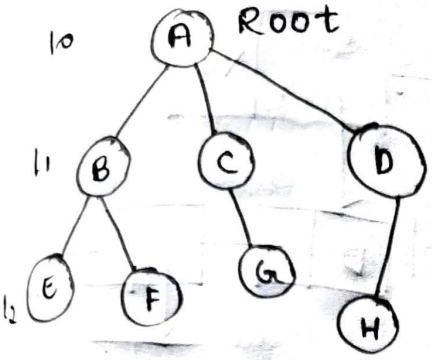
MOD - 4

TREES.



Degree (A) = 3.

Degree (B) = 2.

Degree (C) = 1

Degree (E) = 0.

Leaf Nodes : E, F, G, H.

Subnodes are called as Siblings.

Ancestors of E is A & B.

The no of paths from deepest leaf is height of a tree. ↓

Height : 2

Depth : how deep an element is from the root. ↑.

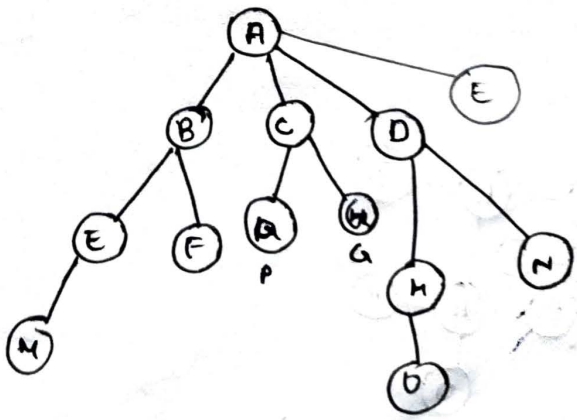
Representation of a Tree:

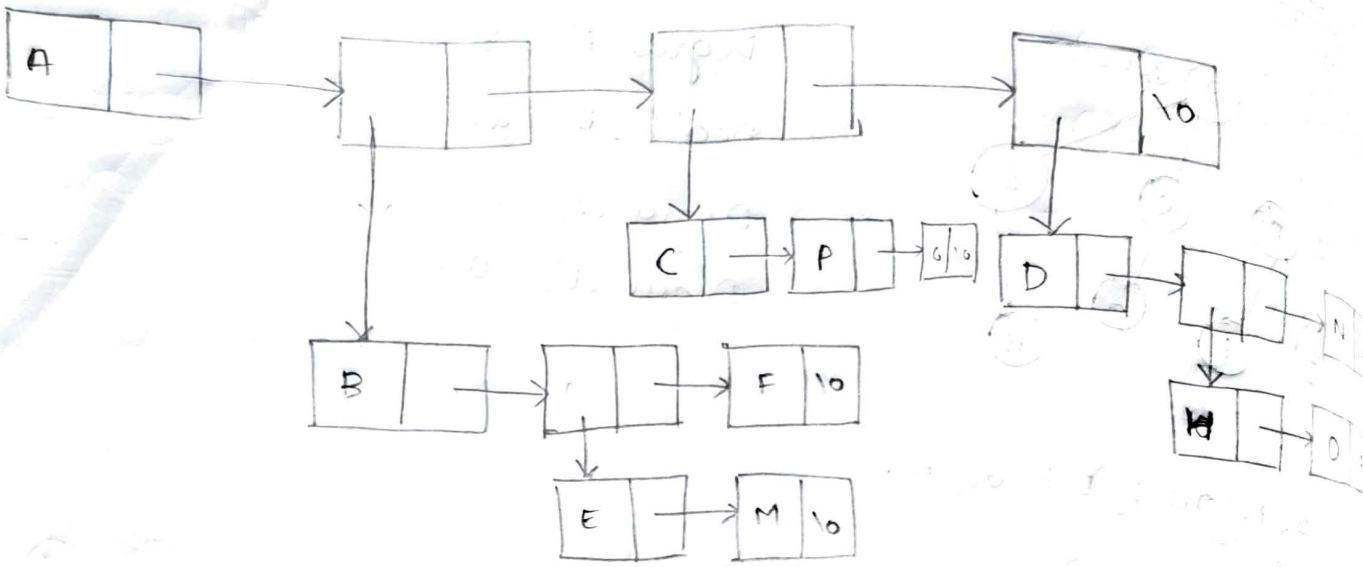
→ List

→ Left child Right sibling

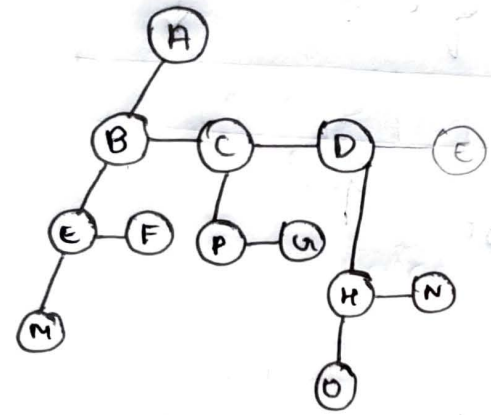
→ Degree - 2.

