

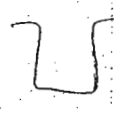
15/6/11

Data structure

Data structure

- A logical or mathematical model

Eg: stack
(or)
LIFO
(or)
FILO



physical structure (4)

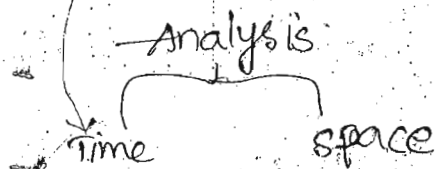
Implementation of DS (model) in physical memory.

Eg: Arraystack, linked list stack



- why data structure is required?

Solving a problem fastly by occupying the optimum memory



1D-Arrays:

p) A is an array represented as follows

A: array [5---5] of elements. Each element occupies two memory cells. The starting location is 1999. Calculate the location of element A[0]

sol: A: array [lb---ub] of elements

Lo: starting location = 1999

C: elements size = count = 2 Books

lb: lower boundary = -5

$$\text{loc } A[i] = L_0 + (i - lb) * C$$

$$\text{loc } A[0] = 1999 + (0 - (-5)) * 2$$

$$= 1999 + 10$$

$$= 2009$$

I level:

{ sahani
weiss
kruse

II Level:

cozeman
Goodrich &
Tamson

{ (Interviews) Drozdek

[]	*
Array	Pointer
static	dynamic
early binding	late binding
compile time	runtime
user friendly	machine friendly
$a[i]$	$*x(i)$

constant pointer pointer variable

→ when calling by name control is inside
 & " outside
 * is used to get content

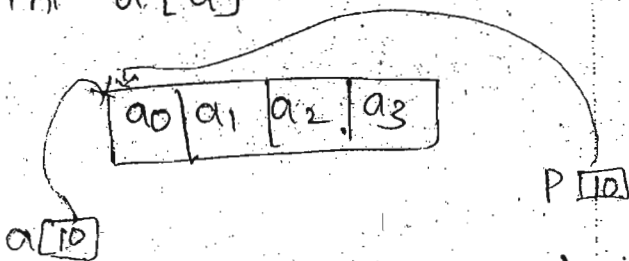
→ $loc\ A[i] = loc + (i - lb) * C$
 $\&A[0] + (i - 0) \rightarrow$ scalar arithmetic

$$loc\ A[i] = A + i$$

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

Example:

int a[4]



$a = \&a[0]$

$10 = 10$

~~a~~ $a+1 \leftarrow$ valid

$\leftarrow a + 10 + 1 = 12$
 invalid constant pointer

int *P

$P = a$

$P+1 \leftarrow$ valid

$10 + 1 = 12$

$P = P+1 \rightarrow$ valid

$P = 12$

An C
 char - 1 byte
 int - 2 "
 float - 4

char $10 + 1 = 11$
 int $10 + 1 = 12$
 float $10 + 1 = 14$

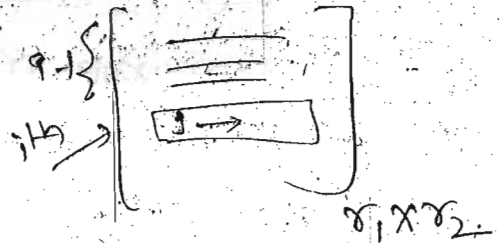
2D Array:

2
p) A: array $[-2 \dots 2, 3 \dots 7]$ of elements. The starting location is 1000. Each element occupies 2 memory cells. Calculate the location of element $A[0,5]$ using i) RMO ii) CMO

RMO

In simple

$$\text{loc } A(i,j) = \text{Lo} + (i-1)r_2 + j-1$$



In detail

$$\text{loc } A(i,j) = \text{Lo} + [(i-b_1)(u_2-b_2+1) + (j-b_2)] \times C$$

A: array $[b_1 \dots u_1, b_2 \dots u_2]$ of elements
row r_1 $\times$ col r_2
 $(u_1-b_1+1) \quad (u_2-b_2+1)$

$$\begin{aligned} \text{loc } A[0,5] &= 1000 + [(0-(-2)) \times 5 + (5-3)] \times 2 \\ &= 1000 + 24 \\ &= 1024 \end{aligned}$$

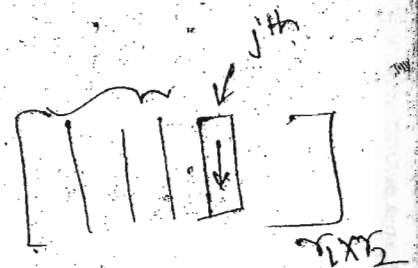
16/6/11

CMO: In simple

$$\text{loc } A(i,j) = \text{Lo} + (j-1)r_1 + (i-1)$$

In detail

$$\text{loc } A(i,j) = \text{Lo} + [(j-b_2)(u_1-b_1+1) - 2D + (i-b_1)] \times C = 10$$



$$\begin{aligned} \text{Loc } A(0,5) &= 1000 + [(5-3) \times 5 \\ &\quad + 0 - (-2)] \times 2 \\ &= 1000 + 24 = 1024 \end{aligned}$$

Lower triangular matrix:

Def: - Is a square matrix
 $- A(i,j) = 0$ iff $i < j$

Size: $1+2+3+\dots+N$

$$\sum_{i=1}^N i = \frac{N(N+1)}{2}$$

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:

RMO:

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

R: $L_0 +$ the no. of elements present in $(i-1)$ rows
 $+ j-1$

$$= L_0 + (1+2+3+\dots+j-1) + (j-1)$$

$$= L_0 + \frac{j(j-1)}{2} + (j-1)$$

CMO:

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

$= L_0 +$ the no. of elements present in $(j-1)$ columns
 $+ (j-1)$

$$= L_0 + [n + (n-1) + (n-2) + \dots + n - (j-2)] + (j-1)$$

$$= L_0 + \frac{(p-1)}{2} [2n + (p-2)(-1)] + (p-1)$$

3

check: $a_{(4,3)}$

$$= 0 + \frac{(3-1)}{2} [2(4) + (3-2)(-1)] + (4-3)$$

$$= 0 + 7 + 1$$

$$= 8$$

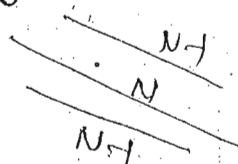
Tri-diagonal matrix:

Def: - Is a square matrix $p \times p$

$$- A(p, q) = 0 \text{ iff } |p - q| > 1$$

~~- Symmetric nature~~

Size:



$$= N + N + N$$

$$= 3N$$

store:

1) RMO:

0	1	2	3	4	5	6	7	8	9
a_{11}	a_{12}	a_{21}	a_{22}	a_{23}	a_{32}	a_{33}	a_{34}	a_{43}	a_{44}

R: $L_0 + \text{no. of elements crossed in } (p-1) \text{ rows}$

$$+ (p - p's lb)$$

$$= L_0 + [3(p-1) - 1]$$

$$+ p - (p-1)$$

$$= L_0 + 0 + p - 1$$

$$p = 1$$

$p's lb$	$p's lb$
4	3
3	2
2	1

Except 1 row

$p \quad p-1$

check:

$$A(3,4) = 0 + 3(3-1) - 1 \\ + 4 - (3-1) \\ = 7$$

CMD:

a_{11}	a_{21}	a_{12}	a_{22}	a_{32}	a_{23}	a_{33}	a_{43}	a_{34}	a_{44}
----------	----------	----------	----------	----------	----------	----------	----------	----------	----------

R: $L_0 + \text{no. of elements present in } (j-1) \text{ columns}$

$$+ j - (j-1)$$

$$= L_0 + 3 \times (j-1) - 1$$

$j \neq 1$

$$+ j - (j-1)$$

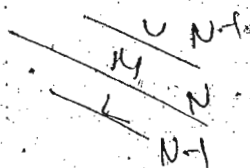
$$= L_0 + 0$$

$$+ j - 1$$

$j = 1$

diagonal by diagonal:

$\{L, M, U\}$



Lower	Main	Upper
a_{21} a_{32} a_{43}	a_{11} a_{22} a_{33} a_{44}	a_{12} a_{23} a_{34}

Condition: $j - j = +1$

$j - j = 0$

$j - j = -1$

$$L_0 + 0$$

$$+ j - 2$$

$$L_0 + 0$$

$$+ j - 1$$

$$L_0 + (N-1)$$

$$+ j - 1$$

$$L_0 + (N-1)$$

$$+ j - 1$$

$$L_0 + N - 1 + N$$

$$+ j - 1$$

$$L_0 + N - 1 + N$$

$$+ j - 2$$

switch (f-j)

4

{

case 1: // Low diagonal

return A [f-2];

(or) A [j-1];

case 0: // Main diagonal

return A [N+f-2];

(or) A [N+j-2];

case -1: // upper diagonal

return A [2n+f-2];

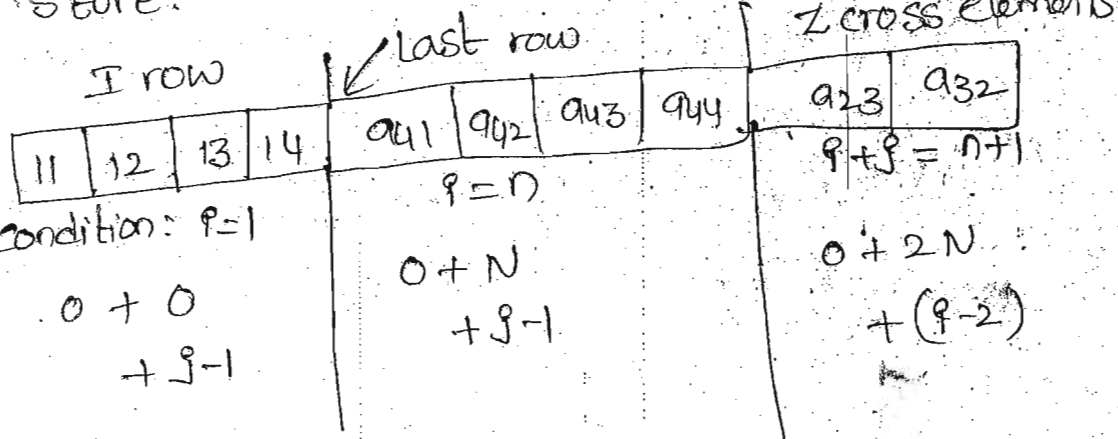
(or) A [2n+j-3];

Default: return 0;

}

Z-Matrix;

Store:



if ($i == 1$)

return $A[i]$;

else if ($i == N$)

return $A[N+1]$;

else if ($i+j == N+1$)

return $A[2N+i-2]$;

else

return 0;

Q) what is the size of square matrix in which the elements are present at $i-j > n+1$

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

4x4

not possible
 $i-j > n+1$

Zero size

if $i+j > n+1$

0	0	0	14	$\frac{i+j}{5} \times$
0	0	23	0	$\rightarrow 6 \checkmark$
0	32	0	0	$\rightarrow 7 \checkmark$
41	0	0	0	$\rightarrow 8 \checkmark$

~~$N-1 + N-2 + \dots + 1$~~

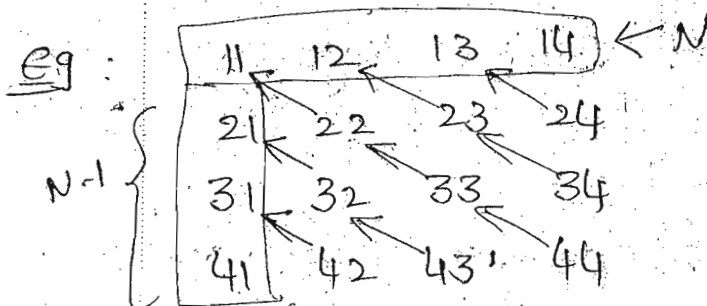
$$= \frac{N(N+1)}{2}$$

Toeplitz matrix

5

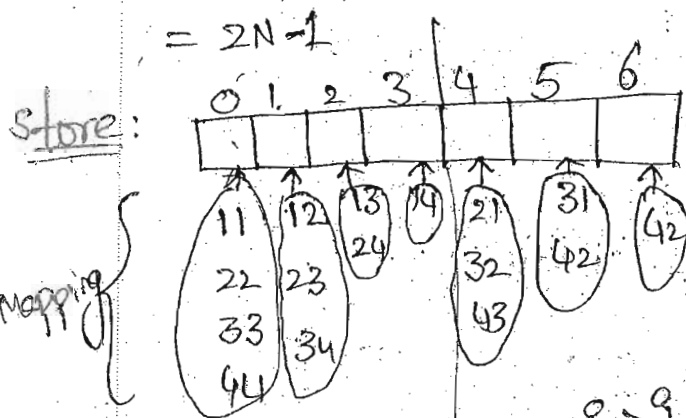
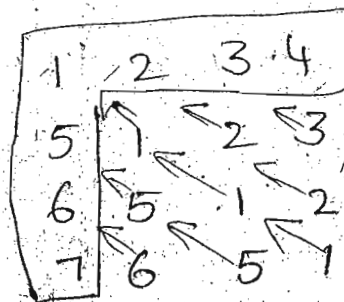
- Square matrix

- $A(i, j) = A(i-1, j-1)$ for all i, j
where $i > 1, j > 1$



Size: $N + N - 1$

Eg:



if $(i \leq j)$

$$= L_0 + 0 + (j - i)$$

$i > j$

$$L_0 + N + i - j - 1$$

if $(i \leq j)$

return $A[j - i]$;

else return $A[N + i - j - 1]$;

17/6/11

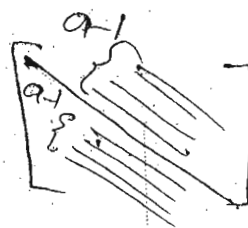
Square band matrix:

- denoted by $D_{n,a}$

where 'n' means $n \times n$ matrix,

'a' means $(a-1)$ diagonals above and below the main diagonal

Size:



$= n + 2 \{ \text{no. of elements present in 'a' diagonals} \}$

$= n + 2 \{ (n-1) + (n-2) + \dots + n - (a-1) \}$

$= n + 2 \cdot \frac{a-1}{2} \{ 2(n-1) + (a-2)(-1) \}$

$= n + (a-1) \{ 2n - a \}$

Symmetric Square band matrix:

Store only the lower diagonals starting from the lowest diagonal in the style of diagonal by diagonals. Then calculate retrieve formula.

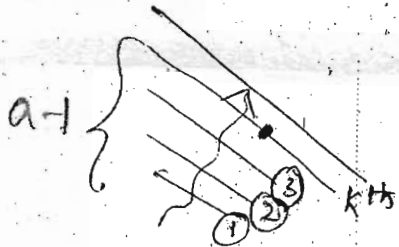
ob: Symmetric

$$A(i, j) = A(j, i)$$

1	2	3	4
2	5	6	7
3	6	8	9
4	7	9	10



$$1 + 2 + \dots + n = \frac{n(n+1)}{2}$$



low

what is the value of k for $a(i, j)$?

$$k = ?$$

D6,4 -

11	12	13	14	15	16
21	22	23	24	25	26
31	32	33	34	35	36
41	42	43	44	45	46
51	52	53	54	55	56
61	62	63	64	65	66

Diagonal elements are circled: 11, 22, 33, 44, 55, 66. The last four are labeled 1, 2, 3, 4 respectively.

element

$a(5, 4)$

D6,4 $\rightsquigarrow$ $k = 3^{rd}$

D6,5 $\rightsquigarrow$ $k = 4^{th}$

the

Ob: 1. "j-j" value remains same/constant throughout

diagonal

2. AS 'a' changes, accordingly 'k' value also getting changed.

$$\therefore k = f(a, i, j)$$

$$k = a - (i - j)$$

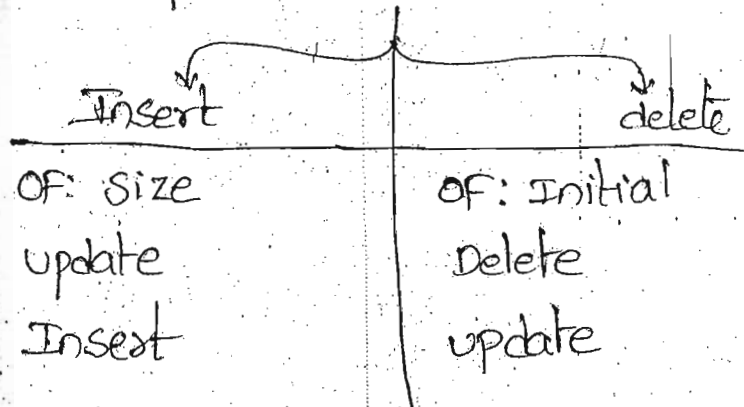
store: dia by dia.

a ₄₁	a ₅₂	a ₆₃	a ₃₁	a ₄₂	a ₅₃	a ₆₄	a ₂₁	a ₃₂	a ₄₃	a ₅₄	a ₆₅
a ₁₁	a ₂₂	a ₃₃	a ₄₄	a ₅₅	a ₆₆						

An arrow labeled '10' points to the element a₅₄ in the first row.

stack

Generic model



Initially $top = 0$

→ Push (S, N, TOP, X)

① if ($top = N$)

write : stack is overflow
exit(1)

② $top = top + 1$

③ $S[top] = X$

④ stop

→ POP (S, N, TOP, Y)

① if ($top = 0$)

write : stack is underflow
exit(1);

② $y = S[top]$

③ $top = top - 1$

④ return y;

ADT of stack:

8

Def: - LIFO (or) FILO model

- one side is open and the other side is closed
- top pointer, pointing to topmost element among the available elements

operations: push(x): Insert element 'x'

pop(): delete topmost element

}

Abstraction:

Hiding the internal implementation details

Multiple stacks:

In an array of size 'm', there are 'n' stacks. The initial configuration is as follows:

for $0 \leq p < n$

$$B[p] = T[p] = \left\lfloor \frac{m}{n} \right\rfloor - 1$$

for $p = n$

$$B[p] = m - 1$$

where

$B = \text{bottom} = \text{fixed}$

$T = \text{top} \neq \text{move}$

fill up the blank

a) $T[p] = T[p+1]$

b) $T[p] = B[p]$

c) $T[p] = B[p+1]$

d) $T[p] = B[p-1]$

stack m

void push(int p, int x)

{ if (?)

print: stack 'p' is overflow

else

$$S[+ + \text{TOP}[p]] = x$$

```

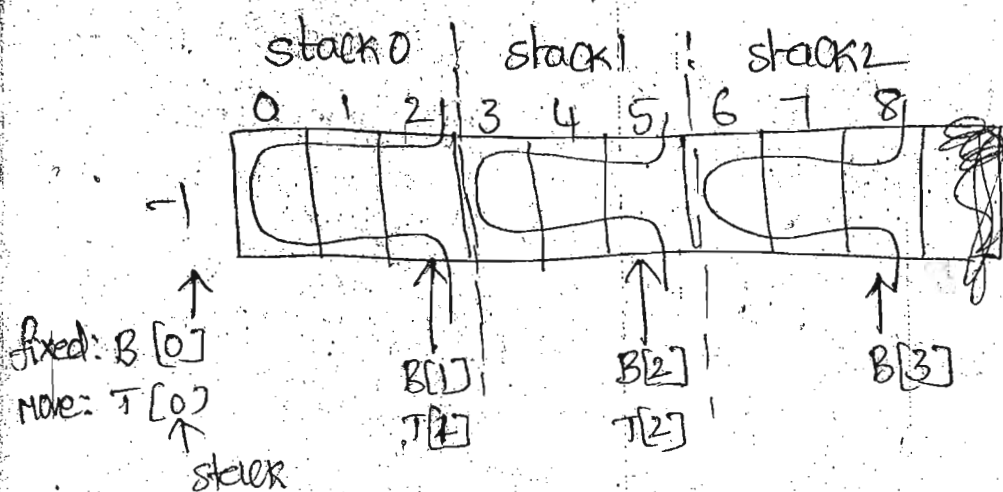
int pop (int p)
{
    if (p < 0)
    {
        print: stack p is underflow
        return 0;
    }
    else
        return S[TOP[p]--];
}

```

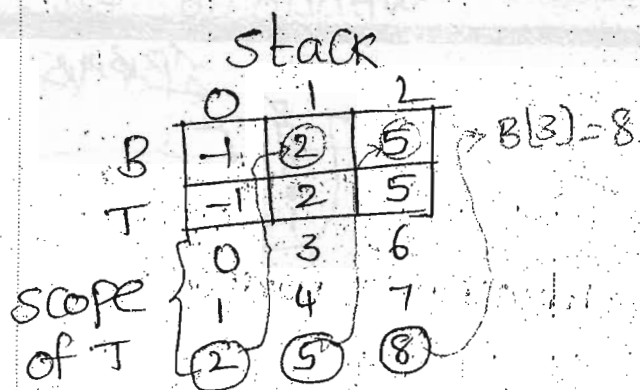
sol:
7/11 consider $m=9, n=3$ stacks
Initial configuration:

$0 \leq p < 3$

	0 stack	1 stack	2 stack	<u>3</u>
$B[p] =$	-1	2	5	$B[p] = 8$
$T[p] =$	-1	2	5	



push overflow: $TOP[p] = Bottom[p+1]$
 $T[p] = B[p+1]$



for underflow: Initial : $TOP[P] = \text{Bottom}[P]$
 i.e. $TOP[P] = B[P]$

stack applications

1. permutations

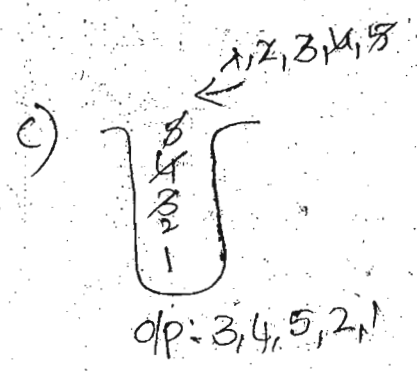
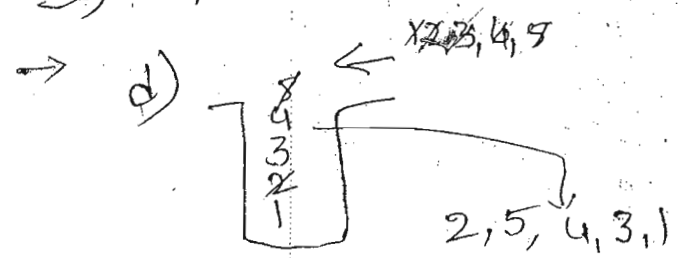
Eg: for n , no. of permutations = $n!$

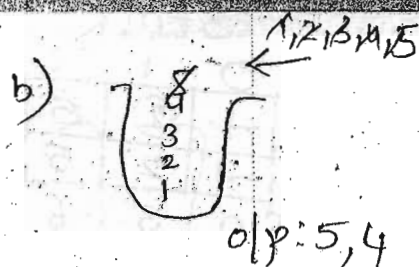
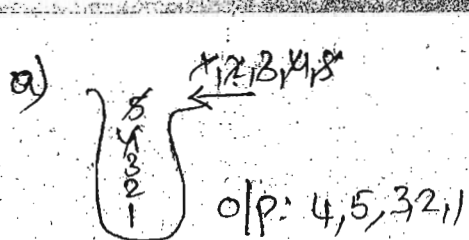
stack is used.

Every element is pushed in and popped out
 logic: Based on the desired op, the desired elements are popped out

P) which of following is invalid stack permutation when input sequence is 1, 2, 3, 4, 5

- a) 4, 5, 3, 2, 1 - valid
- b) 5, 3, 2, 4, 1 - Invalid
- c) 3, 4, 5, 2, 1 - valid
- d) 2, 5, 3, 4, 1 - Invalid





2. polish Notation: (or) LUKASIEWICZ notation

- devised by LUKASIEWICZ, Polish Logician
- It avoids the repeated scanning of infix ~~expression~~ expression
- In one scan, we get result if the expression is either prefix (or) postfix
- machines like compilers prefer either prefix (or) postfix

Rule: The relative position of operands can't be disturbed.

operators: disturbed according to the precedence and associativity rules

Evaluation

↓
Precedence

↓
same

→ Associativity
left / right

Eg:

①

$a + b * c$

$*$ is higher precedence than $+$

prefix

$a + b * c$

$(b * c)$

$a + * b c$

$+ a * b c$

postfix

$a + b * c$

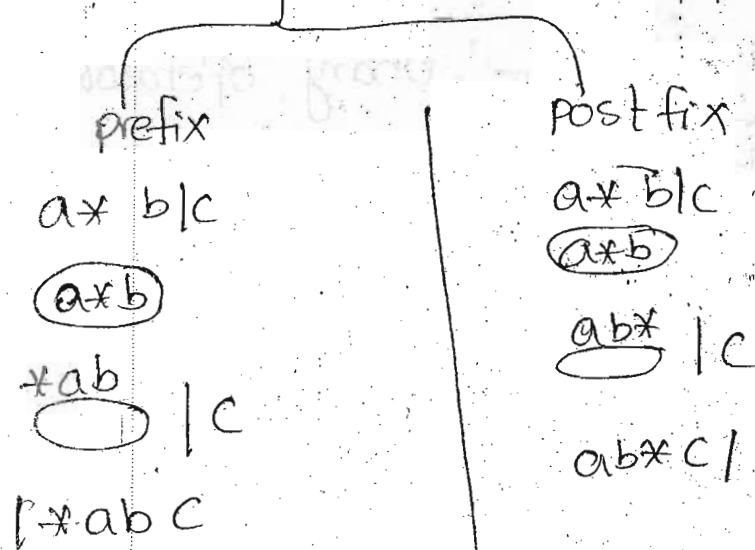
$(b * c)$

$a + b c *$

$a b c * +$

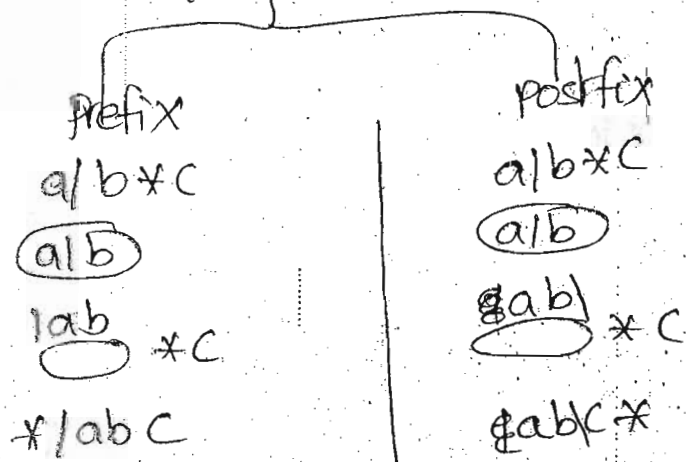
② Left associative

$a * b / c$

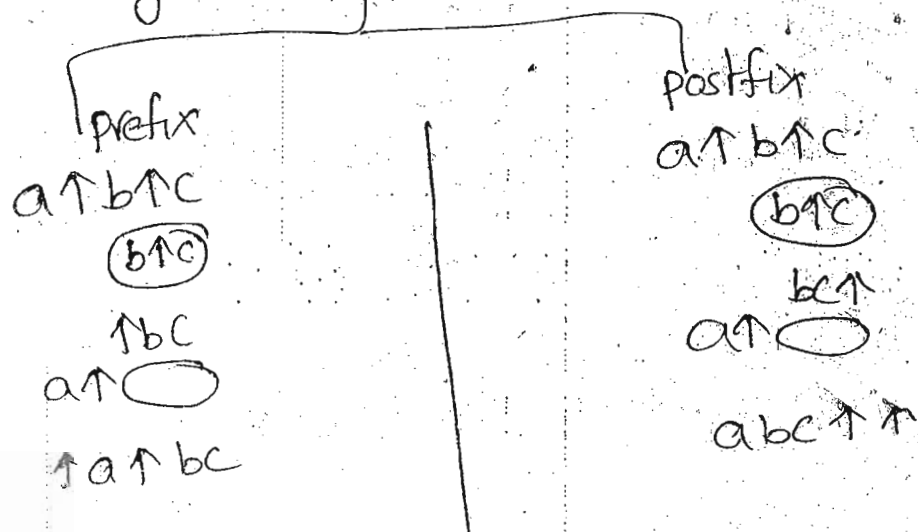


③ Left associative

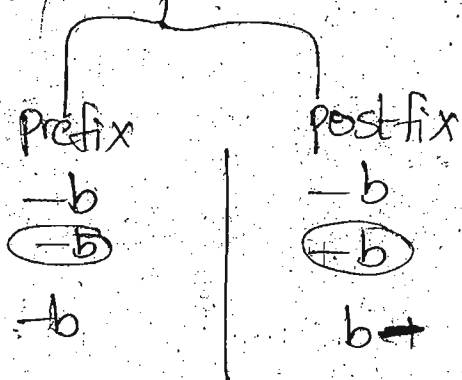
$a / b * c$



④ Right associative

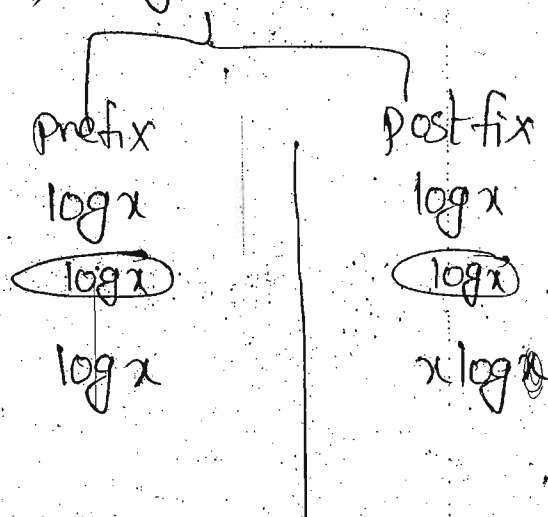


1) $-b$



'-' unary operator

2) a) $\log x$



b) $x!$

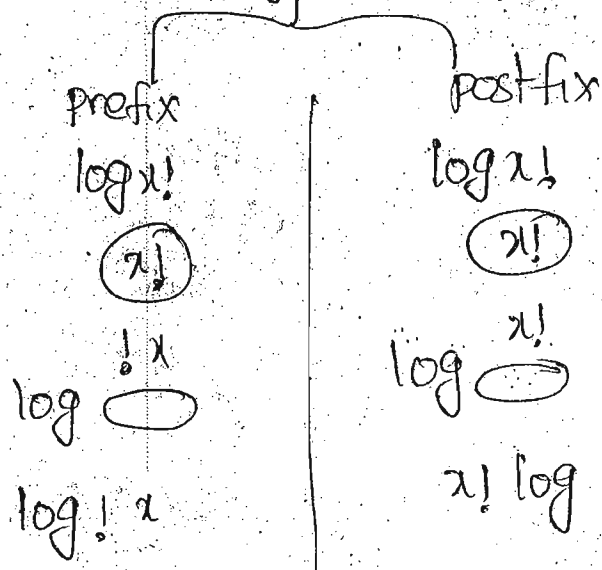
Prefix: $!x$

Postfix: $x!$

3) $\log x!$

Note: $!$ is higher precedence than $\log$

$\log x!$



p) $a = -b + c/d * e + f \uparrow g \uparrow h - i * j$

Note: precedence

- unary
 $\uparrow$: Right associative
 $*$ / : Left "
 $+$ - : left "
 $=$: assignment

Prefix:

$a = -b + c/d * e + f \uparrow g \uparrow h - i * j$

$(-b)$

$-b$

$(f \uparrow g \uparrow h)$

$(g \uparrow h)$

$f \uparrow$

$-b \quad c/d * e \quad f \uparrow g \uparrow h$

(c/d)

$c/d * e$

$-b * c/d * e$

$f \uparrow g \uparrow h$

$(i * j)$

$i * j$

$-b * c/d * e$

$f \uparrow g \uparrow h$

$(-b * c/d * e)$

$+$

$(-)$

$+ -b * c/d * e$

$+$

$(-)$

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

$(-)$

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

$(-)$

$= a - + + -b * c/d * e f \uparrow g \uparrow h i * j$

post-fix

$$a = -b + c/d * e + f \uparrow g \uparrow h - i * j$$

$$(-b) \quad (c/d)$$

$$b - \quad (cd) * e \quad f \uparrow \quad (gh) \uparrow \quad i * j$$

$$(-b) + (cd * e) + (fgh) \uparrow \uparrow - \quad (i * j)$$

$$b - cd / e * + (fgh) \uparrow \uparrow - (i * j)$$

$$b - cd / e * + fgh \uparrow \uparrow + \quad (i * j)$$

$$a = b - cd / e * + fgh \uparrow \uparrow + i * j -$$

$$a b - cd / e * + fgh \uparrow \uparrow + i * j - \pm$$

P) $a < b$ OR $c > d > e$ AND $f < g$

8/11/11

Obs

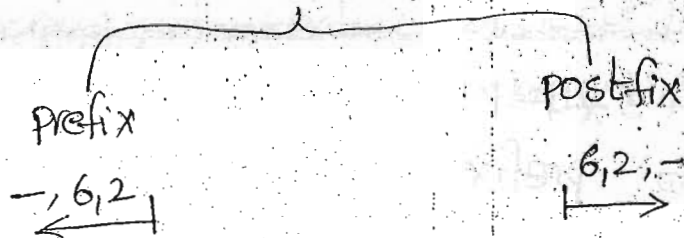
2)

Gate

1)

2)

3)

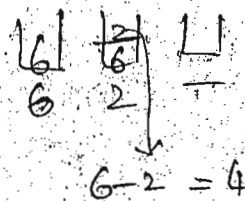


operand: push

operator: (i) pop

{ never pushed in } (ii) evaluate

(iii) push



Obs: 1) In postfix, topmost element is second operand and it is evaluated from left to right

2) In prefix, evaluation is done from right to left and topmost element is first operand.

Gate p) 8, 2, 3, ^, /, 2, 3, *, +, 5, 1, *, + evaluate this postfix

1) which of following is not possible stack content

- a)

8	2	3
---	---	---

 b)

7	5	1
---	---	---

 c)

1	2	3	*
---	---	---	---

 d)

1	6
---	---

operator is not pushed

2) given the status of stack contents after evaluating first 'x'

- a)

6	1
---	---

 b)

1	6
---	---

 c)

5	1
---	---

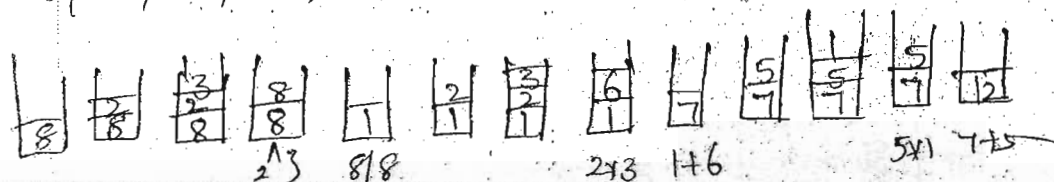
 d)

1	2	3
---	---	---

3) final result is

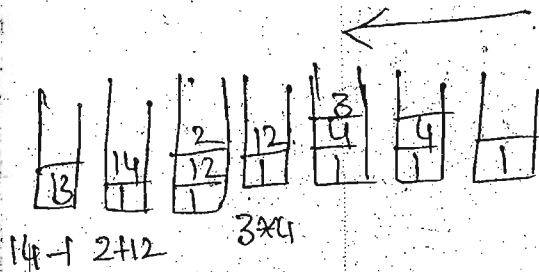
- a) 12 b) -12 c) 13 d) -13

8, 2, 3, ^, /, 2, 3, *, +, 5, 1, *, +



p) Infix $2 + 3 \times 4 - 1$
convert into prefix and evaluate

$- + 2 \times 3 4 1$



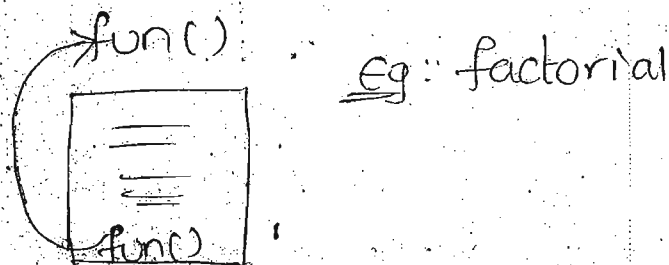
3) Recursion:

Types:

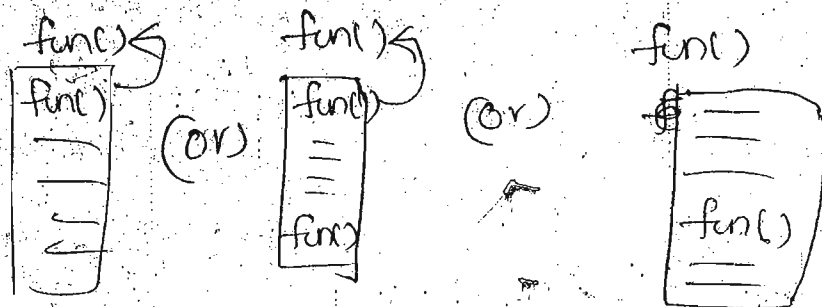
i) Direct recursion

Def: A function calling to itself

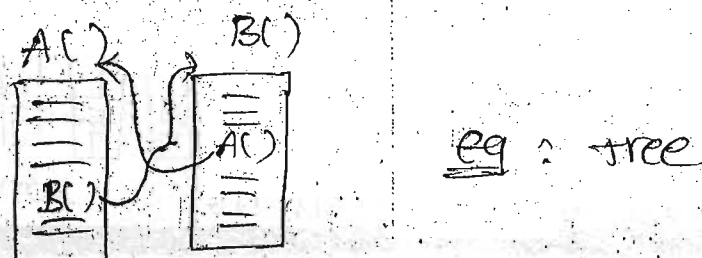
a) Tail recursion



b) Non tail Recursion (or) head recursion



ii) Indirect recursion



iii) excessive recursion

By default: The nature of recursion is excessive, means does not remember the previously evaluated values

eg: fibonacci

iv) Nested Recursion

eg: Ackerman's function

eg:

tail Recursion

1) void func (int x)

{ if (x > 0)

{ 1) printf ("%d", x);

2) func (x-1);

}

func(3) → what is o/p?

