Data * Structure

25-4-2012

Data structure

1) Defination

a logical or mathematical model.

Ex:- stack

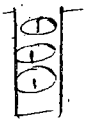
physical structure

1) Defination

implementation of model in physical memory

Array stack

Linked stack



Datatypes :- problem Solving

Q. Why data structures ...?

Sol Solving a problem : FAST by occupying optimum memory

FAST → time analysis

optimum memory → space analysis

ARRAYS

Q. Calculate the location of an element A(0) in an array of a : array [-5 ..... +5] elements and starting location is 1000 and each element occupies 2 memory locations.

Sol

upper boundary − (lower boundary) + 1

⇒ +5 − (−5) + 1

⇒ 11

char 1
int 2
float 4
double 8
long double 10

$Loc(A[i]) = L_b + (i - L_b) * C.$

$A(0) = 1000 + [0 - (-5)] * 2 \Rightarrow 1000 + 10$

Address arithmatic

$$loc(a[i]) = l_o + (i - \text{lower boundary}) * C$$

10	11	12	13
a_3	a_4	a_5	a_6

$loc(A[6]) \rightarrow 10 + (6-3)$

$\Rightarrow 13.$

char $\rightarrow 10 + (1)_{x1} = 11$

int $\rightarrow 10 + (1)_{x2} = 12$

float $\rightarrow 10 + (1)_{x4} = 14$

double $\rightarrow 10 + (1)_{x8} = 18$

long double $\rightarrow 10 + (1)_{x10} = 20$

Java Supports a UNI code. It is a Super Code of ASCII

- UNI code is universal code.

$$loc(A[i]) = l_o + (i - l_b) * C.$$

$A = \text{array}[l_b \ldots u_b)$ of elements.

l_o = Starting location

l_b = lower boundary

C → Count

Row major order ↑ Column major order ↑

Q. Calculate the location of A(0,5) element using i) RMO ii) CMO for A = array [-2 .... 2, 3 ... 7] of elements and starting location is 1000 and each element occupies 2 memory cells.

Sol

2-D arrays

$A = \text{array}[\underbrace{l_{b_1} \ldots u_{b_1}}_{i}, \underbrace{l_{b_2} \ldots u_{b_2}}_{j}]$ of element

i → Row

j → Column

$u_{b_1} - l_{b_1} + 1 \quad \ldots \quad u_{b_2} - l_{b_2} + 1$

$r_1 \times r_2$

* Scientific Notation

for RMO

$$\text{loc. } A(i,j) = \text{lo} + \left[(i - lb_1)(ub_2 - lb_2 + 1) + (J - lb_2)\right] \times C.$$

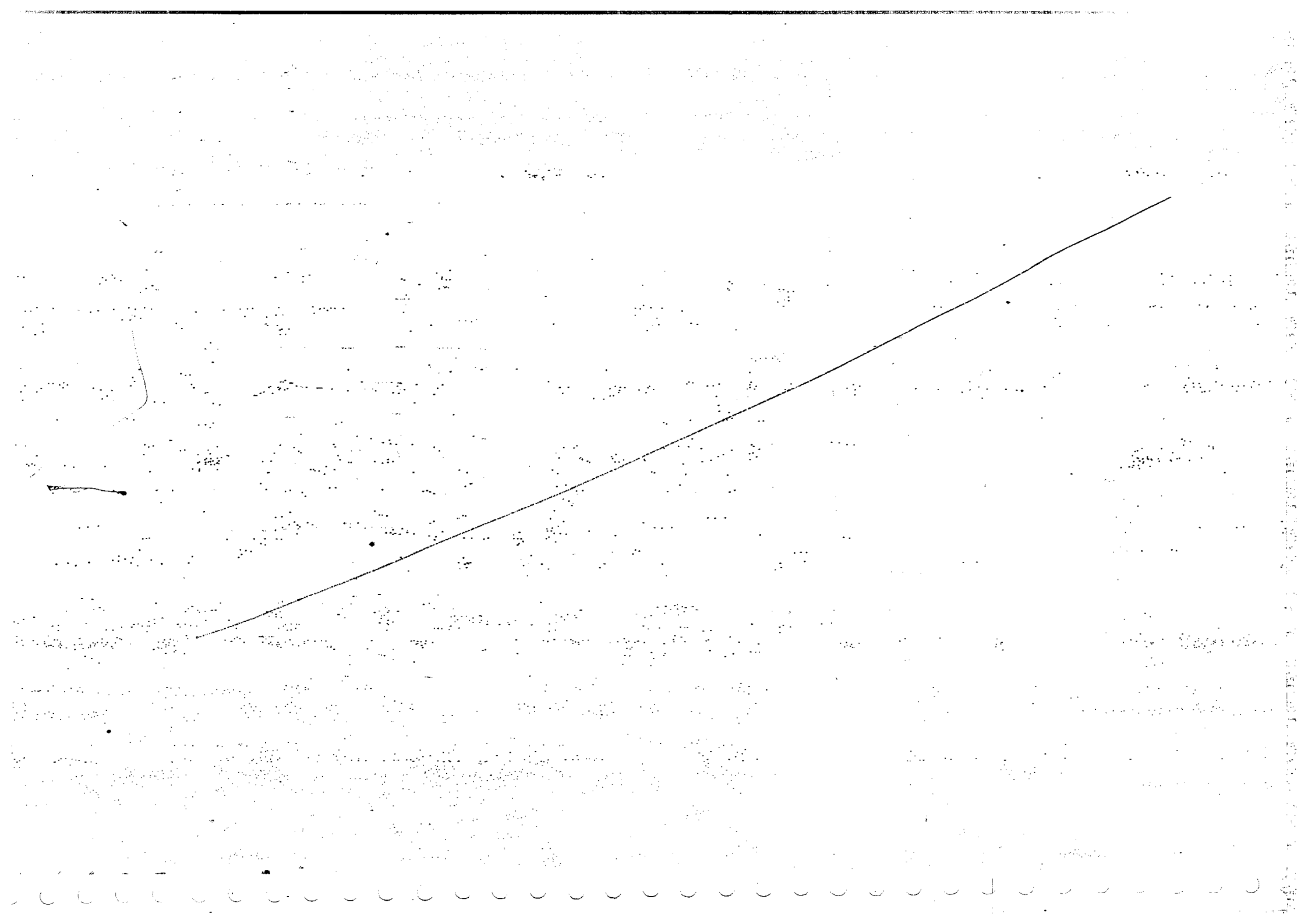
$$\text{loc } A(0,5) = 1000 + \{[0-(-2)](2-3+1) + (5-3)\} * 2.$$

$$\text{loc. } A(0,5) = 1024.$$

for CMO

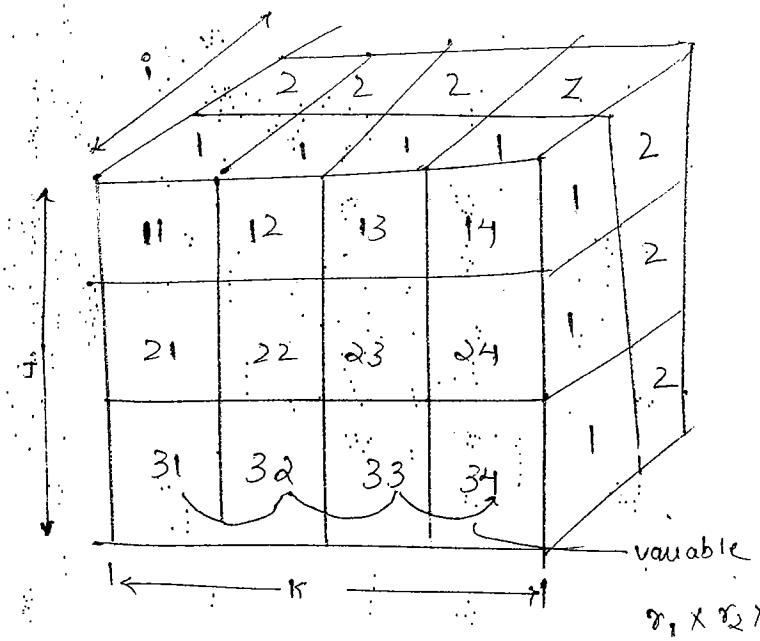
$$\text{loc } A(i,j) = \text{lo} + \left[(J - lb_2)(ub_1 - lb_1 + 1) + (i - lb_2)\right] * C$$

$$\text{loc } A(0,5) = 1024.$$



Date
27-4-2012

3-Dimensional array



$r_1 \times r_2 \times r_3$.

3D RMO ⇒ $Lo + (i-1) r_2 \times r_3 + (J-1) r_3 + (k-1)$.

$2 \times 3 \times 4$.
planes × rows × Columns
i × J × k

- 3.D Column measure order (CMO) ⇒.

$$Lo + (K-1) r_1 r_2 + (J-1) r_1 + (i-1)$$

- 2D RMO ⇒ $Lo + (i-1) r_2 + (J-1)$
- 2D CMO ⇒ $Lo + (J-1) r_1 + (i-1)$

• STACK

ADT → Abstract data type

ADT of stack

• Defination :- One side open, the other side is closed.

→ LIFO or FILO model.

→ Top pointer :- pointing to top most element among the available elements.

• Operations :-

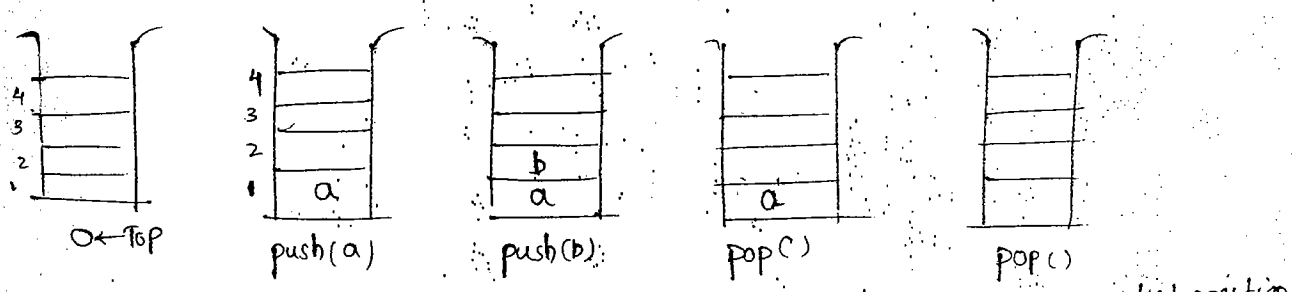
push(x) : insert an element 'x'

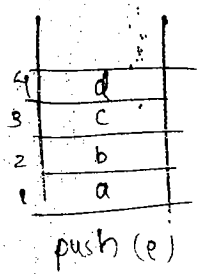
pop(x) : deletes the topmost element.

• Application :-

i) permutation ii) polish Notation iii) Recursion.

Abstraction : hiding the internal implementation details.





Stack

push = insert

① OF = size

② update

③ insert

pop = delete

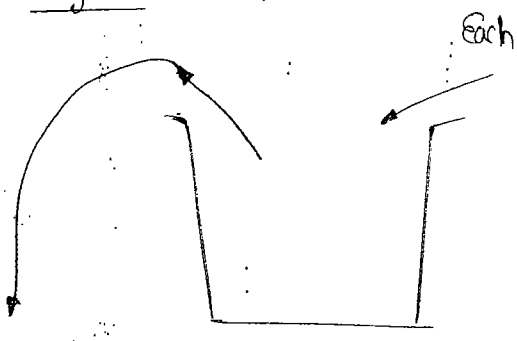
① OF = initial

② delete

③ update

i) permutations :-

Logic :

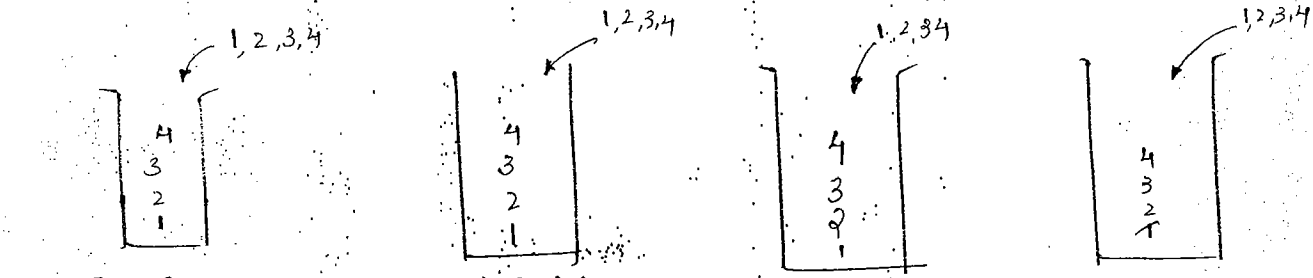


- Based on the desired sequence, the elements are popped out

Q) identify invalid stack permutation for the input sequence 1, 2, 3, 4.

a) 3, 4, 2, 1 b) 4, 3, 1, 2 c) 2, 4, 3, 1 d) 1, 4, 3, 2 e) 1, 4, 3, 5, 2

Sol a) 3, 4, 2, 1 b) 4, 3, 1, 2 c) 2, 4, 3, 1 d) 1, 4, 3, 2



Q) identify valid/invalid stack operations

a) Is empty ( ) → return true/false

b) Get top ( ) → return top most element

i) Is empty (pop (push (push (s, x), y))) = True (false → valid) → invalid

ii) Get Top (push (pop (push (pop (push (s, a), b)), c))) = a (c → valid) → invalid

ii) polish notation :-

→ Devised by polish logician Lukasiniczi.

→ it avoids the repeated scanning of infix expression

→ Compilers prefer { prefix, postfix } in one scan, we get result

INFIX → a+b

prefix → +ab

postfix → ab+

} Names w.r.t operator.

a+b ≠ +ba. This is called parsing error ; +ab = valid.

Rule ① :- The relative position of Operands : Not disturbed.

Rule ② :- The relative position of Operators : Disturbed according to the precedence & associativity rules.

Ex :- i) a+b*c : (precedence example)

prefix :- a+b*c

a+(*bc)

+a(*bc)

+a*bc

post :- a+b*c

bc*

a+(bc*)

a(bc*)+

abc*+

ii) a*b/c (left associative)

prefix :- a*b/c

(a*b)c

*ab/c

/*abc

post :- a*b/c

(a*b)/c

ab*/c

ab*c/

ii) Right associative

a) prefix

$a \uparrow b \uparrow c$

$b \uparrow c$

$\uparrow bc$

$a \uparrow$ ( )

$\uparrow a$ ( )

$\uparrow a \uparrow bc$

b) postfix

$a \uparrow b \uparrow c$

$b \uparrow c$

$bc \uparrow$

$a \uparrow$ ( )

$abc \uparrow \uparrow$

Q.) i) $-b$ ii) $\log x$ iii) $x!$ iv) $\log x!$

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

Sol:

infix	$-b$	$x!$	$\log x$
prefix	$-b$	$!x$	$\log x$
postfix	$b-$	$x!$	$x \log$

prefix

$\log x!$

$x!$

$!x$

$\log$ ( )

$\log$ ( )

$\log !x$

$a + b * c$

$b * c$

$* bc$

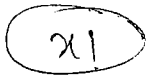
$a +$ ( )

$+ a$ ( )

$+ a * bc$

post

log x!



x! .

log

x! log

a+b*c

b*c

28.4.2012.

/* : same level : left associativity

a = -b + c x d/e - f ↑ g ↑ h + i * J.

a/b*c

Sol

precedance table.

pref

a/b*c.

H

L

- unary

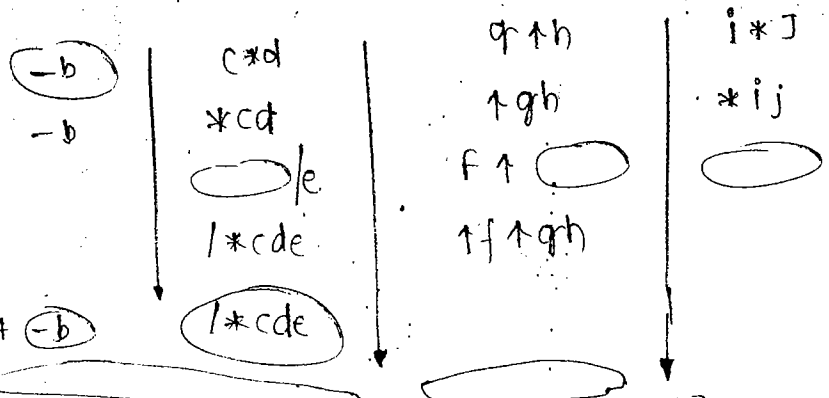
↑ : Right

*/ : Left associative

+ - : Left associative.

=

i> prefix



=> = a + - + - b / * c d e ↑ f ↑ g h * i j

ii) postfix.

-b	c*d	g↑h	i*J
b-	cd*	gh↑	iJ*
	b/e	f↑b	
	b/c	F b↑	
	cd*e/	fgh↑↑	

b- cd*e/+ fgh↑↑ iJ*+

ab-cd*e/+fgh↑↑ij*+=

---

a = -b+c*d/e↑f↑g↑h.+i*j ⇒ infix

~~a = b~~

ii) prefix.

=a+-+-b/*cde↑f↑gh *ij

ii) postfix.

ab-cd*e/+fgh↑↑-iJ*+=

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

Q.) Give the stack top two elements after first * is evaluated

a) 2, 3 b) 5 1 c) 1, 2 d) 16

Q) what is the final result c entire expression is evaluated.

a) 12 b) 13 c) 14 d) 15

Q.) Which of the following not possible stack content.

a) | 8 | 2 | 3 | b) | 7 | 5 | 1 | c) | 1 | 2 | 3 | * | d) | 1 | 6 |

Evaluation

6 - 2 = 4

(6 - 2) prefix

− 6, 2

(6 − 2) postfix

6, 2, −

Concept

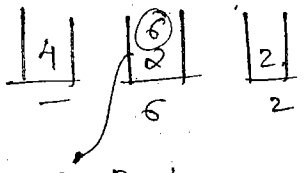
Operand : push

Operator

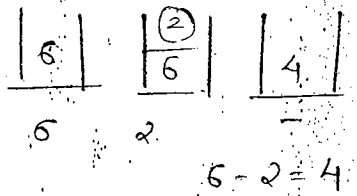
(not pushed in) i) pop

ii) Evaluation

iii) push result



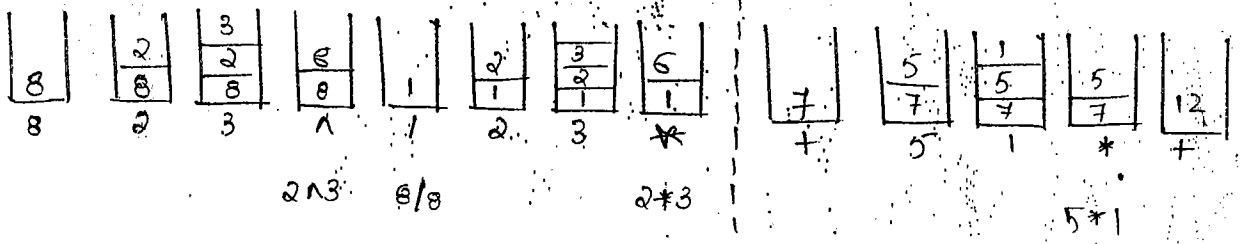
6 − 2 = 4



6 − 2 = 4

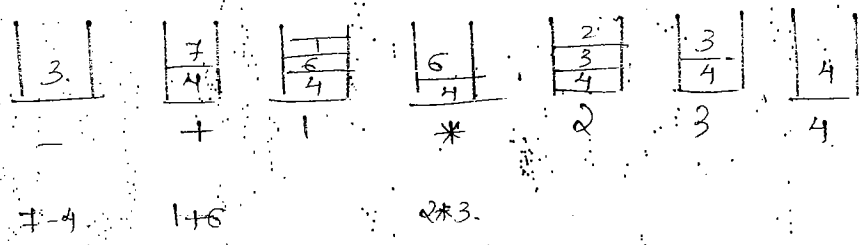
Sol: 8, 2, 3, ∧, /, 2, 3, ∀, +, 5, 1, *, +

i) post fix:

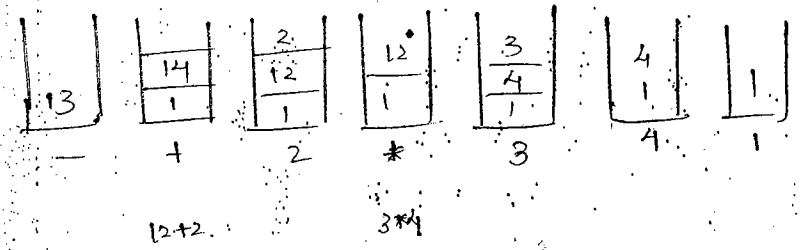


2∧3 8/8 2*3 5*1

Ex:- i) − + 1 * 2 3 4 (postfix evaluation)



ii) −, +, 2, *, 3, 4, 1 (prefix evaluation)



• (RECURSION)

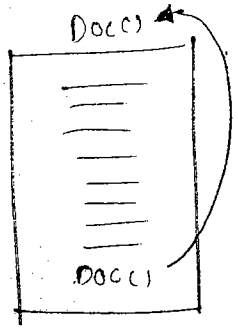
i) Direct Recursion.
   a) Tail Recursion Ex:- factorial
   b) head Recursion Ex:- Tower of Hanoi (ies)

ii) Indirect Recursion : Ex:- Trees.

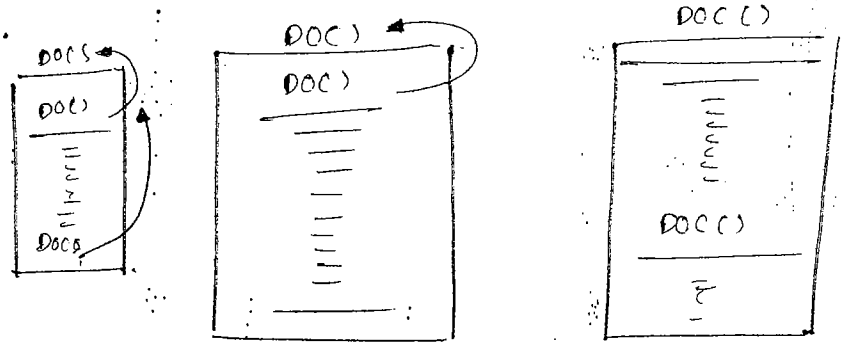
iii) Nested Recursion Ex:- Ackermann's function (JTO) (DRDO)

iv) Excessive Recursion Ex:- fibonacci.

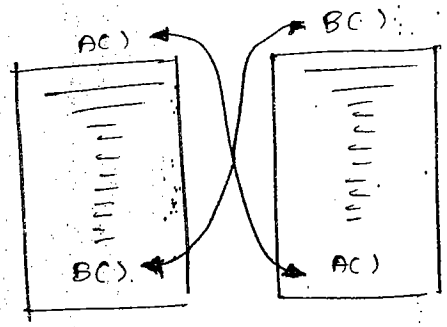
i) Tail Recursion



ii) Head Recursion (or other than last)
(non Tail)



iii) indirect Recursion



Imp iv) Excessive

- Doesn't remember previously evaluated values.

properties

i) Base (or) Termination Condition (or) Anchor step.

ii) Repeated step (or) Recursive step (or) inductive step.

$$fact(N) \begin{cases} 1 & N=0 \text{ Anchor or Base} \\ N * fact(N-1) & \text{otherwise inductive step (Repeated)} \end{cases}$$

```
int fact (int N)
{
  if (N==0)
    Return 1;
  else Return N * fact(N-1);
}
```

$f(4) = 4 * f(3)$

$3 * f(2)$

$2 * f(1)$

$1 * f(0)$

$4 * 3 * 2 * 1 * 1 = 24.$

Q)

```
Void Do (int N)
{
  if (N>0)
  { DO(N-1)
    PRINT :N
    DO (N-1);
  }
}
```

What is the o/p if DO(3)....?

Sol

N = 3 > 0

{

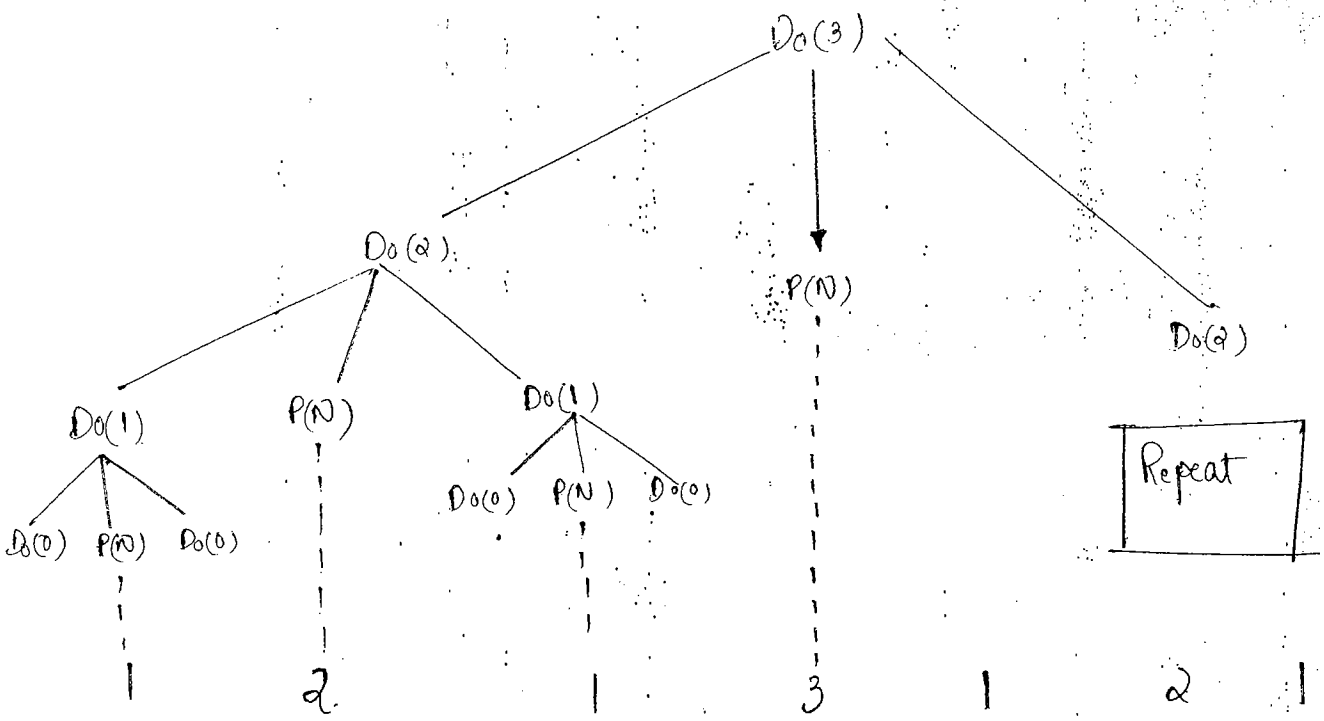
2

2

1

1

}



∴ Ans is 1 2 1 3 1 2 1

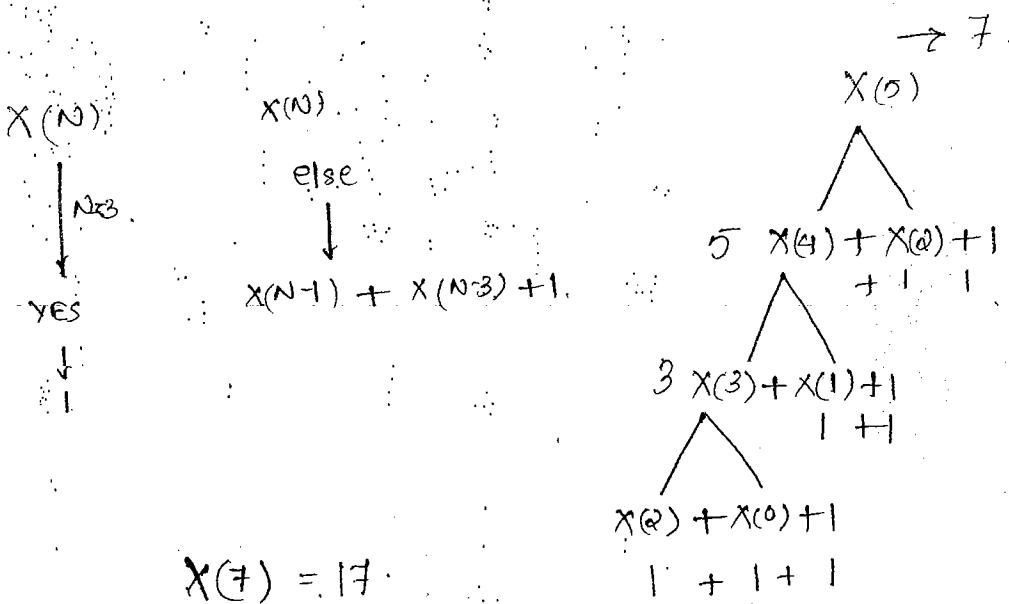
$$fibonacci(N) = \begin{cases} 0, & N=0 \\ 1, & N=1 \\ fib(N-1) + fib(N-2), & N>1. \end{cases}$$

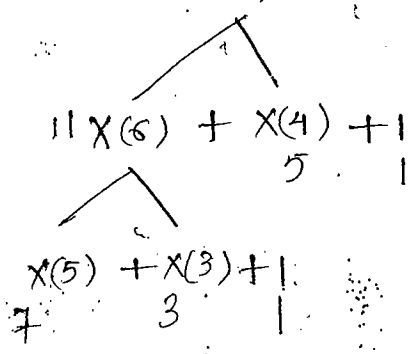
Q> what is the o/p X(X(5)) ---?

```
int X(int N)
{ if N<3
    Return 1
  else
  Return X(N-1) + X(N-3) + 1;
}
```

30.4.2012

Sol





$X(X(5)) = X(7) = 17$

Return value = 17

No. of Calls =

$X(5) = 7$

$X(7) = 17$

24.