1) LIST REPRESENTATION:





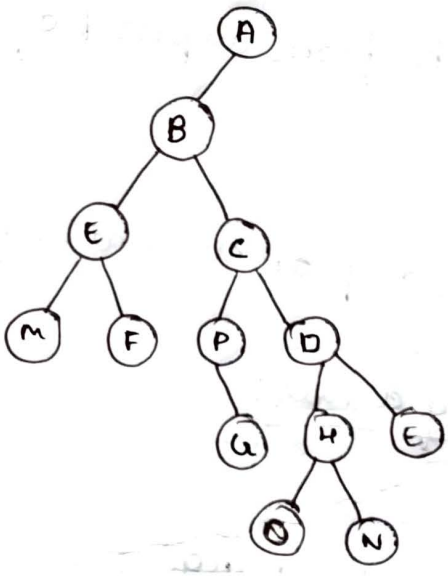
2) LCRS.

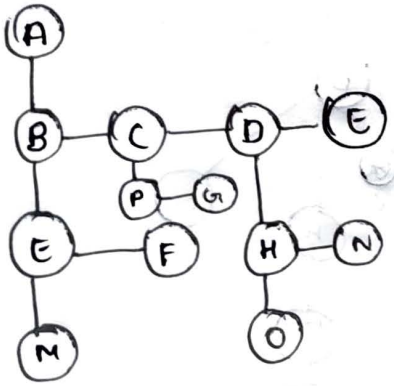


This should be straight

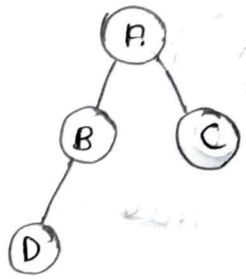
3) Degree - 2 :

(Rotate it clockwise)

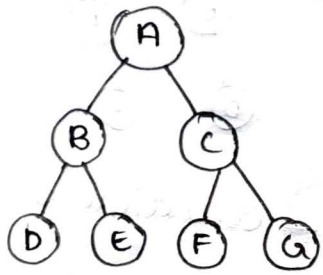




Binary tree:

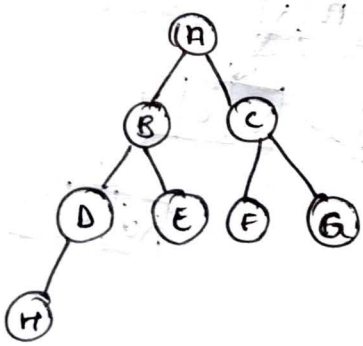


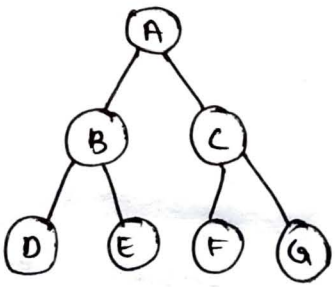
Full Binary Tree: Each node has either 0 or 2 childs.

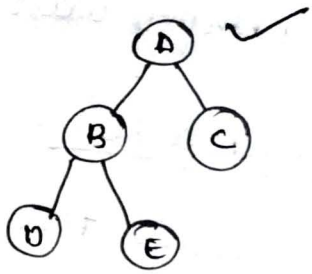


All full can not be complete & vice versa is not possible.

Complete Binary Tree: (Except the last child, it should have 2 childs). And the last leaf node, it should be first leaf i.e it should be on the left most side. (Perfect Binary Tree).



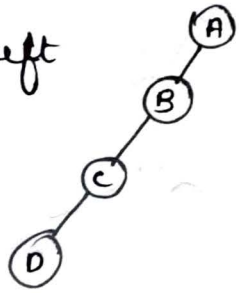




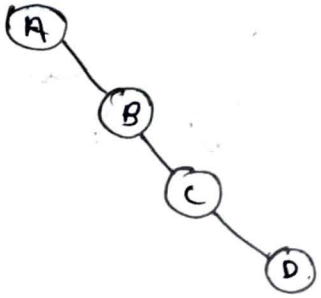
All full Binary tree are CBT but vice versa is not true.

ued Binary Tree:

Left



Right.



Two Representations of Binary Tree

- Array
- Linked List

→ This is not a tree

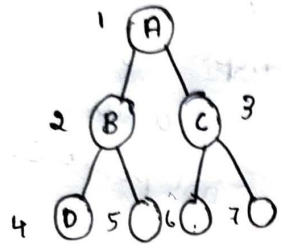
→ This is a cycle

→ This is a forest

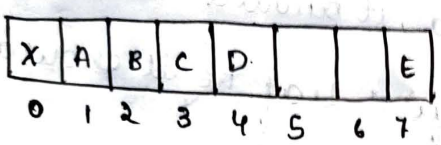
→ Collection of trees is a forest

Array:

Max possible nodes: $2^n - 1$.



