P) $N(H)_{max}$ = Max. no of nodes in a BT

with recursion

$N(H)_{max} = 1$, $L = H = 0$ → ∴ root is at level = 0 (P)

$= N(H-1) + ?$, $(L=H) > 0$

L → Level
H → Height

without recursion

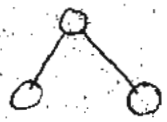
$N(H)_{max} = ?$

Sol:

$L = H = 0 \Rightarrow 0$

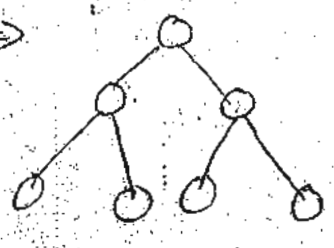
$N(0) = 1$

Sol

$L = H = 0 \Rightarrow$ 

$L = H = 1 \Rightarrow$

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

$L = H = 0 \Rightarrow$ 

$L = H = 1$

$L = H = 2$

$N(2) = 7$

$= 3 + 4$

$= N(1) + 2^2$

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

without recursion:

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

→ $2^3 - 1 = 2^{3+1} - 1$

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

at level = 0

level
height

(b) $N(H)$ = Max. no. of nodes in a BT

@ with recursion

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

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

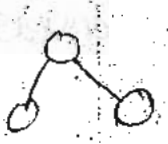
without recursion

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

Sol:

$L = H = 1 \Rightarrow$ O

$N(1) = 1$

$L = H = 1 \Rightarrow$ 

$L = H = 2$

$N(2) = 1 + 2$

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

without recursion:

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

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

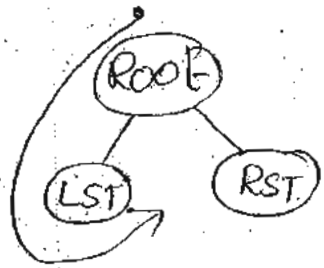
$N(H)_{max} = 2^H - 1$

Recursive algorithms for traversals:

LST - Left subtree
RST - Right subtree

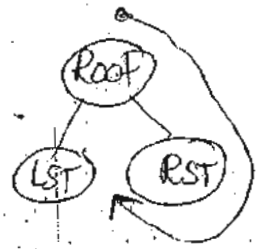
Preorder:

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



converse preorder:

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



Inorder:	converse Inorder:
1. LST	1. RST
2. Root	2. Root
3. RST	3. LST

postorder:	converse postorder
1. LST	1. RST
2. RST	2. LST
3. Root	3. Root

~~C Prog. Code:~~

'c' lang code for Inorder.

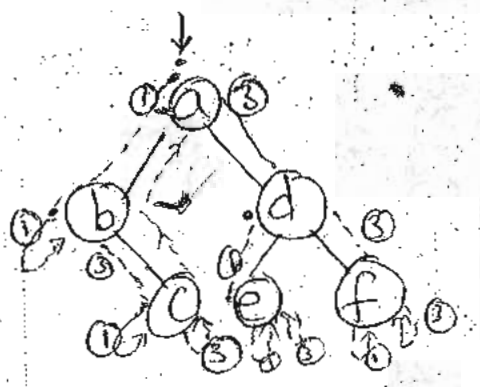
```
void Inorder (struct BTnode *t)
{
    Pf (t)
    {
        1. Inoder (t->Lc);
        2. Printf ("%d", t->data);
        3. Inorder (t->Rc);
    }
}
```

similarly write for other traversals.

p) what does DO() do on following BT

```
void DO(struct BTnode *t)
{
    if (t)
    {
        1. DO (t->Lc);
        2. Pf ("%d", t->data);
        3. DO (t->Rc);
        4. Pf ("%d", t->data);
    }
}
```

