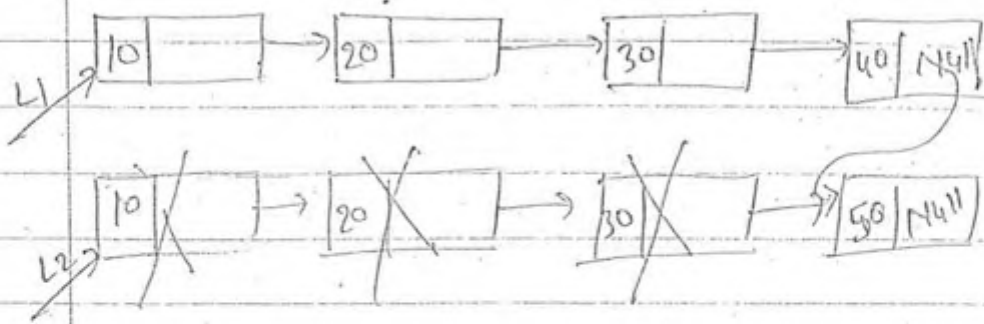
Time complexity:  $O(m)$

## Intersection of 2-Linked-List



$\Rightarrow$  for every element of 1st LL compare total  
2nd list  
 $\Rightarrow O(n^2)$

Union of 2-1431

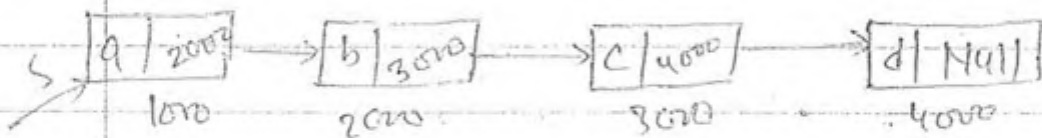


Algo

- (1) Add 1st LL to the ~~another~~ <sup>resulted 3rd linked list</sup> ~~answre~~.
- (2) using 1st LL. remove common element from 2nd linked list.
- (3) Add modified 2nd Linked list to 3rd LL.

Time complexity =  $O(n^2)$ .

Draw back of S. Linked list (linear or single)



- ① Random access is not possible.

|    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|
| 10 | 20 | 30 | 40 | 50 | 60 | 70 |
| 1  | 2  | 3  | 4  | 5  | 6  | 7  |

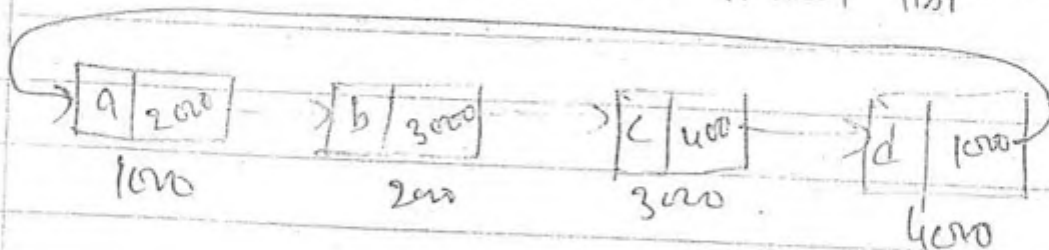
Dis Advantage  
Delete  $\Rightarrow O(n)$   
Insertion  $\Rightarrow O(n)$   
go place  $\Rightarrow O(1)$

Advantage  
Random access.

- ② From Every node we always <sup>move</sup> only one direction. i.e. we can not go back.
- ③ We are not using efficiently last node next part becoz it is always null.

## Circular Single Linked List

If we keep 1st node address in last node next part that single linked list is known as circular linked list.



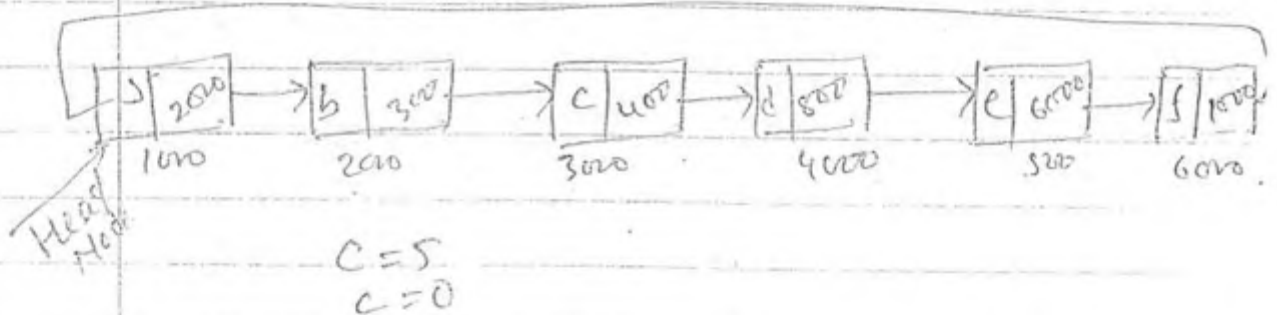
Using CSLL from a given node we can go back also.

### Drawback

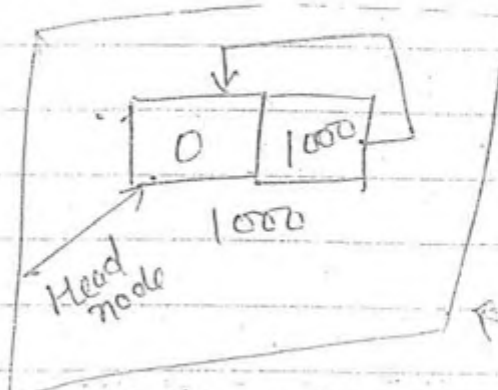
The drawback of CSLL is if will go to infinite loop if we don't write the code properly.

### Head Node

Using headnode we can stop infinite loop in circular linked list.



Using Headnode data part will contain valuable information & like no. of node in the CSLL.

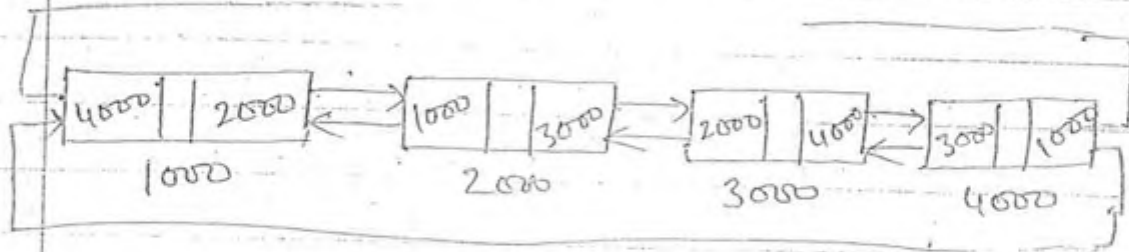


If circular list is empty the diagram look like

drawback  $\Rightarrow$  little bit extra space it will take.



# Double linked list (Double Circular linked list (DCLL))

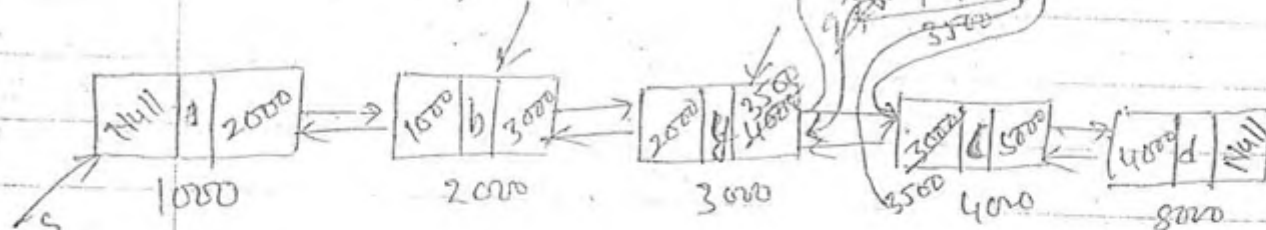


\* The advantage of DLL is if we lost one pointer we will get it back with the help of another pointer.

#P

WACP to insert a node with data  $x$  after a node with data  $y$ . In Double linked list.

$P = S;$



$P = S;$

while (  $P \rightarrow \text{data} \neq y$  &&  $P \rightarrow \text{next} \neq \text{null}$  )

$P = P \rightarrow \text{next};$

if (  $P \rightarrow \text{data} == y$  )

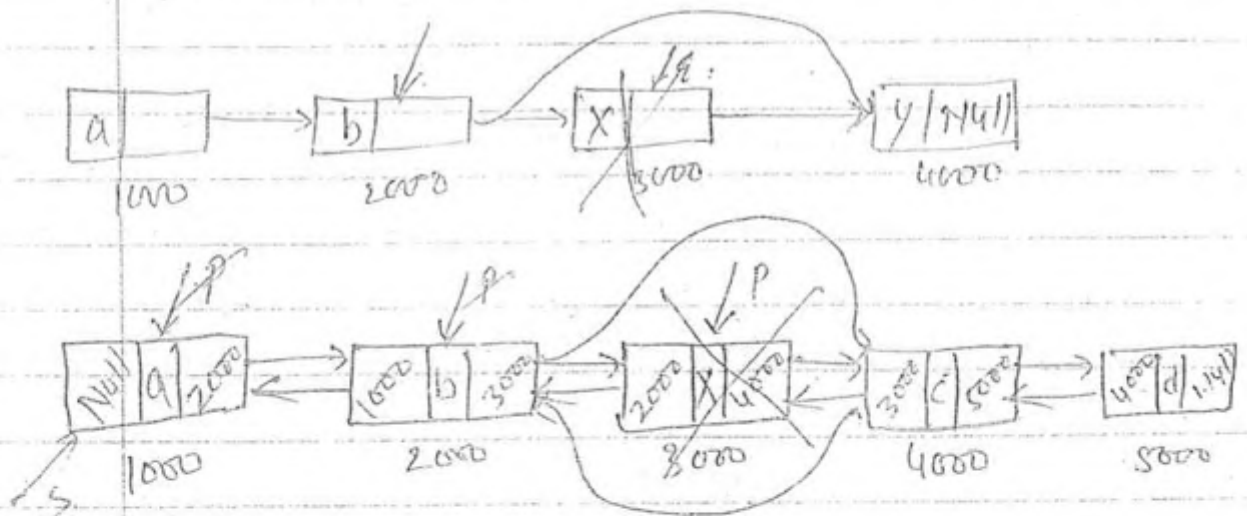


```

 {
 q = Getnode(x);
 q->rp = p->rp;
 p->rp = q;
 q->rp->lp = q;
 q->lp = p;
 }

```

# P WACP to delete a node from Double linked list which contain data as x:



```

 p = s;
 while (p->data != x && p->rp != null)
 p = p->rp;

```

## Polynomial Addition (Using Single LL)

$$P_1 = 10x^5 + 4x^3 + 5x^2 + 15x + 100$$

$$P_2 = 20x^5 + 15x^3 + 10x^2 + 30x + 200$$

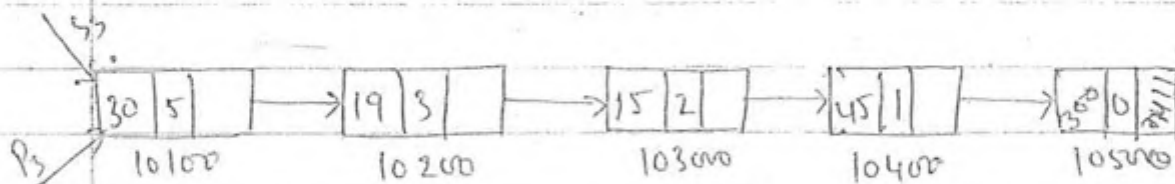
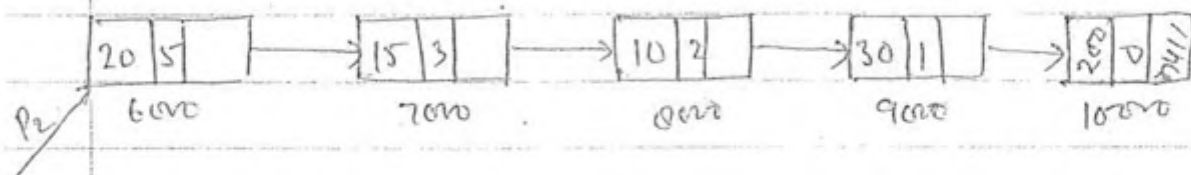
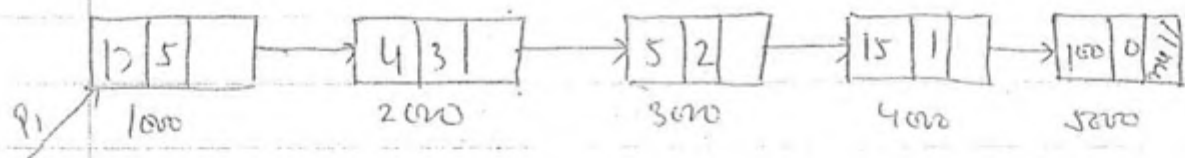
$$P_3 = 30x^5 + 19x^3 + 15x^2 + 45x + 300$$

Struct Poly

```

{
 int coe;
 int pow;
 Struct Poly *next;
};

```



Polyaddition (Struct Poly \*p1, Struct Poly \*p2, Struct Poly\* 

{  $S_3 = p_3;$   $\forall 0(m+n)$   $\forall$  if  $m=n$   
while ( ~~p1~~ p1 != null && p2 != Null ); 0(n)

{ if ( p1 -> pow == p2 -> pow )

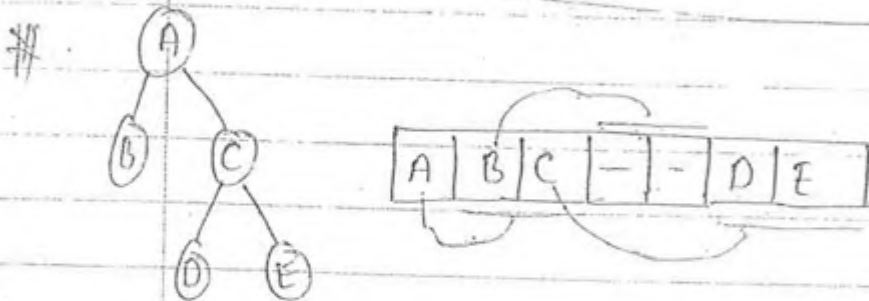
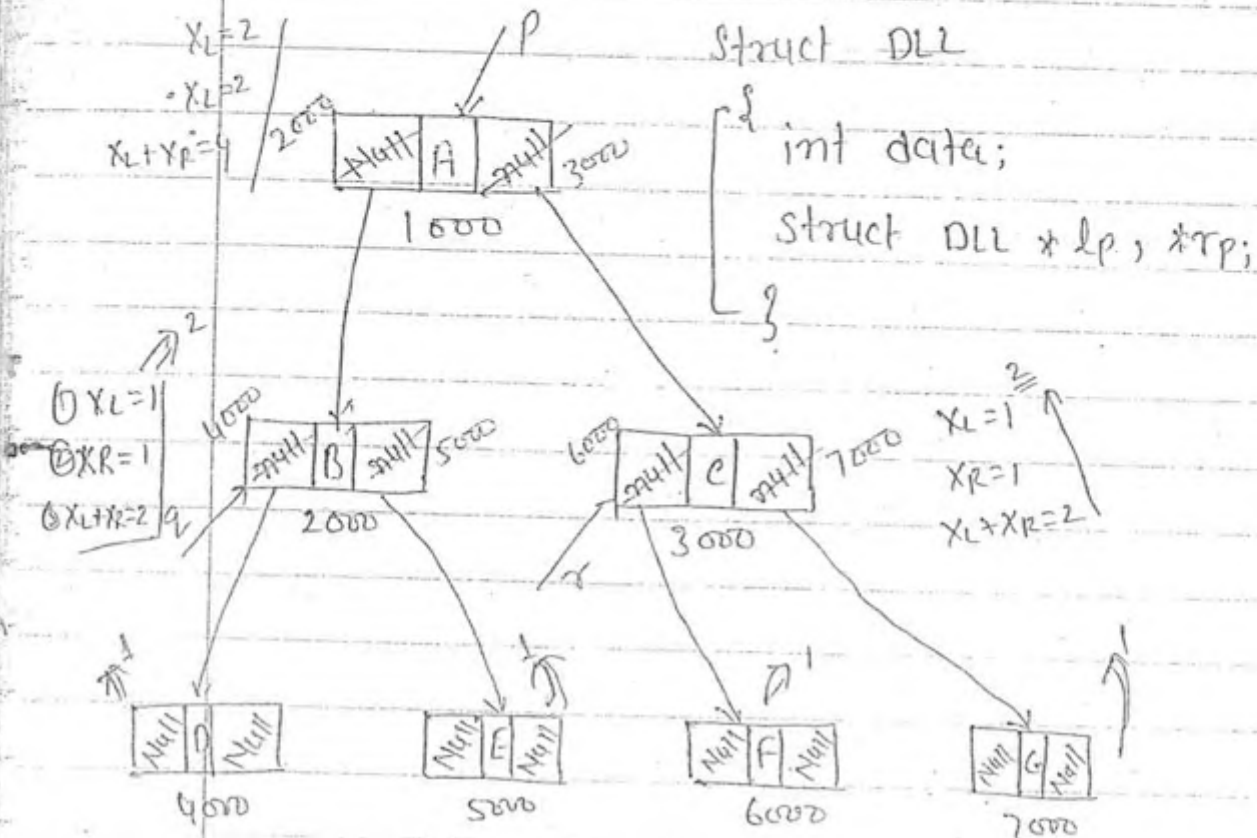
{  
p3 -> pow == p2 -> pow;  
p3 -> coe = p1 -> coe + p2 -> coe;  
p1 = p1 -> next , p2 = p2 -> next;  
p3 = p3 -> next;  
}

else

{ if ( p1 -> pow > p2 -> pow );

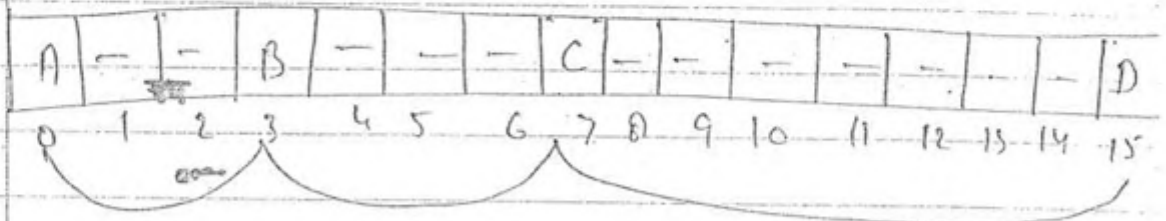
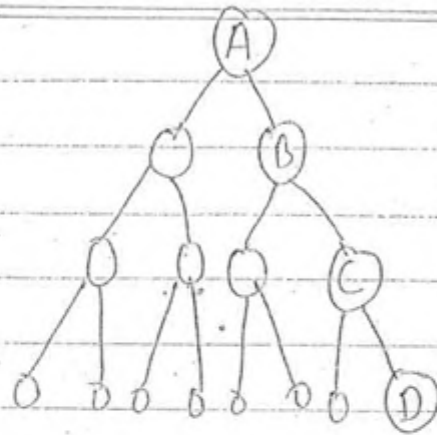
{  
p3 -> pow = p1 -> pow;  
p3 -> coe = p1 -> coe;  
p1 = p1 -> next;  
p3 = p3 -> next;  
}

# Binary Tree



linked list is better approach for Binary tree not a array

#



P

WACP to find no of leaf nodes in the given binary tree.

Numleafnodes(struct Btnode \*p)

```

{
 if (p == null)
 return 0;
 else
 {
 if (p->lp == Null and p->rp == null)
 return 1;
 }
}

```

array

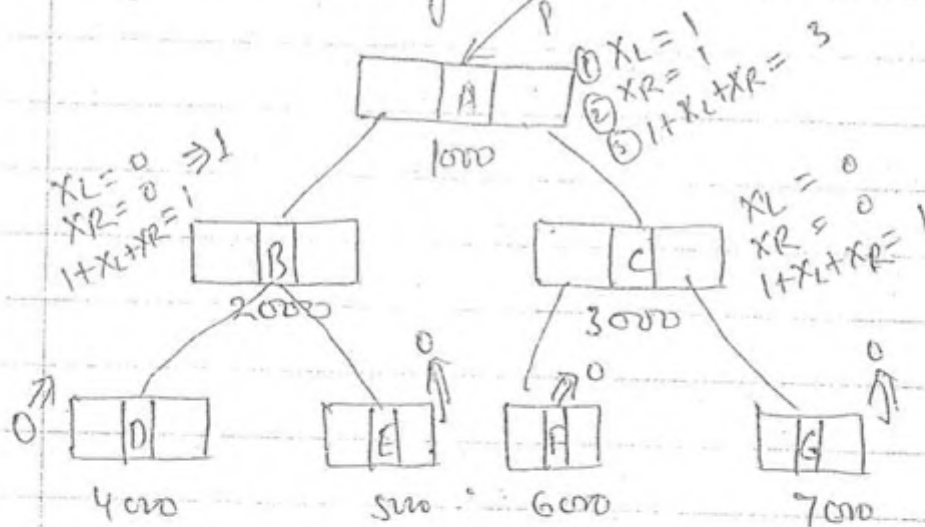
```

else
{
 XL = Numleafnode (P → lp)
 XR = Numleafnode (P → rp)
 return (XL + XR)
}
}
}

```

P

WAP to find no. of internal nodes in the given Binary tree.



leaf node means internal node is zero.



Numinternalnodes (struct. Bt node \*p)

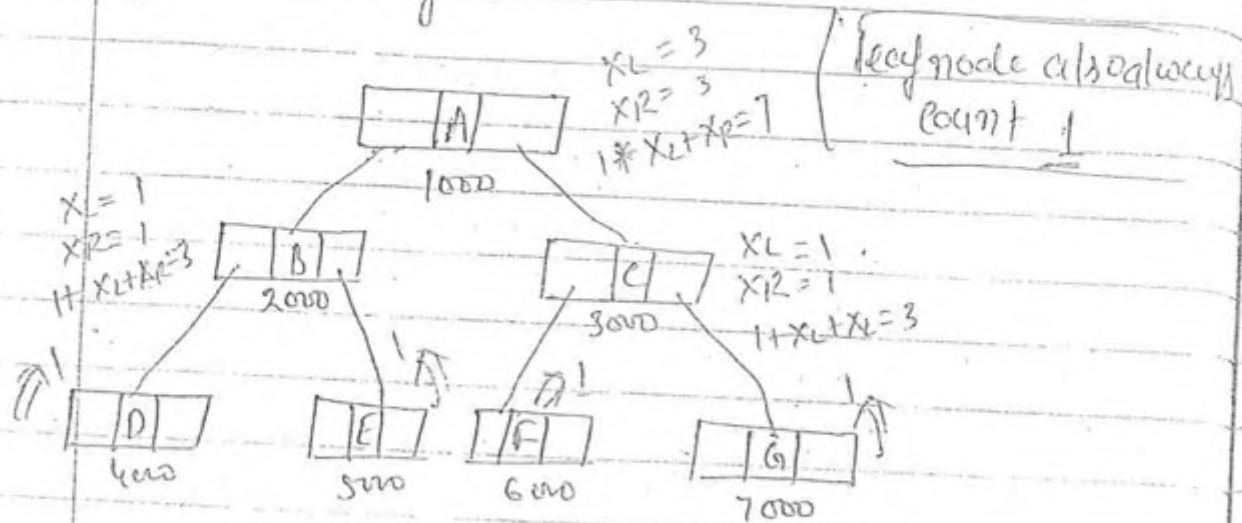
```

{
 if (p == Null)
 return (0);
 else
 {
 if (p->lp == Null && p->rp == Null)
 return (0);
 else
 {
 XL = Numinternalnodes (p->lp)
 XR = Numinternalnodes (p->rp)
 return (1 + XL + XR})
 }
 }
}

```

# 2

WACP to count total no of nodes in the given binary tree.



Total no of nodes (struct. Bt node \* P)

```
if (P == null)
```

```
return 0;
```

```
else
```

```
{
 if (P->lp == null || P->rp == null)
```

```
 return (1);
```

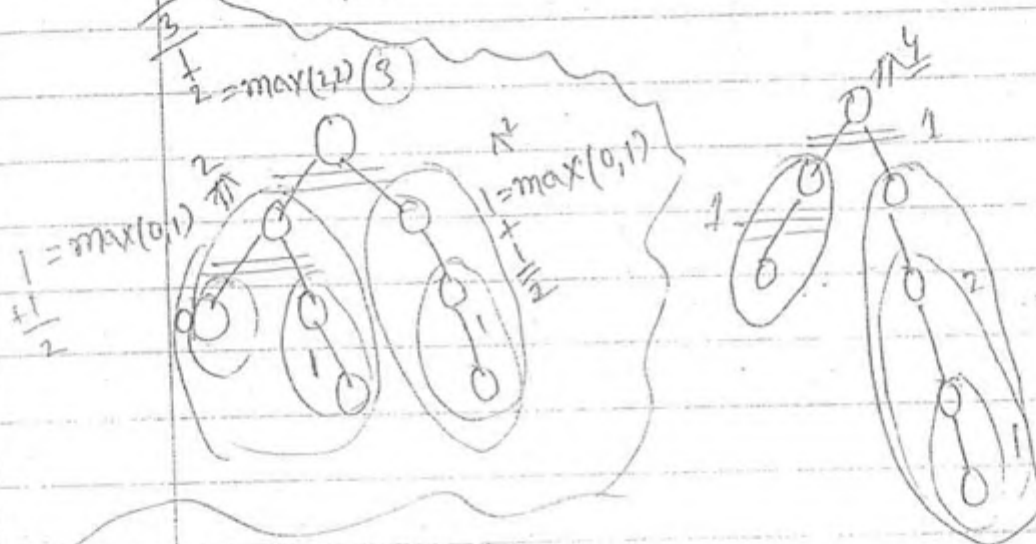
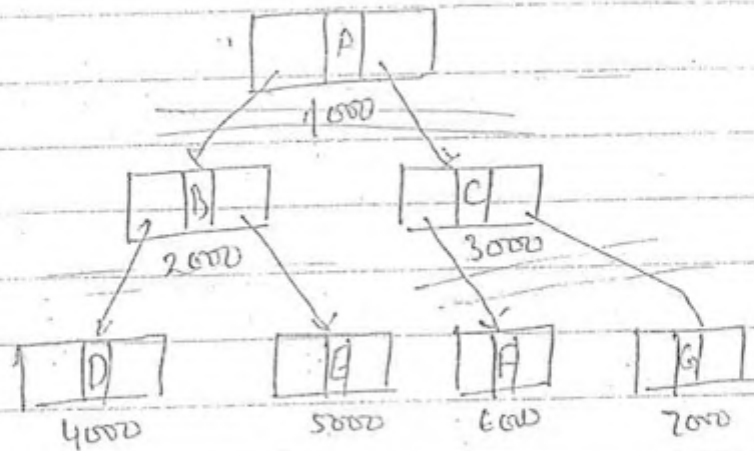
```
else
```

```
{
 XL = total no of nodes (P->lp);
```

```
 XR = total no of nodes (P->rp);
```

```
 return (1 + XL + XR)
```

# 9 WACP to find height of a given binary tree.



Height of Bt ( struct Bt node \*p )

```

{
 if (p == Null)
 return 0;
 else
 {
 if (p->lp == Null && p->rp == Null)
 return 0;
 }
}

```

Date:

{

$XR = \text{Height of BL} (p \rightarrow r_p)$

else

return( $X_{i+1}$ )

3

3

3

WACP to check given binary tree is strict binary tree or not.

SBT

SBT X

11

$(X_L - 2.2 \times R)$

F

1

0

1997, 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020, 2021, 2022, 2023, 2024, 2025, 2026, 2027, 2028, 2029, 2030, 2031, 2032, 2033, 2034, 2035, 2036, 2037, 2038, 2039, 2040, 2041, 2042, 2043, 2044, 2045, 2046, 2047, 2048, 2049, 2050, 2051, 2052, 2053, 2054, 2055, 2056, 2057, 2058, 2059, 2060, 2061, 2062, 2063, 2064, 2065, 2066, 2067, 2068, 2069, 2070, 2071, 2072, 2073, 2074, 2075, 2076, 2077, 2078, 2079, 2080, 2081, 2082, 2083, 2084, 2085, 2086, 2087, 2088, 2089, 2090, 2091, 2092, 2093, 2094, 2095, 2096, 2097, 2098, 2099, 2100, 2101, 2102, 2103, 2104, 2105, 2106, 2107, 2108, 2109, 2110, 2111, 2112, 2113, 2114, 2115, 2116, 2117, 2118, 2119, 2120, 2121, 2122, 2123, 2124, 2125, 2126, 2127, 2128, 2129, 2130, 2131, 2132, 2133, 2134, 2135, 2136, 2137, 2138, 2139, 2140, 2141, 2142, 2143, 2144, 2145, 2146, 2147, 2148, 2149, 2150, 2151, 2152, 2153, 2154, 2155, 2156, 2157, 2158, 2159, 2160, 2161, 2162, 2163, 2164, 2165, 2166, 2167, 2168, 2169, 2170, 2171, 2172, 2173, 2174, 2175, 2176, 2177, 2178, 2179, 2180, 2181, 2182, 2183, 2184, 2185, 2186, 2187, 2188, 2189, 2190, 2191, 2192, 2193, 2194, 2195, 2196, 2197, 2198, 2199, 2200, 2201, 2202, 2203, 2204, 2205, 2206, 2207, 2208, 2209, 2210, 2211, 2212, 2213, 2214, 2215, 2216, 2217, 2218, 2219, 2220, 2221, 2222, 2223, 2224, 2225, 2226, 2227, 2228, 2229, 2230, 2231, 2232, 2233, 2234, 2235, 2236, 2237, 2238, 2239, 2240, 2241, 2242, 2243, 2244, 2245, 2246, 2247, 2248, 2249, 2250, 2251, 2252, 2253, 2254, 2255, 2256, 2257, 2258, 2259, 2260, 2261, 2262, 2263, 2264, 2265, 2266, 2267, 2268, 2269, 2270, 2271, 2272, 2273, 2274, 2275, 2276, 2277, 2278, 2279, 2280, 2281, 2282, 2283, 2284, 2285, 2286, 2287, 2288, 2289, 2290, 2291, 2292, 2293, 2294, 2295, 2296, 2297, 2298, 2299, 2300, 2301, 2302, 2303, 2304, 2305, 2306, 2307, 2308, 2309, 2310, 2311, 2312, 2313, 2314, 2315, 2316, 2317, 2318, 2319, 2320, 2321, 2322, 2323, 2324, 2325, 2326, 2327, 2328, 2329, 2330, 2331, 2332, 2333, 2334, 2335, 2336, 2337, 2338, 2339, 2340, 2341, 2342, 2343, 2344, 2345, 2346, 2347, 2348, 2349, 2350, 2351, 2352, 2353, 2354, 2355, 2356, 2357, 2358, 2359, 2360, 2361, 2362, 2363, 2364, 2365, 2366, 2367, 2368, 2369, 2370, 2371, 2372, 2373, 2374, 2375, 2376, 2377, 2378, 2379, 2380, 2381, 2382, 2383, 2384, 2385, 2386, 2387, 2388, 2389, 2390, 2391, 2392, 2393, 2394, 2395, 2396, 2397, 2398, 2399, 2400, 2401, 2402, 2403, 2404, 2405, 2406, 2407, 2408, 2409, 2410, 2411, 2412, 2413, 2414, 2415, 2416, 2417, 2418, 2419, 2420, 2421, 2422, 2423, 2424, 2425, 2426, 2427, 2428, 2429, 2430, 2431, 2432, 2433, 2434, 2435, 2436, 2437, 2438, 2439, 2440, 2441, 2442, 2443, 2444, 2445, 2446, 2447, 2448, 2449, 2450, 2451, 2452, 2453, 2454, 2455, 2456, 2457, 2458, 2459, 2460, 2461, 2462, 2463, 2464, 2465, 2466, 2467, 2468, 2469, 2470, 2471, 2472, 2473, 2474, 2475, 2476, 2477, 2478, 2479, 2480, 2481, 2482, 2483, 2484, 2485, 2486, 2487, 2488, 2489, 2490, 2491, 2492, 2493, 2494, 2495, 2496, 2497, 2498, 2499, 2500, 2501, 2502, 2503, 2504, 2505, 2506, 2507, 2508, 2509, 2510, 2511, 2512, 2513, 2514, 2515, 2516, 2517, 2518, 2519, 2520, 2521, 2522, 2523, 2524, 2525, 2526, 2527, 2528, 2529, 2530, 2531, 2532, 2533, 2534, 2535, 2536, 2537, 2538, 2539, 2540, 2541, 2542, 2543, 2544, 2545, 2546, 2547, 2548, 2549, 2550, 2551, 2552, 2553, 2554, 2555, 2556, 2557, 2558, 2559, 2560, 2561, 2562, 2563, 2564, 2565, 2566, 2567, 2568, 2569, 2570, 2571, 2572, 2573, 2574, 2575, 2576, 2577, 2578, 2579, 2580, 2581, 2582, 2583, 2584, 2585, 2586, 2587, 2588, 2589, 2590, 2591, 2592, 2593, 2594, 2595, 2596, 2597, 2598, 2599, 2600, 2601, 2602, 2603, 2604, 2605, 2606, 2607, 2608, 2609, 2610, 2611, 2612, 2613, 2614, 2615, 2616, 2617, 2618, 2619, 2620, 2621, 2622, 2623, 2624, 2625, 2626, 2627, 2628, 2629, 2630, 2631, 2632, 2633, 2634, 2635, 2636, 2637, 2638, 2639, 2640, 2641, 2642, 2643, 2644, 2645, 2646, 2647, 2648, 2649, 2650, 2651, 2652, 2653, 2654, 2655, 2656, 2657, 2658, 2659, 2660, 2661, 2662, 2663, 2664, 2665, 2666, 2667, 2668, 2669, 2670, 2671, 2672, 2673, 2674, 2675, 2676, 2677, 2678, 26

Strict binarytree (Struct Btnode \*p)

{

if (p == Null)

return (True);

else

{

if (p->lp == Null && p->rp == Null)

return (True)

else

{

if (p->lp != null && p->rp != null)

return (Strictbinarytree (p->lp) && Strictbinarytree (p->rp))

else

return (False)

}

}

}

# Q

If we apply the code below on B.S.T. What it will return - ?

```
int SomeValue (struct node *node)
{
 struct node *current = node;
 while (current -> left != null)
 current = current -> left;
 return (current -> data);
}
```

Sol<sup>n</sup>

B.S.T.

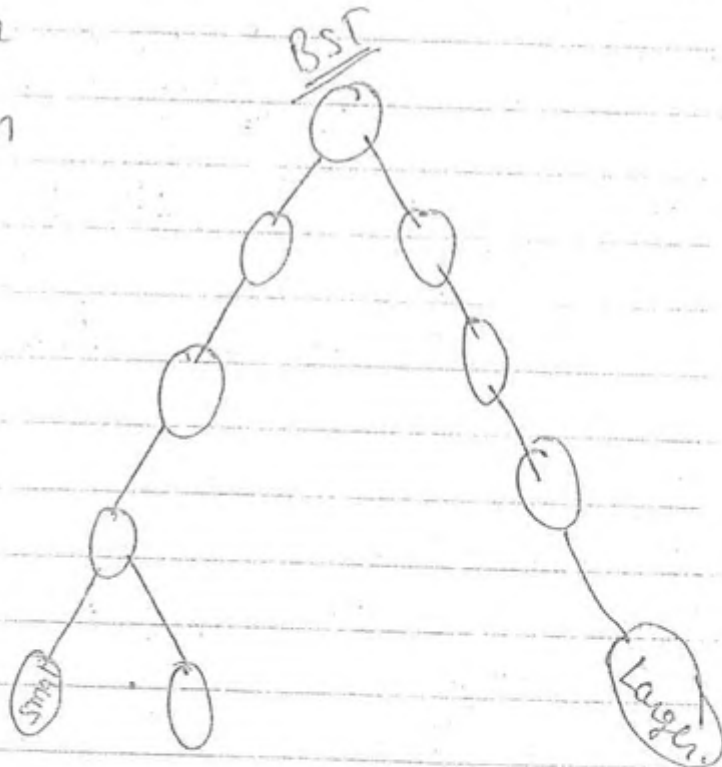
↓

left most value

⇒ Minimum

T.C  $\Rightarrow O(n)$

left or right worst case  $= O(n)$





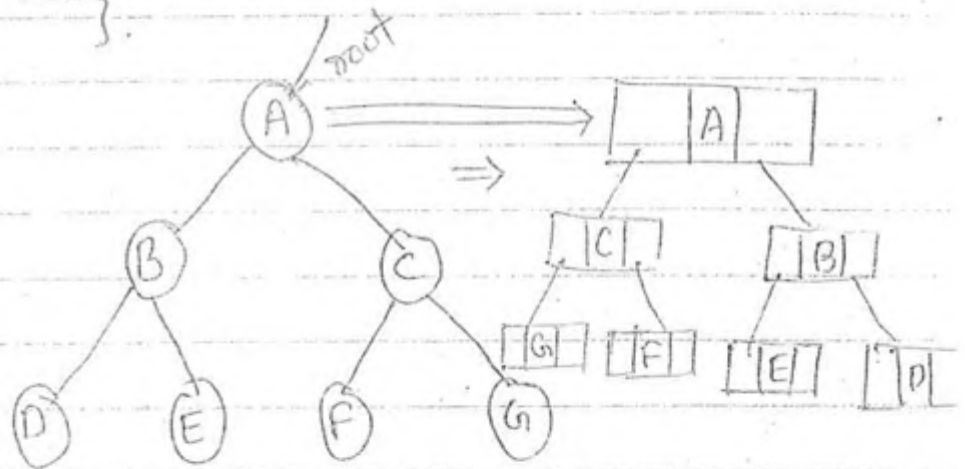
#P

myNode \* copy(myNode \* root)

```

{
 myNode * temp;
 if (root == null) return (null);
 temp = (myNode *) malloc(size of (myNode));
 temp->value = root->value;
 temp->left = copy(root->right);
 temp->right = copy(root->left);
 return (temp);
}

```



above code will create mirror image for given binary tree.