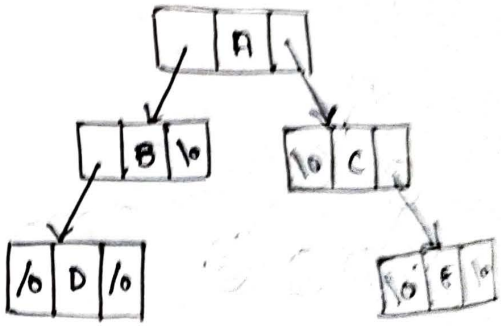
(Blanks are not represented by anything)



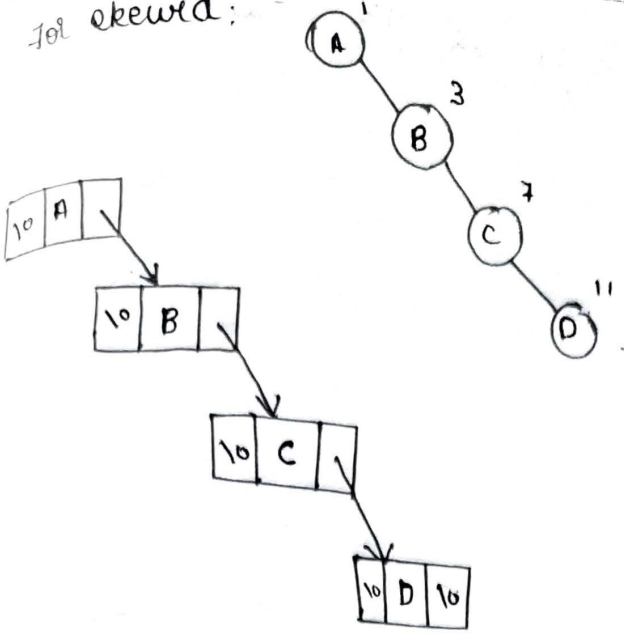
→ Memory wastage, therefore not efficient.

Linked List:

```
struct tree
{
 // Data char c;
 struct Tree *left, *right;
};
typedef struct tree Node;
```



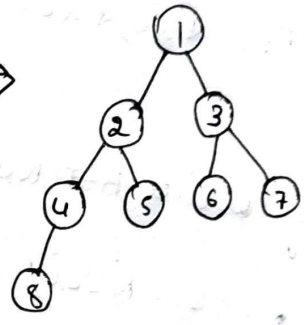
for skewed:



Traversals of a Tree: Imp:

Construction of a tree:

Representation of 1,2,3,4,5,6,7,8 →

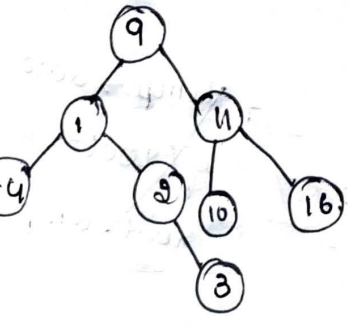


TRAVERSAL:

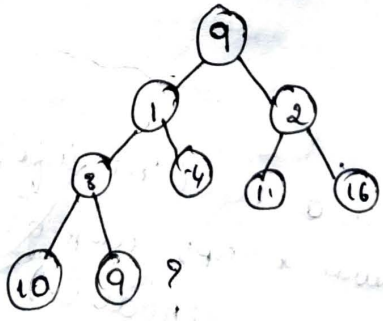
→ Pre Order
→ Post order
→ In-order.

} Lab programs.

Binary search tree: 9 1 2 3 -4 11 16 10 9 ignored.

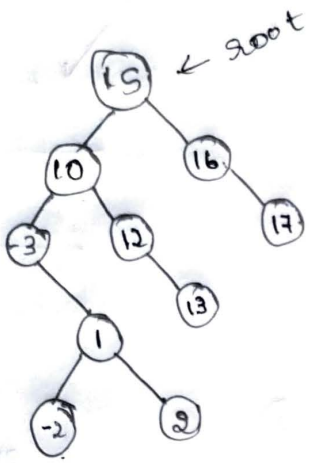


BST



BT

15 10 16 -3 1 2 17 12 13 -2 -3




```
struct BST
{
  int data;
  struct BST *lc, *rc;
};
typedef struct BST Node;
Node *root = NULL;
```

case 1: //read N

```
for(i=0; i<n; i++)
{
  scanf("%d", &info);
  create(info);
}
```

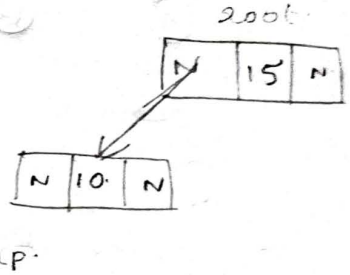
} create (in main).

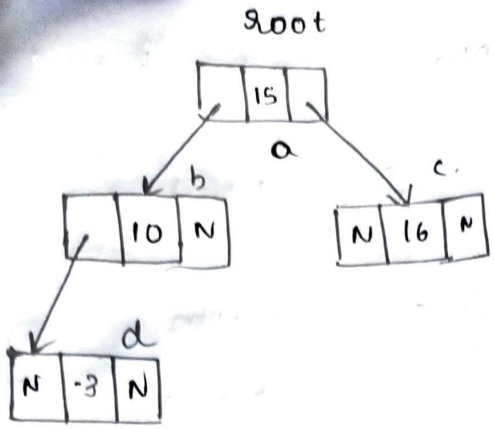
```
void createtree(int info)
{ Node *temp; temp=getnode();
Node *getnode()
{
  Node *ptr;
```