Q.) what is the value of F(6)...?

How many additions are there in evaluating f(7)...?

f → fibonacci function

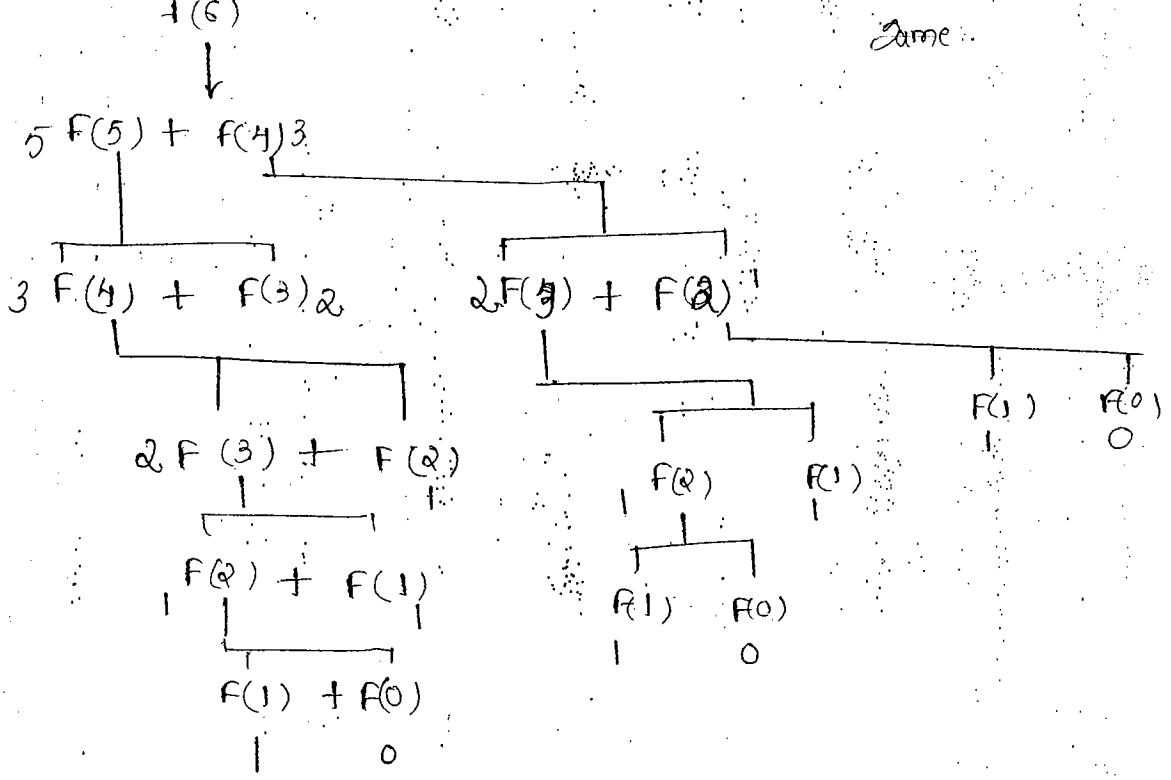
Given:

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

Sol

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

8.



- Calls & invokations both are same.

N	0	1	2	3	4	5	6	7	8
F(N)	0	1	1	2	3	5	8	13	21
Calls.				5	9	15	25	41	
Additions				2	4	7	12	20	

Excessive :- Doesnot Remember the previously evaluates value.

No. of Calls in $F(N) = 2 \cdot F(N+1) - 1$

No of additions $F(N) = f(N+1) - 1$

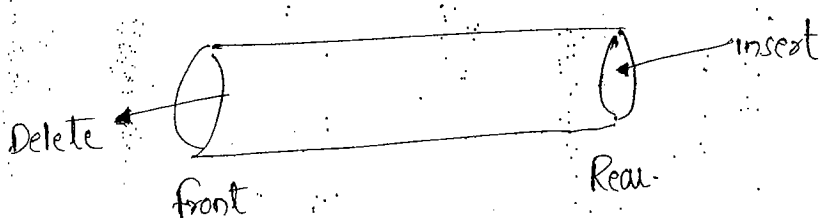
• QUEUE

→ FIFO (or) LILO model.

→ Associated with Two pointers.

i) Rear pointer = insert an element 'x' → enqueue(x);

ii) front pointer = deletes the front element → dequeue();



Q) what is the status of 'Q' after the following sequence of steps. on a linear Queue.

i) addition of 'a' and then 'b'

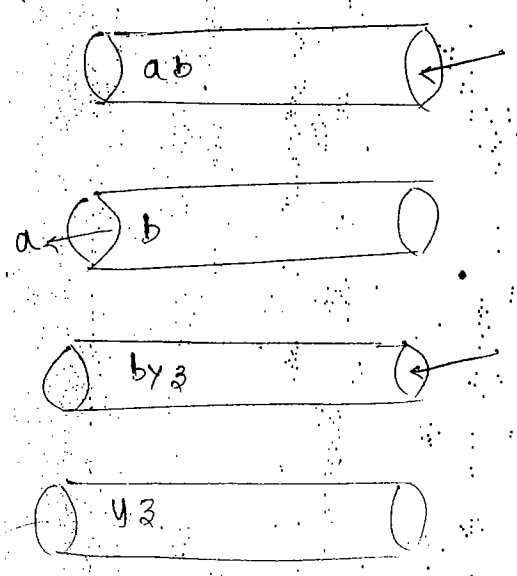
ii) a deletion

iii) addition of y and then z

iv) a deletion

a) Error b) byz c) byz d) ab.

Sol



'a' deletion done not means that delete a → it means that delete the first element.

Ans :- yz.

Q) what does Do( ) do on linear Queue.

algorithm Do (linear queue).

→ if Q is not Empty

Step(1) insert temparary = dequeue ( )

Step(2) Do ( ) ;

Step(3) enqueue (temp);

Option:- a) Q Contents remain intact

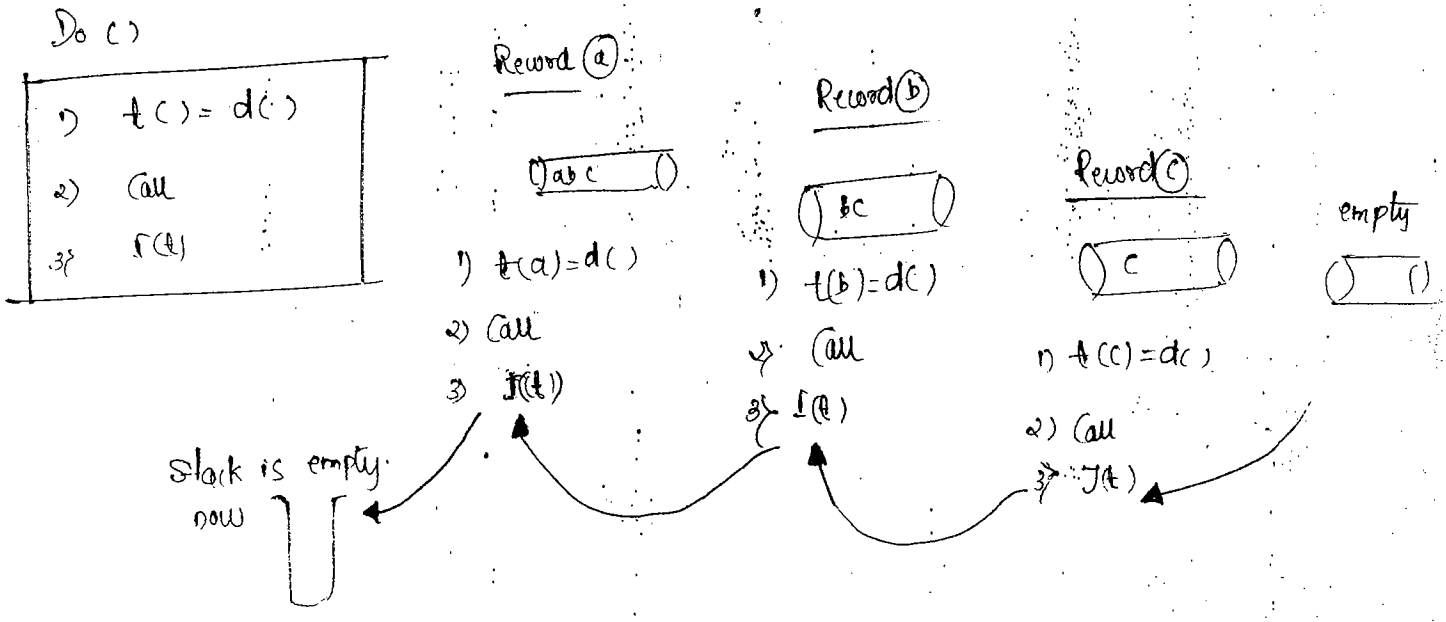
b) Q Contents got reversed.

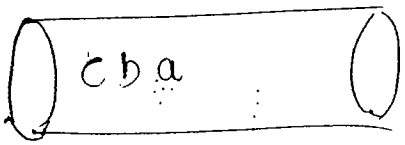
c) Q becomes Empty

d) Q Contents doubled.

Sol

→ Reursion invokes stack implicitely





Q got reversed Contents ∴ Ans (B)

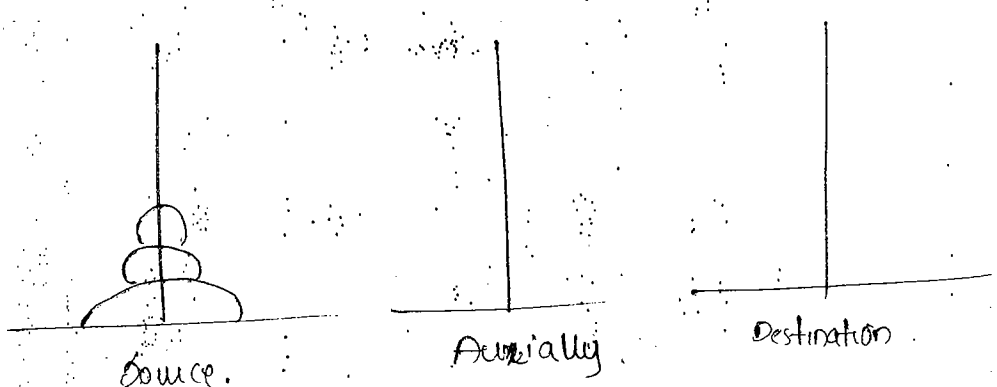
Observation :

In program termination = ϕ is Empty

In Execution termination = Stack is Empty

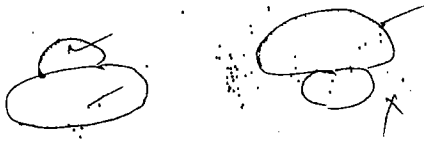
Repeated

Towers of Hanoi



Rule :→ Only one disc is allowed to lift & no restriction for direction

Observation →

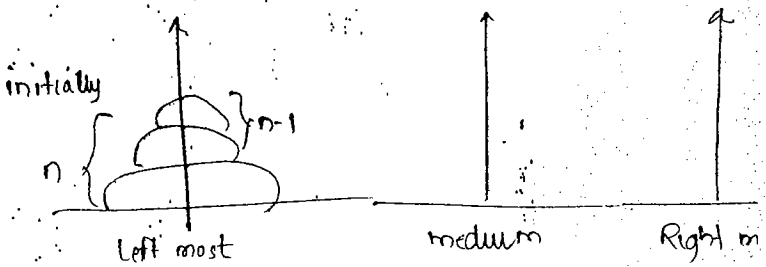


i) allowed ii) Not allowed

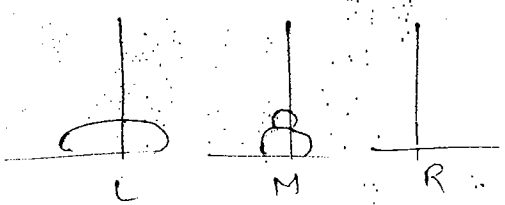
Recursion (Stack application)

Algorithm (Recursive)

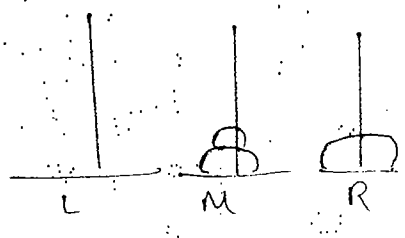
1) Lift (N-1) discs from L to M.
2) Actual move from L to R
3) Lift (N-1) discs from M to R



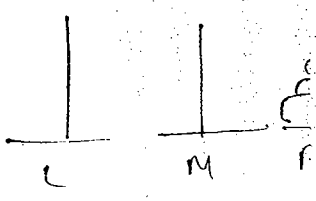
1)

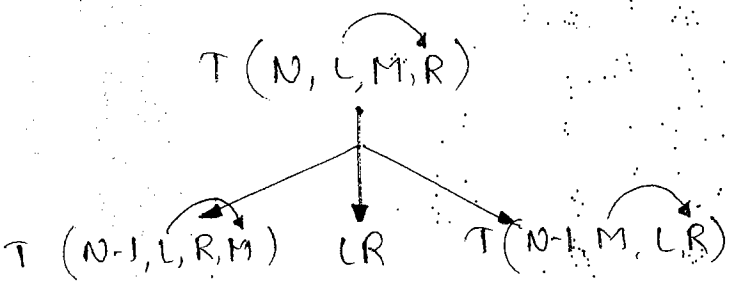


2)

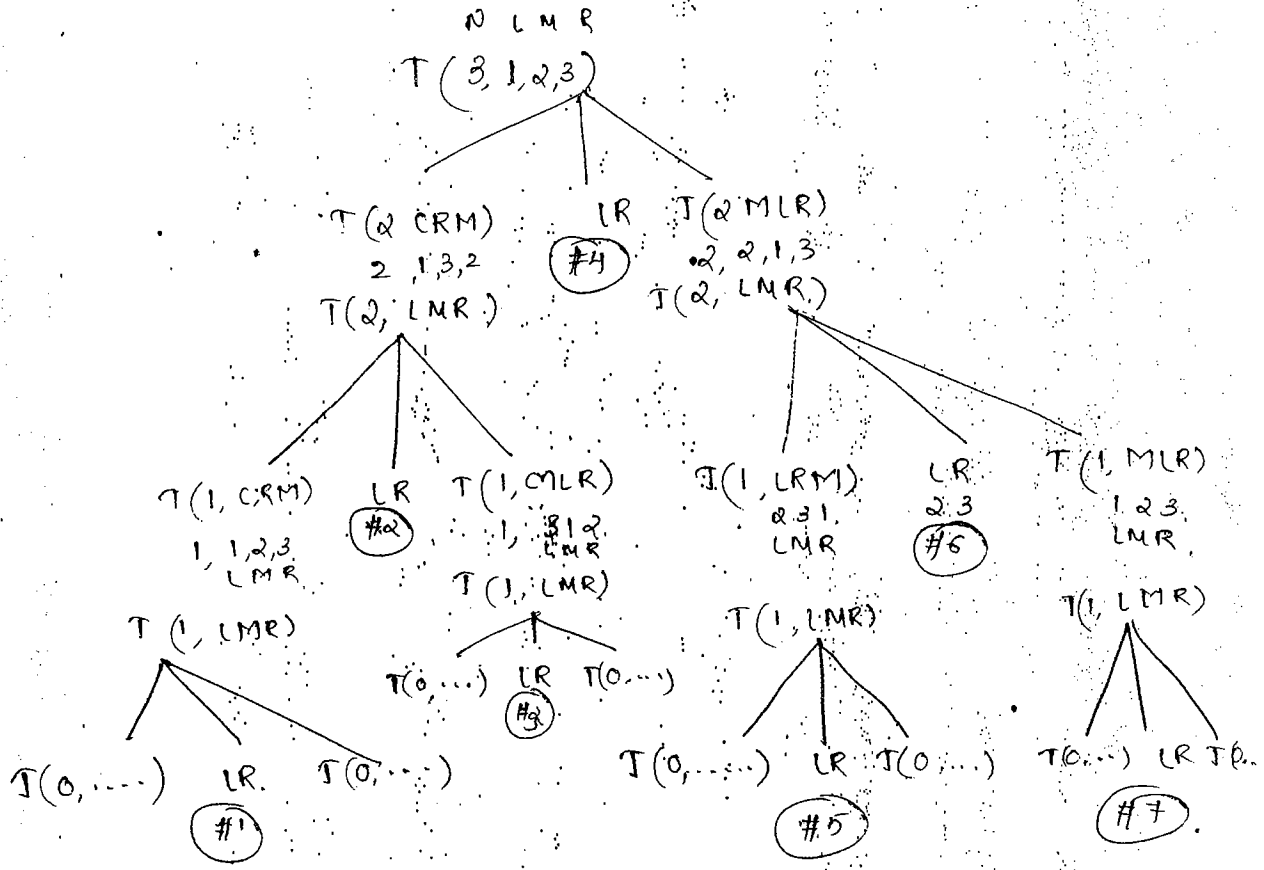


3)

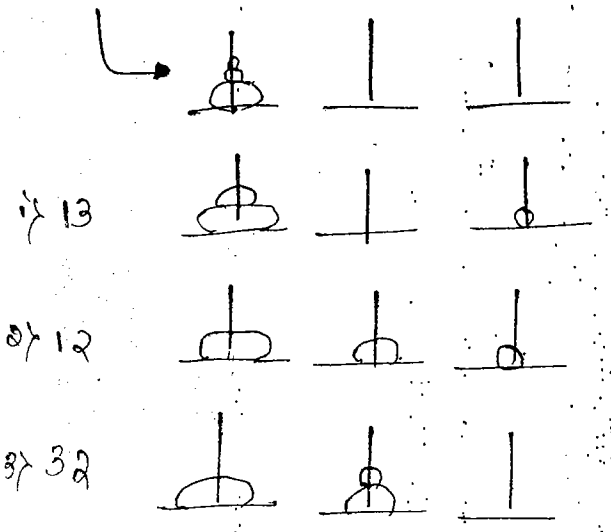


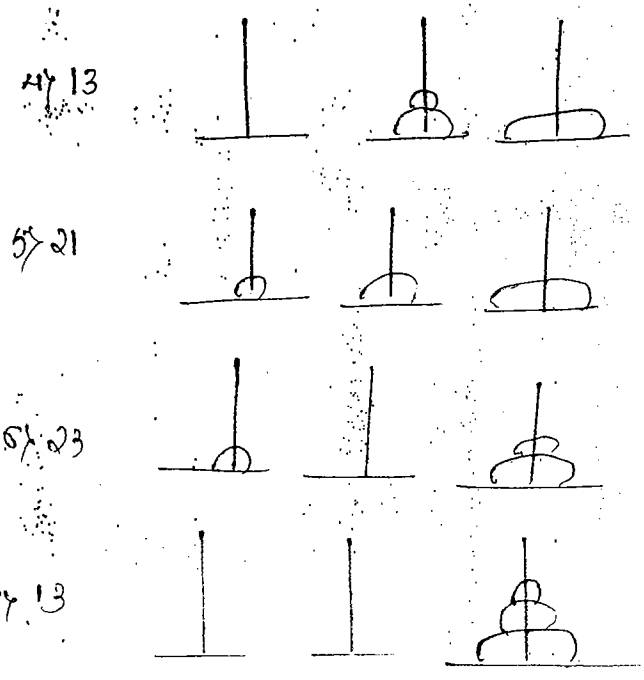


Example :-

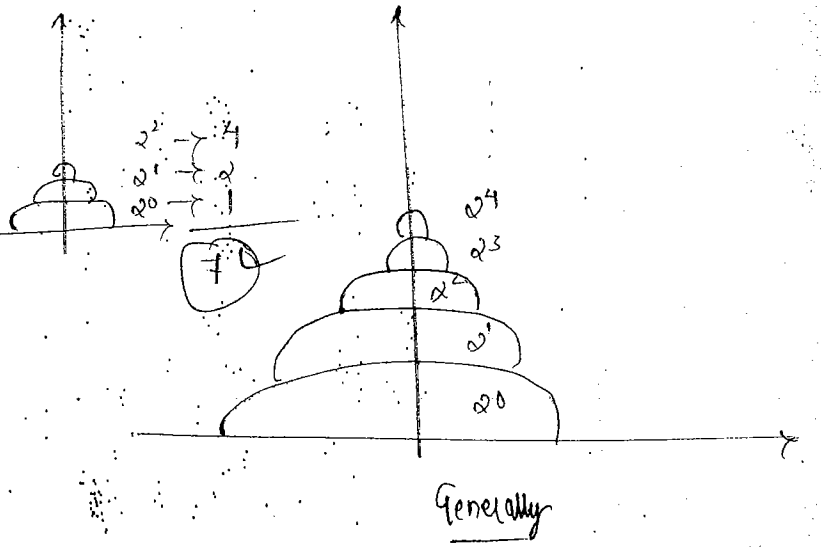


Initially





moves	$2^N - 1$
3	$7 = 2^3 - 1$
4	$15 = 2^4 - 1$
5	$31 = 2^5 - 1$



Generally

Invocation or Calls	$(N+1)$. $2^{N+1} - 1$
3	$15 = 2^4 - 1$
5	$31 = 2^5 - 1$

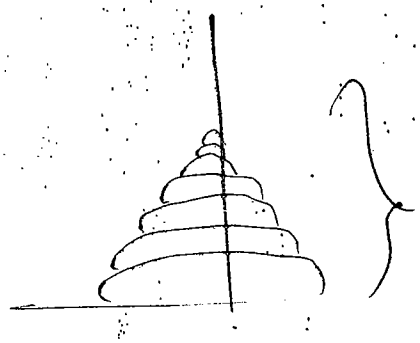
Observation

- Recursion tree traversal = pre order.
- Recursion is user friendly

Q1) No. of moves $= 2^N - 1 \Rightarrow 7$ here $n=3$

Q2) No. of calls $= 2^{N+1} - 1 = 2^4 - 1 = 15$

Q3)



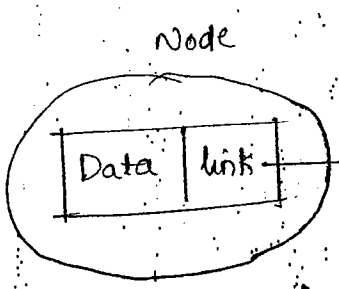
Each disc is moves. . . . ?

2^i where $i = 0, 1, 2, 3, 3 \ldots$

1-5-2012

LINKED LISTS

array	pointer
static	Dynamic
Early binding	late binding
Compile time binding	Runtime binding
ex:- int a[]; invalid (Size)	ex:- int *p;
int a[4]; ✓	

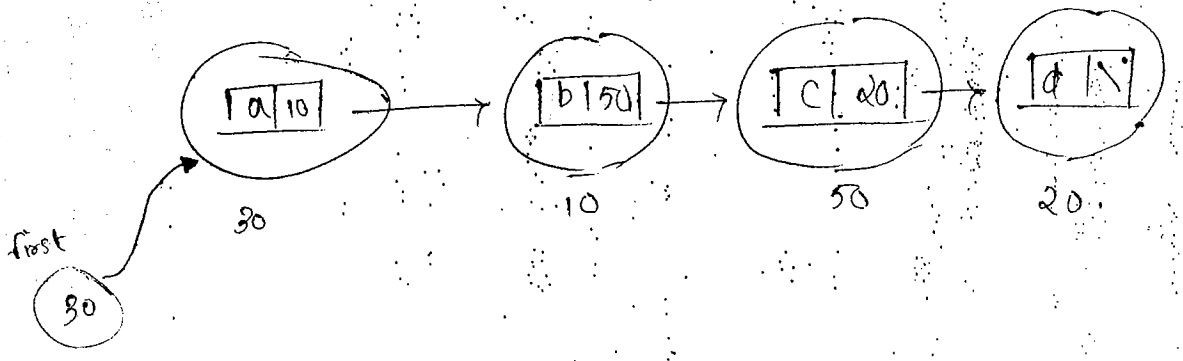


LINKED LISTS.

Data structure :- Node in a self Referential structure.

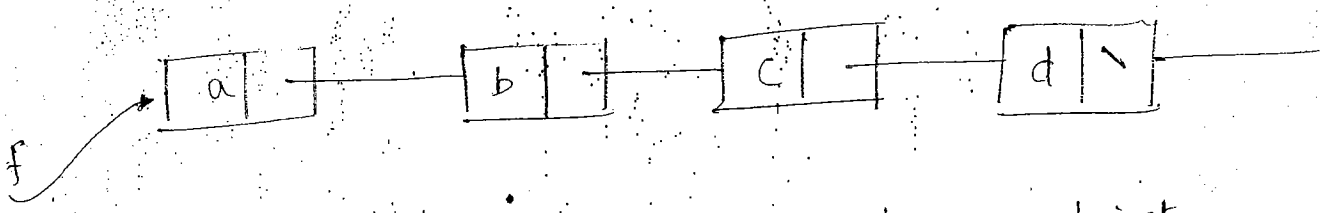
→ Linked list :- Nodes are logically adjacent, physical Scattered.

• array is a physically adjacent & is Contiguous i.e Continuous.

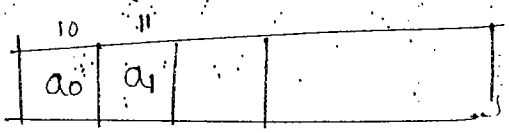


pictorial

• f is the pointer variable holding the address of very first node



Array = Contiguous = Continuous ; Array is physically adjacent.



Single Linked list (SLL) (disadvantage)

1) stepping backwards is not possible (→ search(c) → delete(c))

(we need to update downstream address in the upstream Node)

2) The link part of the last node : not utilized.

Hence Single linked list has become now Circular linked list.

Circular linked list (CLL)

• advantage of CLL.

1) stepping backwards is ~~not~~ possible by moving f

2) The last part of the node : utilized.

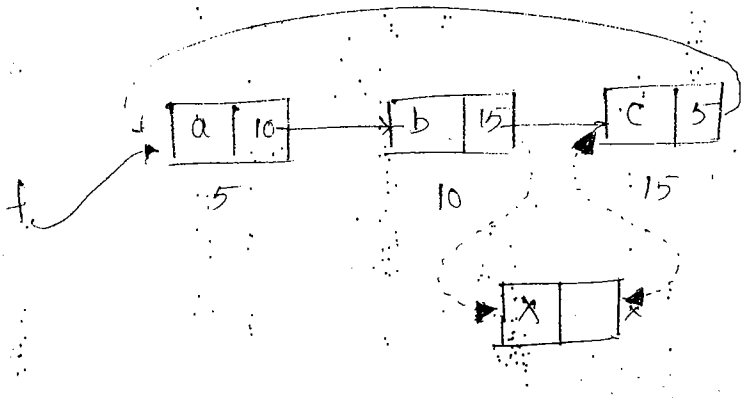
•disadvantage

1) Danger of falling into infinite loop.

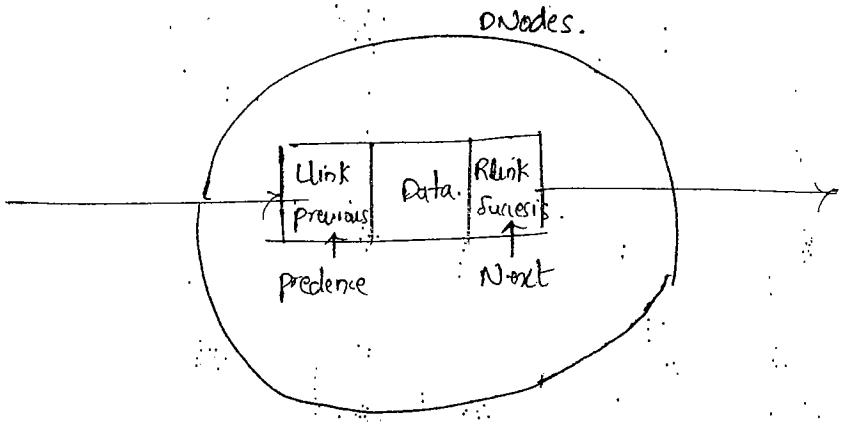
Solution :- 1) head node is attached in the front. or Tail node is appended.