Laigh

Q. A ds is comprised of node each of which has exactly 2-pointers to other nodes with no null pointers. The following c program is to be used to count the no of nodes in the given B.B. It has a mark field. ~~assume~~ to be initially zero for all nodes. There is a statement missing for that code find it.

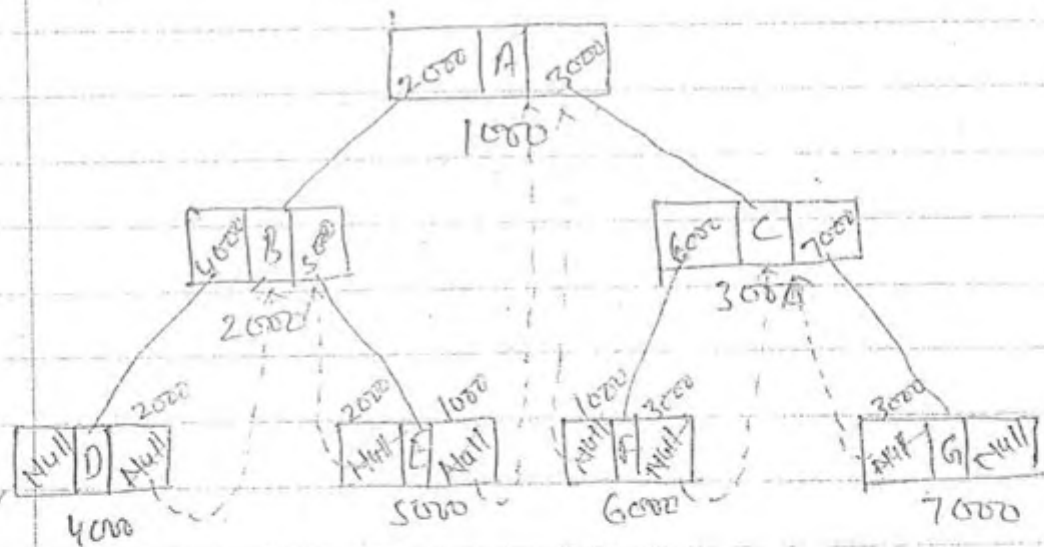
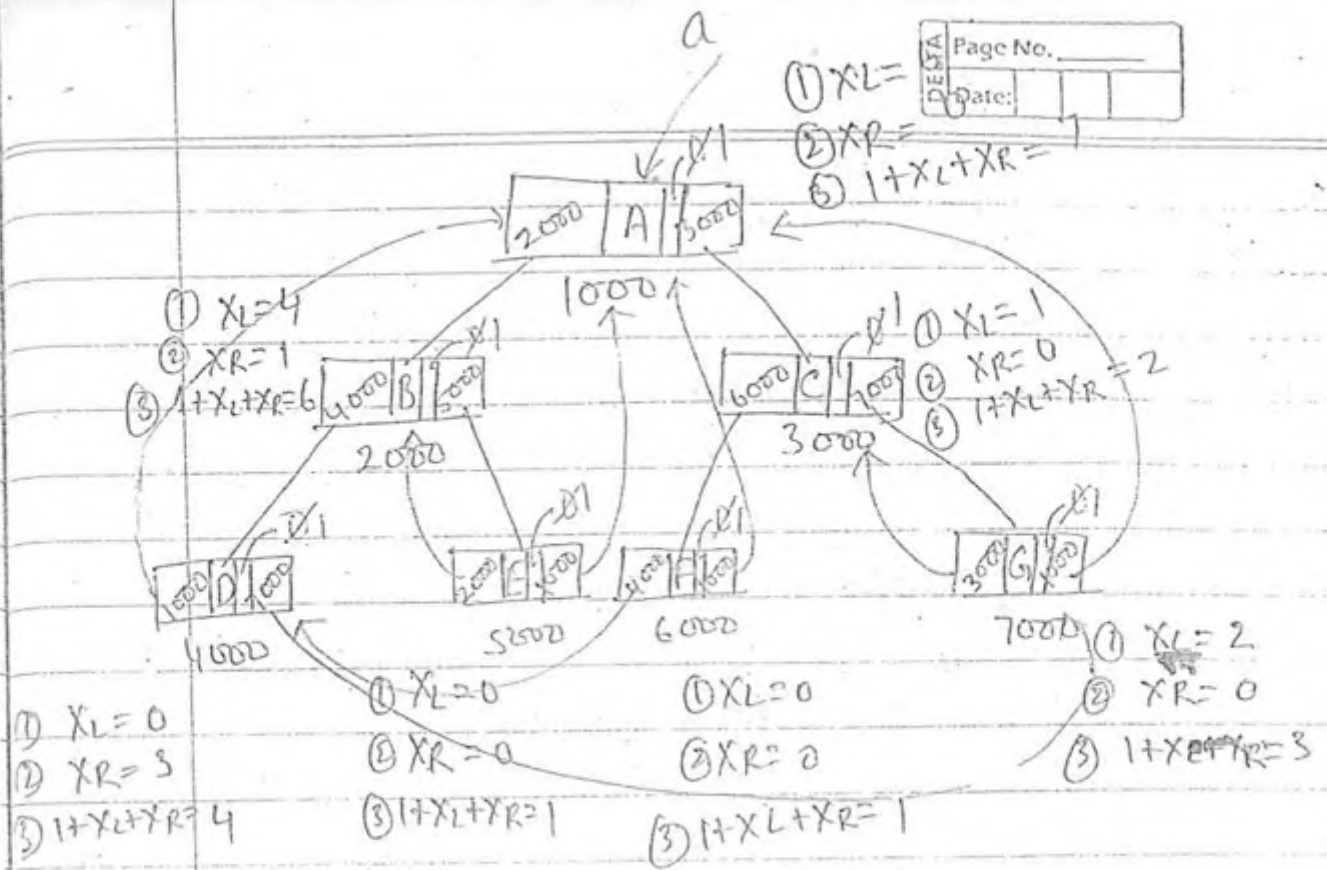
Struct test

```
{
 int info, mark;
 struct test *p, *q;
};
```

int node count ( struct test \*a)

```
{
 if (a -> mark) return(0);
 else
 {
 a -> mark = 1
 return (node count (a -> p) + node
 count (a -> q) + 1);
 }
}
```

- ① XL = 0  
② XR = 3  
③ 1 + XL + XR



Drawback of leaf nodes left pointer and right pointer always null.

We cannot use that space efficiently.

InOrder: - D B E A F C G

Inorder Successor for D

Inorder Predecessor

#P

Inorder Threaded Binary tree.



Inorder Successor  $\Rightarrow$  Right link  
Inorder predecessor  $\Rightarrow$  left link } leaf nodes.

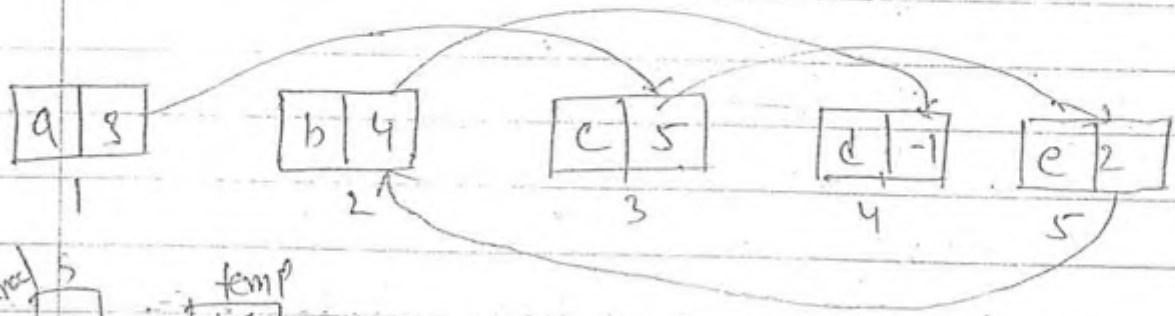
O/P:

# How Implementing L.L. Using Array ?

| data |   |
|------|---|
| 1    | a |
| 2    | b |
| 3    | c |
| 4    | d |
| 5    | e |

| link |    |
|------|----|
| 1    | 3  |
| 2    | 4  |
| 3    | 5  |
| 4    | -1 |
| 5    | 2  |

#



Starting address   
 address

temp   
 1 2 3 4 5 -1

#2 WACP to print data of every node in the given linked list using array representation.

display (data, link, s)

```

{
 int temp;
 temp = s;

 while (temp != -1)
 {
 printf("%d", data[temp]);
 temp = link[temp];
 }
}
o/p:

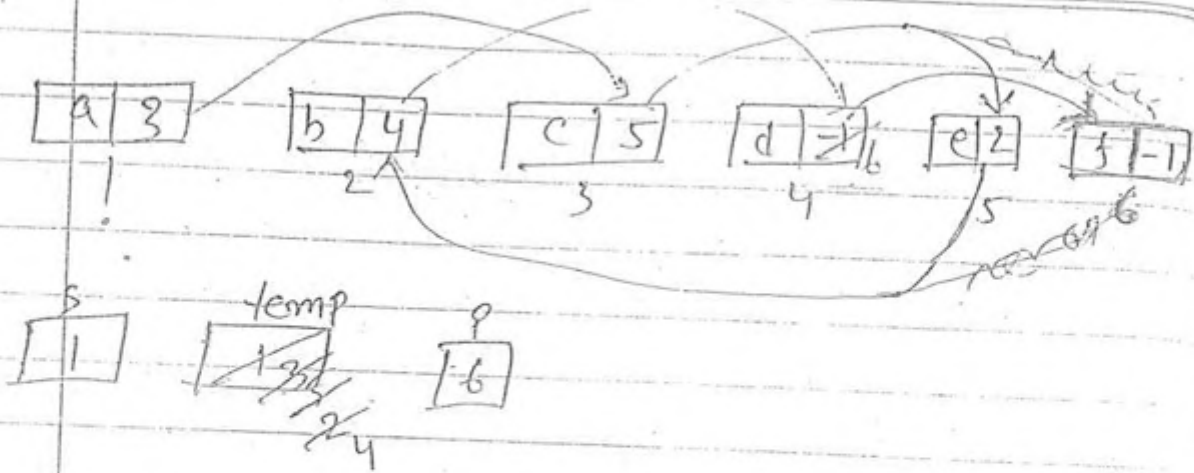
```

o/p! - acebd

# WACP to add a node with data X, in the given linked list using array representation at the end.

| data |   |
|------|---|
| 1    | a |
| 2    | b |
| 3    | c |
| 4    | d |
| 5    | e |
| 6    | f |

| link |    |
|------|----|
| 1    | 3  |
| 2    | 4  |
| 3    | 5  |
| 4    | -1 |
| 5    | 2  |
| 6    | -1 |



add x at end (data, link, s, p)

```
{ int temp;
```

```
temp = s;
```

```
}
```

```
while (link[temp] != -1)
```

```
temp = link[temp]
```

```
link[temp] = p;
```

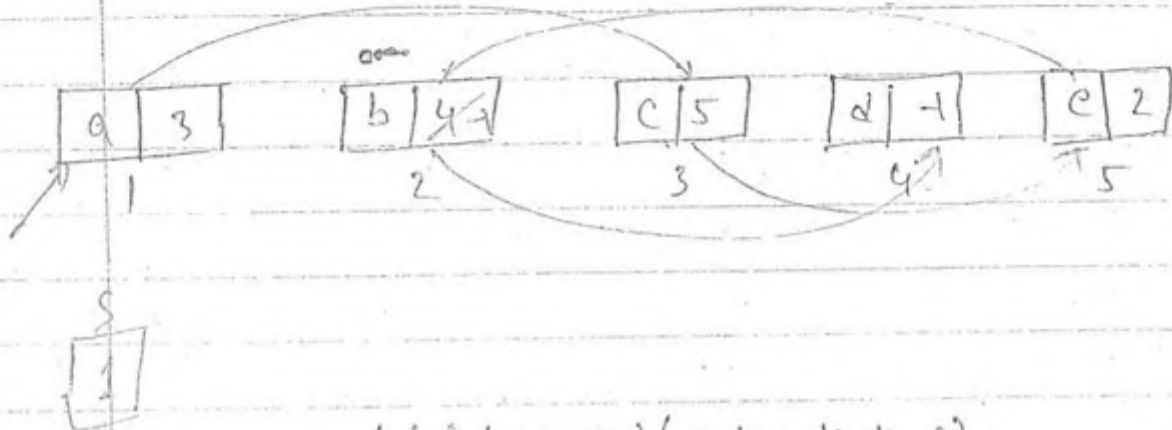
```
};
```



WACP to delete a node from end of the linked list in array representation.

| Data |   |
|------|---|
| 1    | a |
| 2    | b |
| 3    | c |
| 4    | d |
| 5    | e |

| link |    |
|------|----|
| 1    | 3  |
| 2    | 4  |
| 3    | 5  |
| 4    | -1 |
| 5    | 2  |



deletefromend(data, link, s)

```

{
 int t2, t1 = s;
 - if (s == -1)] 0-element
 empty L.L.

```

```

 if (Link[s] == -1)] 1-element
 s = -1

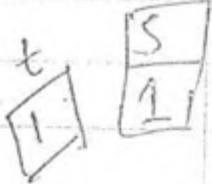
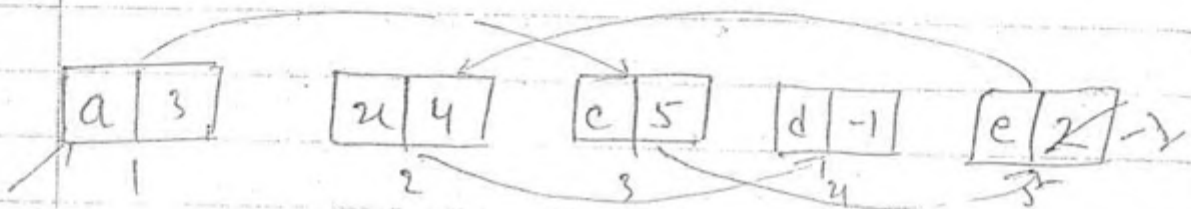
```

while (Link[t1] != -1)

more than one element { t2 = t1; t1 = link(t1); link[t2] = -1;

# WACP to delete a node which contain data as 'u' in the given linked list in the array representation.

| data |   | link |    |
|------|---|------|----|
| 1    | a | 1    | 3  |
| 2    | u | 2    | 4  |
| 3    | c | 3    | 5  |
| 4    | d | 4    | -1 |
| 5    | e | 5    | 2  |



```

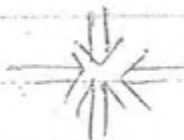
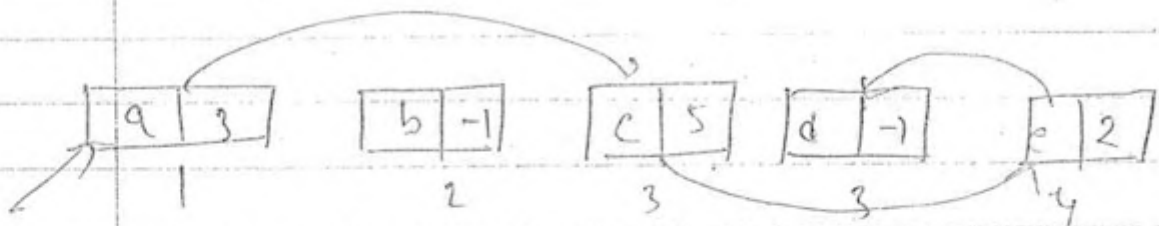
delete from end (data, link, s)
{
 int t1, t2;
 t1 = s;
 do {
 if (s == -1)
 empty;
 if (data[s] == 'u')
 {
 t = s;
 s = link[s];
 link[t] = -1;
 }
 }
}

```

```

else
{
 while (link[t1] != u && link[t1] != -1)
 {
 t2 = t1;
 t1 = link[t1];
 }
 if (data[t1] == u)
 {
 link[t2] = -link[t1];
 link[t1] = -1;
 }
 else
 u is not there.
}
}

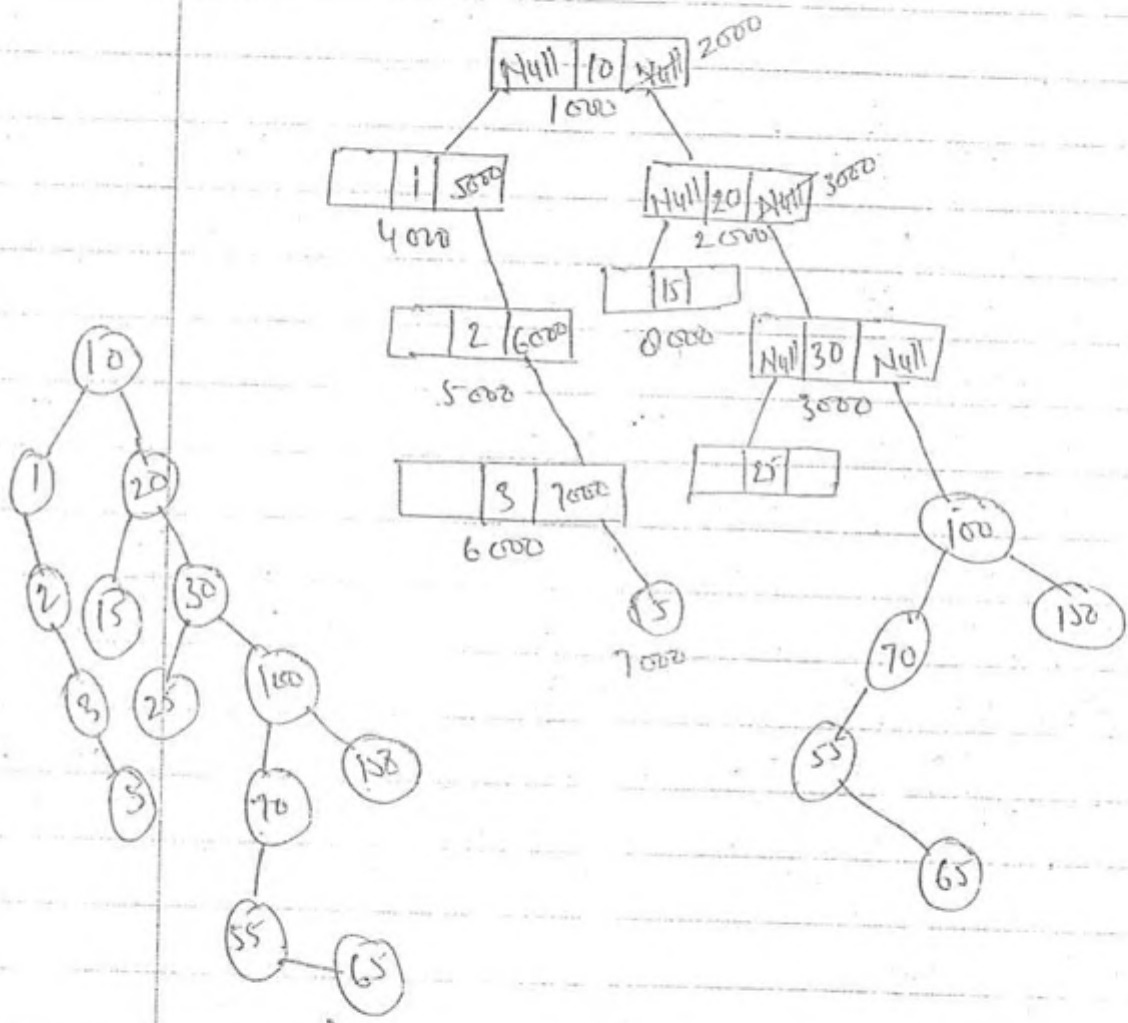
```



# Binary Search Tree

## Insert

Eg 10, 20, 30, 1, 2, 3, 5, 15, 25, 100, 150, 70, 55, 65.



## Insert Algo

① Find correct place of the element to be inserted.

$$\Rightarrow O(n)$$

$$\Rightarrow \Omega(1)$$

$$\Rightarrow \text{Avg is } O(\log n)$$

② Insert the element by changing pointers.

$$\Rightarrow O(1)$$

$$\overbrace{O(n), \Omega(1), O(\log n)}^{\text{Avg}}$$

## Delete

① 0 - Children.

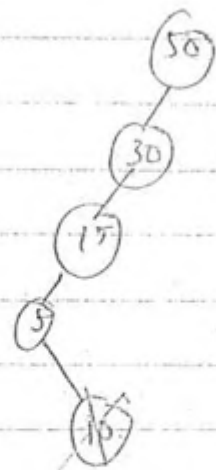
② 1 - children.

③ 2 - children.

$$O(1)$$

①

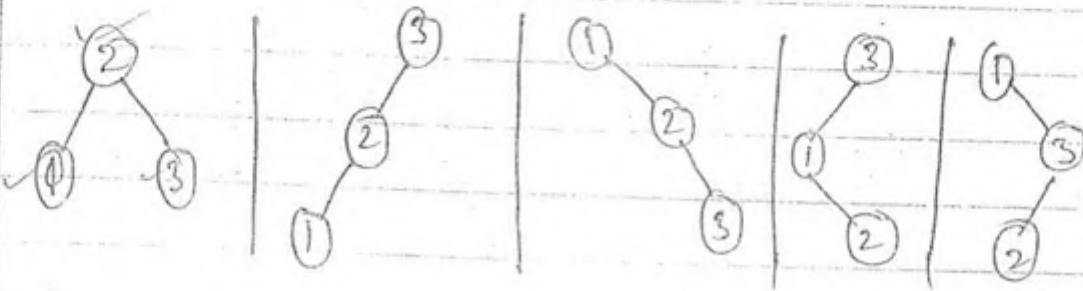
$q \rightarrow rp = \text{null}$   
 $\text{free}(p)$   
 $p = \text{Null}$



# AVL Trees (Adelson, Velski, Landis)

- ① Binary Search Tree.
  - ② Balanced property.
- } maximum  $\log_2 n$

eg  $n = 3 (1, 2, 3)$ .



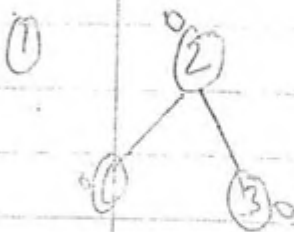
Balance factor: -  $\left| H(LST) - H(RST) \right|$

BST

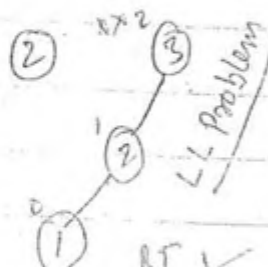


at every node

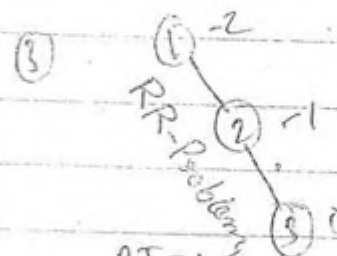
balance factor ( 0 or 1 or -1 )



BST ✓  
BST ✓  $O(\log n)$   
AVL ✓



BST ✓  
BST ✓  
AVL X  
 $O(n)$

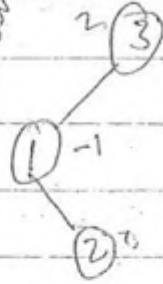


BST = ✓  
BST ✓  
AVL X  
 $O(n)$



(4)

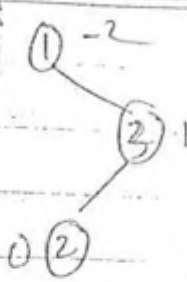
L-R Problem



BT ✓  
BST ✓  
AVL X  $O(n)$

(5)

R-L Problem



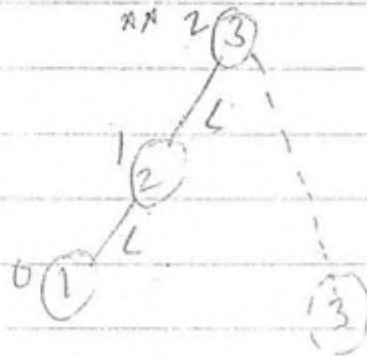
BT ✓  
BST ✓  
 $O(n)$  AVL X

(1)

LL Problem

(rotate top half part right)

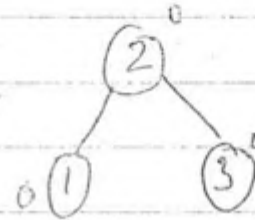
forcefully



AVL X

Inorder = 1, 2, 3

⇒



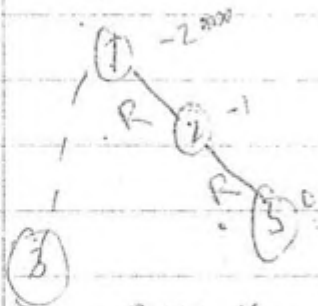
AVL ✓

Inorder = 1, 2, 3

(2)

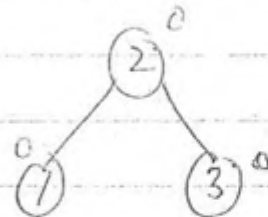
RR Problem (rotate top half part left)

forcefully



AVL X

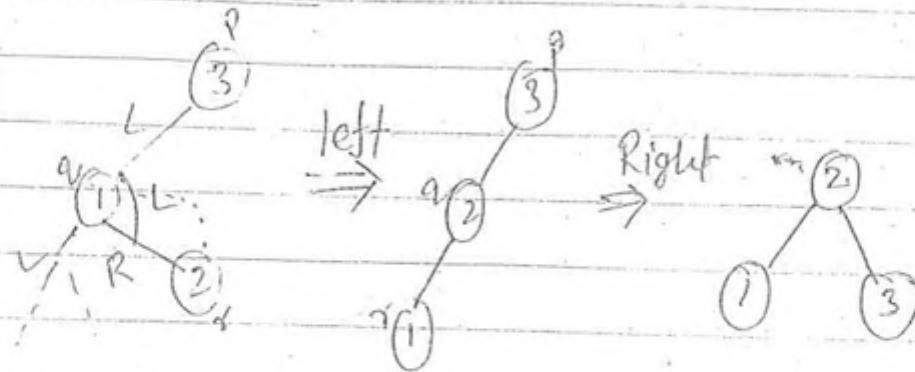
Inorder = 1, 2, 3



AVL ✓

Inorder = 1, 2, 3

### (3) LR Problem

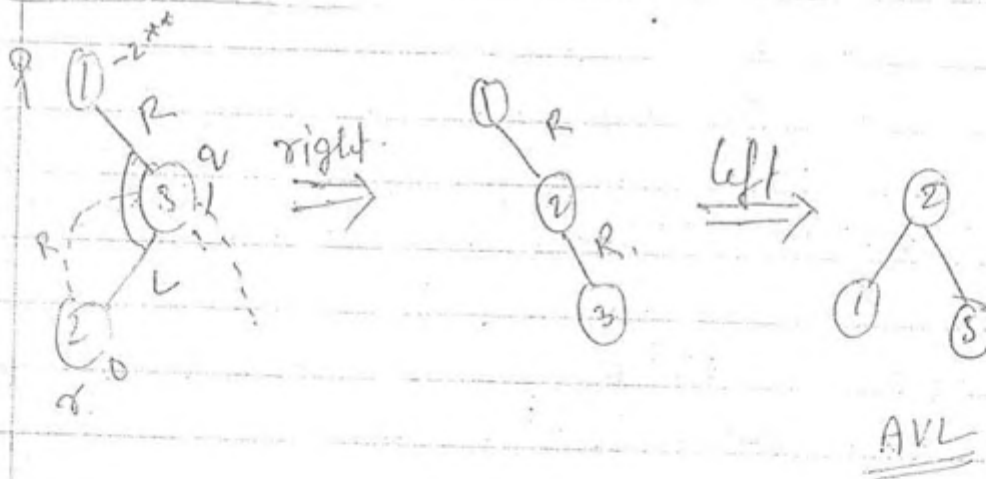


$p \rightarrow lp = r$   
 $r \rightarrow lp = q$

$q \rightarrow rp = p$   
 $r \rightarrow lp = null$

AVL

### (4) RL Problem



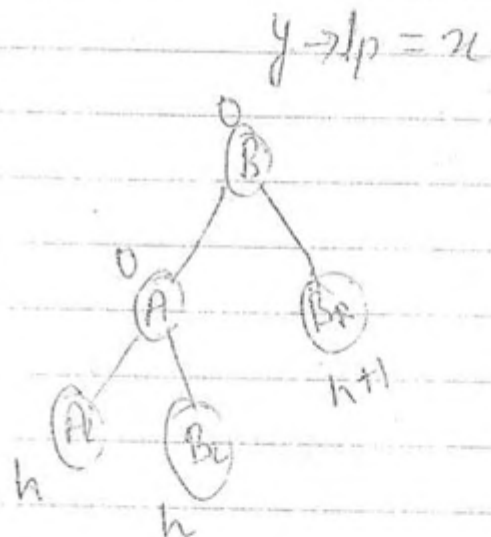
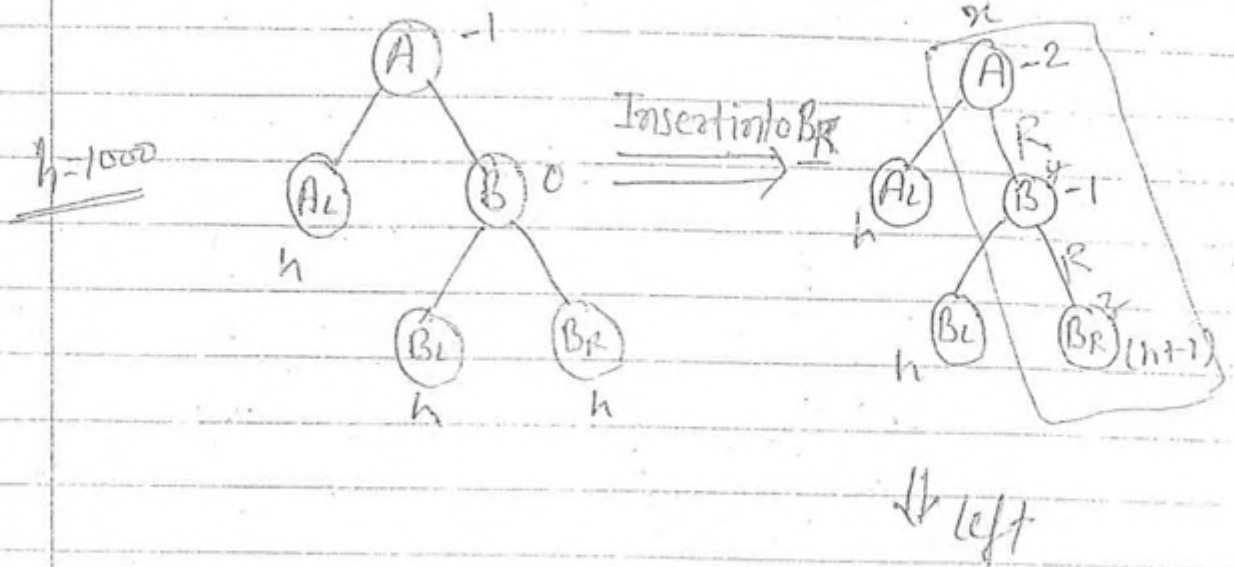
$p \rightarrow rp = r$   
 $r \rightarrow rp = q$

AVL

All of the above four problem takes  $O(1)$

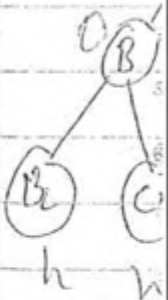
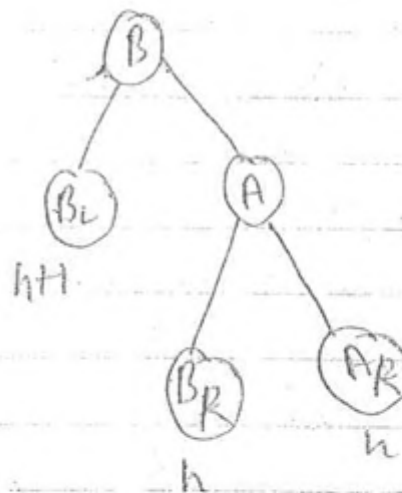
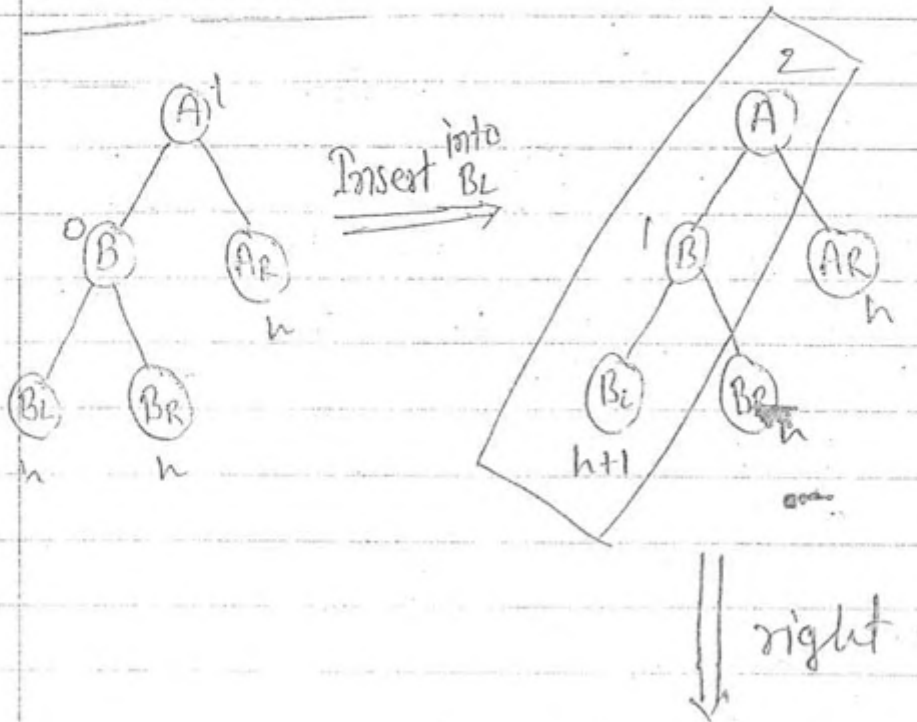
# Insertion

## RR- Problem

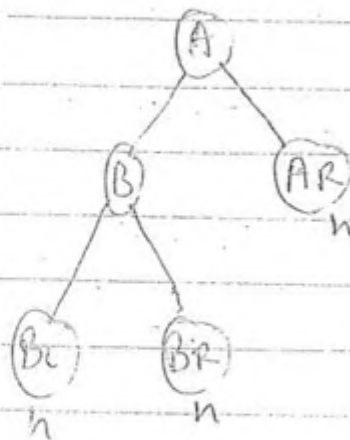


# Insertion

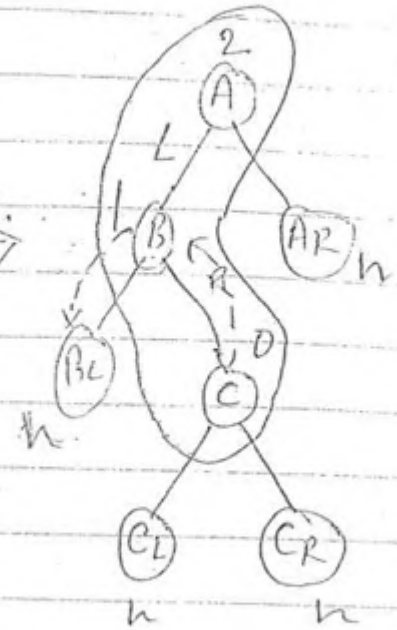
## LL Problem



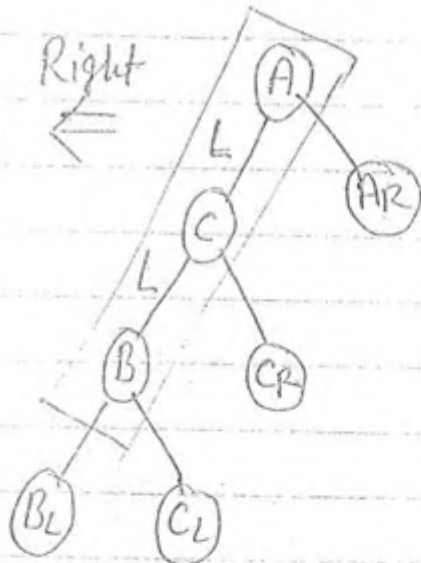
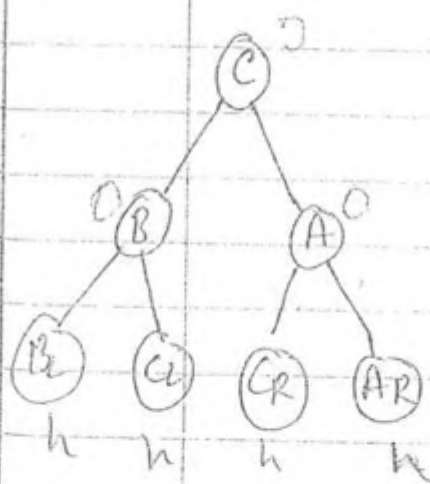
# Insertion LR Problem.