temp → data = info
if(root == NULL)
root = temp

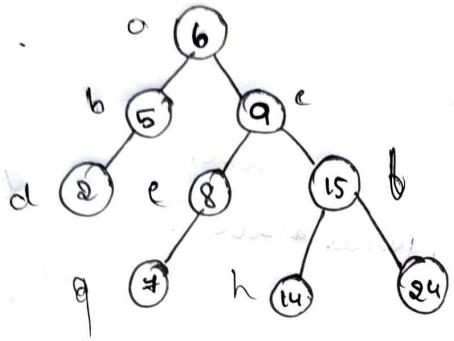
```
ptr = (Node *) malloc (sizeof (Node));
if (ptr == NULL)
{
 exit(0);
 }
ptr -> lc = NULL;
ptr -> rc = NULL;
return ptr;
}
```

```
void create (int info, Node *troot)
{
 Node *temp;
 temp = getnode();
 temp -> data = info;
 if (troot == NULL)
troot = temp;
else if (troot -> data > temp -> data)
  create (info, troot -> lc)
else if (troot -> data < tmp -> data)
  create (info, troot -> rc);
else
  printf ("%d is a duplicate element", info);
}
```





Inorder :



2 5 6 7 8 9 14 15 24

LNR.

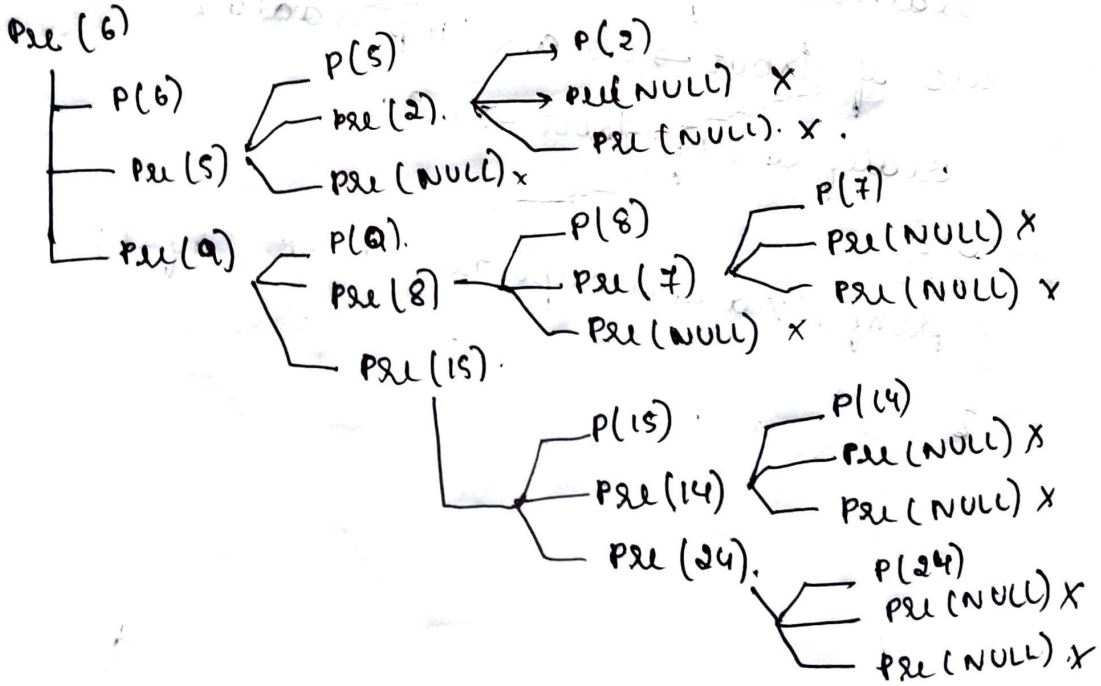
↳ check if travel until NULL.

come one step back

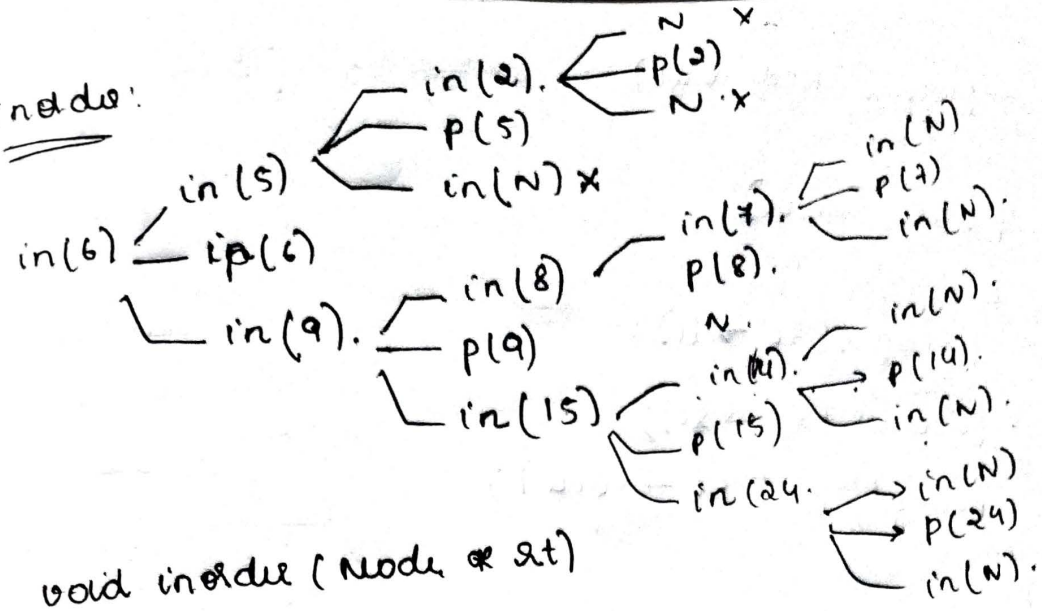
Preorder:

6 5 2 9 8 7 15 14 24.