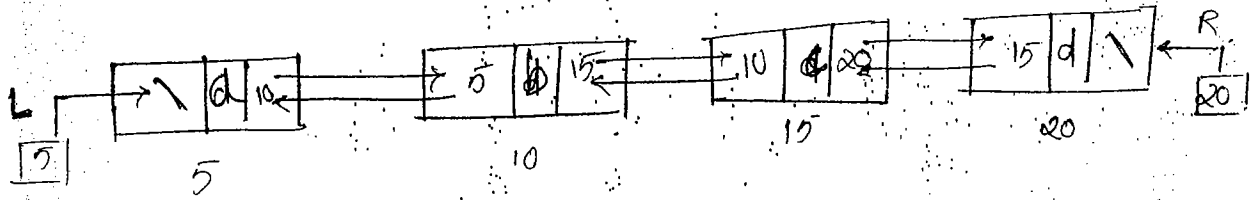
Q. (res 2009) inserting a new Nodes in a CLL requires the modification of _____ no. of link fields.

a) 1 b) 2 c) 3 d) 4.



Doubly linked lists: DLL

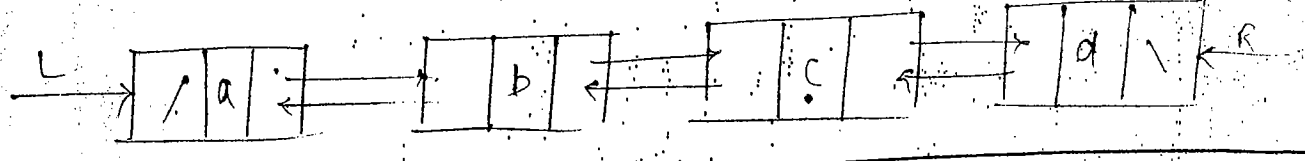




L → address of left most Node.

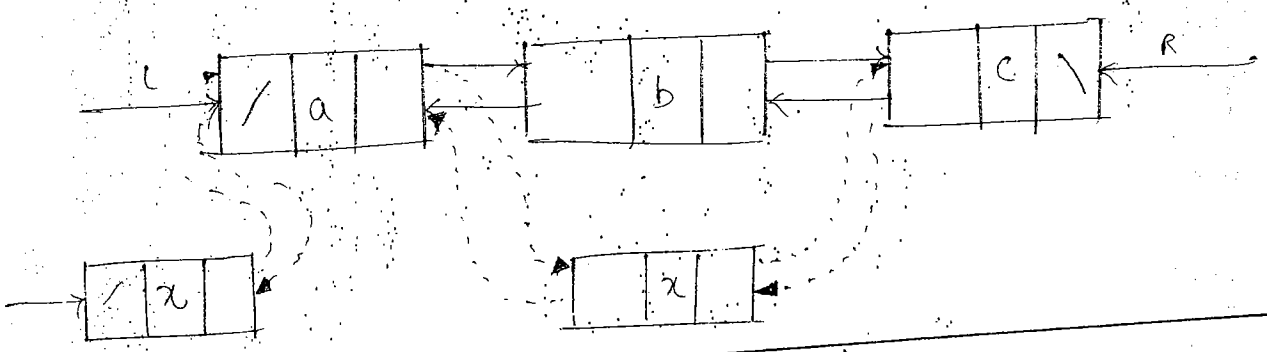
R → address of Right most Node.

pictorial :-

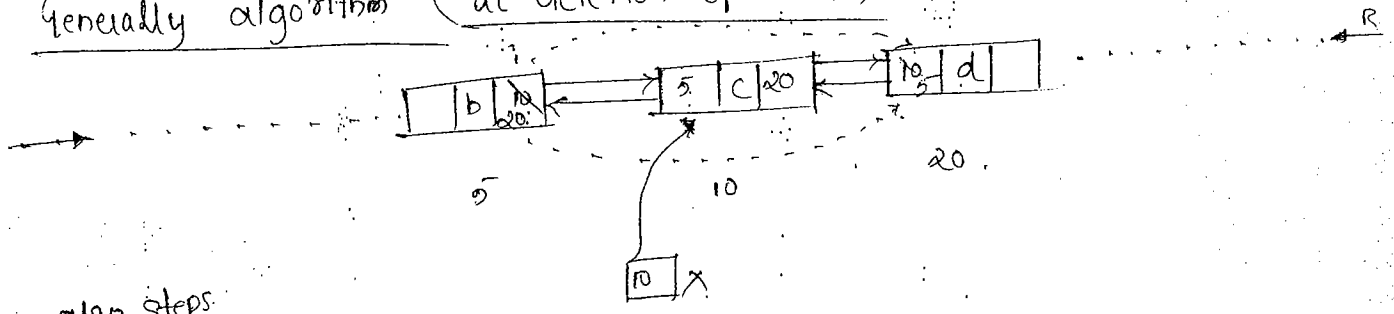


Operation	left most	Right most	else middle
insert	3	3	4
Delete	1	1	2.

Q.1



Generally algorithm (at deletion operation)



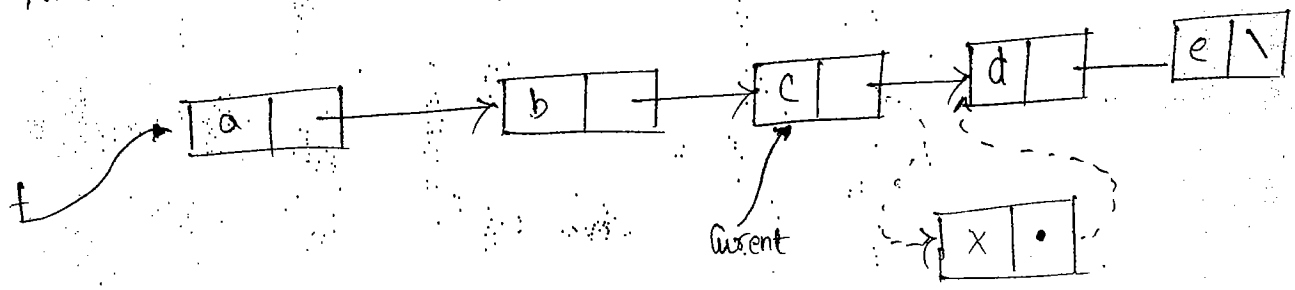
also steps

i) Rptr (Lptr(X)) ← Rptr(X)

ii) Lptr (Rptr(X)) ← Lptr(X)

X- address of Node being deleted in DLC.

Q) Which of the following option inserts X after Current



a) Cur ← new Node (X, Cur)

b) Cur ← New Node (X, Cur.Next)

c) Cur.next ← New Node (X, Cur);

d) Cur.next ← NewNode (X, Cur.next);

Del algorithm

1) Create a Node
2) place X
3) update link part of New Node
4) update link part of Current Node.

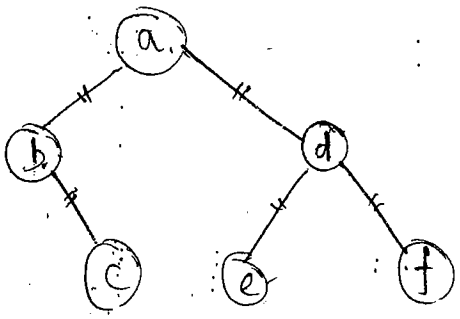
Observation :-

UNIX editor = vi (visual) early version was developed by using Circular doubly linked list

Non linear linked lists :-

→ Binary tree.

atmost two children.



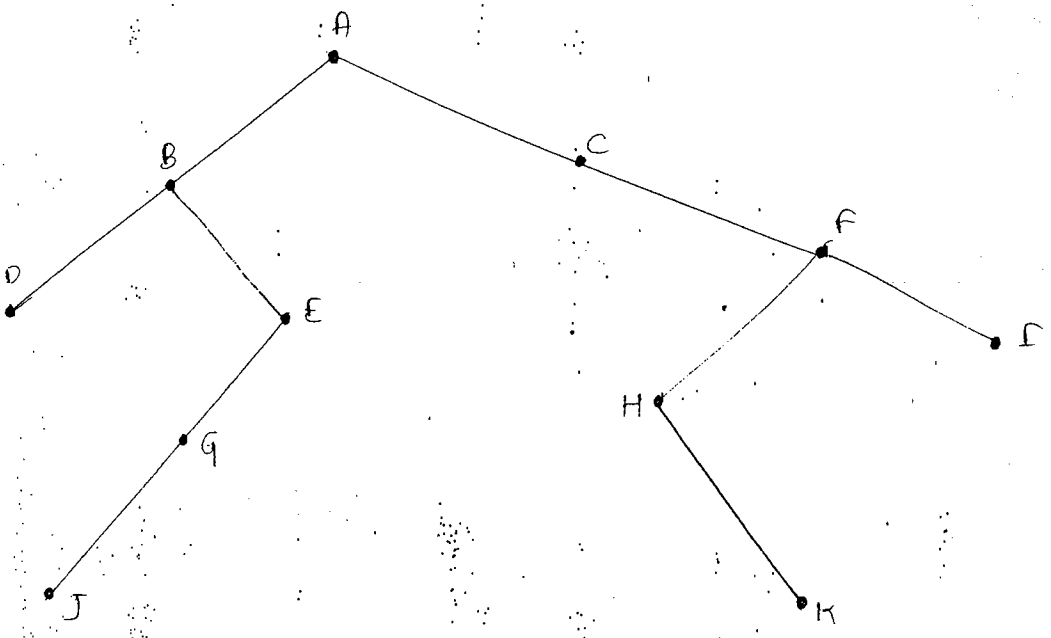
$n = n_0 + n_1 + n_2$

$n_0 = 3$
$n_1 = 1$
$n_2 = 2$
$n = 6$

No. of Branches $B = N - 1$

$B = 5$

Q) post order ...?

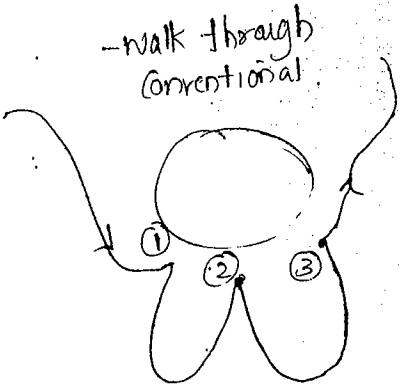


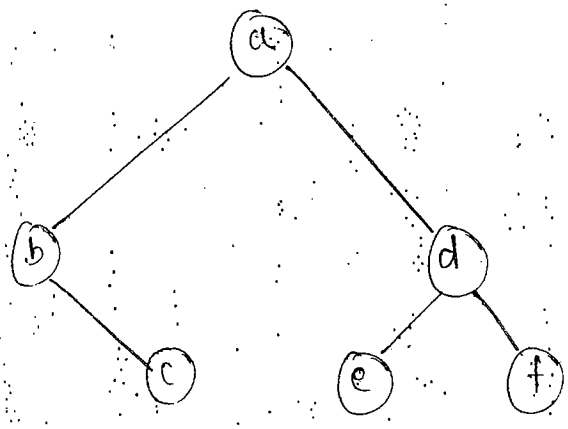
IN → D B J G E A C H K F I

CIN → I F K H C A E G J B D

pre → A B D E G J C F H K I

post →

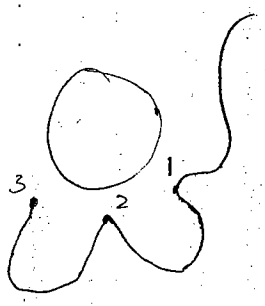




pre(1) → a b c d e f

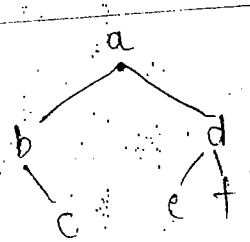
IN(2) → b c a e d f

post(3) → c b e f d a



ii) Converse

Converse

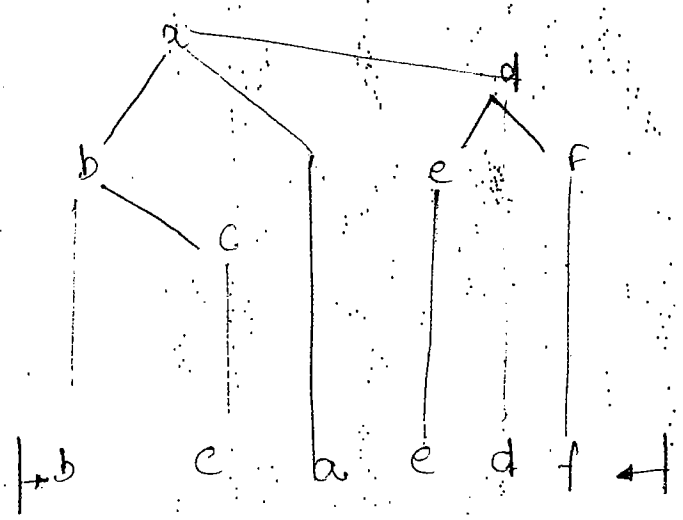


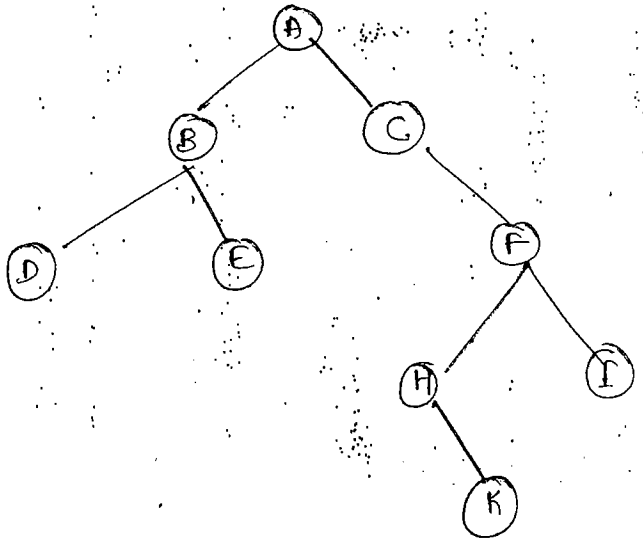
Converse pre = a d f e b c

" IN = f d e a c b

" post = f e d c b a.

Valid only for IN (falling down).



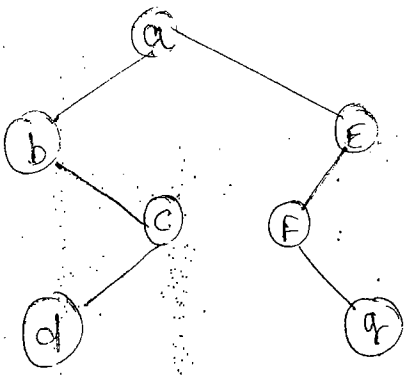


pre = A B D E G J C F H K I

post = D J G E B K H I F C A

C pre = A C F I H K B E G J D (previous question ans)

C post = I K H F C J G E D B A.



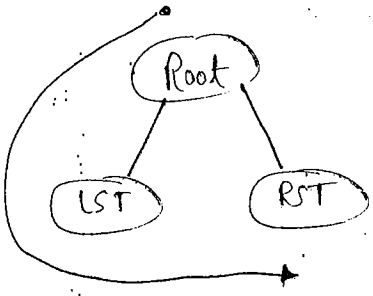
pre → a b c d e f g

post → a c b g f e a

Cpre → a e f g b c d

C post = g f e d c b a

- Recursive algorithm



preorder :-

Steps 1) visit root

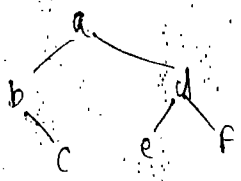
2) Traverse LST in preorder

3) " RST in preorder

- What kind of traversal does the following algorithm describes.

```
algorithm in pre traversal (root)
    if ( root is not NULL)
        1  process (root)
        2  pre traversal (LST)
        3  pre traversal (RST)
    end
```

Ans :- preorder



pre	Cpre	IN	CIN
1) Root	1) Root	1) LST	1) RST
2) LST	2) RST	2) Root	2) Root
3) RST	3) LST	3) RST	3) LST

PF
LST
PF
RST
PF

post	Cpost
1) LST	1) RST
2) RST	2) LST
3) Root	3) Root

LST → left Subtree

RST → Right Subtree

21-5-2012

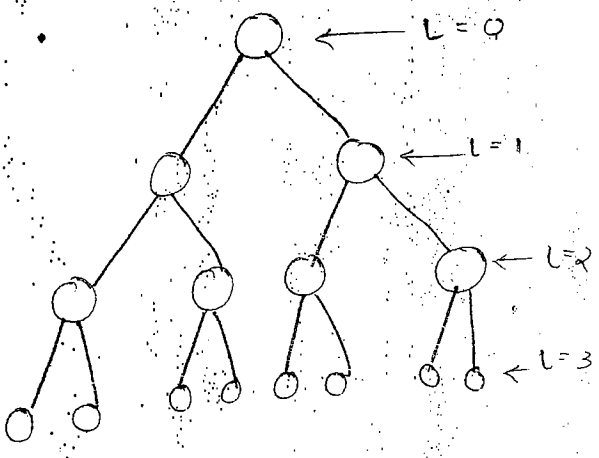
Q) Cal the max. no. of nodes in a B.T. of given height H.

i) With Recursion

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

ii) without Recursion

b) $N_{max}(H) = \boxed{2^{H+1} - 1}$



$N(3) = 7 + 2^3 \Rightarrow 15$

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

Observation:-

i) root is at level '0'

levels = Height = Depth = 2
(0-2)

ii) root is at level '1'

levels = Height = Depth = 3
(1, 2, 3)

Ans :- 2^H

i)

$N(0) = 1$

$N(1) = N(0) + 2^1$

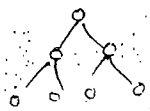
$\Rightarrow 3$

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

$\Rightarrow 7$

L=0

L=0
L=1



L=0
L=1
L=2

ii)

L = H = 0	Total
0	1
1	3
2	$7 = 2^3 - 1$
3	$15 = 2^4 - 1$

→ $\boxed{2^{(H+1)} - 1}$

Q) for a given 3- unlabelled nodes, how many distinct (Separate) BTs can be formulated?

Sol

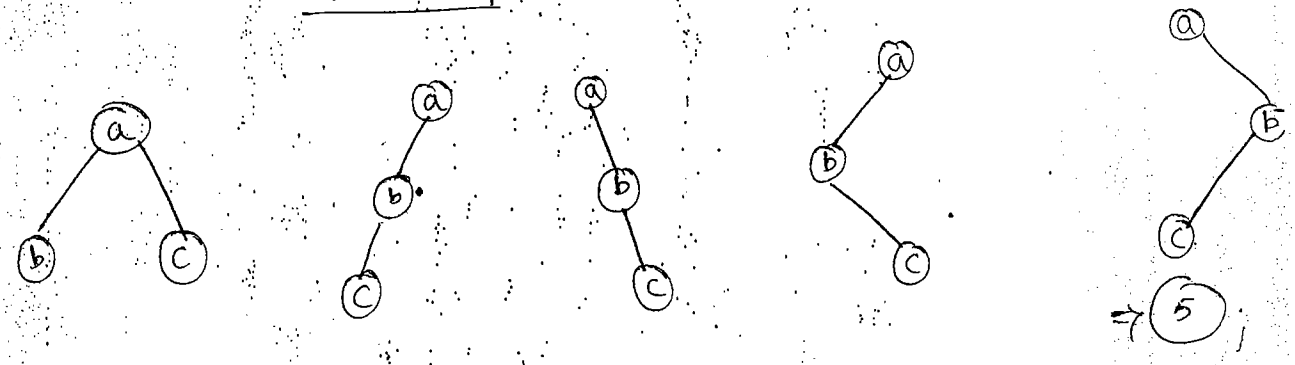
n = no. of nodes = 5

Try $n = 5$

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

$$\Rightarrow \frac{^{10}C_5}{6} \Rightarrow \frac{10 \times 9 \times 8 \times 7 \times 6}{5 \times 4 \times 3 \times 2 \times 1}$$

$$\Rightarrow 42$$



pre = abc

Q) for a given tree pre = ab ; post = ba ⇒ generate Binary Trees...

Sol

pre = ab
post = ba

Observation: To have unique pattern

- IN + pre
- (or)
- IN + post

pre = abcdef

IN = bcaedf

Q) Given :- pre = abcdef ; IN = bcaedf & Generate BT and give post

Sol

Concept :-

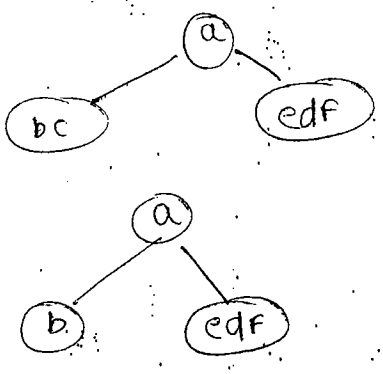
pre :- abc def

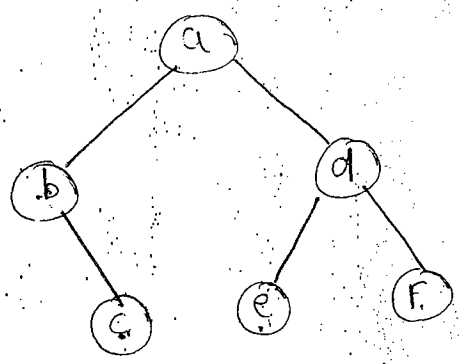
IN. ~~post~~ :- bca edf

Step(1) :- Consider IN order.

Step(2) :- Look at pre to find Next root

IN :- b c a e d f





Check :

pre :- abcdef

IN :- bcaedf

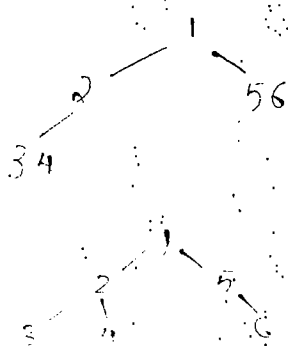
∴ post = cbefda

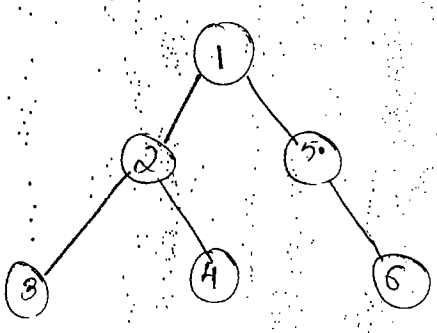
---

ii) pre = 1 2 3 4 5 6

IN ~~post~~ = 3 2 4 1 5 6

IN = 3 2 4 1 5 6



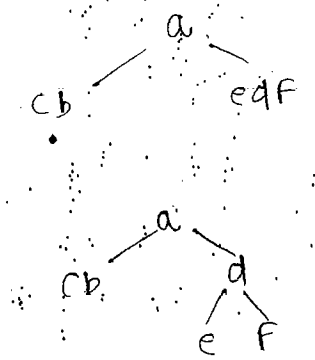


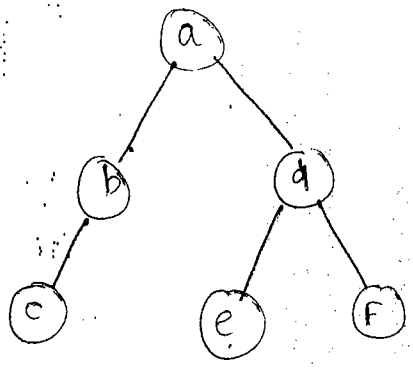
post : 3 4 2 6 5 1

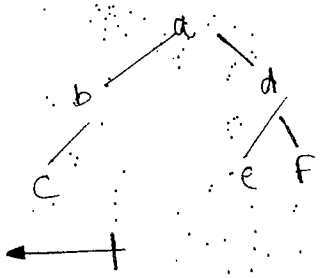
iii) post = cbefda

IN = cbaedf

IN → c b a e d f

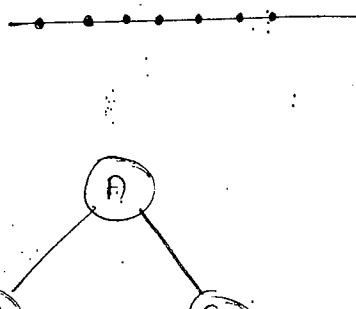






iv) post → C D B F G E A

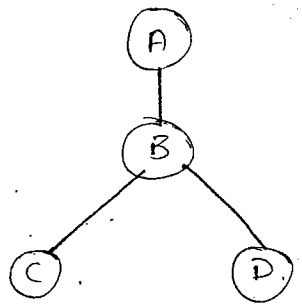
Degree → 0 0 2 0 0 2 2



pre = A B C D E F G

v) post = C D B A

degree = 0 0 2 1

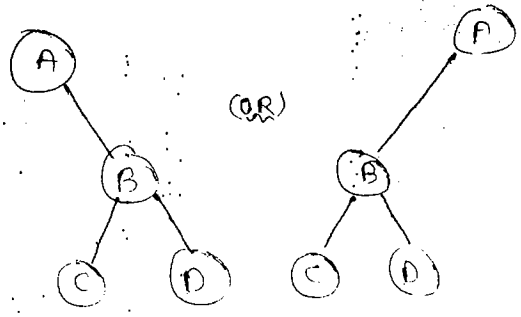


pre → A B C D

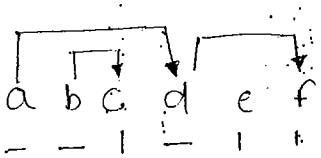
Imp

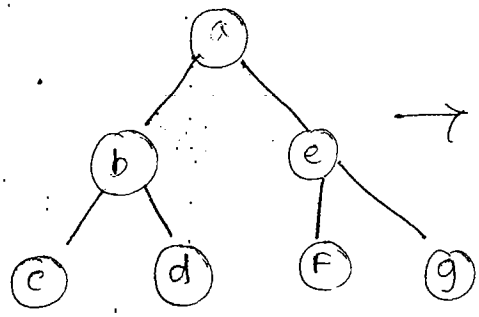
- RPTR
- pre → a b c d e f
- IAG — — 1 — 1 1

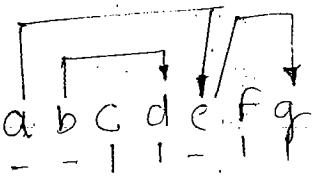
Terminal nodes generates BT

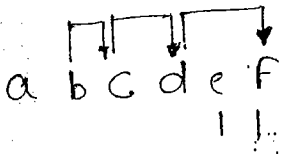


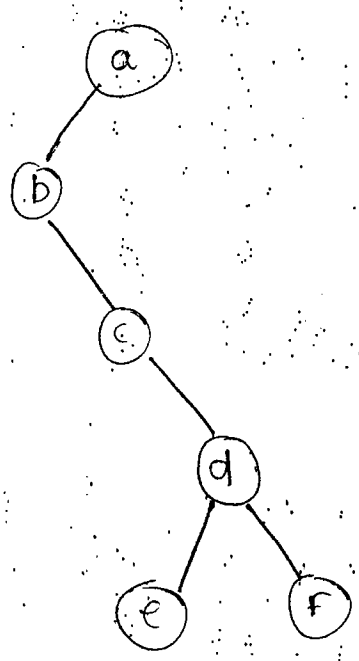
i) RPTR





→ 

ii) 



→ 5 levels

a) Single order

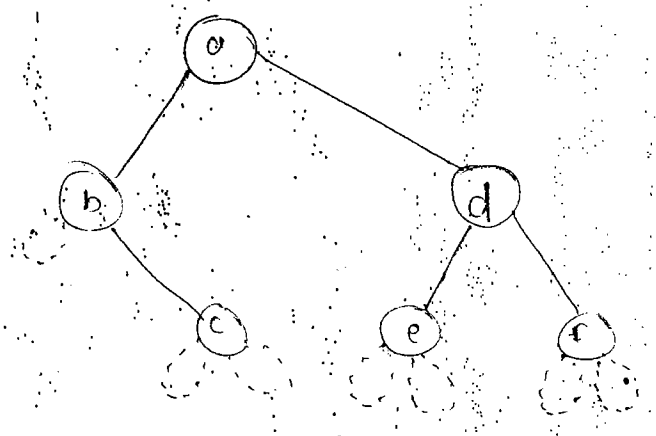
1 → LST
2 → PF
3 → RST

bcaedf

Q) Given

b) double order

1) PF
2 → L
3 → PF
4 → R



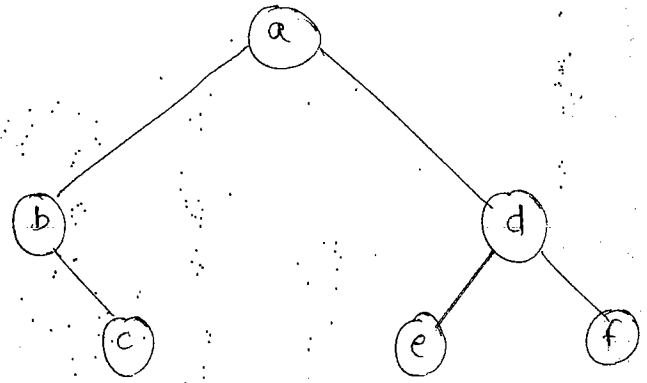
Sol.