Insert into BR.



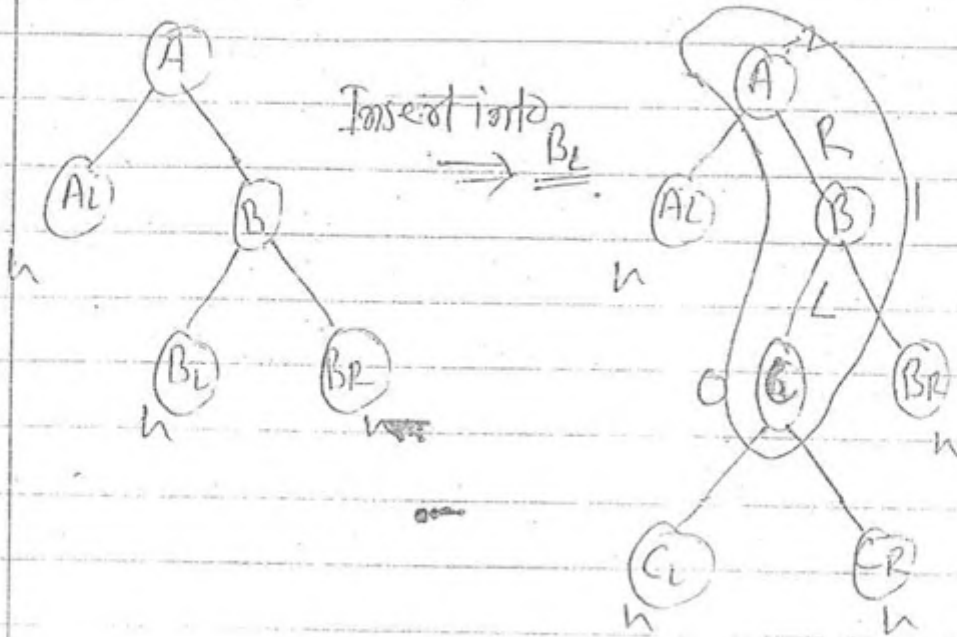
left.



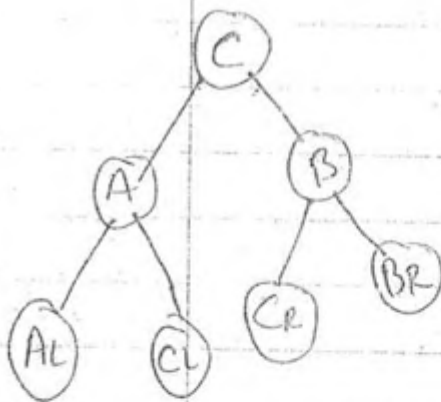
Rotation take  $O(1)$   
but insertion take  $O(\log n)$  time.

# Insertion

## RL Problem

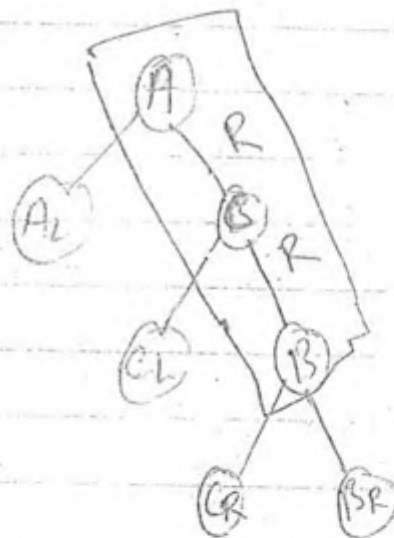


↓ Right.



AVL

Left

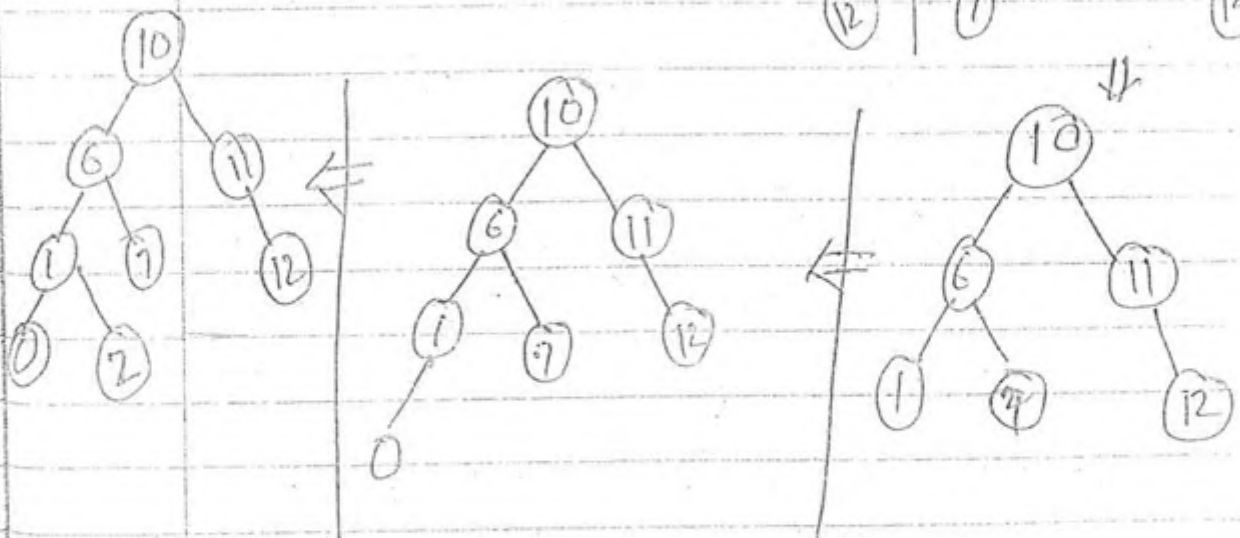
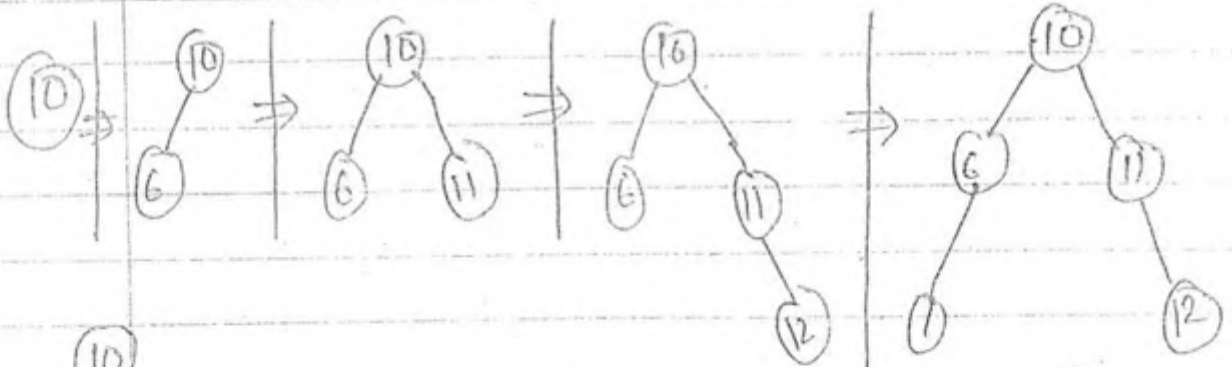
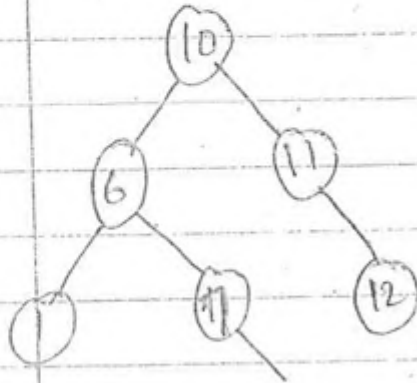


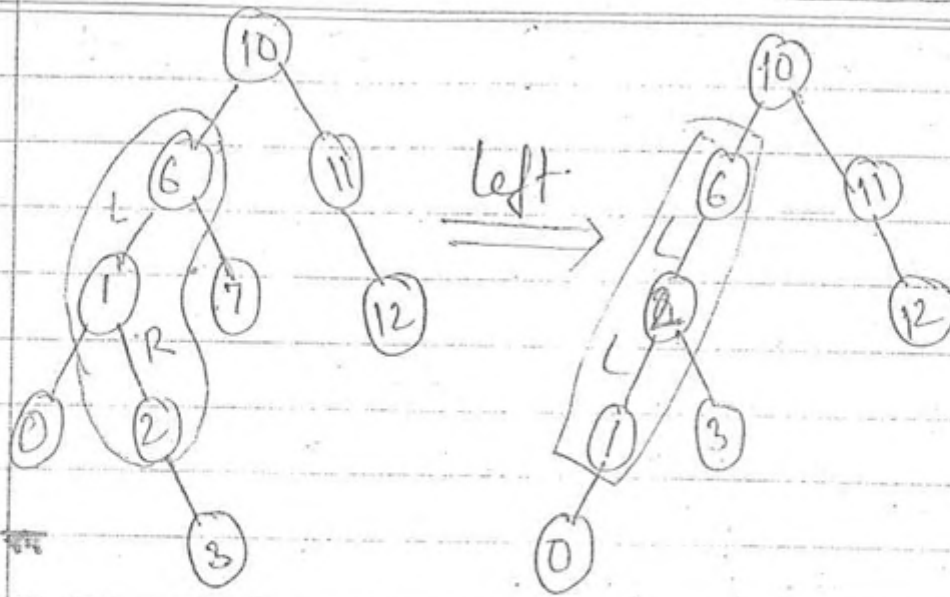
(10)



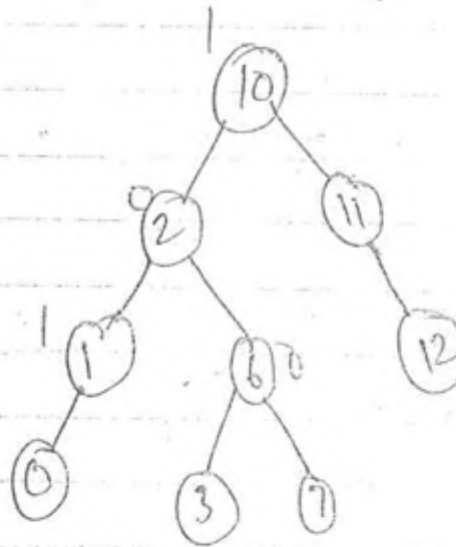


#P What will be the resulting AVL tree if we insert 10, 6, 11, 12, 1, 7, 0, 2, 3 in that order.



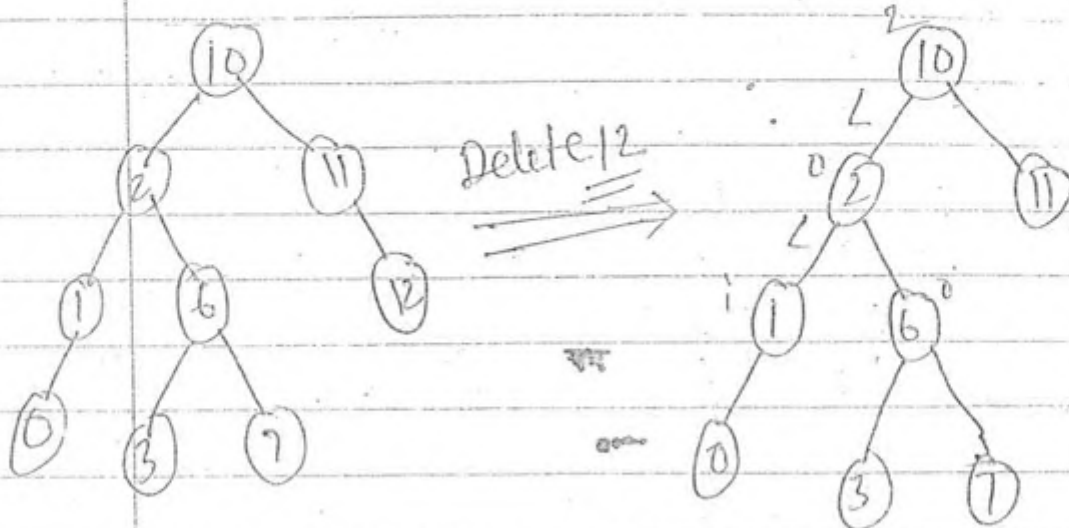


Right

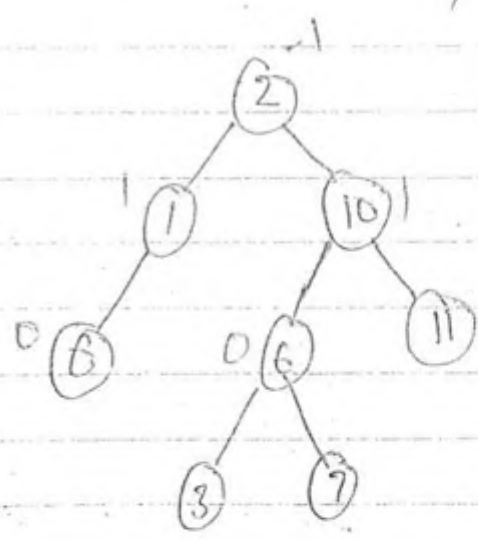


AVL Tree

#1 What will be resulting AVL tree if we delete 12 from above AVL tree.



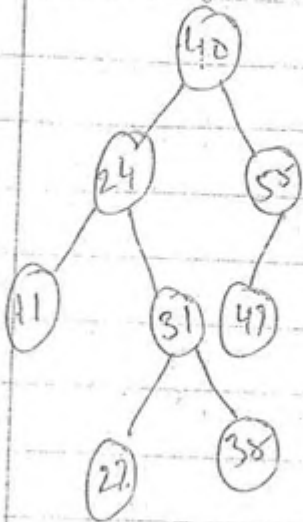
LL and LR got both problem but LL is better becoz LL has less rotation than LR. we do 2 rotations



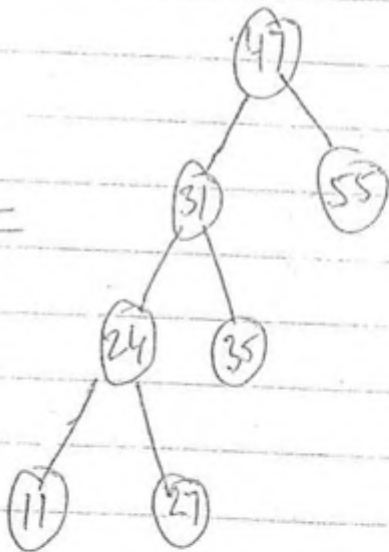
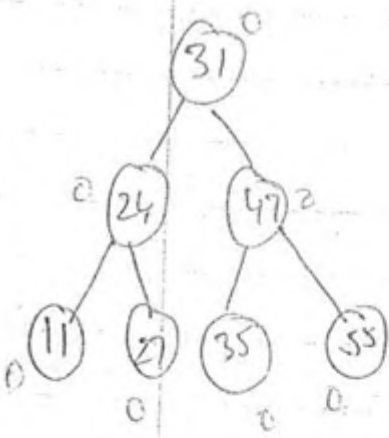
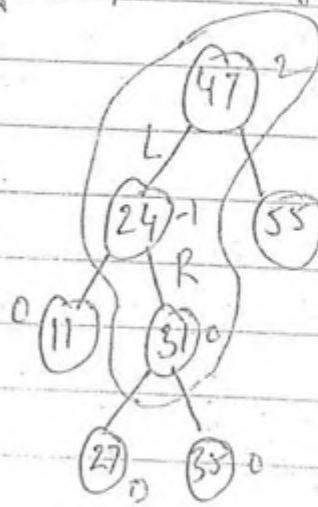
if are LL & LR problems we should go for LL.

#P

Consider the following AVL tree, what will be the resulting AVL tree if we delete 40 from above tree.



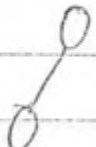
Delete 40

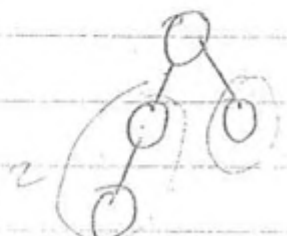


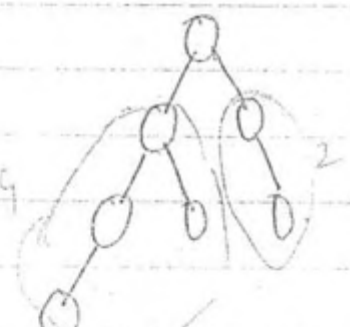
1 better.  
ve. tree.

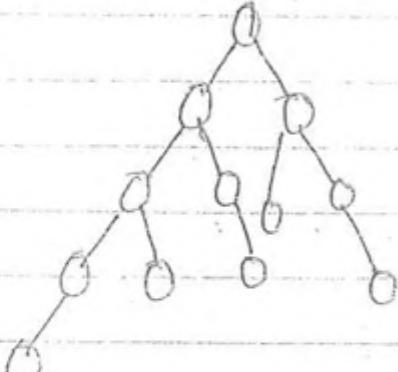
#8 What is the max<sup>m</sup> height of any AVL tree with 7 nodes. ?

$h=0$       0       $\Rightarrow 1$

$h=1$              $\Rightarrow 2$

$h=2$              $\Rightarrow 4$

$h=3$              $\Rightarrow 7$

$h=4$              $\Rightarrow 12$

Min<sup>m</sup> no of nodes in height(h) AVL tree  
is

$$\boxed{\text{Min}(h) = \text{Min}(h-1) + \text{Min}(h-2) + 1}$$

$$h=0 \Rightarrow 1$$

$$h=1 \Rightarrow 2$$

$$h=2 \Rightarrow 4$$

$$h=3 \Rightarrow 7$$

$$h=4 \Rightarrow 12$$

HT

$$h=5 = 12 + 7 + 1$$

$$\underline{\underline{= 20}}$$

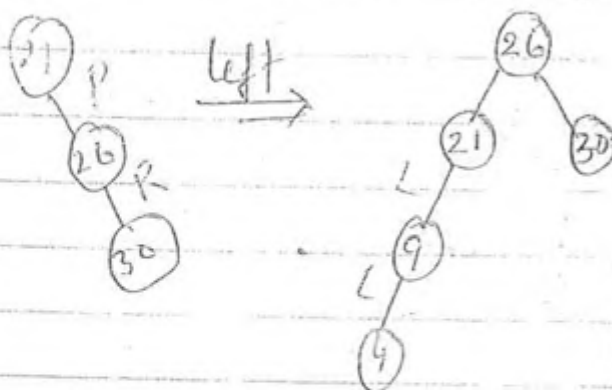
\* P

Create AVL tree for the following nodes:

21, 26, 30, 9, 4, 14, 28, 18, 15, 10, 2, 8, 7.

from above AVL tree delete - 2, 8, 10

18, 4, 9, 14, 7, 15.



Struct AVL

{  
int data;

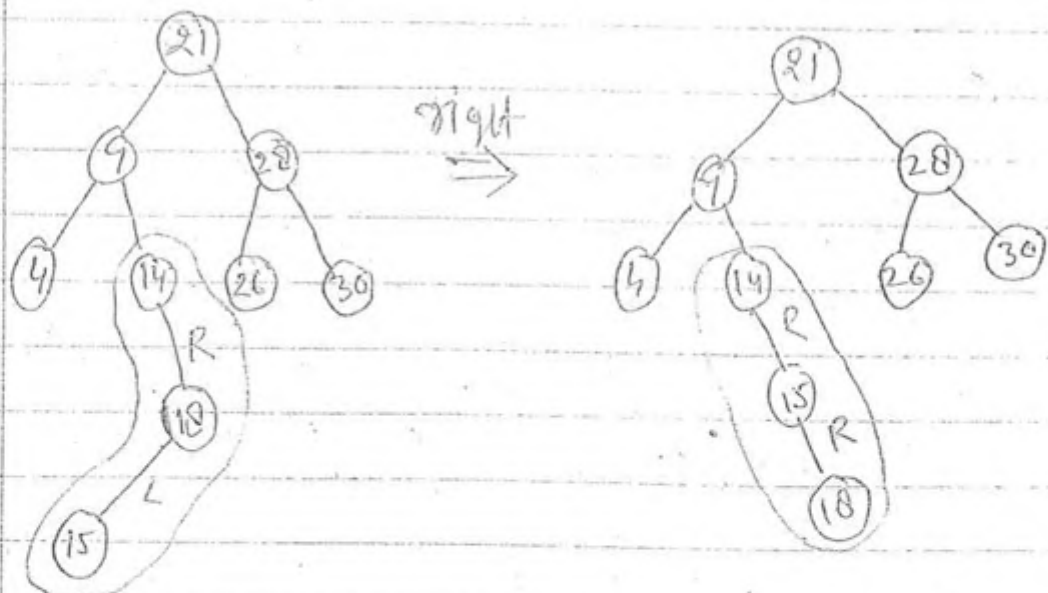
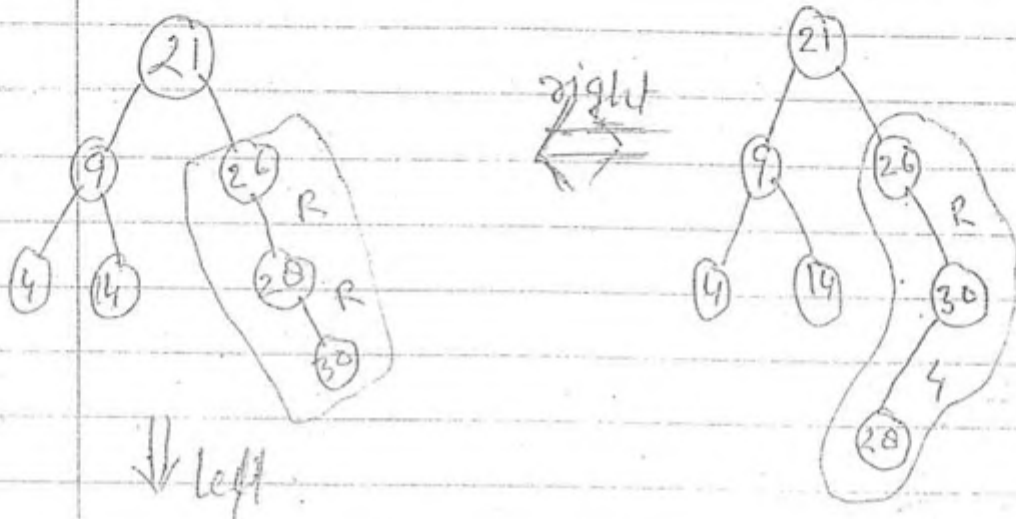
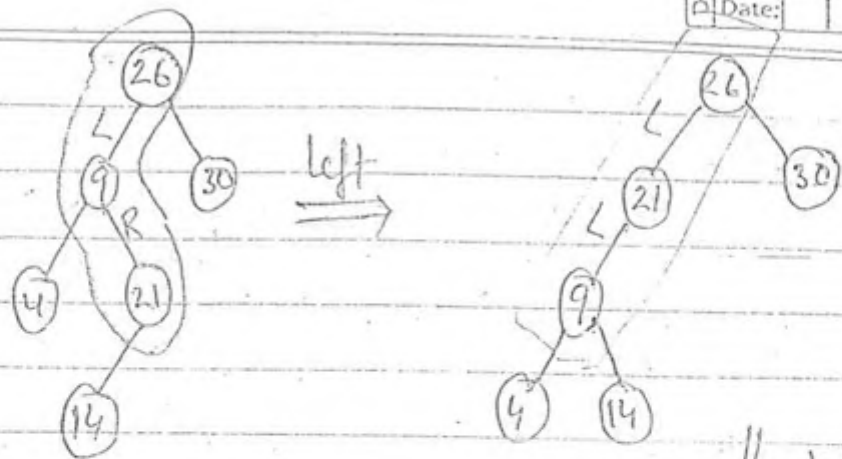
Struct AVL, \*lp, \*rp;

int height;

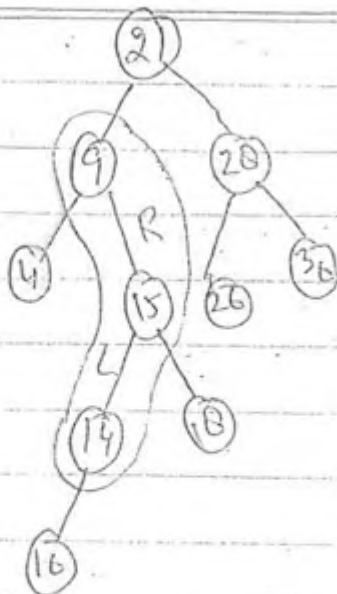
3

every node have height

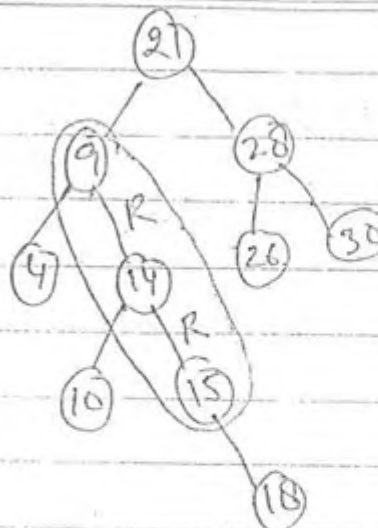




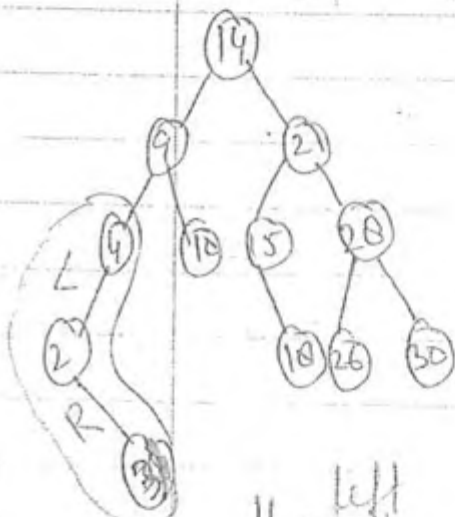
Left



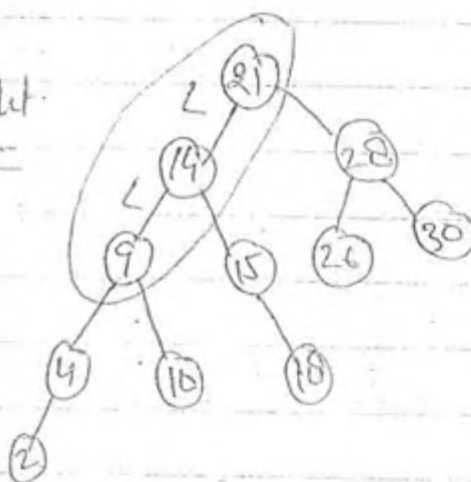
Right



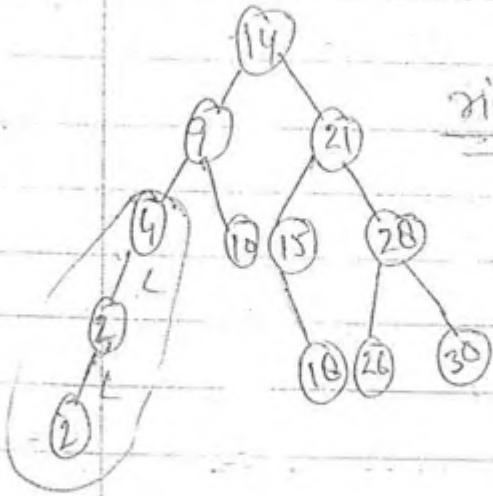
Left



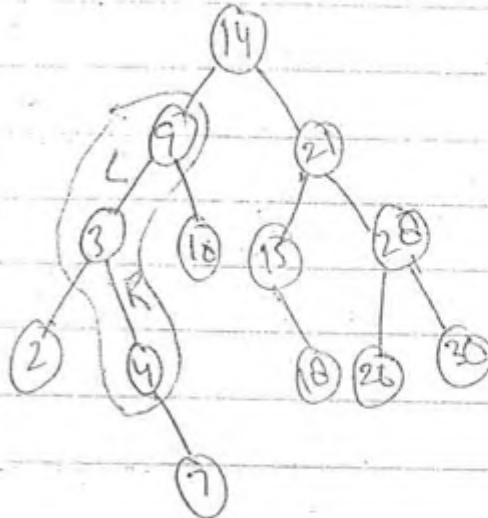
Right

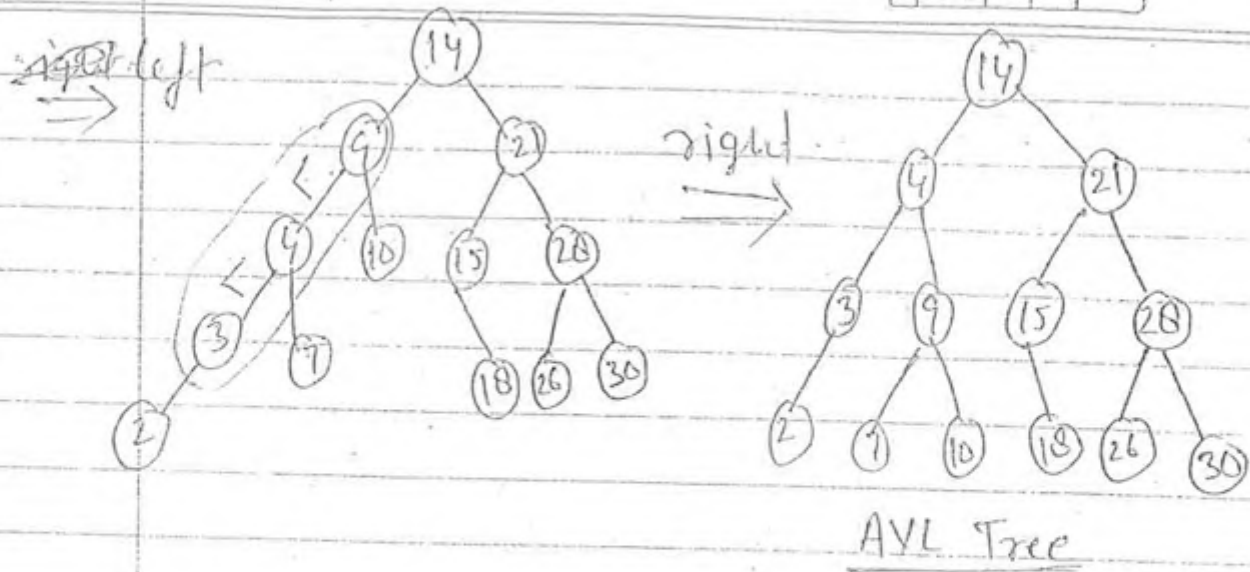


Left

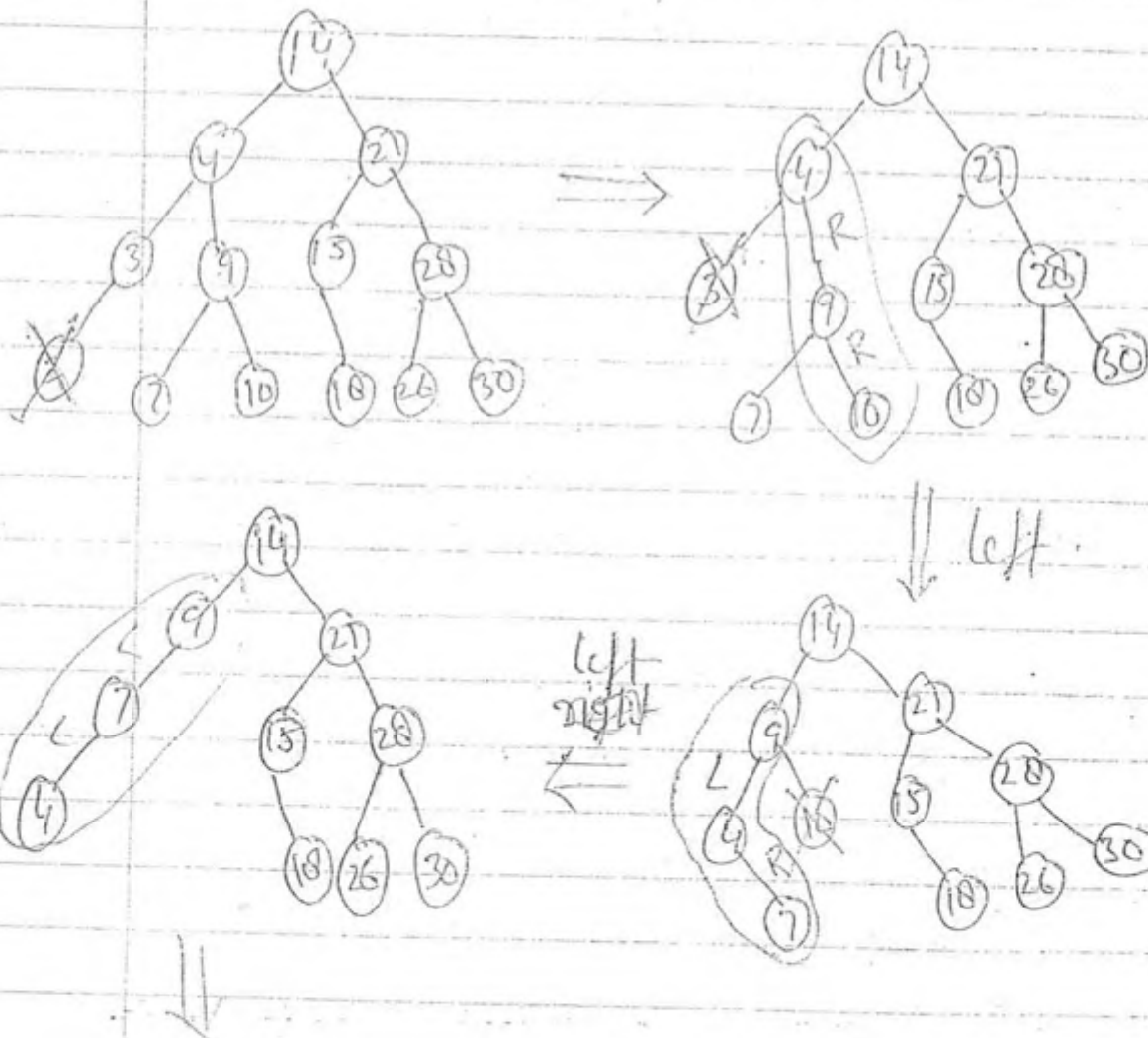


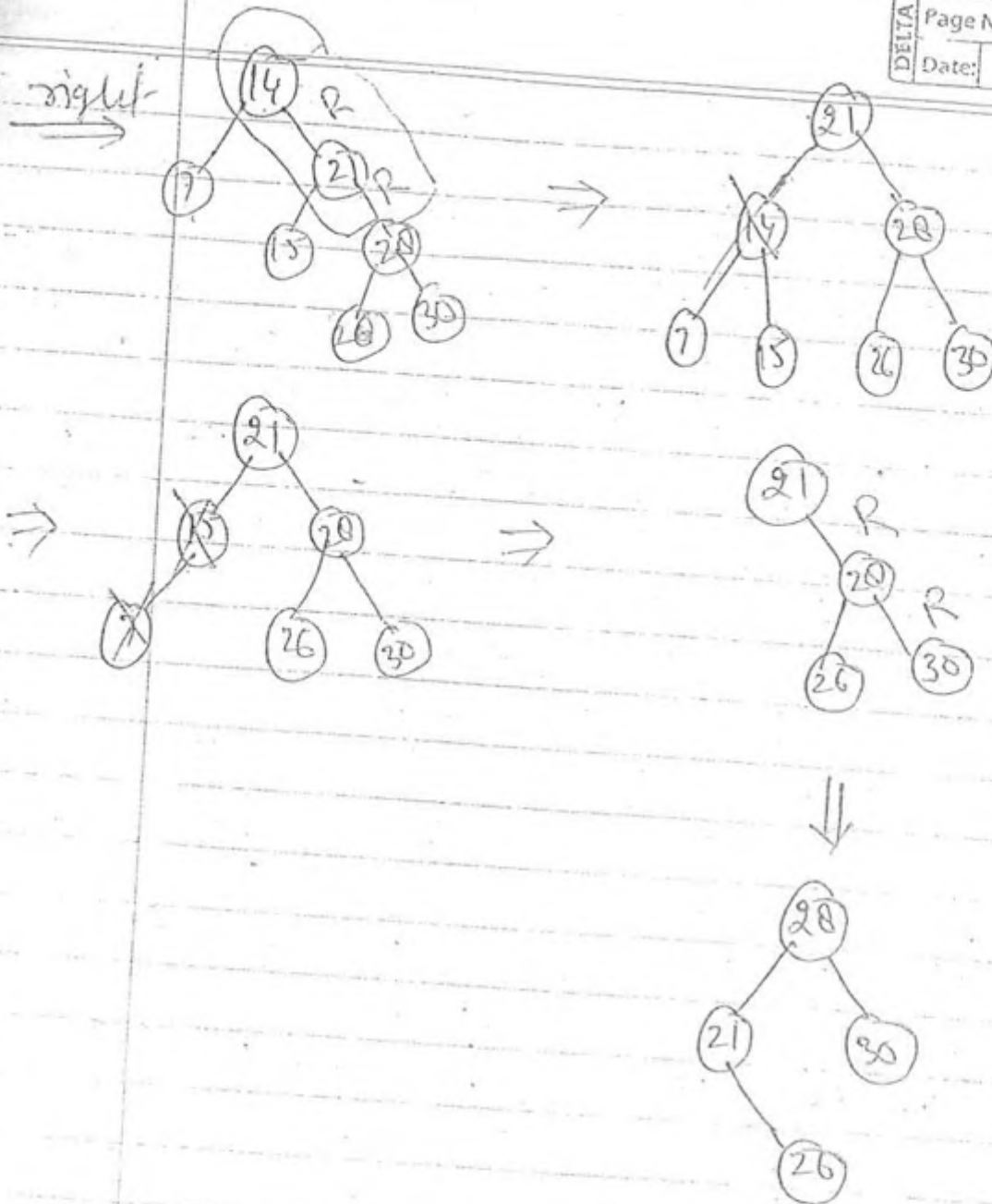
Right





Delete  $\Rightarrow$  2, 3, 10, 18, 4, 9, 14, 7, 15.