NLR.



Inorder:




```
void inorder (Node *rt)
{
    if(rt != NULL)
    {
        inorder (rt->lc);
        printf ("%d", rt->data);
        inorder (rt->rc);
    }
}
```

preorder:

```
void preorder (Node *rt)
{
    if (rt != NULL)
    { printf ("%d", rt->data);
      preorder (rt->lc);
      preorder (rt->rc);
    }
}
```

```
void postfix (Node *rt)
{
  if (rt != NULL)
  {
    postorder (rt -> lc);
    postorder (rt -> rc);
    printf ("%d", rt -> data);
  }
}
```

```
struct BST
{
  int data;
  struct BST *lc, *rc;
};

main()
{
  Node *root = NULL;
c1: for (i=0 to N-1)  → read.
    Node * CreateTree (root, info);  → read.
c2: inorder (root);
c3: preorder (root);
c4: postorder (root);
c5: search (root, key);
}
```

without getno

```
Node * CreateTree (Node * root, int v)
{
 Node * temp ;
 temp = (Node *) malloc (sizeof (Node))
 temp → LC = NULL;
 temp → RC = NULL;
 temp → data = v;
 if (root == NULL)
 {
 root = temp;
 } return root;
 }
  if (v < root → data)
  {
  root → lc = CreateTree (root → lc, v)
  }
  else if (v > root → data)
   {
   root → Re = CreateTree (root → rc, v);
   }
  else
   {
   return root;
   }
  }

void inorder (Node * root)
```

Search (inorder)

```
void s(Node *root, int key)
{
  if(root)
  {
  if(root -> data == key)
    { p.
      return;
    }
  s(root -> lc, key);
  s(root -> rc, key);

void search(Node *root, int key)
 {
   if(!root)
      printf("KNF");
   else if (key < root -> data)
      search(root -> lc, key);
   else if(key > root -> data)
      search(root -> rc, key);
   else
      printf("KF");
 }.
```

main()
s(root,14

inorder / stack:

```
void sit ( Node * root)
{
  Node ar[MS];
  for( ; ; )   (or while(1))
  {
   for( ; root; root = root -> left)
    {
      push (root);
    }
    root = pop();
    if(!root)
     break;
     printf("%d", root -> data);
     root = root -> rc;
    }
  }
```

OR.

```
while (root)
{
  push(root)
  root = root -> left;
}.
```

16

I
root = 2
print 2.
root = N

II
root = 5
print 5
root = N

III
root = 6
print 6.
root = 9

IV
push 9, 8, 7

root = 7.
p(7).
root = N.

V
root = 8
print 8
root = N.

VI
root = 9
print 9
9 = 15

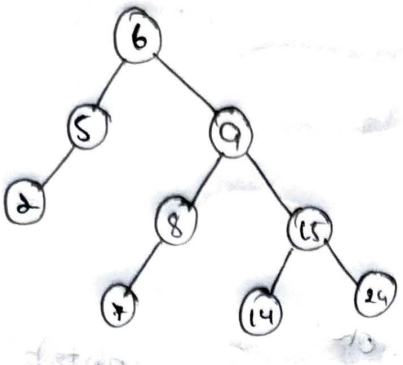
VII
root = 14
print 14
9 = N

VIII
root = 15
print 15
9 = 24

IX: root, 24.
print 24.
root = NULL

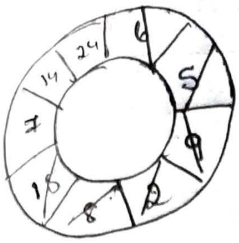
X
breaks the outer for loop.

Level Order Traversing :



6 5 9 2 8 15 7 14 24


```
void CQLOT ( Node *root )
{
  Node cq[MS];
  insert (root);
  for( ; ; )
  {
   if (root)
   root = delete ();
   if (root)
   { printf ("%d", root -> data);
   if (root -> lc)
     insert (root -> lc);
   if (root -> rc)
     insert (root -> rc);
   }
  else
     break;
  }
}
```



root = 6.

print 6.

root = 5

print 5.