→ f(3) → 3

f(2) → 2

f(1) → 1

o/p: 3 2 1

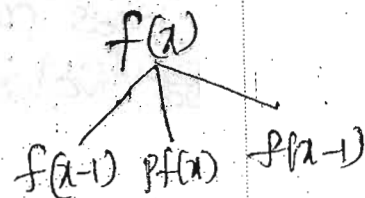
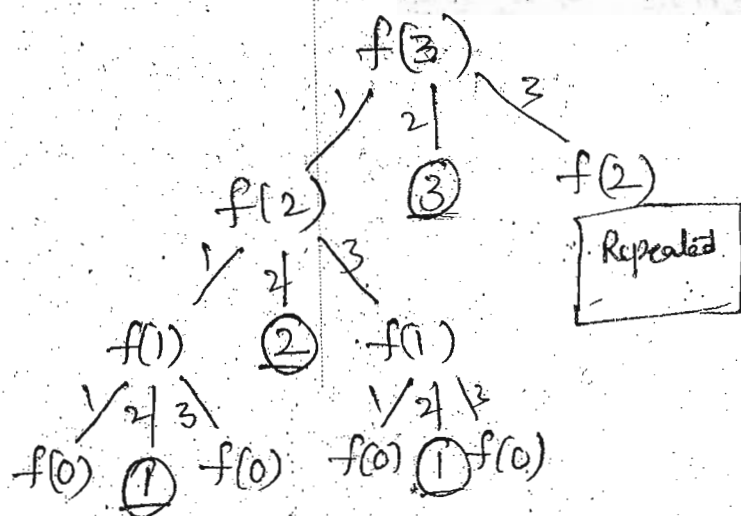
2) void func (int x)

{ if (x > 0)

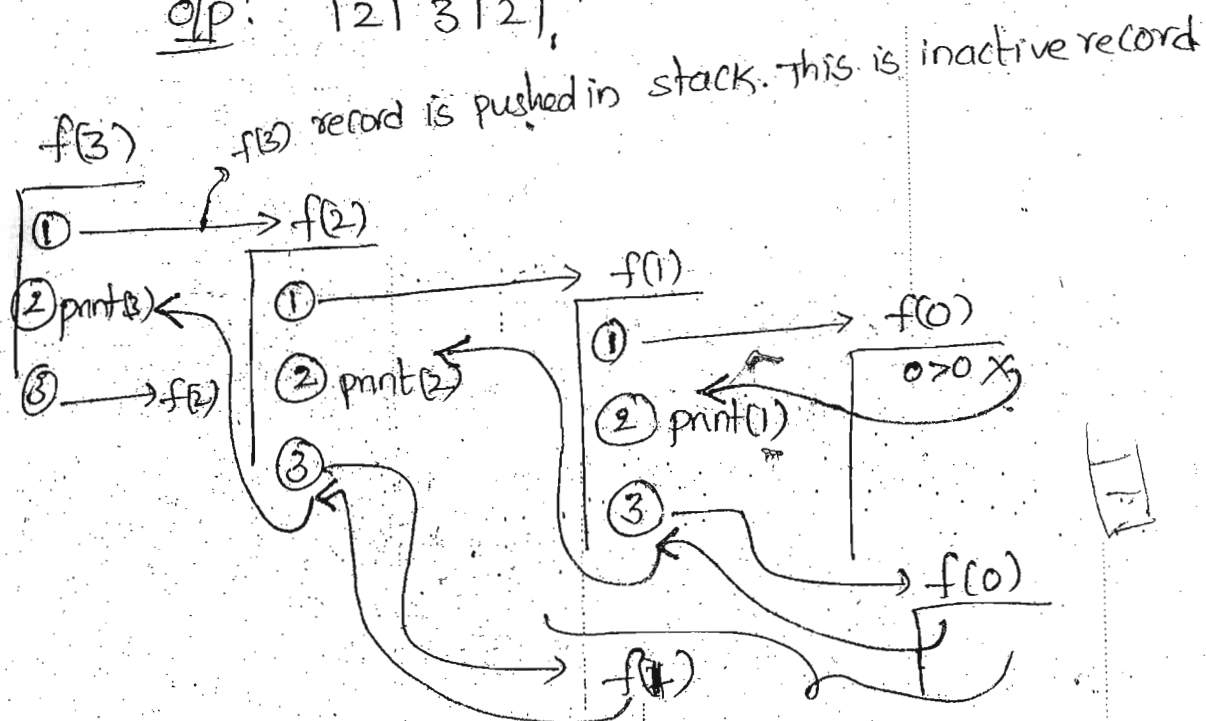
{ 1) func (x-1);

2) printf ("%d", x);

3) func (x-1);



o/p: 1213121



o/p: 1213121

1999G

1) void func(char *x)

{ if (*x != NULL)

{ ① fun(x+1);

② fun(x+1);

③ printf("%c", *x);

}

(00) x[0];

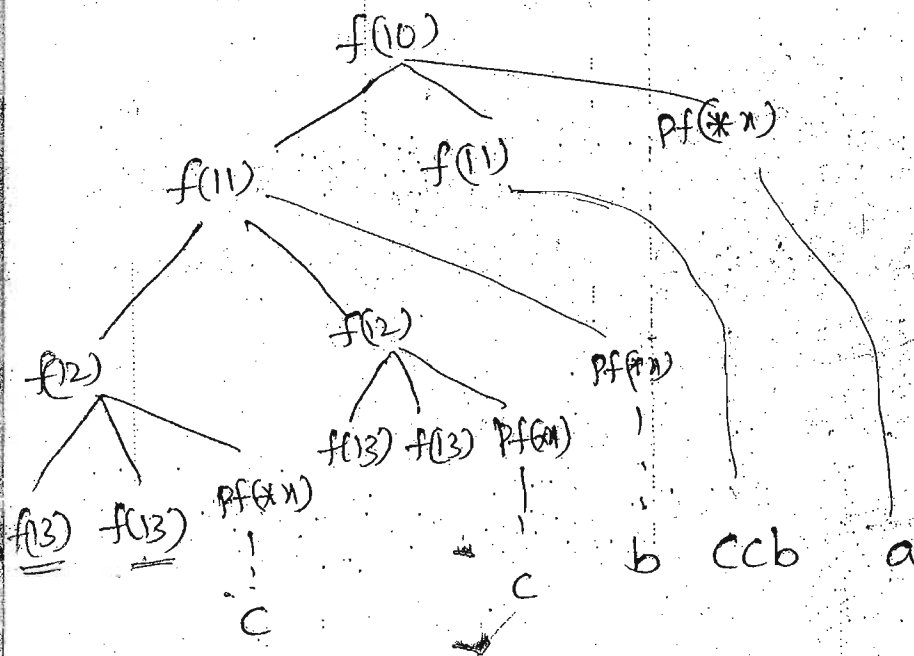
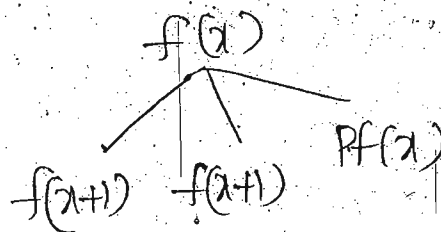
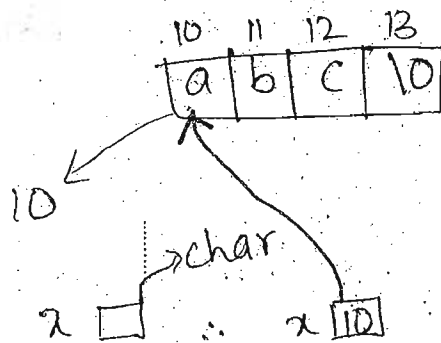
```
void main()
```

```
{
```

```
func("abc");
```

```
}
```

Sol :



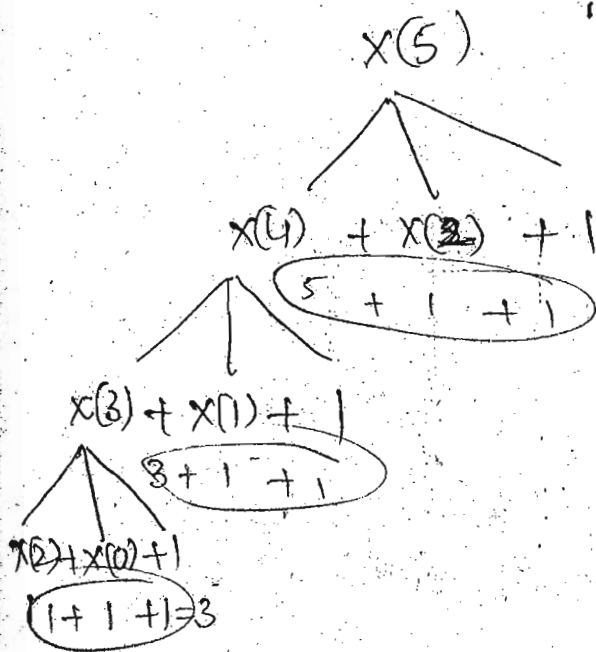
o/p: ccbcbba

```

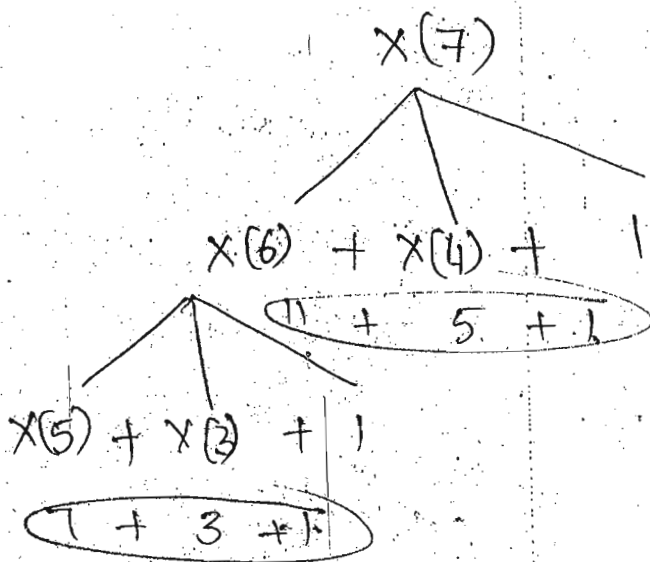
p) int x(int N)
    {
        if (N < 3)
            return 1;
        else
            return x(N-1) + x(N-3) + 1;
    }

```

Q) How many total invocations for evaluating $x(x(5))$:



= 7



= 17

total invocations = 24

for $x(5) = 7$

$x(7) = 17$

$x(x(5)) = 24$

obs: return value = no. of invocations 15

Excessive recursion:

eg: fibonacci series

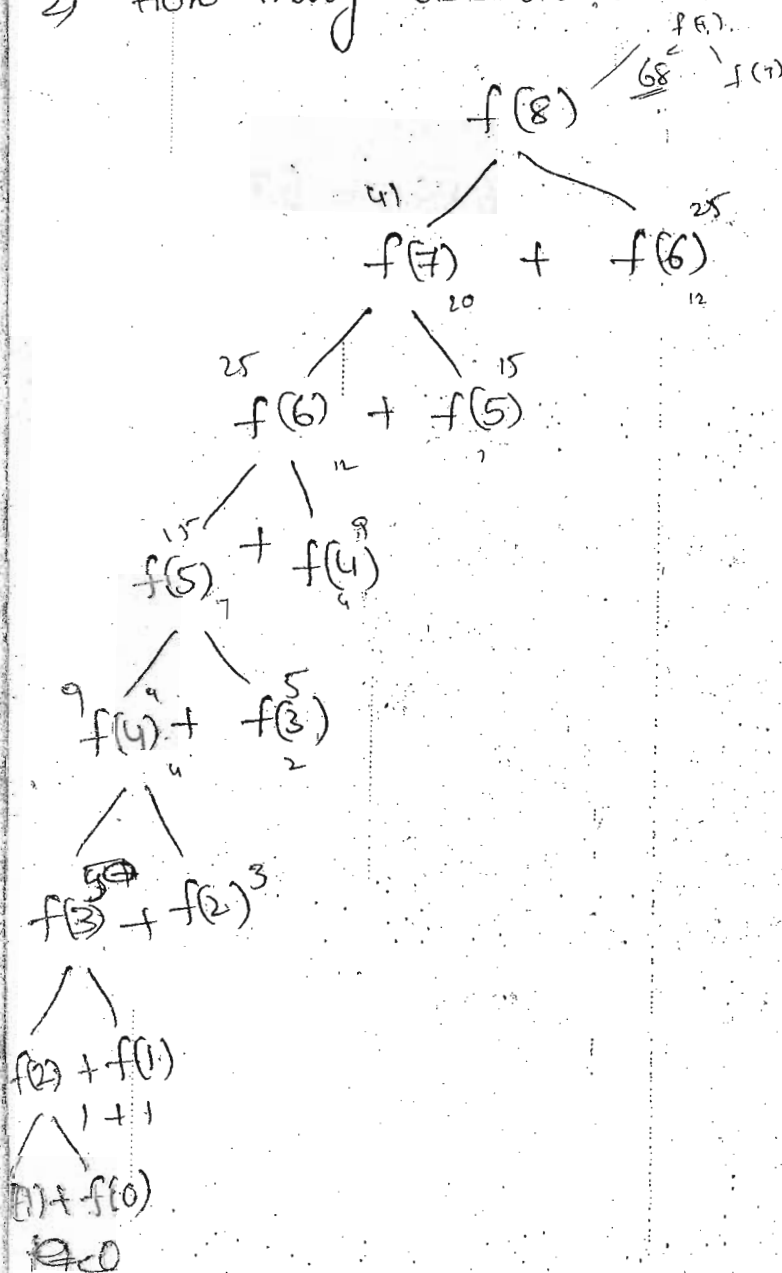
$$\text{fib}(n) = 0, n=0$$

$$= 1, n=1$$

$$= \text{fib}(n-1) + \text{fib}(n-2), n > 1$$

1) How many invocations for evaluating $\text{fib}(8) = ?$

2) How many additions " " " " $\text{fib}(9) = ?$



fib sequence

n	0	1	2	3	4	5	6	7	8	9	10
f(n)	0	1	1	2	3	5	8	13	21	34	55
Invocations				5	9				67		
additions				2	4				33	54	

Invocations of $f(n) = 2 \times f(n-1) - 1$

$$f(8) = 2 \times f(9) - 1$$

$$= 68 - 1 = 67$$

Additions of $f(n) = f(n-1) - 1$

$$f(9) = f(10) - 1$$

$$= 55 - 1 = 54$$

24/11/11

p) fillup the blank with options

- a) n
- b) n-1
- c) n+1
- d) 0

$$\text{pow}(x, n) = \begin{cases} 1, & n=0 \\ 0, & x=0 \\ x \times \text{pow}(x, n-1), & n>0 \\ \frac{1}{x} \times \text{pow}(x, n+1), & n<0 \end{cases}$$

Sol:

$$n=0 \Rightarrow x^n = x^0 = 1$$

$$x=0 \Rightarrow x^n = 0^n = 0$$

$$3^3 = 3 \times 3^2 \Rightarrow x^n = x \times x^{n-1}$$

$$2^3 = 2^1 \times 2^2 \Rightarrow x^n = \frac{1}{x} \times x^{n+1} \quad ; \quad n < 0$$

option C

Modify the above routine in such a way that the no. of multiplications will be less than 10 for evaluating 2^{31}

Hint: the function is not excessive.

Excessive: does not remember the previously evaluated values
Not excessive: Remembers previously evaluated values

Eg

$$2^3 = 2 \times 2^2$$

$$2^2 = 2 \times 2^1$$

$$2^1 = 2 \times 2^0$$

$$2^0 = 2$$

— 3 multiplications

if $x^n = x$ if $n=1$

$$\Rightarrow 2^3 = 2 \times 2^2$$

$$2^2 = 2 \times 2^1$$

$$2^1 = 2$$

→ 2 multiplications

for $2^3 \rightarrow 3$ (or) 2 multiplications

$2^{31} \Rightarrow 31$ (or) 36 multiplications

↓
10 multiplications

Practically is to be decreased
 Division operation

$$2^4 = 2^{\frac{4}{2}} \times 2^{\frac{4}{2}} = 2^2 \times 2^2$$

$$2^5 = 2^{\frac{5}{2}} \times 2^{\frac{5}{2}} = 2^2 \times 2^2 \times 2$$

$$\text{pow}(x, n) = \begin{cases} x & \text{if } n=1 \\ 0 & \text{if } x=0 \\ \text{pow}(x, \frac{n}{2}) * \text{pow}(x, \frac{n}{2}) & \text{if } n \text{ is even} \\ x * \text{pow}(x, \frac{n}{2}) * \text{pow}(x, \frac{n}{2}) & \text{if } n \text{ is odd} \end{cases}$$

$$2^{37} = 2 * 2^{18} * 2^{18}$$

$$2^{18} = 2^9 * 2^9$$

$$2^9 = 2 * 2^4 * 2^4$$

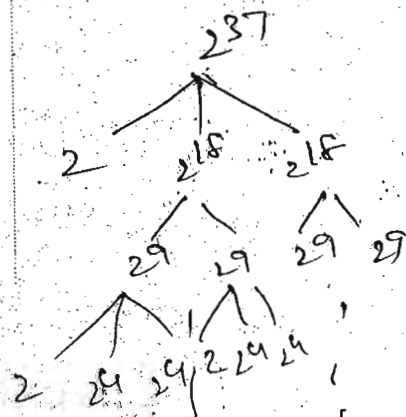
$$2^4 = 2^2 * 2^2$$

$$2^2 = 2 * 2$$

7 multiplications

here if once 2^{18} is evaluated, other is no need to evaluate again as it is not excessive

If excessive then it will be 36 (or) 37 multiplications



Formulate Recursive procedure:

- properties
- ① Base (or) Termination (or) anchor step
 - ② Recursive (or) repeated (or) inductive step

Eg:

Multiplication $\rightarrow$ Repeated additions

Eg: $2 \times 3 = 6 \rightarrow$

$$\begin{array}{r} 2 \\ + 2 \\ + 2 \\ \hline 6 \end{array}$$

Recursion $a \ b$
 2×3

$$= 2 + (2 \times 2)$$

$$= 2 + (2 + (2 \times 1))$$

$$= 2 + 2 + 2 = 6$$

Algorithm:

$$\text{Mul}(a, b) = \begin{cases} a, & \text{if } b=1 \\ a + \text{mul}(a, b-1), & \text{otherwise} \end{cases}$$

Program:

```
int mul(int a, int b)
{
    if (b==1)
        return a;
    else
        return a + mul(a, b-1);
}

void main()
{
    printf("%d", mul(2, 3));
}
```

Division (integer) $\rightarrow$ Repeated subtraction

Eg: $6/2 = 3 \rightarrow$

$$\begin{array}{l} 6-2=4 \\ 4-2=2 \\ 2-2=0 \end{array} \quad \text{Ans } 3$$

$7/2 = 3 \rightarrow$

$$\begin{array}{l} 7-2=5 \\ 5-2=3 \\ 3-2=1 \\ 1-2=? \end{array} \quad \text{Ans } 3$$

Recursion:

$6/2 \Rightarrow \text{Div}(6/2)$

$$= 1 + \text{Div}(4/2)$$

$$= 1 + 1 + \text{Div}(2/2)$$

$$= 1 + 1 + 1 = 3$$

Diagram showing recursive steps: $6-2 \rightarrow 4-2 \rightarrow 2-2 \rightarrow 0$ with $a=b$ at the end.

$$\text{Div}^{a,b}(7,2)$$

$$1 + \text{Div}^{a,b}(5,2)$$

$$1 + \text{Div}^{a,b}(3,2)$$

$$1 + \text{Div}^{a,b}(1,2)$$

$a < b$

0

$$1 + 1 + 1 + 0 \Rightarrow 3$$

Algorithm:

$$\text{Div}(a,b) = 0 \quad \text{if } (a < b)$$

$$= 1 + \text{Div}(a-b, b), \text{ otherwise}$$

Program:

```
int div (int a, int b)
```

```
{
    if (a < b)
        return 0;
```

```
    else
```

```
        return 1 + div (a-b, b)
```

```
}
```

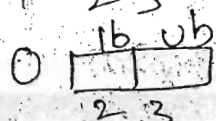
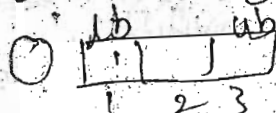
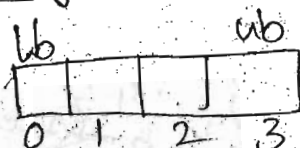
```
void main()
```

```
{
```

```
    printf ("%d", div(7,2));
```

```
}
```

Printing the array elements in reverse order



Left to right

Alg:
$$\text{array}(a, lb, ub) = \begin{cases} \text{Pf}(a[lb]); & \text{if}(lb == ub) \\ \text{① Pf}(a[lb]); \\ \text{② array}(a, lb+1, ub); \end{cases} \text{otherwise}$$

This is tail recursion.

4f to print in reverse order swapping ① & ②

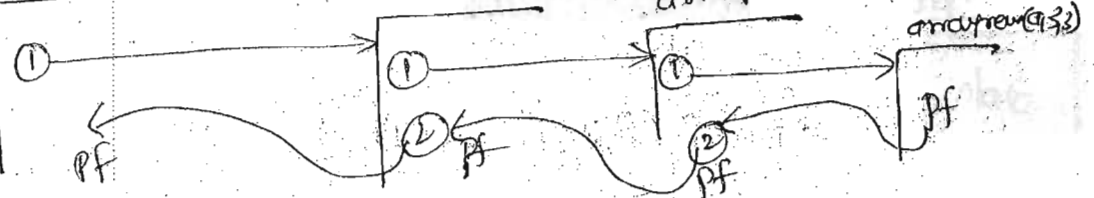
Alg:
$$\text{arrayreverse}(a, lb, ub) = \begin{cases} \text{Pf}(a[ub]); & \text{if}(lb == ub) \\ \text{① arrayreverse}(a, lb+1, ub); \\ \text{② Pf}(a[lb]); \end{cases} \text{other wise}$$

This is head recursion

Eg:

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

arrayreverse(a, 0, 3)



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

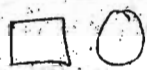
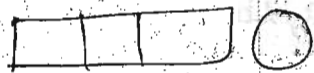
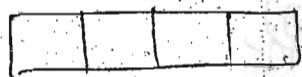
10	-20	30	-40
----	-----	----	-----

-20	30	-40
-----	----	-----

30	-40
----	-----

o/p : -40, 30, -20, 10

Reversing can also be done using tail recursion



$$\text{Reverse}(a, lb, ub) = \begin{cases} \text{pf}(a[ub]); & \text{if}(ub == 0) \\ \left. \begin{array}{l} \text{① pf}(a[ub]); \\ \text{② Reverse}(a, lb+1, ub); \end{array} \right\} & \text{otherwise} \end{cases}$$

Program:

```
int arrayreverse (int a[], int lb, int ub)
```

```
{
    if (lb == ub)
        printf ("%d", a[lb]);
```

```
    else
```

```
    {
```

```
        ① arrayreverse (a, lb+1, ub);
```

```
        ② printf ("%d", a[lb]);
```

```
    }
```

```
}
```

```
void main ()
```

```
{
```

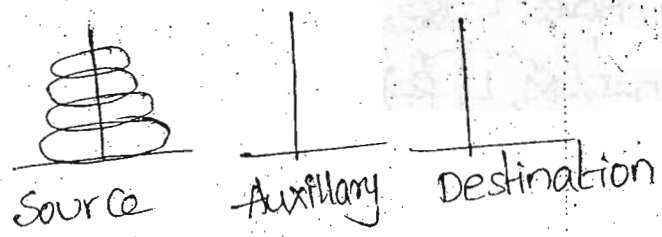
```
    arrayreverse (a, 0, 3);
```

```
}
```

Homework:

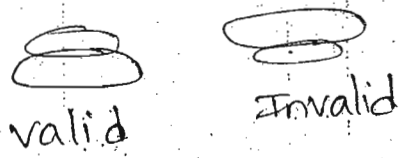
- 1) min. of array elements
- 2) max of array elements
- 3) linear search } iterative → recursion
- 4) Sorting algorithms }

Rules of Hanoi problem:



only one disk can be moved at a time

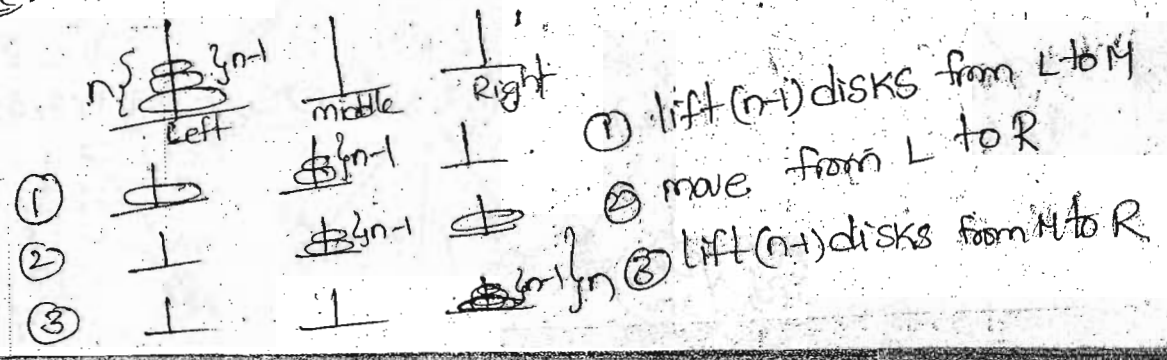
Observation:



- Q How many times each disk is moved?
- Q If the no. of disks, $n=3$ then, after
 - a) How many invocations the first move takes place
 - b) How many invocations the last move takes place
 - c) total no. of moves
 - d) total no. of invocations/calls
- Q Modify the routine in such a way that the no. of calls can be reduced?

Solution:

Note: Recursion is user friendly not machine friendly



```
void TOH(int L, int M, int R)
```

```
{
    if (n != 0)
    {
        ① TOH (n-1, L, R, M);
        ② print: Move L-R
        ③ TOH (n-1, M, L, R)
    }
}
```

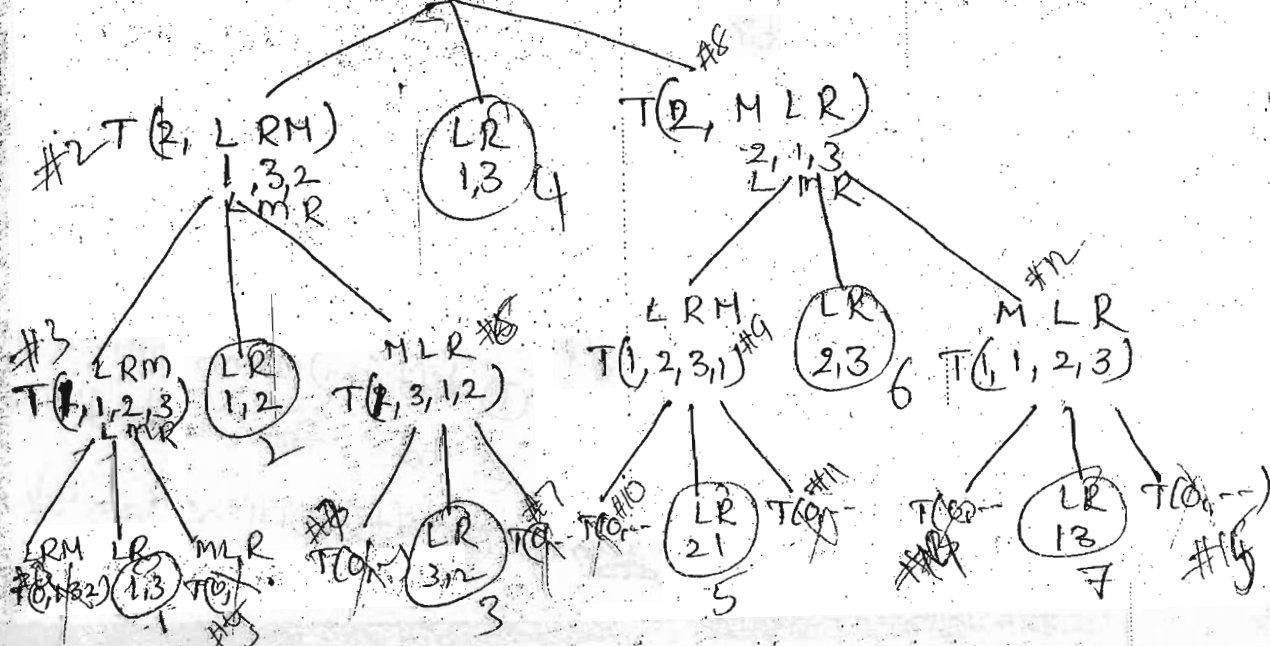
```
void main ( )
```

```
{
    TOH (3, 1, 2, 3);
}
```

ie. $T(n, L, M, R)$



#1 $T(3, 1, 2, 3)$
n L MR



Initially

move

1) 1, 3



2) 1, 2



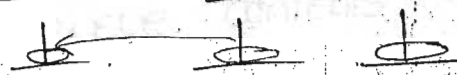
3) 3, 2



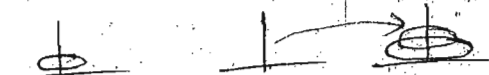
4) 1, 3



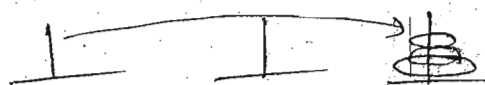
5) 2, 1



6) 2, 3



7) 1, 3



1) first move takes place after 4 invocation
 $n=3 \Rightarrow 4$ invocation

for first move $n+1$ invocations

2) Last move takes place = total calls - 1

3) Total moves = $2^n - 1$

$$n=3 \rightarrow 7 = 2^3 - 1$$

$$n=4 \rightarrow 7+1+7 = 15 = 2^4 - 1$$

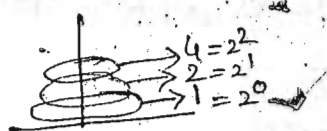
4) Total calls = $2^{n+1} - 1$

$$n=3 \Rightarrow 15$$

$$n=4 \Rightarrow 15 + 15 + 1 = 31$$

5) Each disk is moved how many times

each disk is moved 2^n times



HLG

⑥ $T(0, L, M, R)$
 $\downarrow$
 $n=0?$ is unnecessarily invoked.

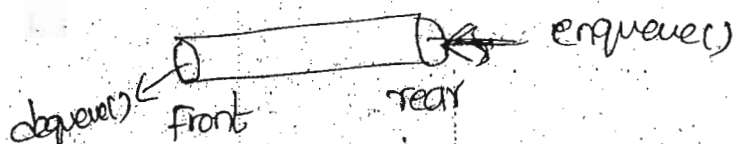
```

∴ void TOH (int n, int L, int M, int R)
{
    if (n == 1)
        print: move L-R
    else
    {
        ① TOH (n-1, L, R, M);
        ② print: move L-R;
        ③ TOH (n-1, M, L, R);
    }
}

```

Queues:

- FIFO (or) LILO model
- Associated with two pointers
 - Rear = enqueue(x): Insert element x
 - front = dequeue(): Delete element



- ⑦ what is the status of queue after following sequence of steps.
- addition of 'a' and then 'b'
 - a deletion
 - addition of 'y' and then 'z'
 - a deletion

(a) $a \ b \ 0 \leftarrow a, b$

(b) $a \leftarrow 0 \ b \ 0$

(c) $0 \ y \ z \ 0 \leftarrow y, z$

(d) $0 \ y \ z \ 0$
 $\swarrow$
 $b \ 0 \ y \ z \ 0$

2008G

(P)

Hw

Reversing queue contents using only enqueue and deque operations. (Refer 31/1/11)

```
void reverse (queue q)
{
    if (q not empty) {
        int x = q.dequeue();
        if (q not empty)
        reverse(q);
        q.enqueue(x);
    }
}
```

a b c

$x = a$
 $2 \rightarrow x = b$
 $2 \rightarrow x = c$
c b a c b c

Initially $F = R = 0$ size of queue
 $\swarrow$ $\nwarrow$ element to be inserted

Algorithm $Q_{insert}(Q, N, F, R, x)$

1. if $(R = N)$
 print: Q is overflow
 exit(1)

2. $R = R + 1$

3. $Q[R] = x$

Algorithm $Q_{delete}(Q, N, F, R, y)$

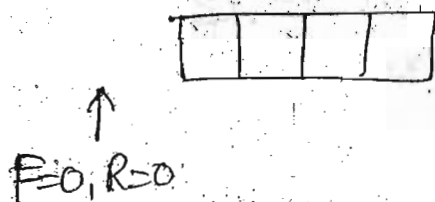
1. if $(F = 0)$
 write: Queue is underflow
 exit(1)

2. $y = Q[F]$

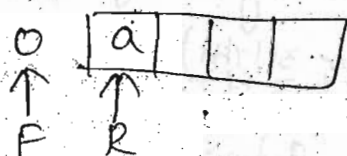
3. $F = F + 1$

According generic model of insertion & deletion.

place to store the deleted element



insert(a)

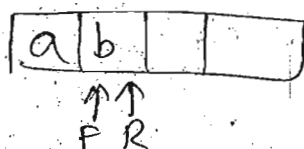


delete()

But gives underflow

∴ when inserted, F is to be increased

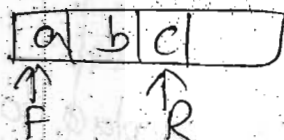
∴ insert(b)



delete()

b comes out

∴ for first element insertion only F is increased



delete()

∴ a comes out.