Note:- The height of AVL tree which contains  $n$  nodes  $= 1.44 \log_2 n$ .

$$1.44 \log_2 n < 2 \log n$$

AVL tree height no more than  $2 \log n$ .

② Inserting an element into an AVL tree which already contain  $n$  elements  
 $= O(\log_2 n)$

③ deleting an element from AVL tree the which already contain  $n$  - element.  
 $= O(\log_2 n)$

④ In a AVL tree insertion deletion ~~take~~ <sup>store</sup>  $O(n)$  the when AVL tree have no height.

```

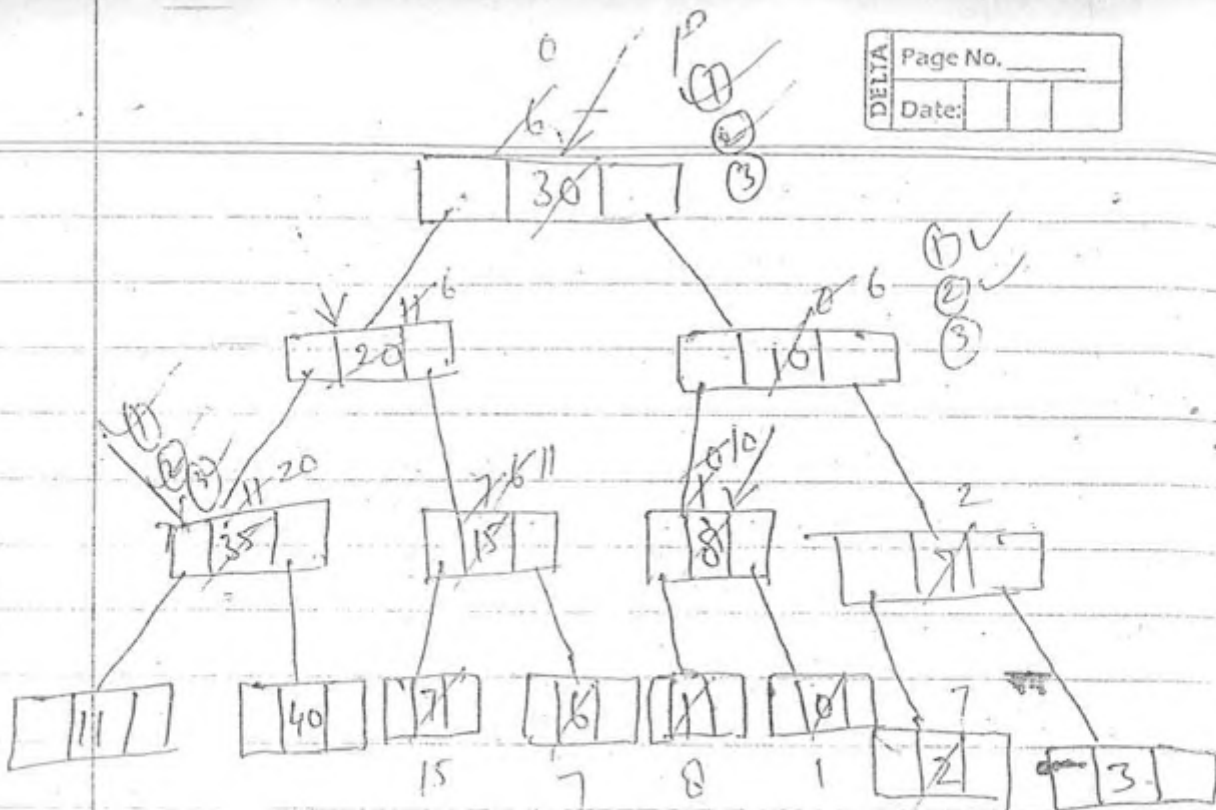
P
A(P)
{
 if (P != Null)
 {
 if (P->data > (P->lp->data))
 swap(P->data, P->lp->data)

 if (P->data > (P->rp->data))
 swap(P->data, P->rp->data)

 A(P->lp)
 A(P->rp)
 }
}

```





min heap

B(P)

```

{
 if (P->lp->lp != Null && P->rp->rp != Null)
 {

```

① B(P->lp)

② B(P->rp)

③ if ( P->data > P->lp->data )

swap ( P->data , P->lp->data )

if ( P->data > P->rp->data )

swap ( P->data , P->rp->data )

}



else

{

if ( $p \rightarrow data > p \rightarrow lp \rightarrow data$ )

swap ( $p \rightarrow data$ ,  $p \rightarrow lp \rightarrow data$ )

if ( $p \rightarrow data < p \rightarrow rp \rightarrow data$ )

swap ( $p \rightarrow data$ ,  $p \rightarrow rp \rightarrow data$ )

}

# B-Tree (Multi way searching)

- ① It is also one of the search tree.
- ② Minimum 2 - children.
- ③ All leaf node in the same level.

## Formula/degree/Order of the B-Tree

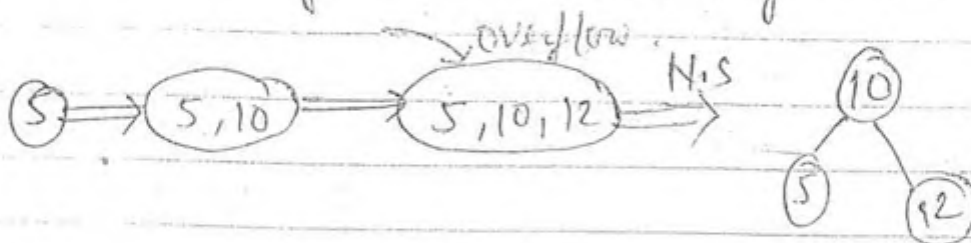
$$\begin{aligned}
 \text{Order of the B-tree} &= m (\text{maximum children}) \\
 &= m + 1 (\text{Keys}) (\text{max}^m \text{Keys}) \\
 &= \left\lceil \frac{m}{2} \right\rceil (\text{min}^m \text{children}) \\
 &= \frac{m}{2} - 1 (\text{min}^m \text{Keys})
 \end{aligned}$$

Ex

Consider the following elements 5, 10, 12, 13, 14, 1, 2, 3, 4, degree = 3

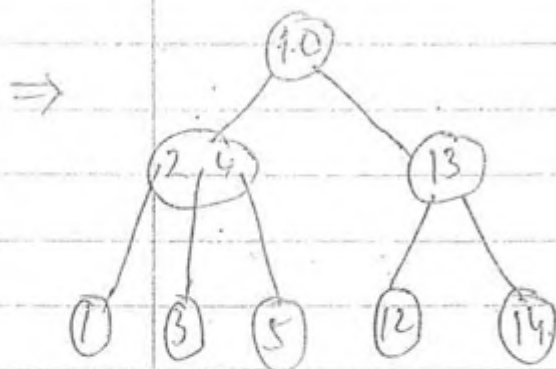
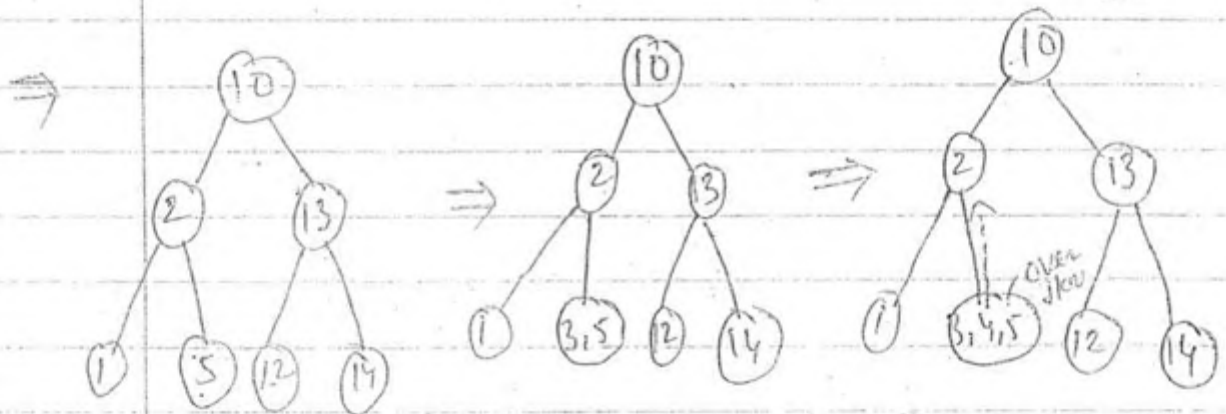
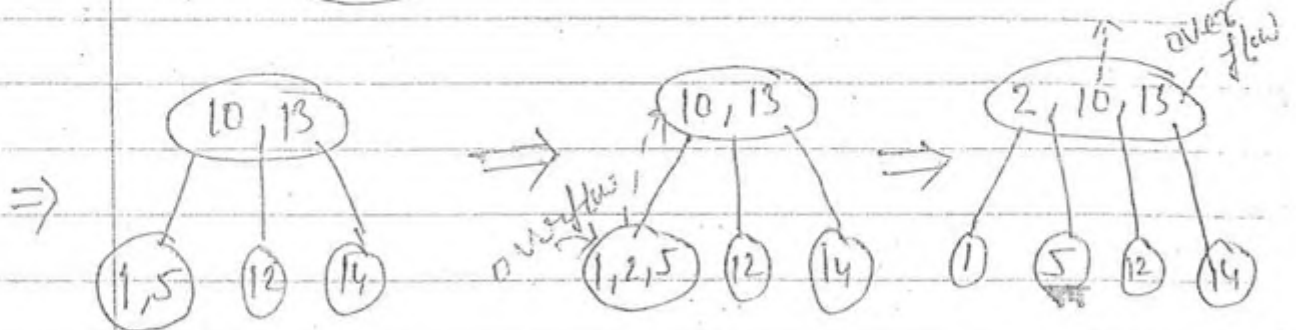
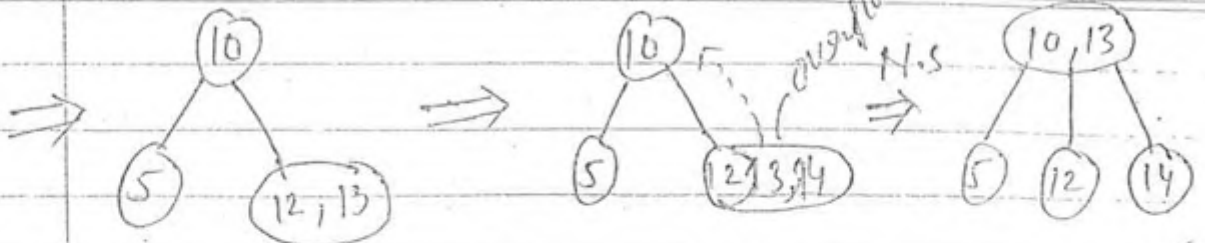
Sol<sup>n</sup>

$$\begin{aligned}
 \text{Max}^m \text{ children} &= 2 & \text{min}^m \text{ children} &= \frac{3}{2} = 1.5 = 2 \\
 \text{Max}^m \text{ Keys} &= 2 & \text{Min}^m \text{ Keys} &= 2 - 1 = 1
 \end{aligned}$$



# When root is overflowed merge it with its children

|             |                |
|-------------|----------------|
| DELTA       | Page No. _____ |
| Date: _____ |                |



$$O(\log_m n)$$

# # Degree 5

5, 10, 12, 13, 14, 1, 2, 4, 20, 18, 19, 17, 16, 15  
25, 23, 24, 22, 11, 30, 31, 28, 29.

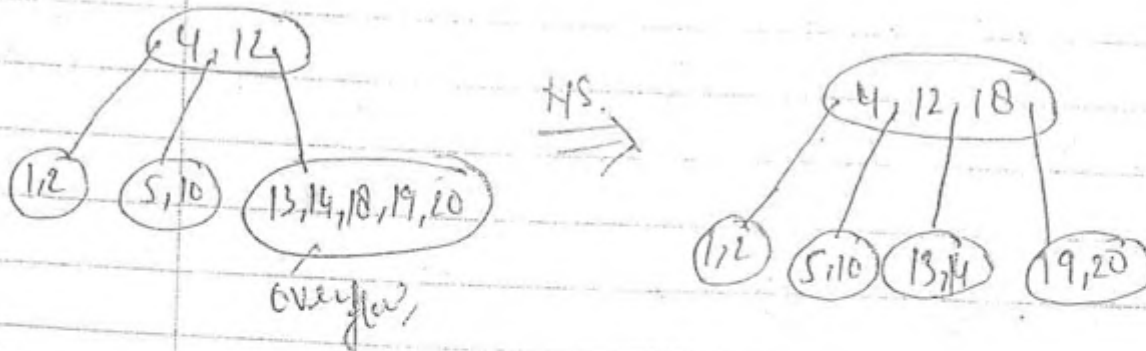
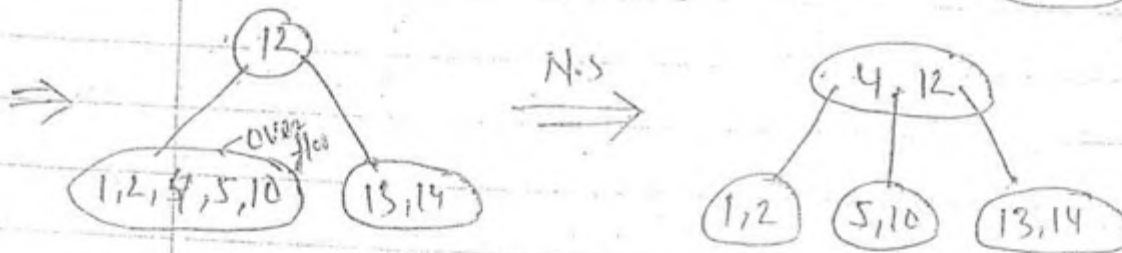
Max<sup>m</sup> children = 5

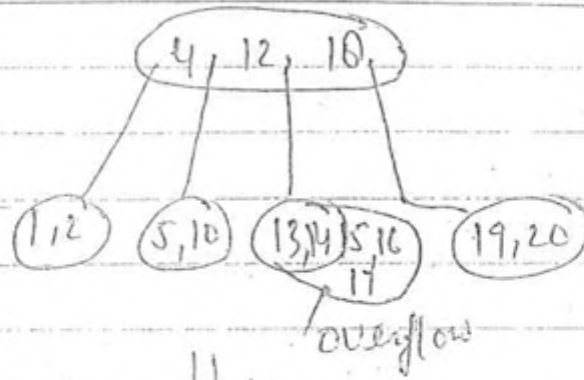
Keys = 4

Min<sup>m</sup> children = 3

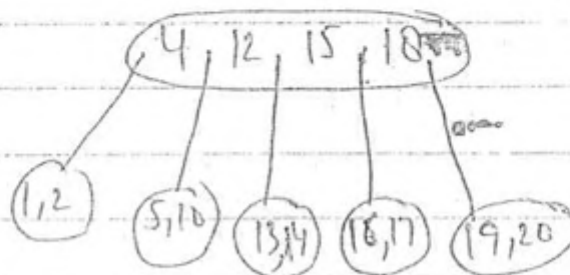
Keys = 2.

overflow

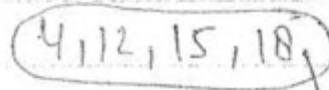




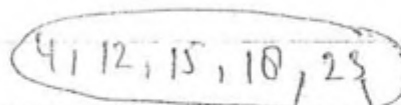
↓ N.S



↓



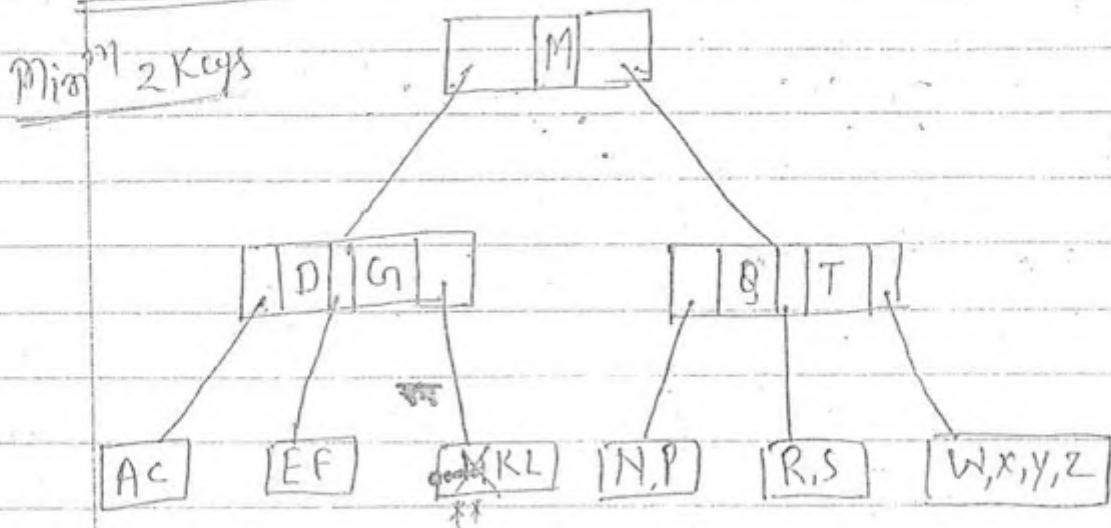
overflow



|      |                                                                         |  |  |
|------|-------------------------------------------------------------------------|--|--|
| DEVA | Page No. _____                                                          |  |  |
|      | Date: <table border="1"><tr><td> </td><td> </td><td> </td></tr></table> |  |  |
|      |                                                                         |  |  |

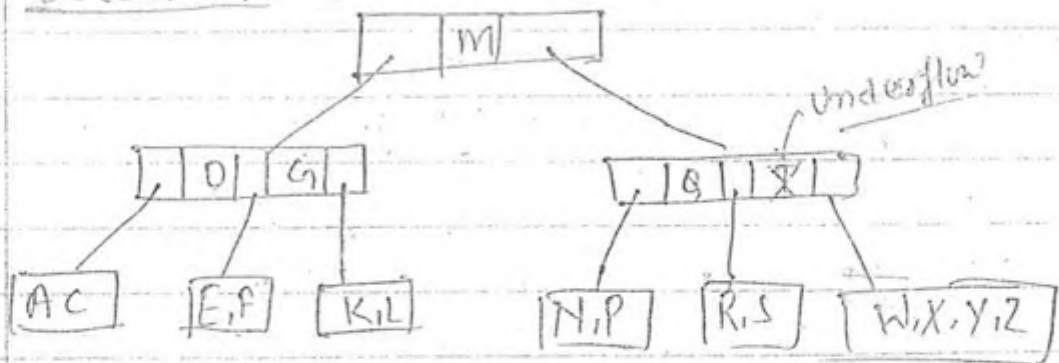


#1 Consider the following B+ tree.  
Deletion



$D = 6$   
 $\text{Max}^m \text{ch} = 6$   
 $K = 5$   
 $\text{Min}^m \text{ch} = 3$   
 $\text{Key} = 2$

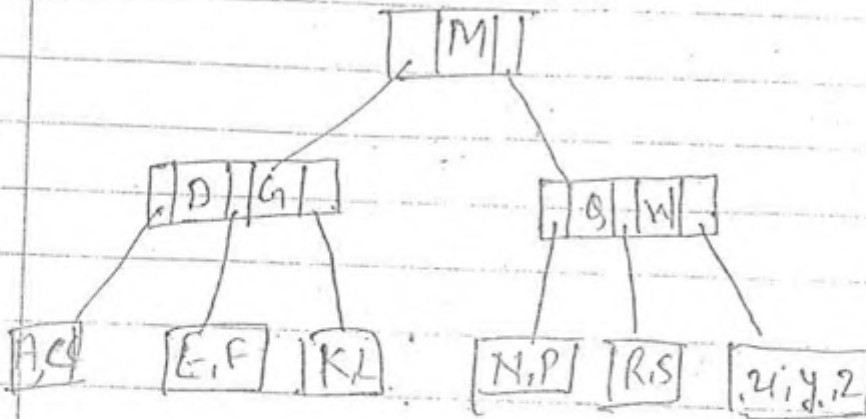
Delete-H



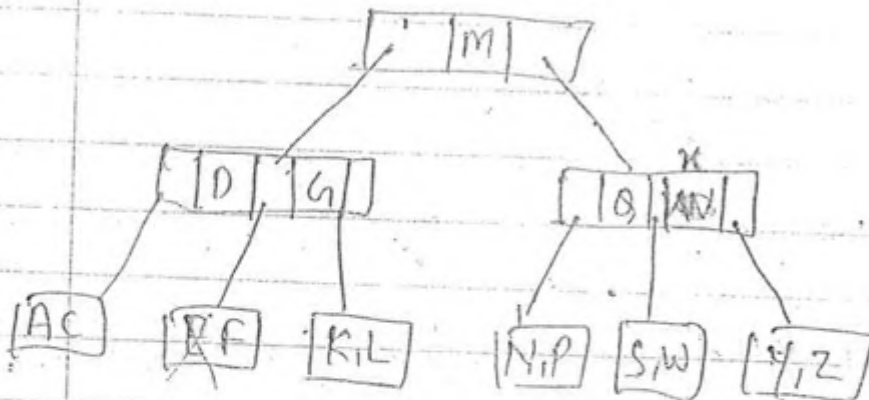
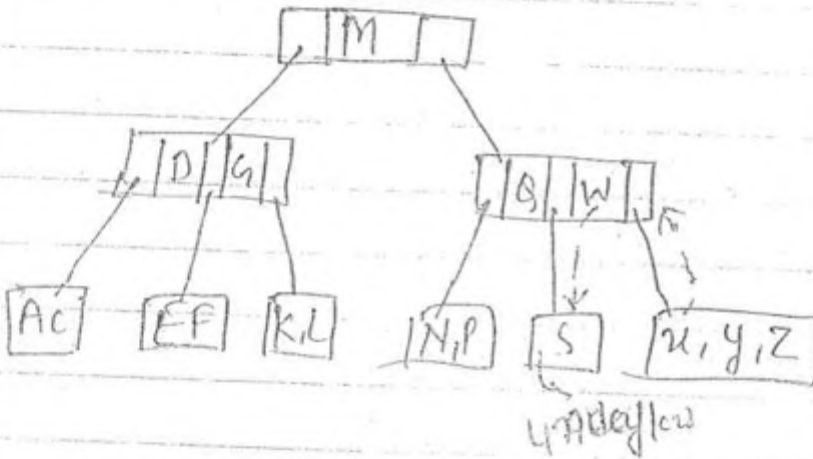
In the above tree after deleting T that node is in underflow but one of the children is having more than the required keys. So it will send

on Key to its parent.

Delete - T

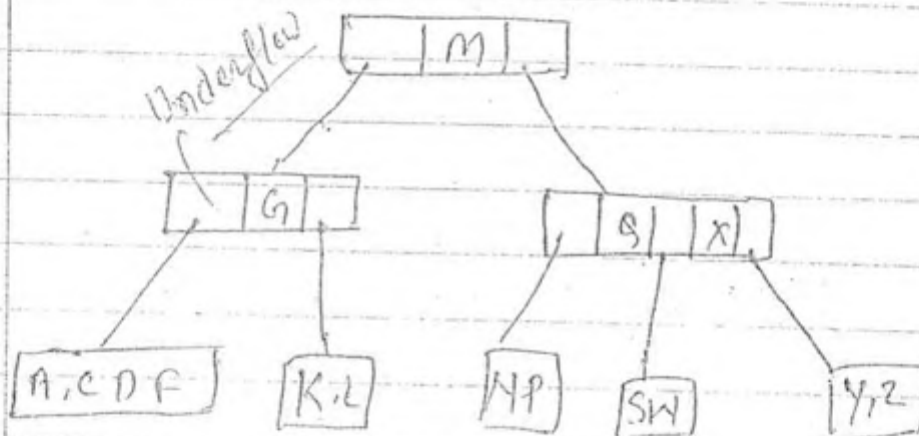
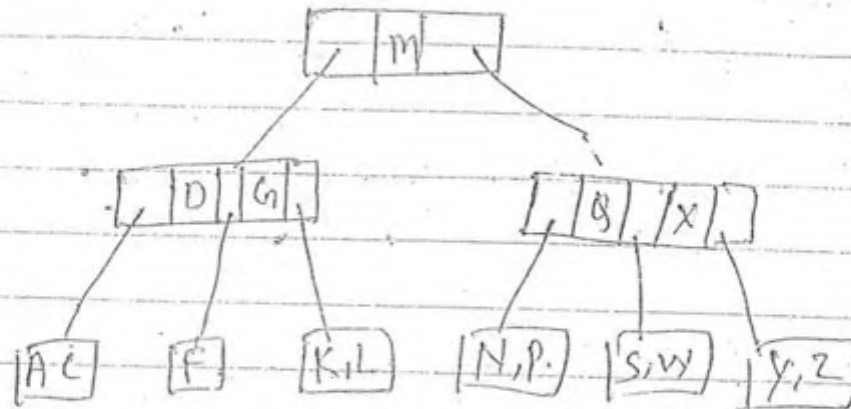


Delete - R



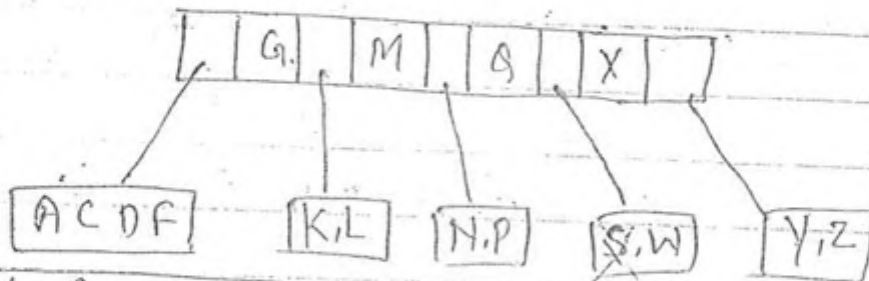
If y  
supla  
of H  
so  
comb  
inc

## Delete E



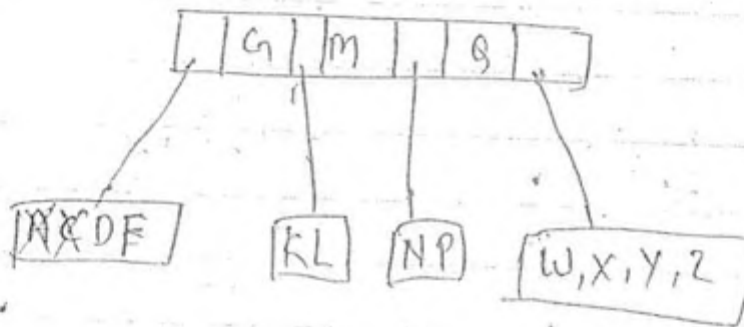
If you delete element E from the above B-tree replacement is one of the Keys from any one of the Sibling. but it's simply it not ready. So final sol<sup>n</sup> is made combining. Node combining nothing but grouping two Sibling including parent.

Now the node G is underflow. No Sibling is ready to help.



Delete A and C. No Problem

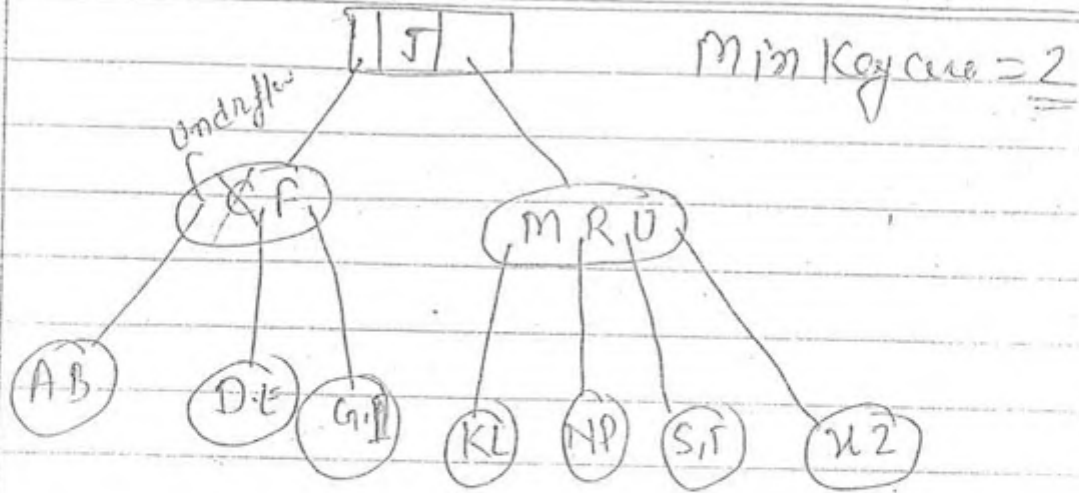
Delete S



# Min<sup>m</sup> condition should satisfied other than the root every

|       |          |  |  |  |
|-------|----------|--|--|--|
| DELTA | Page No. |  |  |  |
|       | Date:    |  |  |  |

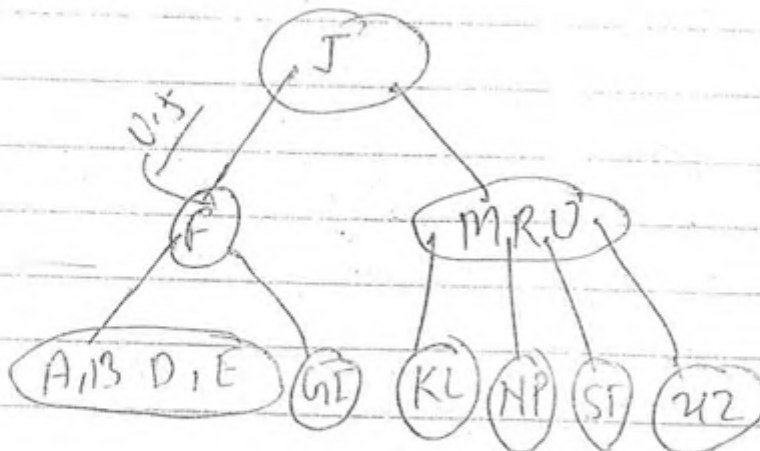
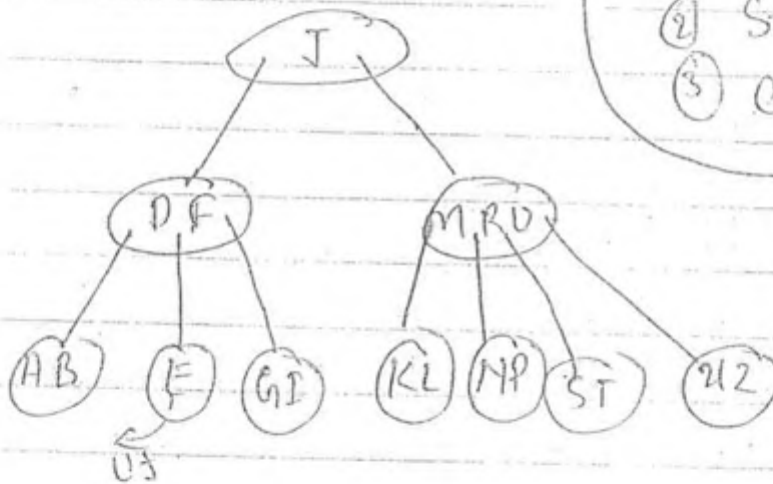
#9



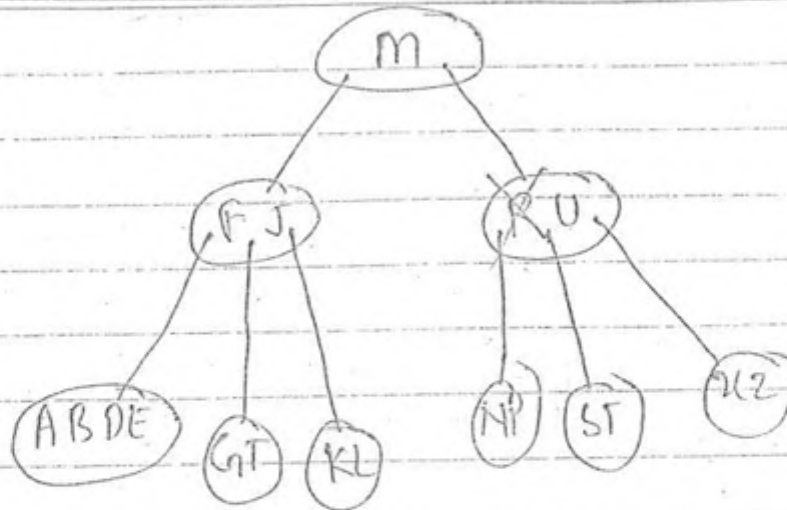
Sol<sup>n</sup>

Delete C

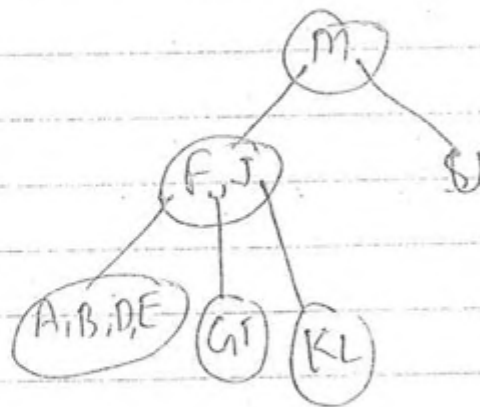
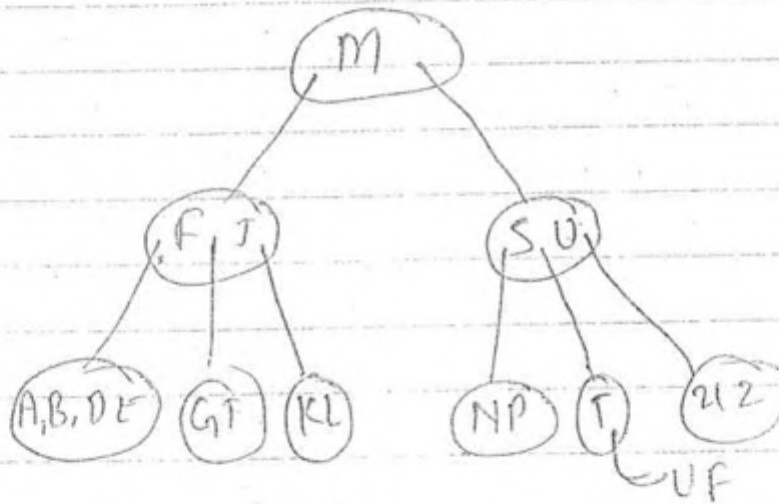
- ① Children  
 ② Sibling  
 ③ Combined



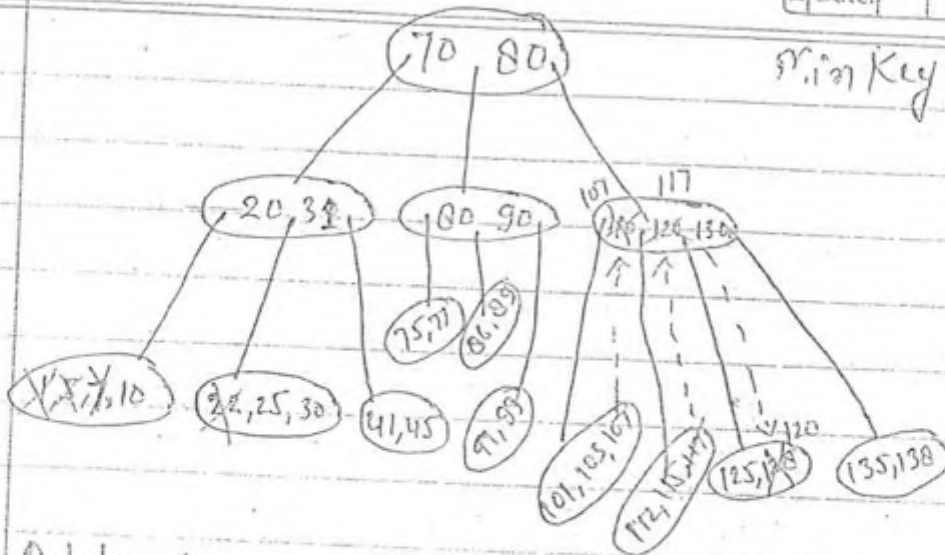




Delete R



Min Key = 2



Delete - 1 — delete directly, no effect

Delete - 22 — " " "