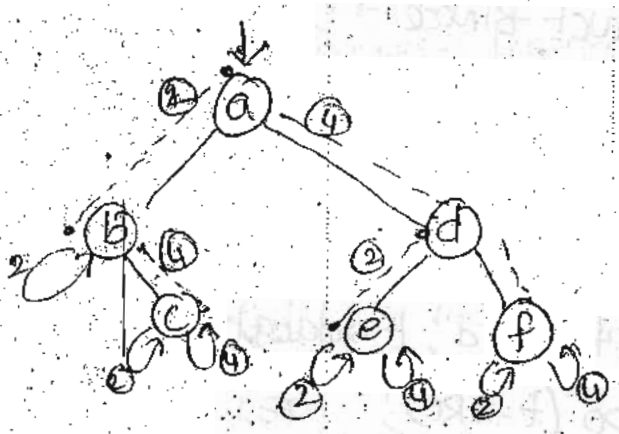
DO
1. Lc
2. Pf
3. Rc
4. Pf



o/p: bccbaeedffda

p) what does TO() do on BT?

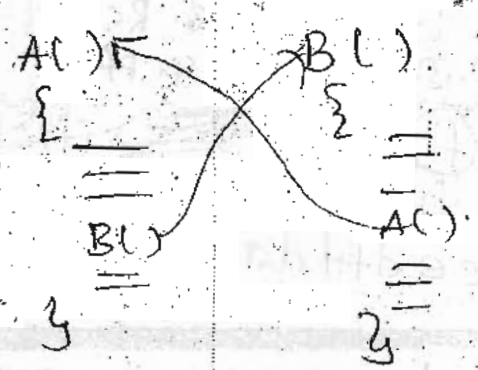
```
void TO (struct BTnode *t)
{
    if (t)
    {
1.      printf ("%d", t->data);
2.      TO (t->LC);
3.      printf("%d", t->data);
4.      TO (t->RC);
5.      printf ("%d", t->data);
    }
}
```



TO
1. Pf
2. RC
3. Pf
4. RC
5. Pf

o/p: abbcccbadeeedfffd a

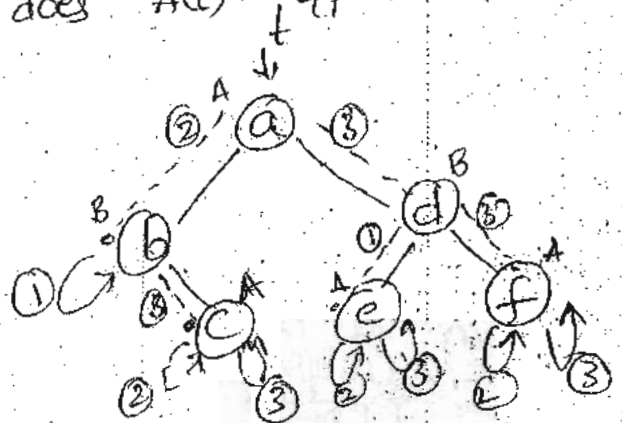
Indirect recursion:



p) consider following routines

```
void A (struct BTNode *t)
{
  if (t)
  {
    printf ("%d", t->data);
    B (t->LC);
    B (t->RC);
  }
}
void B (struct BTNode *t)
{
  if (t)
  {
    A (t->LC);
    printf ("%d", t->data);
    A (t->RC);
  }
}
```

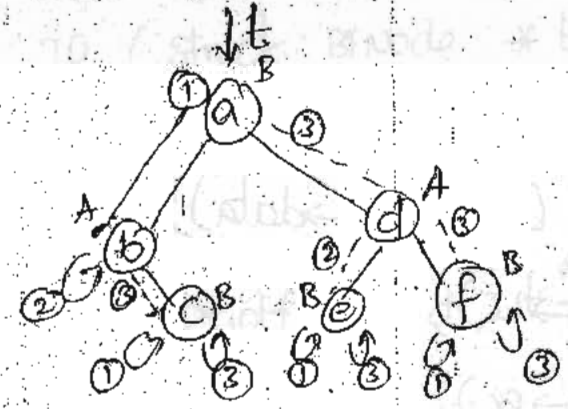
i) what does A(t) dp on following BT



A	B
1. Pf	1. A(LC)
2. B(LC)	2. Pf
3. B(RC)	3. A(RC)

o/p: abcedf

2) what does B(t) o/p on BT



o/p: bcadef

15/8/11
09q
(P)

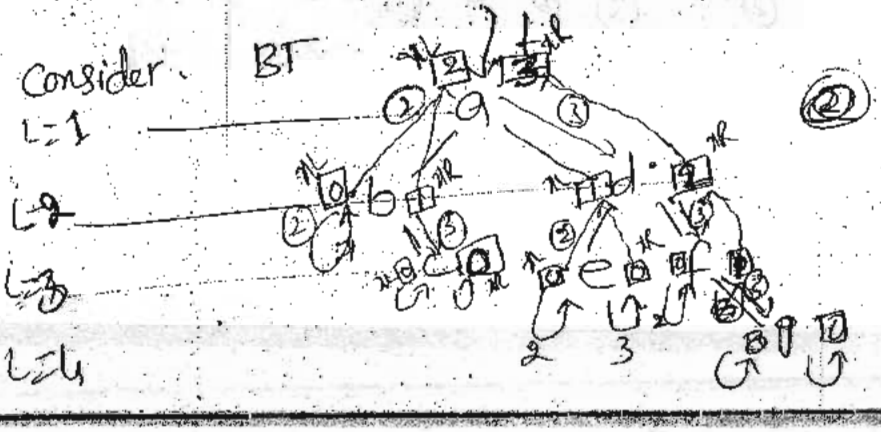
3) int DO (struct BTnode *t)

```
{
1) if (!t)
       return 0;
2) int xL = DO (t->LC);
3) int xR = DO (t->RC);
4) if (xL > xR)
       return = xL+1;
   else
       return = xR+1;
}
```

what DO() return on a BT?

Sol:

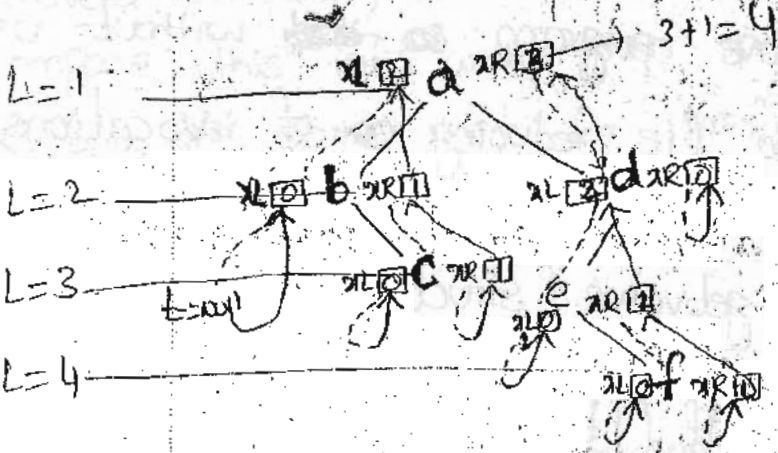
Consider BT



o/p: 4

Sol:

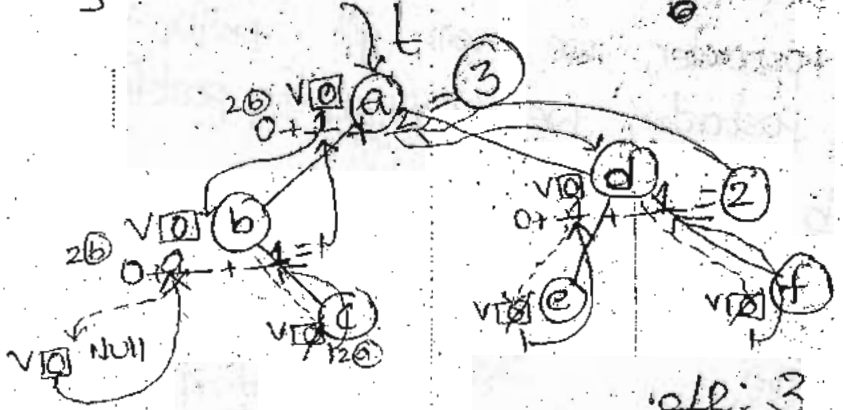
It returns the height / level / depth of tree with root at level 1



(P) what does getvalue() returns on BT ?

```
int getvalue ( struct BTNode *t)
{
1) int value = 0;
2) if (t)
   {
2a) if ((t->LC==NULL) && (t->RC==NULL))
        value = 1;
    else
2b) value = value + getvalue(t->LC) + getvalue(t->RC);
   }
3) return value;
}
```

Sol:



It counts terminal nodes.

p) Modify above program ~~to find~~ without using local variable (ie reducing no. of invocations)

Sol:

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

Applications of trees:

a) Expression (or) Arithmatic trees:

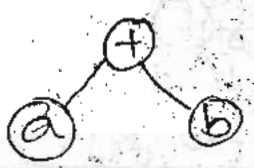
operator = root/parent level

operand = leaf/children level

Significance:

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

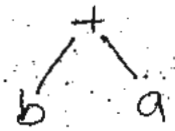
Eg: a+b



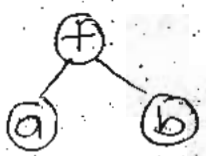
Steps:

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

Eg: a+b



Inorder → Infix = b+a ← Not same
given Infix = a+b ←



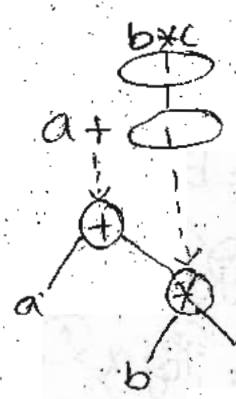
Inorder - Infix = a+b ← same
given = a+b ←

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

Eg:

P) a+b*c precedence:

Sol: a+b*c.



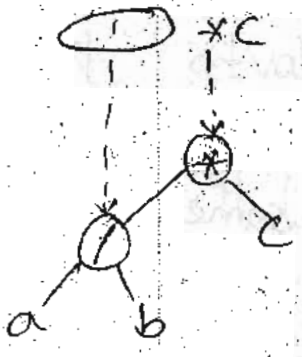
Infix = a+b*c
prefix = +a*bc

p) Left Associative:

a / b * c

a/b

*c

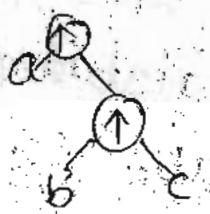


p) Right associative:

a ↑ b ↑ c

b↑c

a↑



Note: Lower precedence operator is at root

p) construct expression trees for

a) −b

prefix = −b

postfix = b −

b) log x

prefix = log x

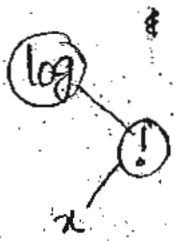
postfix = x log

c) $x!$

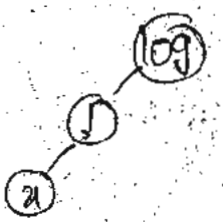
prefix = $!x$

postfix = $x!$

d) $\log x!$



(or)