Algorithm QInsert (Q, N, F, R, X)

1. If (R = N)

print : Q is overflow

exit()

2. R = R + 1;

3. $Q[R] = X$

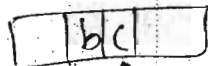
22

*4. if ($F=0$)
 $F=1$

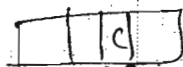
5. stop



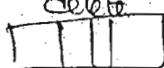
delete



delete



delete



F can't be incremented

∴ if $F=R$ queue empty i.e.
again go to initial position

Algorithm delete (Q, N, F, R, Y)

1. if ($F=0$)

write queue is underflow
exit(1)

2. $Y = Q[F]$

*3. if ($F=R$)
 $F=R=0$

else
 $F=F+1$

4. return Y

→ Note: linear Q is inefficient queue
circular queue is efficient queue

circular Queue:

Algorithm coInsert (Q, N, F, R, X)

1. if $[(R = N \ \&\& \ F = 1) \text{ OR } (R+1) = F]$ $\rightarrow (R+1) \% N = F$
print: Q is overflow.

exit(0)

2. if $(R = N)$

$R = 1$

else

$R = R + 1$

3. $Q[R] = X$

4. if $(F = 0)$

$F = 1$

5. stop

Algorithm coDelete (Q, N, F, R, Y)

1. if $(F = 0)$

write Q is underflow

exit(0)

2. $Y = Q[F]$

3. if $(F = R)$

$F = R = 0;$

else

if $(F = N)$

$F = 1$

else

$F = F + 1$

4. return Y

Reversing the Queue contents:

void do()

{ if (!q.is.empty())

{ ① int temp = dequeue();

② do();

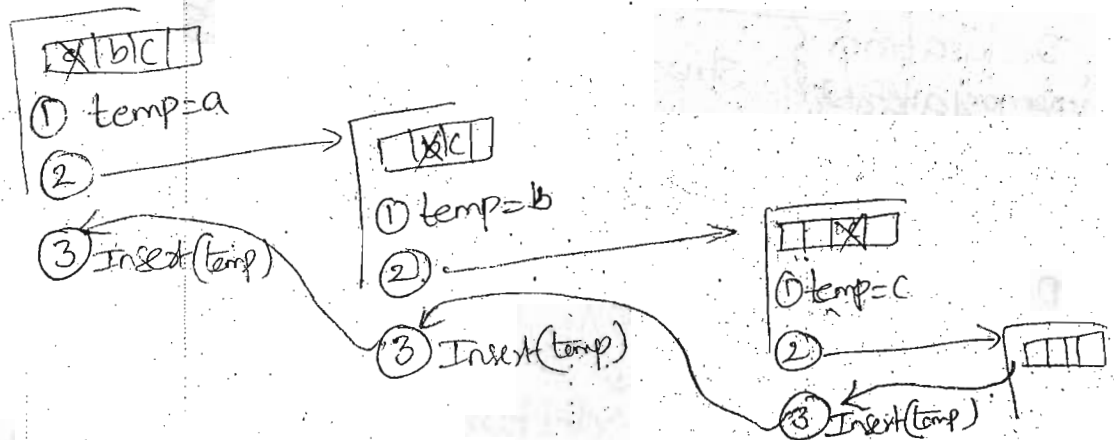
③ enqueue(temp);

{

}

Eg:

a b c



After reversing → c b a

observation:

If 'temp' variable is declared globally the o/p is

c c c

The third step is getting executed while stepping backwards since it is head recursion.

Linked lists

→	Indirect selector	Direct Selector
	$P \rightarrow \text{data}$	$(*P) \cdot \text{data}$
	user-friendly operator	machine friendly operator

→

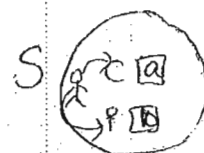
```

struct sample
{
    char c;
    int i;
};
    
```

prototype
{memory is not allocated}

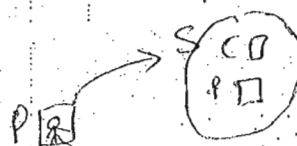
Declaration {
{memory allocated} struct sample s;

Definition {
 $s.c = 'a';$
 $s.i = 10;$
 selector



struct sample *P; sample (holds address)
 P is a pointer expecting
 // sample type

$P = \&s;$ // P holds address of s
 // it is called by name hence control
 is inside 'P'



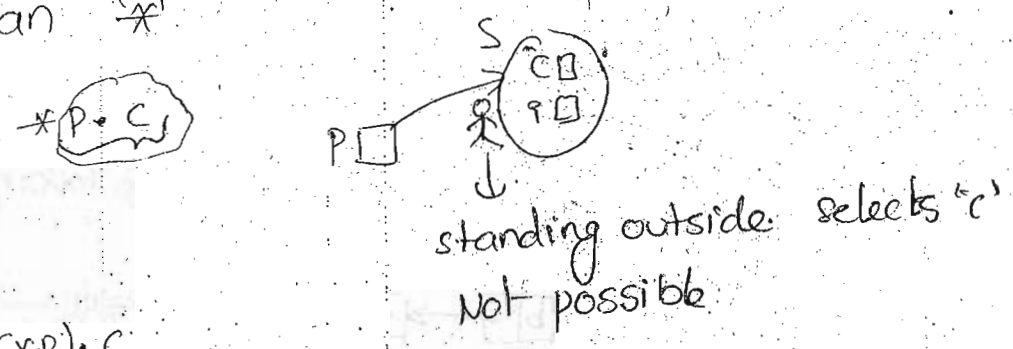
It is same as



*p
↓
selects the content inside 'p' i.e. address of 's'

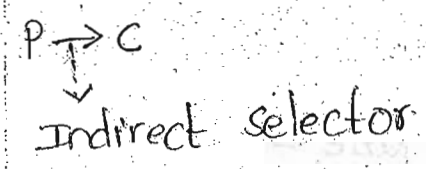
*p.c
↑
Selector i.e. selects 'c' and places values

But core dumped, segmentation fault ~~type~~ errors
come because '*' is higher precedence
than 'c'



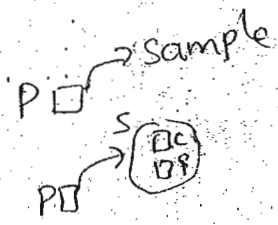
Hence (*p).c

This is same as



→

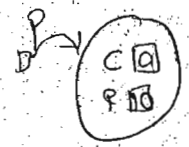
struct sample *p;



p = &s;

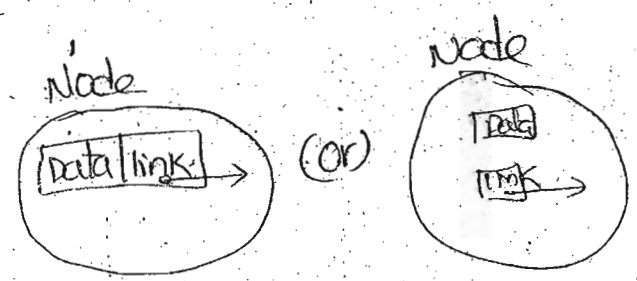
(*p).c = 'a' (or) p->c = 'a'

(*p).i = 10 (or) p->i = 10



→ Definition of linked list:

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



struct node *first = NULL; // Initialization
first -> NULL

Access :

$P \rightarrow \text{Data}$ (or) $(*P) \cdot \text{Data}$

$P \rightarrow \text{link}$ (or) $(*P) \cdot \text{link}$

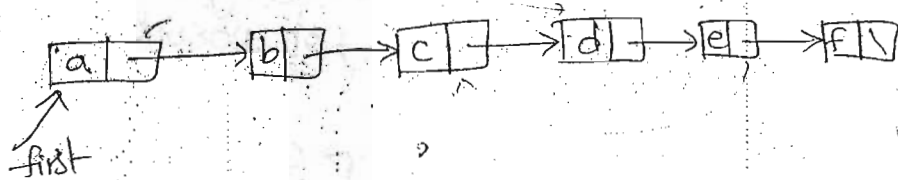
forward :

$P = P \rightarrow \text{link}$

Note:

linked list is a self referential structure

1) what is o/p after executing following steps



struct node *p;

i) $p = \text{first} \rightarrow \text{link} \rightarrow \text{link};$

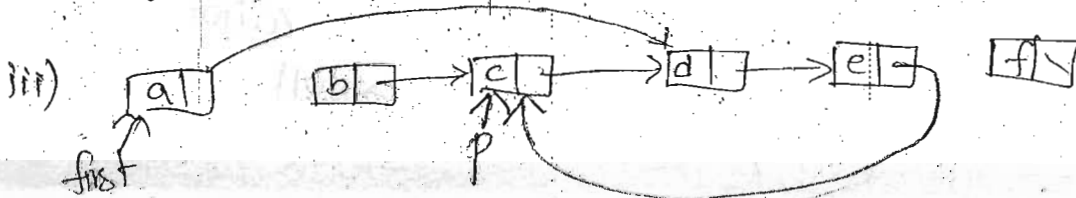
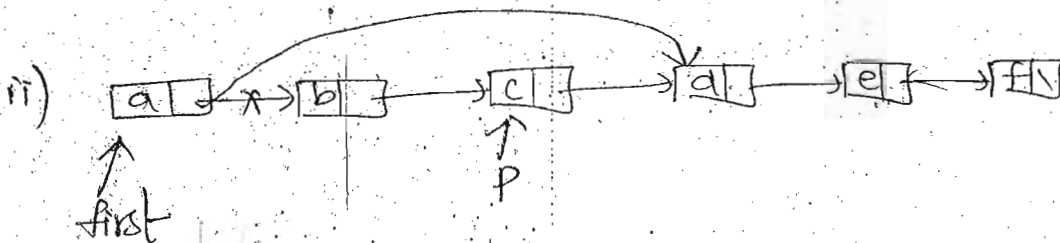
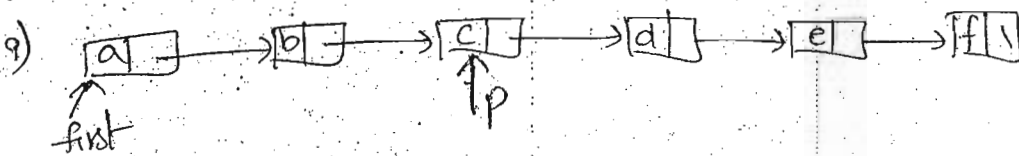
ii) $\text{first} \rightarrow \text{link} = p \rightarrow \text{link};$

iii) $p \rightarrow \text{link} \rightarrow \text{link} = p;$

iv) $\text{printf}(" \%d", f \rightarrow \text{link} \rightarrow \text{link} \rightarrow \text{link} \rightarrow \text{link} \rightarrow \text{data});$

Right side +1
left side same node

Sol:

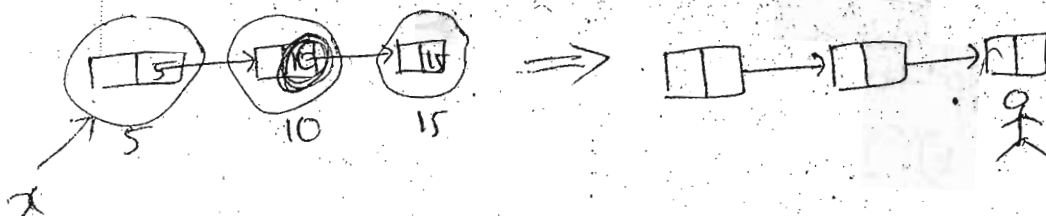


iv) d

Note:

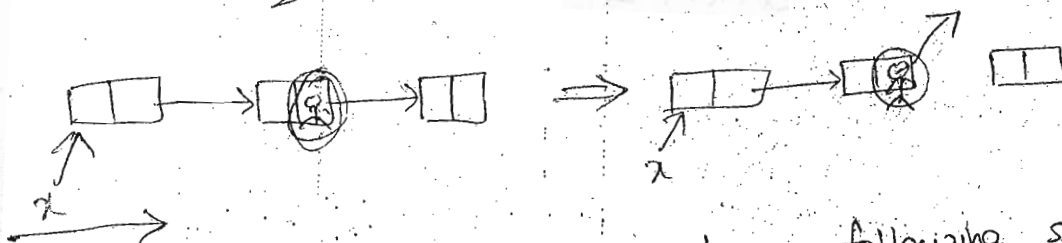
case (i):

$x \rightarrow \text{link} \rightarrow \text{link}$
 ↓
 consider
 i.e. 2 links = 3rd node

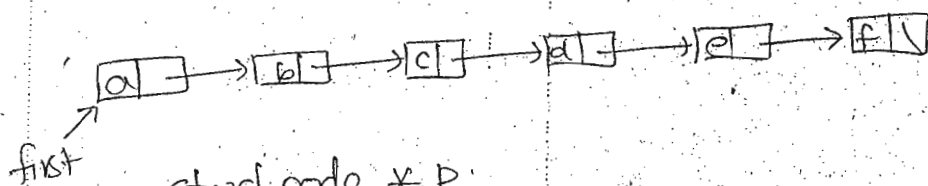


case (ii):

$x \rightarrow \text{link} \rightarrow \text{link} =$
 ↓
 update
 2nd link : update



p) what is o/p after executing following steps



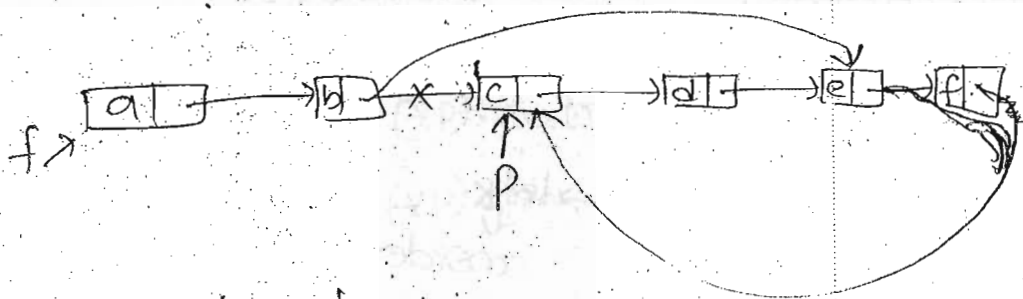
struct node * p;

i) $p = \text{first} \rightarrow \text{link} \rightarrow \text{link};$

ii) $\text{first} \rightarrow \text{link} \rightarrow \text{link} = p \rightarrow \text{link} \rightarrow \text{link};$

iii) $p \rightarrow \text{link} \rightarrow \text{link} \rightarrow \text{link} = p;$

iv) $\text{printf} (" \%d", \text{first} \rightarrow \text{link} \rightarrow \text{link} \rightarrow \text{link} \rightarrow \text{link} \rightarrow \text{data});$



Ans: d

p) fill up the blank with

a) p b) !p c) P → link d) !P → link

1) The following code count the no. of nodes.

```
int DO()
{
```

```
    struct Node *P = f;
```

```
    int counter = 0;
```

```
    while (—)
```

```
    { ++ counter;
```

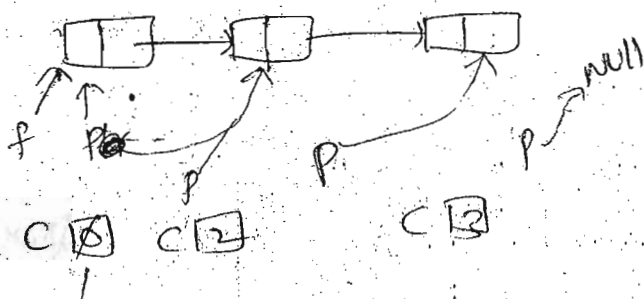
```
      P = P → link;
```

```
    }
```

```
    return counter;
```

```
}
```

Ans: a) p



Note:

1) while (P) (or) while (P != NULL)

This is useful whenever the entire list is to be traversed
e.g. searching, reversing

2) while(!p) ~~(or) while(p != NULL)~~
 (or)
 while (p == NULL)

26

3) while (p->link) (or) while (p->link != NULL)

Observation:

1) while(p) i.e. 'p' is pointing to somebody

2) while(!p) i.e. 'p' is not pointing to somebody

convert above code into recursion

```

int DO (
{
  struct Node *p = p parameter passed from main;
  int counter = 0;
  while (p)
  { ++counter;
    p = p->link;
  }
  return counter;
}

```

while returning increment

→ for recursion function has atleast one parameter
 No local variable
 Iterative loops are not present
 ↓
 convert into conditional loop

```

int struct int DO (struct Node *p)
{
  if (p)
    return 1 + DO (p->link);
  else
    return 0;
}

```

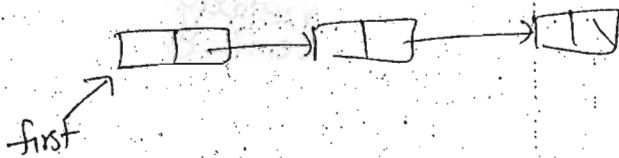
```
void main()
```

```
{
```

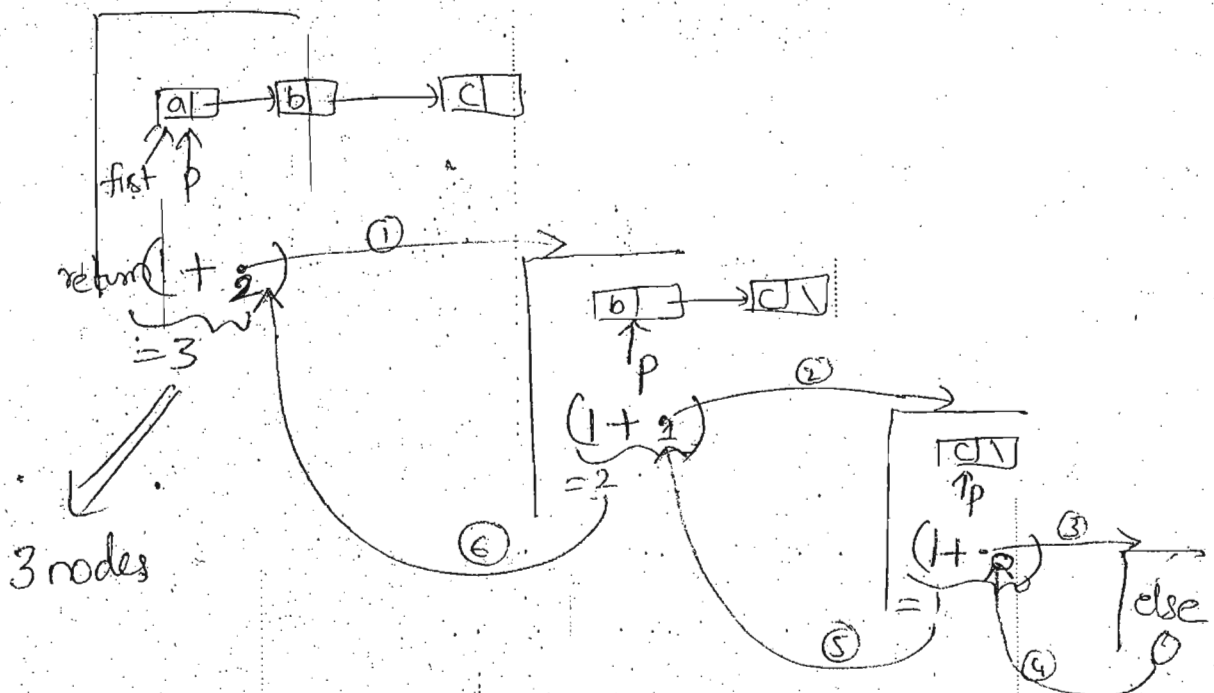
```
    printf("%d", DO(first));
```

```
}
```

Eg.



DO (first)



D) printing the linked list elements in same order

```
void DO (struct Node *p)
```

```
{
```

```
    if (p)
```

```
    {
```

```
        ① print("%d", p->data);
```

```
        ② DO(p->link);
```

```
    }
```

```
}
```

```
void main()
```

```
{
```

```
    DO (first);
```

```
}
```

p) printing the linked list elements in reverse order 22

```
void do do (struct node *p)
```

```
{  
  if (p)
```

```
{
```

```
    ① do (p->link);
```

```
    ② print ("%d", p->data);
```

```
}
```

```
}
```

```
void main()
```

```
{
```

```
    do (first);
```

```
}
```

p) fill up the blank

① p ② q ③ r ④ NULL

The following code reverses a SLL

```
void reverse()
```

```
{
```

```
    struct node *p = first
```

```
    struct node *q = NULL;
```

```
    struct node *r;
```

```
    while (p)
```

```
    {
```

```
        r = q;
```

```
        q = p;
```

```
        p = p->link;
```

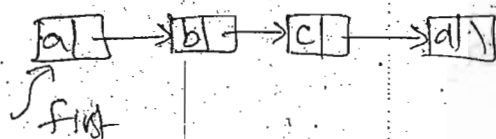
```
        q->link = r;
```

```
    }
```

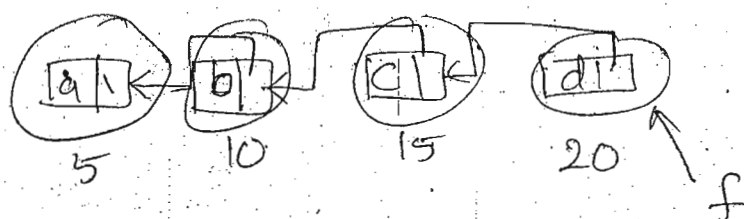
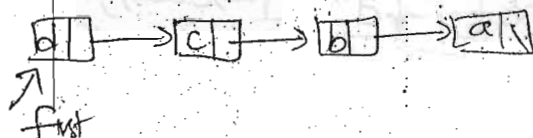
```
    first = —
```

```
}
```

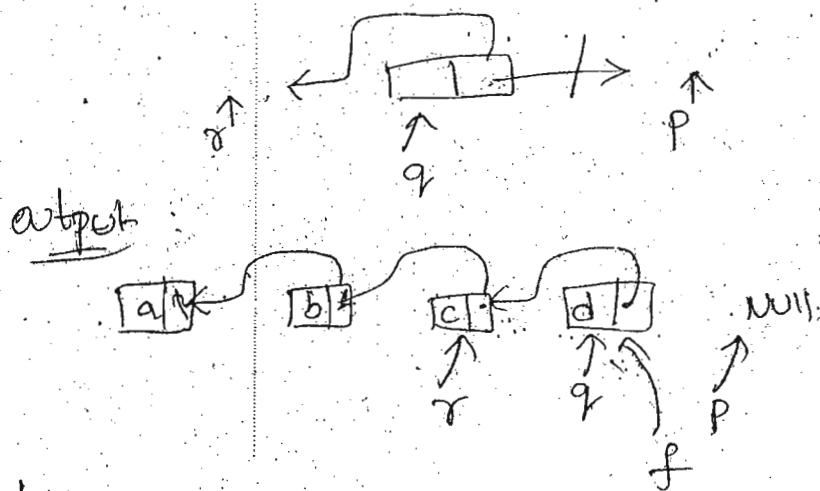
Before reverse



after reverse



Logic



4/8/11

Reversing SLL recursively:

void do(struct Node *p, struct Node *c)

{ if (c == NULL)

{ ① do(c, c->link);

② c->link = p;

else

{ first = p;

```
void main()
```

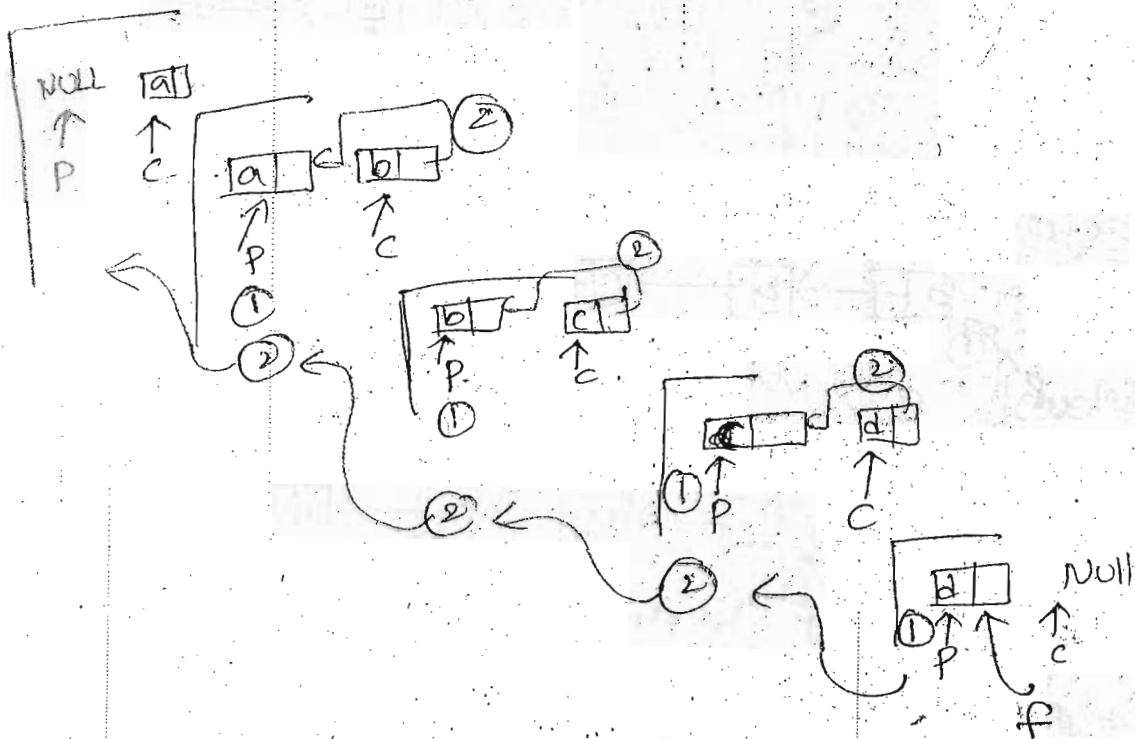
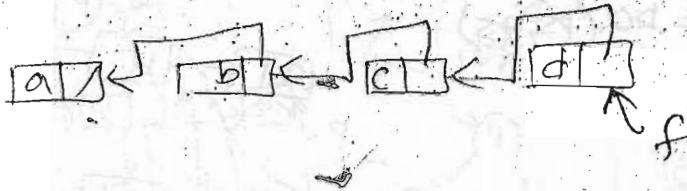
28

```
{
```

```
DO (NULL, first);
```

```
}
```

to make the first node as null, we pass it as an argument



p) what does DO() do on two SLL?

```
struct node* DO (struct node *p, struct node *q)
```

```
{
```

```
struct node *ps;
```

```
struct node *qs;
```

```
if (!p)
```

```
return q; // case(i)
```

```
else if (!q)
```

```
return p; // case(ii)
```

else

{

① $ps = p \rightarrow \text{link};$

// case (iii)

② $qs = q \rightarrow \text{link};$

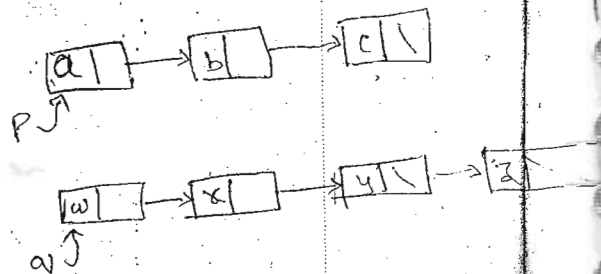
③ $p \rightarrow \text{link} = q;$

④ $q \rightarrow \text{link} = \text{DO}(ps, qs);$

⑤ return $p;$

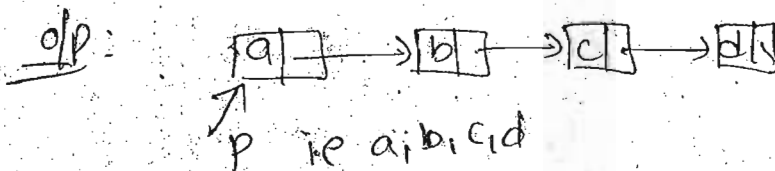
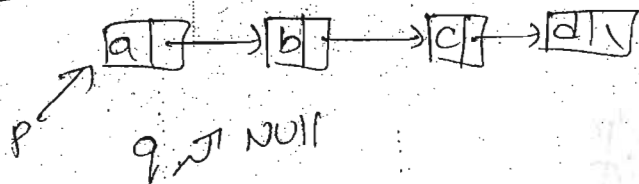
}

}



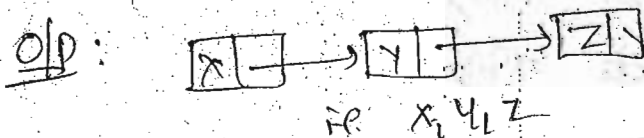
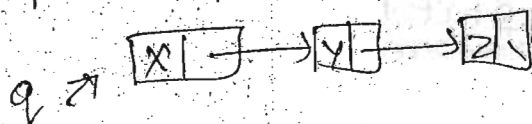
Sol.

Case (i).

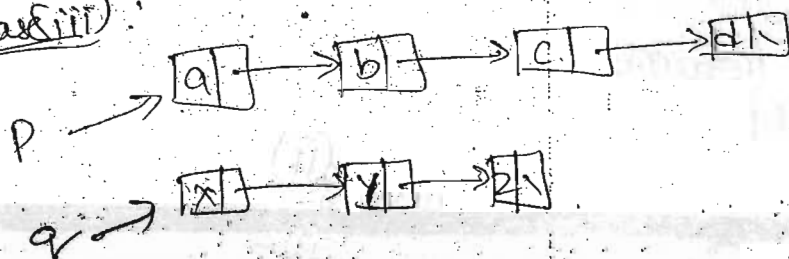


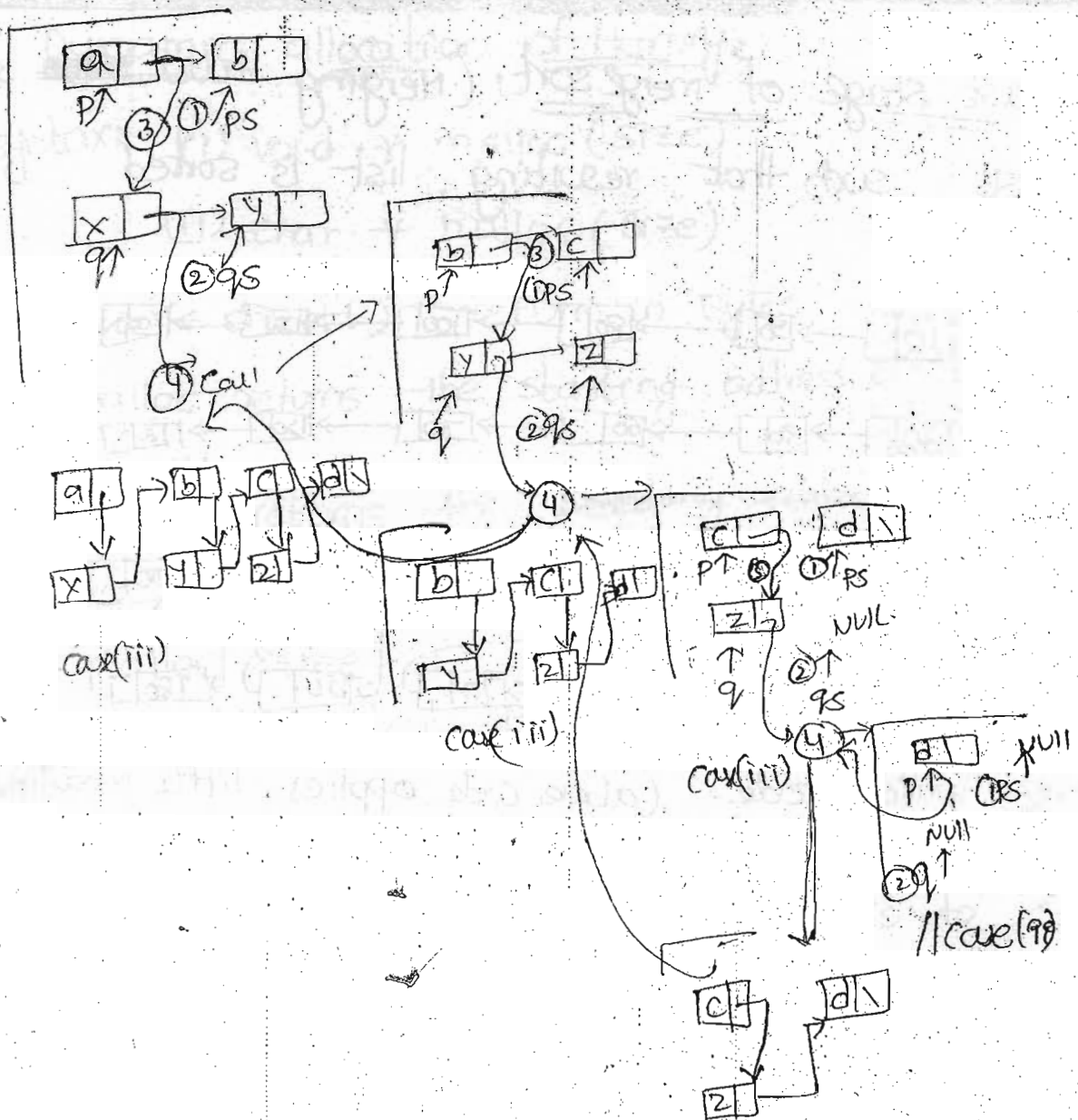
Case (ii):

$p \rightarrow \text{NULL}$

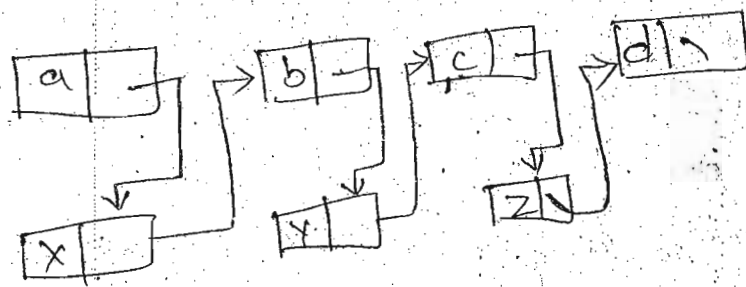


Case (iii):

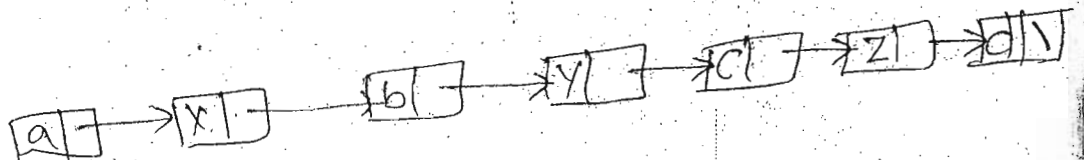




o/p



ie.



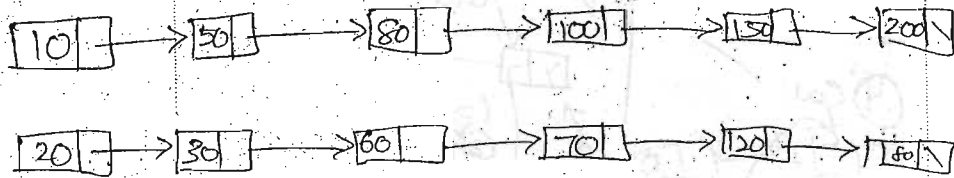
ie.

a, x, b, y, c, z, d

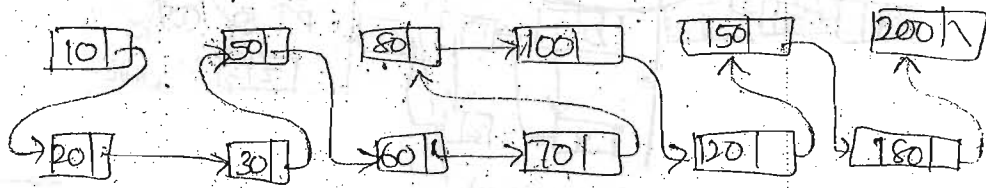
It concatenating two SLLs by choosing the alternative nodes

Last stage of merge sort: (Merging two ~~sorted~~ sorted lists such that resulting list is sorted)

eg.



result:



tho: write code. (above code applies, little modification)

Dynamic allocation of Memory:

syntax: (i) void * malloc (size)

(ii) char * malloc (size)

→ size = sizeof() : return in Bytes

→ malloc returns the starting address of allocated memory

→ malloc returns the ~~starting address~~ NULL if the memory is not allocated

→ malloc x free

→ Declaration x out of scope

→

void * malloc (size)

↓ what type is being stored is to be explicitly mentioned

char * malloc (size)

↓

Here stored data type is by default char. If any other is to be stored, it is mentioned explicitly by using typecasting

eg:

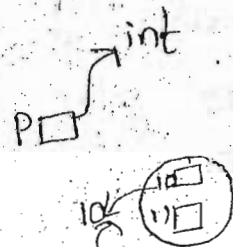
void main()

{

int * p; // pointer declaration

p = (int*) malloc (size of (int));

↓ 'typecasting' is required because we are storing integer.



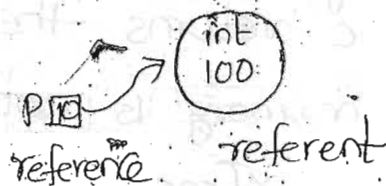
```

if (!P) (or) if (P == NULL)
{
    printf ("no memory");
    exit(1);
}

```

exception handling

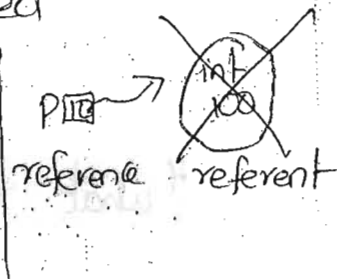
*P = 100; // dereference operator



printf ("%d", *P); // *P = 100;

free(P); // referent ~~is~~ is freed

ie. Memory allocated
by malloc() is freed
by free().



ob: This is dangling pointer.

'P' can ~~be~~ still be reutilized.

Dangling pointer:

pointing to a location which is not present

→ Memory allocated by declaring variables is not freed by calling free(). But it is freed when ever the variable goes out of scope - i.e. reference is removed

eg. { int x; // memory allocated

} → Memory deallocated

eg.

Linked stack.

31

```
struct node * Get node ()  
{  
    struct node *p;  
    P=(struct node *) malloc (sizeof (struct node));  
    return P;  
}
```

return type

declaration

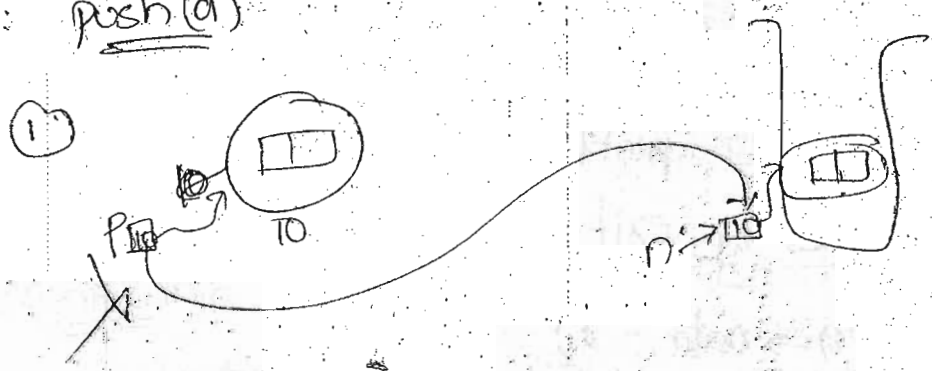
typecasting

```
struct node *top = NULL;
```

```
void push (int x)
```

```
{  
    ① struct node *n = Get node ();  
    if (!n)  
    {  
        printf ("No Memory");  
        exit (1);  
    }  
    ② n -> data = x;  
    ③ n -> link = top;  
    ④ top = n;  
}
```

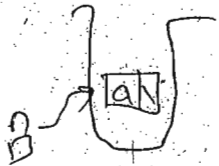
eg: push(a)



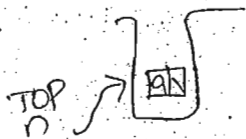
(2)



(3)



(4)



after scope comes out of program, 'n' is deleted



eg: what does do() do on SL

```
struct node *first = NULL;
```

```
void do (struct node int x)
```

```
{ if (!n)
```

```
    struct node *n = getnode();
```

```
    if (!n)
```

```
    {
```

```
        printf ("No Memory");
```

```
    } exit(1);
```

```
    n->data = x;
```

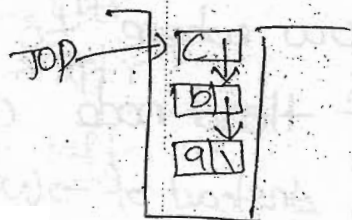
```
    n->link = first;
```

```
    first = n;
```

```
}
```

o/p: Same as push (at 1)

32



ie It inserts elements ahead (first) of list

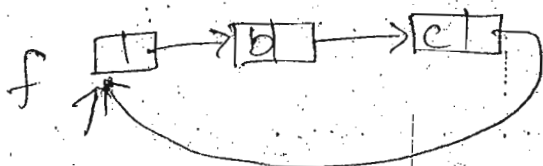
observation:

Linked stack push = insert front of SLL

Disadvantage of SLL:

1. The link part of last node is not utilised
2. stepping backwards is not possible (delete)

Hence SLL $\rightarrow$ circular LL



Advantage of circular LL:

1. the link part of last node is utilised by placing the address of first node
2. stepping backwards is possible by moving forward only in one round

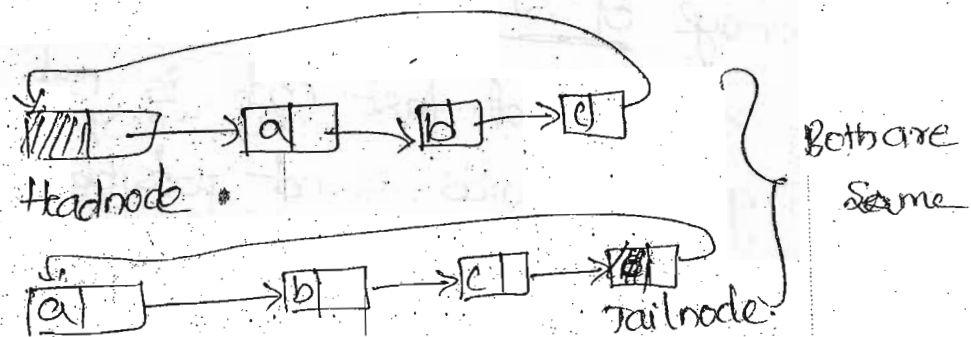
Disadvantage of circular LL:

- Danger of falling into infinite loop

Significance of head node (or) tail node:

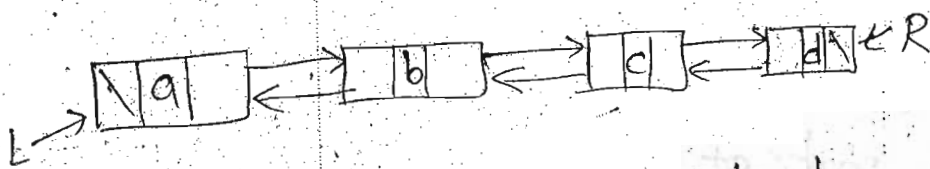
- no danger of falling into infinite loop

- The list can never be empty (atleast head node is present to know where to begin)
- the information part of head node can be utilised (as counter; instead of $O(N)$, $O(1)$)
- Certain operations are easy
 - concatenation ()
 - split ()



8/8/11

Doubly Linked list:



- It is a self referential structure
- ```
struct DNode
```

```
{
 struct DNode *Lptr;
 int data;
```

```
 struct DNode *Rptr;
```

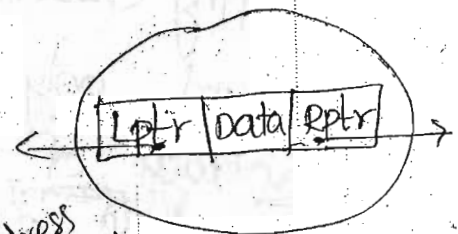
```
};
```

```
struct DNode * L = NULL;
```

```
struct DNode * R = NULL;
```

holds the address of leftmost node

holds the address of rightmost node



// Initialization

## Access:

33

 $t \rightarrow \text{Data}$  $t \rightarrow \text{Lptr}$  $t \rightarrow \text{Rptr}$ Move left: $t = t \rightarrow \text{Lptr}$ Move right: $t = t \rightarrow \text{Rptr}$ 

① How many link fields are getting updated for performing the following operations

|        | Leftmost | Rightmost | else (middle)  |
|--------|----------|-----------|----------------|
| Insert | 3        | 3         | 4 <sup>*</sup> |
| Delete | 1        | 1         | 2              |

 $L \rightarrow \text{NULL}$   
 $R \rightarrow \text{NULL}$  $\leftarrow \text{empty list}$  $\downarrow$   
 $\text{if } (L == \text{NULL})$ 

(OR)

 $\text{if } (R == \text{NULL})$ 

Single node

 $L \rightarrow \boxed{\text{Data} | \text{Lptr} | \text{Rptr}} \leftarrow R$  $\text{if } (L == R)$ 

\*: Insert a node to the right of M, where M is the address of any middle node and the condition is that all the operations should be performed w.r.t. newly inserted node.

most

## Inserting leftmost

void insertleftmost (int x)

{

1) struct Dnode \*n = GetDnode();

2) n → data = x;

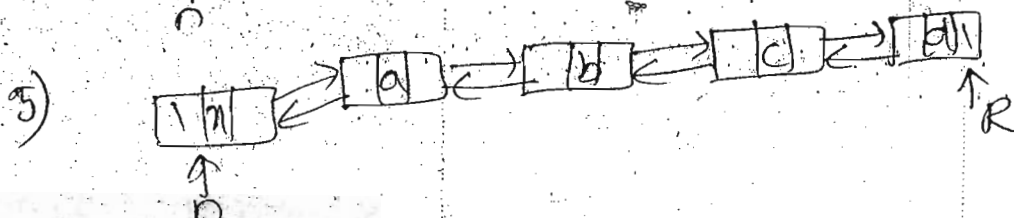
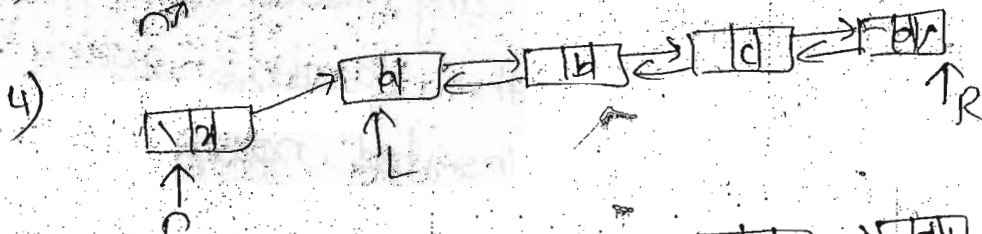
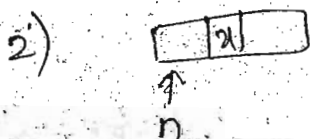
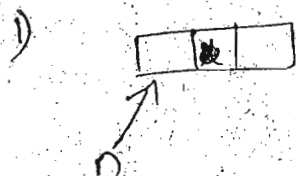
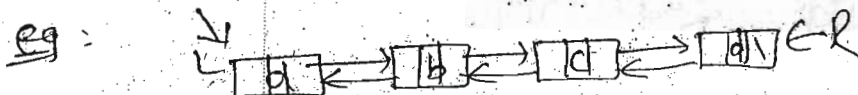
3) n → Lptr = NULL; ①

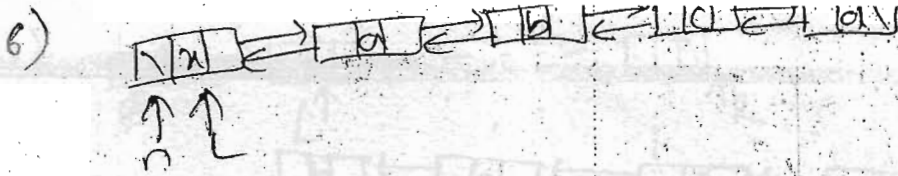
4) n → Rptr = L; ②

5) L → Lptr = n; ③

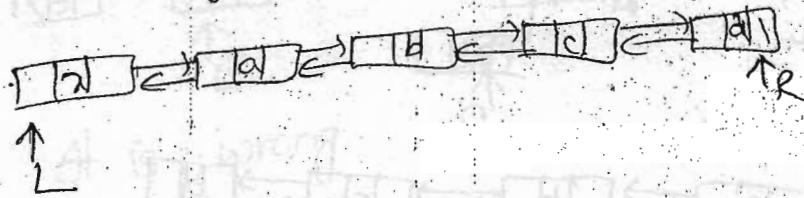
6) L = n;

}

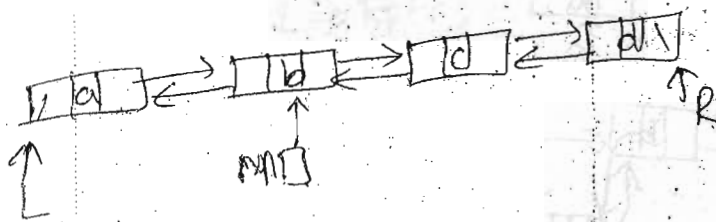




After coming out



Inserting in middle:



void insert right of M (int x)

{

1) struct node \*n = GetNode();

2) n → Data = x;

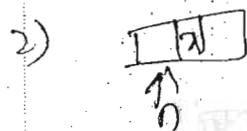
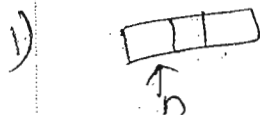
3) n → Lptr = M; ①

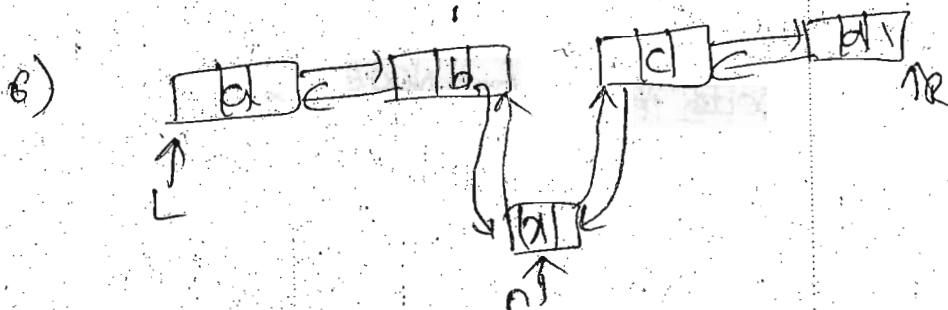
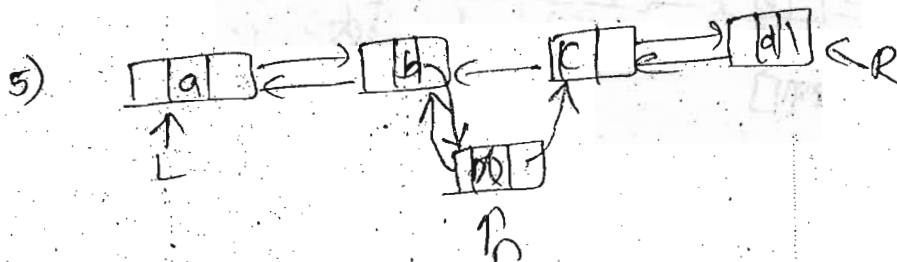
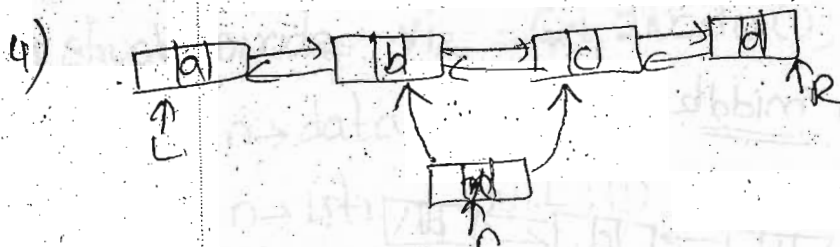
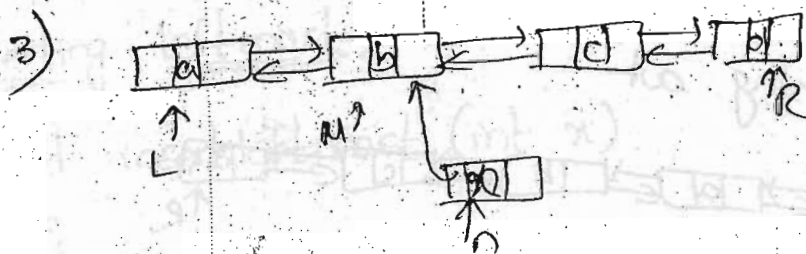
4) n → Rptr = M → Rptr; ②

5)  $\left. \begin{matrix} M \rightarrow Rptr \\ n \rightarrow Lptr \rightarrow Rptr \end{matrix} \right\} = n; \text{ ③}$

6) n → Rptr → Lptr = n; ④

eg:





After coming out

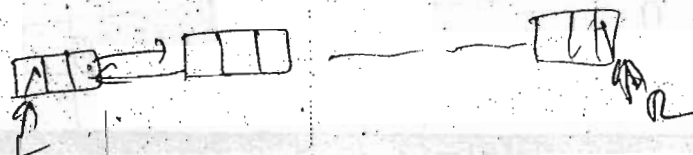


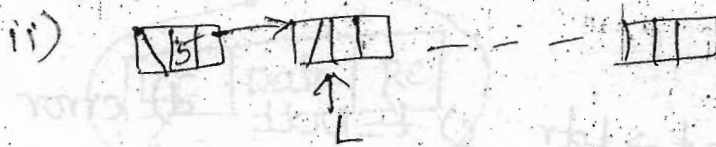
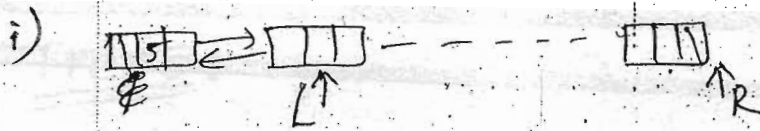
p) Identify whether the following steps delete the left most node

i)  $L = L \rightarrow Rptr;$

ii)  $L \rightarrow Lptr = NULL;$

eg:-





It is wrong.

So the correct steps are

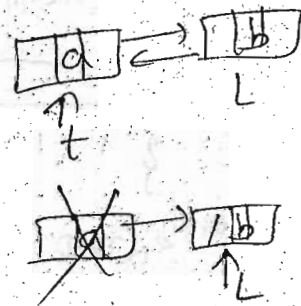
- 1)  $L = L \rightarrow Rptr;$
- 2)  $\text{free}(L \rightarrow Lptr);$
- 3)  $L \rightarrow Lptr = \text{NULL};$

II method:

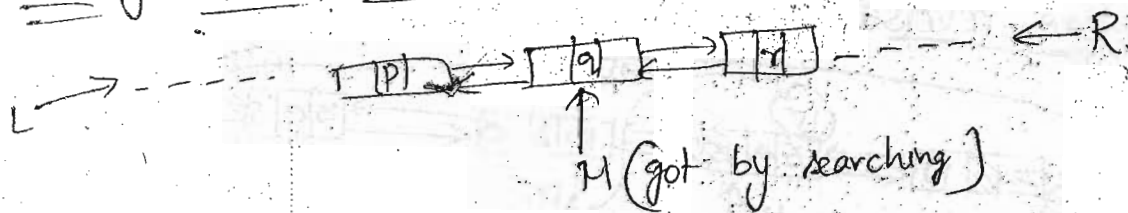
```

struct DNode *t = L;
L = L -> Rptr;
L -> Lptr = NULL;
free(t);

```



Deleting middle element



void deleteMC)

{ 1)  $M \rightarrow Lptr \rightarrow Rptr, M \rightarrow Rptr;$

ii)  $M \rightarrow Rptr \rightarrow Lptr = M \rightarrow Lptr;$

$\text{free}(M);$

}

## Circularly DLL

fill up the blank

a)  $t \rightarrow Rptr$  b)  $t = t \rightarrow Lptr$  c)  $t = \text{NULL}$  d) error

The following code reverses a given circularly DLL

```
void Reverse-CDLL()
```

```
{
```

```
 struct DNode *t = Head->Rptr;
```

```
 while (t != Head)
```

```
{
```

```
 swap (t->Lptr, t->Rptr);
```

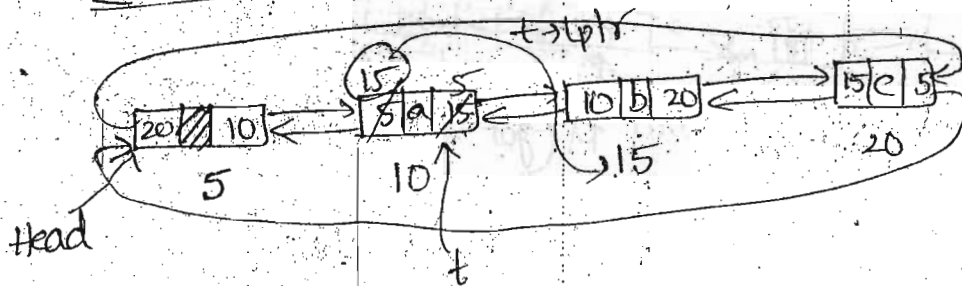
```
 [9];
```

```
}
```

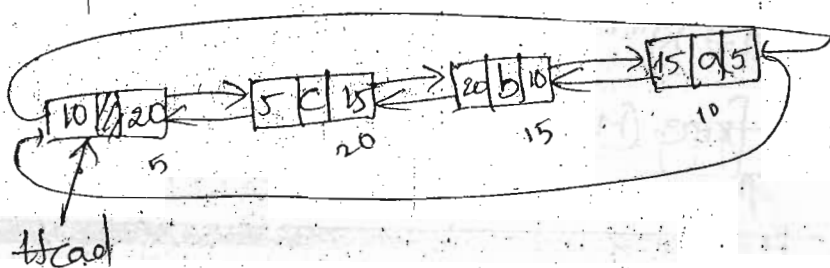
```
 swap (Head->Lptr, Head->Rptr);
```

```
}
```

Before reverse:



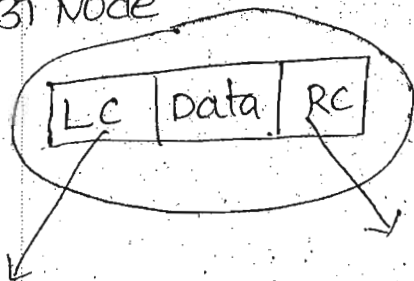
After reverse



## Trees:

36

BT Node



LC = left child

RC = Right child

Rajesh T.K.

struct BTNode

Hence self referential structure

{

struct BTNode \*LC;

int data;

struct BTNode \*RC;

};

struct BTNode \*t = Null; // Initialisation

↑  
Root

Access:

t → data

t → RC

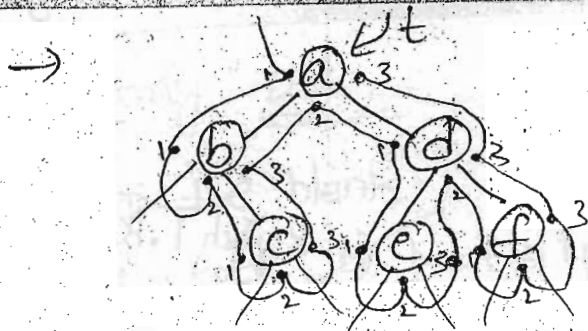
t → LC

Descend Left:

t = t → LC

Descend Right:

t = t → RC



### Traversals:

Inorder: Left, Root, Right  
 (1) (2) (3)

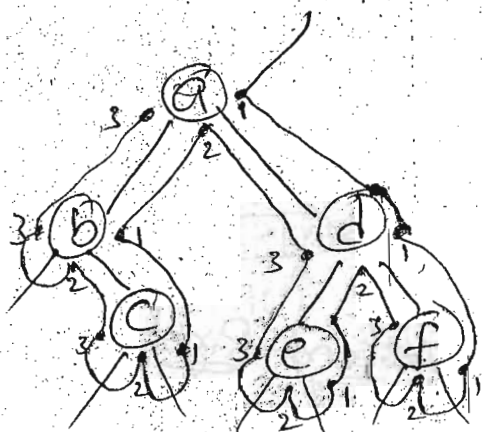
b c a e d f

Postorder: Left, Right, Root  
 (3) (2) (1)

c b e f d a

Pre order: Root, Left, Right  
 (1) (2) (3)

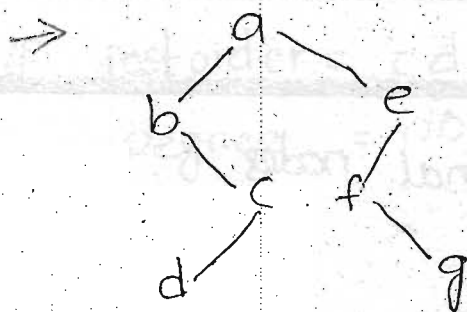
a b c d e f



Converse Inorder: Right, Root, Left  
 f d e a c b

Converse Postorder: f e d c b a Right, Left, Root

Converse Pre order: Root, Right, Left → a d f e c b



Inorder: b d c a f g e

postorder: d c b g f e a

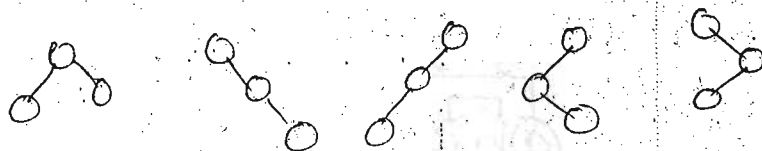
pre order: a b c d e f g

converse Inorder: e g f a c d b

converse postorder: g f e d c b a

converse preorder: a e f g b c d

Q) for a given unlabelled tree with 3 nodes, how many distinct BTs are there?



∴ 5 BT

for 'n' nodes  $\Rightarrow \frac{2^n C_n}{n+1}$

$$n=3 \Rightarrow \frac{{}^6C_3}{4} = \frac{6 \times 5 \times 4}{4 \times 6} = 5$$

$$n=5 \Rightarrow \frac{{}^{10}C_5}{6} = \frac{10 \times 9 \times 8 \times 7 \times 6}{6 \times 120} = 42$$

Q) for a given preorder: abc, how many BTs are there.

$$\frac{2^n C_n}{n+1}$$

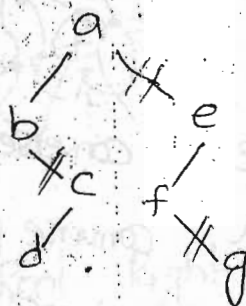
Q) what is the post order for the given preorder:

preorder = a b c d e f

TAG = - - 1 - 1 1

TAG indicates terminal nodes

Eg



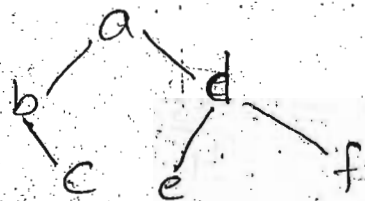
Rptr  
pre = a b c d e f g

TAG = - - - 1 - - 1

Given problem

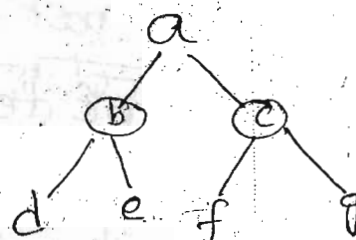
Rptr  
preorder = a b c d e f

TAG = - - - 1 - 1 1



① Given post order = d e b f g c a  
degree = 0 0 2 0 0 2 2  
↑  
outdegree

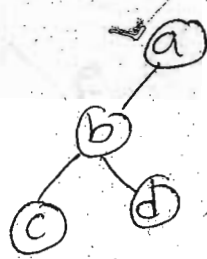
d e b f g c (a)



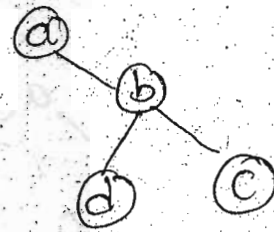
① postorder = c d b a

38

Degree = 0 0 2 1



(or)



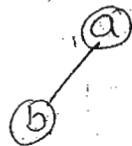
ob: when degree is '1' we cannot distinguish Left (or) Right child

12/8/14

① preorder = a b

postorder = b a

Generate BT



(or)



ob: To have unique pattern, we need Inorder + preorder (or) Inorder + postorder

①

pre = abcdef

In = bcaedf

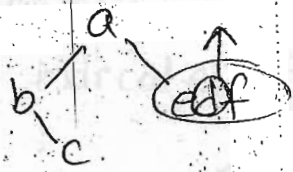
concept:

- consider Inorder
- Look @ preorder to find next node

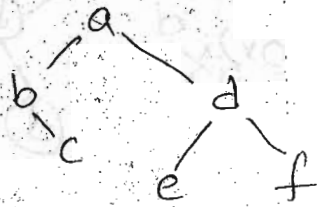
bcaedf

preorder: abcdef

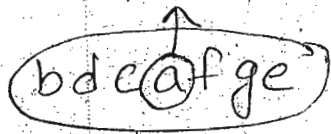




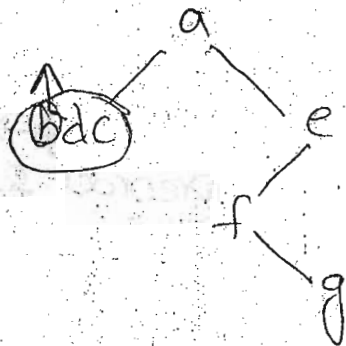
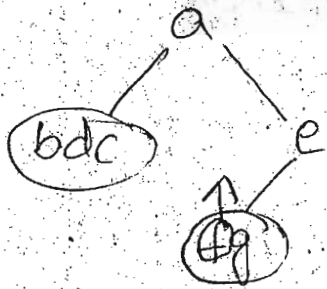
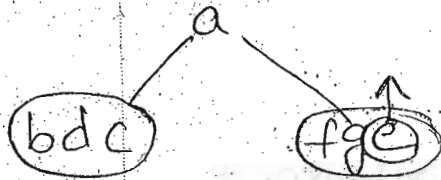
preorder:  $\rightarrow$  a b c d e f

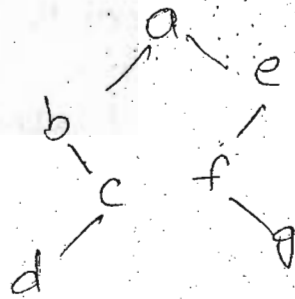
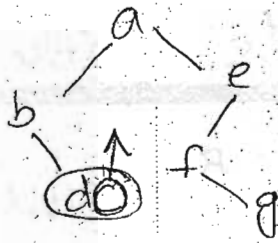


(P) post order = d c b g f e a  
inorder = b d c a f g e

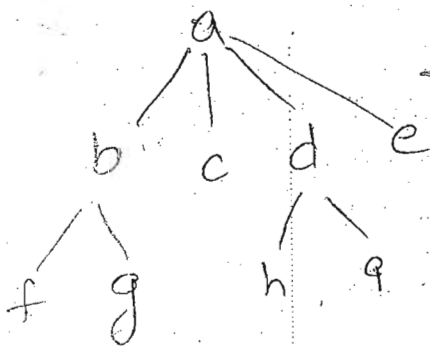


post:  $\leftarrow$  d c b g f e a



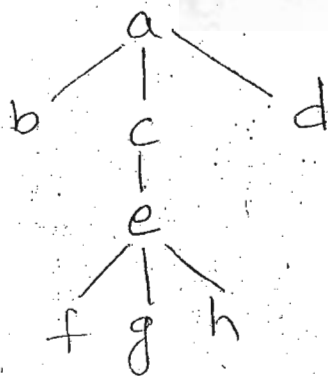


General tree representation: parenthesis representation



$a(b(f,g), c, d(h,i), e)$

\*)  $a(bc(e(fgh))d)$



P)  $N(H)$  = Max, no of nodes in a BT  
 with recursion

$$N(H)_{\max} = 1, \quad L=H=0 \rightarrow \text{root is at level } 0$$

$$= N(H-1) + 2^H, \quad (L=H) > 0, \quad \begin{matrix} L \rightarrow \text{Level} \\ H \rightarrow \text{Height} \end{matrix}$$

without recursion

$$N(H)_{\max} = ?$$

Sol:

$$L=H=0 \Rightarrow 0$$

$$N(0) = 1$$

$$L=H=0 \Rightarrow$$

$$L=H=1 \Rightarrow$$

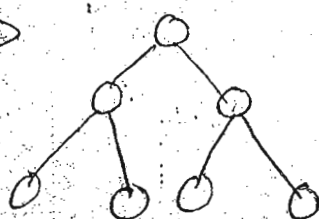


$$N(1) = 3 = 1 + 2^1$$

$$L=H=0 \Rightarrow$$

$$L=H=1$$

$$L=H=2$$



$$N(2) = 7$$

$$= 3 + 4$$

$$= N(1) + 2^2$$

$$\therefore N(H) = N(H-1) + 2^H$$

without recursion:

| L=H | N(H) |
|-----|------|
| 0   | 1    |
| 1   | 3    |
| 2   | 7    |
| 3   | 15   |

$$\rightarrow 2^3 - 1 = 2^{H+1} - 1$$

$$N(H)_{\max} = 2^{H+1} - 1$$

40

at level  $\Rightarrow$

(P)  $N(H) = \text{Max, no. of nodes in a BT}$

rel

height

with recursion

$$N(H)_{\max} = 1, \quad L = H = 1$$

$$= N(H-1) + ? \quad (L = H) > 1$$

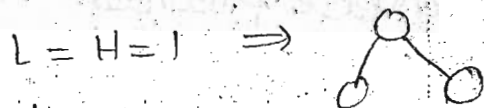
without recursion

$$N(H)_{\max} = ?$$

Sol:

$$L = H = 1 \Rightarrow 0$$

$$N(1) = 1$$



$$L = H = 2$$

$$N(2) = 1 + 2$$

$$= N(1) + 2^{H-1}$$

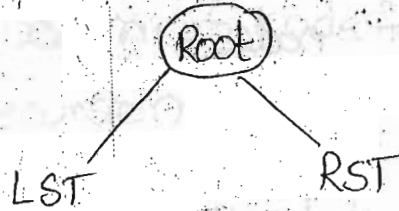
without recursion:

| $L = H$ | $N(H)$ |
|---------|--------|
| 1       | 1      |
| 2       | 3      |
| 3       | 7      |
| 4       | 15     |

$$\Rightarrow 2^3 - 1 = 2^H - 1$$

$$N(H)_{\max} = 2^H - 1$$

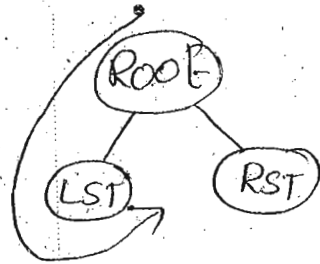
## Recursive algorithms for traversals:



LST - left subtree  
RST - Right subtree

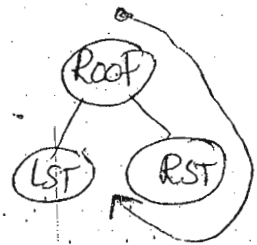
### preorder:

1. Visit root
2. Traverse LST in preorder
3. Traverse RST in ~~preorder~~ preorder



### converse preorder:

1. visit root
2. Traverse RST in converse preorder
3. Traverse LST in converse preorder



### Inorder:

1. LST
2. Root
3. RST

### converse Inorder:

1. RST
2. Root
3. LST

### postorder:

1. LST
2. RST
3. Root

### converse postorder:

1. RST
2. LST
3. Root

coding code:

41

$$q_f(t).$$

5

1. Inorder ( $t \rightarrow Lc$ ),

2. `Printf (" %d", t → data);`

3. Inorder (L  $\rightarrow$  R):

3.