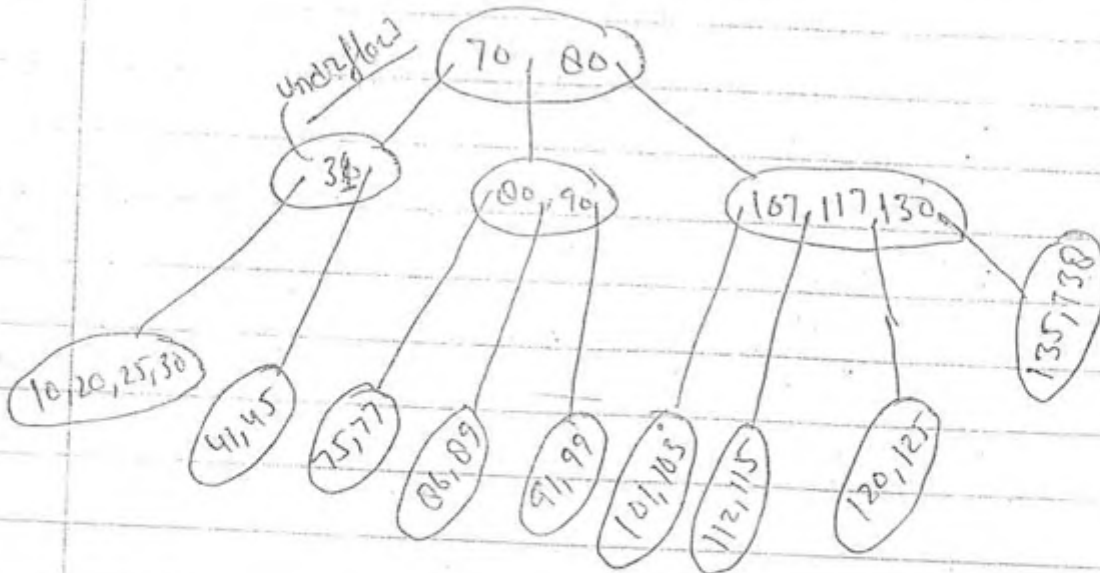
Delete - 5 — " " "

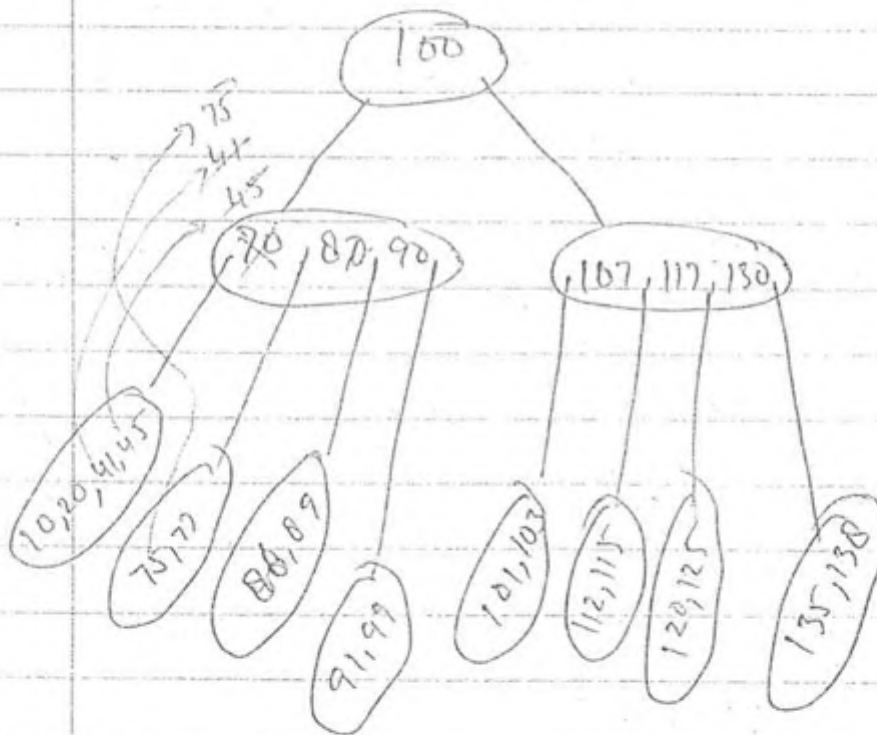
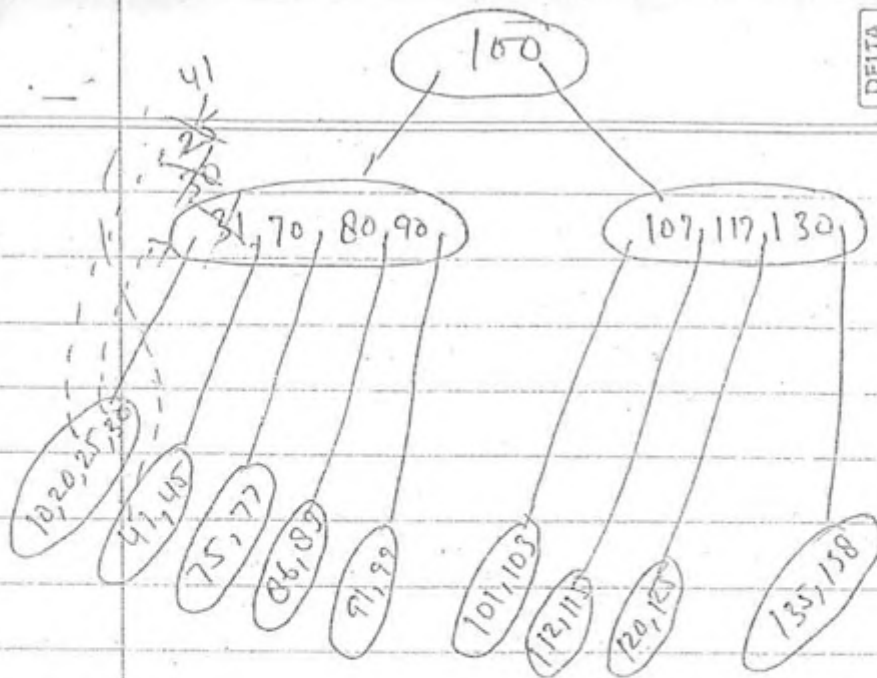
Delete 128 —

Delete 110

Delete 7

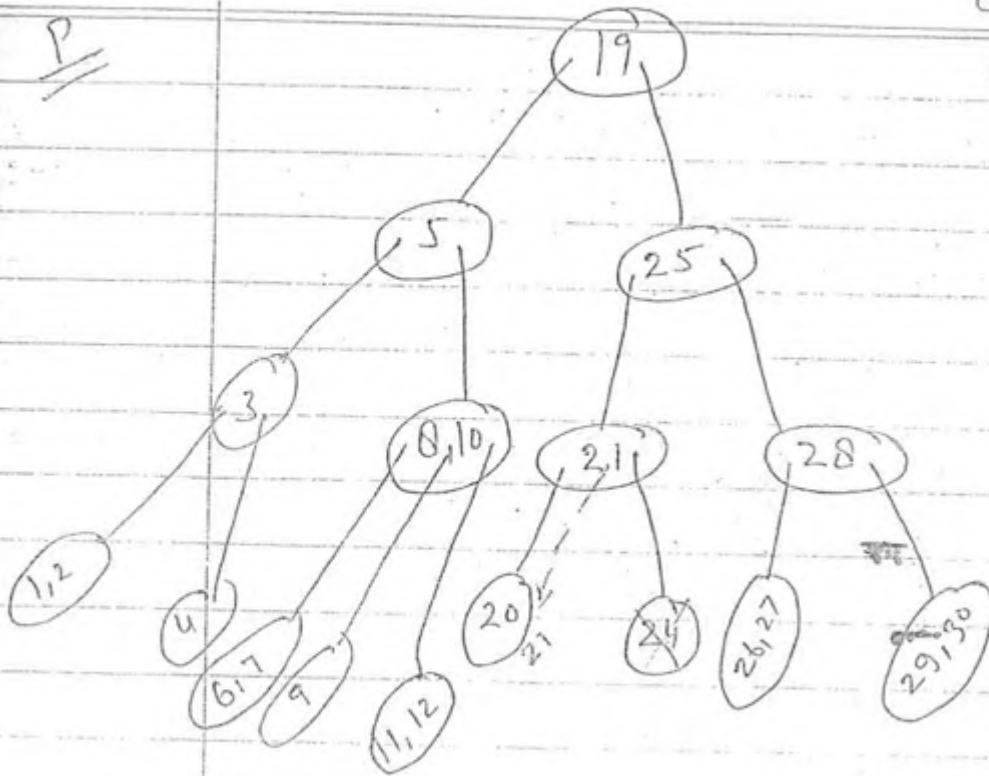
after Deleting (7)



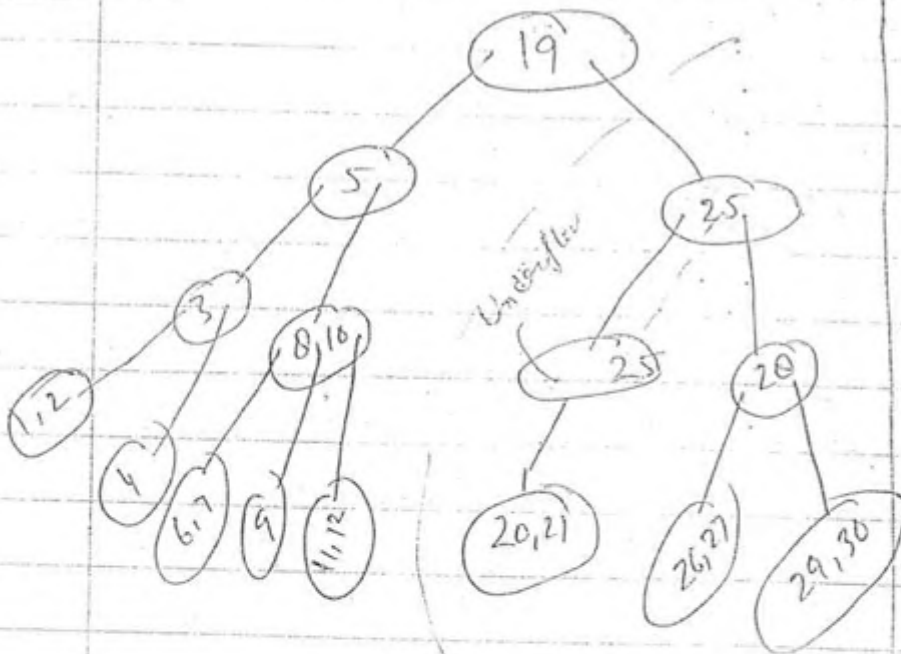


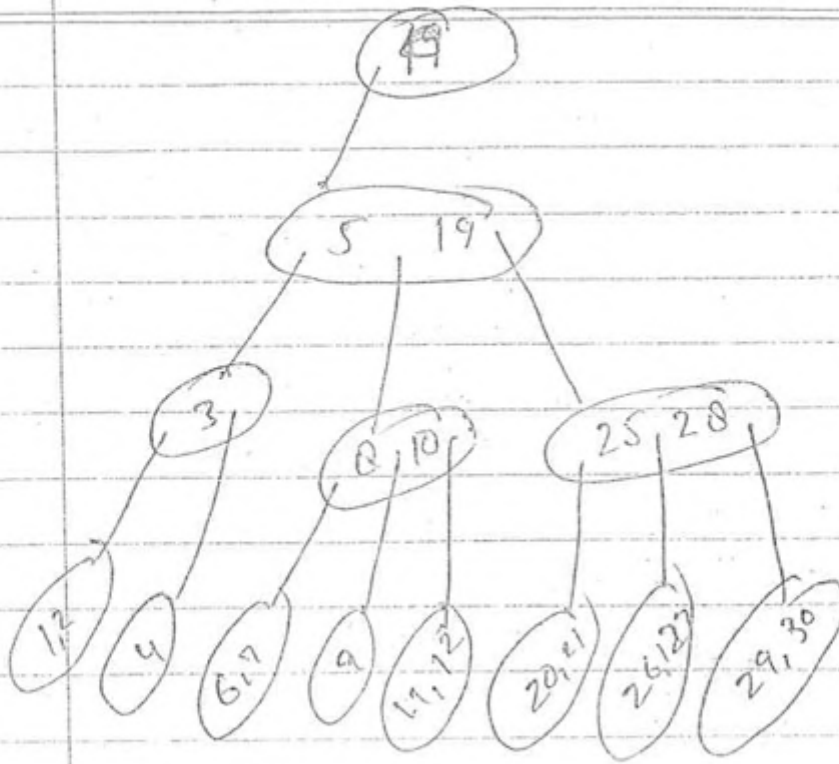
P

Min Key = 1



Delete 24





P B-tree of order 32,  $m=32$ . then.

- ① Find the max<sup>m</sup> key that can accommodate in level 3 B-tree
- ② Find out min<sup>m</sup> no. Key that can accommodate in level 3 B-tree.

Sol<sup>n</sup>

Max<sup>m</sup> ch — 32

Key<sup>m</sup> — 31

Min ch = 16

key = 15

Max<sup>m</sup>

| Level | nodes    | Key                | children |
|-------|----------|--------------------|----------|
| 1     | 1 (root) | 31                 | 32       |
| 2     | 32       | $32 \times 31$     | $(32)^2$ |
| 3     | $(32)^2$ | $(32)^2 \times 31$ | $(32)^3$ |



Min<sup>m</sup>

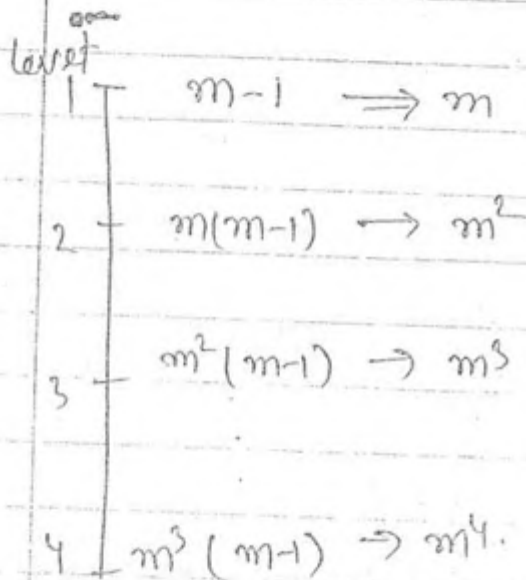
| level | nodes    | Key                | Children. |
|-------|----------|--------------------|-----------|
| 1     | 1        | 15                 | 16        |
| 2     | 16       | $16 \times 15$     | $(16)^2$  |
| 3     | $(16)^2$ | $(16)^2 \times 15$ | $(16)^3$  |

Min<sup>m</sup>

| level | nodes         | Key                     | Children.         |
|-------|---------------|-------------------------|-------------------|
| 1     | 1             | 1                       | 2                 |
| 2     | 2             | $2 \times 15$           | $2 \times 16$     |
| 3     | $2 \times 16$ | $2 \times 16 \times 15$ | $2 \times (16)^2$ |
| 35    |               | 511                     |                   |

P Let  $T$  be a B-tree of order  $m$  of height  $h$ .  
~~if  $n$  is the total key =  $n$ .~~  
 $n = ?$

P Let  $T$  be a B-tree of order  $m$  and height  $h$ .  
 if  $n$  is the number of key elements in  
~~at~~ then the max<sup>m</sup> value of  $n$  is;



$$m-1 + m(m-1) + m^2(m-1) + m^3(m-1) + \dots + m^h(m-1)$$

$$m-1 [ 1 + m + m^2 + m^3 + \dots + m^h ]$$

$$m-1 [ m^0 + m^1 + m^2 + \dots + m^h ]$$

$$m-1 \left[ 1 \left( \frac{m^{h+1} - 1}{m - 1} \right) \right] \Rightarrow \boxed{m^{h+1} - 1}$$

DRPG

Q

Let  $T$  be tree of order  $m$  and height  $h$ . Let  $d = \left\lceil \frac{m}{2} \right\rceil$  and let  $n$  be the no of elements in  $T$ . Then which of the following is True?

| level | nodes        | key               | children     |
|-------|--------------|-------------------|--------------|
| 1     | 1            | 1                 | 2            |
| 2     | 2            | $2 \times (d-1)$  | $2 \times d$ |
| 3     | $2 \times d$ | $2 \times d(d-1)$ | $2d^2$       |

$$= 1 + 2(d-1) + 2d(d-1) + 2d^2(d-1) + \dots + 2d^{h-1}(d-1)$$

$$= 1 + 2(d-1) \left[ d^0 + d^1 + d^2 + \dots + d^{h-1} \right]$$

$$= 1 + 2(d-1) \left[ \frac{d^h - 1}{d - 1} \right]$$

$$= 2d^{h-1} - 2 + 1$$

$\Downarrow$

$$\boxed{2d^{h-1}}$$

# B<sup>+</sup>-Tree

Q

Degree 4

↓

max<sup>m</sup> children = 4

Key = 5

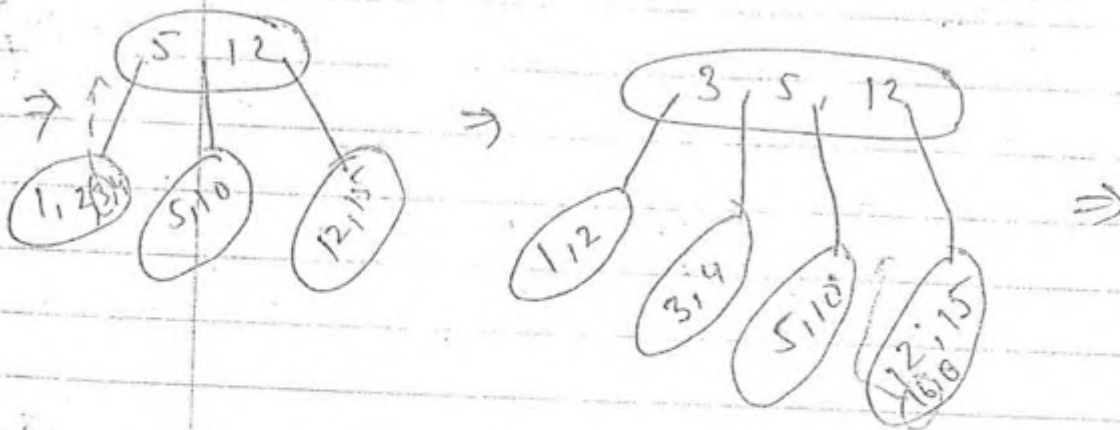
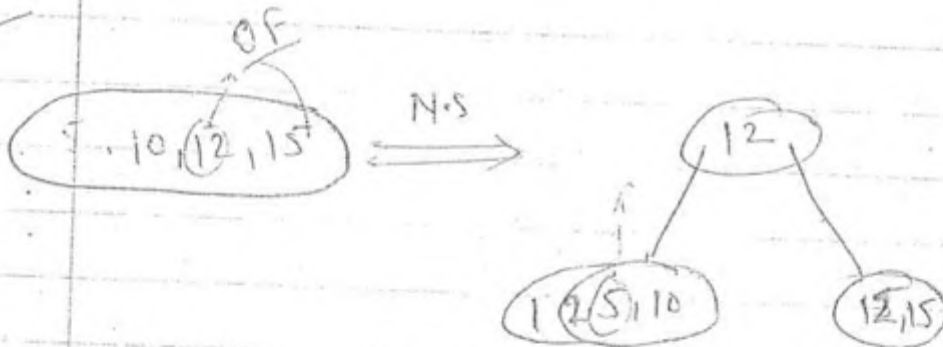
min ch = 2

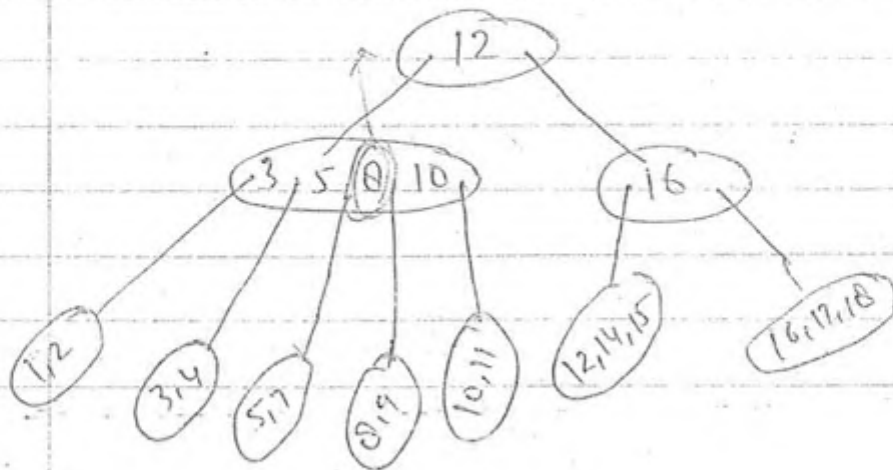
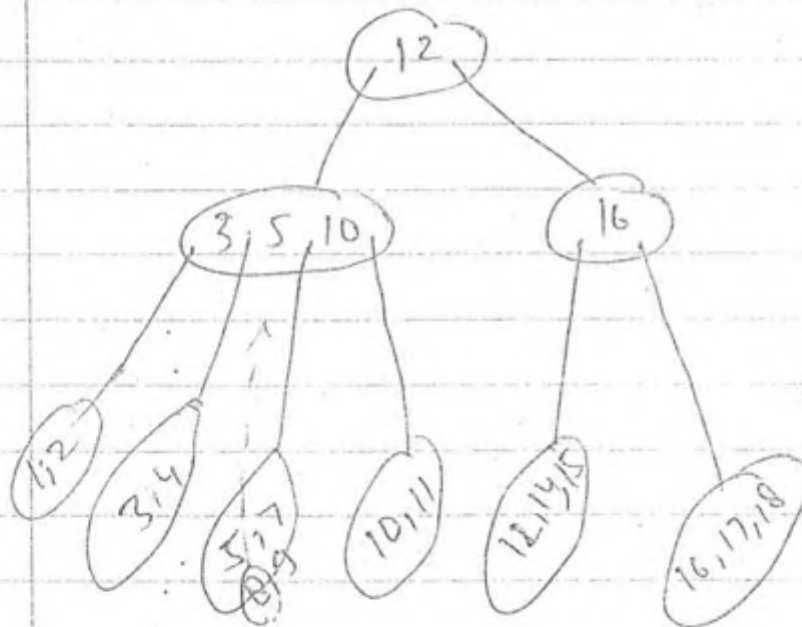
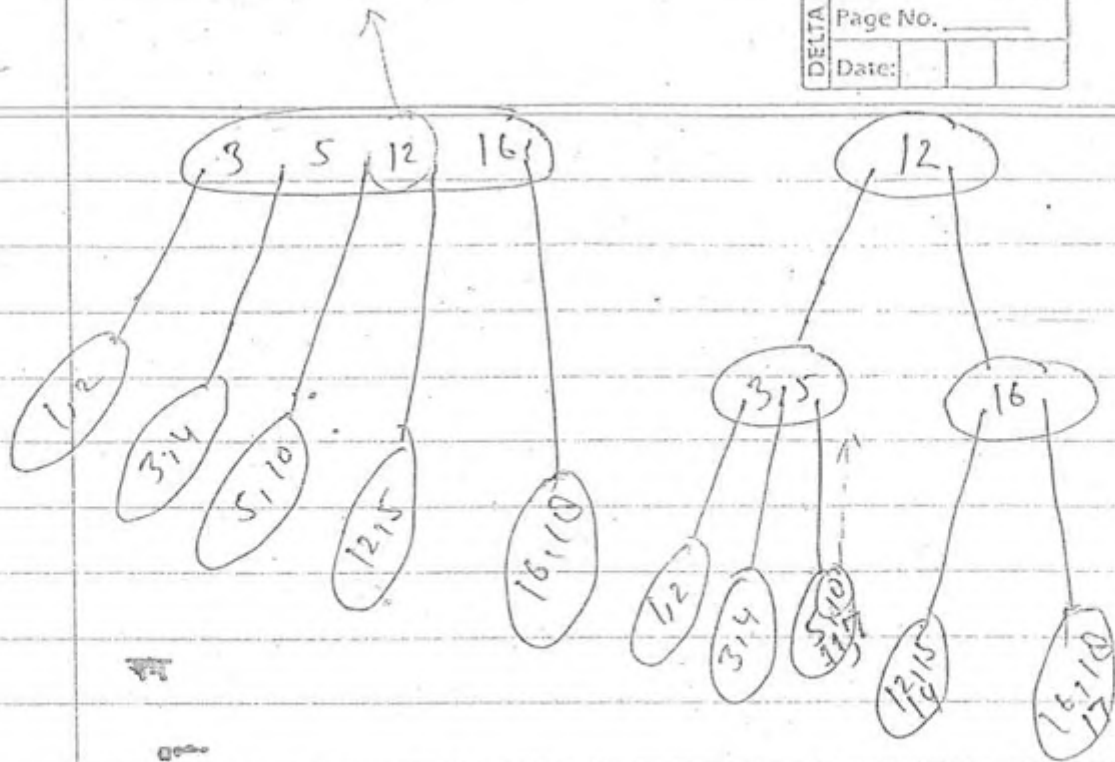
Key = 1

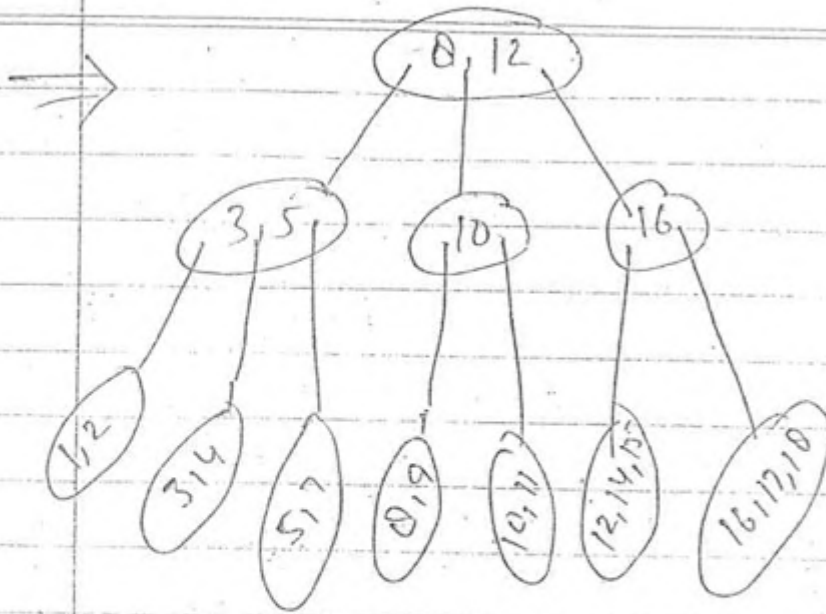
Consider B-Tree for the following

5, 10, 12, 15, 1, 2, 3, 4, 16, 18, 17, 14, 11, 7, 8, 9

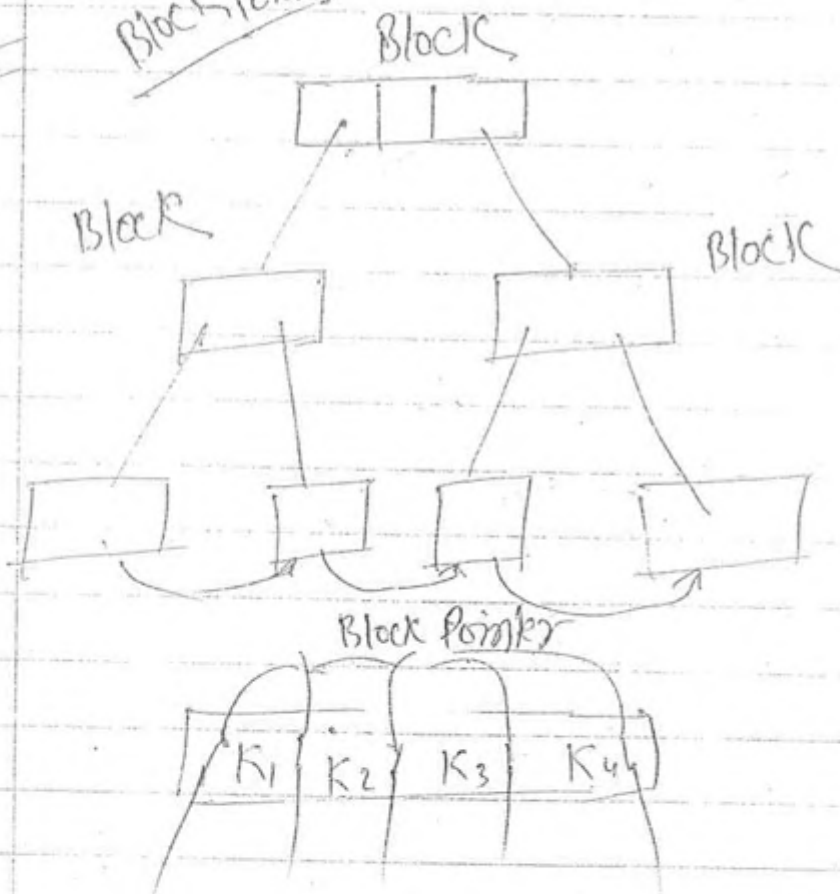
Sol<sup>n</sup>



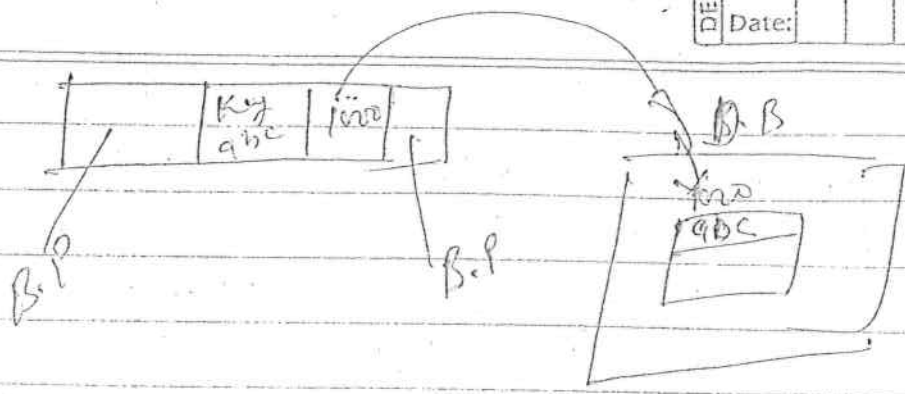




Block pointer





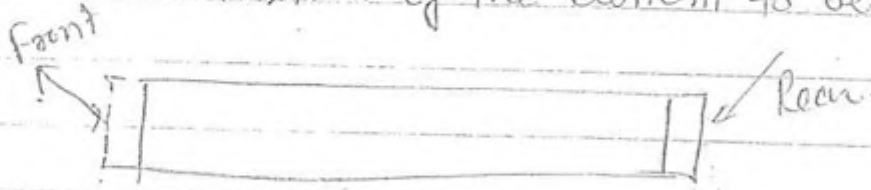


## - Queue

Definition :- One side insertion and another side deletion.

: FIFO

: Front is a variable which contains position of the element to be deleted



: Rear is a variable which contains position of the newly inserted element.

Abstract data type of Queue.

Enque ( Queue , element ) = Queue .

Deque ( Queue ) = element .

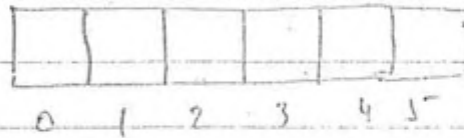
Isfull ( Queue ) = Boolean .

Is empty ( Queue ) = Boolean .

# Enqueue

$N = 6$

Q.



$F = -1$

$R = -1$

(all time both will be -1)

Void Enqueue(Q, N, F, R, u)

```

{
 if (R == N-1)
 {
 Print Queue is full;
 exit(0);
 }
 else
 {
 if (R == -1)
 {
 F = R = 0;
 }
 else
 {
 R++;
 }
 Q[R] = u;
 }
}

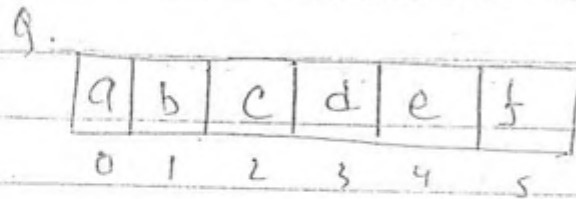
```

Is full()

Not Insertion

remaining

Deque (It is a operation)  
 $N=6$



$F=0$   
 $R=5$

Deque (q, N, F, R)

```

{
 int y;
 if (F == -1)
 {
 Print(" deque is empty")
 exit(1);
 }
 else
 {
 y = q[F]
 if (F == R)
 {
 F = R - 1;
 }
 else
 {
 F++;
 }
 return (y);
 }
}

```

Is empty

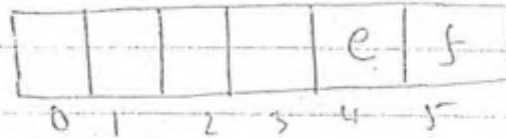
$O(1)$

Is deletion

remaining deletion

In linear time always.  
Compare  $R \geq F$

$N=6$



$F=4$

$R=5$

In the above linear queue, even though four slots are empty, we can not insert because  $R$  can not go back. So Linear queue is not efficient implementation of queue.

So we are going for circular queue in which  $R$  can go back also.

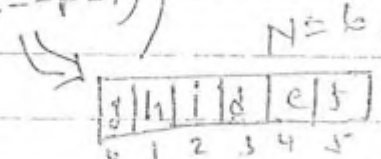
## Circular Queue

In circular queue:  $(R < F)$   
Circular Queue Insertion

CircularQueue( $S, N, F, R, K$ )

if  $(R == N-1 \ \&\& \ F == 0) \ || \ (R == F-1)$

{  
 printf("Queue is full");  
 exit(1);  
 }



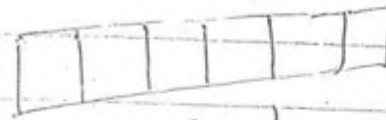
$F=3$   
 $R=2$

else.

{

if (R == -1)

F = R = 0;

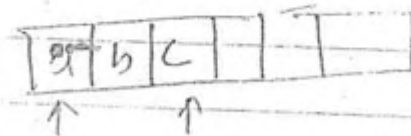
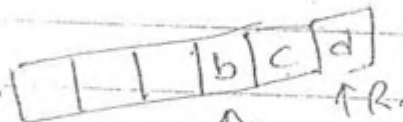


else

if (R == N-1)

R = 0

else R++



}

}

## Circular Queue Deletion

Cdeletion(Q, N, F, R)

{ int y

if (F == -1)

{

printf("Queue is Empty")

.exit(1);

}



else  
{

$y = g[F]$

if ( $F == R$ )

$F = R = -1$

else

if ( $F == N-1$ )

$F = 0$

else

$F++$

return (y);

last Delete.

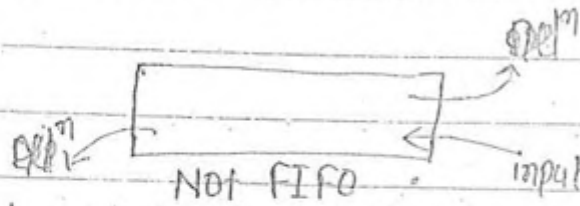
Circular deletion.

remaining()

## Types of Queue

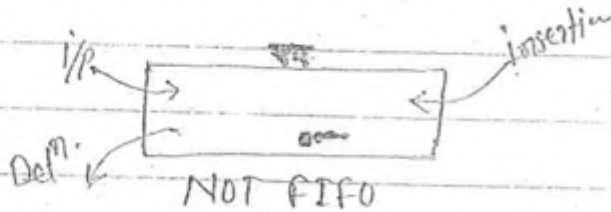
- (1) Input restricted Queue.
- (2) Output restricted Queue.
- (3) Double ended Queue.
- (4) Ascending priority Queue.
- (5) Descending priority Queue.

### ✓ Input Restricted Queue :-

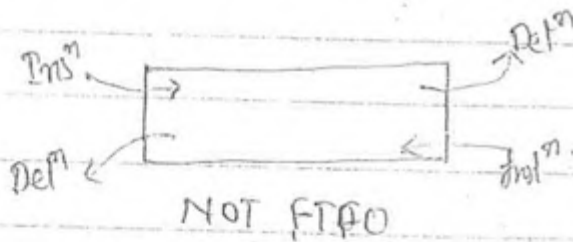


Input will be done only one side but o/p will be done both side.