6 5 9 2 8

copy Tree!

```
Node * copy ( Node * ri)
{
  Node * temp = NULL;
  if(ri)
  { temp = (node *) malloc ( sizeof (Node));
    temp → lc = copy (ri → lc) ;
    temp → rc = copy (ri → rc)
    temp → data = ri → data;
  }
  else
    { return temp;
    }
  }
```

Compare Tree:

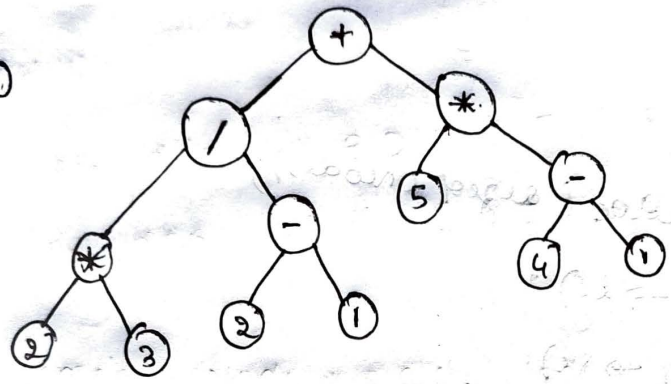
```
int match ( Node * r1 , Node * r2).
  {
    if ( ( !(r1) && !(r2)) || ( (r1 && r2) && (r1 → data
                                    == r2 → data
&& ( r1 & match(r1 → lc , r2 → lc)) && (match(r1 → rc,
```

```
    &a->a(1)).
    return 1;
  else
    return 0;
}.
```

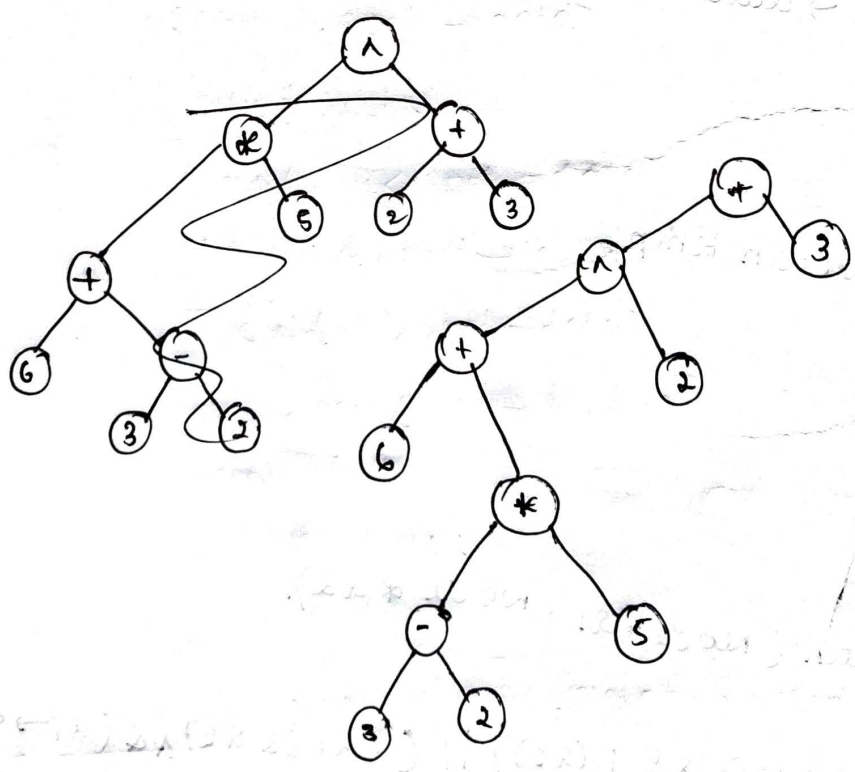
1) 2*3/(2-1)+5*(4-1)

2) ((6+(3-2)*5)^2+3)

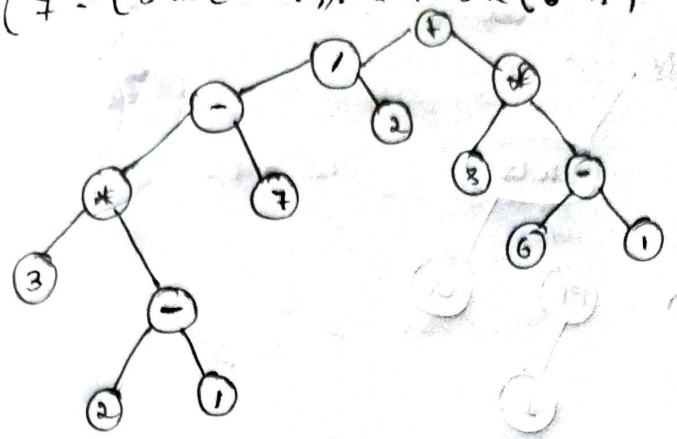
①



②



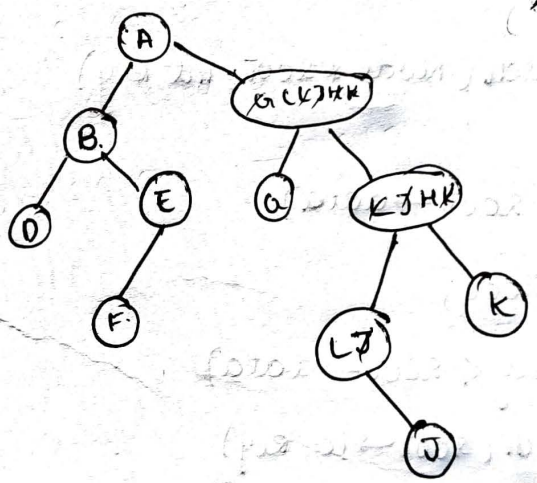
(3) ((7 - (3 * (2 - 1))) / 2 + 3 * (6 - 1))



* Pre order = A B D E F C G H L J K.