Doc) → abbccadeedff
(Double order)

Q) Given :- of Triple order

ToC) 1 → PF
2 → L
3 → Rf
4 → R
5 → P.F

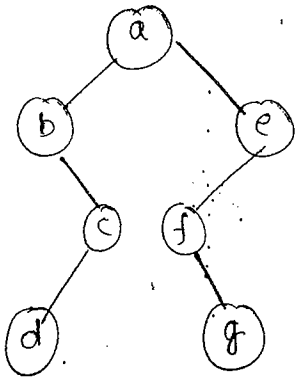
TC → Triple order



→ a b b c c c b a d e e e d f f f d a

Q) Given :-

i) Do( )
1 → Pf
2 → L
3 → PF
4 → R.

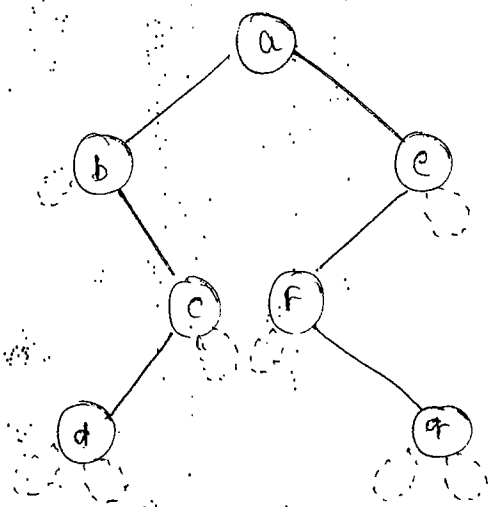


Ans :- a b b c d d c a e f f g g e.

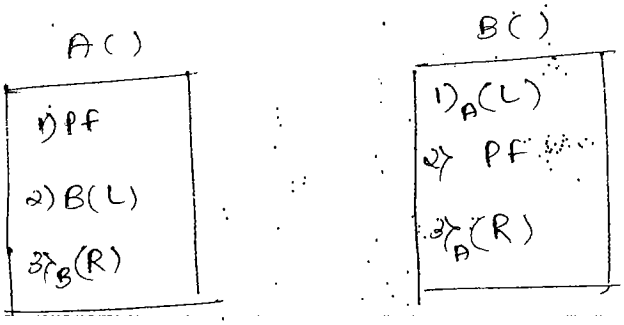
ii) Triple Order

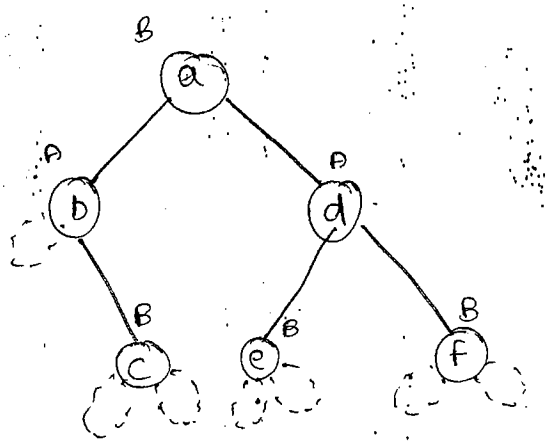
To( )

i) PF
ii) LST
iii) PF
iv) RST
v) PF

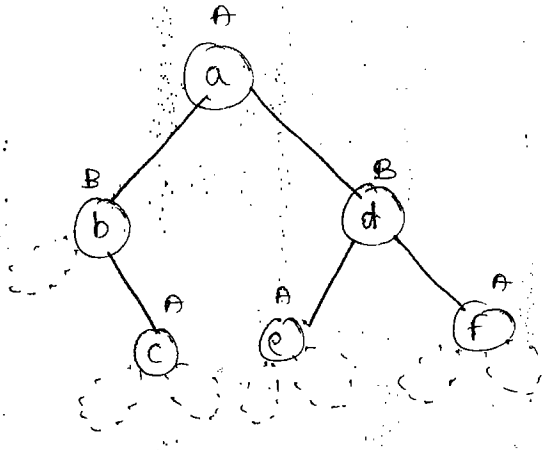


Q. Indirect Recursion.

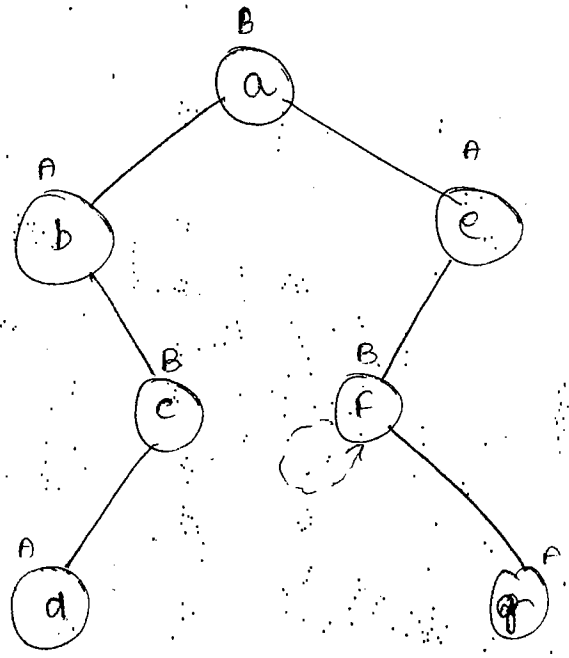




B(t) = bcadef



A(t) = abcedf



B(t) = bdcaefg

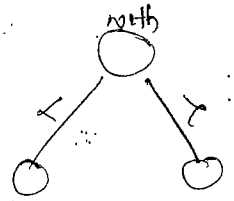
3.5.2012

- BINARY SEARCH TREE (or) lexically ordered (or) Dictionary Tree

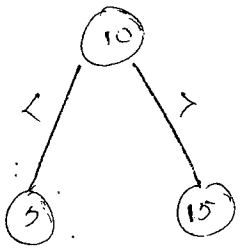
Lexican = word.

Defination :-

Binary Tree



Ex :-



IN :- 5, 10, 15

Significance :- Traverse inorder, get sorted.

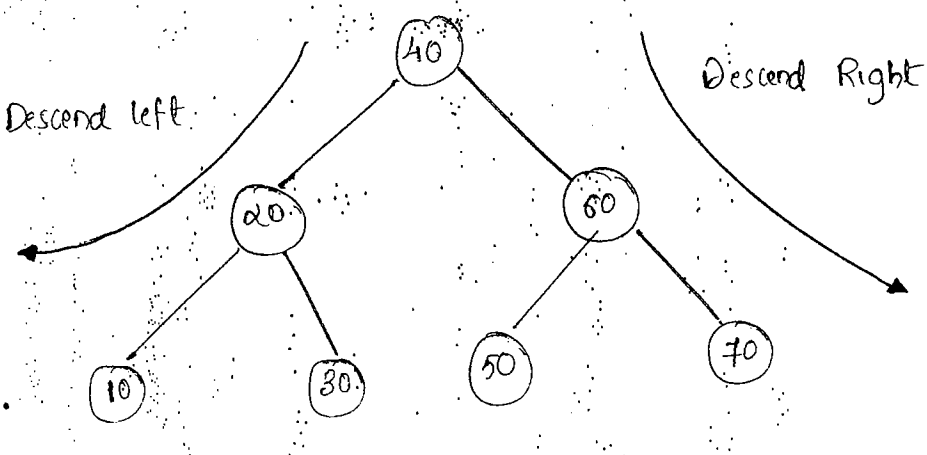
Q:- Create BST for the nodes 40, 20, 30, 60, 70, 50, 10.

procedure :- Create BST (or) insert a node in BST

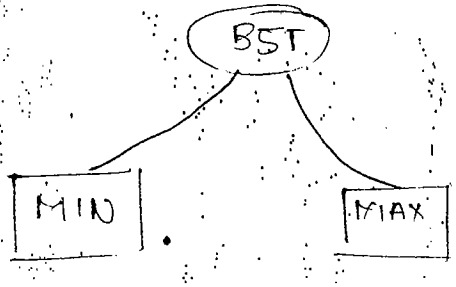
Step 1 :- very first node = root node.

Step 2 :- Start Comparing from the ~~side~~ root node, for inserting the Subsequent Nodes.

Sol

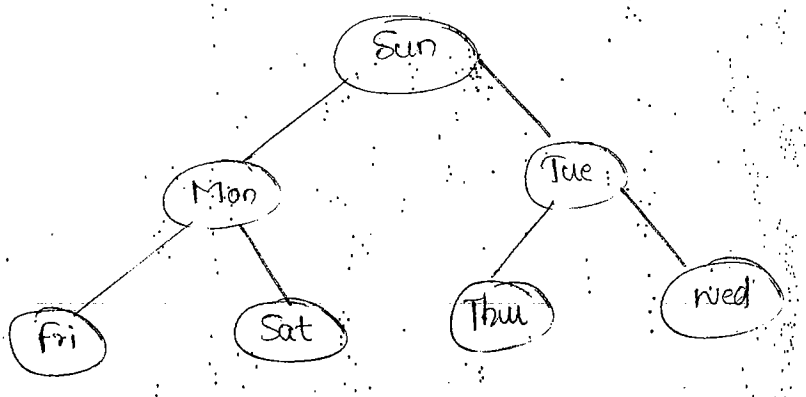


IN-Sorted :- 10 20 30 40 50 60 70



Q. Create LOBT or BST for Sun, Mon, Tue, Wed, Thu, Fri, Sat.

Sol



IN :- Fri MON Sat Sun Thur Tue wed.

ies DRDO JTO.

Q. Identify Valid Search Sequence on a BST for node 88.

a) 111, 11, 99, 22, 55, 95, 85, 88

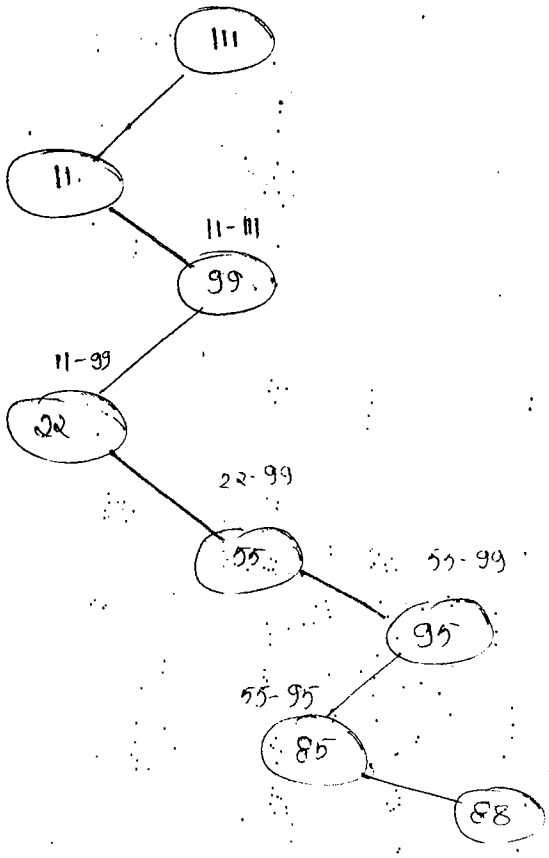
b) 333, 33, 53, 93, 65, (95), 85

c) 1, 1111, 111, 91, 71, (101) 88

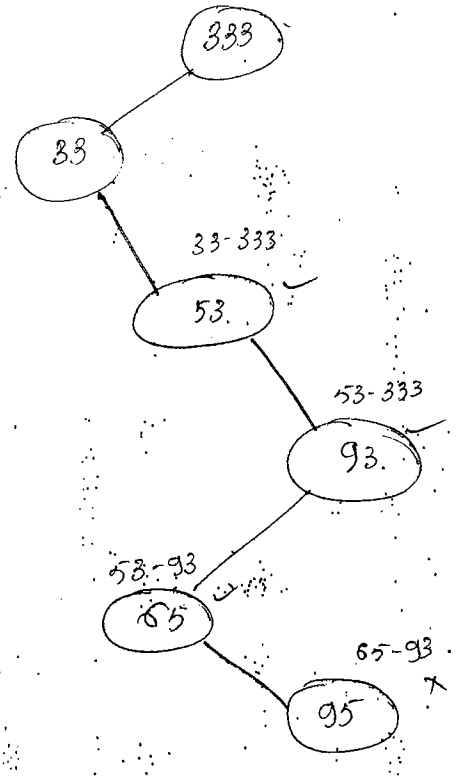
d) 40, 70, 100, (60) 80, 90 85, 88.

Sol a) 111 11 99 22 55 95 85 88.

for 88 :-



Ex: 333, 33, 53, 43, 63, 93, 88



Deletion procedure :-

Goal :- Even after deletion, the order should be intact. (remain unchanged)

Observation :- Tree can be disturbed but order remain unchanged.

A Node to be deleted:	Action
Case(i) :- No children	: Simple delete it.
ii) Only one child (or) subtree	: Connect to its grand parent.
iii) Both children or subtrees	: Replace with.

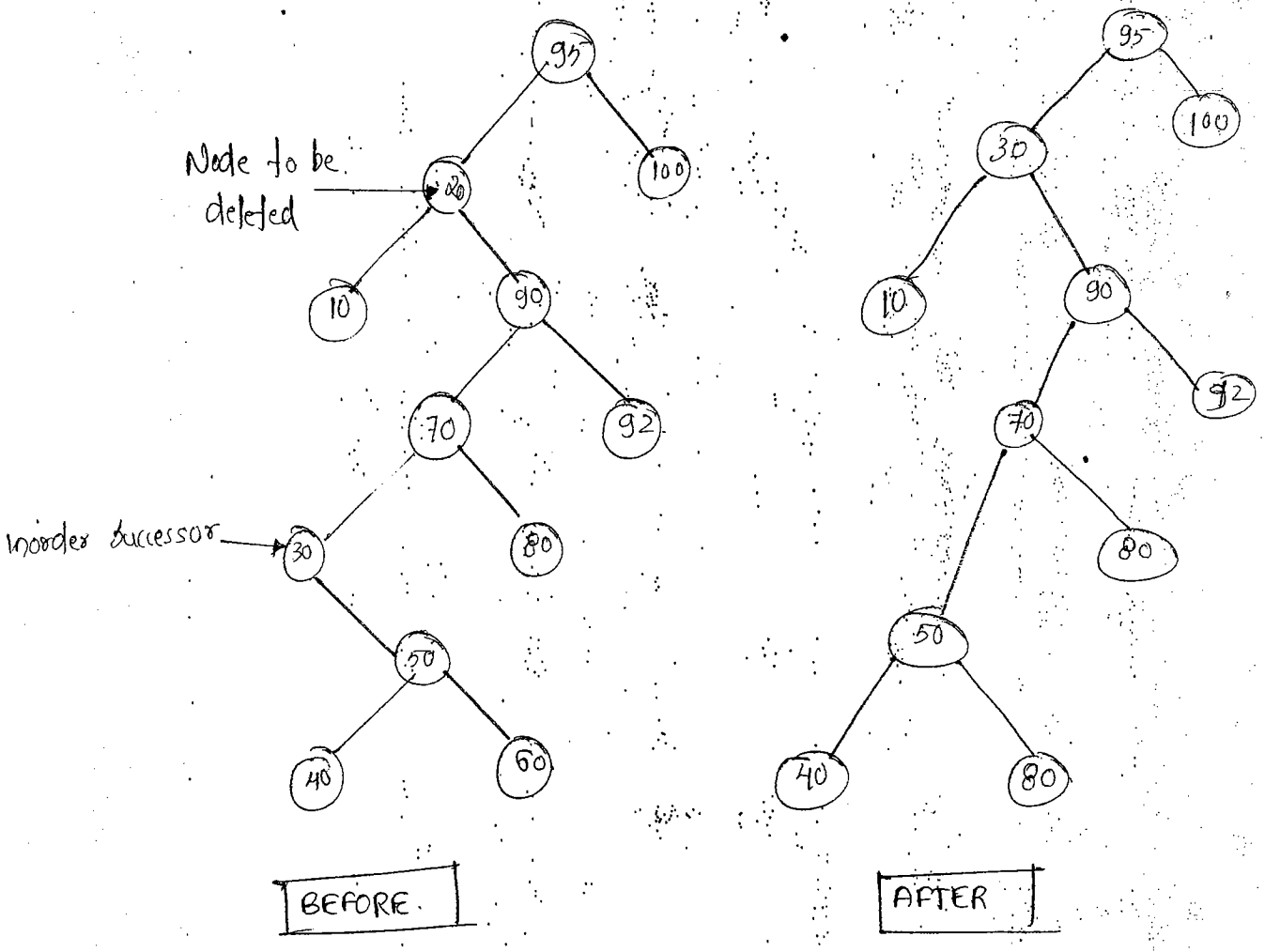
default action ——→ a) inorder successor (or) min. of RST

if insists, (asked) ——→ b) inorder predecessor (or) target of LS

Logic for the inorder Successor

1) Jump Right.

2) Go on descend left

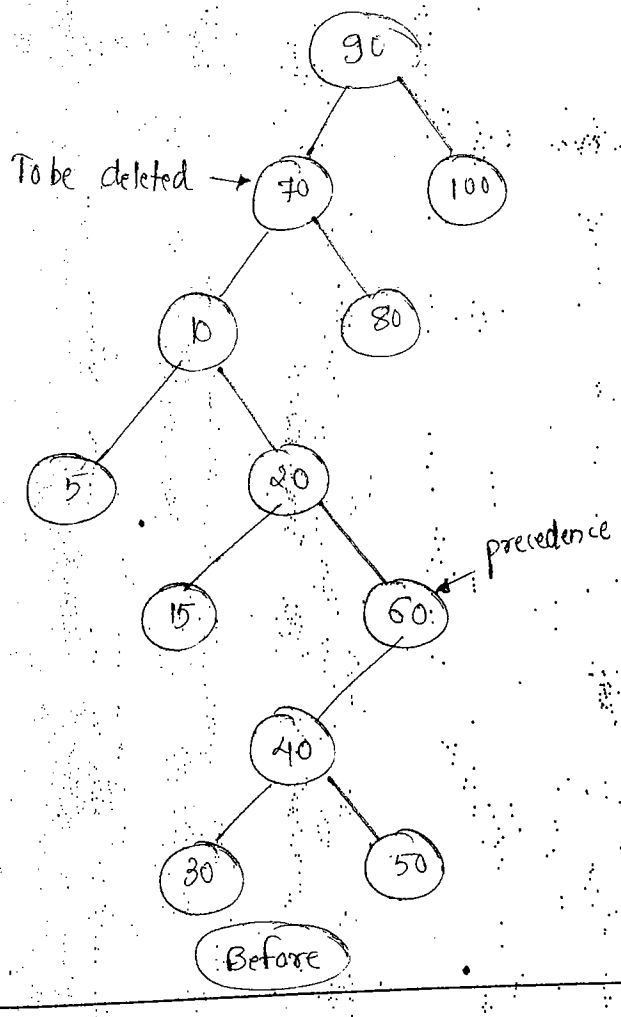


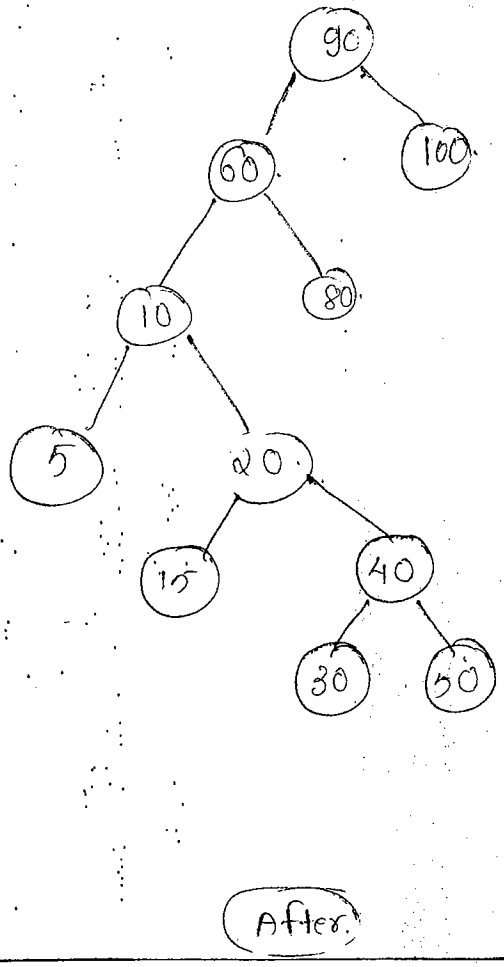
Q) INORD pred (or) largest of 7

1) Jump left

2) Go on descend Right

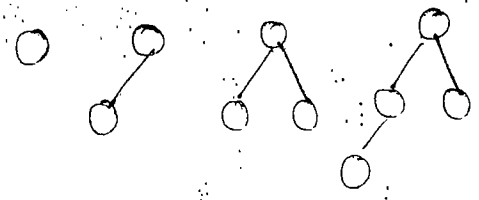
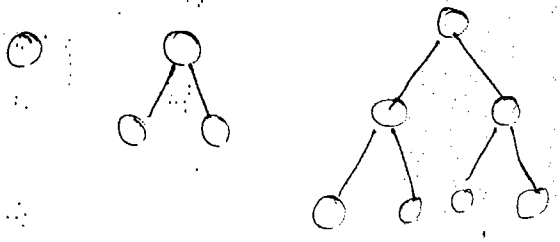
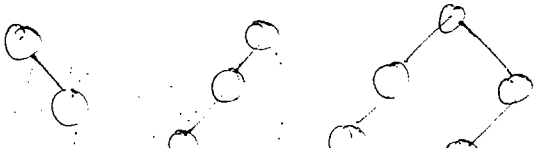
Sol if insisted to work over precedence.





✗ S_1 ⟶ all CBT's are FBT's ⟶ false

S_2 ⟶ All FBT's are CBT's ⟶ True

CBT Complete BT	FBT full BT
→ Seq (array) [Sequential] → L to R in Sequence.	Defination :- @ any level, the max. no of nodes.
Valid :- 	
invalid 	

• HEAPS

Defination :- Sequential Representation (array) and Complete Binary tree.

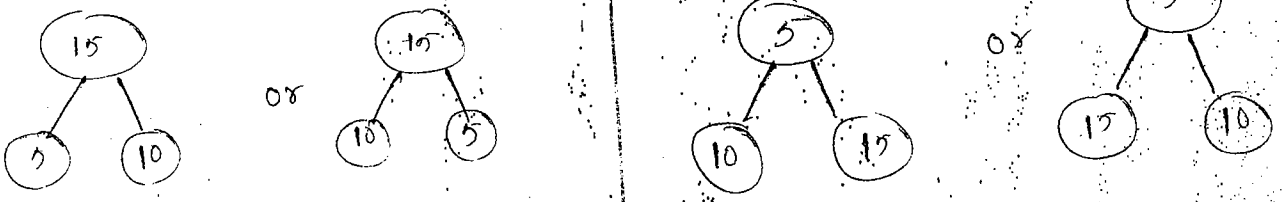
i) ith Node children → 2.i , 2.i+1

ii) Jth Node parent : [J/2]

Heaps

MAX heap | MIN heap

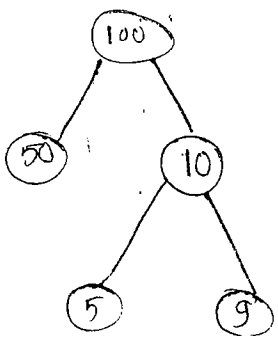
→ parent/root level : MAX | min



Q. identify MAX heap

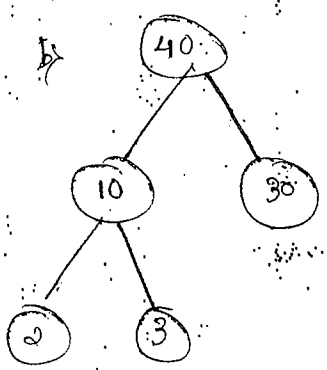
Sol

a)



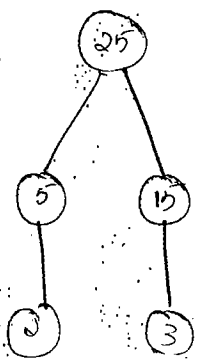
Invalid

b)



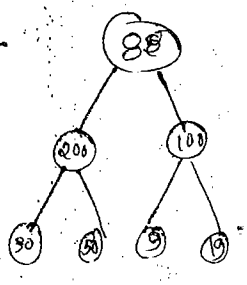
Invalid

c)



Invalid

d)



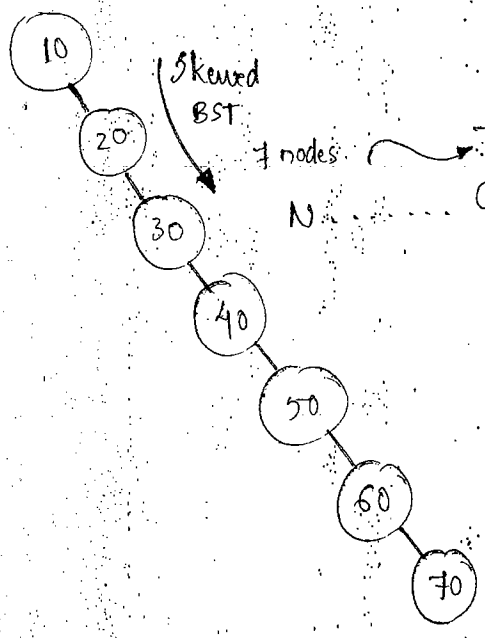
Invalid

Ex: Create BST

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

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

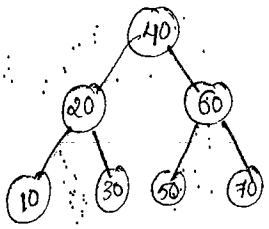
i) Search for 70



Skewed BST

7 nodes ⇝ 7 attempts

N ..... O(N)



Full BST (or) Bushy BST

⇒ O(log₂N)

7 nodes ⇝ 3 attempts

⇒ $\log_2 7$

≈ $\log_2 8 = \log_2 2^3$

⇒ 3.

Goal :- Apply rotations, Skewed BST ⇝ Bushy BST : Search time $O(\log_2 N)$

- Height Balanced BST :- AVL Tree

→ Devised by Russian mathematicians Adelson Velski, Landi

Defination :- - BST

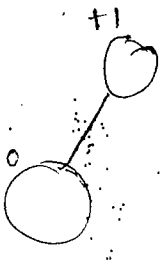
- with Balance factor : $h_L - h_R = \begin{cases} +1 \\ 0 \\ -1 \end{cases}$

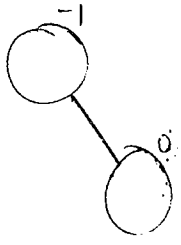
where h_L = height of LST (or) Level of the left Side Tree.

h_R = " " R " " " Right Side Tree.

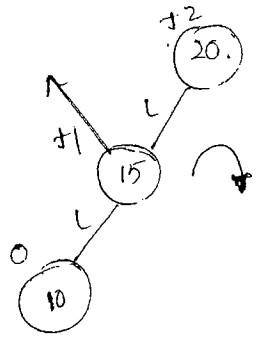
- ±2 node is violated Node.