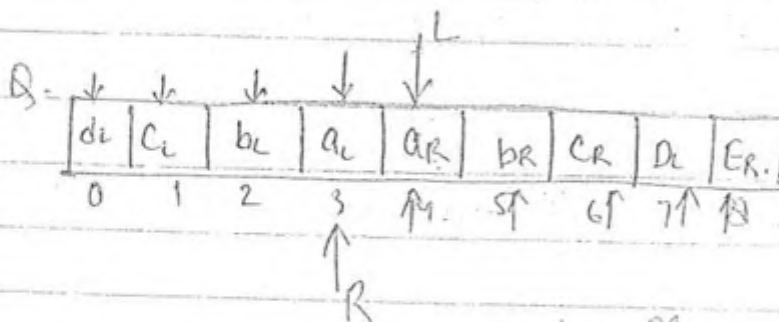
### ✓ Output Restricted Queue :-



### ✓ Double Ended Queue (DEQ) ⇒ It is data structure.



### Double Ended Queue (operation)



$$M=9$$

$$L = \frac{M}{2}$$

$$R = \frac{M}{2} - 1$$

### Insert left

```

if (L == 0) full
else
{
 L--
 Q[L] = x
}

```

### Deletion left

```

if (L == M/2) empty
else
{
 y = Q[L]
 L++
}
return(y);

```

### Insert right

```

if (R == M-1) full
else
{
 R++
 Q[R] = x
}

```

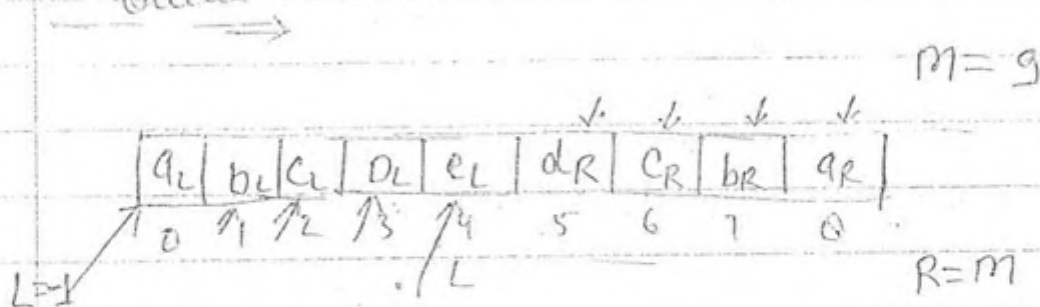
### Deletion right

```

if (R == M/2 - 1) empty
else
{
 y = Q[R]
 R--
}
return(y);

```

Efficient way of implementing Double Ended Queue



### Insert left

```

if (R == L+1) full)
else
{
 L++;
 Q[L] = u;
}

```

### Delete left

```

if (L == -1) empty.
else
{
 y = Q[L];
 L--;
}
return(y);

```

### Insert Right

```

if (R == L+1) full)
else
{
 R++;
 Q[R] = u;
}

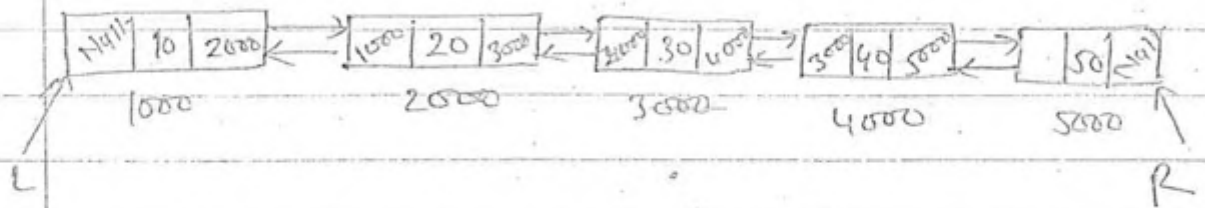
```

### Delete Right

```

if (R == m) empty.
else
{
 y = Q[R];
 R++;
}
return(y);

```



### Delete Left

① if  $(L == \text{Null})$  empty.

② if  $(L == R)$  free(L)  
 $L = R = \text{null}$

③  $L = L \rightarrow \text{rp}$   
 free( $L \rightarrow \text{lp}$ )  
 $L \rightarrow \text{lp} = \text{null}$

### Delete Right

① if  $(R == \text{Null})$  empty.

② if  $(L == R)$  free(L);  
 $L = R = \text{Null}$

③  $R = R \rightarrow \text{lp}$   
 free( $R \rightarrow \text{rp}$ )  
 $R \rightarrow \text{rp} = \text{Null}$

### Insert Left

$P \rightarrow \text{rp} = L$

$L = \text{lp} = P$

$L = P$

$P = \text{Null}$

### Insert Right

$R \rightarrow \text{rp} = P;$

$P \rightarrow \text{lp} = R;$

$R = P;$

$P = \text{Null};$

## Priority Queue

Insertion and Deletion can be done at any place depending upon Priority.

Ascending Priority:- We Always delete min<sup>m</sup> element each time.

|                   |                   |                   |                   |                   |                   |
|-------------------|-------------------|-------------------|-------------------|-------------------|-------------------|
| 50 <sub>5th</sub> | 45 <sub>4th</sub> | 15 <sub>2nd</sub> | 10 <sub>1st</sub> | 25 <sub>3rd</sub> | 60 <sub>6th</sub> |
| 0                 | 1                 | 2                 | 3                 | 4                 | 5                 |



Ascending Priority Queue

10, 15, 25, 45, 50, 60  
(min).

Descending Priority:- We Always delete max<sup>m</sup> element each time.

|                   |                   |                   |                   |                   |                   |
|-------------------|-------------------|-------------------|-------------------|-------------------|-------------------|
| 50 <sub>2nd</sub> | 45 <sub>3rd</sub> | 15 <sub>5th</sub> | 10 <sub>6th</sub> | 25 <sub>4th</sub> | 60 <sub>1st</sub> |
| 0                 | 1                 | 2                 | 3                 | 4                 | 5                 |



60, 50, 45, 25, 15, 10  
(max).

Implementing Priority Queue

- ① Unordered array
- ② Sorted array
- ③ Min heap or Max heap [heap Properties]
- ④ B<sup>+</sup> tree

} Using Array

↑  
Priority



## UnOrdered Array

|    |    |    |   |    |    |   |    |
|----|----|----|---|----|----|---|----|
| 60 | 50 | 90 | 5 | 25 | 30 | 1 | 15 |
| 0  | 1  | 2  | 3 | 4  | 5  | 6 | 7  |

### Insertion

$O(1)$

### Deletion

Find min in the given array of  $n$ -element and delete

finding min  $\downarrow$   
 $O(n) + O(n) \rightarrow$   $O(n)$  Sifting

## Sorted Array

60, 50, 90, 5, 25, 30, 1, 15, 90

|   |   |    |    |    |    |    |    |    |
|---|---|----|----|----|----|----|----|----|
| 1 | 5 | 15 | 25 | 30 | 50 | 60 | 70 | 90 |
| 0 | 1 | 2  | 3  | 4  | 5  | 6  | 7  | 8  |

### Insertion

$O(n)$

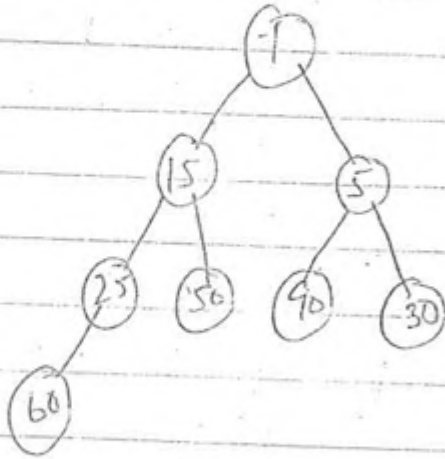
### Deletion

$O(1)$  [using Marker]

$O(n) + O(1)$   
 $\downarrow$   
 $O(n)$

## ✓ Min heap

60, 50, 90, 5, 25, 30, 1, 15



Deletion  
 $O(\log n)$

Insertion  
 $O(\log n)$

$O(\log n)$

Page 4  
8

## ✓ B<sup>+</sup> - Tree

Insertion  $\Rightarrow O(\log n)$

Deletion  $\Rightarrow \log n$

$O(\log n)$

Time

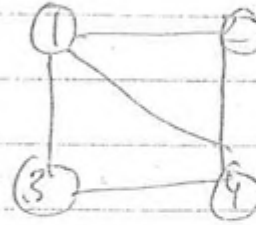
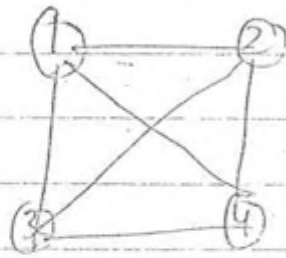
Insertion  $= O(\log n)$

Deletion  $= O(1)$

$\log n$

# Work Book

③



Planner

Page 4

⑧

Auxy =

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}$$

3rd.

↓

A<sub>24x4</sub>

$$\begin{bmatrix} a_{11} & 0 & 0 & 0 \\ a_{21} & a_{22} & 0 & 0 \\ a_{31} & a_{32} & a_{33} & 0 \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}$$

B<sub>24x4</sub>

$$\begin{bmatrix} b_{11} & 0 & 0 & 0 \\ b_{21} & b_{22} & 0 & 0 \\ b_{31} & b_{32} & b_{33} & 0 \\ b_{41} & b_{42} & b_{43} & b_{44} \end{bmatrix}$$

$$B^T = \begin{bmatrix} b_{11} & b_{21} & b_{31} & b_{41} \\ 0 & b_{22} & b_{32} & b_{42} \\ 0 & 0 & b_{33} & b_{43} \\ 0 & 0 & 0 & b_{44} \end{bmatrix}$$

$$C = \begin{bmatrix} a_{11} & b_{11} & b_{21} & b_{31} & b_{41} \\ a_{21} & a_{22} & b_{22} & b_{32} & b_{42} \\ a_{31} & a_{32} & a_{33} & b_{33} & b_{43} \\ a_{41} & a_{42} & a_{43} & a_{44} & b_{44} \end{bmatrix}$$

$$b[3,1] = c[1,4]$$

$$i, j \quad j; i+1$$

(10)

$$i = 4, 5, 6:$$

$$a[i+1] = a[i] \quad \text{Side effect}$$

$$a[6] = 6$$

(12)

$$n = 1000 \Rightarrow 10 \text{ ms}$$

$$1000 * \log_{10} 1000 \Rightarrow 10 \text{ ms}$$

$$3000 C = 10 \text{ ms}$$

$$C = \frac{10}{3000} \text{ ms}$$

## COUNTABLE & UNCOUNTABLE SETS.

Date - 21/07/10

1)  $\Sigma^* \rightarrow$  It is a discrete set,

2)  $L \subseteq \Sigma^*$

### Lexicographic order

$\hookrightarrow$  every language is a grammar.

$\hookrightarrow$  Recursive enumerable is a Turing enumerable.

$\hookrightarrow$  below RE language in chomsky hierarchy, all are Turing enumerable.

3)  $\hookrightarrow 2^{\Sigma^*}$  contains every possible language, (regular, cfl, csl. ...)  
but, It is Uncountable.

4)  $\hookrightarrow L_{RE} \xrightarrow{\text{map}} \text{Set of all RE language and It is "Countable Infinite"}$

5)  $\hookrightarrow \text{Set of all Turing M/c is Countable Infinite. and discrete.}$

for eg:-

$$Q = \{q_0, q_1, q_2\}$$

$$\Gamma = \{a, b, \square\}$$

$$\{L, R\}$$

6)  $L_{RE}, L_{CFG}, L_{CSG}, L_{REG}$  all are Countable Infinite.

### (CANTOR'S DIAGONALISATION THEOREM)

If  $S$  is Countable Infinite

Then,  $2^S$  is Uncountable Infinite.

Let  $S = \{a, b\}$

↳ A real number is uncountable Infinite.

↳ All the language are Countable.

not only  
Infinite

Which are Countable Infinite and Uncountable Infinite.

only,  $2^{\Sigma^*}$  and  $L_{not RE}$  is Uncountable Infinite.

↳ every Turing M/C is in discrete Set. and it can also perform Computation.

→ Gödel's Incompleteness Theorem :- famous Mathematician that told math is Incomplete



## Properties of REC and RE languages:-

1) RE defn:- A language is RE if and only if it accepts or there

exists a Turing machine.

↳ It is also called as semidecidable.

REC  
1) defn:- A language is REC if and only if there exists a Turing machine, <sup>i.e. (TM)</sup> and which halts on every string 'w' where,  $w \in \Sigma^*$ .

↳ It is also called as decidable,

↳ when 

|    |   |         |   |              |
|----|---|---------|---|--------------|
| TM | → | accepts | → | Halt         |
|    | → | rejects | → | Halt or Hang |

Limited version of RE is REC.

- 2) Both RE and REC are enumeration procedure, that run - an TM or Turing <sup>enumerable</sup>
- 3) While Both RE and REC are not membership algorithm
- 4) Only REC have a membership algorithm.

→ REC  
→ RE  
→ CSL  
→ CFL  
→ RL

→ all are Turing enumerable

⇒ Since, L not RE is enumeration procedure, but not Turing enumerable

→ If both  $L$  and  $L^c$  are Turing enumerable, then  $L$  is REC.

⇒ If both  $L$  and  $L^c$  is RE, then both are REC.

→ If  $L$  is RE, then  $\bar{L}$  may or may not be RE, because of closure property.

⇒ If  $L$  is REC, then  $\bar{L}$  is REC.

→ If  $L$  and  $\bar{L}$  are complementary language,

Case I) Both  $L$  and  $\bar{L}$  is REC. i.e. not RE.

Case II) Both  $L$  and  $\bar{L}$

$L$  is RE and not REC,  $\bar{L}$  is not RE  
means, one of them is RE and not REC, and other is not RE.

(DECIDABILITY)

RICE'S Theorem :-

Every Non Trivial question regarding RE language is Undecidability.

$L_1$  is RE

$L_2$  is RE

$L \cup L_c$  is RE ? ⇒ decidable

RE language  $\rightarrow$  Regular?

$\rightarrow$  Using RICE'S Theorem, above one is Undecidable.  
Such question is non Trivial question.

Non Trivial  $\rightarrow$  Undecidable.

Trivial Question  $\rightarrow$  decidable.

$L_1 \rightarrow$  which RE  $\rightarrow$  Regular?

$L_2 \rightarrow$  which is RE  $\rightarrow$  Regular?

} o/p is

Trivial question

Thus decidable

particular RE  $\rightarrow$  decidable

arbitrary RE  $\rightarrow$  Undecidable

$\rightarrow$  Every finite language is decidable.)

Undecidable Problems for Semidecidable Problems

- 1) Halting Problem (HP)
- 2) Blank Tape Halting Problem (BTHP)
- 3) State entry problem (SEP)
- 4) Post correspondence problem (PCP)
- 5) Modified post correspondence problem (MPCP)
- 6) RE membership (REM)

1) HP :-  $TM(M, w) \in \Sigma^*$ ,  $M$  will Halt or Hang?

↳ Undecidable or Semidecidable

2) BTHP :-

1001001001

Thus,  $TM(M, -) \in \Sigma^*$ ,  $M$  will Halt or Hang?

↳ Undecidable or Semidecidable

3) SEP :-  $TM(M, w)$ ,  $M$  will enter to particular state ( $q$ )?

↳ Undecidable or Semidecidable

for  $P_1 \leq P_2$   
 $P_1(UD) \Rightarrow P_2(UD)$   
 $P_2(D) \Rightarrow P_1(D)$

HP  $\leq$  BTHP  
 HP  $\leq$  SEP  
 HP  $\leq$  PCP  
 REM  $\leq$  MPCP  
 (UD) (UD)

4) PCP :-

A PCP Solution is <sup>un</sup>decidable,

when  $(u_i, u_j, u_k \dots) = (v_i, v_j, v_k \dots)$

~~Not~~ ~~Undecidable~~ or ~~Semidecidable~~

↳ decidable for only Unary alphabet  $\{1\} = \Sigma$

$\rightarrow G_1 = G_2$  i.e. equivalent of. Grammar is undecidable,

for eg:-  
 $(u_1, u_2, u_3) \rightarrow \text{Set 1}$   
 $(10, 00, 110)$   
 $(v_1, v_2, v_3) \rightarrow \text{Set 2}$   
 $(11, 001, 0110)$

PCP solution exist?

$u_2 \rightarrow 00$   $u_1$   
 $10$   
 $v_2 \rightarrow 001$   $v_1$   
 $11$   
 $v_3$

Therefore

$u_2$   $u_3$   
 $00$   $110$   
 $001$   $110$   
 $v_2$   $v_3$

It means there exist a PCP solution

Thus,  $\epsilon$  undecidable,

MPCP:-

A solution is said to be MPCP,

$$u_i u_j u_k = v_i v_j v_k$$

eg:- (RE but not REC)

universal  
g. language

binary code

A language is RE but not REC  $\Rightarrow L_u = \{ T \mid e(T) \in L(T) \}$

A language is not RE  $\Rightarrow L_d = \{ T \mid e(T) \notin L(T) \}$

eg of

diagonalisation  
language

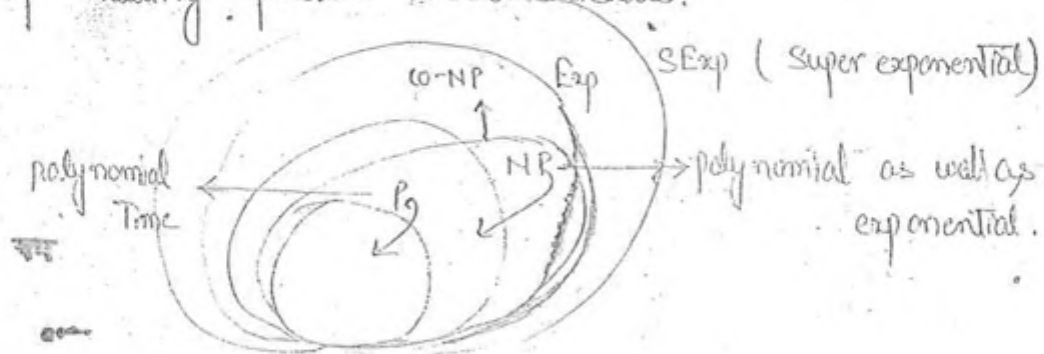
$\rightarrow$  decidable for only binary alphabet, i.e.  $\Sigma = \{0, 1\}$

# Computational Complexity

↳ Decidable

⇒ In this topic only speed matter.

↳ A Halting problem is Undecidable.



fast speed:  $\leftarrow$  P-class problem  $\Rightarrow$  U Deterministic ( $n^i$ )

↳ It is also called as "Tractable" or "Cook-Karp Thesis".

slow speed  $\leftarrow$  NP-class problem  $\Rightarrow$  U nondeterministic ( $n^i$ )

↳ It is also called as "Intractable".

⇒ Mainly NP is said to be exponential.)

$$\Rightarrow \boxed{P \subseteq NP}$$

⇒  $\boxed{P \subset NP} ? \rightarrow$  It is not sure | prove till now

$P = NP ? \rightarrow$  It is also not sure | prove till now

↳ In IIT Bombay, It is proved that,

$$\underline{(P \neq NP)}$$



~~NP Complete~~

$$\Rightarrow \boxed{Exp = \bigcup \text{Nondeterministic}(K^n)}$$

NP Hard

defn:- If a problem (L) is NP Hard, iff, for all the languages (L')

such that,  $L' \in NP$

$$\& \quad L' \leq_p L$$

$\hookrightarrow$  polynomial time reduction.

NP Complete

defn:- If a problem (L) is NP Complete, if and only if, for all the language (L'),

such that  $L' \in NP$

$$L' \leq_p L$$

$$\& \quad L \in NP$$

✓ every NP Complete problem is NP Hard but converse is not True.

$$\text{i.e. } \boxed{L \in NPC \Rightarrow L \in NPH}$$

$\hookrightarrow$   $\boxed{\text{If any 'NP' Complete problem belong to 'P' i.e. } NP \in P, \text{ then } P = NP.}$

## Closure property

↳ 'P' is closed under  $(\cup, \cap, L^c, \cdot, *)$

↳ 'NP' is closed under  $(\cup, \cap, \cdot, *)$

for  $NP \rightarrow L^c$  is open problem.

## Co-NP

$$\text{Co-NP} = \{ \bar{L} \mid L \in \text{NP} \}$$

Theorem 2

If NP is closure under  $L^c$  then  $\text{NP} = \text{Co-NP}$

Theorem 3

If  $\text{NP} = \text{Co-NP}$  then  $P = \text{NP} \xrightarrow{\text{Co-NP}} \text{false}$ .

Theorem 4

If  $P = \text{NP}$  then  $\text{NP} = \text{Co-NP} = P \rightarrow \text{True}$ .

$$P \subseteq \text{NP} \cap \text{Co-NP} \rightarrow \text{True}$$

$$\text{If } L_1 \leq_P L_2 \text{ \& } L_2 \leq_P L_3 \Rightarrow L_1 \leq_P L_3$$

~~$L_2 \leq_P L_3$~~

if for,  $L_1 \leq_p L_2$ , then,

if  $L_2$  is Decidable  $\xRightarrow{\text{then}}$   $L_1$  is decidable

if  $L_2$  is RE  $\xRightarrow{\text{then}}$   $L_1$  is RE

if  $L_2$  is REC  $\xRightarrow{\text{then}}$   $L_1$  is REC

if  $L_2$  is P  $\xRightarrow{\text{then}}$   $L_1$  is P

if  $L_2$  is NP  $\xRightarrow{\text{then}}$   $L_1$  is NP

2)

for,  $L_1 \leq_p L_2$ ,

if  $L_1$  is NPH  $\xRightarrow{\text{then}}$   $L_2$  is NPH

if  $L_1$  is NPC  $\xRightarrow{\text{then}}$   $L_2$  is NPH

if  $L_1$  is NPC &  $L_2$  is NP  $\xRightarrow{\text{then}}$   $L_2$  is NPC

Ans

$\forall L_1 \notin \text{NPH}$

$L_1 \leq_p L_2 \in \text{NPH}$

$\Rightarrow L_2$  is NPC  $L_2 \leq_p L_3$  &  $L_3$  is NP.

then o/p  $L_3$  is NPC

$\hookrightarrow$  every NP Hard problem is reducible the NP Hard  $\rightarrow$  True.

But it NPC

$\rightarrow$  false or may or may not

## NP-Complete problem :-

Bounded Tiling  
↓

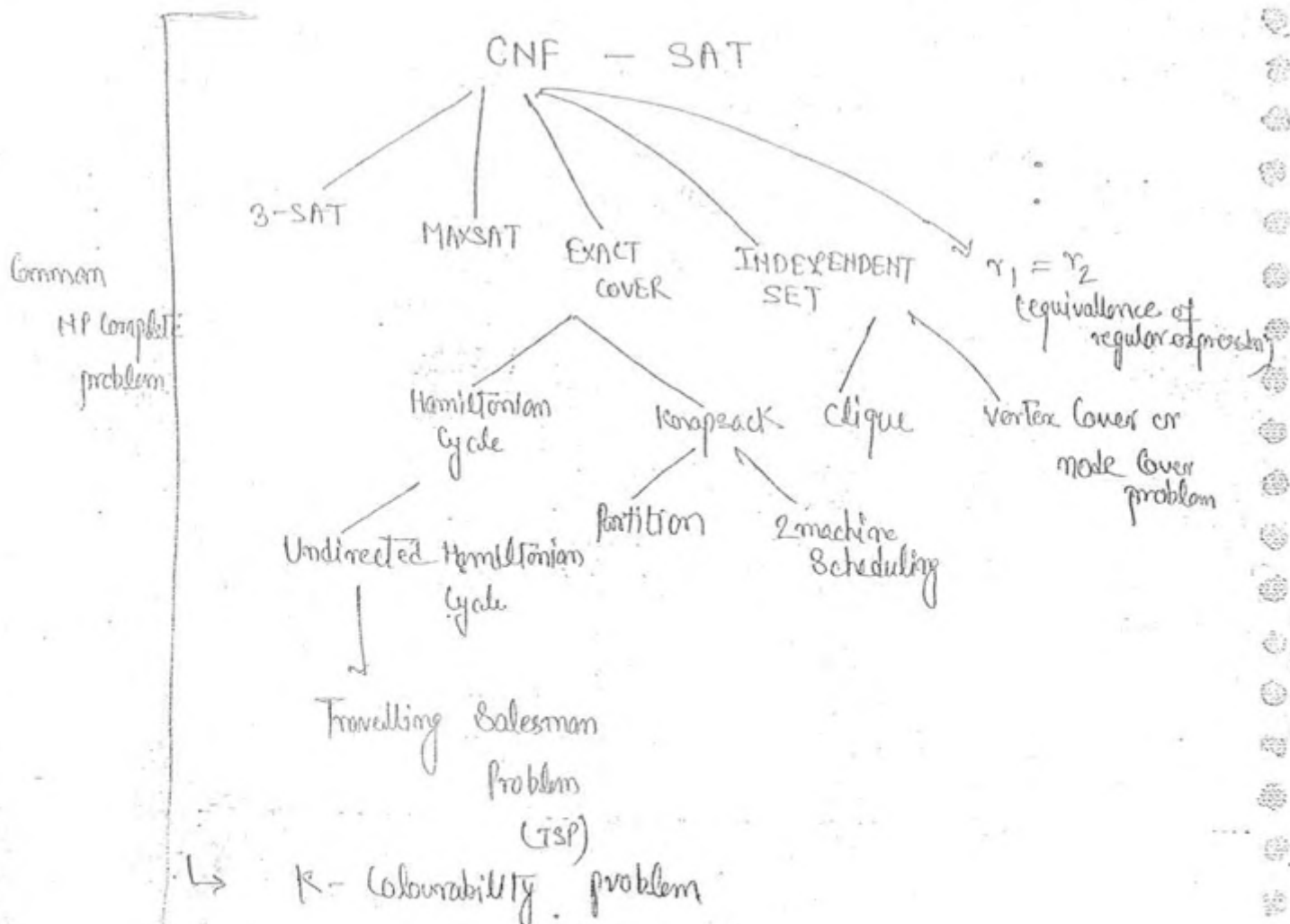
CNF-SAT | satisfy Conjunctive normal form

$$\therefore \boxed{\text{Tautology} + \text{Contradiction} \Rightarrow \text{SAT (satisfy)}}$$

0 ← Contradiction ⇒ UNSAT

1 ← Tautology ⇒ SAT

$(T + C) \Rightarrow$  Some time SAT or some time UNSAT.

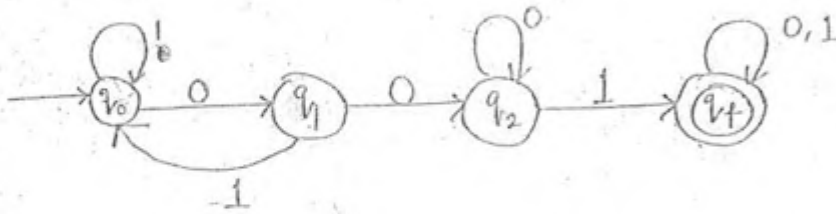


| <u>P - problem</u>                                                                                                                                            | <u>NPC</u>                                                                                                                                                                                                                                  |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>↳ 2-SAT</p> <p>↳ Unary partition</p> <p>↳ Shortest path.<br/>(Dijkstra's Algo)</p> <p>↳ 2- Colourability</p> <p>↳ equivalence of Dfa's</p> <p>↳ MINCUT</p> | <p>↳ 3-SAT</p> <p>↳ partition (general)</p> <p>↳ longest path.</p> <p>↳ K- Colourability.</p> <p>↳ equivalence of NFA's &amp;<br/>Regular expression,<br/>i.e. <math>M_1 = M_2</math><br/>only <math>r_1 = r_2</math>.</p> <p>↳ MAXCUT.</p> |

THE END

## Ques 1

i) String that "containing the substring 001"



$$R.E. \Rightarrow (1 + 01 + 001)^* (\lambda + 0 + 00)$$

(True or False)

ii) By Using Arden's Theorem,

$$\text{If } R = Q + RP, \Rightarrow R = QP^*$$

$$q_0 = q_0 1 + q_1 1 + \lambda \quad \text{--- (1)}$$

$$q_1 = q_0 0 \quad \text{--- (2)}$$

$$q_2 = q_1 0 + q_2 0 \quad \text{--- (3)}$$

$$q_f = q_2 1 + q_f (0 + 1) \quad \text{--- (4)}$$

Sol

from eq (4)

$$1) \quad q_f = q_2 1 + q_f (0 + 1)$$

$$R = Q + RP \Rightarrow R = QP^*$$

$$\boxed{q_f \Rightarrow q_2 1 (0 + 1)^*}$$

from eq (3)

$$2) \quad q_2 = q_1 0 + q_2 0$$

$$\rightarrow \boxed{q_2 = q_1 0 0^*}$$

$$\begin{cases} q_f = q_1 0 0^* 1 (0 + 1)^* \\ \Rightarrow q_0 = q_0 1 + q_1 1 + \lambda \\ \text{Since, } q_1 = q_0 0 \end{cases}$$



## Application of Nfa & Dfa.

Date:- 13 Sep, 2010

- 1) Lexical Analysis { scanner plays an important role in it }
- 2) Text editor { find, replace }
- 3) Units group { pattern search }
- 4) Spell checker
- 5) String Match is not possible,

↳ pattern matching is done by finite automata while,  
finite automata cannot do string matching.

## Application of Moore & Mealy M/c.

↳ Sequential Circuit design.  
or  
digital Circuit design.

## Variation of dfa / nfa.

$$\delta(q_0, a) = \{ (q_1, \overset{(L)}{b}), (q_2, \overset{(R)}{a}) \}$$

L → Left  
R → Right

(Variation of dfa, nfa)

$$1) \quad 2 \text{ dfa, nfa} \rightarrow \text{dfa, nfa} + R \text{ or } L$$

$$2) \quad \text{dfa, nfa} + R \leftrightarrow L + R/w \equiv TM.$$

$$3) \quad \text{dfa, nfa} + \text{stack} \rightarrow PDA$$

$$\hookrightarrow \text{nfa} + \text{stack} \xrightarrow{\text{(equivalent)}} NPDA.$$

$$\hookrightarrow \text{dfa} + \text{stack} \xrightarrow{\text{(equivalent)}} DPDA.$$

$$4) \quad \text{dfa} / \text{nfa} + 2\text{stack} \xrightarrow{\text{(equivalent)}} TM$$

$$5) \quad \text{dfa} / \text{nfa} + 2\text{counter} \equiv TM$$

Counter  $\rightarrow$  It is nothing but a stack.

$$\text{i.e. } \Sigma = \{z, 1\}$$

$\downarrow$  counter  
 $\uparrow$  stack symbol

$$6) \quad \underbrace{\text{dfa/nfa} + 1 \text{ counter}}_{\downarrow \text{less powerfull.}} < \underbrace{\text{dfa/nfa} + 1 \text{ stack}}_{\downarrow \text{more powerfull.}}$$

(nfa + 1 counter is better than nfa.)

Note:

while,

$$\text{nfa} + 1 \text{ counter} < (\text{nfa} + 1 \text{ stack}) \rightarrow CFL$$

$$\text{nfa} + 2 \text{ counter} \equiv \text{nfa} + 2 \text{ stack}$$

~~nfa + 1 stack~~

Note:

CFL

TM

nfa & dfa < nfa + 1 counter

< nfa + 1 stack

< nfa + <sup>counter</sup>~~2 stack~~  $\equiv$  nfa + 2 stack

## (Chapter - 2)

### { Context free Languages } (INDEX)

- 1) Standard CFL's, grammar,  $CFL \rightarrow L$   
 $L \rightarrow CFL$
- 2) Derivation, L.M. Derivation & R.M. Derivation.
- 3) Derivation Tree, Ambiguity of Grammar & Languages, Partial derivation tree, Sentential form.
- 4) PARSING: Membership Algorithm,  $\swarrow$  BFP  
 $\searrow$  CYK algo  
Complexity, Programming language & TOC,  $O(n)$   
Linear time Compiler, S grammar, LL grammar,  
LR grammar.
- 5) Algorithm of CFL's, 7 algorithm, 1) Removal of  $\epsilon$ -production  
2) Removal of unit production;  
3) " " useless production;  
4) " " LR  
5) " " L factoring  $\rangle$  Remove ambiguity.

6)  $CFL \rightarrow CNF$

7)  $CFL \rightarrow GNF \rightarrow PDA$

In Greibach normal form, length is of " $n$ "  
while, In Chomsky normal form, length is of " $2n - 1$ ".

5) PDA :- Programming of DPDA & NPDA

$CFL \rightarrow PDA \Rightarrow$  Algorithm is used to.  
Convert Context free language  
into PDA.

6) Closure & decidability of CFL's, pumping Lemma for  
CFL & Linear.

$\hookrightarrow$

(Context free language).

$\hookrightarrow$  Every regular language is Context free language.

while Context free language is not regular language.

for eg:-

$a^n b^n : n \geq 0$

1) Grammar:-  $S \rightarrow aSb \mid \lambda$   
 $\hookrightarrow$  very restricted in nature

$w \mid n_a(w) = n_b(w)$   
called balanced parentheses

$S \rightarrow (aSb \mid bSa \mid SS \mid \lambda)$

$\epsilon \rightarrow (aSb \mid SS \mid \lambda)$   
properly balanced parentheses

$w w^R$

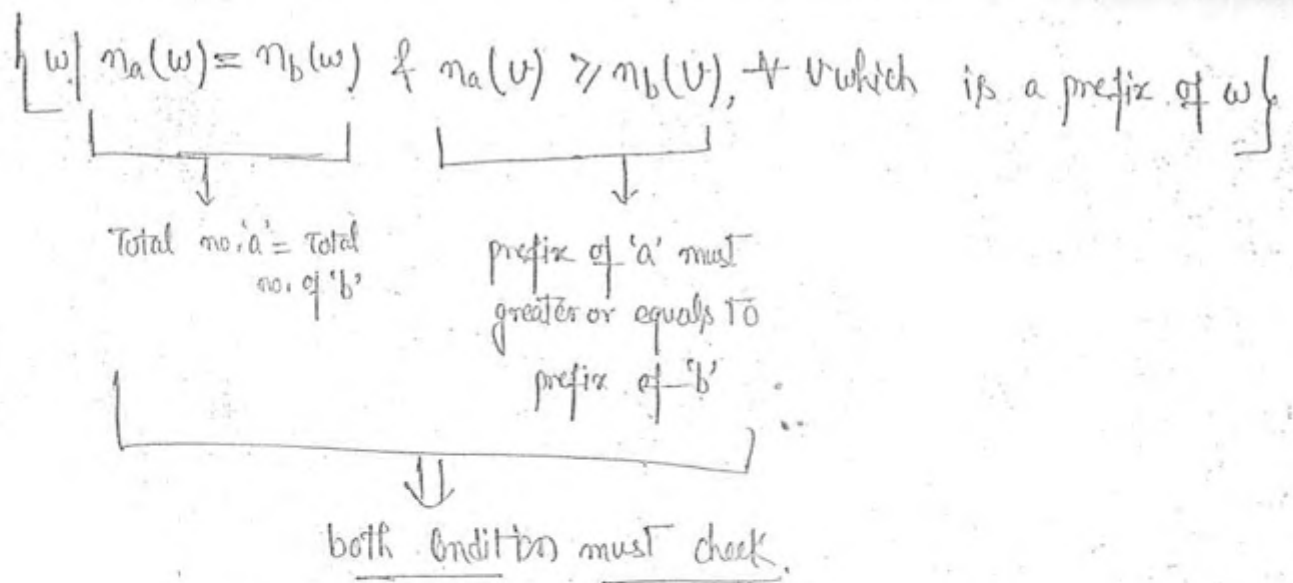
$\rightarrow w (a^R b^R) w^R$

old palindrome

$S \rightarrow aSa \mid bSb \mid a \mid b$

even palindrome

$S \rightarrow aSa \mid bSb \mid \lambda$



Note:- (properly balanced is always ~~sub~~ proper subset i.e. c of balanced parameters.)

A finite automata can not do counting, Thus its regular expression always comes under, either  $V^*T$  or  $T^*V$ .

But,  $a^*b^*$  or  $a^*sb^*$  is only come under, C.F.L because it can do counting.

$$V \rightarrow T^*V + VT^* + (T^*VT^*)$$

$w^k$  is always in the form of ~~w~~ palindrome;



$$L_1 = \{ ww^R \mid w \in (a,b)^* \} \rightarrow \text{Even palindrome}$$

$$L_2 = \{ w(a+b)w^R \mid w \in (a,b)^* \} \rightarrow \text{odd palindrome}$$

$$L_3 = \{ w \mid w = w^R \} \rightarrow \text{All palindrome. (even + odd)}$$

Thus,

$$L_3 = L_1 + L_2 \text{ or } L_1 \cup L_2$$

eg:  $a^n b^m$

$$a^n b^m; m \geq 1$$

Gramm. :-  $S \rightarrow a^m a s b \mid a b$

eg:  $a^n b^{2n}$

$$a^n b^{2n}; n \geq 1$$

Gram:-  $S \rightarrow a s b b \mid \lambda$

eg:  $a^{2n} b^n$

$$a^{2n} b^n; n \geq 0$$

Gramm:-  $S \rightarrow a a s b \mid \lambda$

eg:  $a^m b^n$

$$\{ a^m b^n; m \geq n, 4m = 5n+1 \}$$

$$\{ a^m b^{5n+1}; m \geq 0 \} \text{ or } \{ a^m b^{5n+1}; n \geq 0 \}$$

we always take condition by default  $n, m \geq 0$  if not given.

for eg:-

$$a^n b^m ; n \geq m ; n, m \geq 0$$

$$S \rightarrow AS_1$$

$$S_1 \rightarrow aS_1 b \mid \lambda$$

$$A \rightarrow \cdot aA \mid \lambda \rightarrow a^*$$

$$S \rightarrow a^* a^n b^n \rightarrow \text{This is nothing but,}$$

$$a^{n+m} b^n \text{ or } a^{n+m} b^n$$

$$\text{Thus, } \left( a^{n+m} b^n ; n \geq m ; n, m \geq 0 \right) \text{ This is equivalent to } \left( a^n b^m ; n \geq m ; n, m \geq 0 \right)$$

Ques

$$S \rightarrow AS_1 \mid S_1 B$$

$$S_1 \rightarrow aS_1 b \mid \lambda$$

$$A \rightarrow aA \mid \alpha \quad \alpha \quad a$$

$$B \rightarrow bB \mid \beta \quad \beta \quad b \quad \lambda$$

Condition to check

Sol

$$S \rightarrow AS_1 \rightarrow A a S_1 b$$

$$\rightarrow A A a S_1 b$$

$$\rightarrow \underline{a a a a b}$$

$$\Rightarrow a^n b^m$$

$$\text{Thus, } [a^n b^m] \begin{cases} \text{if } n \geq m \text{ or } n < m \\ \text{if } n \neq m \end{cases}$$

Ques.

$$S \rightarrow AS_1 \rightarrow n-m = 2k$$

$$S_1 \rightarrow as_1b \mid \lambda \rightarrow n=m$$

$$A \rightarrow aaaS_1 \mid aa$$

Sol.

$$S \rightarrow AS_1 \rightarrow aaaS_1 \rightarrow AS_1$$

$$\rightarrow aaaaaS_1b \rightarrow AaS_1b$$

$$\rightarrow \underline{aaaaa}b \rightarrow \underline{aaaa}bb$$

$$\text{final O/P} \Rightarrow (a^n b^m; n-m = 2k); k \geq 1$$

Ques

$$n_a(w) = 2n_b(w)$$

$$S \rightarrow aasb \mid asbb \mid ss \mid \lambda$$

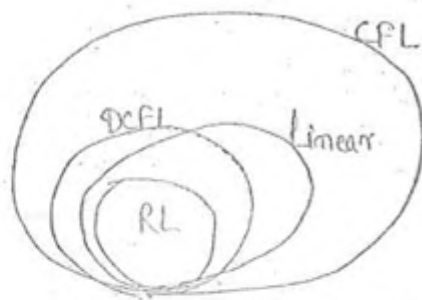
Possible  
Combination  
or order

$$S \rightarrow \begin{matrix} a \overset{\substack{\text{S can be put here (previous or after)}}}{S} a S b \\ a S b S a \\ b S a S a \mid \lambda \end{matrix}$$

for eg:-  $[w \# w^R \mid w \in \{a, b\}^* \& \# \notin \{a, b\}^*]$

Grammar

$$S \rightarrow aSa \mid bSb \mid \#$$



from the above fig, every DCFL and Linear grammar contains Regular grammar and all three are under Context free language,

Linear grammar

$$N \rightarrow T^* + VT^* + T^*V + T^*VT^*$$

- ↳ Every DPDA can do linear comparison.
- ↳ Push and pop is not possible in DPDA.
- ↳ while, Push and pop is possible in HPDA.

for eg:-  $\frac{aab}{\text{push}} \mid \frac{baa}{\text{pop}}$

$\frac{a}{\text{push}} \rightarrow a$   
 $\frac{b}{\text{push}} \rightarrow b$   
 $\frac{a}{\text{pop}} \rightarrow b$   
 $\frac{b}{\text{pop}} \rightarrow a$

[ If  $a \rightarrow a$  comes push  
 If  $a \rightarrow b$  comes pop ]

$\Rightarrow S \rightarrow asb \mid \lambda \Rightarrow$  Linear grammar.

$\Rightarrow S \rightarrow asb \mid bsa \mid ss \mid \lambda \Rightarrow$  Non linear grammar:  
but  
DCFL.

eg:- if  $w \in R$

$S \rightarrow asb \mid bsb \mid \lambda$

It is CFL but not  
DCFL.

ii)  $w \mid n_a(w) = n_b(w) \rightarrow$

It is not linear, but it  
is DCFL.

iii)  $a^n b^n : n > 0 \rightarrow$

it is linear as well as DCFL.

eg:-  $a^m b^n ; m \leq n \leq 3m$

or linear CFL

$\hookrightarrow$  It is CFL, but not DCFL.

$S \rightarrow asb \mid asbb \mid asbbb \mid \lambda$

$\rightarrow S \rightarrow asb \rightarrow aasbb \rightarrow aabb$

$\rightarrow S \rightarrow asb \rightarrow aasbbb \rightarrow aobbb$

Ques find a language correspond to the grammar:-

$$G \Rightarrow \begin{cases} S \rightarrow aaB \\ A \rightarrow aaBa \\ B \rightarrow bbAa \\ A \rightarrow \lambda \end{cases}$$

$L = ?$

$$\rightarrow S \rightarrow aaB \rightarrow aabbAa \rightarrow aabbbaBa$$

$$\begin{array}{l} \rightarrow aB \\ \rightarrow aabbAa \\ \rightarrow aabbbaBa \\ \rightarrow aabbbaBaAa \\ \rightarrow aabbbaBaAaAa \\ \rightarrow aabbbaBaAaAaAa \end{array} \quad \begin{array}{l} \rightarrow aabbbaBaAa \\ \rightarrow aabbbaBaAaAa \\ \rightarrow aabbbaBaAaAaAa \end{array}$$

$$S \rightarrow xSy \mid z$$

$$\rightarrow x^n z y^n \quad [S \xrightarrow{n} \text{remove by } z]$$

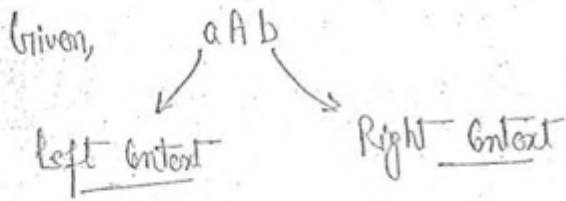
$$S \rightarrow ab \underbrace{(bbaa)}_{PQ}^n \underbrace{bba(ba)}_{PQ}^n ; n \geq 0$$

$$\therefore \hookrightarrow B \rightarrow (bbaa)^n bba(ba)^n \quad \rightarrow \text{Taking a Common}$$

$$(PQ)^n P = P \cdot (QP)^n$$

$$\Rightarrow \{ aabb (aabb)^n (ab)^n a ; n \geq 0 \}$$





→ If the grammar is Context Sensitive, then we can write the grammar as :-

$$x^T A y \rightarrow x^T A^T y$$

for eg:- if  $aAb \rightarrow aAab$   $\therefore A \rightarrow Aa$ .

$$\text{ii) } bAc \rightarrow bBc,$$

iii)  $\begin{matrix} x & y \\ aA\lambda \end{matrix} \longrightarrow \begin{matrix} x & \lambda \\ \lambda Aa \end{matrix}$   
left context      right context

$$\text{ing } aA \rightarrow \lambda A \lambda$$

Note :-

↳ Every context free grammar is context sensitive grammar.

Derivations

defn:  $w \in L(G)$  if and only if at least one derivation for it using production of  $G$ .

LMD { left most Derivation } :- left most choice.

for eg:-

$$\begin{array}{lcl}
 S \rightarrow AB & (aa^*b^*) & S \rightarrow AB \\
 A \rightarrow aaA \mid \lambda \rightarrow aa^* & & \rightarrow aaAB \\
 B \rightarrow bB \mid \lambda \rightarrow b^* & & \rightarrow aaABb \\
 & & \rightarrow \underline{aab}
 \end{array}$$

$$aab \in L(G)$$

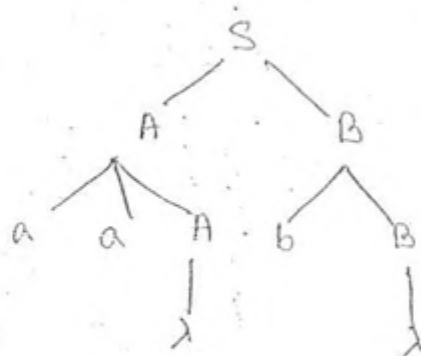
L.M.D.  $\Rightarrow S \Rightarrow AB \xrightarrow{\text{(Left most)}} aaAB \rightarrow aaB \rightarrow \underline{aab}$

we cannot give B value because why we use  $\lambda$ .  
 $LMD = RMD = aab$  i.e. we use  $\lambda$ .

R.M.D.  $\Rightarrow S \Rightarrow AB \xrightarrow{\text{(Right most)}} Abb \rightarrow Ab \Rightarrow aaAb \rightarrow \underline{aab}$

Derivation  $\Rightarrow S \Rightarrow AB \rightarrow aaAB \rightarrow aaABb \rightarrow \underline{aab}$   
 or  
 (neither LMD nor RMD)

(Derivation Tree)



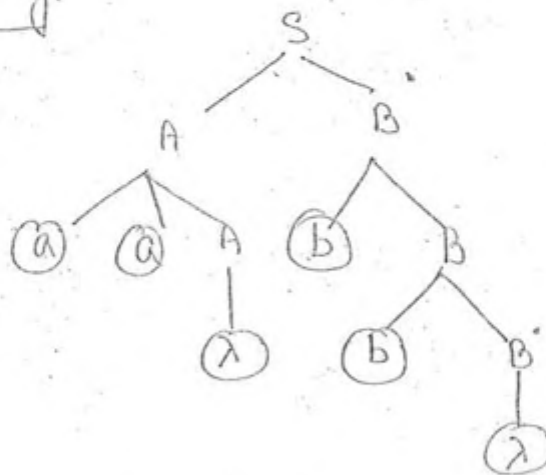
↳ A derivation Tree must always unique.

↳ In derivation Tree, ambiguity is not allowed.

↳ It has 1 LMD and 1 RMD,

↳ In derivation tree, from left to right move, towards leaf nodes it always gives the string.

for eg:-



String  $\Rightarrow$  aa bb

Note:

YIELD of a derivation tree always gives Sentence.

YIELD of a Partial derivation Tree always gives Sentential form.

↳ Starting of derivation tree is always started from 'S' (Root node)

↳ While Root of partial derivation tree can be any one in the tree, not necessary of 'S' (Root node)  
i.e. (Root must be any where in the tree)

→ A grammar 'G' is unambiguous if and only if

$\forall w \in L(G)$ , w must have exactly one derivation tree,

→ A grammar 'G' is ambiguous, if and only if

~~(at least)~~  $\exists w \in L(G)$ , 'w' must have at least two or more derivation tree.

Defn

A grammar 'G' is unambiguous, if and only if

$\forall w, w \in L(G)$ , w must have exactly one LMD. or one RMD.

→ A language 'L' is inherently ambiguous if and only if every grammar that generates 'L' is ambiguous.

for eg:-  $[a^m b^m c^n]$

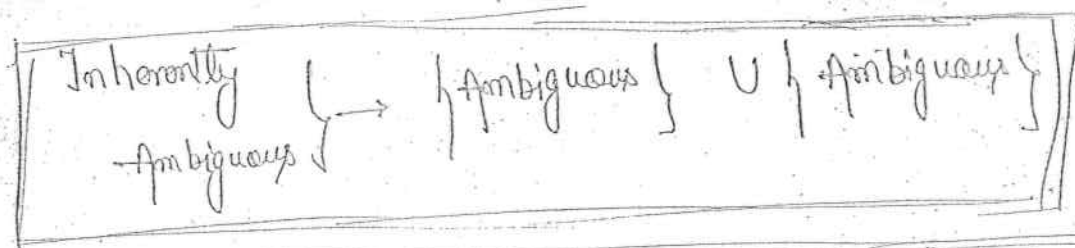
$S_x \rightarrow S_1 C$   
 $S_1 \rightarrow a S_1 b \mid \lambda$   
 $C \rightarrow c C \mid \lambda$

$U [a^n b^m c^m]$

$S_y \rightarrow A S_1$   
 $S_1 \rightarrow b S_1 c \mid \lambda$   
 $A \rightarrow a A \mid \lambda$

Inherently ambiguous

i.e.



~~for eg:-~~

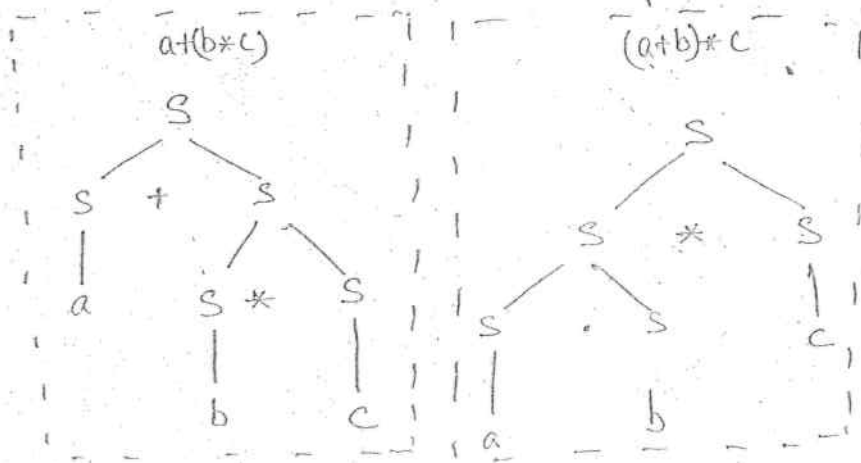




Date:- 14/09/10

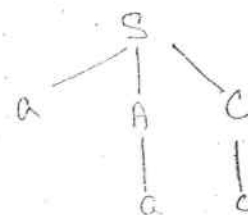
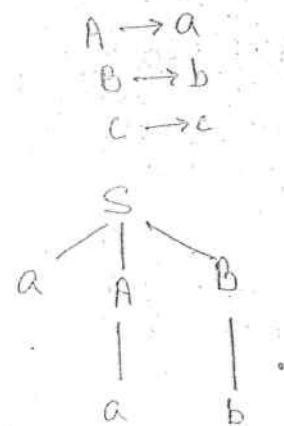
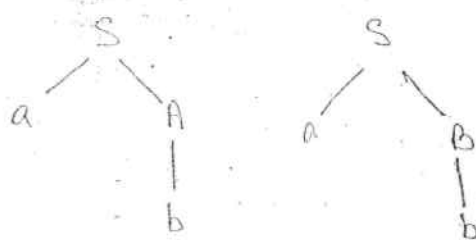
⇒ Ambiguous grammar:-

eg:-  $S \rightarrow S+S \mid S*S \mid a \mid b \mid c$



⇒ Left factoring Ambiguity:-

eg:-  $S \rightarrow aA \mid aB$  or,  $S \rightarrow aAB \mid aAC$   
 $A \rightarrow b$  or,  $S \rightarrow Aa \mid Ab$   
 $B \rightarrow b$



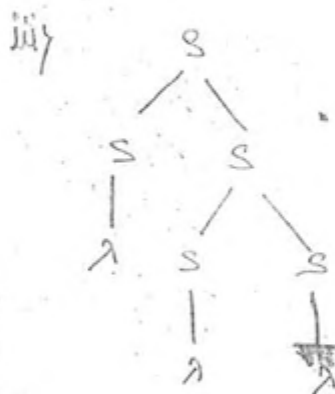
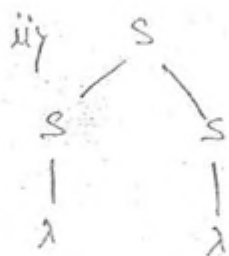
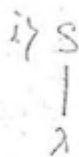
↳ If two SS is given, then there is more chance of Ambiguity.

for eg:-

$$S \rightarrow a | b | SS | \lambda$$

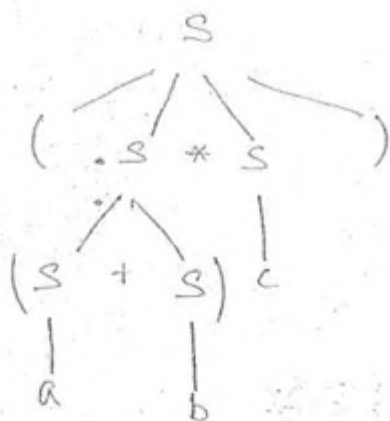
O/p  $\Rightarrow$  Ambiguous

derivation Tree

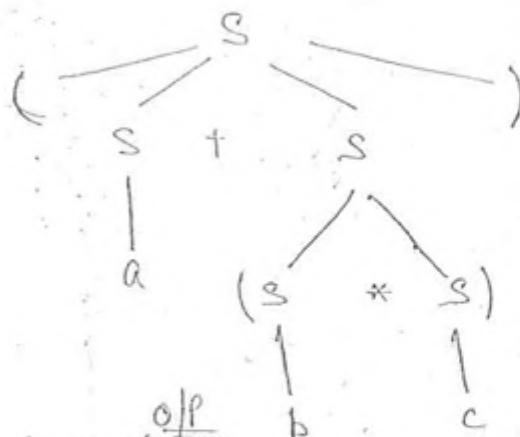


eg ii)

$$S \rightarrow (S+S) | (S*S) | a | b | c$$



O/p  $\Rightarrow (a+b)*c$



O/p  $\Leftarrow (a+(b*c))$

Since  $((a+b)*c) \neq (a+(b*c))$

It is not ambiguous

## PARSING & Membership Algo.

→ [Another name of derivation tree is called "Parse Tree".]

→ A membership algo. represents,

$$\neg \omega; \omega \in \Sigma^* \text{ \& } \omega \in L(G) :- \text{Yes or NO}$$

$\omega \rightarrow$  Word

→ [If we remove these two production from any set of production then grammar will generate expandable string.]

$$\begin{aligned} S &\rightarrow \lambda \quad \{ \text{Null production} \} \\ S &\rightarrow A \quad \{ \text{Unit production} \} \end{aligned}$$

Removal of  $\lambda$

$$L(G) = L(G')$$

$$L(G') = L(G) - \lambda$$

→ [those CFL of  $L(G)$  in which for every  $\lambda$ -free CFL, that generates  $(L(G'))$  which does contain  $\lambda$  or  $\lambda$ -free grammar, then it is called as set of production without  $\lambda$  containing]

BFP :- Brute force Parsing.

↳ It will only work after removing ' $\lambda$ ' & 'Unit' production.

→ Complexity of Brute force Algorithm. is

$$\boxed{O(K^n)} \Rightarrow \text{In worst case,}$$

$$\hookrightarrow |P| + |P|^2 + |P|^3 + \dots + |P|^{2n}$$

It means, it holds  $\boxed{2n \text{ rounds}}$

Thus, the no. of steps required :-

$$\boxed{\text{no. of steps} = \frac{|P| (|P|^{2n} - 1)}{|P| - 1}}$$

$$\hookrightarrow (\text{BFP is a membership algo}) \Rightarrow |P|^{2n}$$

$$\hookrightarrow \text{It comes under NP Problem,} \Rightarrow [ |P|^2 ]^n$$

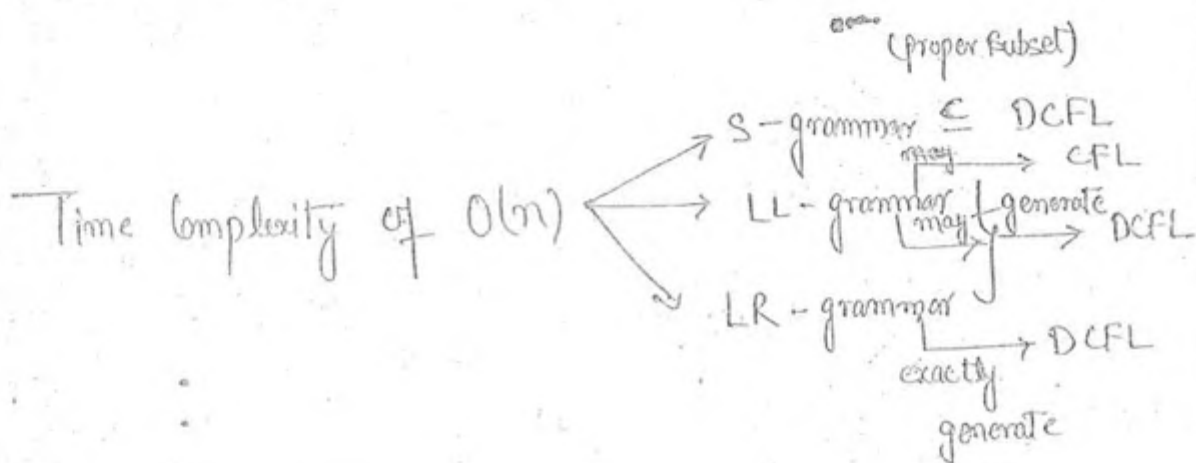
$$\boxed{\text{NP problem comprises exponential + polynomial}} \Rightarrow K^n$$

# CYK Algorithm.

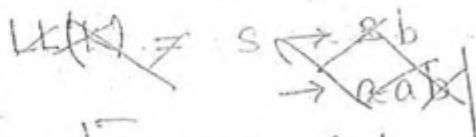
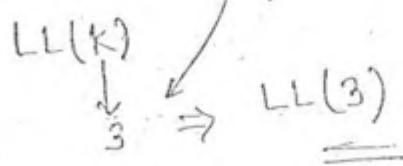
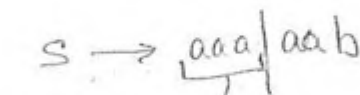
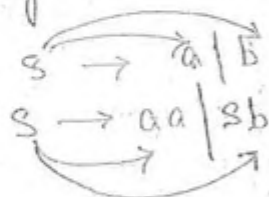
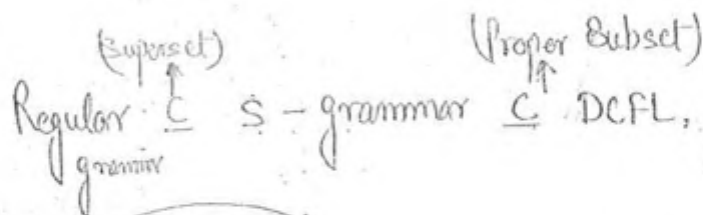
⇒ The best membership algorithm for CFL, is of  $O(n^3)$

$$\boxed{\text{Time Complexity} = O(n^3)}$$

Complexity of CYK Algo ⇒  $O(n^3)$



(S-grammar is the Super Set of Regular grammar)



$LL(k) = ?$   
 $LR(k) = ?$

⇒ Not ⇒ (Read in compiler subject)

→ If a grammar is ambiguous, then we do not have any LL(k) for any value of 'k'. However,

A grammar is unambiguous, then it is not ~~not~~ mean that grammar is  $LL(k)$  for any value of 'k'.

→ for  $LL(K)$  we just have to check by using

Predictive parse table,

→ Similarly for LR(K),

Every  $LL(k)$  and  $LR(k)$  is unambiguous for one any some value of 'k'.

for eg:-  $S \rightarrow aSb \mid ss \mid a \mid b$

Q/P.:- Neither,  $LL(k)$  nor  $LR(k)$ .

(Backup - NAUR form)

$\langle \text{if-statement} \rangle ::= \text{if } \langle \text{Statement} \rangle \text{ then } \langle s \rangle$   
 $\quad \quad \quad \text{else } \langle s \rangle$   
 $\quad \quad \quad \downarrow$   
 $\quad \quad \quad \langle \text{variable} \rangle$

(Type 2' grammar)  $I \rightarrow a S b S c S$



$$\langle \text{Term} \rangle ::= \langle \text{Term} \rangle * \langle \text{factor} \rangle$$

$$\begin{array}{l} T \rightarrow T * F \\ F \rightarrow F + T \end{array} \quad \therefore \left[ \begin{array}{l} T \rightarrow \text{Term} \\ F \rightarrow \text{factor} \end{array} \right]$$

↳ There is no grammar in "Semantics"

In Semantics  $\Rightarrow$  problem is only "Run Time error"

Properties of LL and LR grammar

1) Every "LL" and "LR" grammar is unambiguous.

2) for  $LL(K)$  and  $LR(K)$

where,  $K \rightarrow$  look ahead symbol,

There,  $LL(0)$  exist,

$LR(K) \rightarrow$  "look ahead Symbol"

$$LL(K) \rightarrow LL(K') \quad K' > K$$

$$LR(K) \rightarrow LR(K') \quad K' > K$$

But Converse is not true.

$$\text{i.e. } LL(1) \rightarrow LL(2) \Rightarrow \boxed{\text{True}}$$

$$\text{But } LL(2) \rightarrow LL(1) \Rightarrow \boxed{\text{false}}$$

3) Every DCFL is unambiguous,

i.e.  $LR(K) \xleftrightarrow{\text{gives}} DCFL$ ,

Since, every Regular language is a DCFL,

Thus, Every Regular language is unambiguous,

But,  $\boxed{\text{Every } LR(K) \xrightarrow{\text{can't generate}} \text{Regular language}}$

$\therefore \left[ \begin{array}{l} \rightarrow \text{Some Regular grammar is ambiguous, and some are} \\ \text{unambiguous.} \end{array} \right]$

4) for any ~~CFG~~ fixed value of  $K \exists$  algo. to find out if  
given CFG 'G', is  $LL(K)$  or  $LR(K)$ .

5) If a grammar 'G' is  $LR(K)$ ,  $K \geq 1$ ,  $\exists$  algo to equivalent  
 $LR(1)$ .

6)  $LR(0) \nsubseteq LR(1)$

7) for any DCFL  $\xrightarrow{\text{with prefix}} \text{DCFL } LR(0)$   
 $\xleftarrow{\text{without prefix}} LR(1)$

8)  $LR(0)$  grammar exist only if and only if for  
DCFL with prefix property.

→ language 'L' is said to be prefix property if and only if no proper prefix of  $w \in L$  must be in 'L'.

$$L = \{aab, ab\}$$

$$\{ \underbrace{a, aa, \lambda}, a, \lambda \} \rightarrow LR(0)$$

$$L = \{aab, \lambda, a\} \rightarrow \text{only } LR(1) \text{ not } LR(0).$$

$\Rightarrow$   $\lambda$  is a proper prefix of any word <sup>or string</sup>  $w$ , except ' $\lambda$ '

i)  $a^n b^n; n \geq 0 \Rightarrow LR(0) \text{ not exist,}$

ii)  $a^n b^n; n \geq 1 \Rightarrow LR(0) \text{ exist,}$

☞ If a language (L) has no prefix property, then  $L\$$  will have prefix property.

$$L\$ = \{ w\$ \mid w \in L, w \in \Sigma^*, \$ \notin \Sigma \}$$

for eg:  $L = \{aab, aa\} \Rightarrow \text{not proper prefix.}$

$$L = \{ \textcircled{a} \textcircled{a} b \$, \textcircled{a} \textcircled{a} \$ \} \Rightarrow \underline{LR(0)}$$

$\left[ \rightarrow \text{ If } L \text{ is not a } \underline{\text{DLFL}} \text{ then in order to make } \underline{\text{LR}(0)} \text{ make it } \underline{L\$}, \right]$

$\rightarrow$  A proper prefix, does not consist  $\$$  (sign) but in order to make it  $\underline{\text{LR}(0)}$  we have to add  $\$$ .

$\left[ \rightarrow \text{ for every } \underline{\text{DLFL}}, L \text{ must produce } \underline{\text{LR}(0)} \text{ or } \underline{L\$} \text{ must produce } \underline{\text{LR}(0)}. \right]$

### ALGORITHM.

$\rightarrow$  1. Removal of  $\lambda$  (Null) production;

$A \rightarrow \lambda \Rightarrow$  called null ( $\lambda$ ) production.

eg:-

$A \rightarrow BC$   
 $B \rightarrow \epsilon \lambda$   
 $C \rightarrow \lambda$

$A \xRightarrow{*} \lambda$

Nullable

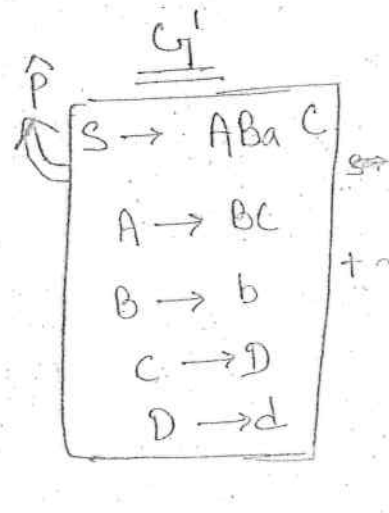
Steps:-

1)  $N =$  all Nullable Non-terminal.

2) put all nullable variable to null in every possible permutation.

for eg:-

$$\left. \begin{array}{l} S \rightarrow ABaC \\ A \rightarrow BC \\ B \rightarrow b | \lambda \\ C \rightarrow D | \lambda \\ D \rightarrow d \end{array} \right\} \Rightarrow G$$



Step 1)  $N = \{B, C\}$

$$S \rightarrow BaC | AaC | ABa | aC | AaBa | a$$

Round 2)  $N = \{B, C, A\}$

$$A \rightarrow c | B$$

Round 3)  $N = \{B, C, A\}$

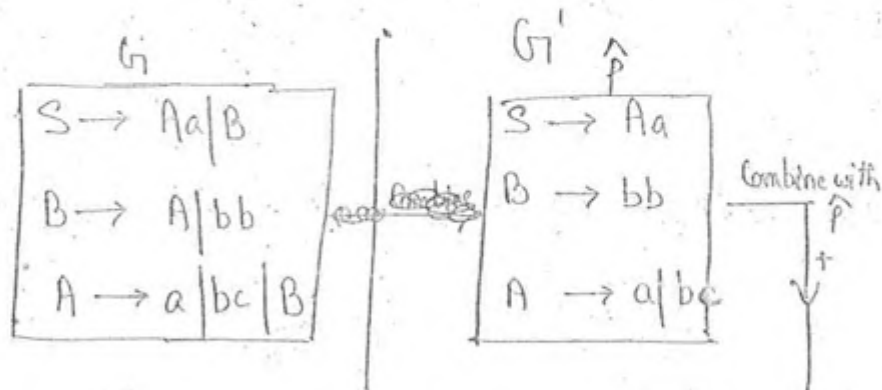
(After combining we get)

$$\left. \begin{array}{l} S \rightarrow ABaC | BaC | AaC | ABa | aC | AaBa | a \\ A \rightarrow BC | c | B \\ B \rightarrow b \\ C \rightarrow D \\ D \rightarrow d \end{array} \right\} \Rightarrow \underline{\text{final o/p.}}$$

## 2. (Removal of Unit production)

Removal of  
 ↳ Always give first preference to null production then Unit production, then Use less prod.

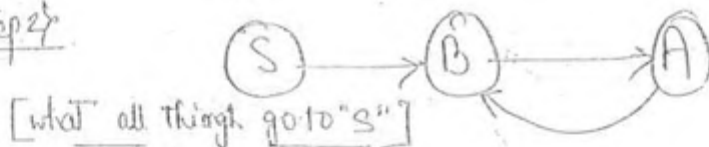
for eg:-



Step 1: make a unit production Dependency graph.

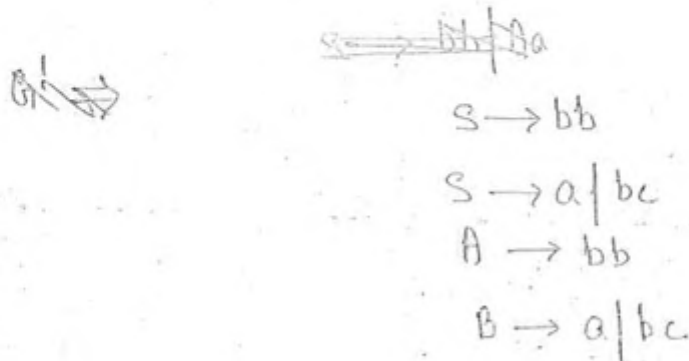
↳ make a node for every Variable,

Step 2:



$$S \xRightarrow{*} B \quad A \xRightarrow{*} B \quad B \xRightarrow{*} A$$

$$S \xRightarrow{*} A$$





Thus,

$G' \Rightarrow$

$$\begin{array}{l} S \rightarrow Aa|a|bb|bc \\ A \rightarrow a|bb|bc \\ B \rightarrow a|bb|bc \end{array}$$

$\Rightarrow$  final o/p.

### 3) Removal of Useless production

$\hookrightarrow$  A production containing useless variable either on left or right called Useless product.

$\Rightarrow$  Any problem solve by Algorithm is "decidable"

Useful Variable :-  $\hookrightarrow$  It must generate Terminal symbol.

for egr-

$$S \rightarrow aA|ab$$

$\Rightarrow$  Any Variable specified, Variable / Terminal Must be given in 'S'.

$$A \rightarrow a|bb \rightarrow \text{Useful}$$

Useless variable  $\leftarrow$

$$B \rightarrow b$$

$\Rightarrow$  (Converse of it is Useless Variable)

def'n :- (Useful Variable)  $U = \{ X \mid S \xRightarrow{*} uXv \xRightarrow{*} w$

$uv \in (TUV)^{*};$

$w \in L(G) \}$

for eg:-

$$S \rightarrow aS \mid A \mid c$$

$$A \rightarrow a$$

$$B \rightarrow aa$$

$$c \rightarrow acb$$

Step 1

We go through with <sup>all</sup> ~~Usefull~~ Variable. and delete Variable that not generate any Terminal

Step 2

Contains Terminal ( $T^*$ ) and Usefull Variable,

1)

$$U = \{A, B\}$$

$$U = \{A, B, S\}$$

$$U = \{A, B, S\}$$

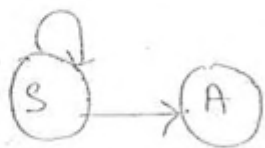
2)

$$S \rightarrow aS \mid A$$

$$A \rightarrow a$$

$$B \rightarrow aa$$

Graph



$$U = \{S, A\}$$

$$\text{Now, } S \rightarrow aS \mid A$$

$$A \rightarrow a$$

$$\text{Thus, } S \rightarrow aS \mid a$$

$$\rightarrow \boxed{aa^*} \text{ final o/p}$$

## Removal of Left Recursion.

for eg:-  $G_1 = A \rightarrow A\alpha_1 | A\alpha_2 | A\alpha_3 | \dots | A\alpha_n | \beta_1 | \beta_2 | \beta_3 | \dots | \beta_n$

shift  
A $\alpha_1$   
shift

In this step.

$$G'_1 = A' \rightarrow \alpha_1 A' | \alpha_2 A' | \alpha_3 A' | \dots | \alpha_n A' | \lambda$$

In this we not stop with  $\beta$ , we stop with by ' $\lambda$ '

$$A \rightarrow \beta_1 A' | \beta_2 A' | \beta_3 A' | \dots | \beta_n A'$$

for eg:-  $S \rightarrow \frac{S+S}{\alpha_1} | \frac{S * S}{\alpha_2} | \frac{a}{\beta_1} | \frac{b}{\beta_2} | \frac{c}{\beta_3} \} \rightarrow G_1$

$$G'_1 \Rightarrow \begin{aligned} S' &\rightarrow +SS' | *SS' | \lambda \\ S &\rightarrow aS' | bS' | cS' \end{aligned}$$

## Removal of Left factoring.

$$A \rightarrow \alpha x | \alpha y | \alpha z$$

$$\therefore \alpha, x \in (V \cup T)^*$$

$$\left\{ \begin{aligned} A &\rightarrow \alpha bC | \alpha bD | \alpha bA' \\ &\rightarrow \alpha AC | \alpha AB | \alpha AA' \end{aligned} \right\}$$

Thus, It can be written as:-

$$A \rightarrow \alpha b A''$$

$$A'' \rightarrow C | D | A'$$

## Algorithm.

CFG  $\rightarrow$  CNF / Chomsky Normal form

$\swarrow$   $\lambda$ , UNIT  
(Remove) from set of rules

eg:-

Remove the null, unit & ~~rules~~ production;

$$S \rightarrow aA | aBB$$

$$A \rightarrow aaA | \lambda$$

$$B \rightarrow bB | bbC$$

$$C \rightarrow B$$

Step 1

Removal of Null,

Here, only A is nullable,

$$S \rightarrow aA | aBB | a$$

$$A \rightarrow aaA | aa$$

$$B \rightarrow bB | bbC$$

$$C \rightarrow B$$

Step 2

Removal of Unit production,

$$C \rightarrow B \text{ / Unit production}$$

$$S \rightarrow aA | aBB | a$$

$$A \rightarrow aaA | aa$$

$$B \rightarrow bB | bBc$$

$$C \rightarrow bb | bBc$$

Step 3 Removal of Useless:-

B & C are Useless,

$$\text{Thus, } \begin{cases} S \rightarrow aA | a \\ A \rightarrow aaA | aa \end{cases}$$

$$aa(aa)^*$$

$$\text{Thus, } S \rightarrow aa(aa)^*a$$

$$\text{final o/p } [a^{2n+1}; n \geq 0]$$

CNF

Thus, CNF of this is:-

$$\text{Let, } D_a \rightarrow a$$

Thus

$$\begin{cases} S \rightarrow D_a A | D_a \\ A \rightarrow D_a D_a A | D_a D_a \end{cases} \Rightarrow \text{original grammar}$$

$$D' \rightarrow D_a A$$

Ques

$$S \rightarrow ABa$$

$$A \rightarrow aa b$$

$$B \rightarrow Ac$$

}

$\Rightarrow$

Convert it into CNF.

Soln

Let  $D_a \rightarrow a$

$$D_b \rightarrow b$$

$$D_c \rightarrow c$$

Thus,

$$S \rightarrow ABD_a$$

$$A \rightarrow D_a D_a D_b$$

$$B \rightarrow AD_c$$

$$\begin{aligned} S &\rightarrow AD' \\ D' &\rightarrow BD_a \\ A &\rightarrow D_a D'' \\ D'' &\rightarrow D_a D_b \\ B &\rightarrow AD_c \\ D_a &\rightarrow a \\ D_b &\rightarrow b \\ D_c &\rightarrow c \end{aligned}$$

 $\Rightarrow$  final o/p.

(Theorem of Chomsky Normal form)

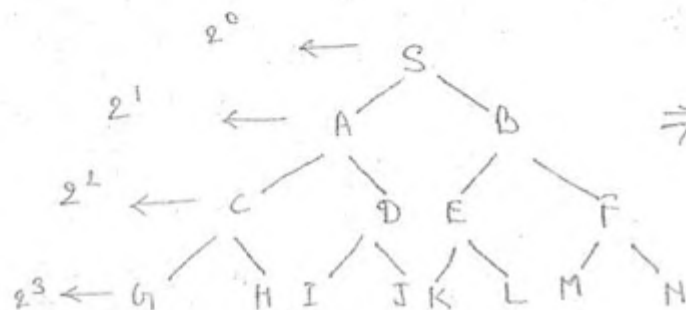
1) Derivation of any string of length  $n$ , when  $G$  is CNF has  $(2n-1)$  steps



2. Minimum height of derivation tree for a string of length 'n',  
 i.e.  $|w|=n$ , G is CHF is  $\lceil \log_2 n \rceil + 1$ .