In-order = D B F E A G C L J H K

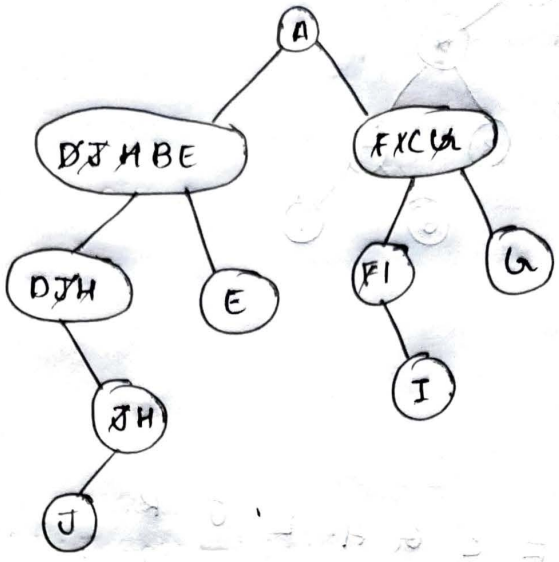
A B D E F C G H L J K



← see from this side

* Post order : J H D E B I F G C A.

In-order. D J H B E A F I C G.



* H.W:

Pre - order: A B D H J E C F I G

Post order: J H D E B I F G C A

search: Node * search (Node *root, int key)

```
{
 if (key == root → data)
   return root;
  else if (key < root → data)
root = search (root → lc, key)
  else if (key > root → data)
root = search (root → rc, key)
  else
    return NULL;
}
```

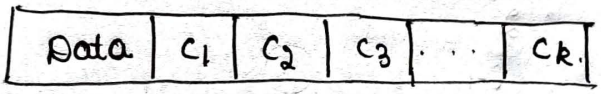
without recursion: Node *searchIteration (Node * root, int key).

```
{
  while (root)
  {
```

```
if (key == root->data)
    return root;
else if (key < root->data)
    root = root->lc;
else
    root = root->rc;
}
return NULL;
}
```

on General Tree

Theorem: If t is K-ary tree (a tree with degree k) with 'n' nodes each having a fixed size as shown in the figure:

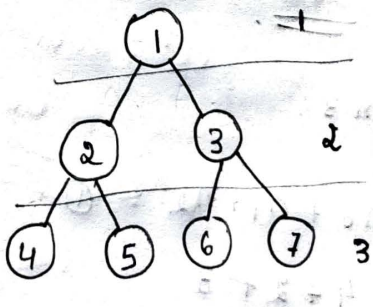


then $n(k-1)+1$ of nk child fields are zero where $n >= 1$

From textbook you write:

Properties of BT.

1: The max no of nodes on level i of a binary tree is 2^{i-1} where $i \geq 1$.



$2^{1-1} = 2^0 = 1$

$2^{2-1} = 2^1 = 2$.

2: The max no of nodes in a binary tree of depth k is $2^k - 1$.

$n_1 + n_2 + n_3 = 1 + 2 + 4 = 7 = 2^3 - 1$

$= 7$

3. The relation between no of leaf nodes with respect to degree to nodes for any non empty binary tree 't' if n_0 represents no of leaf nodes, n_2 , no of nodes of degree 2. The relation is $n_0 = n_2 + 1$.

$4 = 3 + 1$

$4 = 4$

Proof for Theorem 1: Base → root = 1

Hypothesis →

step $(i-1) = 2^{i-2}$

$i = 2 * 2^{i-2}$

$= 2^{i-2+1}$

$= 2^{i-1}$

Proof for Theorem 2:

$2^k - 1$

$\sum_{i=1}^{R} 2^{i-1}$

$= 2^R - 1$

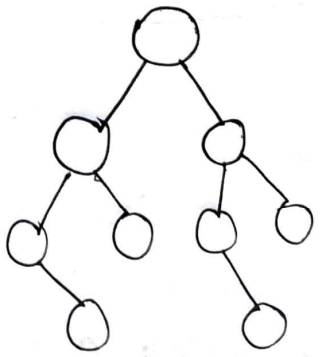
Proof for Theorem 3:

$n = n_0 + n_1 + n_2$ — ①

$= 4 + 2 + 3$

$n = B + 1$ — ②

$B = n_1 + 2n_2$ → ③



(c) $n = n_1 + 2n_2 + 1$

$n = n_0 + n_1 + n_2$

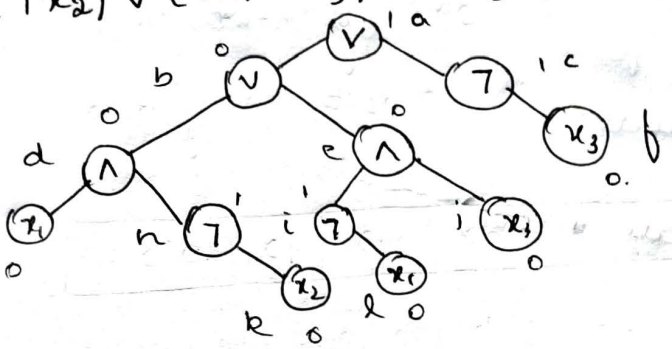
$n = n_1 + 2n_2 + 1$

$n_0 + n_1 + n_2 = n_1 + 2n_2 + 1$

$n_0 = n_2 + 1$

(satisfiability problem).