wrong

Inorder = $x!\ \log$ ← Infix

given Infix = $\log x!$ different

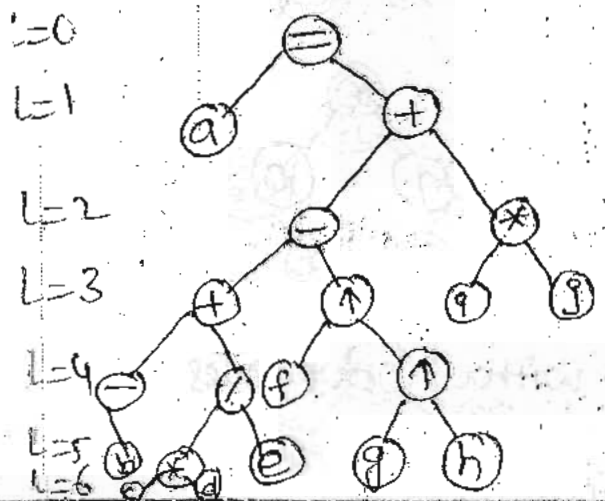
e) consider

$a = -b + c * d / e - f \uparrow g \uparrow h + 9 * 9$

i) what is @ root

= (lower precedence)

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



precedence

-, - unary

↑ : Right associative

×/ : Left associative

+ - : Left associative

∴ Height = 6

(counting edges from leaf)

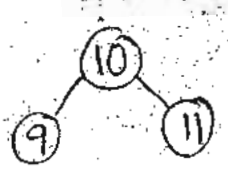
Binary Search Trees: (BST)

- Also called Dictionary trees (or) lexically ordered Binary trees

Definition:

→ Binary tree with

Eg:

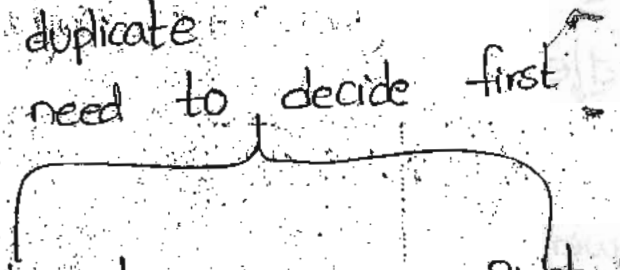


Significance:

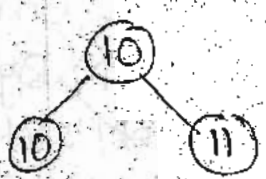
If we traverse inorder, we will get sorted list

→ with duplicate

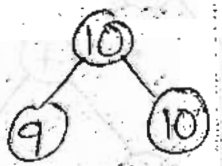
we need to decide first



Eg



Eg



Note:

Default BST are without duplicates

Creation of BST:

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

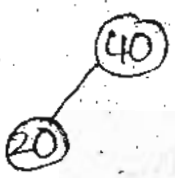
Eg:

create BST with

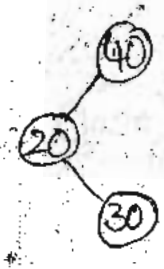
40, 20, 30, 60, 50, 70, 10

→ 40

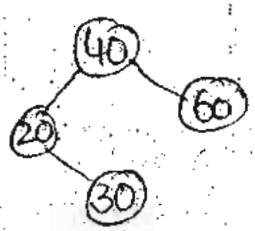
→ 20



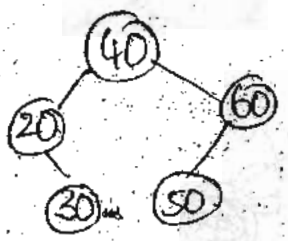
→ 30



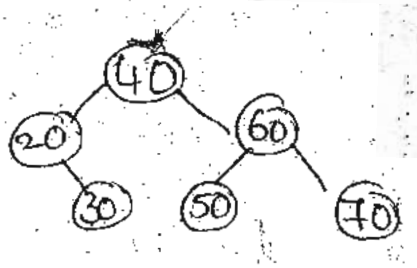
→ 60



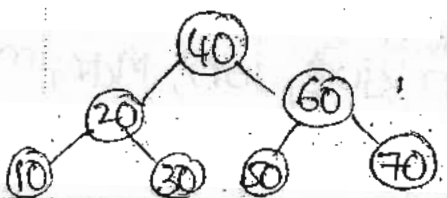
→ 50



→ 70



→ 10



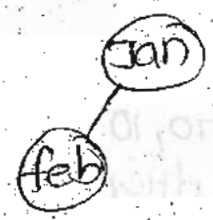
Inorder

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

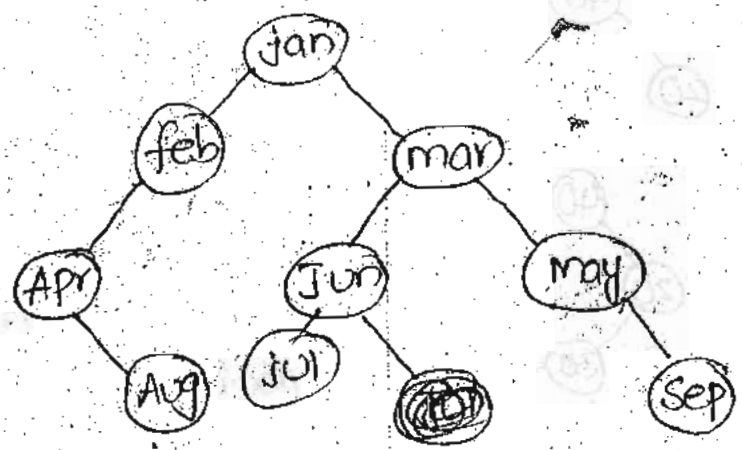
p) Create a BST for months of year (consider first 3 letters of a month)

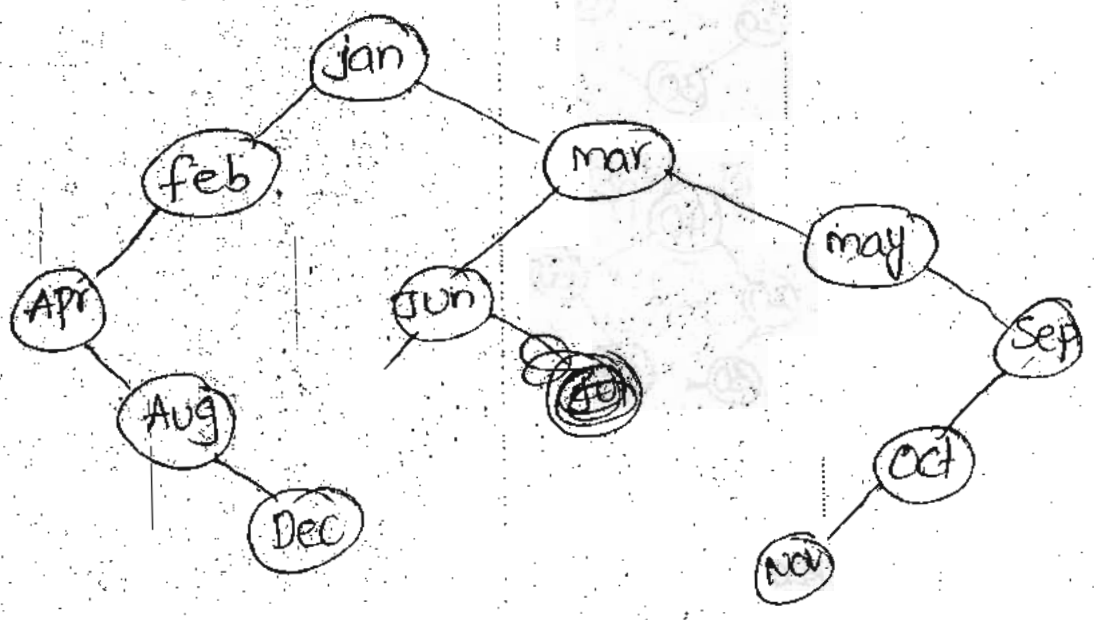
Sol:

JAN



feb





Inorder:

Apr, Aug, Dec, feb, jan, jul, jun, Mar, may, nov, oct, sep