3

p) what does `DO()` do on following BT

```
void DO(struct BNode *t)
```

§ if (4)

3

1. DO ( $t \rightarrow LC$ );

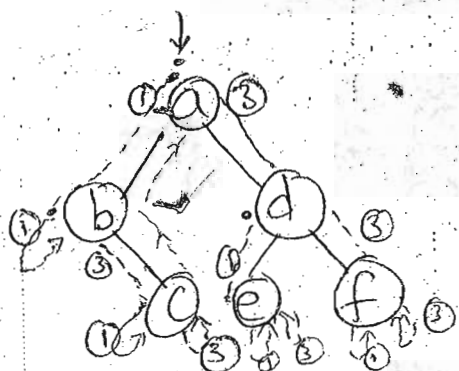
2. pf ("x.d", t → data);

3.  $DO(t \rightarrow RC)$ ;

```
4. pf ("%d", t → data);
```

3

3



o/p: bccbaeedffda

DO

1. LC

2. Pf

3. RC

4. Pf

Q) what does TO() do on BT?

```
void TO (struct Bnode *t)
```

```
{
```

```
 if (t)
```

```
 {
```

```
 1. printf ("%d", t->data);
```

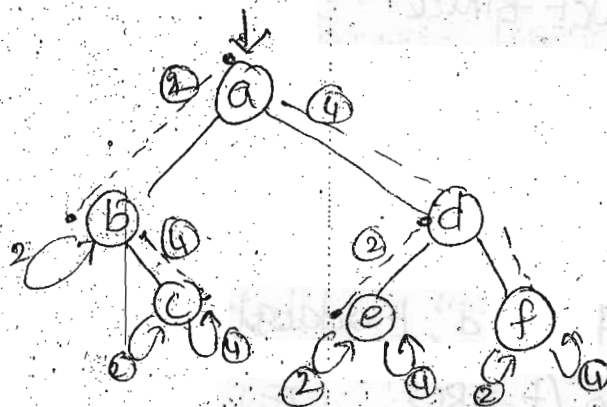
```
 2. TO (t->LC);
```

```
 3. printf ("%d", t->data);
```

```
 4. TO (t->RC);
```

```
 5. printf ("%d", t->data);
```

```
 }
}
```



TO

1. pf

2. RC

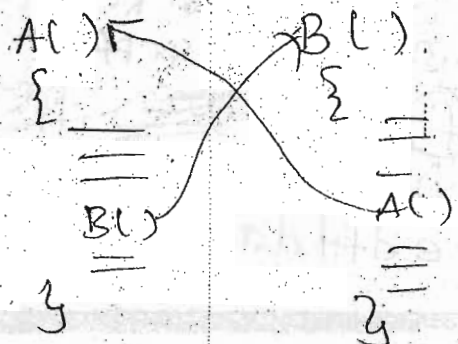
3. pf

4. RC

5. pf

o/p: abbcccbadeeedfffd a

Indirect recursion:



p) consider following routines

void A (struct BTnode \*t)

```

{
 if (t)
 {
 printf ("%d", t->data);
 B(t->LC);
 B(t->RC);
 }
}

```

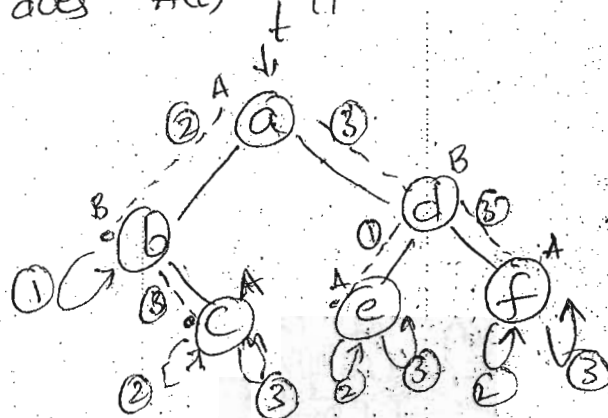
void B (struct BTnode \*t)

```

{
 if (t)
 {
 A(t->LC);
 printf ("%d", t->data);
 A(t->RC);
 }
}

```

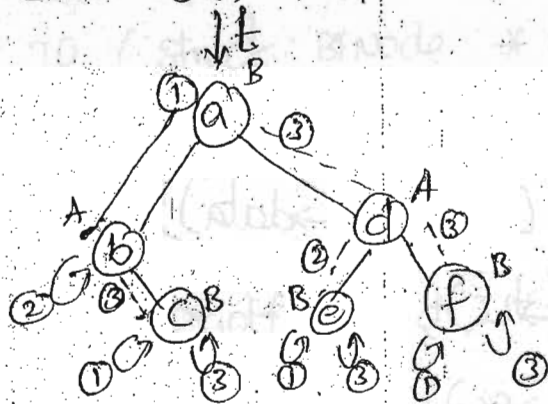
1) what does A(t) d/p on following BT



d/p abc edf

| <u>A</u> | <u>B</u> |
|----------|----------|
| 1. Pf    | 1. A(LC) |
| 2. B(LC) | 2. Pf    |
| 3. B(RC) | 3. A(RC) |

2) what does  $BT(t)$  dp on BT



op: bcadef

15/8/11  
09G  
(P)

p) int DO (struct BTnode \*t)

{ if (!t)

return 0;

2) int xL = DO (t->LC);

3) int xR = DO (t->RC);

4) if (xL > xR)

return = xL + 1;

else

return = xR + 1;

}

what DO() return on a BT?

Sol:

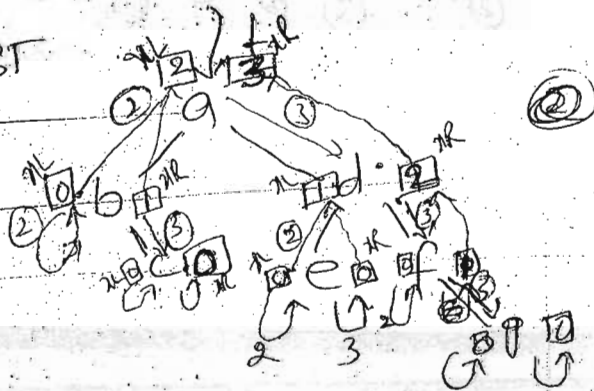
Consider

L=1

L=2

L=3

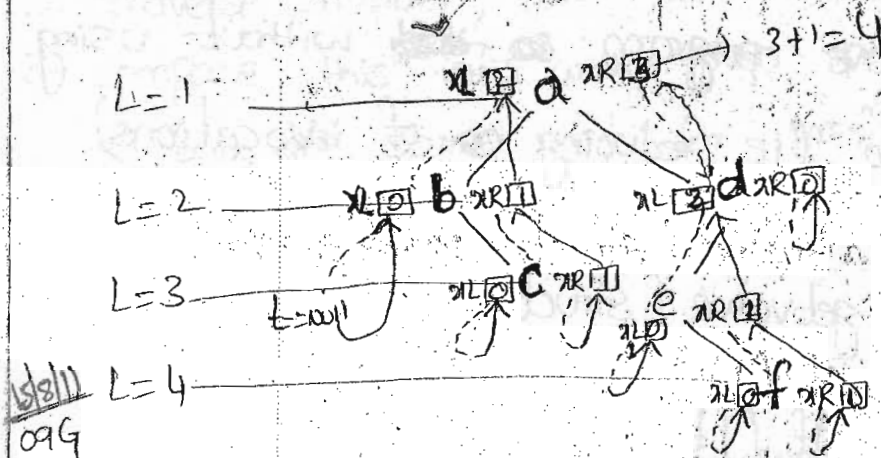
L=4



op: 4

Sol:

It returns the height / level / depth of tree with root at level 1. 43.



Q) what does getvalue() returns on BT?

int getvalue ( struct. BTnode \*t)

{  
1) int value = 0;

2) if (t)

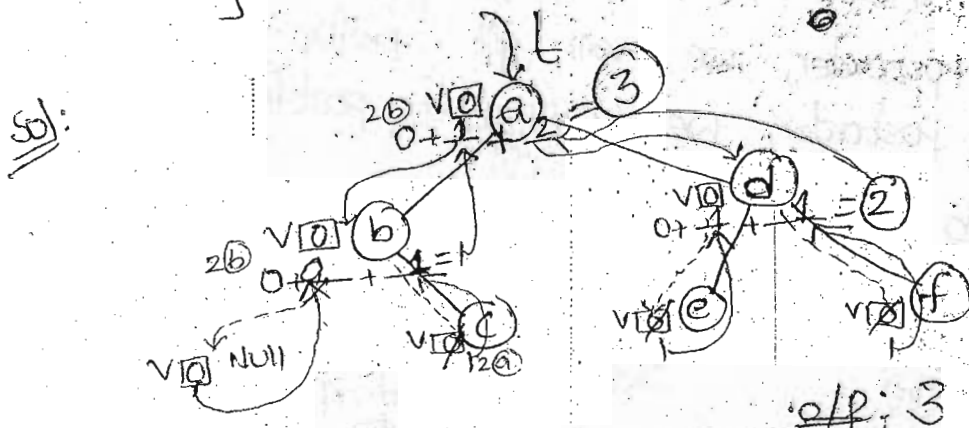
{  
a) if ((t->LC == NULL) && (t->RC == NULL))

value = 1;

else

b) value = value + getvalue(t->LC) + getvalue(t->RC);  
}

3) return value;  
}



o/p: 3

It counts terminal nodes.

p) modify above program ~~so that~~ without using local variable (ie reducing no. of invocations)

Sol:

```
int getvalue (struct BNode *t)
{
 if (!t)
 return 0;
 if ((t->LC == NULL) && (t->RC == NULL))
 return 1;
 else
 return getvalue(t->LC) + getvalue(t->RC);
}
```

Applications of trees:

a) Expression (or) Arithmetic trees:

operator = root/parent level

operand = leaf/children level

Significance:

- 1) Traverse inorder, we will get Infix
- 2) Traverse preorder, we will get prefix
- 3) Traverse postorder, we will get post-fix

Eg:  $a+b$



Step

1. G

2. T

3. C

4. F

Eg:

I

Eg:

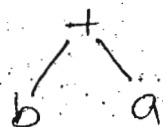
p)

S

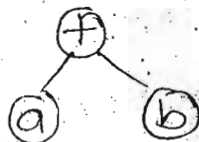
Steps:

1. Generate BT
  2. Traverse Inorder, get Infix
  3. compare this infix with given infix
- If both are same, go for prefix, postfix

Eg:  $a+b$



Inorder  $\rightarrow$  Infix =  $b+a$  ↖ Not same  
 given Infix =  $a+b$  ↘



Inorder - Infix =  $a+b$  ↖ same  
 'given' =  $a+b$  ↘

Hence prefix =  $+ab$  (preorder)  
 postfix =  $ab+$  (postorder)

Eg:

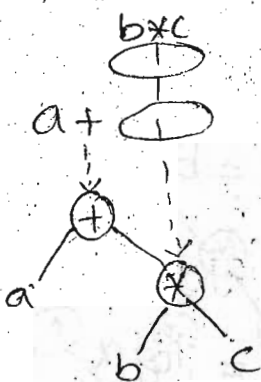
p)

$a+b \times c$

precedence:

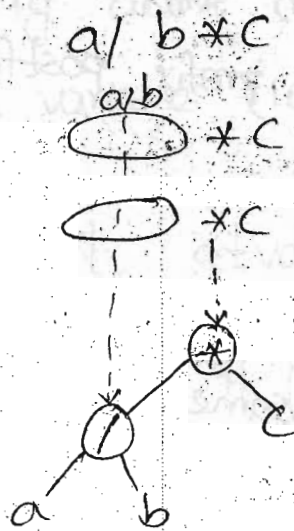
Sol:

$a+b \times c$

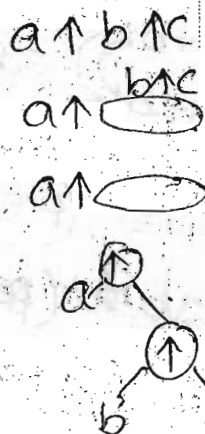


Infix =  $a+b \times c$   
 prefix =  $+a \times bc$

p) Left Associative:



p) Right associative:



Note: Lower precedence operator is at root

p) -construct expression trees for

a)  $-b$



prefix =  $-b$

postfix =  $b -$

b)  $\log x$



prefix =  $\log x$

postfix =  $x \log$

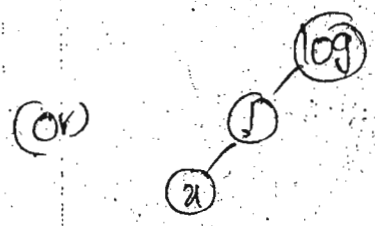
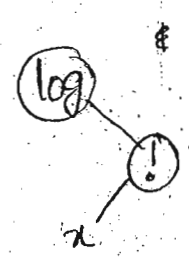
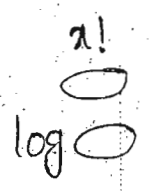
c)  $x!$

prefix =  $!x$

postfix =  $x!$



d)  $\log x!$



(or)

wrong

Inorder =  $x! \log$  ← Infix

given Infix =  $\log x!$  ← different

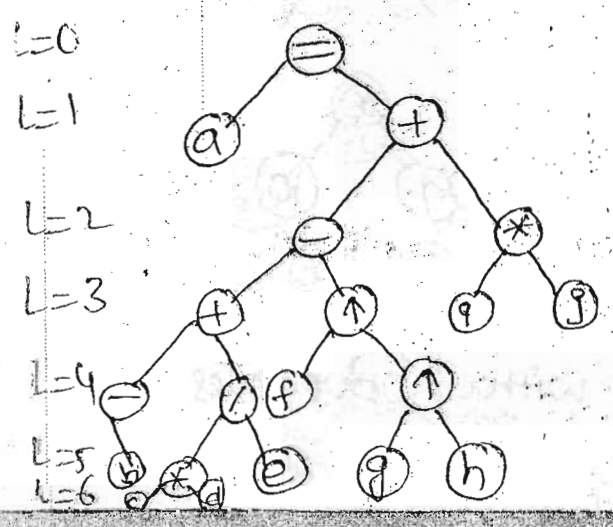
e) consider

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

1) what is @ root

= (lower precedence)

2) How many levels are there in the expression tree if the root is at level = 0



precedence

- unary

$\uparrow$ : Right associative

$\times$ : left associative

$+-$ : left associative

Height = 6  
(Counting edges from leaf)

## Binary Search Trees: (BST)

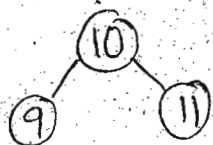
— Also called dictionary trees (or) lexically ordered Binary trees.

Definition:

→ Binary tree with



Eg:



Significance:

If we traverse inorder, we will get sorted list

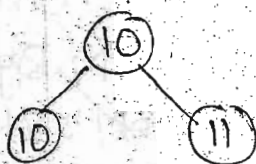
→ with duplicate

we need to decide first

left biased



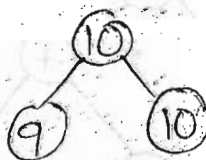
Eg:



Right Biased



Eg:



Note:

Default BST are without duplicates

## creation of BST:

46

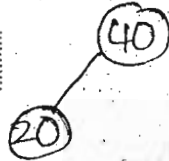
1. very first node is root node
2. start comparing from root node and insert the subsequent nodes

Eg:

create BST with

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

→ 40

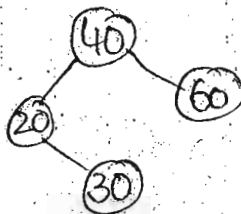


→ 20

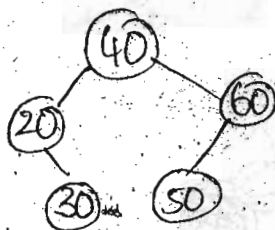
→ 30



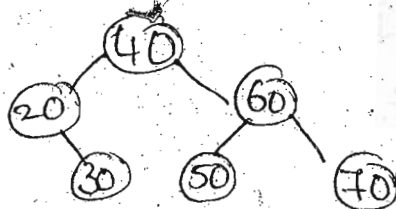
→ 60



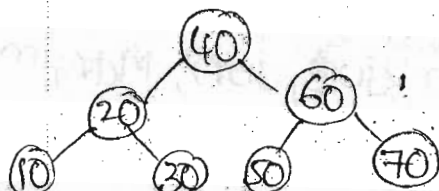
→ 50



→ 70



→ 10



Inorder:

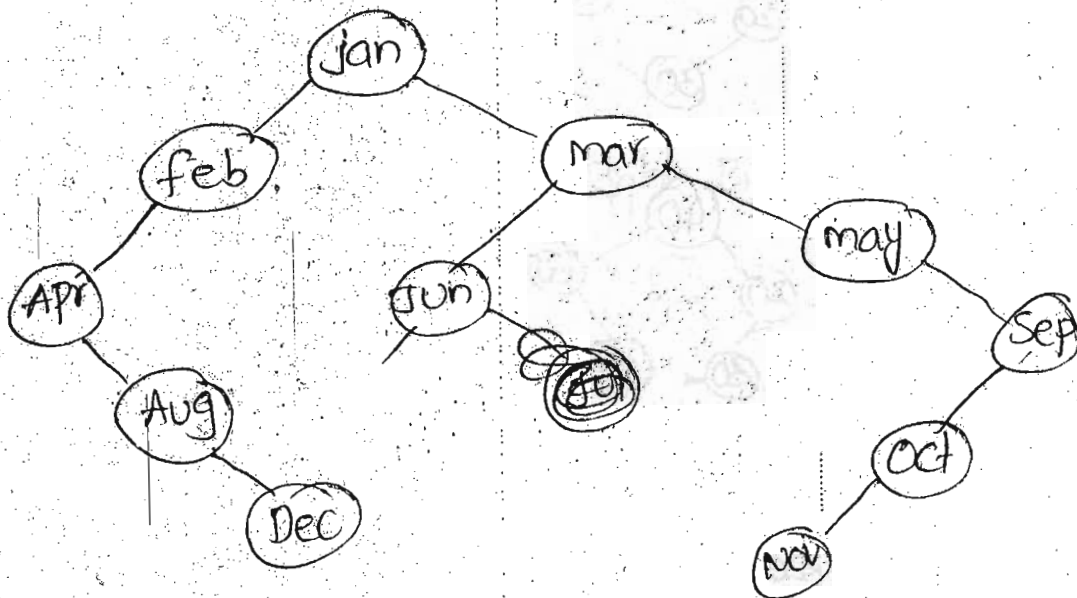
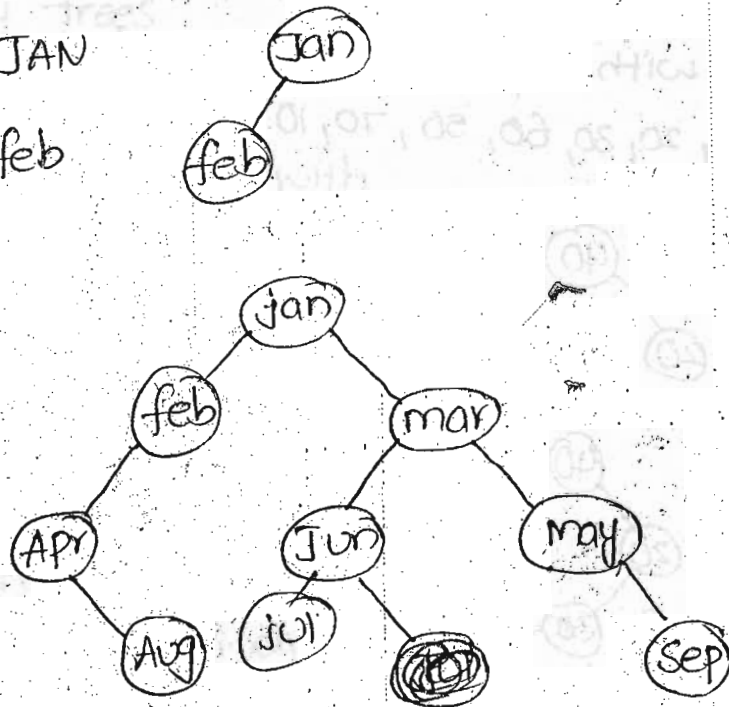
sorted list: 10, 20, 30, 40, 50, 60, 70

p) create a BST for months of year (consider first 3 letters of a month)

Sol:

JAN

feb



Inorder:

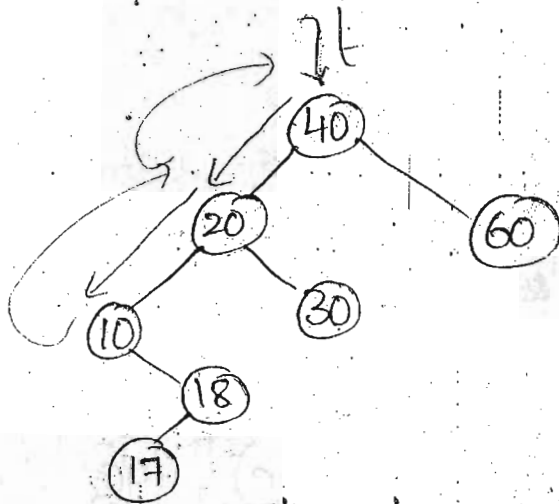
Apr, Aug, Dec, feb, jan, Jul, Jun, Mar, may, Nov, Oct, Sep

first

```

P) int do (struct BTnode *t)
 {
 if (!t->LC)
 return (t->data);
 else
 return do(t->LC);
 }

```

Sol:

It returns the minimum element in BST

obs: To get maximum element (greatest element) traverse descend right.

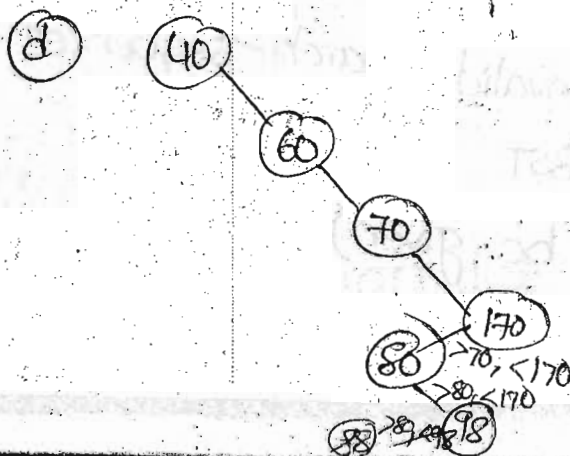
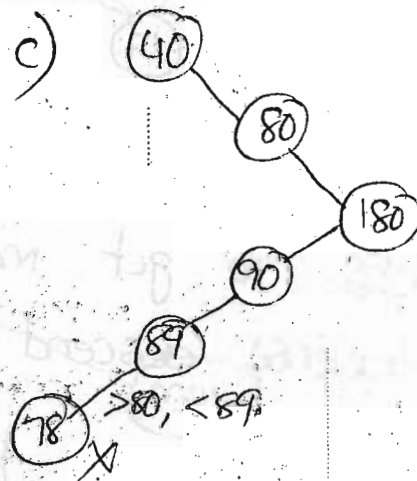
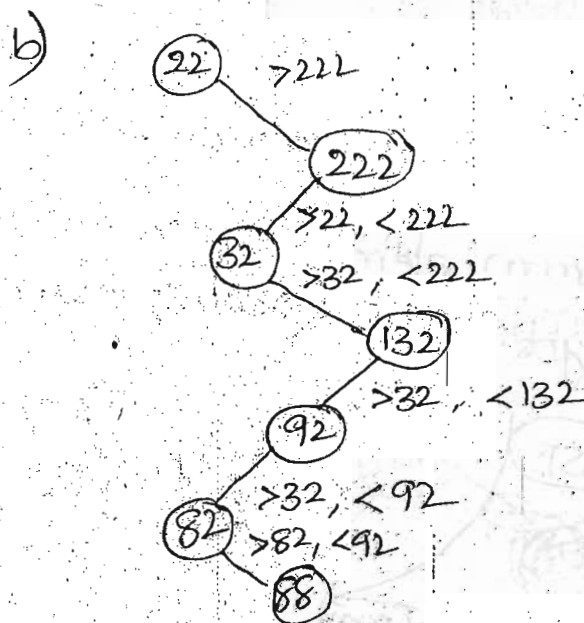
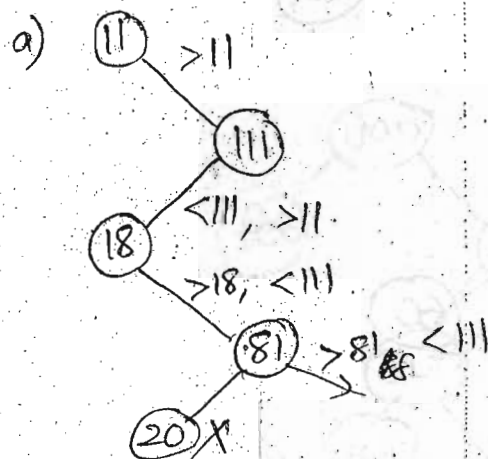


P) Identify valid/Invalid search sequences to search for 88 on a BST  
(BST will not be given)

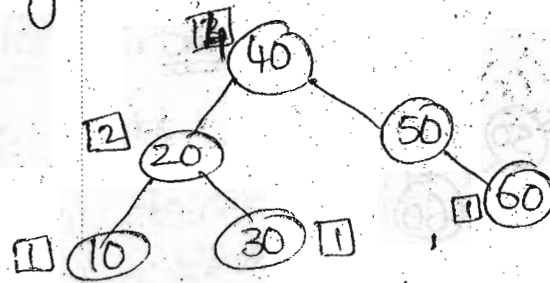
Sep

Oct  
Sep

- a) 11, 111, 18, 81, 20, 90, 60, 70, 88 - Invalid
- b) 22, 222, 32, 132, 92, 82, 88 - valid
- c) 40, 80, 180, 90, 89, 18, 88 - Invalid
- d) 40, 60, 70, 170, 80, 98, 88 - valid



Q what is the Index (rank) of node 50 in the following BST



Sol: Index = No. of nodes in left subtree + 1

$\therefore$  Index of node 50 = 0 + 1 = 1

Note:  
Index indicates the position of the node in the inorder (of root) inorder

Deletion of node from BST:

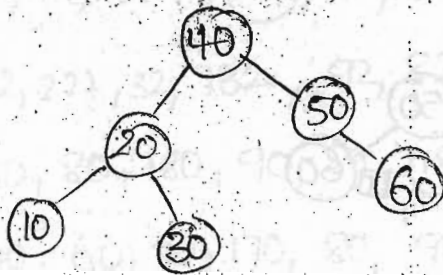
Even after the deletion, the order must be intact  
ie remains unchanged

ie tree can be modified but not order

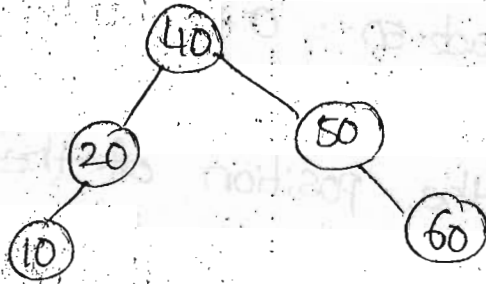
procedure:

| A node to be deleted                          | Action                                                                                                                                   |
|-----------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------|
| <u>case(i):</u> no children                   | i) simply delete                                                                                                                         |
| <u>case(ii):</u> only one child (or) subtree  | ii) connect to its grand parent                                                                                                          |
| <u>case(iii):</u> Both children (or) subtrees | Replace with<br>a) Inorder successor (or) min. of Right subtree (RST) } default<br>b) Inorder predecessor (or) max of Left subtree (LST) |

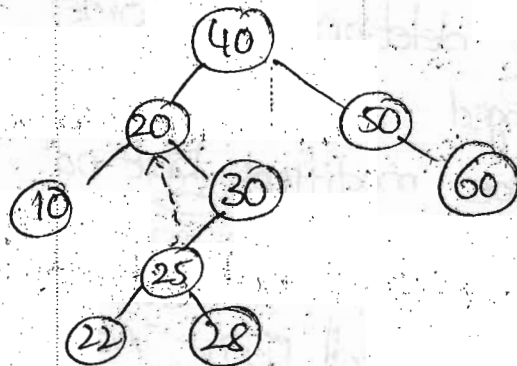
Eg:



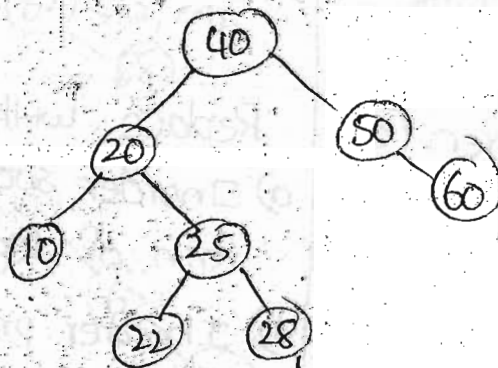
case(i): Delete 30



case(ii): ~~do~~ ~~do~~ consider



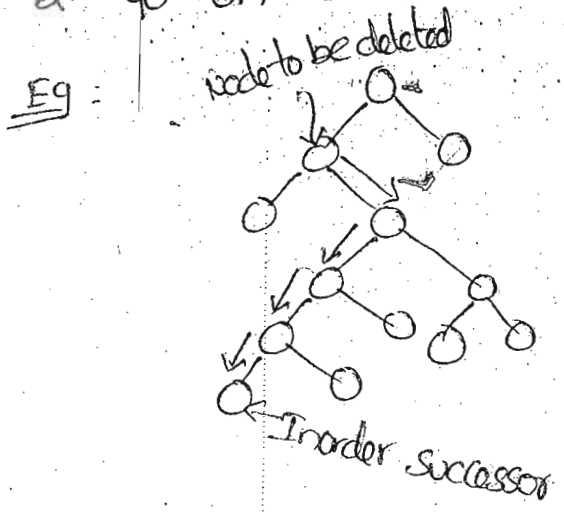
Delete 30



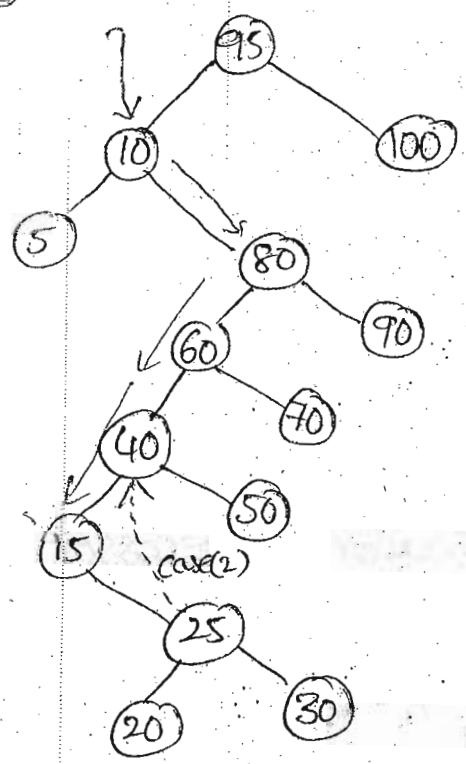
Case (iii):

Logic for Inorder Successor:

1. Jump right
2. Go on descend Left

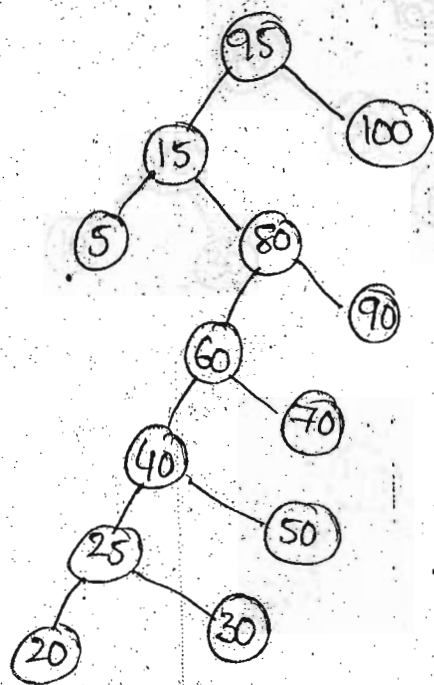


Eg:



Delete 10.

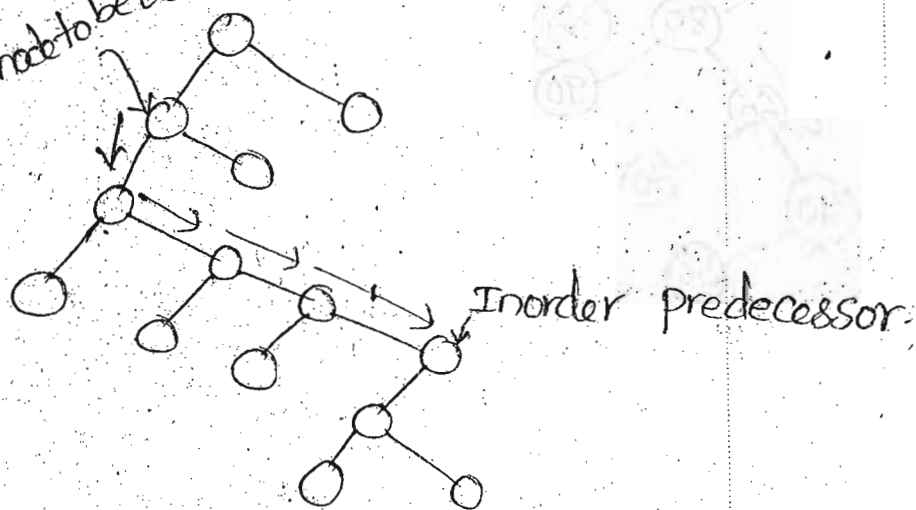
Inorder Successor for '10' is '15'



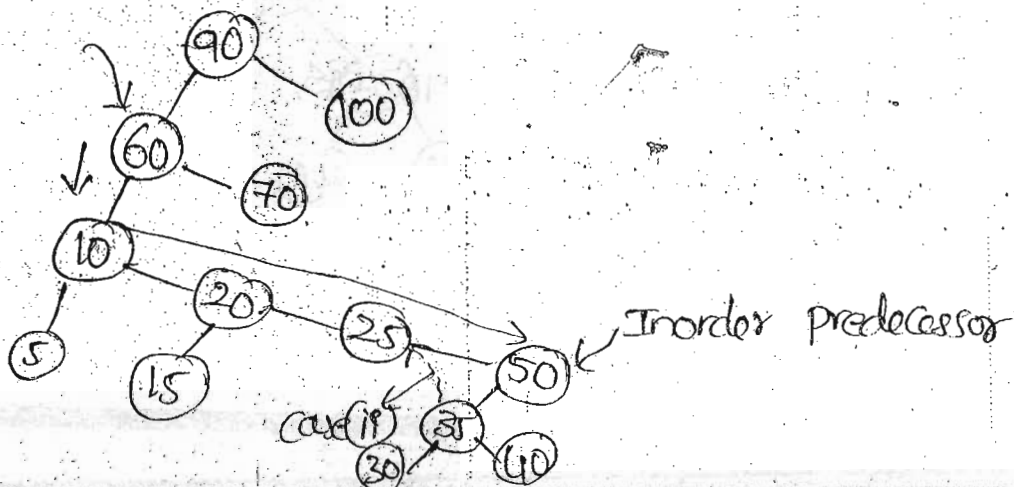
Logic for Inorder predecessor:

1. Jump left
2. Go on descend right

Eg: node to be deleted

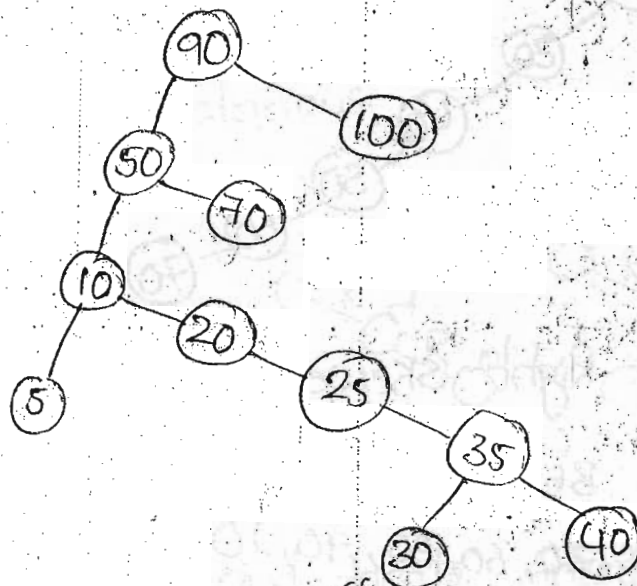


Eg:



Delete 60

Sol: Inorder predecessor for node 60 is node 50.



AVL Search Trees: (Krusse)

→ devised by Adelson Velski } Russian mathematicians  
Landis } (1962)

→ Height Balanced Trees

Definition:

BST with

balance factor

$$bf(n) = HL - HR = \begin{cases} +1 \\ 0 \\ -1 \end{cases}$$

HL - Height of left subtree

HR - Height of right subtree

→  $bf(n) = \pm 2 \leftarrow$  violated node = Ancestor

Need to go for AVL trees:

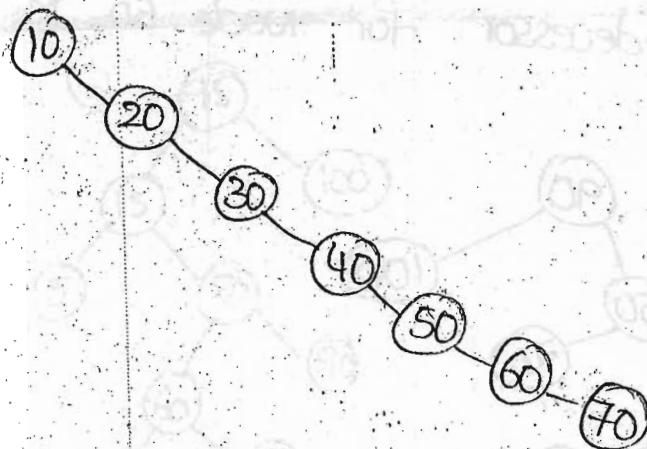
create BST

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

Sol: Search for 70 takes 7 attempts

for 7 nodes  $\rightarrow$  7 attempts

for 'n' nodes  $\rightarrow O(N)$  attempts

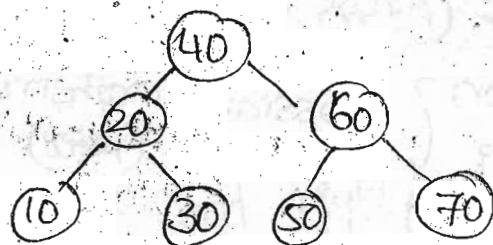


Height = Max. depth

Right skewed BST

Eg: create BST

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



for 70 node  $\rightarrow$  3 attempts

for 7 nodes  $\Rightarrow$  3 attempts

$$= \log_2 7 \approx \log_2 8 = 3$$

- so skewed BST becomes bushy BST (full)  
by rotations

Goal of AVL Trees:

- minimise search times

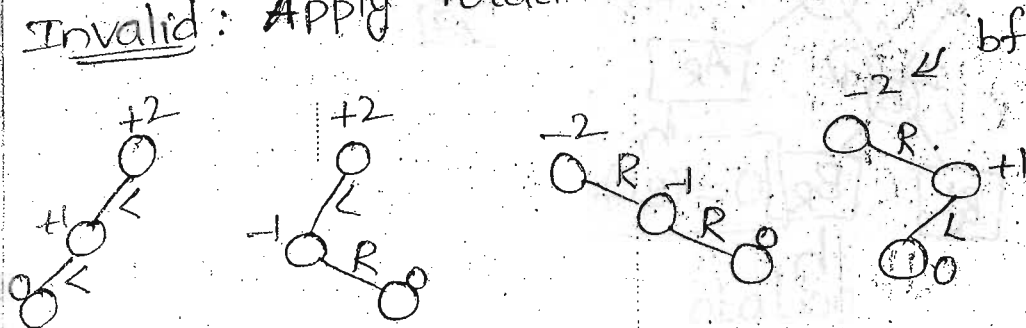
$$O(N) \rightarrow O(\log N)$$

- It can be achieved by using rotations.

valid: no need to apply rotations.



Invalid: Apply rotations.



Nomenclature:

1. Insert a node and locate Ancestor(A)
2. Traverse from (A), towards the newly inserted node
3. consider only TWO hops ~~leaves~~ from (A) to give nomenclature of LL, LR, RR, RL.

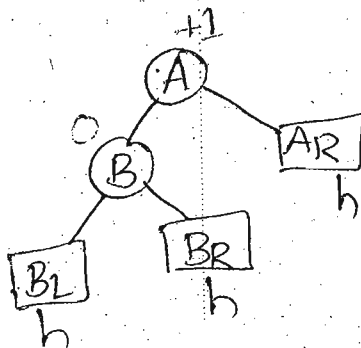
Rotations:

① LL Rotation: single rotation

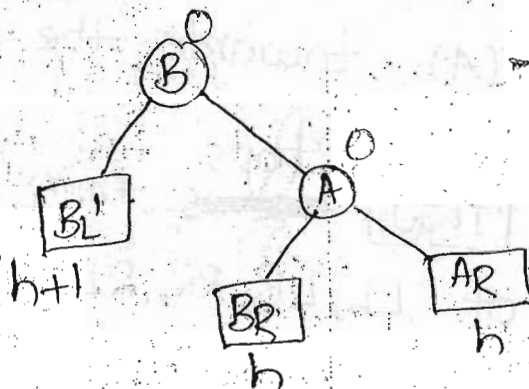
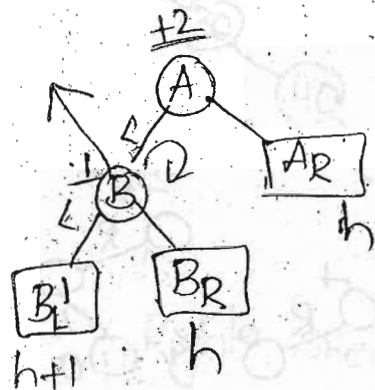
○ node

□ subtree

a) Initial



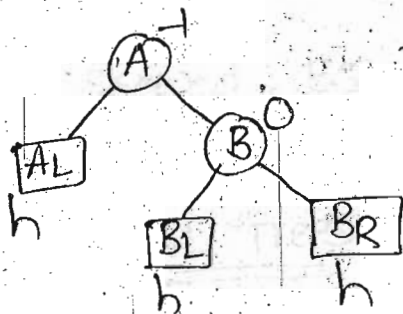
⑥ Insert a node in  $BL$



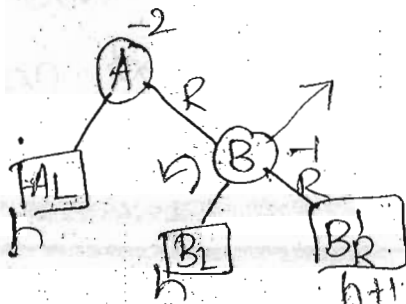
$$\therefore bf(A) = bf(B) = 0$$

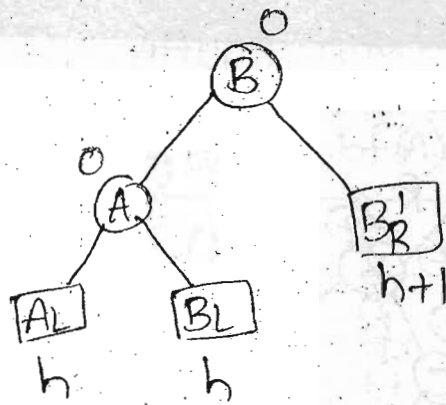
② RR rotation: Single rotation

① Initial



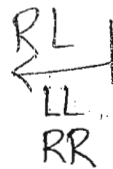
⑥ Insert a node in  $BR$



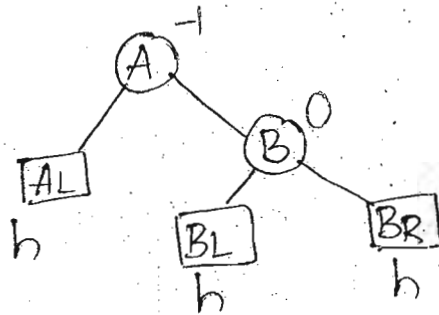


$$bf(A) = bf(B) = 0$$

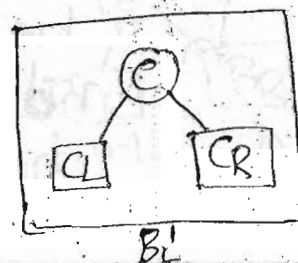
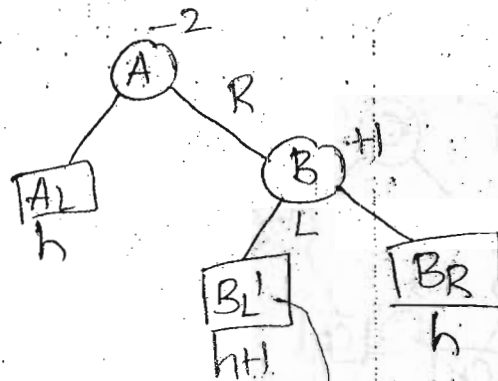
③ RL rotation: double rotation



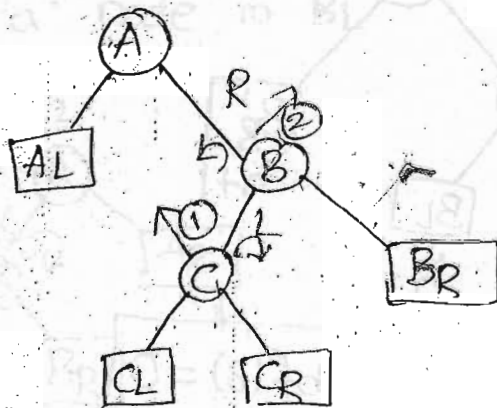
① Initial



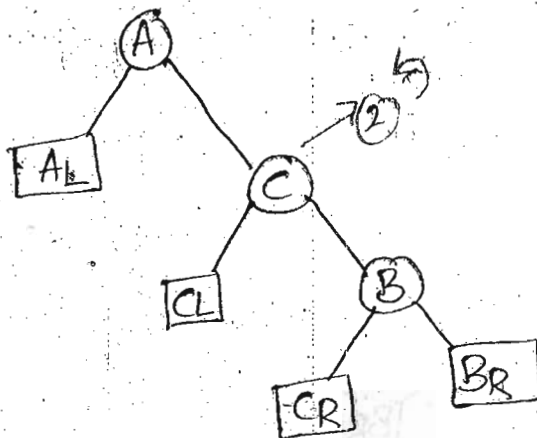
② Insert a node in  $B_L$



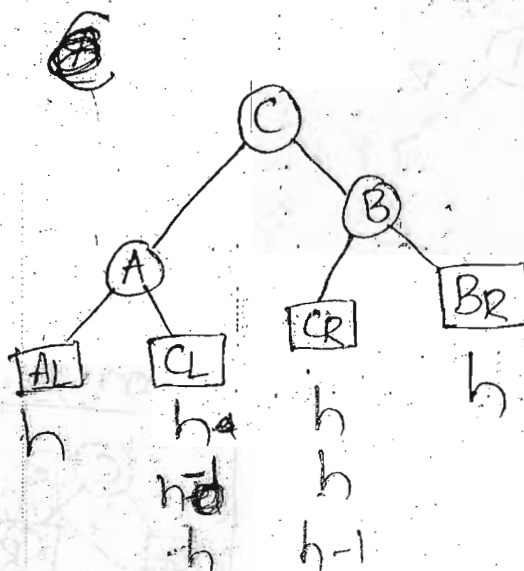
i-c.



LL



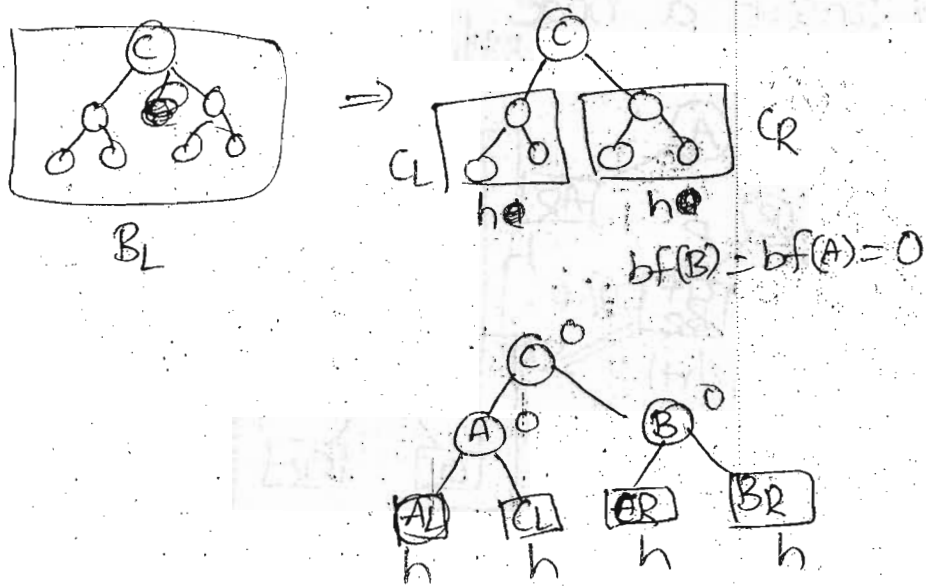
RR



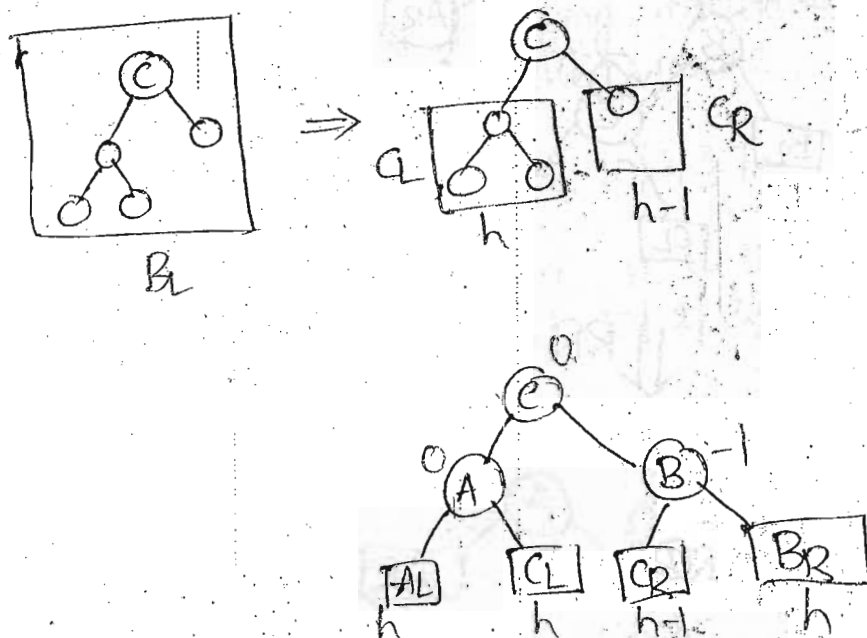
| Before rotation | After rotation |         |
|-----------------|----------------|---------|
| $bf(C)$         | $bf(B)$        | $bf(A)$ |
| 0               | 0              | 0       |
| +1              | -1             | 0       |
| -1              | 0              | +1      |

}  $bf(C)=0$

Sol: 1)  $bf(C)=0$

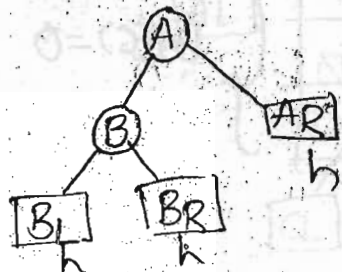


2)  $bf(C)=1$



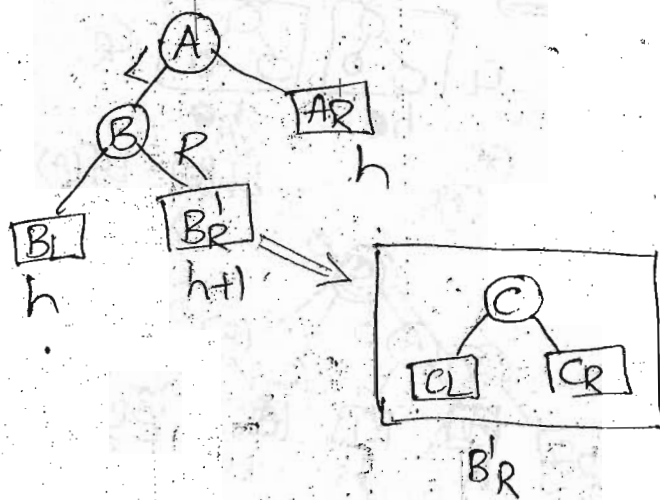
# 4) LR Rotation: Double rotation

a) Initial

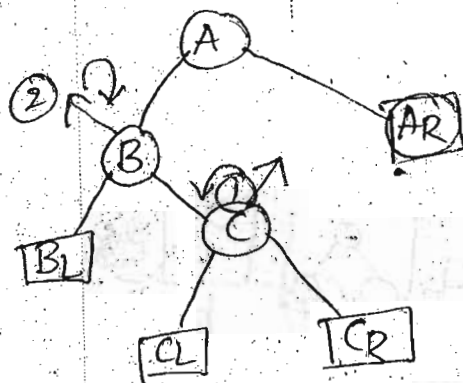


LR  
RR  
LL

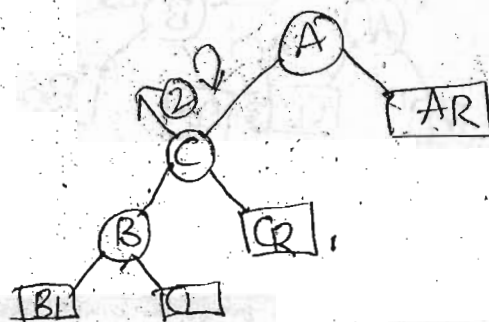
b) Insert a node in BR



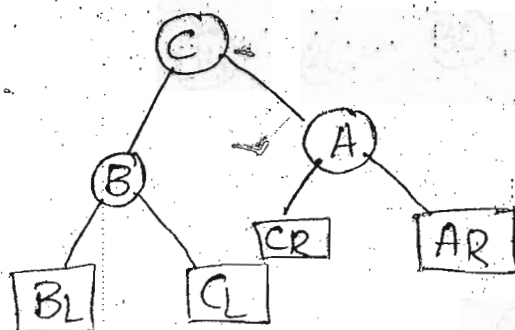
ie.



RR



LL



| Before rotation | After rotation<br>→ |       |
|-----------------|---------------------|-------|
| bf(C)           | bf(B)               | bf(A) |
| 0               | 0                   | 0     |
| +1              | 0                   | -1    |
| -1              | +1                  | 0     |

bf(C) = 0

16/8/11

### Creation of AVL Tree:

Procedure:

1. Insert a node as per BST
2. configure if imbalance

balance it by applying rotations with the nodes

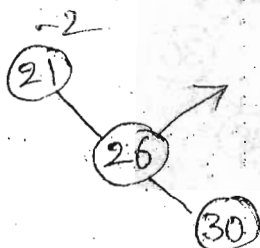
p) create AVL Tree

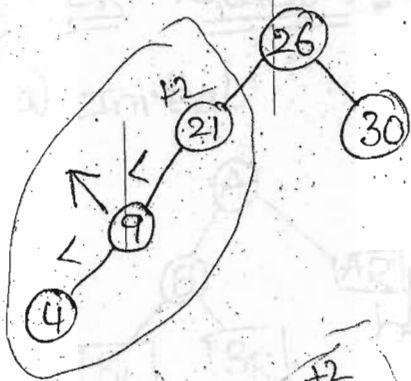
21, 26, 30, 9, 4, 14, 28, 18, 15, 10, 2, 3, 7

So:

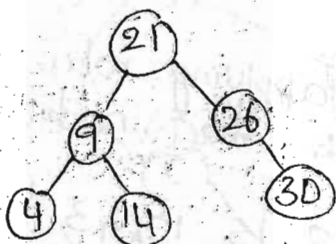
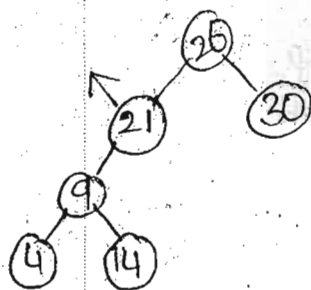
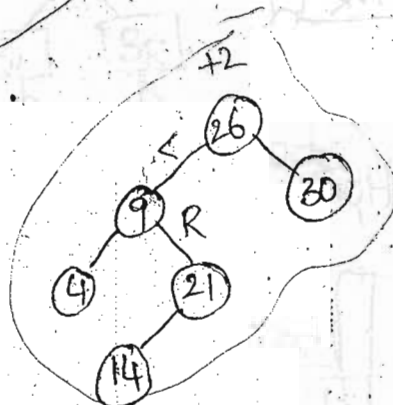
21, 26, 30 | 9, 4, 14 | 28 | 18, 15 | 10 | 2 | 3 | 7

RR LL LR RL RL LL LR LR

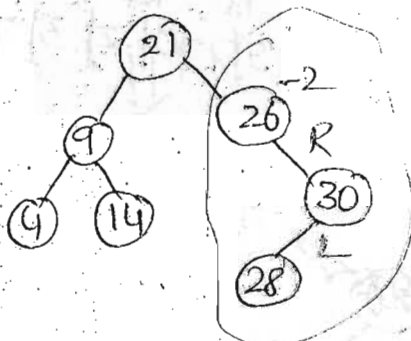


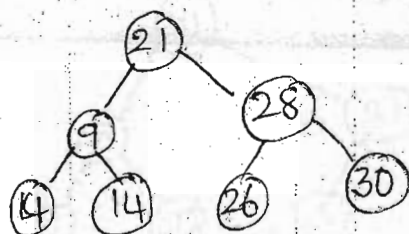


Insert: 14

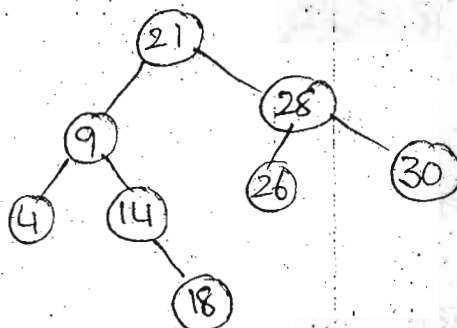


Insert: 28

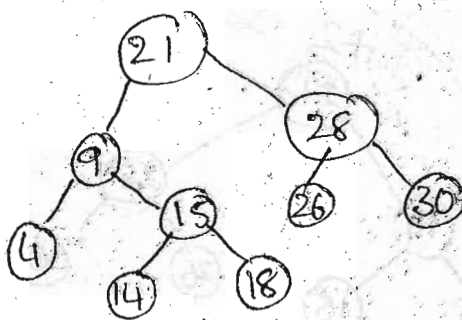
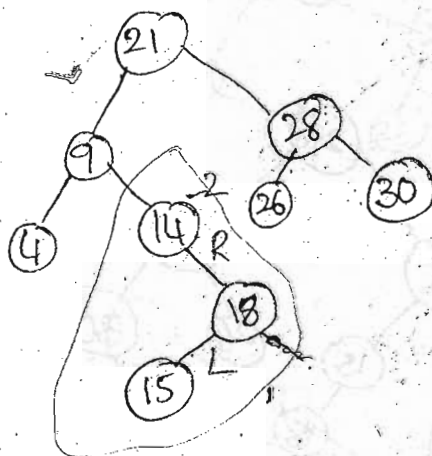




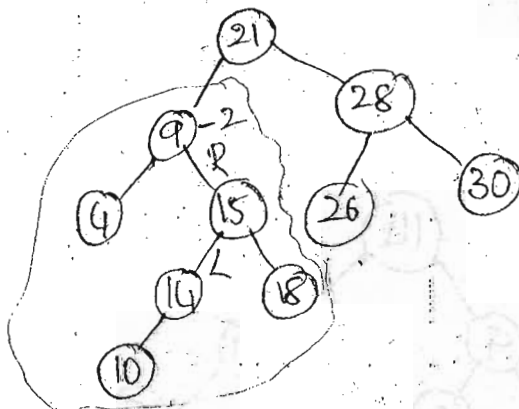
Insert: 18

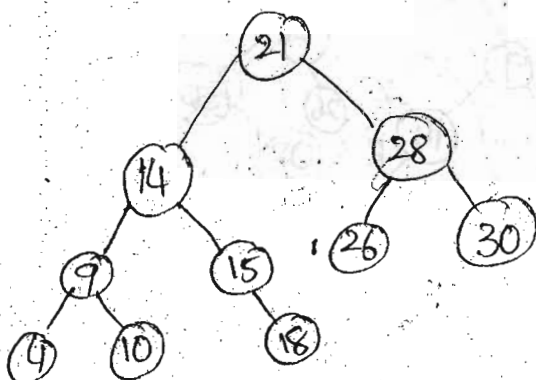
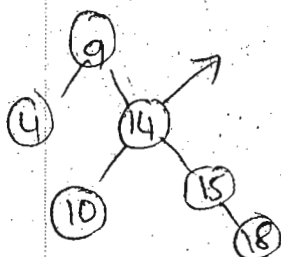
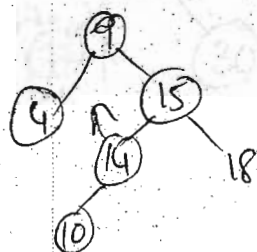


Insert: 15

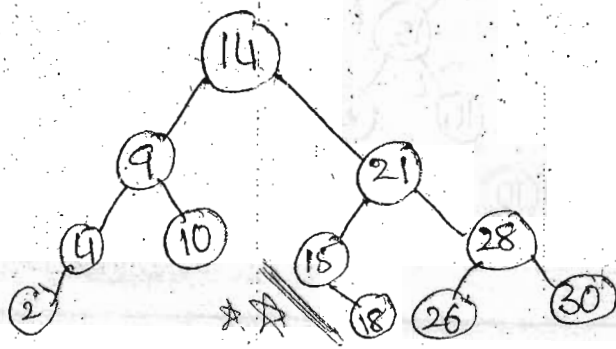
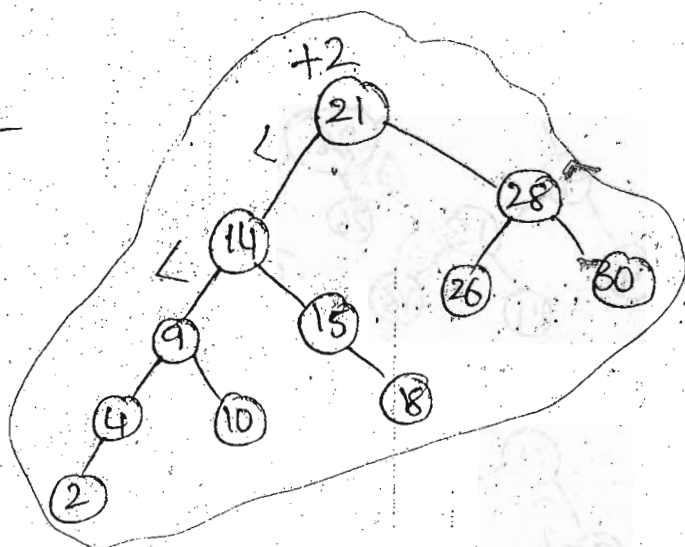


Insert: 10

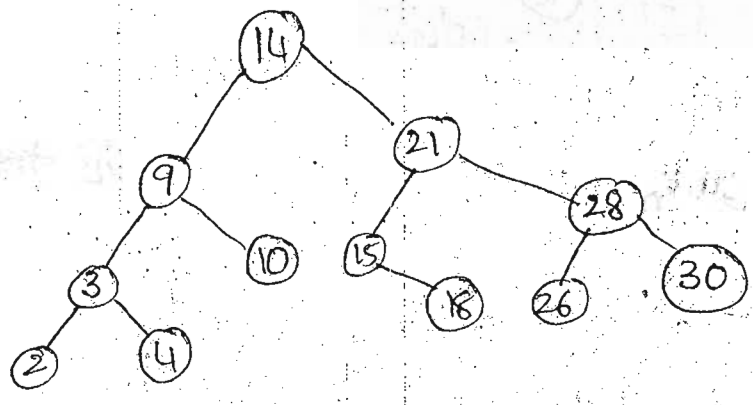
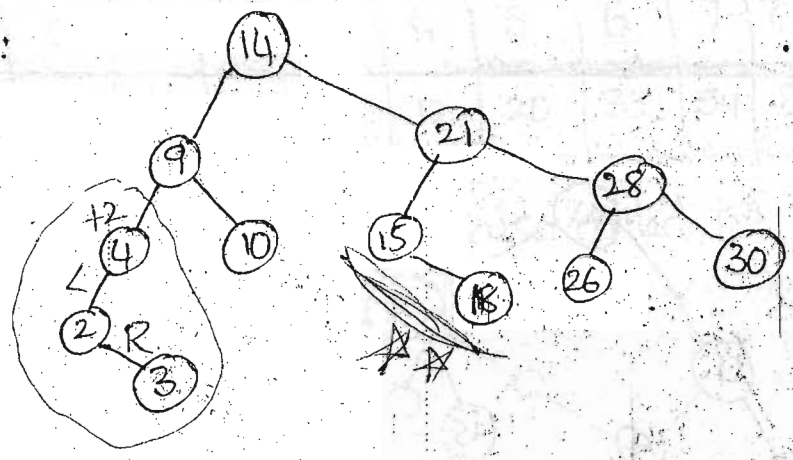




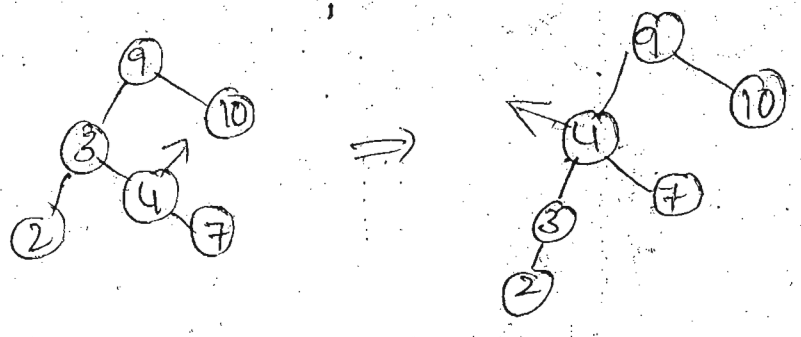
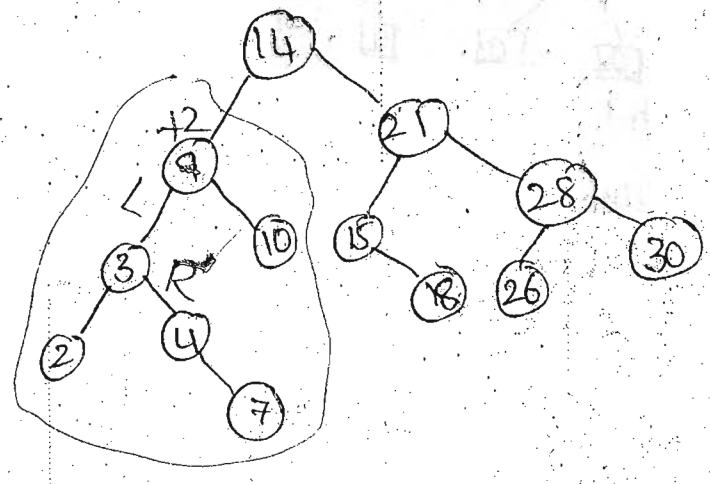
Insert: 2

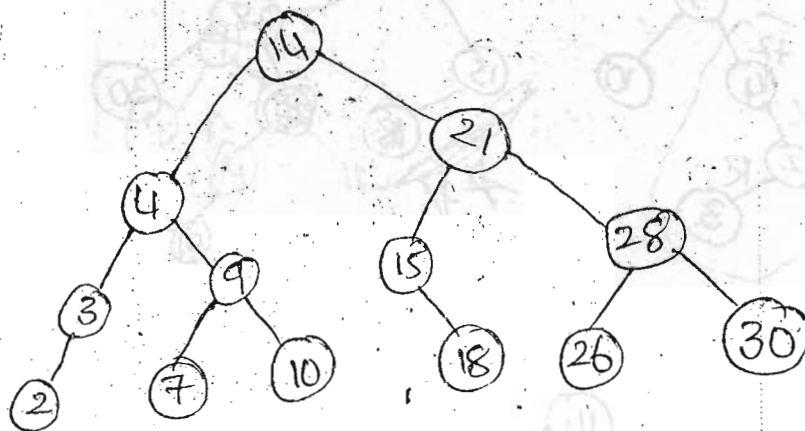


Insert 3:



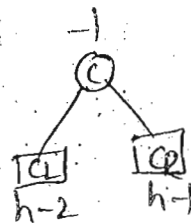
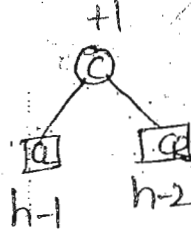
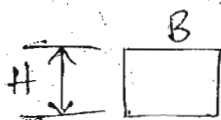
Insert 7:





q) calculate min no. of nodes in AVL tree of height  $h=8$  (level starts at 0)

Sol :

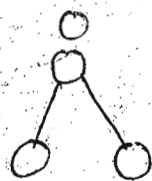


$$L=0 \rightarrow 0$$

$$N(0)=1$$

$$L=1 \Rightarrow L=0$$

$$L=1$$

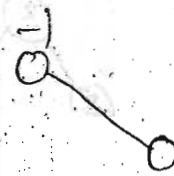


$$N(1)=3$$

~~This is not considered~~



$$N(1)=2$$



$$N(1)=2$$

$$N(H)_{\min}$$

$$= \begin{cases} 1 & L=0 \\ 2 & L=1 \\ 1+N(H-1)+N(H-2) & L>1 \end{cases}$$

looks like fibonacci series

| $L=H=D$       | 0 | 1 | 2 | 3 | 4  | 5  | 6  | 7  | 8  |
|---------------|---|---|---|---|----|----|----|----|----|
| $N(H)$<br>min | 1 | 2 | 4 | 7 | 12 | 20 | 33 | 54 | 88 |

52

Eq: the following AVL trees also called as Fibonacci heaps

collection of trees  
not heap sort heap

1)  $L=0$

$L=1$

$L=2$

$N(2) = 4$

Left biased

(or)

$N(2) = 4$

Right biased

2)

$L=0$

$L=1$

$L=2$

$L=3$

$N(3) = 7$

(or)

$N(3) = 7$

AVL deletion:

Note:

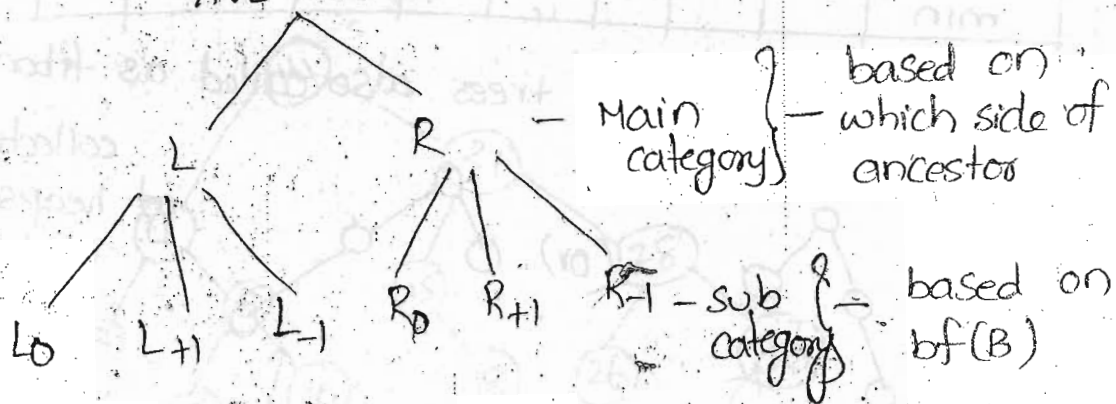
- A single insertion may cause the maximum no. of two rotations

- A single deletion may cause the maximum no. of multiple rotations (worst case)

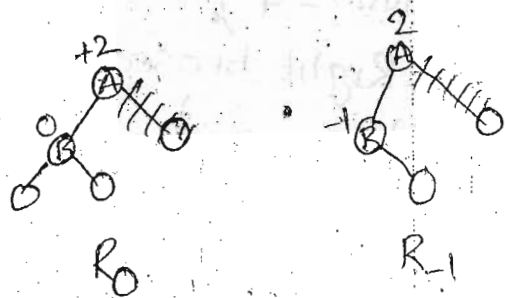
$O(\log_2 n)$

- Every insertion and every deletion may not cause rotation

# AVL deletion

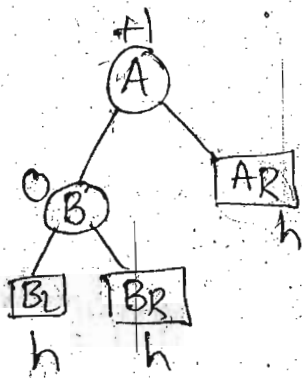


Eg:

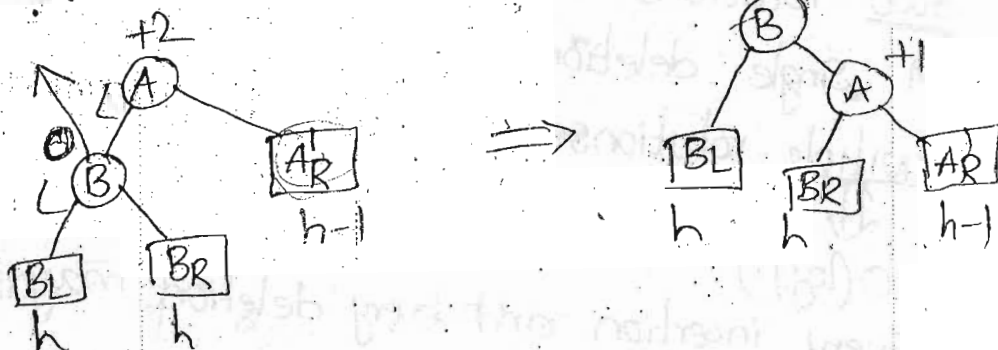


R0:

a) Before deletion

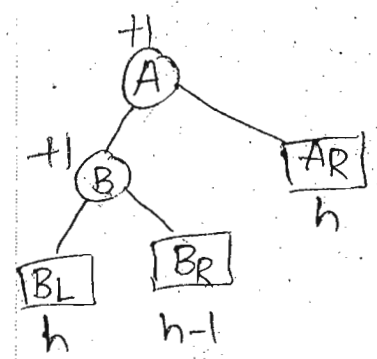


b) deleting a node in AR

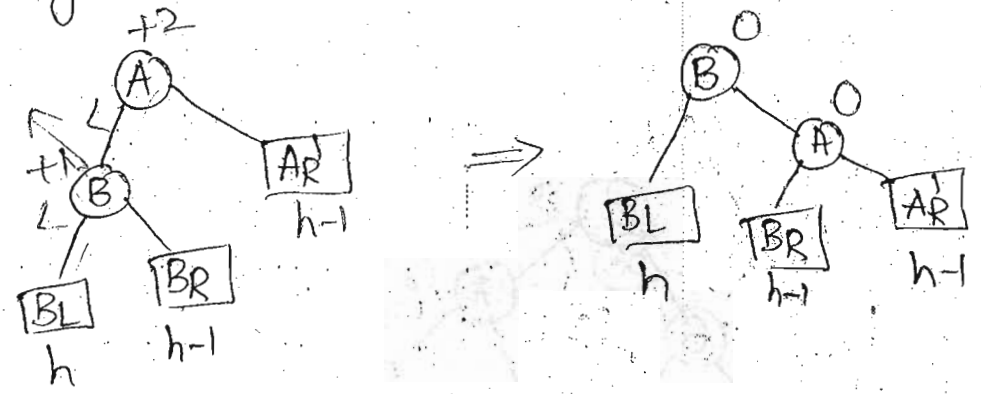


R<sub>1</sub>

a) Before deletion

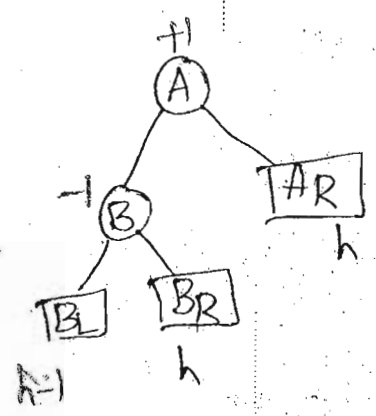


b) deleting a node in AR

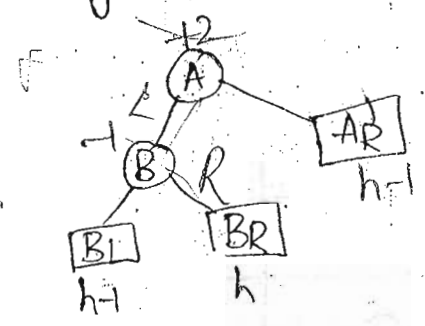


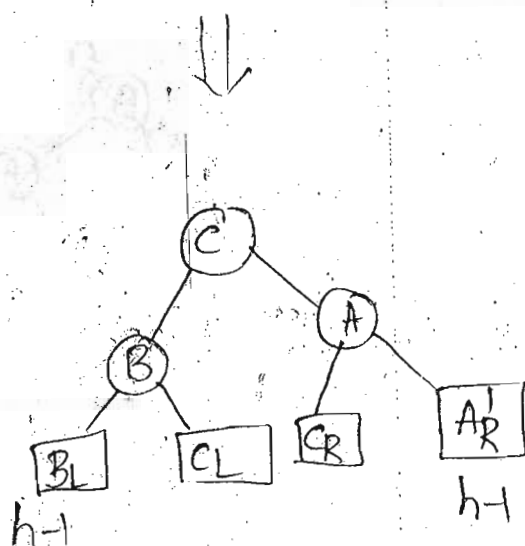
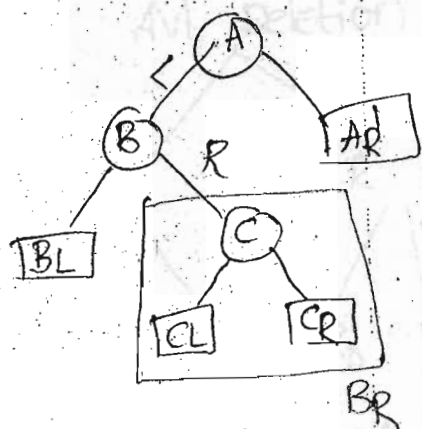
R<sub>-1</sub>

a) Before deletion



b) deleting a node in AR





|                 | Similar to | bf(B)                                           | bf(A) | Remark<br>100% same |
|-----------------|------------|-------------------------------------------------|-------|---------------------|
| R <sub>0</sub>  | LL         | -1                                              | +1    | NO                  |
| R <sub>+1</sub> | LL         | 0                                               | 0     | Yes                 |
| R <sub>-1</sub> | LR         | $bf(C) \begin{cases} +1 \\ 0 \\ -1 \end{cases}$ |       |                     |

|                 | Similar to | bf(B)                                           | bf(A) | Remark<br>100% same |
|-----------------|------------|-------------------------------------------------|-------|---------------------|
| L <sub>0</sub>  | RR         | +1                                              | -1    | NO                  |
| L <sub>+1</sub> | RL         | $bf(C) \begin{cases} +1 \\ 0 \\ -1 \end{cases}$ |       |                     |
| L <sub>-1</sub> | RR         | 0                                               | 0     | Yes                 |

# AVL deletion:

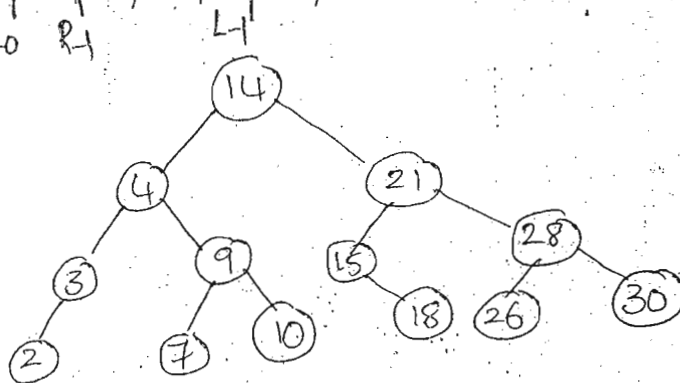
## Procedure:

1. Delete a node as per BST
2. configure if imbalance  
multiple rotations then balance it by applying rotations.

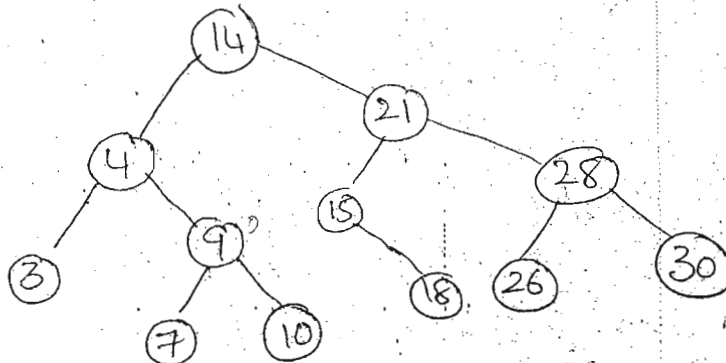
Eg: Delete the following nodes

2, 3, 10, 18, 4, 9, 14, 7, 15

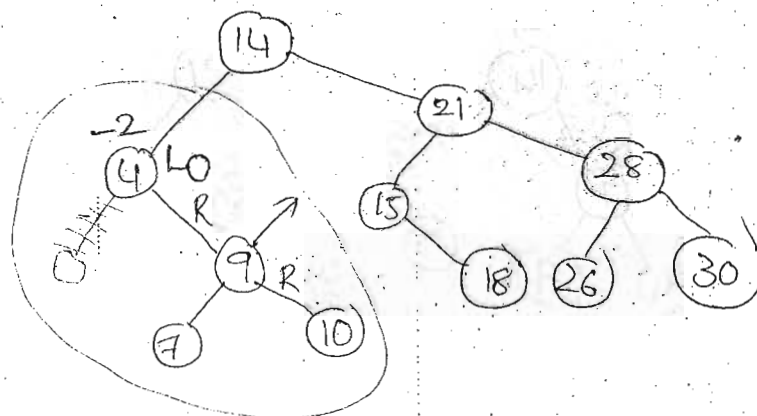
Sol:

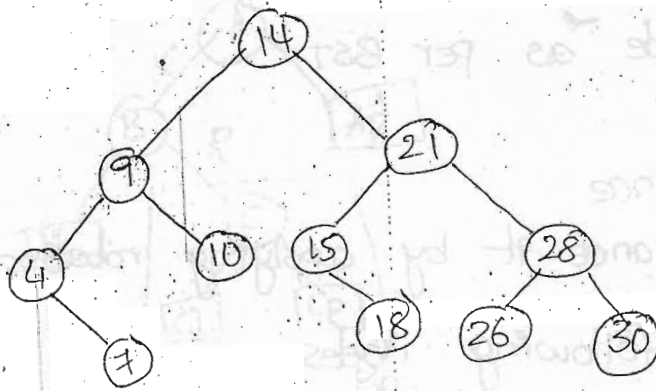


Delete 2:

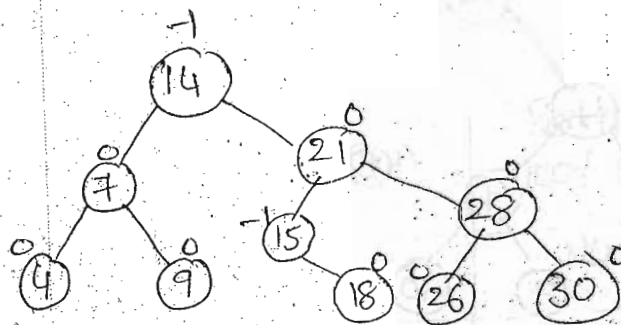
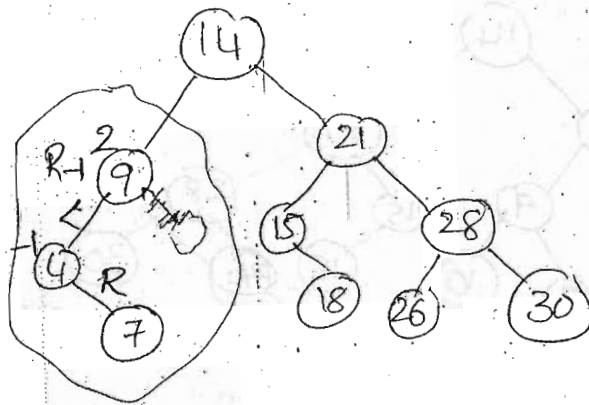


Delete 3:

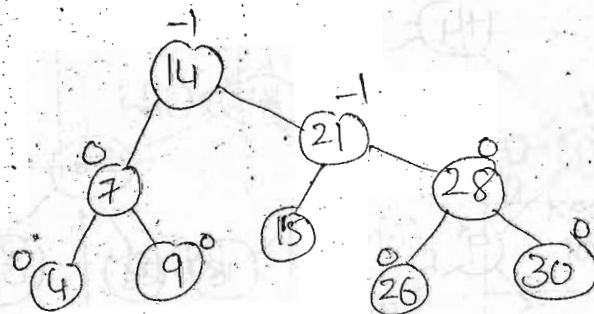




Delete 10:

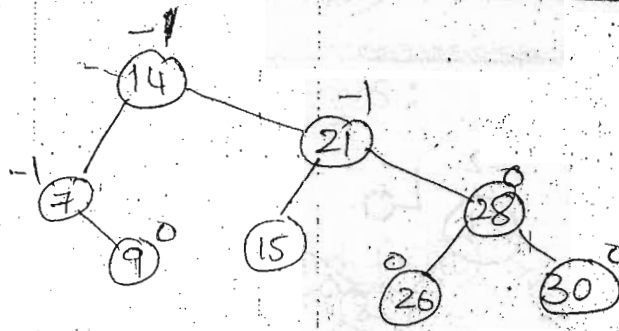


Delete 18:

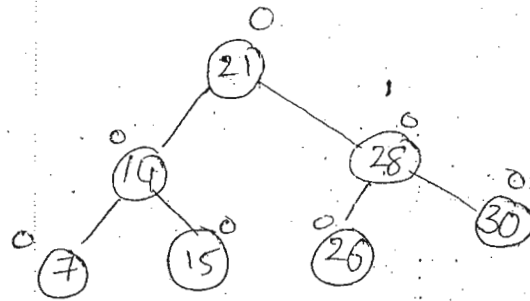
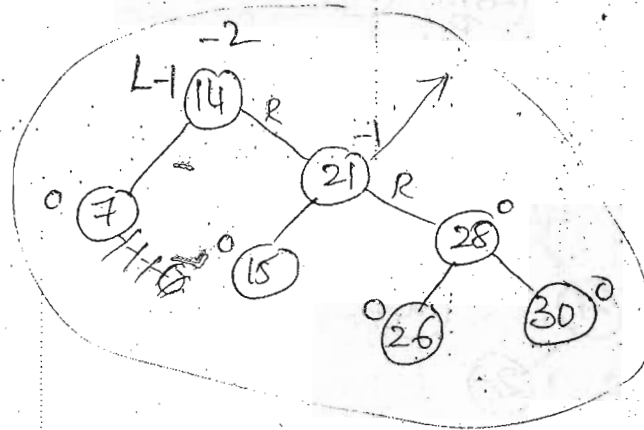


Delete 4:

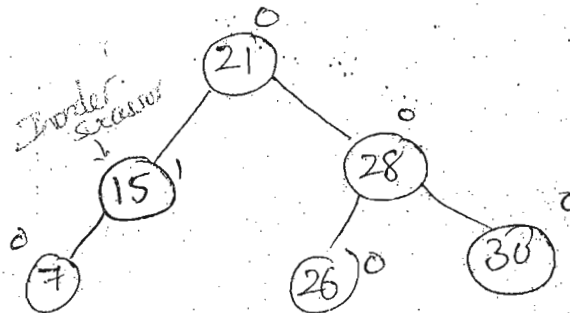
60



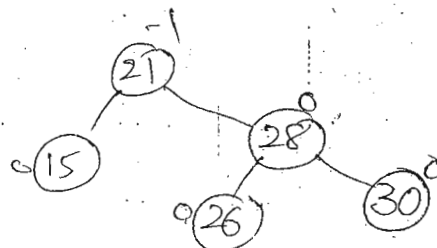
Delete 9:



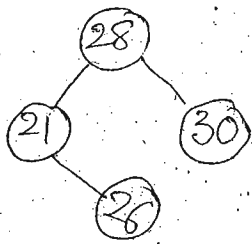
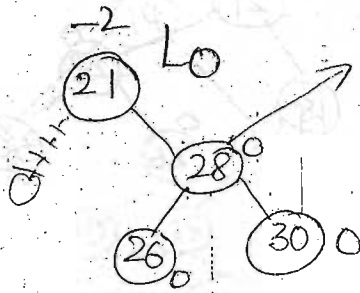
Delete 14:



Delete 7:



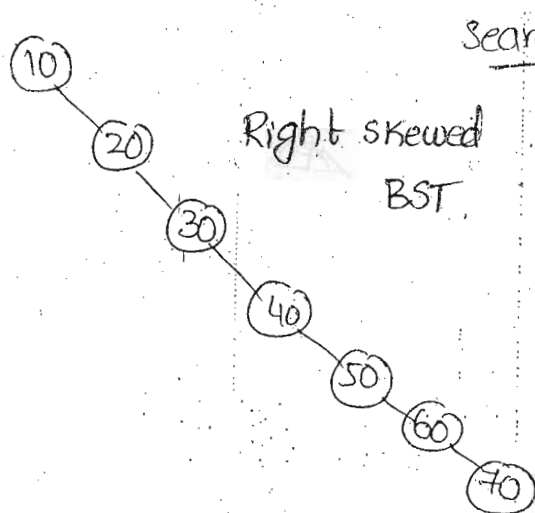
Delete 15:



# splay trees :- (weiss)

61

need to go for splay trees :



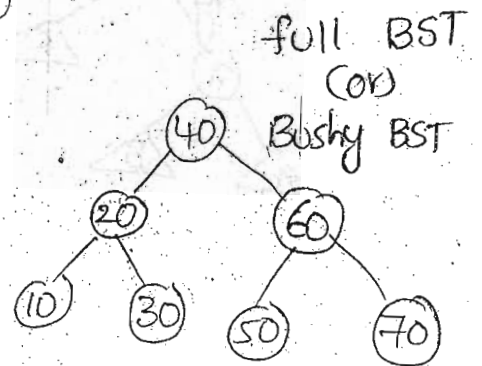
Right skewed  
BST

1 time  $\rightarrow$  7 attempts

splay @ root

999 times  $\times$  1 attempt  $\rightarrow$  999 attempts/  
probes

1000 times  $\rightarrow$  1006 probes



1 time  $\rightarrow$  3 attempts

1000 times  $\times$  3  $\rightarrow$  3000 attempts

Note:

In the world of IT, when a record is retrieved, it has been observed that the same record is going to be accessed for SEVERAL TIMES.

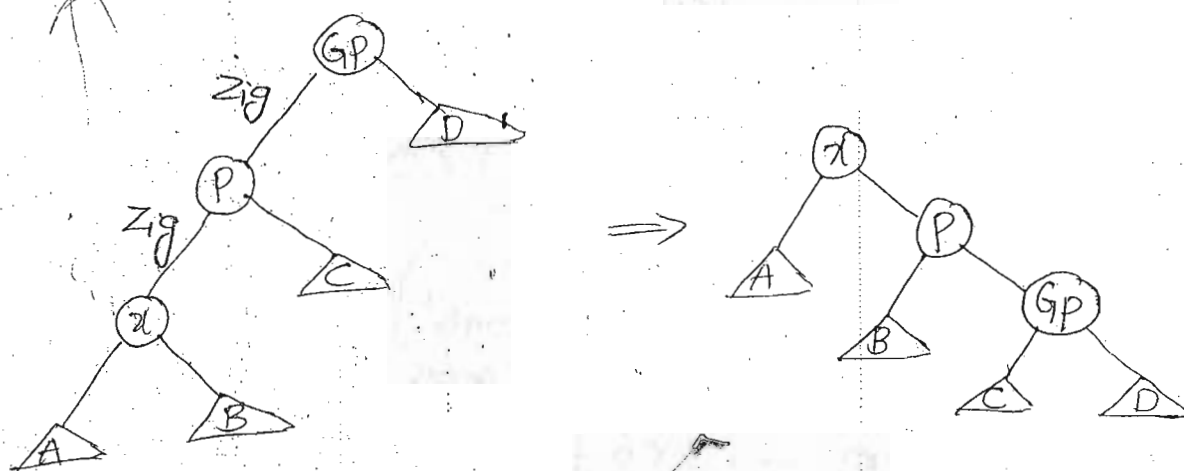
So splaying that record towards root requires the following rotations

- i) Zig
- ii) Zig-Zig
- iii) Zig-Zag

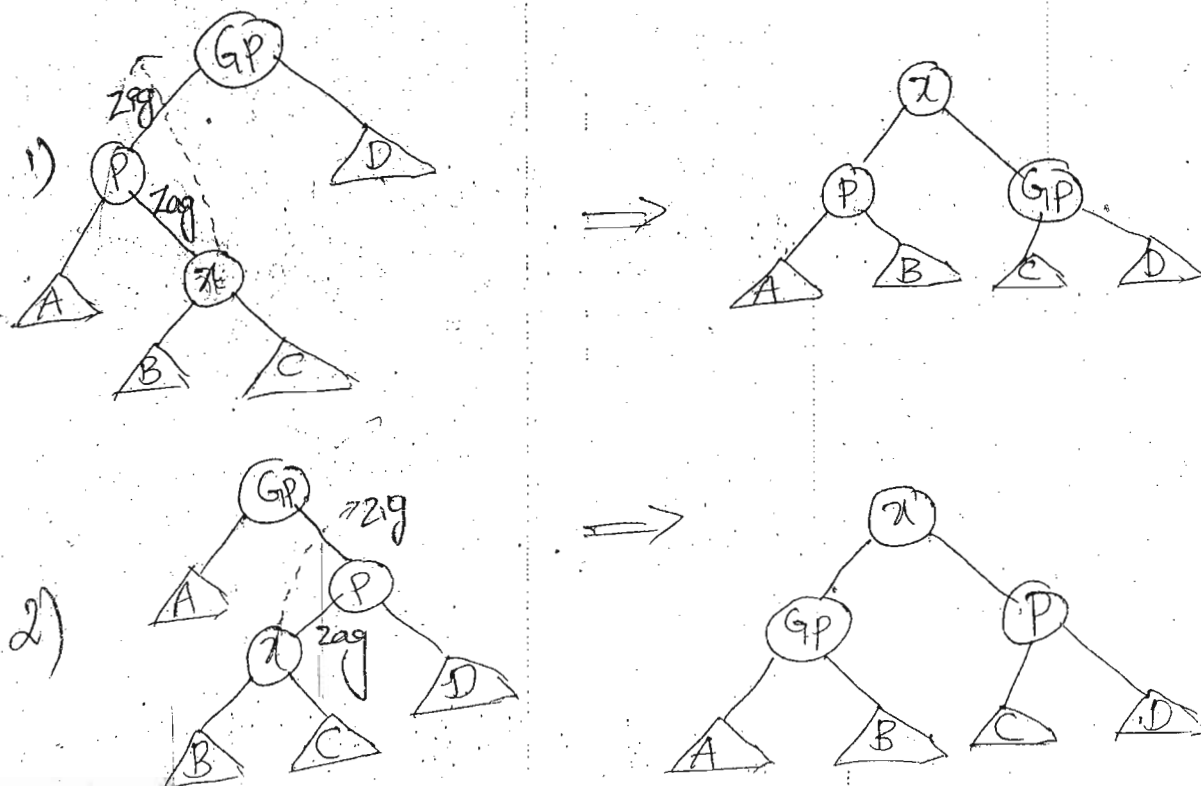
I. Zig:



II. Zig-Zig



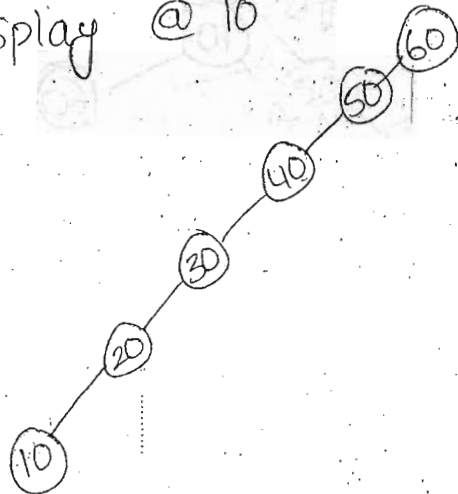
III. Zig-zag



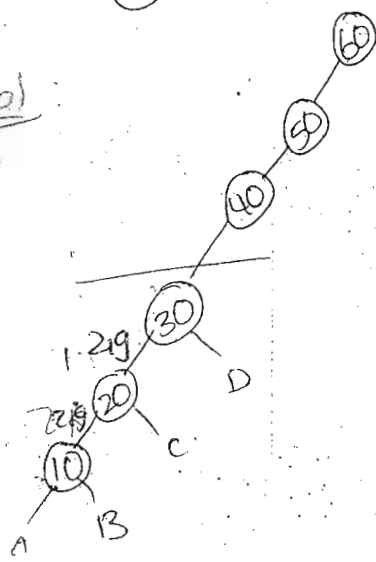
Note

By one rotation, we can splay only two levels.

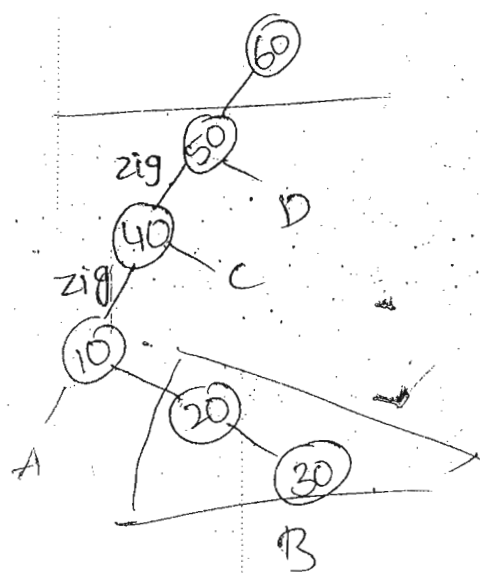
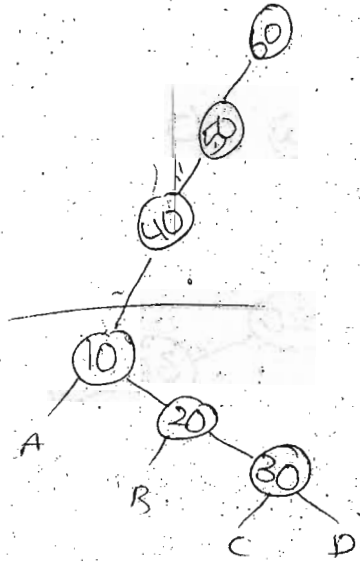
Eg: Splay @ 10



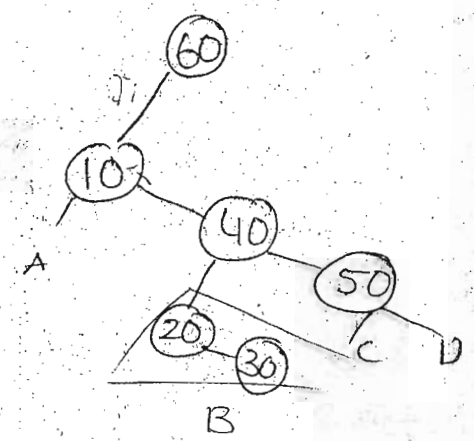
Sol

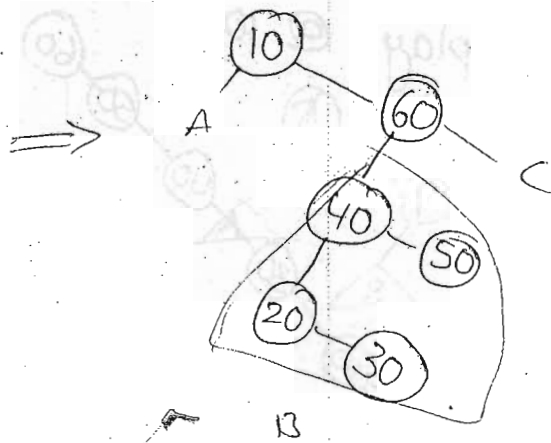
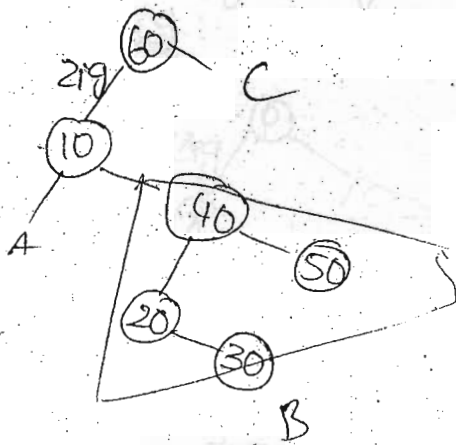


$\Rightarrow$



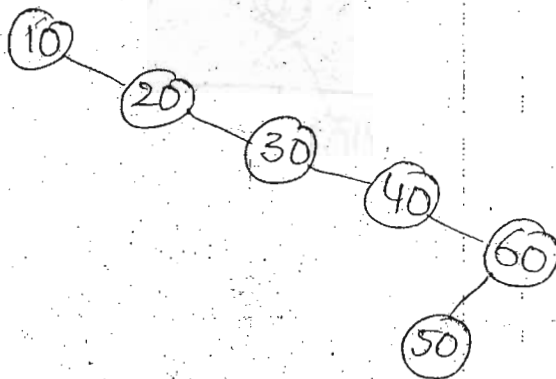
$\Rightarrow$



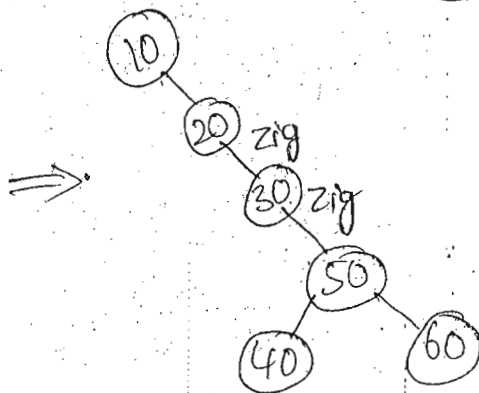
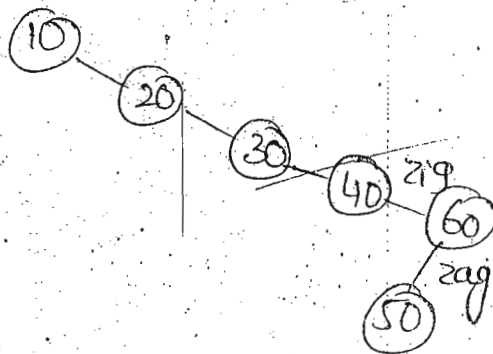


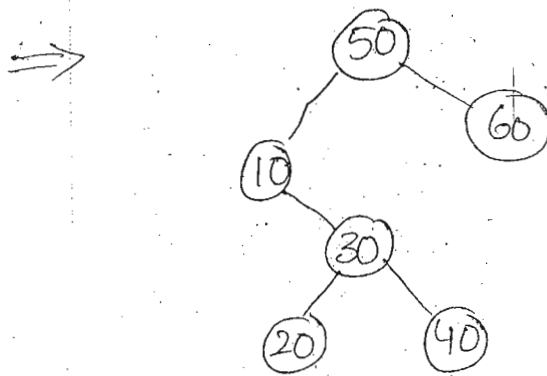
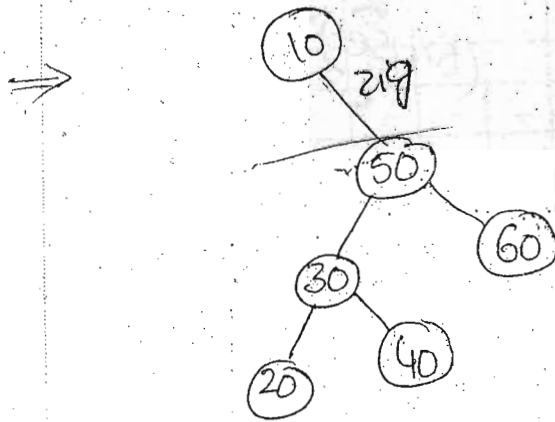
Eg :

splay @ 50

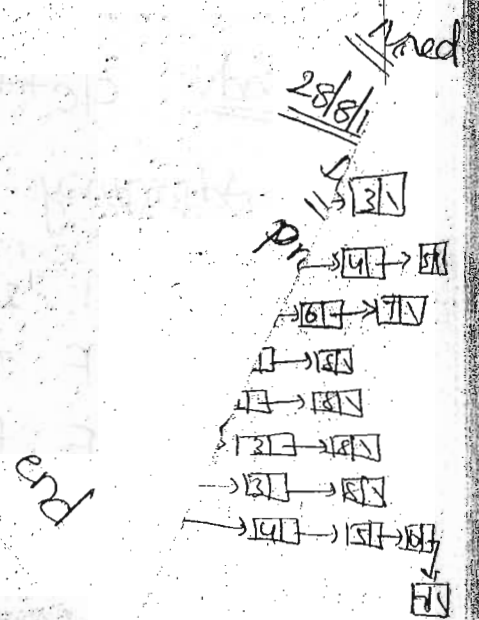


Sol :





Note:  
By default, splay trees are BST but not AVL tree

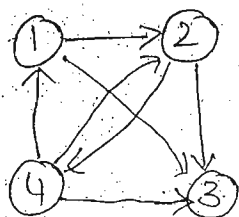


# Graphs (Krose)

$$G=(V, E)$$

$V \rightarrow$  vertices

$E \rightarrow$  edges



## Representation in Memory:

I) Set:

| <u>vertex</u> | <u>adjacency sets</u> |
|---------------|-----------------------|
| 1             | {2, 3}                |
| 2             | {3, 4}                |
| 3             | $\emptyset$           |
| 4             | {1, 2, 3}             |

adv: pascal supports sets as datatype

disadv: c/c++ does not support sets as datatype

II) Adjacency matrix (or) Table

|   |   |   |   |   |
|---|---|---|---|---|
|   | 1 | 2 | 3 | 4 |
| 1 | F | T | T | F |
| 2 | F | F | T | T |
| 3 | F | F | F | F |
| 4 | T | T | T | F |

Adv: Array: simple datatype in all lang.

Disadv: static (can't add new vertices/edges)

### III) Contiguous valence factor

adjacency matrix

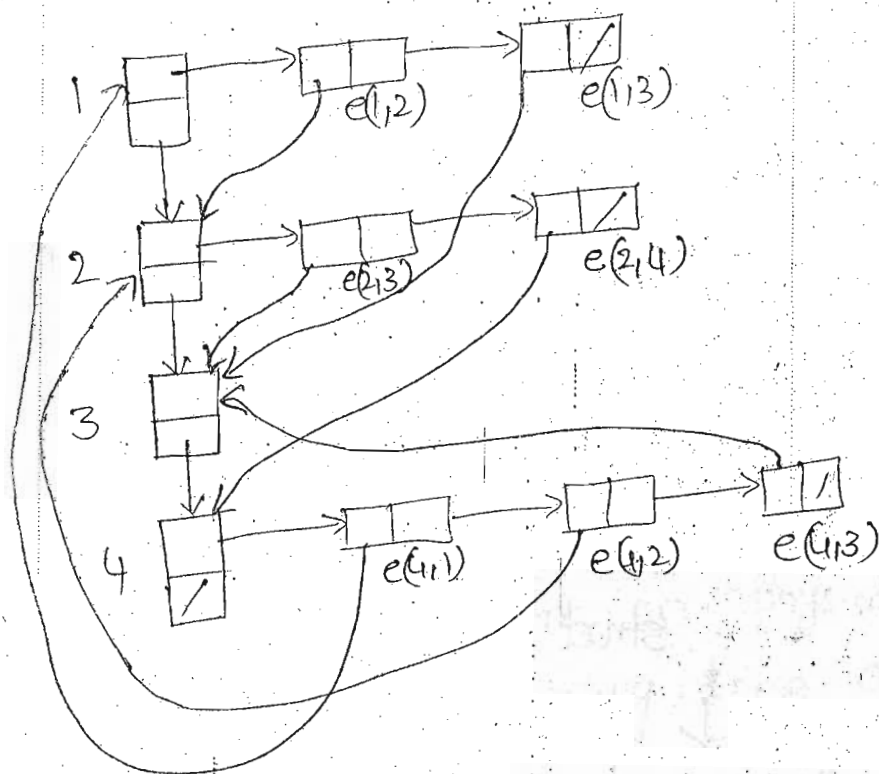
64

|       |   |   |   |   |   |   |   |   |
|-------|---|---|---|---|---|---|---|---|
| 1     | 2 | 2 | 3 | - | - | - | - | - |
| 2     | 2 | 3 | 4 | - | - | - | - | - |
| 3     | 0 | - | - | - | - | - | - | - |
| n=4   | 3 | 1 | 2 | 3 | - | - | - | - |
| 5     | . | . | . | . | . | . | . | . |
| 6     | . | . | . | . | . | . | . | . |
| Max=7 | . | . | . | . | . | . | . | . |

adv: limited dynamic

disadv: more space wastage

### IV) Linked representation



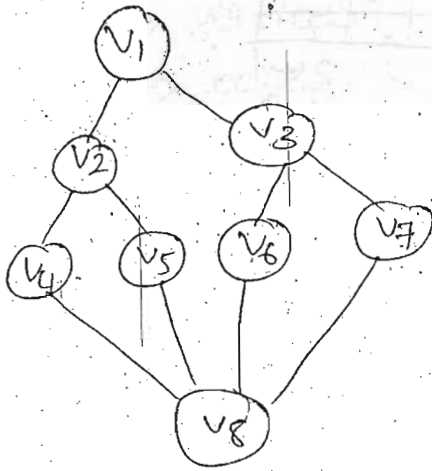
adv: dynamic

disadv: more space for links

ges)

IV) Hybrid = {array + linked lists}

- used in real world



visited[] A

|   |                |                         |
|---|----------------|-------------------------|
| F | V <sub>1</sub> | → [2] → [3]             |
| F | V <sub>2</sub> | → [1] → [4] → [5]       |
| F | V <sub>3</sub> | → [1] → [6] → [7]       |
| F | V <sub>4</sub> | → [2] → [8]             |
| F | V <sub>5</sub> | → [2] → [8]             |
| F | V <sub>6</sub> | → [3] → [8]             |
| F | V <sub>7</sub> | → [3] → [8]             |
| F | V <sub>8</sub> | → [4] → [5] → [6] → [7] |



enum bool { false, true }

struct Graph

{

int n; // no. of vertices

bool visited[10]; // n ≤ 10

struct Node \*A[10];



Note: Graph is not self referential structure

28/6/11

Depth First Search (DFS):

procedure DFS(V)

visited(V) = true

for each vertex 'w', adjacent for 'v'

if not visited(w)

call DFS(w)

end

Q) Traversal: DFS

① Starting vertex is  $V_1$

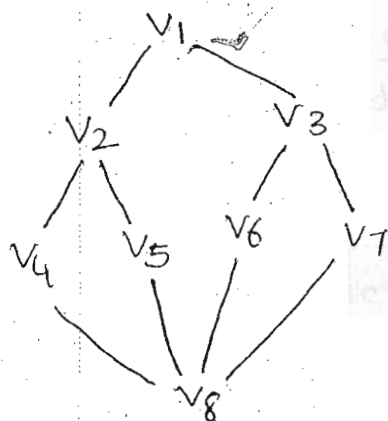
a) ~~what~~ what is the sequence of exploration

b) what is the stack contents

c) what vertices are not pushed in?

d) what vertices are pushed in for more than once?

Sol:



Its physical structure is hybrid structure

Given graph is a data structure. If we work on this data structure, more than one pattern can be obtained.

But if we work on physical structure (memory) unique pattern is possible.

∴ so we need to work on physical structure so that unique DFS spanning tree can be obtained

| V     | w     | stack contents       |
|-------|-------|----------------------|
| $V_1$ | $V_2$ | $V_1$                |
| $V_2$ | $V_1$ | $V_1, V_2$           |
| $V_4$ | $V_2$ | $V_1, V_2, V_4$      |
| $V_8$ | $V_4$ | $V_1, V_2, V_4, V_8$ |
|       | $V_5$ |                      |

|   |                                     |       |                         |
|---|-------------------------------------|-------|-------------------------|
| T | <input checked="" type="checkbox"/> | $V_1$ | → $V_2$ → $V_3$         |
| T | <input checked="" type="checkbox"/> | $V_2$ | → $V_1$ → $V_4$ → $V_8$ |
| T | <input checked="" type="checkbox"/> | $V_3$ | → $V_1$ → $V_6$ → $V_7$ |
| T | <input checked="" type="checkbox"/> | $V_4$ | → $V_2$ → $V_8$         |
| T | <input checked="" type="checkbox"/> | $V_5$ | → $V_2$ → $V_8$         |
| T | <input checked="" type="checkbox"/> | $V_6$ | → $V_3$ → $V_7$         |
| T | <input checked="" type="checkbox"/> | $V_7$ | → $V_3$ → $V_6$         |
| T | <input checked="" type="checkbox"/> | $V_8$ | → $V_4$ → $V_5$ → $V_6$ |

| V     | w     | stack contents                 |
|-------|-------|--------------------------------|
| $v_5$ | $v_2$ |                                |
|       | $v_8$ | $v_1, v_2, v_4, v_8$           |
|       | pop   |                                |
| $v_8$ | $v_6$ | $v_1, v_2, v_4, v_8$           |
| $v_6$ | $v_3$ | $v_1, v_2, v_4, v_8, v_6$      |
| $v_3$ | $v_1$ |                                |
|       | $v_6$ |                                |
|       | $v_7$ | $v_1, v_2, v_4, v_8, v_6, v_3$ |
| $v_7$ | pop   |                                |
| $v_3$ |       |                                |

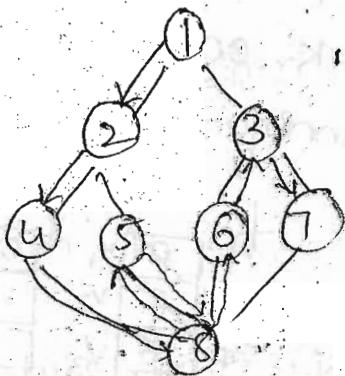
already all  $v_5$  adjacent  
are explored.

Note:

when to push?

If a node has atleast one unexplored adjacent  
node

→



It is DS. But Remember  
we are working with  
physical structure

sequence :  $v_1, v_2, v_4, v_8, v_5, v_8, v_6, v_3, v_7$

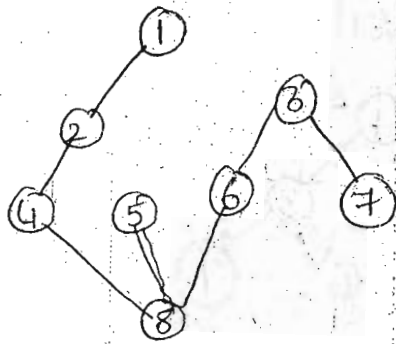
stack :  $v_1, v_2, v_4, v_8, v_8, v_6, v_3$

not pushed in:  $v_5, v_7$

more than once:  $v_8$

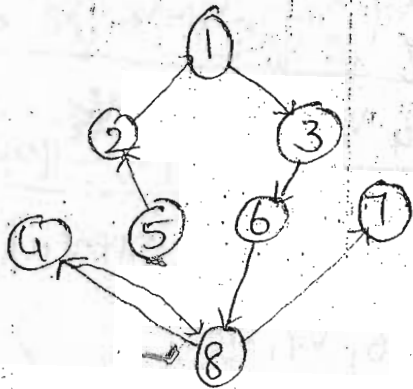
spanning tree is

66



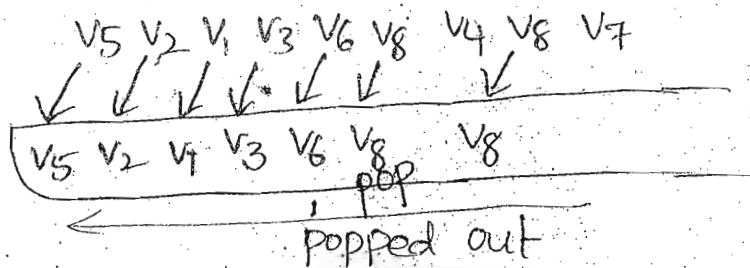
p) If the starting vertex is  $V_5$  then repeat above a, b, c, d

Sol :



sequence :

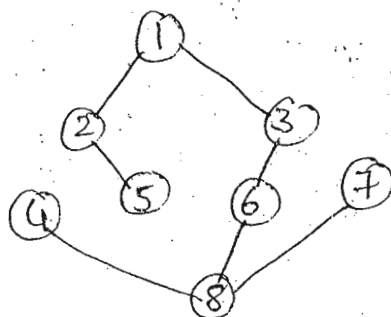
stack :



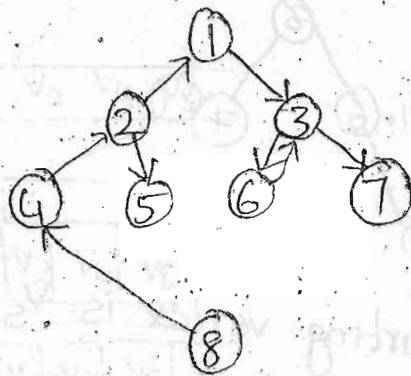
not pushed in:  $V_4, V_7$

pushed in more than once:  $V_8$

∴ DFS spanning tree is



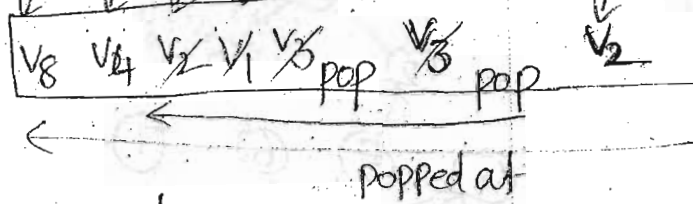
d) If the starting vertex is  $V_8$ , then repeat above



Sequence:

$V_8, V_4, V_2, V_1, V_3, V_6, V_3, V_7, V_2, V_5$

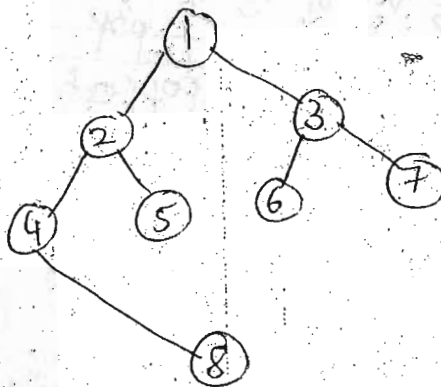
stack:



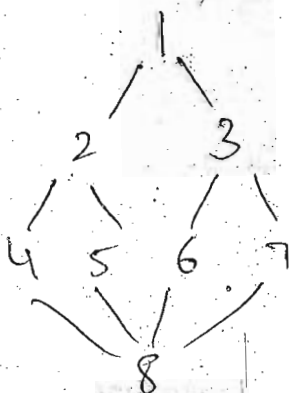
Not pushed in :  $V_6, V_7, V_5$

more than once :  $V_3, V_2$

$\therefore$  DFS spanning tree is



(7)



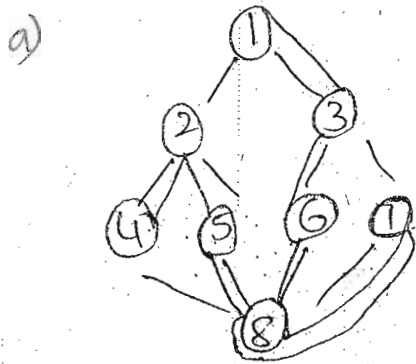
Ques

Traversal: DFS. If starting vertex is  $v_1$ , identify the valid DFS sequences

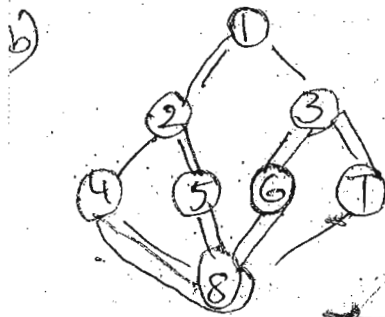
67

- a) 1 3 6 8 7 5 2 4
- b) 1 2 5 8 4 6 3 7
- c) 1 2 4 8 7 3 6 5
- d) 1 3 7 8 4 2 5 6
- e) 1 2 4 5 8 6 7 3

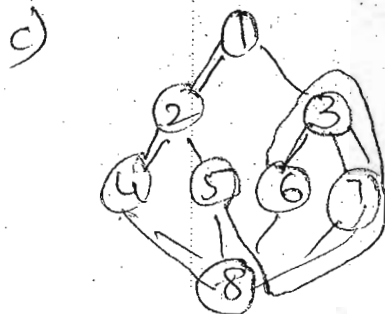
Sol: Here more than one sequence.  
so we need to work on Datastructure



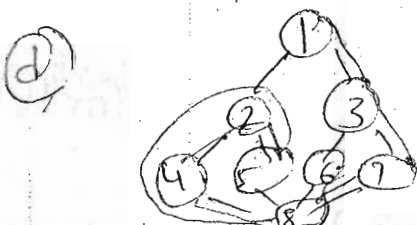
— valid



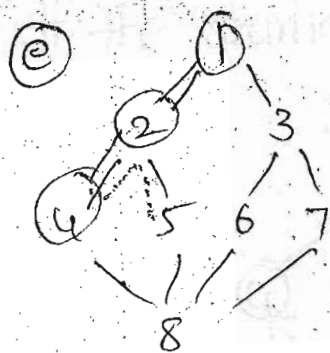
— valid



— valid



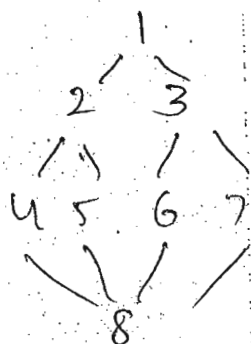
— valid



5  $\rightarrow$   $\therefore$  Invalid

p) If the starting vertex is  $v_5$  then identify Invalid sequences

- a) 5 2 4 8 6 7 3  $\rightarrow$  Invalid  
 b) 5 8 6 1 3 1 2 4  $\rightarrow$  Invalid  
 c) 5 2 4 8 7 3 6 1  $\rightarrow$  valid  
 d) 5 8 6 3 7 1 2 4  $\rightarrow$  valid



Breadth First Search (BFS):

procedure BFS (V)

visited[V] = true;

enqueue(V);

while Queue is not empty

$v = \text{dequeue}()$ ;

    for all vertices 'w', adjacent for 'v'  
         if not visited(w)

```

 } visited(w) = True;
 enqueue(w)
 }
 }

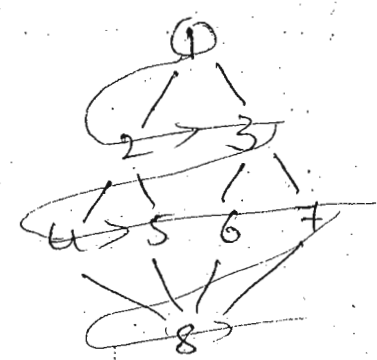
```

end



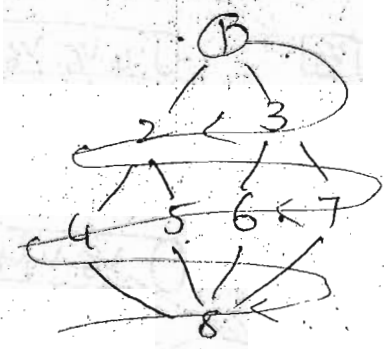
BFS

L-R BFS (left to right)



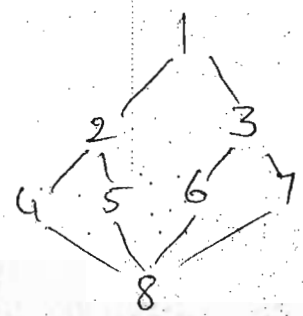
1, 2, 3, 4, 5, 6, 7, 8

R-L BFS (Right to left)



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

① Identify whether the above pseudo code is LR BFS (or) RL BFS

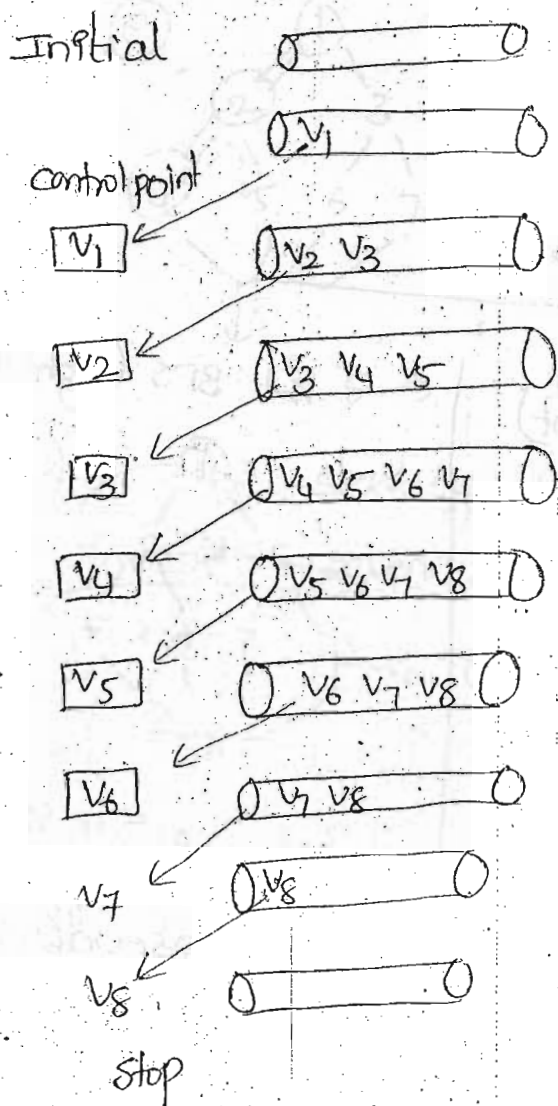


starting vertex is  $v_1$

Assume we are working on physical structures

Then we need to find the status of queue.

Initially queue is empty



∴ No ~~re~~ node is inserted more than once  
every vertex is getting visited

Q) what is the status of queue after '4' is deleted

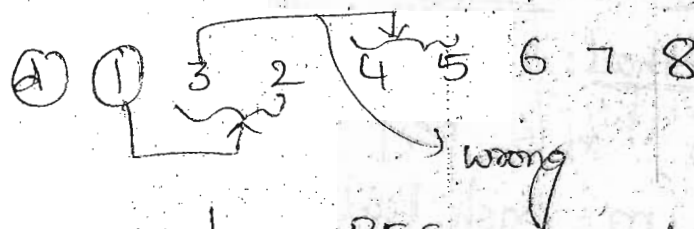
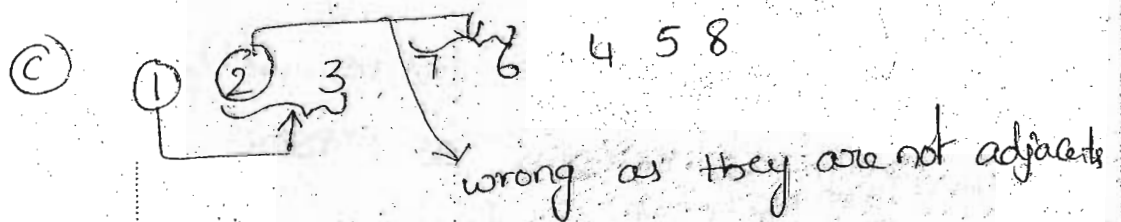
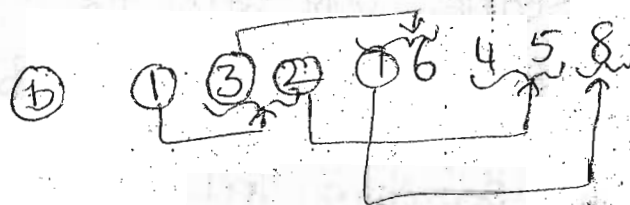
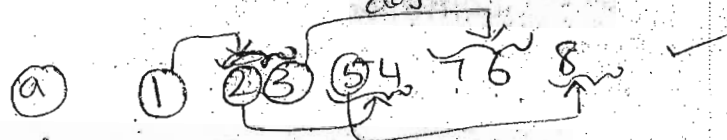
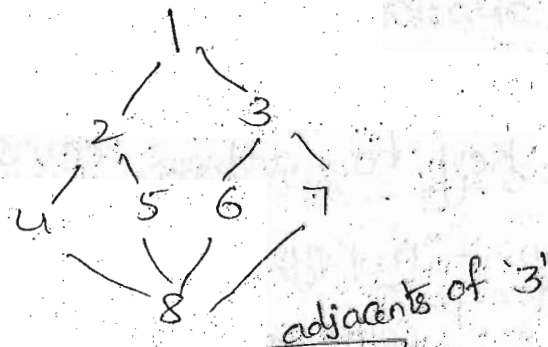
Sol:  $v_5, v_6, v_7, v_8$

— The above code does not distinguish L-R BFS  
(or) R-L BFS. It all depends on the way the  
graph is represented in physical memory  
i.e. physical structure

Q) Traversal : BFS. If starting vertex is  $v_1$   
Identify valid/Invalid sequences.

- a) 1 2 3 5 4 7 6 8 - valid
- b) 1 3 2 7 6 4 5 8 - valid
- c) 1 2 3 7 6 4 5 8 - Invalid
- d) 1 3 2 4 5 6 7 8 - Invalid

Sol:- More than one sequence.  
Hence we need to work on Data structure graph



| → DFS                         | BFS                         |
|-------------------------------|-----------------------------|
| - Recursive                   | - Not Recursive (Iterative) |
| - stack (Implicitly)          | - queue (Explicitly)        |
| - In algorithm keyword - EACH | Keyword - ALL               |
| - Similar to PREORDER         | - similar to level order    |

- |                                                                                                                                                                |                                                                                                                   |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------|
| <ul style="list-style-type: none"> <li>- Certain vertices are not pushed in</li> <li>- more than once pushed</li> <li>- Termination: Stack is empty</li> </ul> | <ul style="list-style-type: none"> <li>- All are pushed</li> <li>- only once</li> <li>- Queue is empty</li> </ul> |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------|

## Hashing

Definition:

Hash function — Key to address transformation

Perfect hash function — 'n' keys Vs 'n' locations without collisions  
(PHF)

minimal PHF — Keys > locations but collision starts only after the entire hashtable is filled up

Goal:

- Hashing is a searching technique
  - Search time is to be  $O(1)$  but not fulfilled.
- still it remains as vision

Types of hash functions:

① Division method:

$$H(x) = x \% m, \quad x \text{ is Key}$$

where  $m$  = hash table size

Knuth,  $m$  is prime number. It is a suggestion

Eg:  $m = 5$

$$H(x) = x \% 5$$

§

| Key | 69 | 77 | 100 | 83 | 16 | 18 |
|-----|----|----|-----|----|----|----|
| loc | 4  | 2  | 0   | 3  | 1  | 3  |

70

|   |     |
|---|-----|
| 0 | 100 |
| 1 | 16  |
| 2 | 77  |
| 3 | 83  |
| 4 | 69  |

Collision

p) which of following is suitable hash function for storing the keys in the range 1 to 1000

(a)  $H(x) = x \bmod 1000 \rightarrow 0 - 999$

(b)  $H(x) = (x+1) \bmod 1000 \rightarrow 0 - 999$

(c)  $H(x) = x \bmod (1000+1) \rightarrow 0 - 1000$

(d)  $H(x) = (x \bmod 1000) + 1 \rightarrow 1 - 1000$

② Folding method:

fold shifting method

| Key | loc |
|-----|-----|
| 123 | 000 |
| 214 | 548 |
| 911 | 999 |

Fold Boundary Method

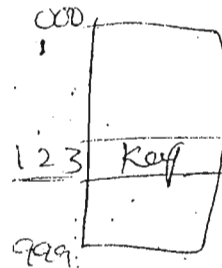
| Key | loc |
|-----|-----|
| 123 | 000 |
| 211 | 334 |
| 334 | 999 |

### ③ Mid Square Method:

Key  $\rightarrow$  loc  
12345  $\rightarrow$  ?

$$(12345)^2 = 152390625$$

suppose

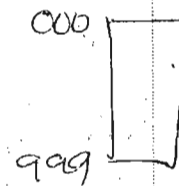


### ④ Truncation method:

Key  $\rightarrow$  loc

|        |       |
|--------|-------|
| 123456 | 456   |
| 78967  | 967   |
| 312456 | 456 x |
| 256456 | 456 x |

It is fast  
But more collisions



30/8/11

### Collision Resolution Techniques

open addressing (or) closed hashing

separate chaining

linear  
probing

Primary  
clustering

quadratic  
probing

Secondary  
clustering

Random  
probing

Random  
clustering

Double hashing/  
Rehashing

clustering is reduced  
but not avoided

Disadvantages

Q) which of the following choices gives a possible order in which the key values could have been inserted for a hash function  $A(K) = K \cdot 10$  with linear probing

Sol:

Q) K  
10

Q) K

Q) H

Key  
will

(a) 46, 42, 34, 52, 23, 33

(b) 34, 42, 23, 52, 33, 46

(c) 46, 34, 42, 23, 52, 33

(d) 42, 46, 33, 23, 34, 52

Given hash table

|   |    |
|---|----|
| 0 |    |
| 1 |    |
| 2 |    |
| 3 | 42 |
| 4 | 23 |
| 5 | 34 |
| 6 | 52 |
| 7 | 46 |
| 8 | 33 |
| 9 |    |

|   |    |
|---|----|
| 0 |    |
| 1 |    |
| 2 | 42 |
| 3 | 23 |
| 4 | 34 |
| 5 | 52 |
| 6 | 46 |
| 7 | 33 |
| 8 |    |
| 9 |    |

Sol:

(a)

|     |    |    |    |    |    |    |
|-----|----|----|----|----|----|----|
| Key | 46 | 42 | 34 | 52 | 23 | 33 |
| loc | 6  | 2  | 4  | 2  | 3  | 3  |

|   |    |
|---|----|
| 0 |    |
| 1 |    |
| 2 | 42 |
| 3 | 52 |
| 4 | 34 |
| 5 | 23 |
| 6 | 46 |
| 7 | 33 |
| 8 |    |
| 9 |    |

Diagram showing insertion sequence for (a):  
 - 42 is inserted at index 2.  
 - 52 is inserted at index 3.  
 - 23 is inserted at index 3 (collision with 52).  
 - 34 is inserted at index 4.  
 - 52 is inserted at index 2 (collision with 42).  
 - 46 is inserted at index 6.  
 - 33 is inserted at index 7.

Diagram

(b)

|     |    |    |    |    |    |    |
|-----|----|----|----|----|----|----|
| Key | 34 | 42 | 23 | 52 | 33 | 46 |
| loc | 4  | 2  | 3  | 2  | 3  | 6  |

|   |    |
|---|----|
| 0 |    |
| 1 |    |
| 2 | 42 |
| 3 | 23 |
| 4 | 34 |
| 5 | 52 |
| 6 | 33 |
| 7 | 46 |
| 8 |    |
| 9 |    |

Diagram showing insertion sequence for (b):  
 - 42 is inserted at index 2.  
 - 23 is inserted at index 3.  
 - 34 is inserted at index 4.  
 - 52 is inserted at index 5.  
 - 33 is inserted at index 6.  
 - 46 is inserted at index 7.

(c)

|     |    |    |    |    |    |    |
|-----|----|----|----|----|----|----|
| Key | 46 | 34 | 42 | 23 | 52 | 33 |
| loc | 6  | 4  | 2  | 3  | 2  | 3  |

|   |    |
|---|----|
| 0 |    |
| 1 |    |
| 2 | 42 |
| 3 | 23 |
| 4 | 34 |
| 5 | 52 |
| 6 | 46 |
| 7 | 33 |
| 8 |    |
| 9 |    |

Diagram showing insertion sequence for (c):  
 - 42 is inserted at index 2.  
 - 23 is inserted at index 3.  
 - 34 is inserted at index 4.  
 - 52 is inserted at index 5.  
 - 46 is inserted at index 6.  
 - 33 is inserted at index 7.

P) How many different insertions sequences of the key values using the same function and probing will result in the hash table shown above

- (a) 10 (b) 20 (c) 30 (d) 40

Sol:

|   |    |
|---|----|
| 0 |    |
| 1 |    |
| 2 | 42 |
| 3 | 23 |
| 4 | 34 |
| 5 | 52 |
| 6 | 46 |
| 7 | 33 |
| 8 |    |
| 9 |    |

Case(i): TO store 52  
 42 23 34 | 52 46 33  $\Rightarrow 3! = 6$   
 variable fixed

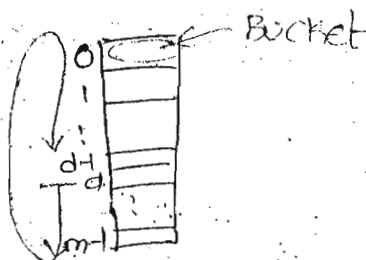
Case(ii): TO store 33  
 52 33  $\Rightarrow 4! = 24$   
 variable fixed  
 30

i.p. 42 23 34 | 52 33  
 (46)

Linear probing:

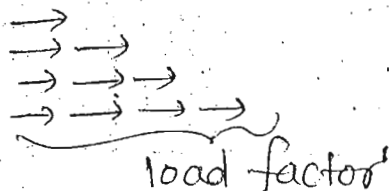
Sequence:  $d, d+1, d+2, \dots, m-1, 0, 1, \dots, d-1$

HDD:  $d$  (say)



Primary clustering:

The trend is for long sequence of preoccupied positions still becomes longer, primarily at one place.



Eg:

16: 6,  
 26: 6, 7,  
 36: 6, 7, 8,  
 46: 6, 7, 8, 9

|   |    |
|---|----|
| 0 |    |
| 1 |    |
| 2 |    |
| 3 |    |
| 4 |    |
| 5 |    |
| 6 | 16 |
| 7 | 26 |
| 8 | 36 |
| 9 | 46 |

Sn  
 bec  
 be

## Quadratic probing:

73

$$h + i^2$$

$$i = 1, 2, 3, \dots$$

disadvantage: Secondary clustering

The trend is for long sequence of preoccupied positions still become longer, primarily at different places

Eg:

| Key                   | 16 | 26 | 36 | 46 |
|-----------------------|----|----|----|----|
| loc                   | 6  | 6  | 6  | 6  |
| $H(00) = 11 \cdot 10$ |    |    |    |    |

$$H(16) = 6$$

$$H(26) = 6$$

collision

say  $h = 6$

$$i=1 \Rightarrow h + i^2 = 6 + 1^2 = 7 \cdot 10 = 7$$

$$H(36) = 6$$

collision

say  $h = 6$

$$i=1 \Rightarrow h + i^2 = 6 + 1^2 = 7 \cdot 10 = 7$$

collision

$$i=2 \Rightarrow h + 2^2 = 6 + 4 = 10 \cdot 10 = 0$$

$$H(46) = 6$$

collision

say  $h = 6$

$$i=1 \Rightarrow h + i^2 = 6 + 1^2 = 7$$

$$i=2 \Rightarrow h + 2^2 = 6 + 2^2 = 10$$

$$i=3 \Rightarrow h + 3^2 = 6 + 9 = 15 \cdot 10 = 5$$

→ In quadratic probing all the buckets are not compared because of quadratic nature. So Linear probing is better

|   |    |
|---|----|
| 0 | 36 |
| 1 |    |
| 2 |    |
| 3 |    |
| 4 |    |
| 5 | 46 |
| 6 | 16 |
| 7 | 26 |
| 8 |    |
| 9 |    |

## Random probing:

RNG: Random Number Generator

↓  
No number should be repeated within  
random space

$$Y_{\text{new}} = Y_{\text{old}} + C$$

$Y_{\text{new}} = \text{seed value}$

~~Yold~~  $C = \text{constant}$

$m = \text{Hash table}$

Let  $Y_{\text{new}} = 9$

$C = 4$

$m = 11$

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

$$H(x) = x \% 11$$

| Key | 17 | 28 | 39 | 50 |
|-----|----|----|----|----|
| loc | 6  | 6  | 6  | 6  |

|    |    |
|----|----|
| 0  |    |
| 1  |    |
| 2  |    |
| 3  | 39 |
| 4  |    |
| 5  |    |
| 6  | 17 |
| 7  | 50 |
| 8  |    |
| 9  |    |
| 10 | 28 |

17: 6

28: 6, 10

39: 6, 10, 3

50: 6, 10, 3, 7

Goal: change 'c', get new sequence. This is applied in double hashing.

74

## Double hashing / Rehashing

$$H_1(x) = x \% m \text{ (given)}$$

if collision

$$y = y + c \% m$$

you need 'c'

$$H_2(x) = [x \% (m-2)] + 2 \text{ (given)}$$

Eg:

| key | 17 | 28 | 39 | 50 |
|-----|----|----|----|----|
| loc | 6  | 6  | 6  | 6  |

$$H_1(17) = 6$$

$$H_1(x) = x \% 11$$

$$H_2(x) = (x \% 9) + 2$$

$$H_1(28) = 6 = \text{Yold}$$

collision

we need c

$$c = H_2(x)$$

$$H_2(28) = (28 \% 9) + 2$$

$$= 1 + 2 = 3$$

$$\therefore c = 3$$

$$Y_{\text{new}} = Y_{\text{old}} + c$$

$$= 6 + 3$$

$$= 9 \% 11 = 9$$

$$H_1(39) = 6 = \text{Yold}$$

collision

we need c

$$c = H_2(39)$$

$$H_2(39) = (39 \% 9) + 2 = 5$$

$$Y_{\text{new}} = Y_{\text{old}} + c$$

$$= 6 + 5 = 11 \% 11 = 0$$

|    |    |
|----|----|
| 0  | 39 |
| 1  |    |
| 2  | 50 |
| 3  |    |
| 4  |    |
| 5  |    |
| 6  | 17 |
| 7  |    |
| 8  |    |
| 9  | 28 |
| 10 |    |

$$17: 6$$

$$28: 6, 9$$

$$39: 6, 0$$

$$50: 6, 2$$

clustering is reduced

$$H_1(50) = 6 = \text{old}$$

we need  $C = H_2(x) = H_2(50)$

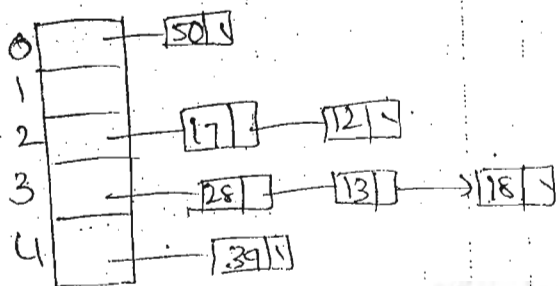
$$H_2(50) = (50 \cdot 9) + 2 = 7$$

$$Y_{\text{new}} = 6 + 7 = 13 \cdot 11 = 2$$

chaining:

| Key | 17 | 28 | 39 | 50 | 12 | 13 | 18 |
|-----|----|----|----|----|----|----|----|
| loc | 2  | 3  | 4  | 0  | 2  | 3  | 3  |

$$H(x) = x \cdot 5$$



open addressing

disadvantage:

- Collided records require more probes
- Deletion is not possible
- overflow problem

chaining

advantages

- Less probes
- possible
- NO overflow problem