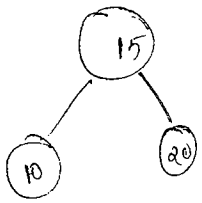
Valid →



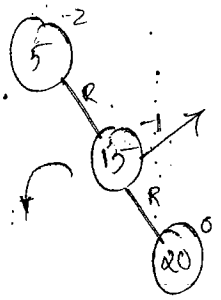


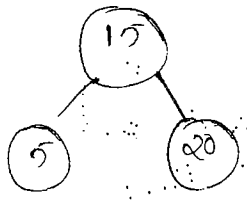
I) LL imbalance



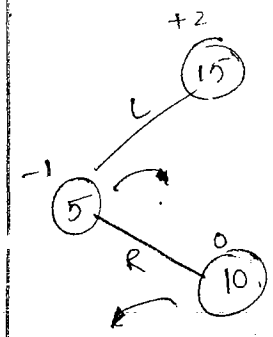


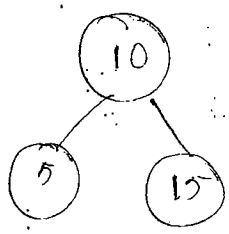
II) RR imbalance



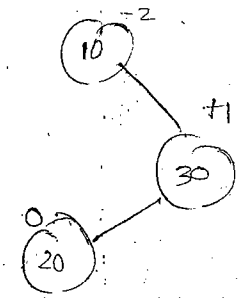


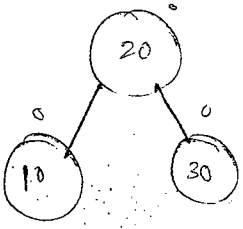
III) LR imbalance





IV) RL imbalance



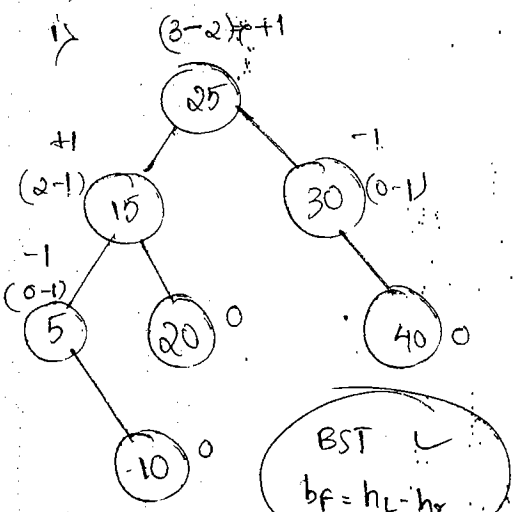


Jump

±2 ⇒ Violation, stop

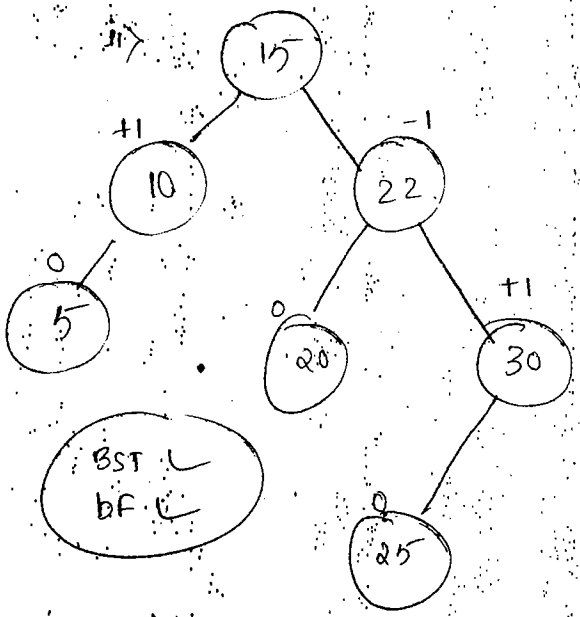
Q. Identify valid AVL Trees.

i)



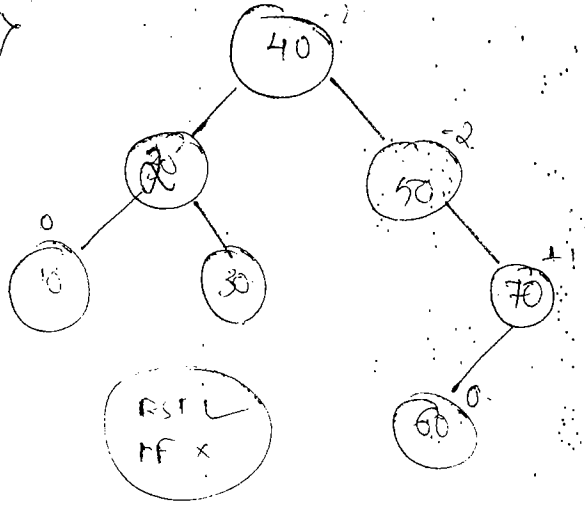
BST ✓
$bf = h_L - h_R$

ii)



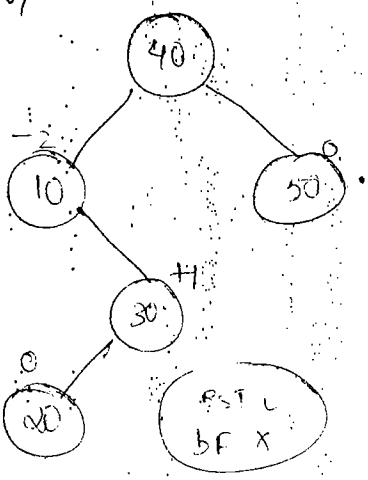
BST ✓
bf ✓

iii)



BST ✓
bf ✗

iv)

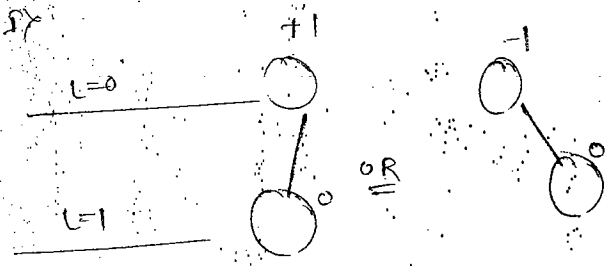


BST ✓
bf ✗

Q1) i)

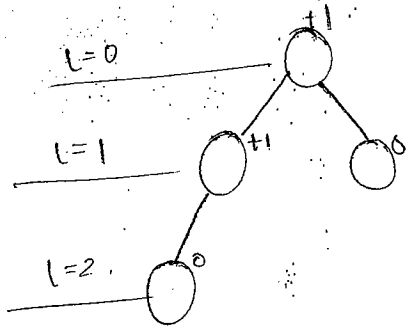
Q) Calculate minimum no. of nodes for a given height (5). root is at level '0'

Sol i)



$N_{min}(1) = 2.$

ii)



$N_{min}(2) = 4.$

$N_{min}(2) = 1 + N(1) + N(0)$

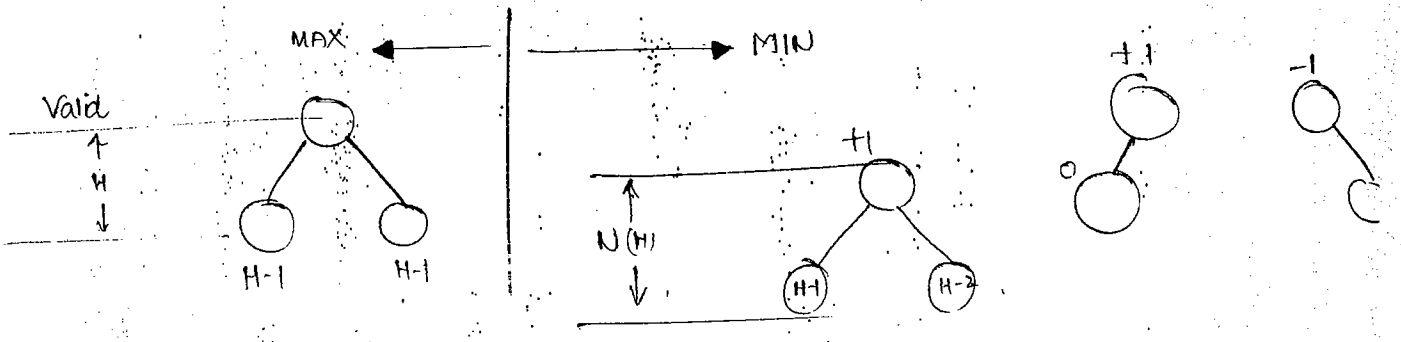
$\Rightarrow 1 + 2 + 1$

$\Rightarrow 4.$

iii)

$N_{min}(3) = 7$

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



$1 + N(H-1) + N(H-2)$

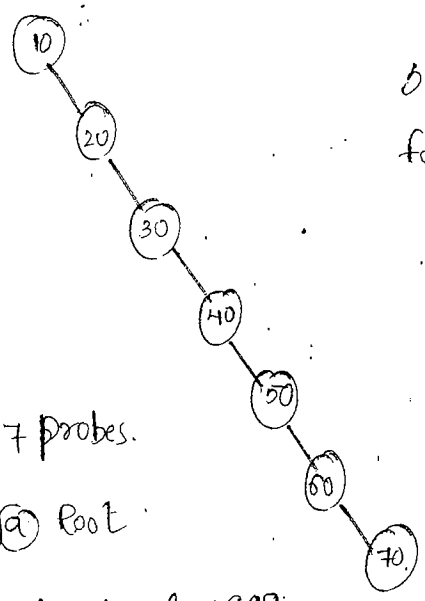
Fibonacci $f(n) = f(n-1) + f(n-2)$

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

H	0	1	2	3	4	5
N(H)	1					

• ⟨ SPLAY TREE ⟩

→ In the world of information technology it has been observed that the same record is going to be accessed for several times then splaying that record towards root requires the following rotations ——— 1) Zig 2) Zig-Zig 3) Zig-Zag



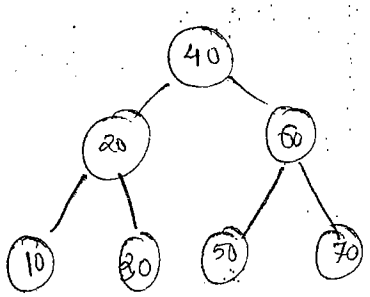
Search for 70.
for 1000 times

1 time = 7 probes.

Splay @ Root

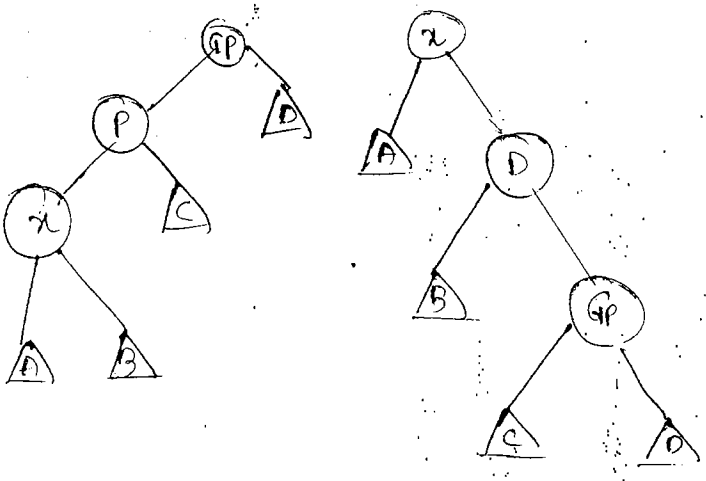
999 times × 1 attempt × 999

1000 times = 1006 attempts / probe

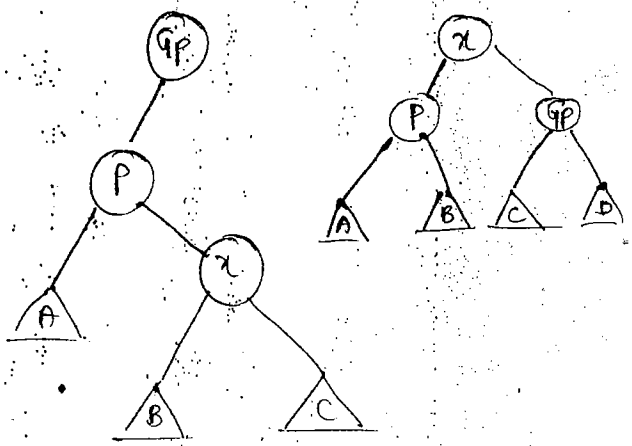


1000 times × 3 attempt = 3000 probes

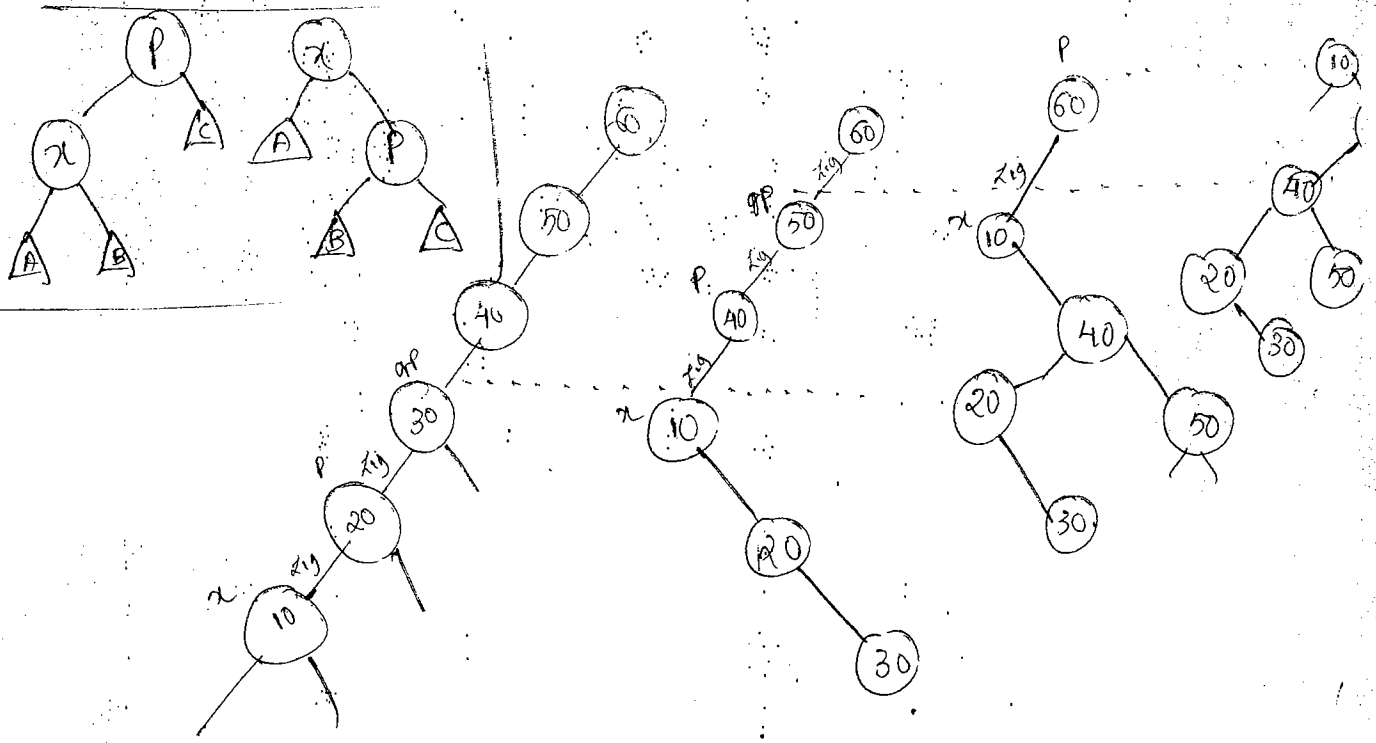
II - Zig-Zig



III - Zig-Zag



1) Using this method.



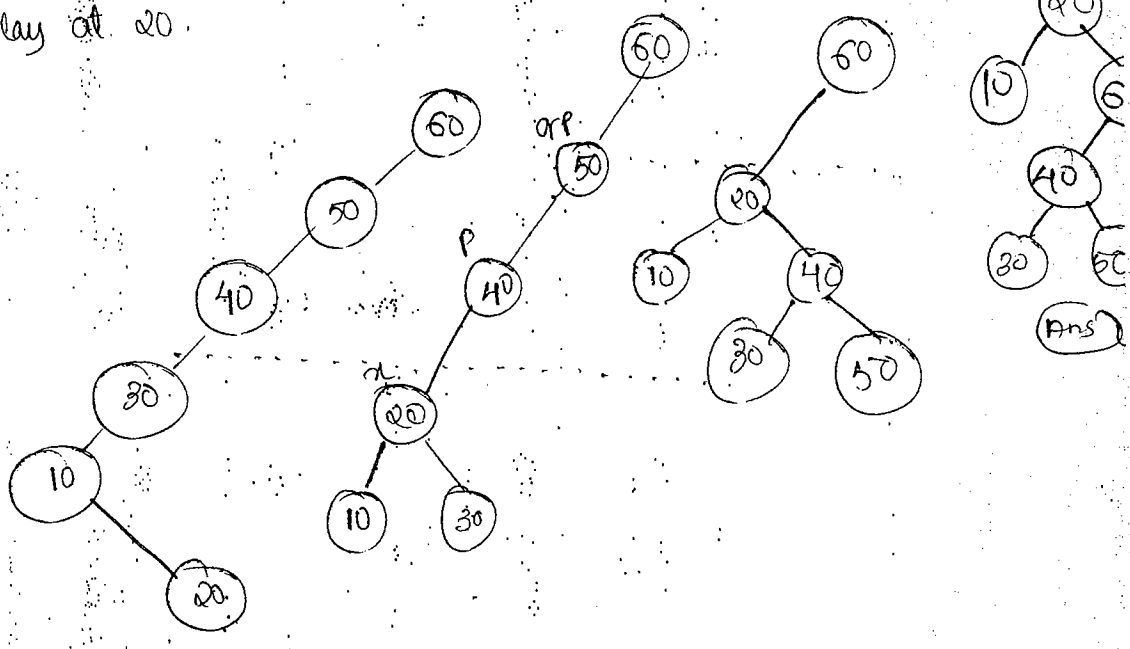
Splay Tree =

BST =

IN = 10, 20, 30, 40, 50, 60.

Q) Find Splay at 20.

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



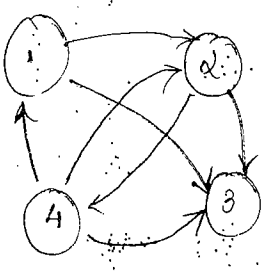
GRAPHS

Representation

i) sets

Vertices	adjacent
1	{2, 3}
2	{3, 4}
3	ϕ
4	{1, 2, 3}

Given: Directed graph



ii) Adjacent matrix

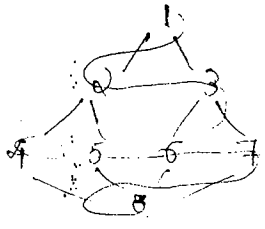
	1	2	3	4
1	F	T	T	F
2	F	F	T	T
3	F	F	F	F
4	T	F	T	F

Graph Traversals

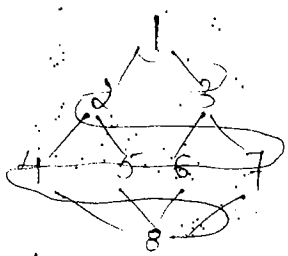
DFS	BFS
i) Depth first Search	ii) Breadth first Search
ii) Similar to preorder	ii) level order
iii) Recursive stack	iii) Iterative Queue.
Termination :- Stack empty	Q is empty
in algorithm :- (Each)	in algorithm : [all]

BFS

L.R R.L



O/P => 1, 2, 3, 4, 5, 6, 7, 8



O/P => 1, 3, 2, 7, 6, 5, 4, 8

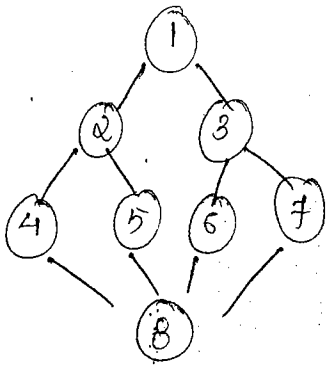
Q) Identify valid BFS Search Sequences. if starting vertex is v;

a) 1 3 2 7 6 4 5 8 → invalid

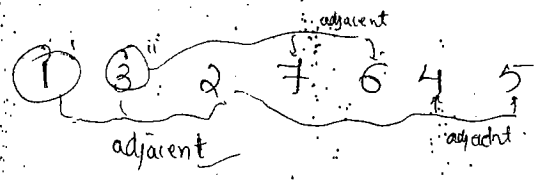
b) 1 2 3 5 4 7 6 8 → valid

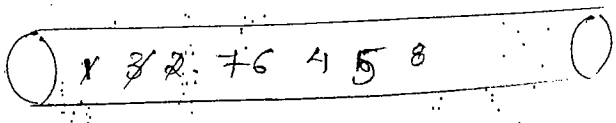
c) 1 2 3 7 6 5 4 8 → invalid

d) 1 3 2 4 5 7 6 8 → invalid



Sol:

a) 



- adjacent of 1 is 2,3 ; next is 3,2
- adjacent of 3 is 6,7 ;
- adjacent of 2 is 4,6 ;

b) 5 2 8 4 1 6 7 3. → valid

adjacent of 5 is 2,8 , adjacent of 2 is 4,1 , adjacent of 8 is 6,7 ; hence valid.

c) 5 2 8 4 1 3 6 7 → wrong

d) 5 8 2 4 7 6 1 3 → valid

e) 5 8 2 4 6 1 3 7 → wrong.

4.5.2012

Graph	Tree
vertex explore	node visit

• Depth first Search

→ Recursive Routine (stack)

```
procedure DFS (v)
    visited (v) = True.
    for each adjacent 'w' for 'v'
        if not visited (w)
            Call DFS (w);
end.
```

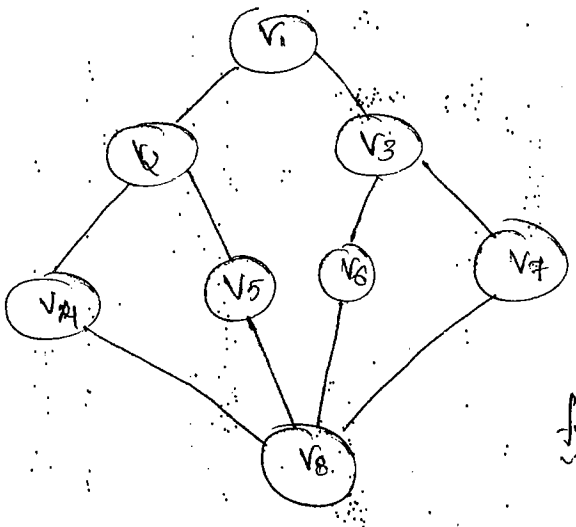


fig: a

Observation

→ Step back only when the adjacent is already visited.

→ move forward (Depth) ; atleast on unexplored adjacent is there.

St. vertex (v1)

i) Valid :- 1 2 5 8 4 6 3 7

ii) Invalid :- 1 2 5 8 4 6 7 3

Q. identify valid/invalid DFS Search Sequences. if. → Depth first Search.

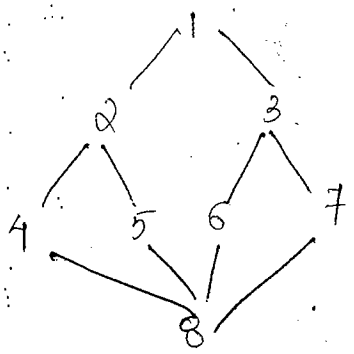
St. vertex (V1).

a) 1 3 6 7 8 5 4 2

b) 1 2 3 6 7 5 4 8.

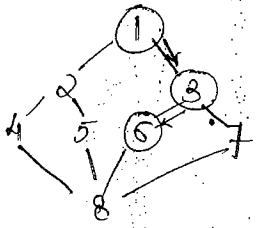
c) 1 2 5 4 8 6 3 7

d) 1 3 6 8 7 5 2 4



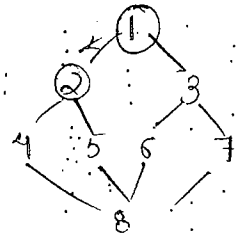
Sol

a)



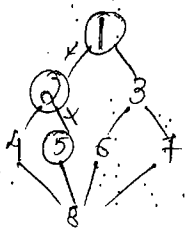
- 6 to 7 can't move hence invalid

b)



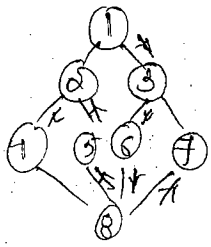
- hence invalid.

c)



- hence invalid

d)



- hence valide

- in case d, after 7 we have to goto 5, already both the nodes are visited hence step back to 8 then move to 5 then 2 & 4

~~c) 8, 1~~

Q. Starting vertex is (V5)

a) 5 2 4 8 6 3 7 1

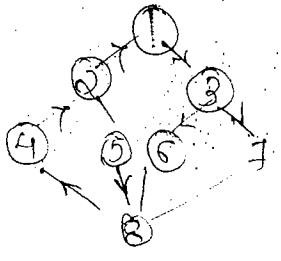
b) 5 8 6 3 7 1 2 4

c) 5 8 4 2 1 3 6 7

d) 5 2 1 3 6 7 8 4.

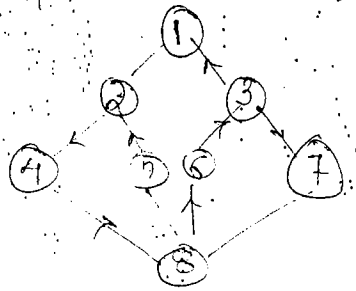
Sol

a)



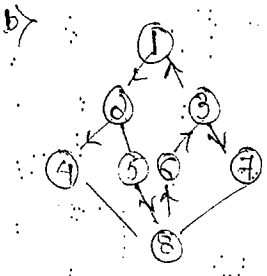
Valid

a)



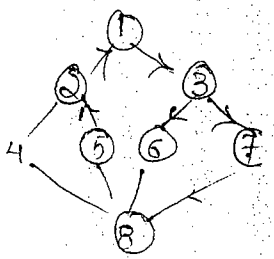
Valid

b)



Valid

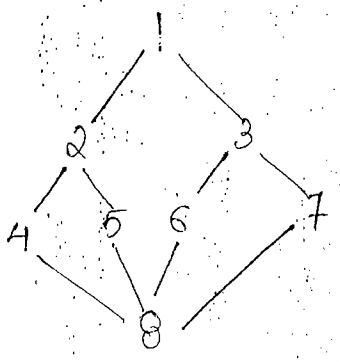
d)



invalid

Q> identify valid/invalid DFS/BFS search sequence.

a) 7 3 8 1 2 4 5 6



6 is missing hence not BFS.

7 3 8 1 2 4 5 6

Ans:- neither BFS or nor DFS.

b) 2 4 5 1 8 6 7 3

- neither DFS nor BFS

c) 4 2 5 8 6 3 7 1

- valid for DFS!

• HUFFMAN (TREE) ALGORITHM (prefix tree : Binary tree)

Q> Encode H.T for the given data.

Symbols	a	b	c	d	e	f	g
frequency function	10	15	12	3	4	13	1

Defination:- An algorithm constructs an optimal prefix code by repeatedly merging the minimum height trees.

A binary tree (ty):- A data structure in which a left branch represents '0' and a right branch represents '1'. The path to ~~be~~ a node ~~indication~~ indicate its representation.

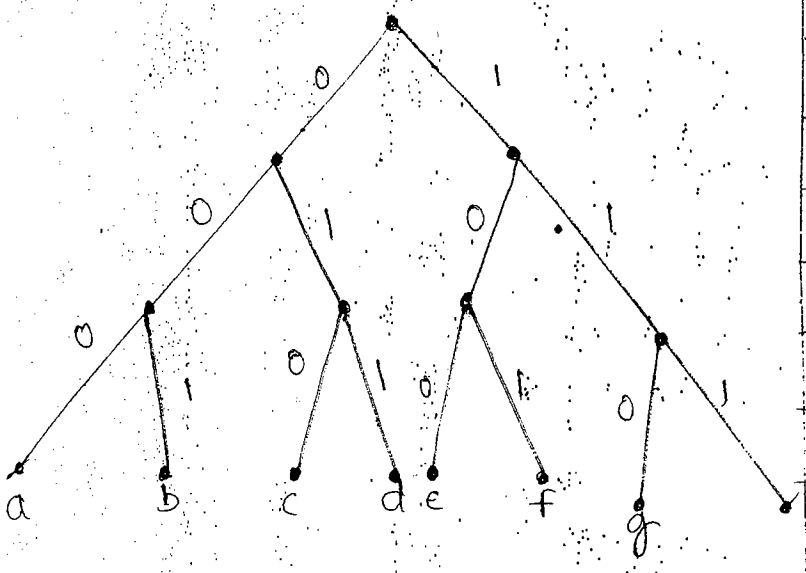
Goal :- To reduce the no. of bits required (Zip or Compression tech

logic :- More repeated or frequency character should be represented in less no. of b and vice versa.

---

i) Without huffman → Refer above table.

Normal tree

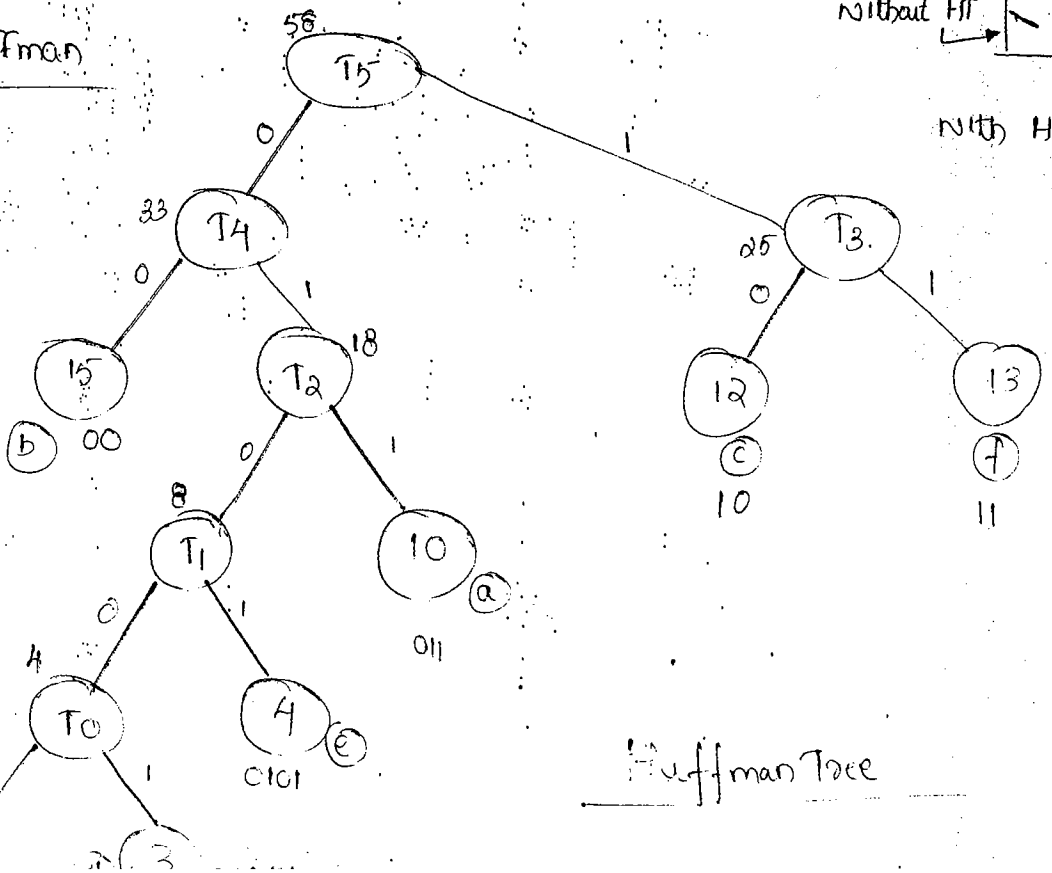


	Symbols	frequency	no. of bits	Total
011	a	10	000	30
00	b	15	001	45
10	c	12	010	36
01001	d	3	011	9
0101	e	4	100	12
11	f	13	101	39
01000	g	1	110	3
				174

Without HT → 7 x 4

With HT

ii) With huffman



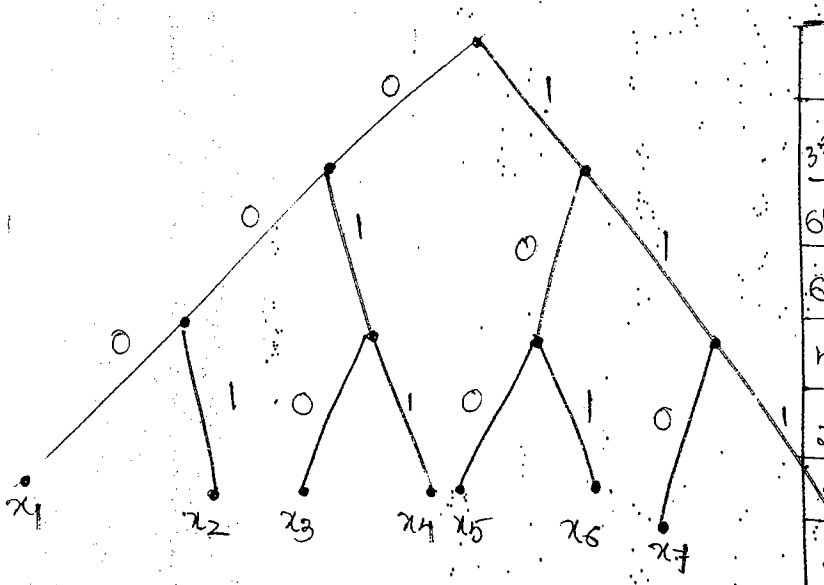
Huffman Tree

Q.) Encode H.T for the given data

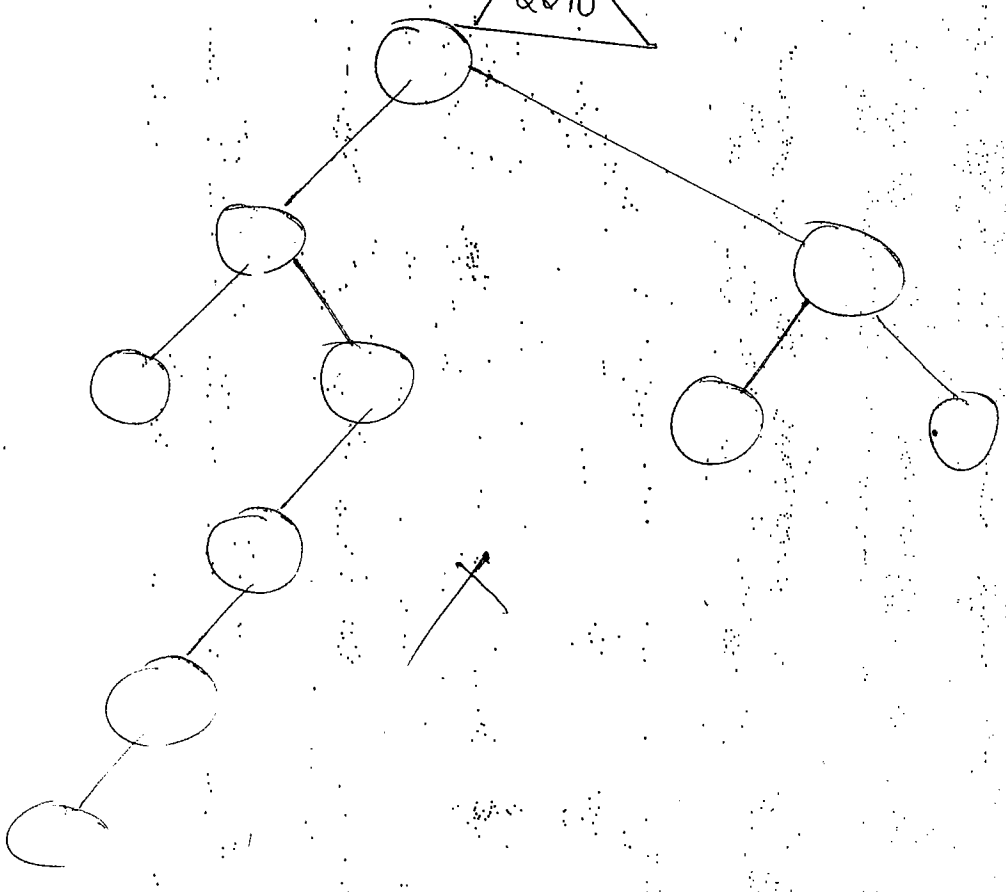
Symbols	x_1	x_2	x_3	x_4	x_5	x_6	x_7
probalities	0.35	0.3	0.2	0.1	0.04	0.005	0.005

Sol

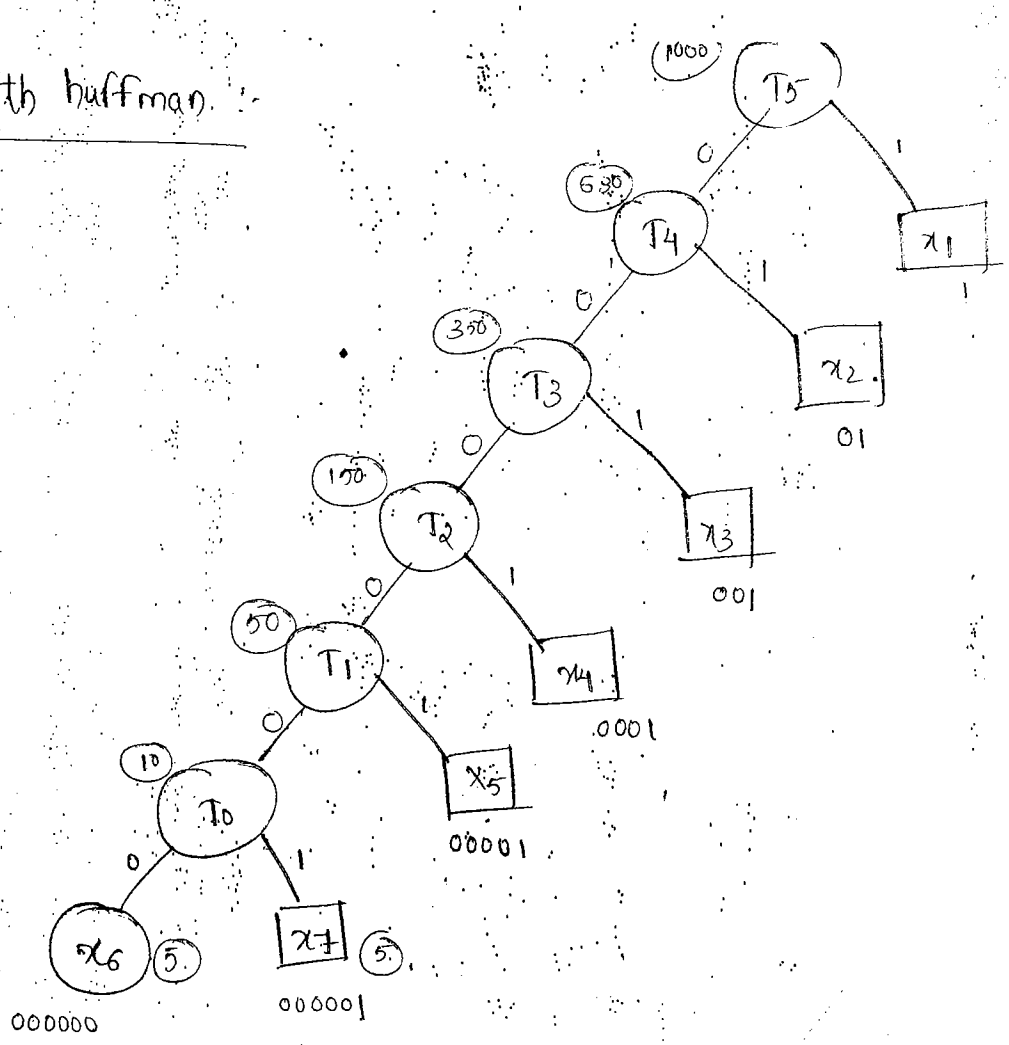
i) without huffman



		Symbols	frequency	No. of bits	Total
350	1	x_1	0.35	000	1.05
600	01	x_2	0.3	001	0.9
600	001	x_3	0.2	010	0.6
400	0001	x_4	0.1	011	0.3
200	00001	x_5	0.04	100	0.12
30	000001	x_6	0.005	101	0.015
30	000000	x_7	0.005	110	0.015
2210					3000

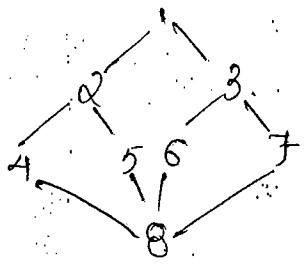


iii) with huffman :-



Huffman Tree.

Q1



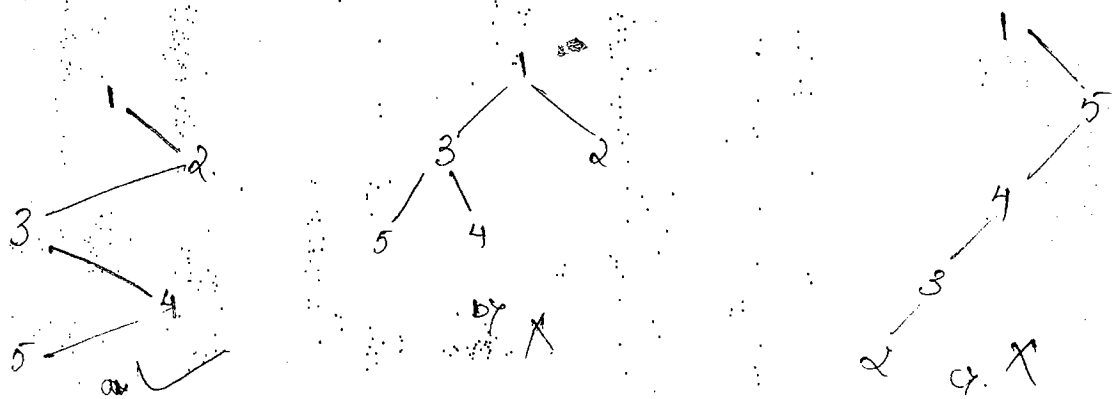
1	5	2	4	8	6	3	7 1
x 2	5	8	6	7	3	1	2 4
3	5	8	4	2	1	3	6 7
x 4	5	8	7	6	3	2	5 4

Q2

POST	1	5	4	3	2 1
IN	1	3	5	4	2

What is BT ... ?

Sol:



• HASHING

→ Searching Technique

→ Goal : Search time = order(1) : but not fulfilled.

HF :- Hash function = key to address transformation

$$H(x) = x \bmod m$$

where x = Key

m = hash table size.

Key	address
69	4
47	2
50	0
18	3
71	1

0	50
1	71
2	47
3	18
4	69

Searching

1) Linear Search
2) Sequential Search.

here $m = 5$ (0-4)

Search for 69 : 69 % 5

$$5 \overline{)69} \ (13$$

$$\underline{65}$$

$$4$$

4 is the address location where 69 is stored.

Q) Find the suitable HF for storing the elements in the address range from 1 to 1000

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

b) $H(x) = (x+1) \bmod 1000$: 1 - 1000

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

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

(0 - 999)

Sol

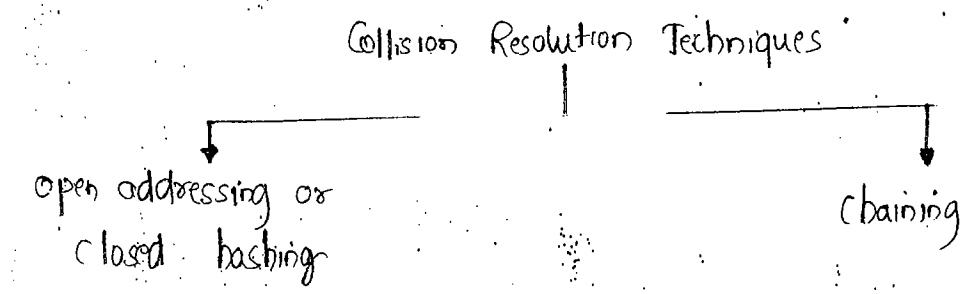
Q. find the no. of collisions for storing the following keys with a Hash function $H(K) = K \bmod 13$.

K = key	130	36	96	12	9	32	45	71	200	300
Location.	0	10	5	12	9	6	6	6	5	1
							①	②	③	

Sol

∴ No. of Collisions = 3.

• COLLISION RESOLUTION TECHNIQUES

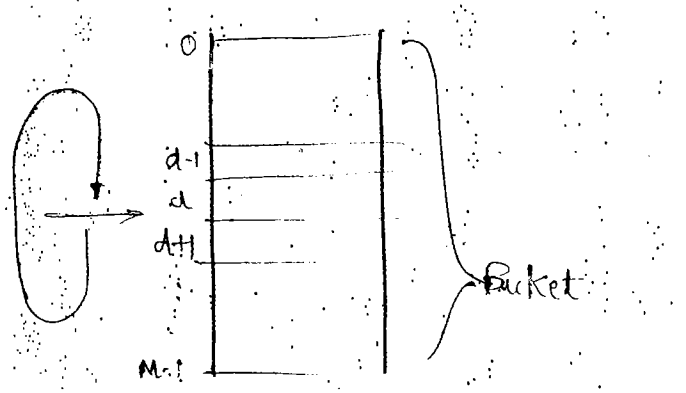


Ex:- Linear probing
L P

Defination :-

LP :- Once the bucket is full, then linearly come down to find the next empty bucket.

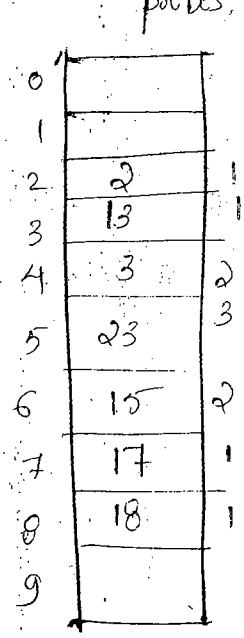
Sequence :- d, d+1, d+2, . . . m-1, 0, 1, ... d-1



Q> The Key 17, 18, 13, 2, 3, 23, 15 are inserted into an empty hash table of length 10 with H(K) = K % 10 using linear probing.
What is the Resultant Table.

Sol.

Key	17	18	13	2	3	23	15
H(K) = K % 10	7	8	3	2	3	3	5



advantage

- Once the empty bucket is encountered then the element is not found in the entire table without comparing all the buckets.

Q> Which of the following choices given a possible order in which the key valid could have been inserted into the hash table given below.

a> 46 42 34 52 23 33

b> 34 42 23 52 33 46

c> 46 34 42 23 52 33

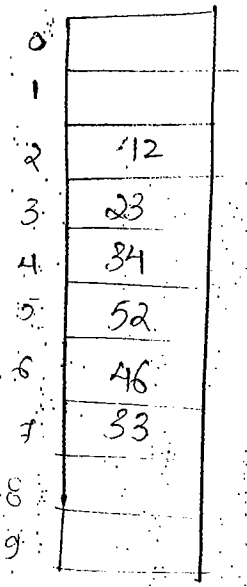
d> 42 46 33 23 34 52

Sol.

a>

Key	46	42	34	52	23	33
Key.ID Location	6	2	4	2	3	3

- invalid.



b>

Key	34	42	23	52	33	46
Location	4	2	3	2	3	6

- invalid

c>

Key	46	34	42	23	52	33
Location	6	4	2	3	2	3

- invalid → Come down and store in next address.

d>

Key	42	46	33	23	34	52
Location	2	6	3	3	4	2

- invalid.

Observation :-

To have the above same effect, how many such different insertion sequences could have been there = 30

Case (i) To insert ~~variable~~ 52

variable	fixed
42, 23, 34	52, 46, 33

$31 = 6$

Case (ii) To insert 33.

it should definitely come after 5.

variable	fixed
46, 42, 23, 34	52, 33.

$41 = 24$

Case (i) and Case (ii) are mutually exclusive.

$$So = 6 + 24 = 30.$$

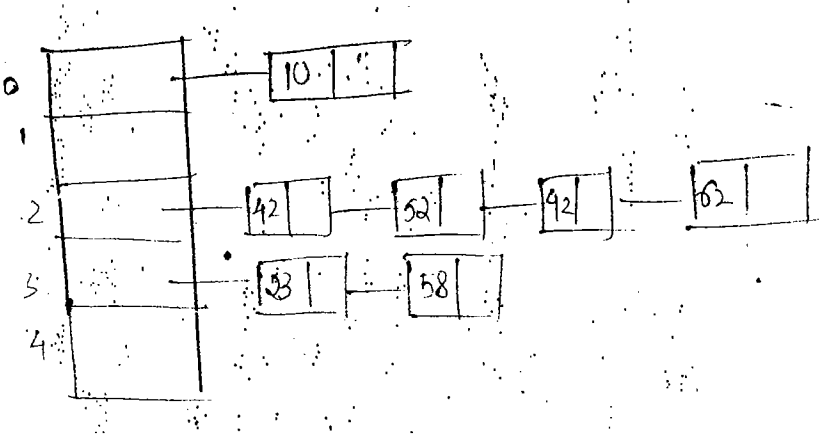
Disadvantage of open addressing or closed hashing

1) Overflow.
2) Deletion not possible
3) Collided records requires more probes.

Advantage of Chaining

1) No overflow
2) deletion is posible.
3) Collided records requires less probes

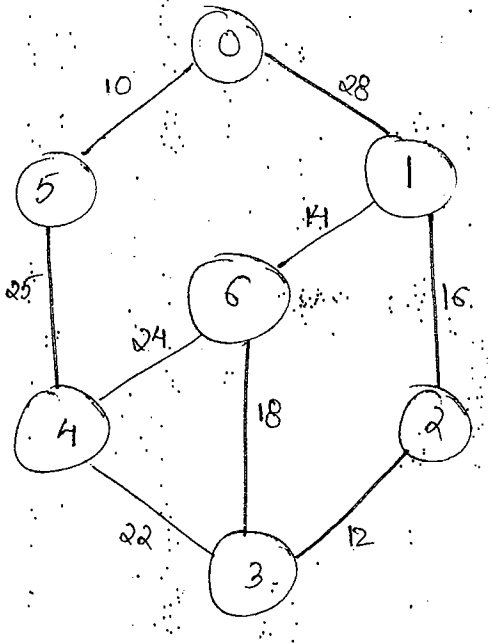
K	23	42	52	92	58	62
K·1.5	3	2	2	2	3	2



5.4.2012

• < MINIMUM COST SPANNING TREES >

1) we must use only edges with in the edges.
2) We must use exactly (n-1) edges.
3) " " " not use edges that would cause cycles.



Kruskal algorithm

• A set of selected edges form a forest.

Prim's algorithm

• A set of selected edges forms a Tree.

edges	Cost ascending order (min)	edges	Corresponding Cost order
(0,5)	10 → Add	(3,4)	22 → Add
(2,3)	12 → Add	(4,6)	24 → Discarded
(1,6)	14 → Add	(4,5)	25 → A Now N-1 edges, rest of
(1,2)	16 → Add	(0,1)	28 } Not Considered
(3,6)	18 → Discarded (because cycle)		

- always start with minimum edge only.

$n = x\ (0 \text{ to } 6) = 7$

$e = x - 1 = 6$

e → edge.
n → node.

ii) PRIM algorithm

eligible edges

0 = ⟨10, 28⟩

5 = ⟨25⟩

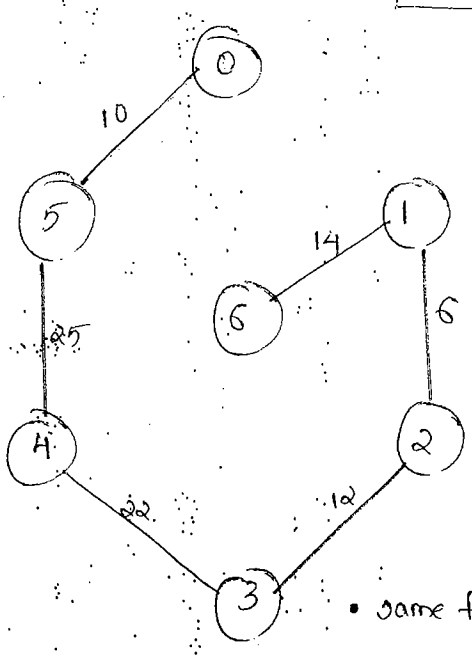
4 = ⟨22, 24⟩

3 = ⟨12, 18⟩

2 = ⟨16⟩

1 = ⟨14⟩

stop



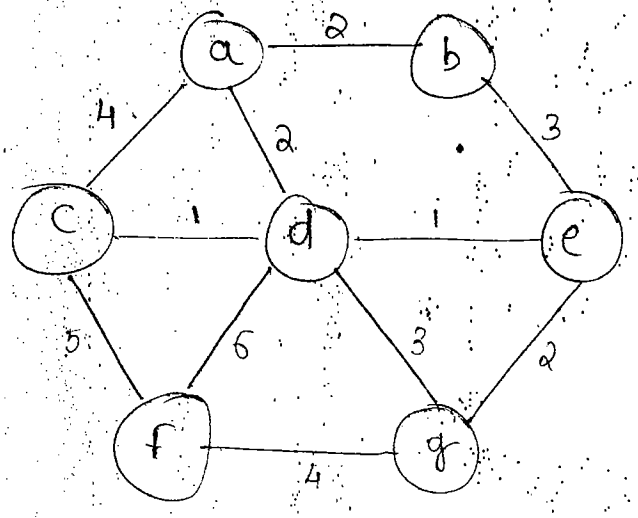
- same for both algorithm

Observation

it can be started from any vertex.

Q.2.

Sol



- ⟨ SEARCH ⟩

Search

Linear or Sequential search	Binary search
• Input = unsorted	• I/P = Sorted.
• Time = O(N)	• Time = O(log n)
O → order	

Termination

```
while (l ≤ H)
{
    M = [ (L+H) / 2 ]
    if X < A[M]
        H = M-1
    else if X > A[M]
        L = M+1
    else found : M position
}
```

Q.

	1	2	3	4	5	6	7	8	9	10
A	75	151	203	275	318	489	524	591	667	727

Sol

ex:- Search for 275

low → 1
high → 10

L	H	M
1	10	5
1	4	2
3	4	3
4	4	(4)

ex 2:- Search for 555

L ≤ H	M
1 ≤ 10	5
6 ≤ 10	8
6 ≤ 7	6
7 ≤ 7	7
8 > 7	

Stop :- Not found

(SORTING)

1) Selection Sort
2) Bubble Sort
3) Insertion
4) Radix
5) Merge
6) Heap sort

⇒ Selection Sort

Select the minimum and place in its appropriate position by swaping.

	①	②	③	④	⑤	⑥	⑦	⑧	⑨	⑩
1	47	11								
2	23	23	23							
3	11	74	74	36						
4	65	42	42	42	38					
5	38	65	65	65	65	42				
6	44	38	38	38	42	65	44			
7	36	44	44	44	44	44	65	65		
8	74	36	36	74	74	74	74	74	74	
9	99	99	99	99	99	99	99	99	99	87
10	87	87	87	87	87	87	87	87	87	99

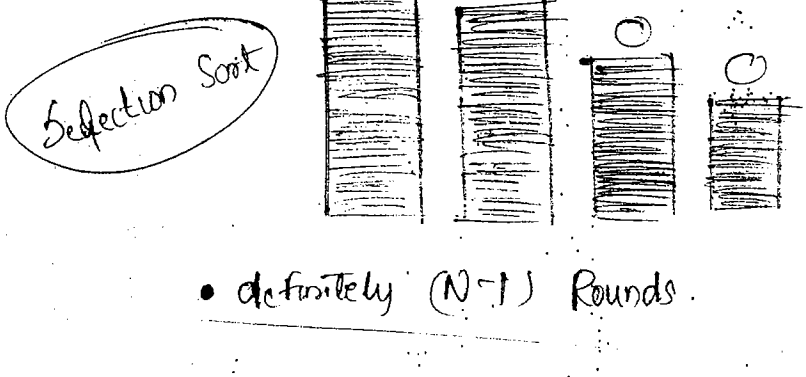
⇒ Bubble Sort

from unsorted to sorted. The movement of the elements are bubbling up as if a bubble moves from down to top.

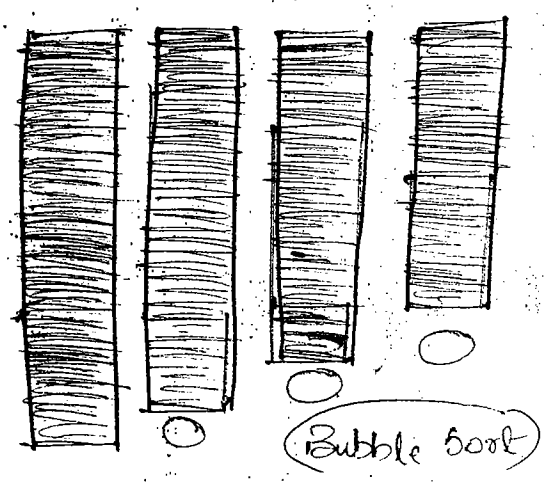
```
a(i) > a(i+1)
    exchange;
```

if no exchange, then stop.

- No exchang stop



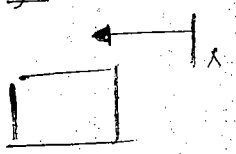
- definitely (N-1) Rounds.



①	②	③	④	⑤	⑥
42	23	23	11	23	23	23
23	42	11	23	38	36	36
74	11	42	38	36	38	38
11	65	38	42	42	42	42
65	38	44	36	44	44	44
38	44	36	44	65	65	65
44	36	65	65	74	74	74
36	74	74	74	87	87	87
99	87	87	87	99	99	99
87	99	99	99			

no exchange then stop.

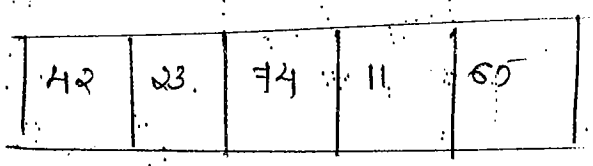
logic

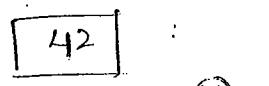


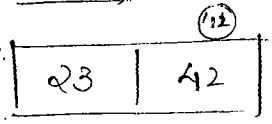
iii) insertion Sort

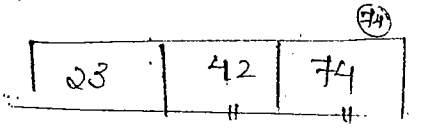
- insert a node by comparing with its predecessor.

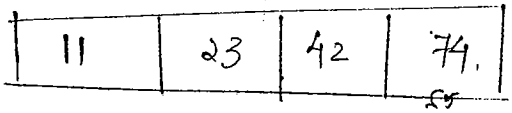
Given :-

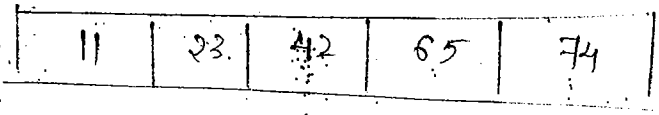








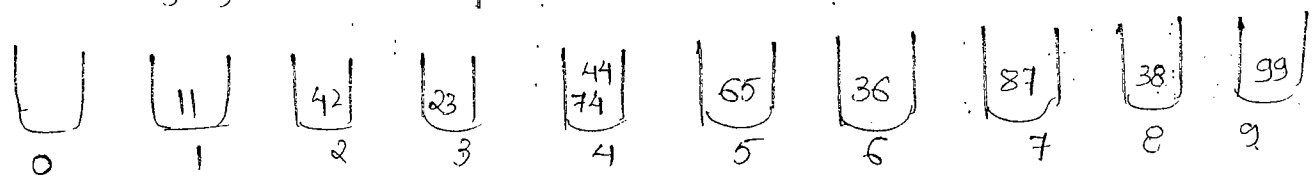




4. Radix Sort

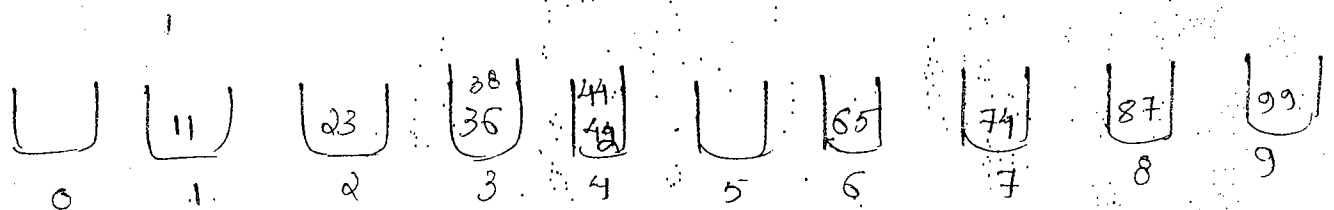
I Round :- unit place

merging bottom to top.



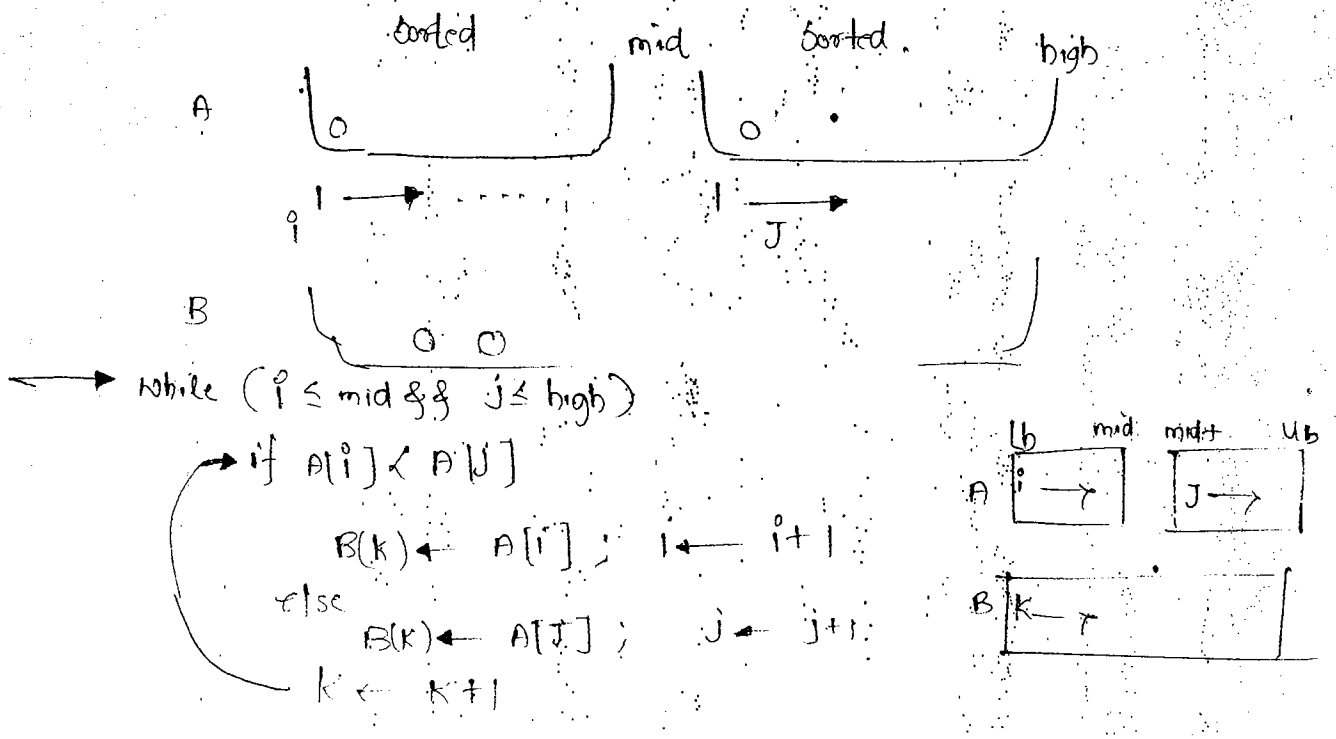
→ 11, 42, 23, 74, 44, 65, 36, 87, 38, 99.

II Round :- Tenth place



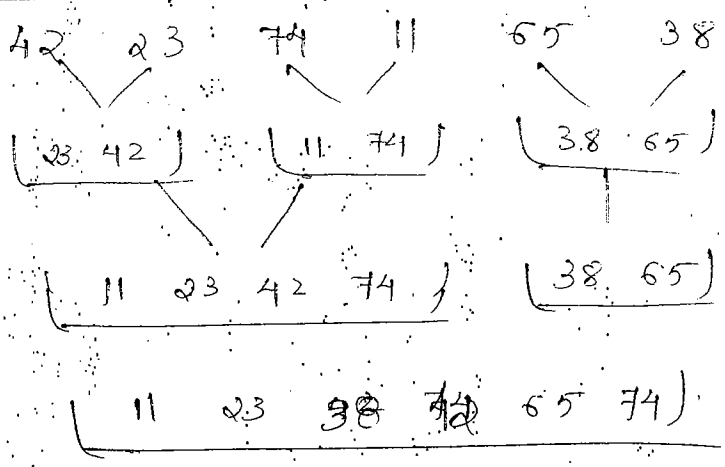
→ 11, 23, 36, 38, 42, 44, 65, 74, 87, 99

5. Merging Sort

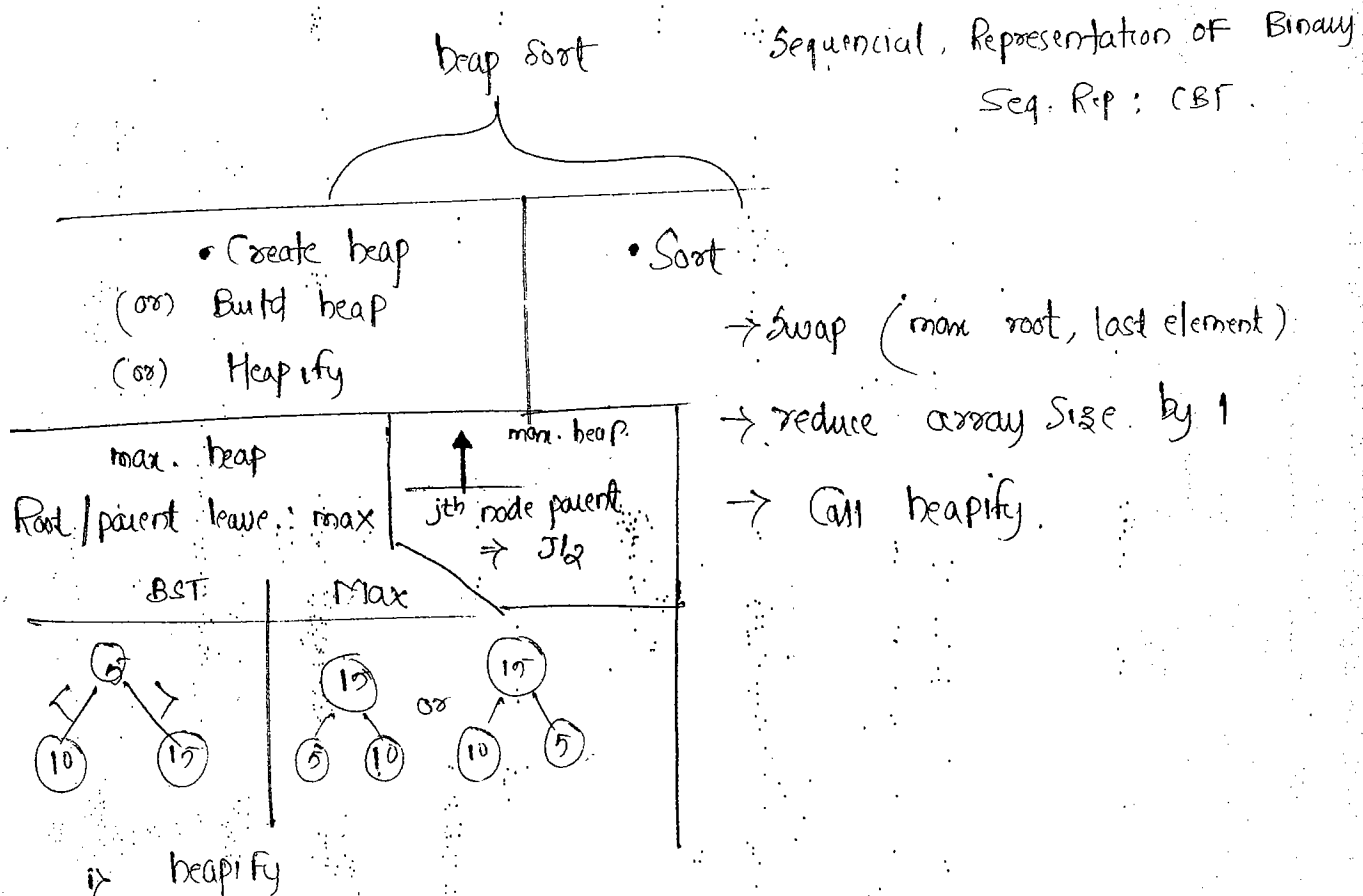


4) Heap Sort

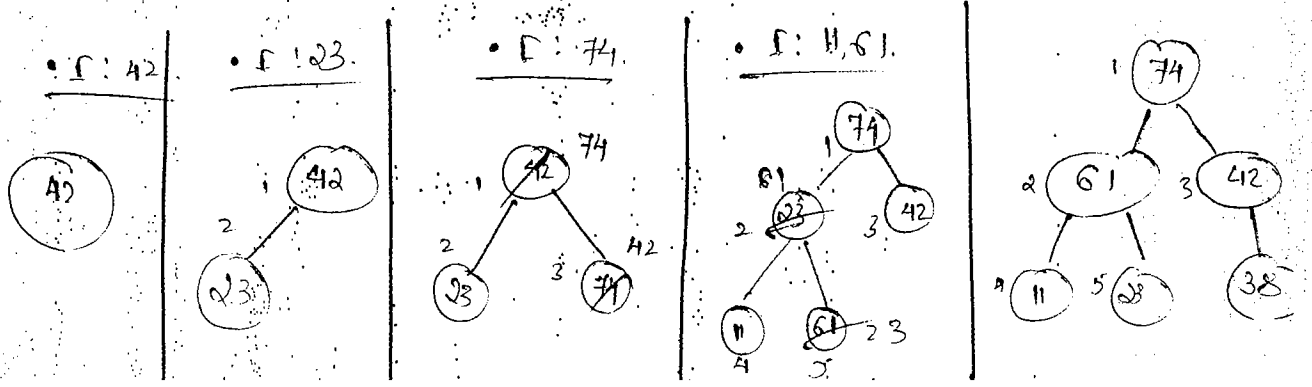
Two way merge :-



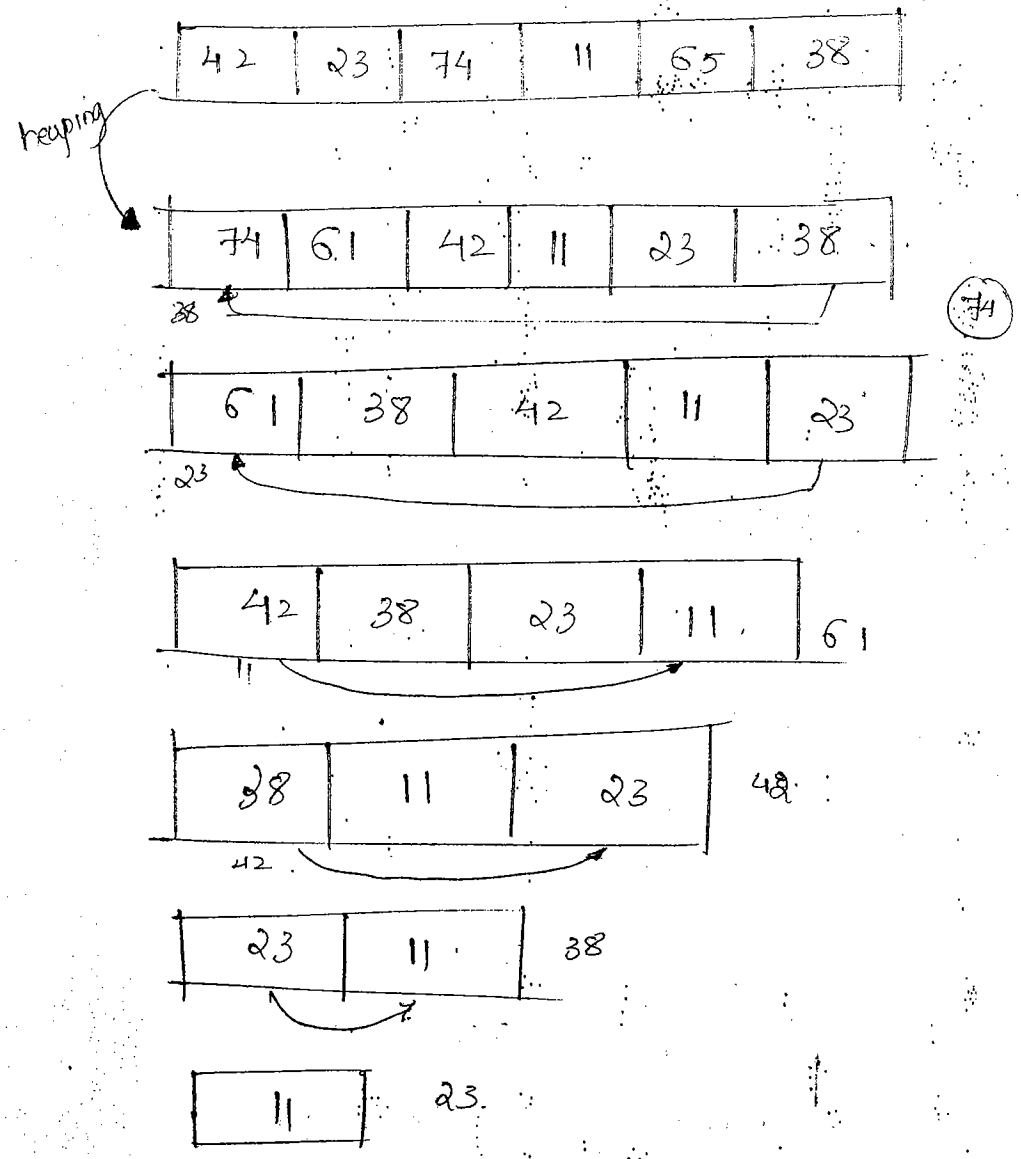
6) Heap Sort :-

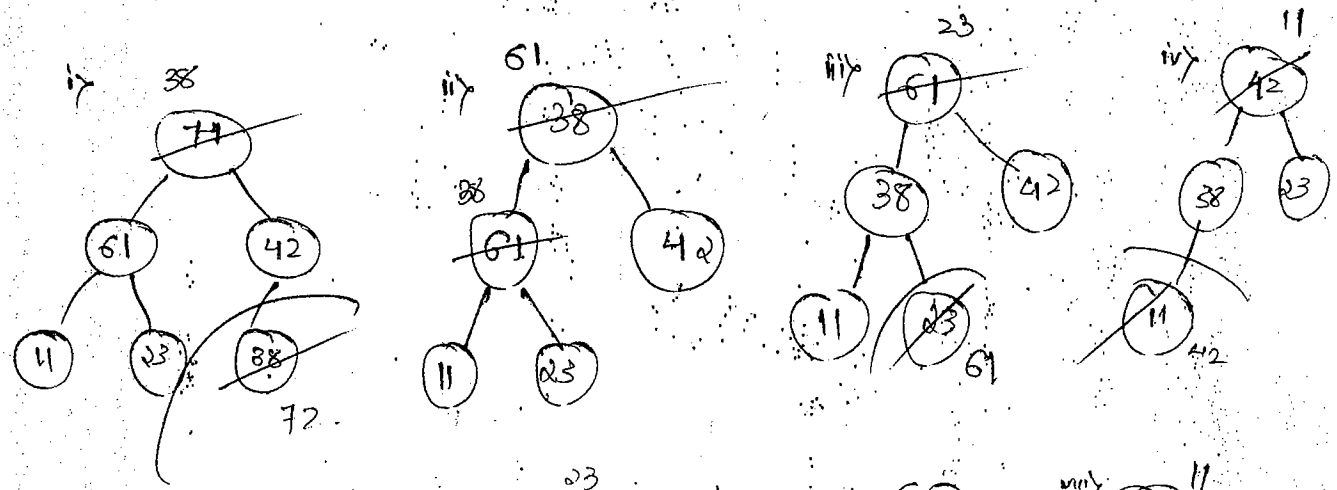


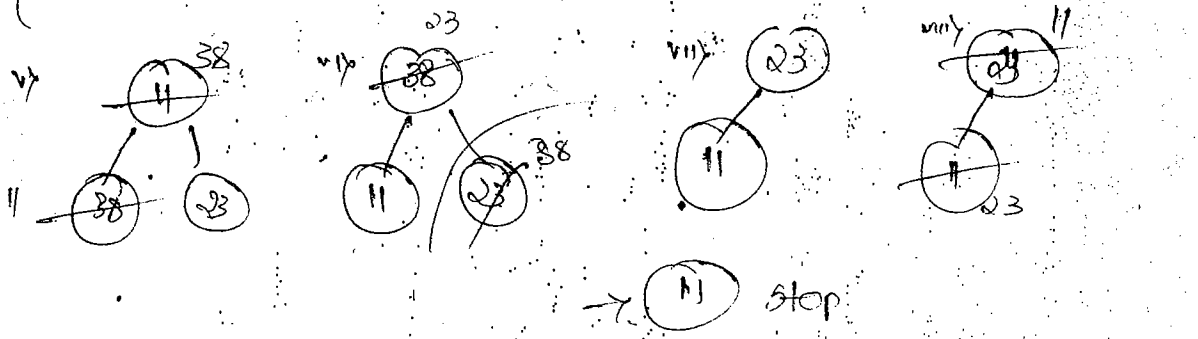
i) heapify



Heap Sort :-


42 23 74 11 65 38
heaping
74 61 42 11 23 38
38
74
61 38 42 11 23
23
42 38 23 11
61
11
38 11 23
48
42
23 11
38
11
23.


i)
38
74
61
42
11
23
88
72
ii)
61.
38
88
61
42
11
23
iii)
23
61
38
42
11
23
61
iv)
11
42
38
23
11
42


v)
38
11
38
23
vi)
23
38
11
23
38
vii)
23
11
viii)
11
23
→ Stop

1.5.2012

(Elements of high level programming : Pascal and 'c')

Machine	year	inventor	country
1) Abacus	5000 years ago	-	china
2) Napier bones	1619	John Napier	Scotland.
3) pascal line	1647	Blaise paxal	france.
4) Differential Engine	1821	charles Babbage → father of Computer	England.
5) Analytical Engine.	1834		

→ 1st programmer : lady Ada lovelace (1825-1852 : 36ye

Generation of Computers

Generation	year	characteristics	Model
I	1940-55	large and heavy electronic vaccum tubes	UNIAC.
II	after 55	Transistor	IBM 7070.

- IBM :- international Business machines
  1924 : started : american Scientist : Hauman Hollerith
- universal automatic Computer
  (ENIAC : Electronic Numerical integrator and Computer)

III	after 1960	IC	IBM 370.
to	after 1970	LSI (µp)	
V	at present	A.I : artificial intelligence	Super Computer.

Classification of Computer — based on
- speed
- accuracy
- memory

{ mi
mi

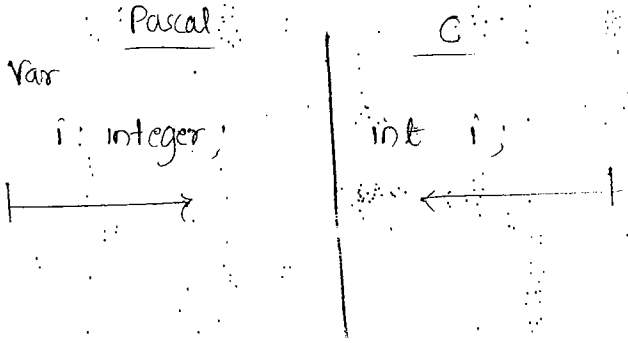
ALGOL 60 : 1960

BCPL 1967 : Martin Richards : Basic Combined programming language

B : 1970 . Ken Thomson

C : 1972 : Dennis Ritchie. ▸ Pascal : 1971 : Niklaus Wirth (To develop OS : UNIX)

Java : 1994 : James gosling of Sun micro system.



• C language is the mirror image of pascal language

C Operators and Expressions

	precedence	Associativity	
16)	posfix .	→ left	high
15)	prefix, unary, !, &	→ Right	
14)	( ) Typecast	→ Right	
13)	* / %		
12)	+ −		
11)	<< >>		
10)	Relatwie <, ≤, >, ≥ =		
9)	== !=		
8)	bit level { & AND, ^ EXOR, | OR		
5)			
4)	logical { && , ||		
2)			
1)	Assignment	→ RIGHT	low

Compound Assignment

i) $x += y \Rightarrow x = x + y$

ii) $x /= y \Rightarrow x = x / y$

iii) $x \%= y \Rightarrow x = x \% y$

iv) $x *= y \Rightarrow x = x * y$

v) $x -= y \Rightarrow x = x - y$

Q) if $x = 2$ then $x *= 3+2$

Sol

$x *= 3+2$

$x *= 5$	$x = x*3+2$
$x = x*5$	$\Rightarrow 2*3+2$ ✗
$\Rightarrow 2*5$	$\Rightarrow 6+2$
$x = 10$	$x = 8$
• Valid ✓	• invalid because '+' is higher precedence over *= Compound assignment

---

Q) $x = 40$, $y = 4$, $z = 4$ then PRINT $X = Y == 3$

Sol

$x = 40$, $y = 4$, $z = 4$

i) $X = y == 3$

$X = (4 == 4)$

$X = 1$

ii) $x = 1$, $y = 4$, $z = 4$

PRINT $x == y == 2$

$1 == 4$ (false)

output = 0

Observation :-

Computer	True	False
PRINT	1	0
READ	Non zero	0 or Null.

Q) $x=y=z=1$ then $++x \;\&\&\; ++y \;||\; ++z$

Sol

$x=y=z=1$

if $++x \;\&\&\; ++y \;||\; ++z$

- false ANDING with whatever is FALSE
- True ORing with whatever is True

$\Rightarrow$ 2 && 2 || whatever

$\Rightarrow$ T && T || whatever

$\therefore$ (2, 2, 1)

---

unary (++) : increment operator

	post fix	prefix	Remark
stand alone	$x++$ $x = x+1$	$++x$ $x = x+1$	} No difference
Association :	$y = x++$ • present value of x in assigned and later x is incremented.	$y = ++x$ here first incremented and assigned.	

Ex :-

```
if x = 5
PRINT ++x
PRINT x
```

o/p → 6, 6

```
if X = 5
PRINT X++
PRINT X
```

o/p → 5, 6

```
if X = 5
X++
PRINT X
```

o/p → 6.

```
if X = 5
Y = X--
PRINT X, Y
```

o/p → 4, 5

```
if X = 5
Y = --X
PRINT X, Y
```

o/p → 4, 4

```
X = 5
X--
PRINT X
```

X → (4)

standalone

Q: $x = y = z = 1$ then $++x \;||\; ++y \;\&\&\; ++z$

Sol $x = y = z = 1$

$++x \;||\; ++y \;\&\&\; ++z$

$\Rightarrow$ 2 || (&& true coming whatever)

2; 1, 1

iii) $++x \;\&\&\; ++y \;\&\&\; ++z$

Sol $++x \;\&\&\; ++y \;\&\&\; ++z$

2; 2, 2

iv) $++x \;||\; ++y \;||\; ++z$

Sol 2; 1, 1

Q: $x = y = z = -1$ then

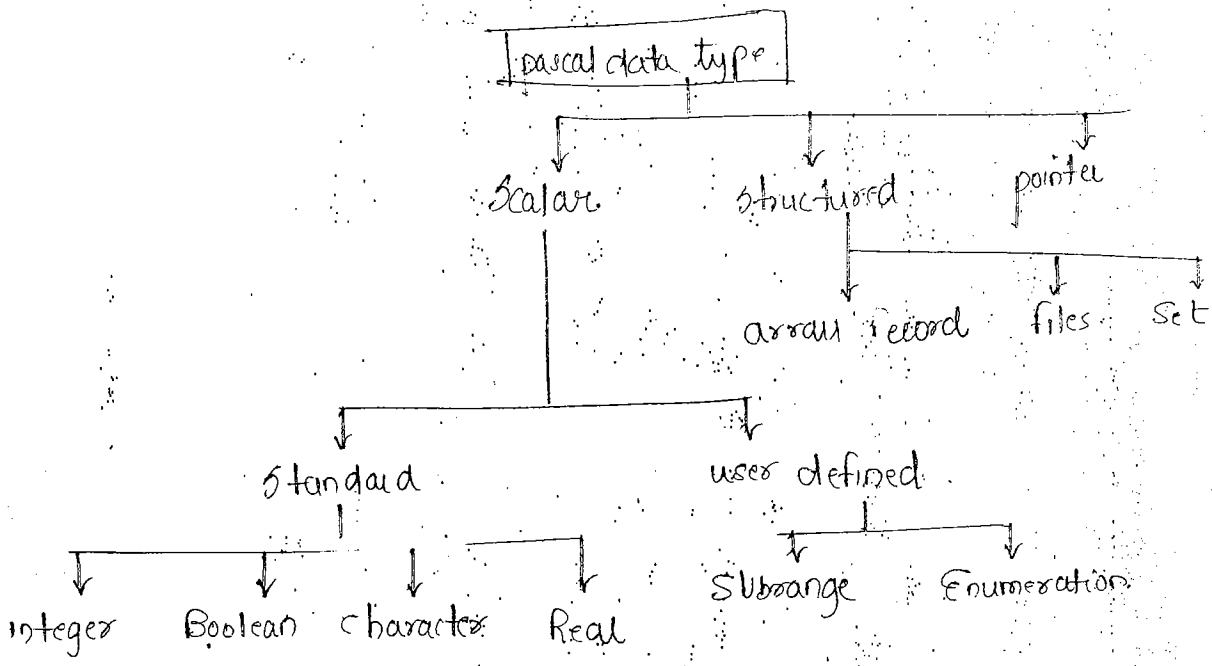
i) $++x \;\&\&\; ++y \;||\; ++z$ ⟶ 0; -1, 0

ii) $++x \;||\; ++y \;\&\&\; ++z$ ⟶ 0, 0, -1

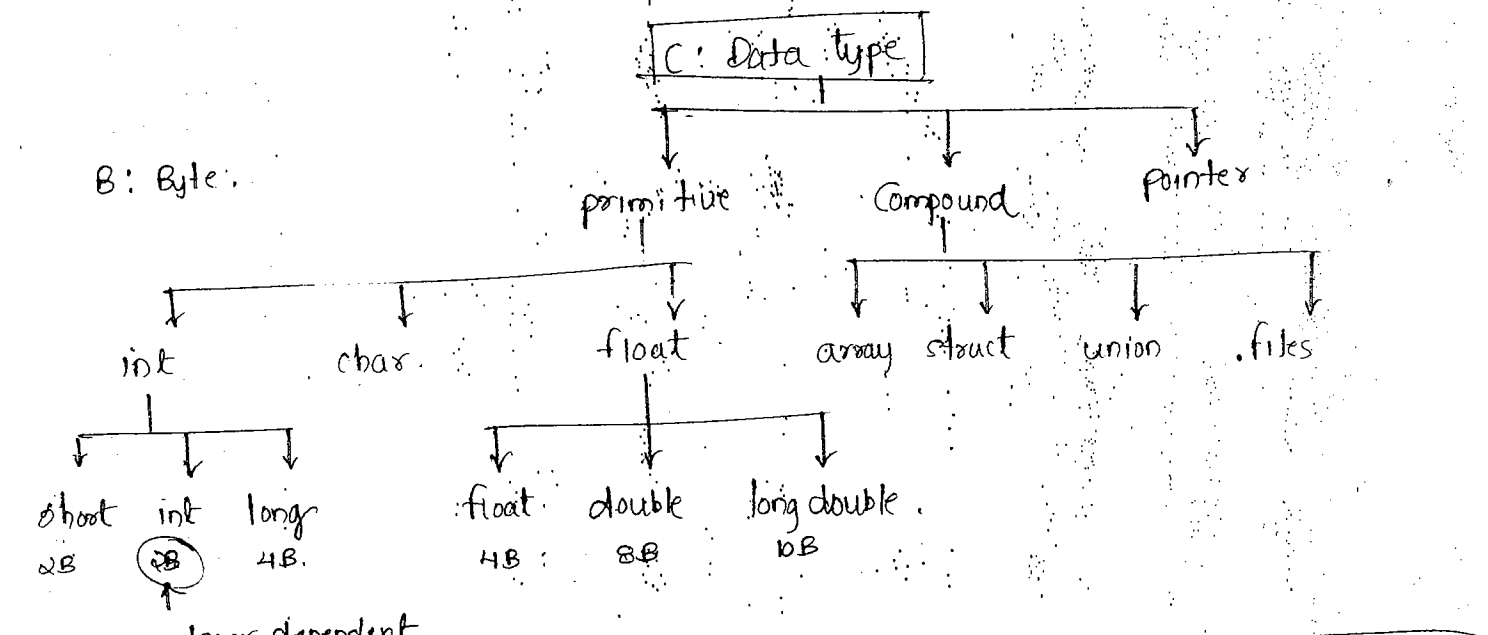
iii) $++x \;\&\&\; ++y \;\&\&\; ++z$ ⟶ 0; -1, -1

iv) $++x \;||\; ++y \;||\; ++z$ ⟶ 0, 0, 0

pascal data types

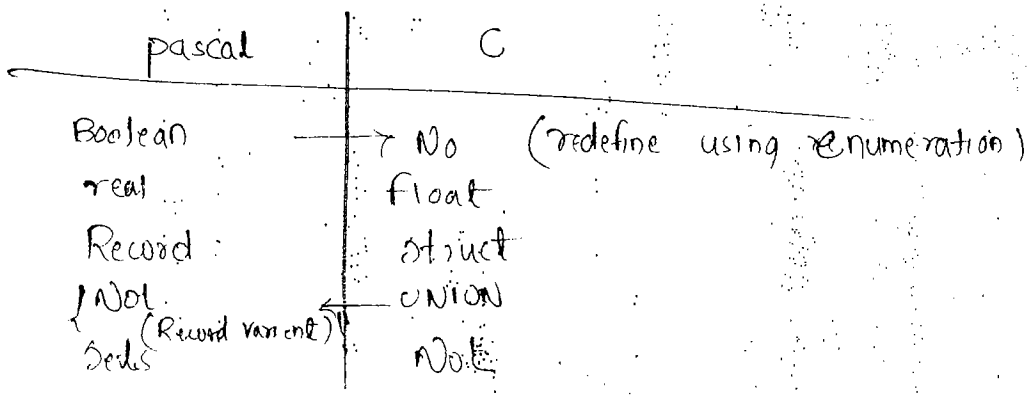


C: Data types



B: Byte.

Observation



pascal

```
var
    i : integer;
    r : real;
    c : character;
    b : boolean
```

Ex:-

```
i := 10;
r := 2.8;
c := 'a'
b := false
```

C-language

```
int i;
char c;
float f;
```

```
i = 10;
c = 'a';
f = 2.8;
```

i) pascal

```
var
    d : digits;
    f : family;
```

```
Type
    digits = 0.....9;        (* sub range *)
    family = (father, mother, bro, sister);   (* enumeration : list down *)
```

ii) C-language

```
enum family { father, mother, brother, sister };

enum months { Jan, feb, march, ... };
```

lower boundary = 0.

```
enum months { Jan = 1, feb, march };
```

```
Type
    one dimensional = array [1...4] of integers;
Two dimensional = array [1...2] of
                  array [1...3] of integers;

var
    a : one dimen;
    b : Two dimen;
```

(OR)

```
var
    a : array [1...4] of integer;
```

```
int a[4];
```

a_0	a_1	a_2	a_3

```
int b[2][3];
```

00	01	02
10	11	12

- array is an homogeneous elements (Similar data type)
- structure or record ⟺ heterogeneous elements (Dissimilar data type)

i) in C- language

```
Struct Sample
{
  char c;
  int i;
};
```

Prototype

memory is Not allocated

```
Struct Sample s;
```

Declaration

memory is allocation

```
s.c = 'a';
s.i = 10;
```

Defination

(Direct Selector)

ii) in pascal

```
Type Sample = record
       c : character;
       i : integer;
     end;
```

Prototype

```
Var s : Sample;
```

Declaration

```
with s do
begin
  c := 'a';
  i := 10;
end
```

Definition

Q. Write a pascal program for Student's record.

Sol:

```
Student
Type  Sample = record
      name = array [a,b.....z] of characters;
      Marks = array [1,2,.....] of integers;
      End;
Var x : student;
With x do
      begin
        name := "KARAN";
        marks := 90;
      End;
```

C: UNION

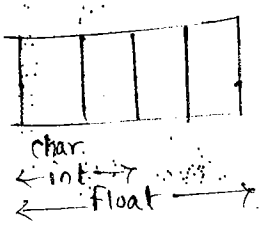
→ Heterogeneous

→ memory is allocated for the larger component and all components share the memory

→ better utilization of available memory

→ always holds the latest value.

Ex:-
```
UNION Sample
{
  char c;
  int i;
  float f;
};
Sample s;
s.c = a;
s.i = 10;
```



C: STRUCTURE

→ heterogeneous

→ memory is allocated for the components or field or members

→

Ex:-
```
struct Sample
{
  char c;
  int i;
  float f;
};
struct Sample s;
s.c = a;
s.i = 10;
```

c() // 1
i() // 2
F(m) // 4

≥ 7

Pascal : RECORD VARIANT

→ allow us to set up a record whose precise a structure may slightly be different for different variables.

→ Achieved by using Case.

Rules :-

1) varient is allowed only in one field.

2) must be last record.

Q1

```
Type   status = (married, divorced, widowed);

       marital = record
                   name : array [1....20] of characters;
                   age : integer;
                   Sex : (male, female);
                   Case 'status' of

       married : (children: integer;
                  Spouse name : array [1....20] of character;);

       divorced : (divorced date : record; month : integer; year : integer;
                   end;
                  );

       widow : (year of death : integer; insurance : boolean;);
                   end;
```

Ternary Operator (or) Conditional Operator

Condition Expression ? True expression : False expression

Ex:-

Max = (a>b) ? a : b

Say a = 5 b = 3 → 5>3 → True → 5 → Max = 5

a = 5 b = 30 → 5>30 → false → b = 30 → MAX = 30

Example :-

i) a = 2 b = 3 c = 4

c += a>b ? a-- : ++b

2>3 → F → 4

c += 4

a = 2 b = 4 c = c+4 → 8

2 ; 4 ; 8

ii) c *= a<b ? a-- : ++b

2<3 → T

c *= 2

c = c*2

c = 1*2

a = 2 c = 2

1, 3, 2

9.5.2014

STORAGE CLASSES IN C

i) Automatic
ii) Static
iii) Register
iv) Extern.

Local variable :- Variables are declared in a block and visibility is confined to that block.

Global variable :- visibility is applicable to the entire program and can be accessed from any block / function.

Automatic

→ local variable
→ At the time declaration, automatically created and when it goes out of scope automatically destroyed / vanished.
→ Auto keyword : optional.
→ When not initialized, it holds Junk values
→ Everytime reinitialized

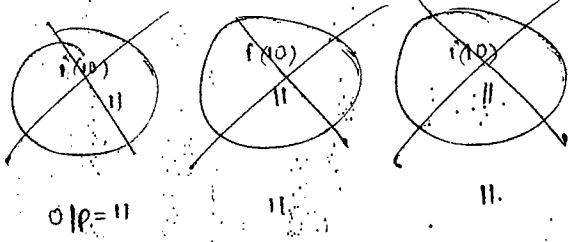
Static

→ local variable
→ As the name indicates static it persists even after out of scope.
→ Mandatory
→ When not initialized it holds '0'
→ Only on the first involcation, it is initialized

automatic

```
Void Do()
{
  int i=10;
  printf("%d", ++i);
}

Void main()
{
  Do();
  Do();
  Do();
}
```

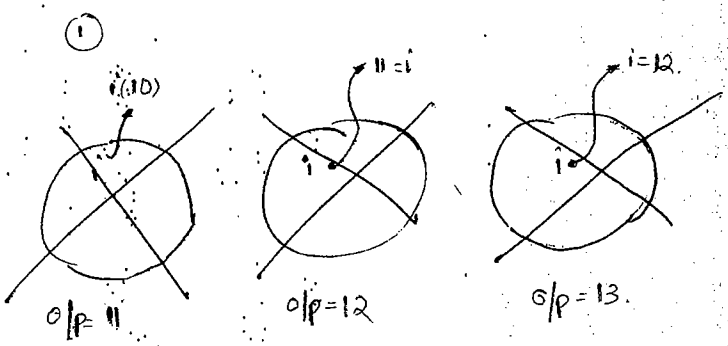


o/p = 11 11 11.

Static

```
Void Do()
{
  Static int i=10;
  printf("%d", ++i);
}

Void main()
{
  Do();
  Do();
  Do();
}
```



o/p = 11 o/p = 12 o/p = 13.

ARRAYS AND POINTERS

pointer variable :- variable holds address of another variable (data type

P*q → multiplication

int *i; → reference address of another variable.

*P = 20 → Dereference operator. (Referencing by the address)
value stored in the reference of address.

- One byte is mapped to another byte is called **loosely typed**

C: is a loosely typed but pointer is a strongly typed.

Observation :- on 16 bit machine

```
char *cp;
int  *ip;
float *fp;
```

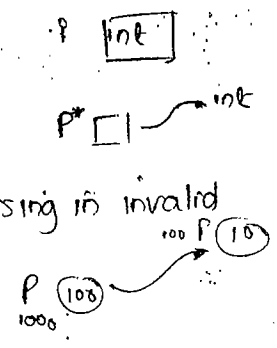
- all pointers are capable of holding addresses and address size is machine dependent.

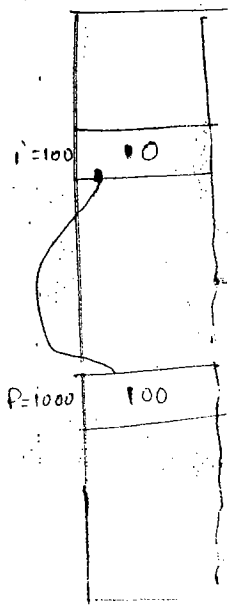
```
char c;   // 1B
int  i;   // 2B
float f;  // 4B
```

C: loosely typed but with pointers strongly typed but pascal & ctt is strongly type

name = inside
& = outside
* = Content (Jump)

```
int i;
int *P;
P = 100   // absolute addressing is invalid
P = &i;
*P = 10;
```






```
Printf("%d", i); PF("%u", &i); PF("%u", P);
PF("%d", *P); PF("%u", &P);
o/p ; 10, 100, 100, 100, 1000
```

[ ]	*
Early binding	later binding
Static binding	Dynamic
Compile time binding	Run time
a[i] user friendly operator	* [a+i] machine friendly operator
Constant pointer a+1 ; valid a = a+1 ; invalid Read Only ; Can't write	pointer variable p = p+1 ; valid

a = Name of array ⇒ starting address of array

a = &a[0];

```
int a[] = {10, 11, 12, 13};
int i = 3;
printf("%d", i[a]);
```

Loc a(i) = lb + (i - lb) * c ;

Loc a(i) = &a(a) + (i - 0)

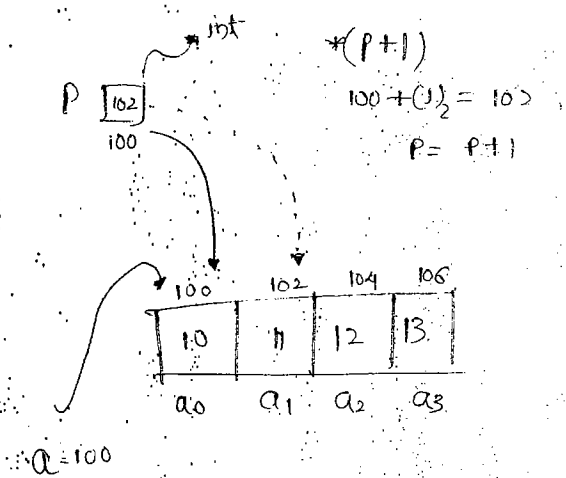
Loc a(i) = a + i ;

a[i] = * (a+i)

→ Scalar arithmetic

i[a]
⇒ * (i+a)
⇒ * (a+i)
⇒ a[i]

Example ②




```
int *p;
p = &a[0];   (or)
p = a
```

```
printf("%d", *(p+1));
printf("%d", p[1]);
```

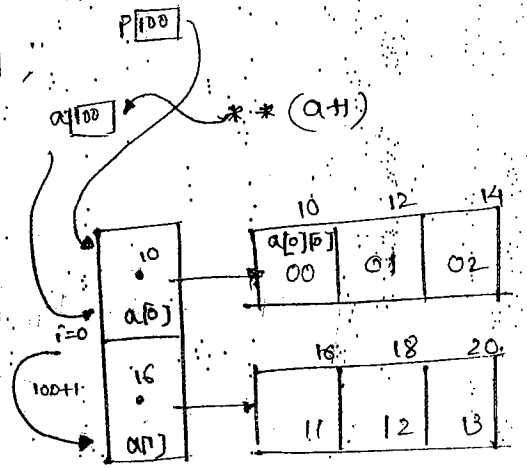
above : same.

Valid
a+1
100 + (1)·2
⇒ 102

2-D dimensional array and pointer

P[ ][ ] ⇒ *P[ ] ⇒ **P all are same.

```
int a[2][3];
int **p;
p = a;
// valid
```



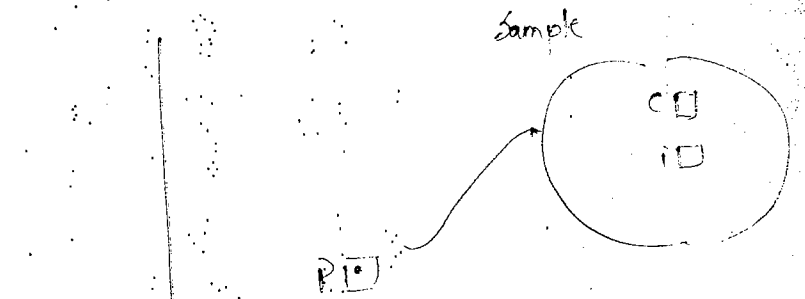
a [i] [j]
* (i + j)
* (a[i] + j)
* (* (a+i) + j)

a[0] = *[a+0] = *a

*a	a[0]
**a	* [* (a+0)+0]
* (*a+1)	* (a[0]+1) = a[0][1]
* *(a+1)	* (a[1]+0) ⇒ a[1][0]
**a+1	a[0][0] + 1

pointer and structure

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




```
struct Sample s;
struct Sample * P;
P = &s;
```


```
*P.c = 'a'
*P.i = 10
```

} invalid

above = Segmentation fault (core dumped)

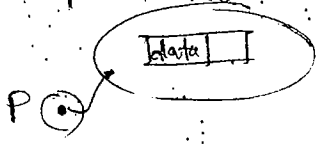
. is higher precedence over *

(*P).c = 'a' (or) P ⟶ c = 'a'

(*P).i = 10 (or) P ⟶ c = 10

⟶

→ indirect selection
→ user friendly operator
P ⟶ data



→ Direct selection.
→ machine friendly
(*P).data

Subroutines or Subprocedure or routines

i) pascal

pascal (it doesnot Supports without keywords)

procedure	function
→ Doesnot return a value but Carryout Certain process.	→ Return a value.
→ procedure Swap (a, b) process	function Ex:- 1 fact (4) return = 4×3×2×1 = 24.

ii) C

- Supports procedure and function without keywords.

	Ex:
Void Swap (a, b) ↑ procedure.	int Max() ↑ return type integer function

---

P: function

```
function MAX(a:integer; b:integer):
                    integer ;
begin
    if (a>b)
      then MAX := a;
    else
      MAX := b;
end;
```

C: function

```
int MAX (int a, int b)
{
    if (a>b)
      Return a;
    else
      Return b;
}
```

P : procedure

```
procedure Do(a:integer, b:integer);
begin
  ____
  ____
  ____
end;
```

C - procedure

```
void (Do int a, int b)
{
   process
}
```

<RECURSION> (or) Non-iterative Statement

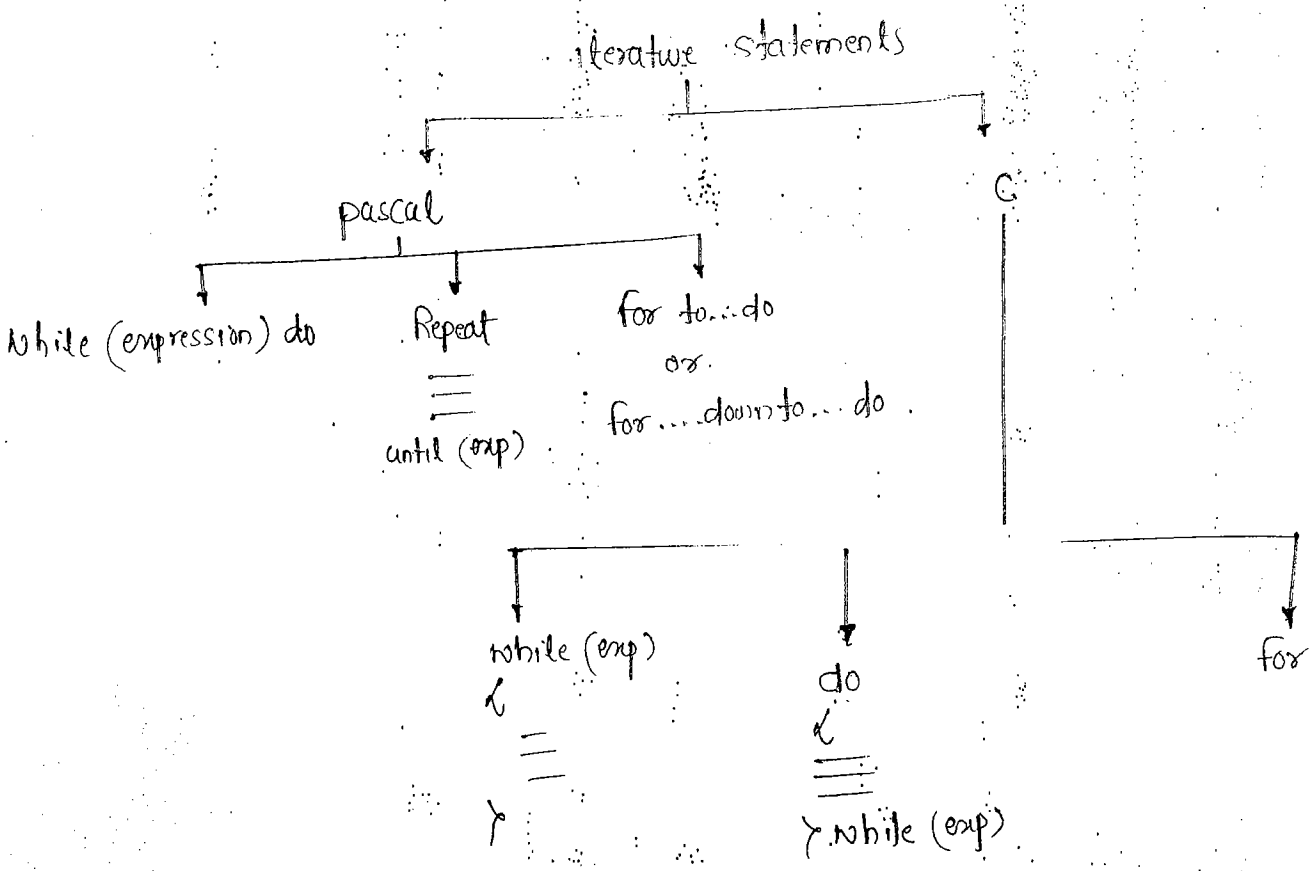
i) pascal

```
function power ( x : integer ; y : integer) : integer ;
  begin
    if (N=0)
      then power := 1 ;
    else
      if (x=0)
        then power := 0 ;
        else if (N>0)
          then power := x * power (x, N-1) ;
          else if : (N<0)
          then power = 1/x * power (x, N+1);
  end ;
```

ii) C

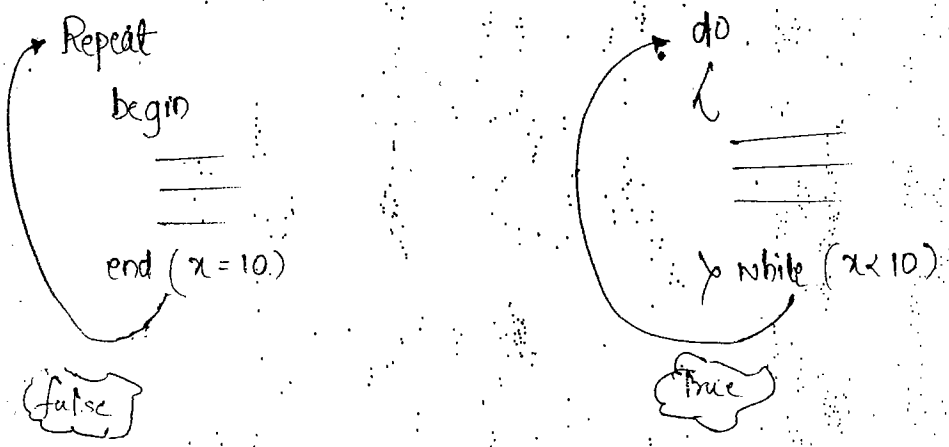
```
int pow (int x, int N)
{
  int (N==0)
      Return 1 ;
  else if (x==0)
      return 0 ;
  else if (N>0)
      return x * pow( x, N-1 );
  else if (N<0)
      return 1/x * pow( x, N+1);
}
```

Iterative statements (or) NON RECURSION



while	do while
1) Top testing	1) Bottom testing
2) Once condition is true then only the body of the loop is executed.	2) Min. of once, the body of the loop is executed. Then condition is evaluated.

Observation:



Overall structure of program

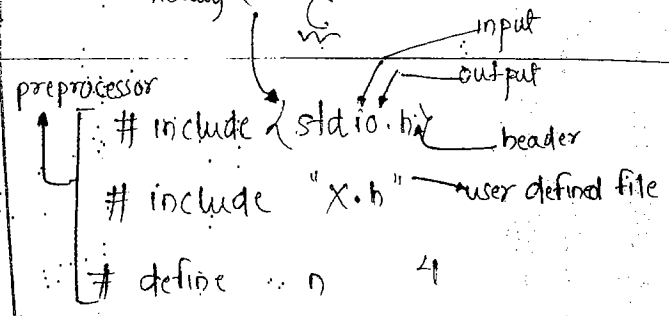
Pascal

```
program Total (input, output);
    label
    Constant
    Type
    Variance
    procedure
    function
begin
    ____
    ____
    ____
    ____
end
```

- Core part is very large.

C

library (built in)




```
void main()
{
    ___
    ___
    ___
}
```

- Core part of C program is very small.

I/P and O/P statements

	Pascal	C language
i) Input	read (a) ; readln (a);	Scanf ("%d", & a)
ii) Ouput	write (a) ; writeln (a); ln = newline	printf ("%d", a) \n → new line.

Q. write a program to print Sum of first 100 integers both in pascal and C language.

Sol

pascal language

```
program sum (input, Output)
  var
   i, Sum := integer
  begin
   i:=1
   Sum := 0;
   while ( i ≤ 5) do;
    begin
      Sum := Sum + i;
      i := i+1;
    end;
  writeln (Sum);
end
```

```
for i:=1 to 5 do
  begin
    Sum := Sum + i;
  end
```

C Language

```
#include <stdio.h>
void main()
{
int i, Sum;
  i=1;
  Sum=0;
  while (i<=5)
  {
    sum += i;
    ++i;
  }
  printf ("%d", Sum);
}
```

for (initialization; condition; updation)

for (i=1; i ≤ 5 ; ++i)

Q. When do you refer for

Sol When we know well in advance for how many time the body of the loop is executed.

Ex:- Print primes below first 100 integer.

10.5.2012

parameter passing Technique

pascal

→ Call by (pass by) value (cv)

→ Call by variable (cr)
reference

C

→ Only Call by value
but in c++, we can call by value and reference.

arguments = parameters

Actual parameters (main) ex:- a,b

formal parameters (routine) ex:- x,y.

pascal :-

```
program pass(input, output);
    var a,b: integer;
    procedure change( var x: integer; y: integer)
      begin
        y:= y-x;
        x:= x+1;
      end;
begin
    a:= 15;
    b:= 10;
    write(a,b);
    change(a,b);
    write(a,b);
end
```

(Variable)
pass by reference
pass by value.

Sol

Logic :-

change (a,10)

change (x, y□)

y[10]

i) y = 10-2 = 8

ii) x = 15+1 = 16

x → a[15] → 16

b[10]

change (a,b)

i) Call by value :-

i) Whenever there is a ~~ref~~ invokation from the main, the actual parameters values are passed hence the name Call by value.

ii) memory is allocated for formal parameters and the passed values are clamped into them.

iii) Whatever updations are done for the formal parameters will not effect the actual parameter.

ii) Call by reference (Variable)

i) whenever there is a invokation from the main, the formal parameters will reference to actual parameter. hence the name Call by reference.

ii) memory is not allocated for the formal parameters.

iii) whatever updations are done for formal parameters will effect the actual parameter.

---

i) Call by value

```
program par(input,output)
var a,b : integer;
    procedure change (a : integer)
      begin
        a=a+5;
      end;
begin
    x:=3 ;
    write(x);
    change(x);
    write(x);
end.
```

change(3)
change(a)
a(3) 8
3+5=8
x=3
o/p = 3
change(x);
o/p = 3.

change for Call by ref.

ii) Call by reference

```
(var a: integer)
  change(x)
change(var a)
```

a
a = a+5
a = 8

x 8

o/p = 3.
change(x);
o/p = 8.

Q. IES. 2003

Sol

```
var i, j : integer
procedure y (P,q : integer) ⇒ y (var p:integer; var q: integer)
                                  → o/p will change i.e 3,2.
  begin
    p := p-q ;
    q := p+q ;
    p := q-p ;
  end
begin
    i := 2
    j := 3
    y (i, j)
end
```

o/p will remains same as the previous value i.e 2 & 3.

if both parameters to y are passed by reference, then what are the values of i, J.

a) 0, 2 b) 1, 5 c) 2, 3 d) 3, 2.

Q. IES 1999.

Sol

```
function Calc ( var A: real ; B : real) : real
          var X, Y : integer
  beain
      X := 3.0;
      Y := 3.0;
      Calc := 5.0 * A + (B - A)
  end
```

Calc block.
Cal (x, 1.0)
Cal (A, B□)
local { X = 3, Y = 3 ; A = 7 ; B = 1
5 * 7 + (1 - 7) ⇒
X = 7.0 ; mai
Y = 1.0 ;

a) 15
b) 29
c) 13
d) 31

if this function was caued

```
X := 7.0;
Y := 10;
R := Cal(X, Y)
```

The value of R would be

Q: write a pascal program to swap the two given numbers.

Sol

```
program pair(input, output)
var a,b : integer;
procedure swap (var x:integer; var y:integer)
    var t : integer
  begin
      t := y;
      y := x;
      x := t;
  end;
begin
 a:= 2; b := 3;
write('before swap');
write (a,b);
 swap ( a,b);
writeln('After swap');
  write (a,b);
end
```

* if var is not there o/p will be (2,3).

logic

swap (a,b)

swap (x,y)

x y t 3

a 2 3 b 3 2

swap (a,b)

∴ o/p = 3,2.

C++

Stand alone.

int &x = a;

x is an alias to 'a'

by passing

swap (int &x, int &y)

swap (a, b)

Q. Write the above same program for swaping the integer in C & C++.

Sol

in C++

```
#include <iostream.h>
void swap (int &x, int &y)
{
  int t;
  t = y;
  y = x;
  x = t;
}
void main()
{
  int a,b;
  a=2; b=3;
  cout << "before swaping";
  cout << a; cout << b; swap(a,b);
  cout << "after swaping";
  cout << a << b;
}
```

in C language

```
#include <stdio.h>
void swap (int *x, int *y)
{
  int temp;
  temp = *y;
  *y = *x;
  *x = temp;
}
void main()
{
  int a=2, b=3;
pf(" before swaping
  a=%d, b=%d", a,b);
swap(&a, &b);
pf(" after swaping
  a=%d, b=%d", a,b);
}
```

- C doesnot supports call by reference.