3. Maximum height of derivation tree for a string of length 'n',  
 i.e.  $|w|=n$ , G is CHF is "n".

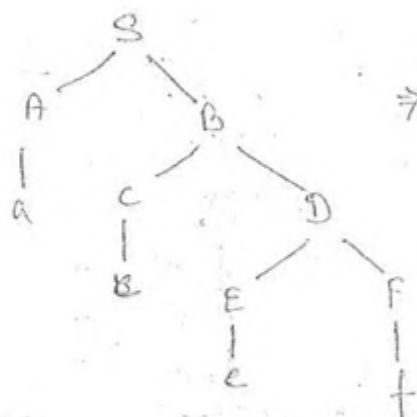
$\Rightarrow$  for min. height  $\Rightarrow$



$\Rightarrow$  Most balanced  
 gives min. height.

$$2^h \rightarrow n \quad [h \rightarrow \text{height}]$$

Thus,  $(h = \log_2 n)$  Thus, min. height is  $\lceil \log_2 n \rceil + 1$



$\Rightarrow$  Most Unbalanced  
 Tree,  
 gives max. height,

$(n-1) \rightarrow \text{height}$   
 when, terminal increases

$$(n-1) + 1 \Rightarrow n$$

Thus, max. height is  $(n)$ .

### NOTE:-

- S-Grammar is always in Greibach normal form, while GNF not GNF.
- Every S-Grammar is LL(1) grammar, but converse is not true.

(Conversion of CFG to GNF)

GNF form :-

$$\begin{array}{l} N \rightarrow TV^* \\ \rightarrow T \end{array}$$

eg:-

$$S \rightarrow asb \mid bsa \mid \lambda$$

Thus,  $S \rightarrow asb \mid bsa \mid ab \mid ba$ .

Note:- If the front part of set of production is terminal  
We can convert it into variable.

for eg:-  $S \rightarrow D_a S D_b \mid D_b S D_a \mid D_a D_b \mid D_b D_a$

for eg:-  $S \rightarrow A B a \mid A a$   
 $A \rightarrow a b \mid b B C \mid b$  substituting "A"

then  $S \rightarrow a B B a \mid b B C B a \mid b B a \mid a B a \mid b B C a \mid b a$

$\begin{bmatrix} D_a \rightarrow a \\ D_b \rightarrow b \end{bmatrix}$   
 $\rightarrow a B B D_a \mid b B C B D_a \mid b B D_a \mid a B D_a \mid b B C D_a \mid b D_a$

eg 2

$$S \rightarrow Aa | Bb$$

$$A \rightarrow aAb | aB$$

$$B \rightarrow b$$

$$S \rightarrow aAba | aBb | bBb | aBAa$$

$$A \rightarrow aAa | aB$$

$$B \rightarrow b ; D_b \rightarrow b ; D_a \rightarrow A$$

$$\text{Thus, } S \rightarrow aAD_bDa | aBD_a | bBb | aBA \underline{Da}$$

No. of steps in derivation of  $|w| = n$  length of string in GNF  
is " $n$ "

Complexity of GNF is  $O(n)$

# Machine

## (Push Down Automata)

PDA  $\rightarrow$  NPDA

$\rightarrow$  (Every DPDA is NPDA but converse is not true.)

$\rightarrow$  general method to represent PDA.

$\{Q, \Sigma, \Gamma, \delta, q_0, Z, F\}$

$\rightarrow$  Start Symbol,

$Z \in \Gamma$  and  $Z$  can not push into stack.

$\Sigma = \{a, b\}$

$\Gamma = \{Z, a\}$

[Instead 'a' there may be '0'  
, " of 'b' " " " " " ]

$\rightarrow$  Dead Configuration is allowed in both DPDA and NPDA.

Current State  $q$   $\rightarrow$  Current IP  $\rightarrow$  Current Stack.  
 $\delta(q, a, b)$

$\rightarrow$  Non deterministic is always "Search & back track"  
(NPDA, NFA)  
for egr chess game,

⇒ four operation of stack :-

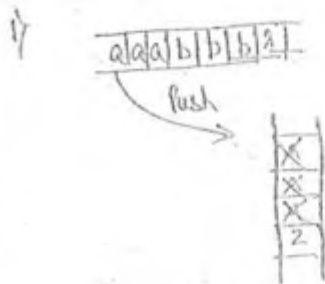
- 1) push  $\rightarrow \delta(q_0, a, b) \Rightarrow q_1, (q_1, ab)$
- 2) pop  $\rightarrow \delta(q_0, a, b) = (q_1, \lambda)$
- 3) replace  $\rightarrow \delta(q_0, a, b) = (q_1, c)$
- 4) do nothing  $\rightarrow \delta(q_0, a, b) = (q_1, b)$

PDA program.

$a^n b^n, n \geq 0$

$n_a(w) = n_b(w)$

work



Here a will pop the b,  
& b will pop the a,

$$\delta(q_0, a, z) = (q_0, az)$$

$$\delta(q_0, b, z) = (q_0, bz)$$

$$\delta(q_0, a, a) = (q_0, aa)$$

$$\delta(q_0, b, b) = (q_0, bb)$$

$$\delta(q_0, a, b) = (q_0, ab)$$

$$\delta(q_0, b, a) = (q_0, ba)$$

$$\delta(q_0, \lambda, a) = (q_1, a)$$

$$\delta(q_0, \lambda, b) = (q_1, b)$$

$$\delta(q_0, a, z) = (q_1, az)$$

$$\delta(q_1, a, a) = (q_1, aa)$$

$$\delta(q_1, b, a) = (q_2, \lambda)$$

$$\delta(q_2, b, a) = (q_2, \lambda)$$

$$\delta(q_2, \lambda, z) = (q_0, z)$$

$$\delta(q_0, a, z) = (q_1, az)$$

$$\delta(q_0, b, z) = (q_1, bz)$$

$$\delta(q_1, a, a) = (q_1, aa)$$

$$\delta(q_1, a, b) = (q_1, \lambda)$$

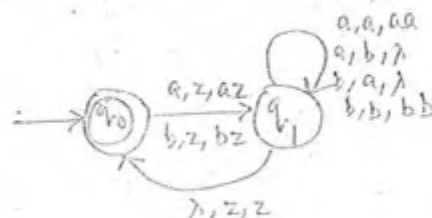
$$\delta(q_1, b, a) = (q_1, \lambda)$$

$$\delta(q_1, b, b) = (q_1, bb)$$

$a^n b^{2n}, n \geq 0$

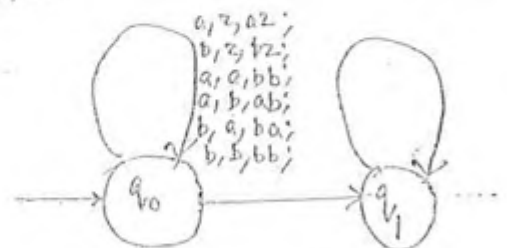
$$\delta(q_0, a, z) = (q_1, aaz)$$

$$\delta(q_1, a, a) = (q_1, aaaa)$$

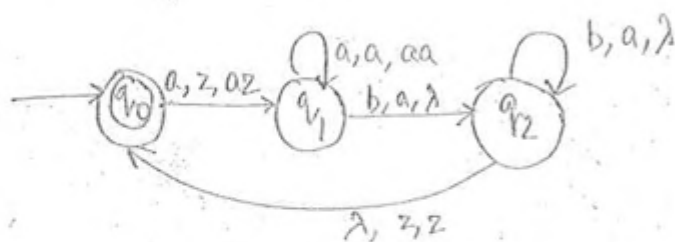


$$\delta(q_1, b, b) = (q_1, \lambda)$$

$$\delta(q_1, a, a) = (q_1, \lambda)$$



eg ii) state diagram of  $a^n b^n; n \geq 0$



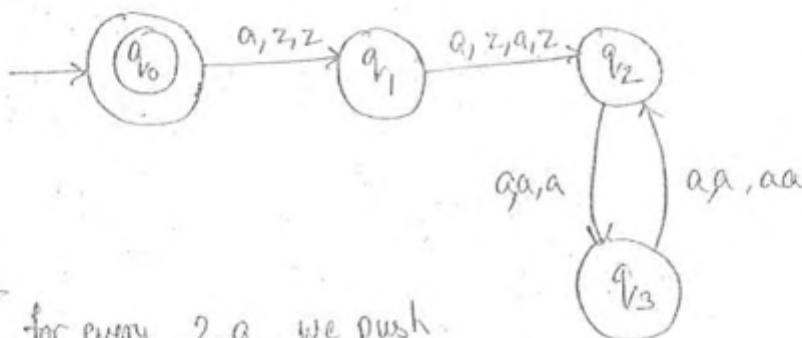
~~Not a DFA~~

$L(L) \rightarrow$

$L \neq \bar{L}$   $L \rightarrow$  Regular

$L \neq L^c$  is (PE)  
 $L = \emptyset$

eg iii)  $L = \{a^{2n}, b^n; n \geq 0\}$



[for every 2 'a' we push  
1 'b']

eg iv)  $L = \{a^n b^{n+1}; n \geq 0\}$

eg v)  $L = \{a^n b^n c; n \geq 0\}$

eg vi)  $L = a^m b^n; m \geq n$  } See by own

$m \leq n$

$m \neq n$

$m > n$

$m < n$



$\Rightarrow \{ \text{CFL} \cap \text{CFL} \} \Rightarrow \text{Not } \underline{\underline{\text{CFL}}}$ .

for e.g.  $\overset{(\infty)}{\uparrow} \boxed{a^n b^m c^m} \cap \boxed{a^m b^n} c^m = \boxed{a^n b^n c^n} \rightarrow \underline{\underline{\text{CSL}}}$

CFL                      CFL

$\Rightarrow \boxed{A \cap B = (A^c \cup B^c)^c} \Rightarrow \text{De Morgan's Law}$

|                                 | CFL |
|---------------------------------|-----|
| $\rightarrow$ Membership Algo.  | ✓   |
| $\rightarrow$ finite / Infinite | ✓   |
| $\rightarrow$ Emptiness         | ✓   |

If  $L$  is CFL  $\rightarrow$  'satisfied' Pumping lemma.  
while Converse is not True

$\Rightarrow$  ~~if~~  $S$  is useful, the language is not empty.

$\Rightarrow$  ~~if~~  $S$  is useless, the language is empty.

$\hookrightarrow$  equivalence of Pda  $\rightarrow$  undecidable  
 " of dfa  $\rightarrow$  decidable  
 " of nfa  $\rightarrow$  decidable  
 " of ambiguity  $\rightarrow$  Undecidable.

$L = \Sigma^*$   $\rightarrow$  Undecidable,  
 $L_1 \cap L_2 = \emptyset$   $\rightarrow$  Undecidable.

| True                                                                                                         | false                                                                                                   |
|--------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------|
| $\{ \text{Direct} \} \quad p \rightarrow q$<br>$\{ \text{Contrapositive} \} \quad \sim q \rightarrow \sim p$ | $q \rightarrow p \quad \{ \text{Converse} \}$<br>$\sim p \rightarrow \sim q \quad \{ \text{Inverse} \}$ |

(ww  $\rightarrow$  Linear  $\rightarrow$  Surely Satisfy the Pumping lemma for Linear)

$S \rightarrow \lambda$  (only in Context Sensitive language)  $\rightarrow$  S does not appear on RHS

$A \rightarrow \lambda$  (never in Type I language) (may or may not)

$\Rightarrow$  Two different grammar may generate same languages.

$G' \Rightarrow S \rightarrow SS | aS | bS | a | b$

$G \Rightarrow S \rightarrow aS | bS | a | b$

$L(G') = L(G)$



$\Rightarrow L(G)$  is define up,  $w \in \Sigma^*$  |  $S \xrightarrow[G]{*} w$

$\Rightarrow S \xrightarrow[G]{*} \alpha$   $\alpha$  is called as Sentential form,

Def  $(a^m b^n ; m \leq n \leq 3n)$

Grammar  $\Rightarrow S \rightarrow aSb \mid aSbb \mid aSbbb \mid \lambda$

$\delta(q_0, a, z) = (q_1, az) \quad (q_1, aaz) \quad (q_1, aaaaz)$

$\delta(q_1, a, a) = (q_1, aa) \quad (q_1, aaa) \quad (q_1, aaaa)$

(Conversion of CFG to Machine)

for eg:-



eg:-

$S \rightarrow aA$

$A \rightarrow aABC \mid bB \mid a$

$B \rightarrow b$

$C \rightarrow c$

$\Rightarrow$  The above grammar is already in the form of Greibach Normal form,  $\left[ \begin{array}{l} V \rightarrow TV^* \\ \rightarrow T^* \end{array} \right]$

Here, Variable portion kept in the stack  
and Terminal kept in the Tape.

$S \Rightarrow aA \Rightarrow abB \Rightarrow \underline{\underline{abb}}$

Tape

Stack

$$\delta(q_0, \lambda, z) = \{(q_1, s_z)\}$$

$$\delta(q_1, \lambda, z) = \{(q_1, z)\}$$

$$\delta(q_1, a, s) = \{(q_1, A)\}$$

$$\delta(q_1, b, A) = \{(q_1, ABC), (q_1, \lambda)\} \Rightarrow \text{represent NPDA}$$

$$\delta(q_1, b, A) = \{(q_1, B)\}$$

$$\delta(q_1, b, B) = \{(q_1, \lambda)\}$$

$$\delta(q_1, c, C) = \{(q_1, \lambda)\}$$

→ If the grammar has  $n$  production maximum commands are  $(n+2)$ .

↳ Any CFG, accepted by machine (NDPA | DPDA) is of at most 3 states.

*Signature*

(Question to be ask):-

⇒ a Type '0' grammar / RE grammar can generate Regular grammar,

i)  $\hookrightarrow$  True / false,      ii) Pumping lemma  $\Rightarrow$  Are, ?

→ every CFG generate CSL (that are under CFL)

→ every regular grammar is a linear grammar,  $\{(\text{subset}) | (\text{total})\}$

then, RE grammar or Type '0' grammar can also generate Regular grammar  
(True / false)

→ length of string is  $|\Sigma^n|$  and the no. of string of length  $\Sigma^n$   
is  $|\Sigma|^n$  True / false

→  $a^n b^n c^n$

set of productions:-

$S \rightarrow aSBC | aBC$

$CB \rightarrow BC,$

$OB \rightarrow ab,$

$BB \rightarrow bb,$

$bC \rightarrow bc,$

$cC \rightarrow cc$

$\Rightarrow$  equivalent  
to CSL  
but not CSL  
due to  $CB \rightarrow BC$

True / False





Date: 16/07/10

↳ atleast 1 a's and 2 b's.

$$\left\{ \begin{aligned} &(a+b)^* a (a+b)^* \star b(a+b)^* \star b(a+b)^* \\ &+ \\ &(a+b)^* b(a+b)^* \star a(a+b)^* \star b(a+b)^* \\ &+ \\ &(a+b)^* b(a+b)^* \star b(a+b)^* \star a(a+b)^* \end{aligned} \right\}$$

~~B~~ →

B → C  
B → -

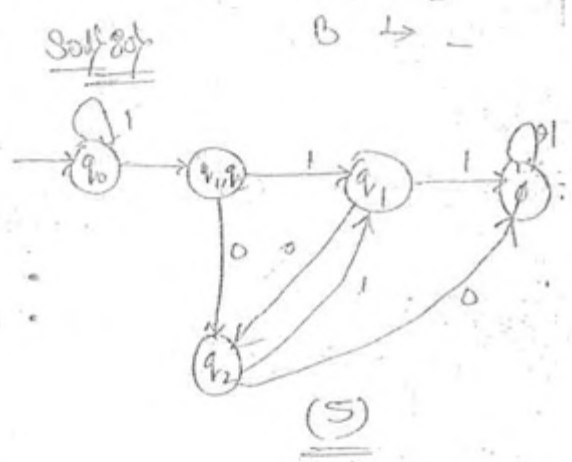
C → -  
B → C  
C → B  
B → -

$(00)^* 0 \rightarrow$  odd zeros

$00^* 0 \rightarrow$  even zeros

$$(00)^* 0 \neq 00^* 0$$

$$(00)^* 0 = \underline{\underline{0(00)^*}}$$

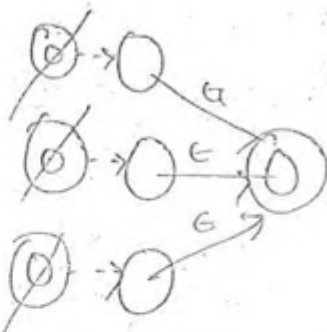


$$\Rightarrow 0^* 1^* \Rightarrow 0^* + 1^* + 0^* 1^*$$

$$\Rightarrow 0^* 1^* 2^* \Rightarrow 0^* + 1^* + 2^* + 0^* 1^* + 0^* 2^* + 1^* 2^* + 0^* 1^* 2^*$$

Que 85

Q



change final to non final state

WCLW

But in dfa it is not possible, because, in dfa, null move is not possible.

Que 86

$$L = (a+ba)^* (e+b)$$

$$h(L) = (h(a)+h(b) \cdot h(b))^* (e+h(b))$$

eg:-

$$L = \{01\}$$

$$h(0) = 00$$

$$h(1) = 11$$

$$h^{-1}(L) = \emptyset$$

$$h(h^{-1}(L)) = h(\emptyset) = \emptyset$$

Run cry csl

ambn

aca

$$\left. \begin{array}{l} S \rightarrow aA/aB \\ A \rightarrow Ab/\lambda \end{array} \right\}$$

$$\Rightarrow \left. \begin{array}{l} S \rightarrow aA \\ \rightarrow aAb \\ \rightarrow aaAbb \\ \rightarrow aaaAbbb \\ \rightarrow a^n b^n \end{array} \right\}$$

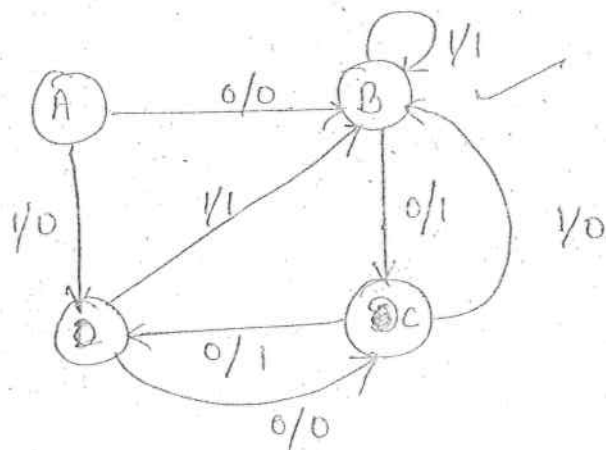
$$\left. \begin{array}{l} S \rightarrow aCa \\ C \rightarrow aCa/b \end{array} \right\}$$

$$\left. \begin{array}{l} S \rightarrow aCa \\ \rightarrow aaCa \\ \rightarrow aaaCa \\ \rightarrow a^n b a^n \end{array} \right\}$$

$$U, L^c \Rightarrow \oplus$$

$$\begin{aligned} A \oplus B &\Rightarrow (A-B) \cup (B-A) \\ &\Rightarrow (A \cap B^c) \cup (B \cap A^c) \end{aligned}$$

Soln 1324



Soln

a) 01

b) 10

c) 101

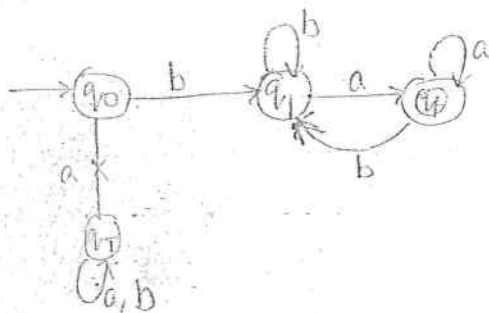
d) 110

$\Rightarrow$  last bit condition also asked?

(1)

Ans  $\Rightarrow$  Not Neither starting with 'a' nor ending with 'b' }  
opposite

Soln  $\Rightarrow$  either starting with 'b' or ending with 'a'.



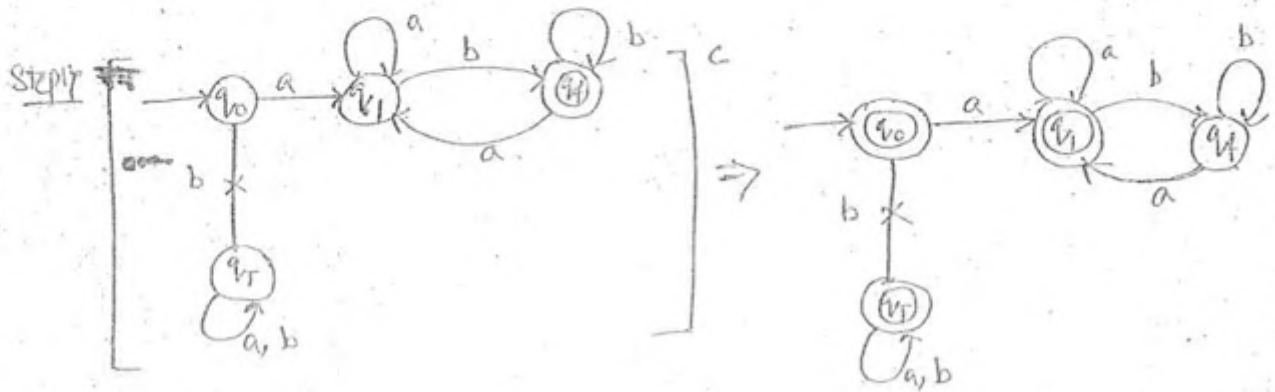
20/09/10

## Context free grammar

Ques Not starting with 'a' or not ending with 'b'.

↳ [Not starting with 'a' or not ending with 'b']<sup>c</sup>

→ [starting with 'a' and ending with 'b']



r.e.  $\Rightarrow$

$$\lambda + a(a+b)^* + (a+b)^*b$$

or

r.e.  $\Rightarrow$

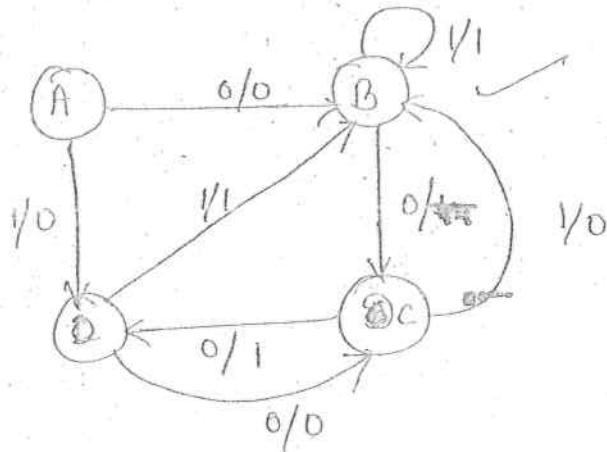
$$\lambda + a(a+bb^*a)^* + b(a+b)^*$$

$$U, L^c \Rightarrow \oplus$$

$$A \oplus B \Rightarrow (A-B) \cup (B-A)$$

$$\Rightarrow (A \cap B^c) \cup (B \cap A^c)$$

Soln 1324



Soln

a) 01

b) 10

c) 101

d) 110

$\Rightarrow$  But bit Condition also asked?

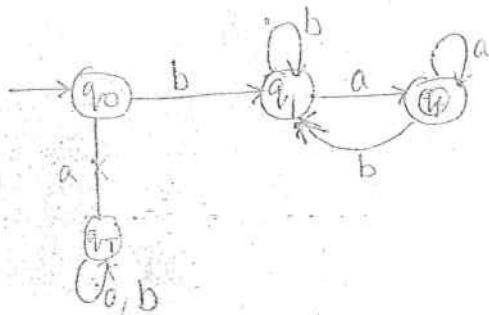
(1)

Ans  $\Rightarrow$

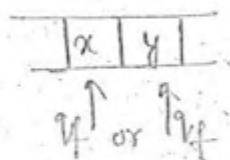
Not  
Neither

Starting with 'a' ~~not~~ ending with 'b' }  
 $\downarrow$  opposite

Soln  $\Rightarrow$  either starting with 'b' or ending with 'a'.



$\hookrightarrow x q_f y \rightarrow$  Tells about the final state whether on left or right.

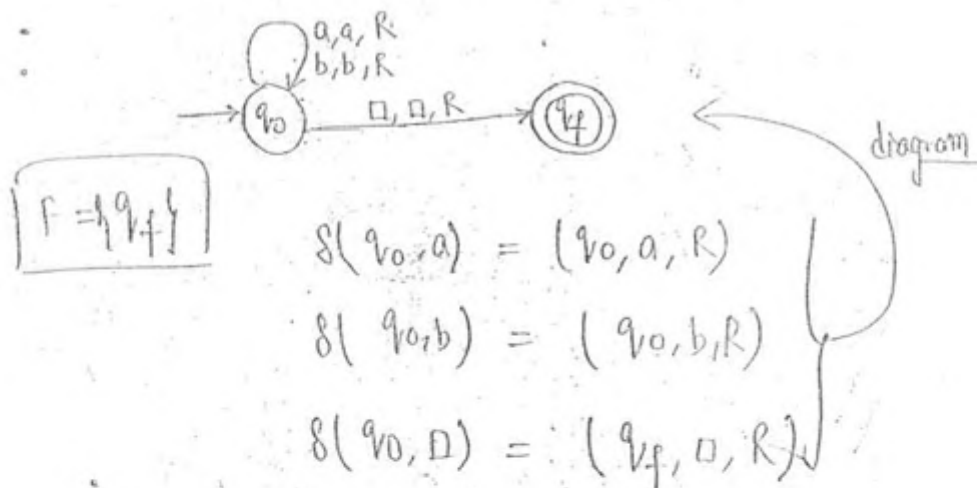


atb

$\hookrightarrow x, y \in \Gamma^+$   $\rightarrow$  It shows Input Tape.

Ques Converting a machine into language.

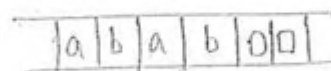
$$M \rightarrow L(M)$$



Ques What language accepting by TM.

Sol

$$(a+b)^+$$



Ques

What is set of string (w) it will halt,

Sol

$$(a+b)^+$$



Ques 3  $w \rightarrow \text{lang.}$

Sol  $\phi$  (empty)

Ques 4 Contents of top of set,

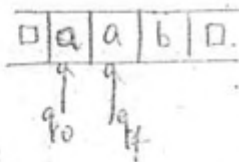
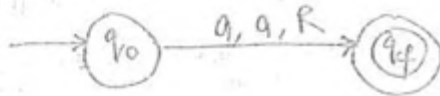
Sol Unchanged (By default always)

If it goes 'L', it will hang

$\Rightarrow$

$\left. \begin{array}{l} a a R \\ b a R \end{array} \right\} \text{Contents of top is } a^+$

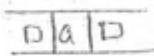
eg ii



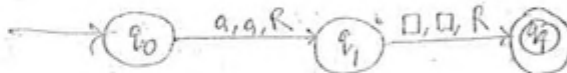
$o/p \Rightarrow a(a+b)^*$

iii

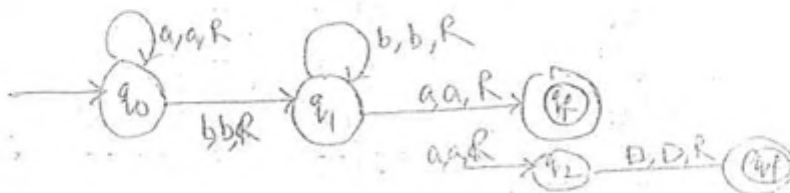
$\nmid$



$o/p \Rightarrow (a) \text{ or (exactly) or (only } a)$



iv

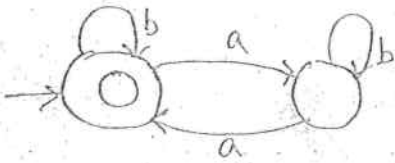


$\Rightarrow a^* b b^* (a+b)^*$

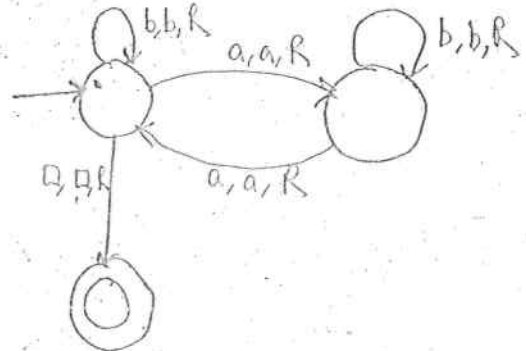
$\Rightarrow a^* b b^* a$

Even a's

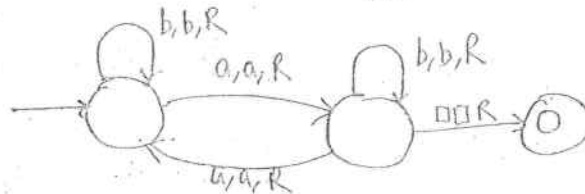
DFA



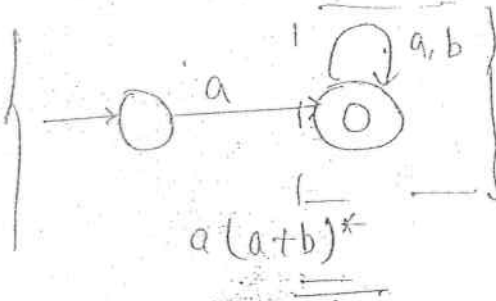
TM



or



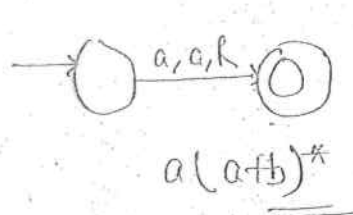
DFA

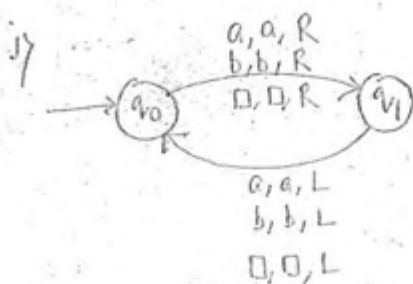


Turning M/C  
does not  
allow it,

Rewriting  
this,

TM





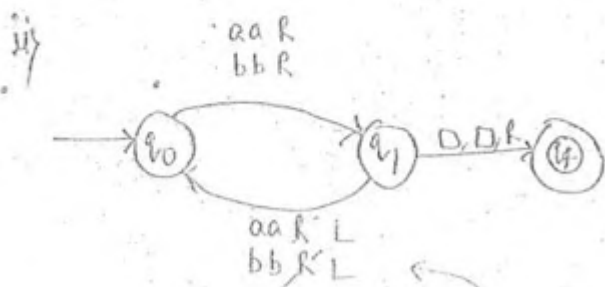
$$\Sigma = \{a, b\}$$

$$\Gamma = \{a, b, \square\}$$

$$L(M) = \emptyset$$

$$\text{halt} = \emptyset$$

$$\text{hang} = (a+b)^+$$



$$\text{o/p} \Rightarrow w \mid \text{length of } w \text{ is odd}$$

$\Rightarrow$  If condition changed

then output

$\square a \square$  ✓

$\square b \square$  ✓

$\square a \square$  Hang

$\square ab \square$  Hang

$\square ba \square$  Hang

$\square bbb \square$  Hang

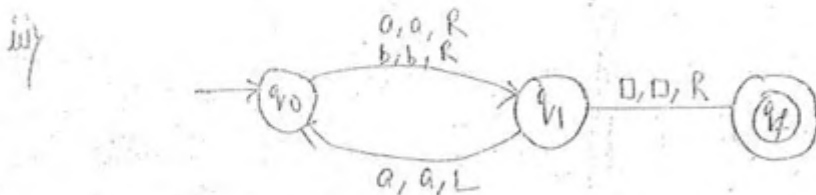
$$L(M) = \{a, b\}$$

$$\text{halt} = \{a, b\}$$

$$\text{hang} = \{(a+b)^+\} - \{a, b\} \text{ or } \downarrow$$

$$w \mid \begin{matrix} \text{length of} \\ w \end{matrix} \geq 2$$

Content of Tape  $\Rightarrow$  Unchanged.



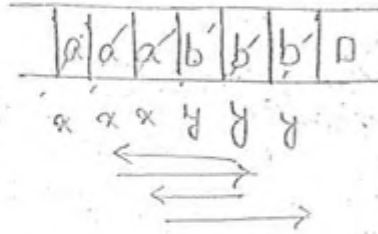
$$\text{o/p} \quad L(M) = \{a, b\}$$

$$\text{halt} \Rightarrow (a+b) \mid (a+b)^+$$

$$\text{hang} \Rightarrow (a+b)a(a+b)^+ \mid \text{Anything starting with } a$$

### 3. Turning Machine as Accepter

$$\{ \underline{a^n b^n} ; \underline{n \geq 1} \}$$



$$\delta(q_0, a) = (q_1, x, R)$$

$$\delta(q_1, a) = (q_1, a, R)$$

$$\delta(q_1, y) = (q_1, y, R)$$

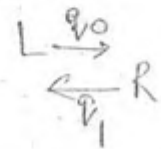
$$\delta(q_1, b) = (q_2, y, L)$$

$$\delta(q_2, y) = (q_2, y, L)$$

$$\delta(q_2, a) = (q_2, a, L)$$

$$\delta(q_2, x) = (q_0, x, R)$$

(When L and Right move  
stat must change)

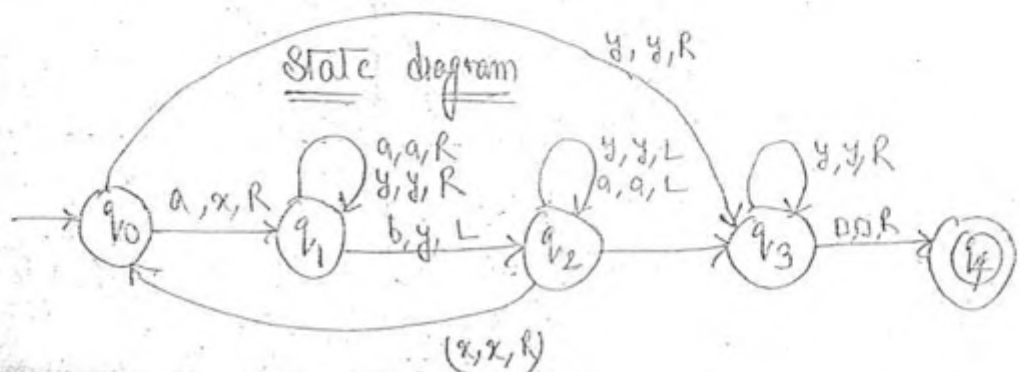


(It is only possible when, a is finished)

$$\delta(q_0, y) = (q_3, y, R)$$

$$\delta(q_3, y) = (q_3, y, R)$$

$$\delta(q_3, \square) = (q_4, \square, R)$$



$$\therefore q_0 = q_0 1 + q_0 0 1 + \Lambda$$

$$q_0 \Rightarrow q_0 (1 + 01) + \Lambda$$

$$\Rightarrow R = R P + Q$$

$$R = Q P^*$$

$$\therefore q_0 = \Lambda (1 + 01)^*$$

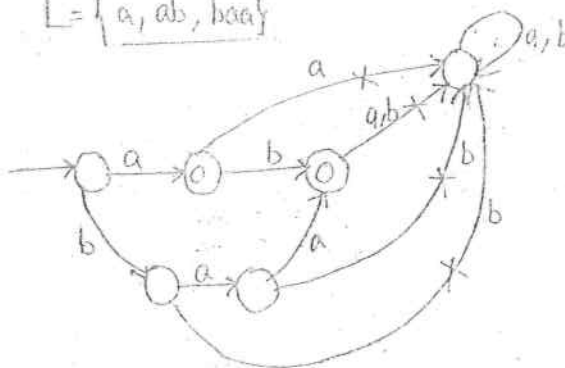
$$\hookrightarrow q_1 = q_1 00^* 1 (0+1)^*$$

$$\Rightarrow q_0 000^* 1 (0+1)^*$$

$$q_1 = \boxed{(1+01)^* 000^* 1 (0+1)^*}$$

Ques  $L = \{a, ab, baab\}$

Sol

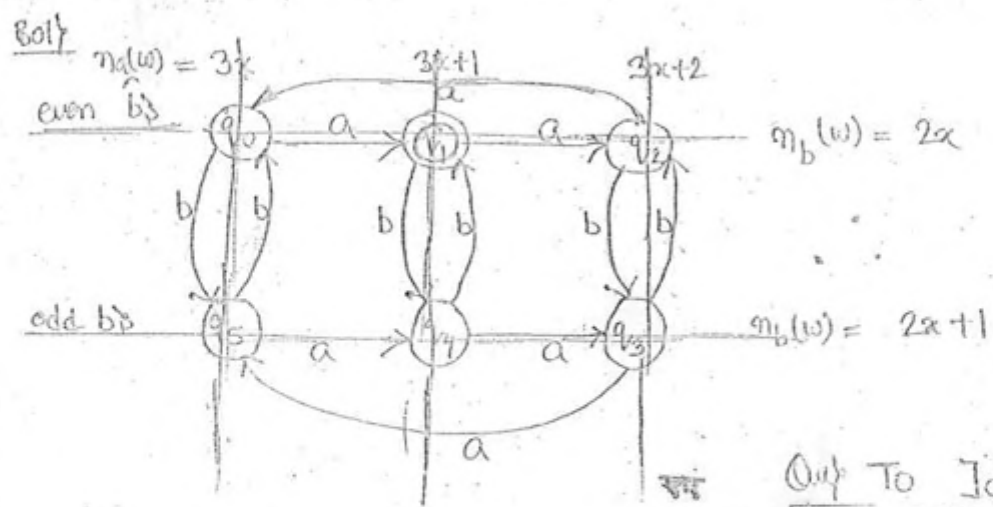


DFA Design

Ans

Exmp-9

$(n_a(w) \bmod 3 = 1 \quad \& \quad n_b(w) \bmod 2 = 0)$



then

Ques To Identify final state?

T.O.C - AUTOMATA

106