* $(x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_3) \vee \neg x_3$.



= 1

** Pseudocode:

```
for(all 2^n combinations)
{
  generate the next combination
  replace the value
  do postolder evaluation to get value of the
  if (root -> value)
  { printf("satisfied problem");
    return;
  }
}
```

```
printf("SCNA");
```

~~void postordereval(node *root)~~

```
typedef enum {and, or, not, true, false} logical;
struct BTSP
{
  logical data;
  short int value;
  struct BTSP *lc, *rc;
};
typedef struct BTSP Node;
void postordereval (Node *root)
{
  if (root)
  {
    postordereval (root->lc);
    postordereval (root->rc);
    switch (root->data)
    {
      case 'not' : root->value = !(root->rc->value);
                   break;
      case 'or' : root->value =
                  (root->lc->value || root->rc->value);
      case 'and':
      case 'false' : root->value = 0;
      case 'true' : root->value = 1;
    }
  }
```

Backing:

① $(x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_3) \vee \neg x_3$.

1) 0 0 0

2) x_3 x_2 x_1

3)

poe(a) → poe(b), poe(c), switch(v)

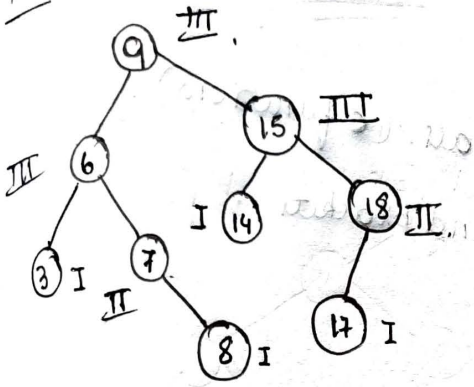
poe(b) → poe(d), poe(e), s(v)

poe(d) → poe(g), poe(h), s(n)

poe(g) → poe(N), poe(N), s(f)

poe(h) → poe(N), poe(b), s(7)

To delete:



case 1 (if leaf node).

d(3)

↳ search(3)

↳ if found

delete node having 3.

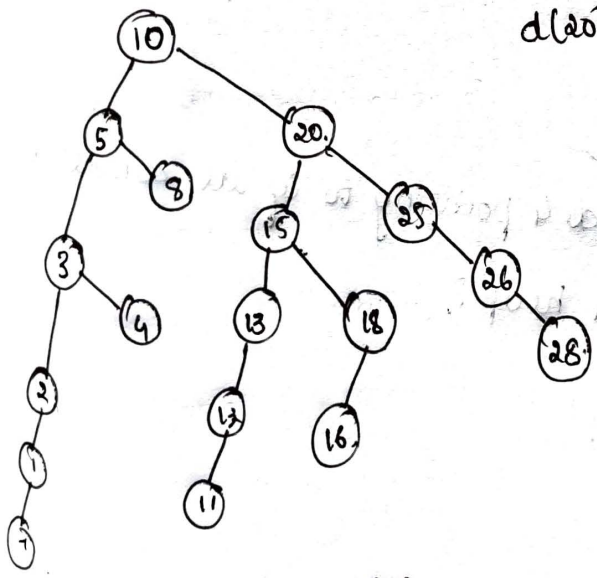
↳ if node.

II (single child node)

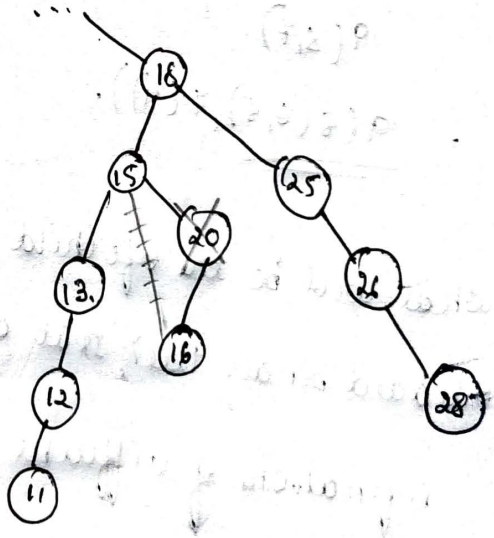
III (two children node)

Inorder:

3 6 7 8 9 14 15 17 18.



d(20).



LNR - inorder

LRN - postorder.

NLR - pre-order.

level order

In → pre

→ post

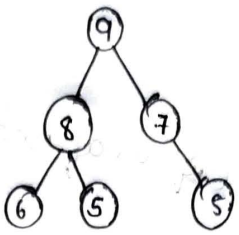
→ level

Imp :

* ((6+(3-2)*5)^2+3).

Pre - ABDEFCGHLJK.

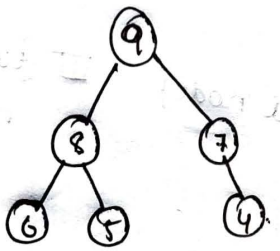
a) Is it a binary tree?



Explaination.

* Write the preorder, inorder, postorder traversal. (c functions).
* Write the list notations using pairs of round brackets.

eg:



9(8,7)

= 9(8(6,5),7(4)).

* What will be the left child of node 4 pointing to if it converted a threaded binary tree? Is it a max heap?
* Definations of diffuent trees: