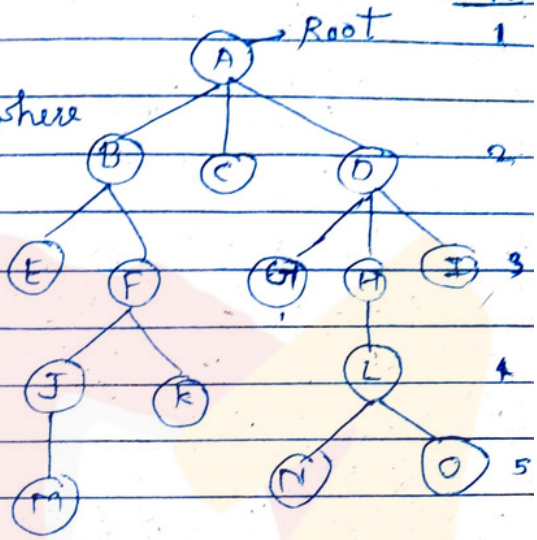
30/05/22

TREE

Non-linear Data structure

Tree - It is a collection of nodes or vertices and edges.

"Tree is a collection of nodes where one of the node is taken as root node and the rest of the nodes are divided into disjoint subset and each subset is a tree or subset."



Terminology

* Root - The very first node on the top is a root.
* Parent - A node is a parent to its very next descendants

  Eg:- "B is a parent of E and F."

  or "E and F are child of B."

* Siblings are children of the same parent.

  Eg:- E and F are sibling to each other.

* Descendents - Descendents are all those set of nodes which can be reach from a particular node or under that node.

  Eg:- For 'B' → E, F, J, K, M all these are descendent of 'B'.

  For 'D' → G, H, I, L, N, O

Date
Page

* Ancestors -

Eg:- 1) For 'M' - J, F, B, A are ancestors.

2) For - 'N' - L, H, D, A are ——

* Degree of nodes - Number of children (direct children)

Eg:- Degree of L is = 2
(Because it has two children)

Degree of D is = 3 (G, H, I)

Degree of N = 0

—— A = 3

* External nodes (leaf nodes) (non-terminal n

Nodes with degree 0 are called as leaf nodes.

Eg:- E, M, K, G, N, O, I are leaf no

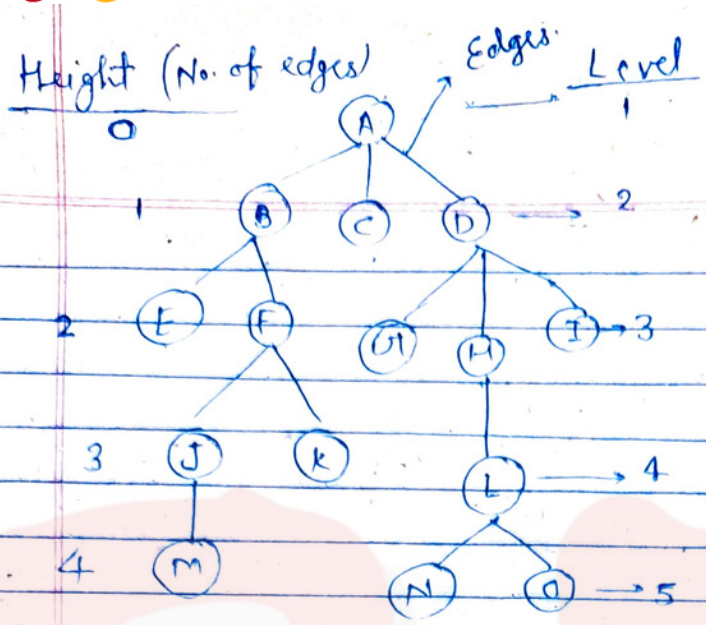
* Internal nodes (non-leaf nodes or non-terminal node

Nodes with those nodes whose degree is greater than 0 are called as non-leaf nodes.

* Levels - It is start from 1 onwards.

Hight - —— 0 " "

* Forest - Collection of trees.



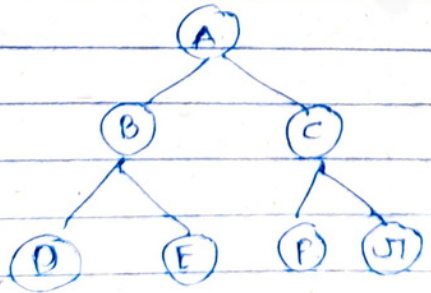
* No. of edges = No. of nodes − 1

* **Binary Tree**

A tree of degree(t) 2 means every node can have max^m two children, not more than two children. It can have less than two children but not more than two.

degree(t) = 2 (Binary tree)

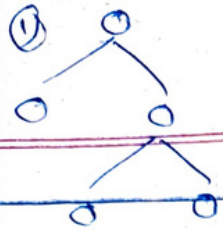
(degree of tree)



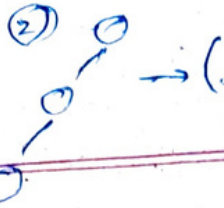
'or', If a tree has only either zero children or one children or at most two children then it is called Binary tree.

children = {0, 1, 2}

Eg:-

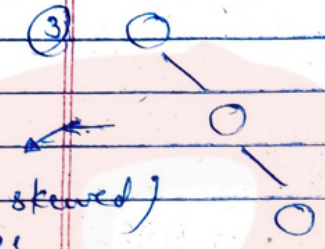


(Binary tree)



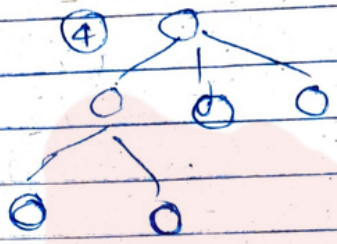
→ (It is also called as left skewed tree)

(Binary tree)



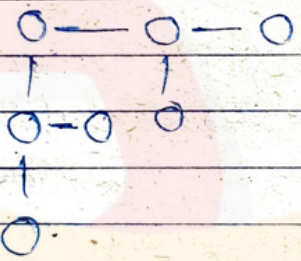
~~Right~~ (Right skewed tree)

(Binary tree)



→ (It has three childs)

(Not Binary tree)



(Binary tree)

No. of Binary tree for a given set of nodes -

Date
Page

$n = 3$

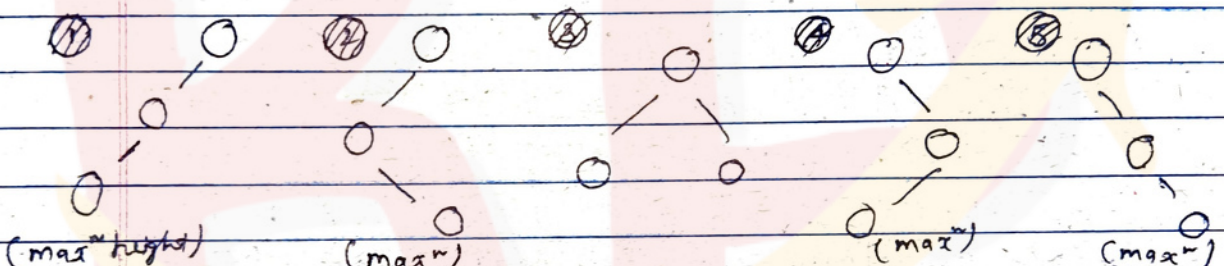
Unlabled nodes — ○ ○ ○ (Three nodes)

Labled nodes — (A) (B) (C)

i) Unlabled Nodes -

○ ○ ○ , $n = 3$

(How many binary trees nodes generated using three nodes)



→ (Five different binary trees are generated using three nodes)

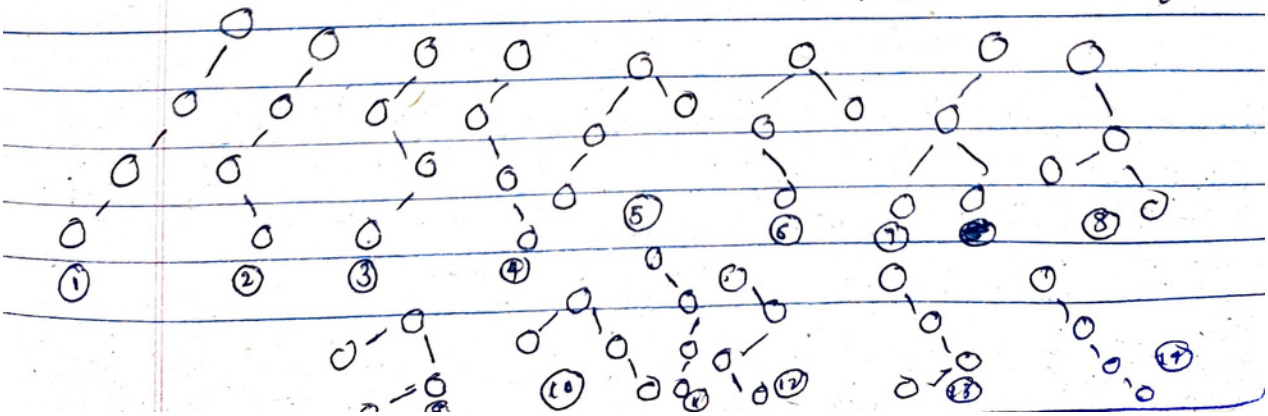
→ No. of tree with max^m height = 4

$T(3) = 5$

→ If $n = 4$

○ ○ ○ ○ then $T(4) = 14$

No. of tree with max^m height = 8



* No. of binary tree for given 'n' nodes = (Unlabelled)

$$T(n) = \frac{^{2n}C_n}{n+1}$$

Catalan Number

n : no. of nodes

If n = 5,

then no. of Binary trees =

$$= \frac{^{2*5}C_5}{5+1} = \frac{^{10}C_5}{6}$$

$$= \frac{10 \times 9 \times 8 \times 7 \times 6}{5 \times 4 \times 3 \times 2 \times 1} = 42$$

$\therefore T(5) = 42$

* No. of binary tree with max^m height for n nodes -

When :

n = 3

then no. of binary tree with max^m height :

$= 4 = 2^2$

Similarly,

→ n = 4

max^m height = $8 = 2^3$

→ n = 5

max^m height = $16 = 2^4$

$\therefore$ for n nodes -

Tree with max^m height = 2^{n-1}

n	0	1	2	3	4	5	6
$T(n) = \frac{2nC_n}{n+1}$	1	1	2	5	14	42	

$T(6) = 1 \times 42 + 1 \times 14 + 2 \times 5 + 5 \times 2 + 14 \times 1 + 42 \times 1 = 132$

$\therefore T(6) = T(0) \times T(5) + T(1) * T(4) + T(2) \times T(3) + T(3) \times T(2) + T(4) \times T(1) + T(5) \times T(0)$

$$\therefore \boxed{T(n) = \sum_{i=1}^{n} T(i-1) * T(n-i)}$$

Recursive formula for finding no. of binary trees.

ii) Labled Nodes -

$n = 3$

(A) (B) (C)

→ No. of binary tree for given 'n' nodes (Labled nodes)

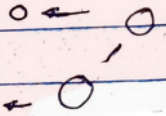
$$\boxed{T(n) = \frac{2nC_n}{n+1} * n!}$$

* Height v/s Nodes -

→ If height of a tree is given then how many min^m or max^m nodes required to reach that height -

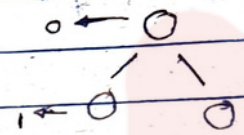
Eg;- 1) If height = 1

min_nodes : 2



n = 2

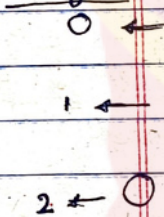
max - nodes = 3



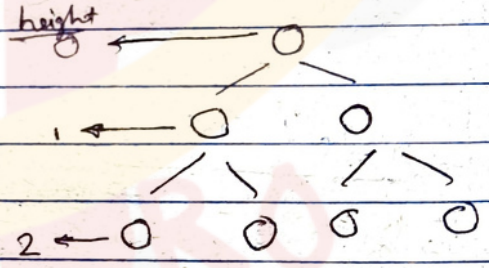
n = 3

2) If height = 2

min-nodes = 3

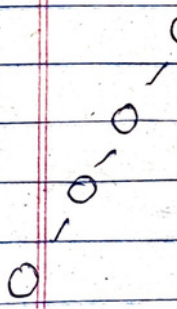


max^m - nodes = 7

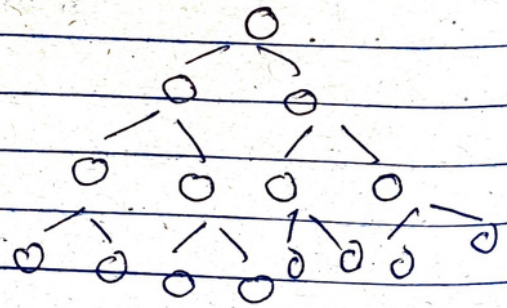


3) If height = 3

min-nodes = 4



max^m - nodes = 15



If height = h

then $\boxed{\text{min}^m \text{ nodes} = h+1}$

$\boxed{\text{max}^m \text{ nodes} = 2^{h+1} - 1}$

This formula is generated by using G.P.

→ If nodes of a tree is given then -

$\text{min}^m \text{ height}(h) = \log_2(n+1) - 1$

$\text{max}^m \text{ height}(h) = n - 1$

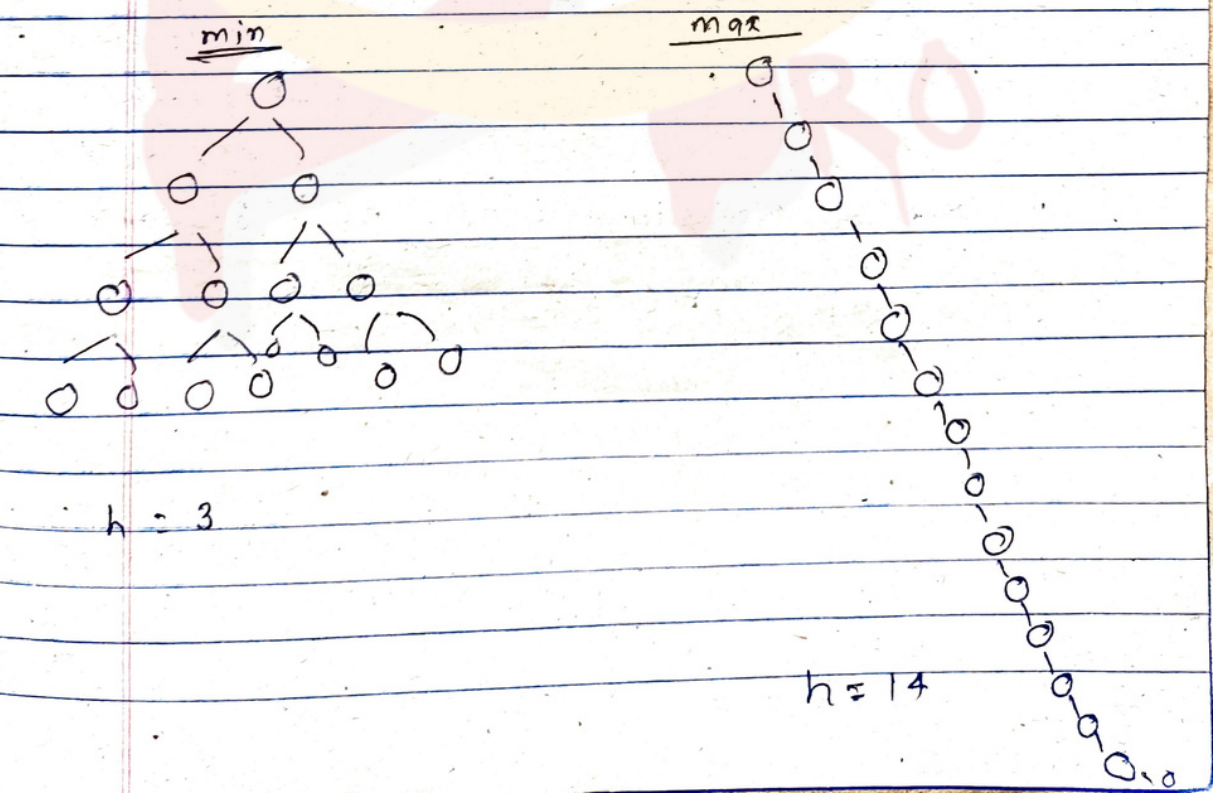
Eg:- $n = 15$

min height $= \log_2(15+1) - 1$

$= \log_2 16 - 1 = 4 - 1 = 3$

max height $= n - 1$

$= 15 - 1 = 14$



Eg- $n = 3$

min^m height = $\log_2(3+1) - 1$

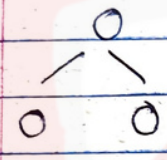
$= \log_2 4 - 1$

$= 2 - 1 = 1$

max^m height = $n - 1$

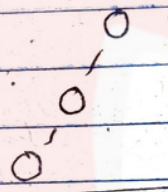
$= 3 - 1 = 2$

min height



$h = 1$

max^m height



$h = 2$

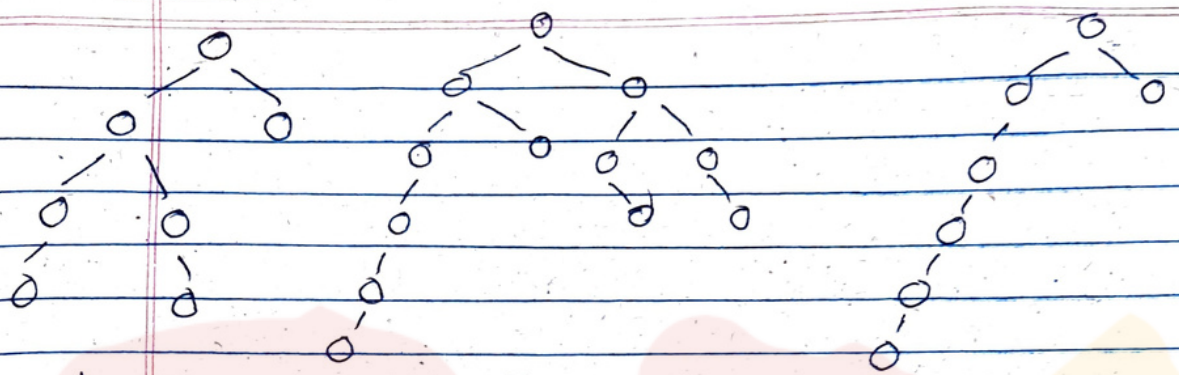
* Height of Binary tree-

$\log_2(n+1) - 1 \leq h \leq n - 1$ (means that height of a tree ranges from $\log n$ to n)

* No. of Nodes in a Binary tree - (Based on height)

$h + 1 \leq n \leq 2^{h+1} - 1$

* Relationship b/w internal node and external node =



deg(2) = 2
deg(1) = 2
deg(0) = 3

deg(2) = 3
deg(1) = 5
deg(0) = 4

deg(2) = 1
deg(1) = 4
deg(0) = 2

$$\therefore \boxed{deg(0) = deg(2) + 1}$$

or, $\boxed{e = i + 1}$

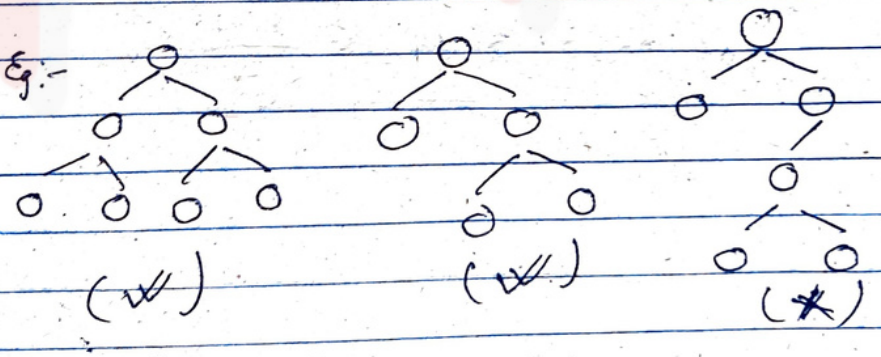
05/05/22

* Strict binary tree (Proper binary trees or complete binary tree)

In general binary tree, node have zero, one or two children but in strict binary tree can have a either a zero children or exactly two children.

children = {0, 2}

Eg:-

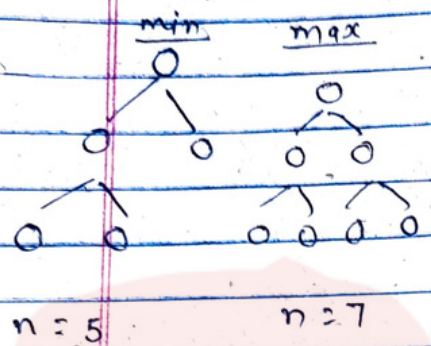


(✗) (✗) (✓)

Height v/s Nodes (in strict binary tree)

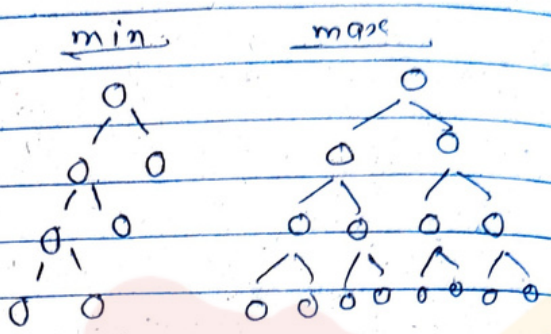
Date
Page

→ If h = 2



n = 5 n = 7

~~If h = 3~~



n = 7 n = 15

* If height 'h' is given -

min nodes $n = 2h+1$

Max^m nodes $n = 2^{h+1} - 1$

* If 'n' nodes are given -

Min^m ~~height~~ $h = \log_2(n+1) - 1$

Max^m height $h = \frac{n-1}{2}$

* Height ranges from -

$$\log_2(n+1) - 1 \leq h \leq \frac{n-1}{2}$$

$\because n = 2^{h+1} - 1$

$\Rightarrow n+1 = 2^{h+1}$

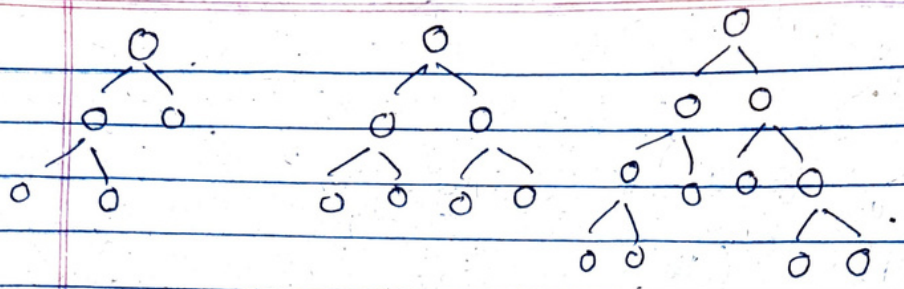
$\Rightarrow \log_2(n+1) = \log_2 2^{h+1}$

$\Rightarrow \log_2(n+1) = h+1$

$\Rightarrow \log(n+1) - 1 = h$

Date
Page

Internal node v/s External node



$i = 2$ | $i = 3$ | $i = 5$

$e = 3$ | $e = 4$ | $e = 6$

$$e = i + 1$$

* m-ary / n-ary trees -

Here m/n is degree of the tree.

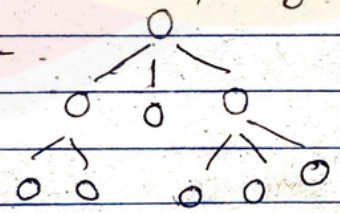
Eg:- 3-ary tree (A tree with degree 3)

$= \{0, 1, 2, 3\}$

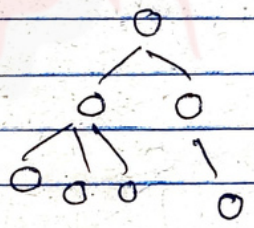
→ In this tree, every node have the capacity to have three children.

If it has less than three children then it is also valid.

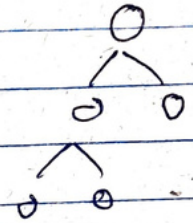
Eg:-



(3-ary tree)



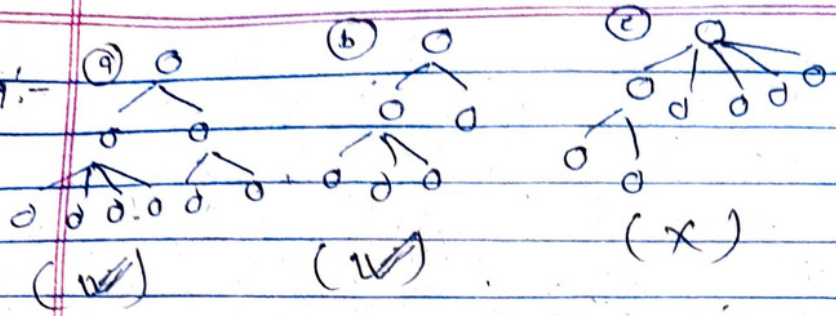
(3-ary tree)



(3-ary tree)

Eg:- i) 4-ary tree - {0, 1, 2, 3, 4}

Eg:-



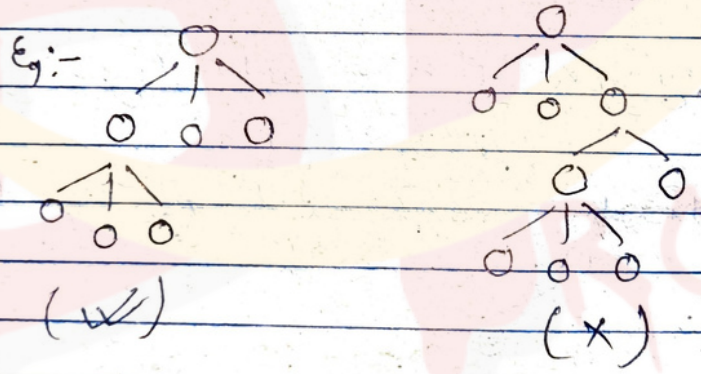
(✓) (✓) (✗)

* Strict m-ary trees -

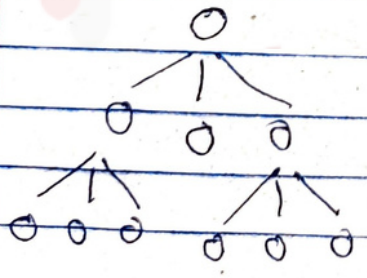
In this tree, every node can have either a zero children or exactly m-children

Eg:- Strict 3-ary tree - {0, 3}

Eg:-



(✓) (✗)



(✓)

Date
Page

n = no. of nodes
Here m = degree of the tree
h = height

In strict m-ary tree -

→ If height is given

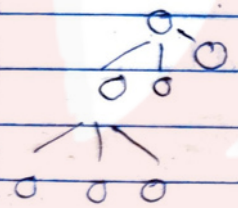
Min nodes $n = mh + 1$

Max nodes $n = \frac{m^{h+1} - 1}{m - 1}$

Eg :- In strict 3-ary tree -

$h = 2$

min

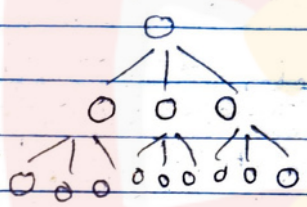


$n = 7$

or $n = mh + 1$

$= 3 \times 2 + 1 = 7$

max



$n = 13$

or $n = \frac{m^{h+1} - 1}{m-1}$

$= \frac{3^{2+1} - 1}{3-1} = \frac{27-1}{2}$

$= \frac{26}{2} = 13$

→ If 'n' nodes are given -

Min height $h = \log_m [n(m-1) + 1] - 1$

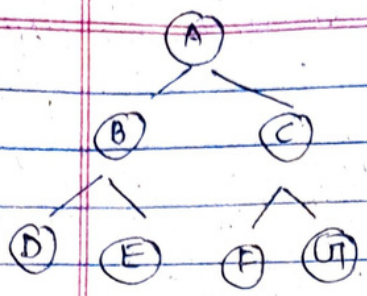
Max height $h = \frac{n+1}{m}$

* Relation b/w internal & external node -

$$\boxed{e = (m-1)i + 1}$$

* Representation of Binary tree

Date
Page

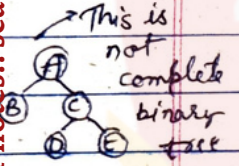


The given binary tree for array representation must be a complete binary tree

i) Array Representation :- (Just fill the nodes level by level)

T | A | B | C | D | E | F | G |

element	index	Left child	R-child
A	1	2	3
B	2	4	5
C	3	6	7
	i	$2*i$	$2*i+1$



T | A | B | C | D | E | X

T | A | B | C | - | - | D | E | W

This is array representation of that tree which is not complete binary tree.

→ If an any element is at index 'i' then

→ element = i then

L-child = $2*i$

R- " = $2*i+1$

Parent = $\left\lfloor \frac{i}{2} \right\rfloor$ (floor value)

$\left\lfloor \frac{3}{2} \right\rfloor$ (floor value) $\Rightarrow$

$= \lfloor 1.5 \rfloor = 1\frac{1}{2}$

$\left\lceil \frac{3}{2} \right\rceil$ (ceil value)

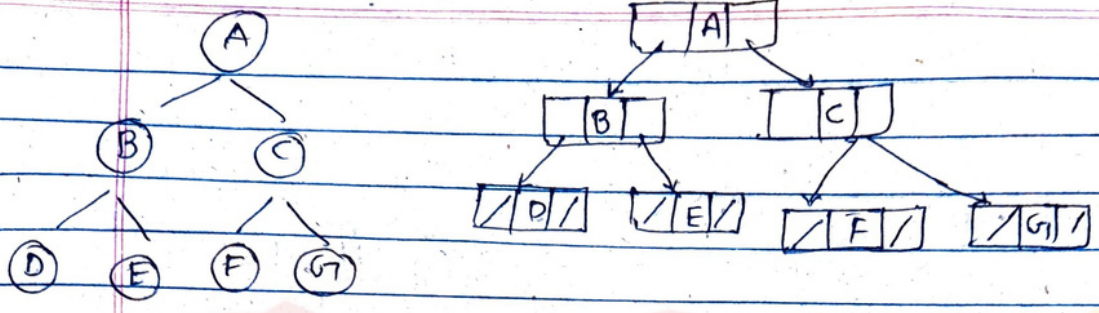
$= \lceil 1.5 \rceil$

$= 2$

(When we fill the element in the array then this formula automatically followed.)

* Linked Representation of Binary tree

Date
Page

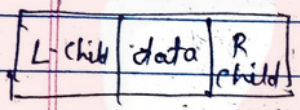


Here n = 7

then NULL = n+1 Pointers

(In any linked binary tree, if there are n- nodes then no. of null pointer is always 'n+1')

Node




```
struct Node
{  struct Node *lchild;
   int data;
   struct Node *rchild;
}
```

07/05/22

Full and complete binary tree

* Full binary tree -

A binary tree of height 'h', having max^m no. of nodes.

Eg:- If a tree has height '2' then to called as full binary tree it has 7 (max^m) nodes

If the height of a tree is 2 then it must have 7 nodes (max^m) for it to be called as a full binary tree.

→ A tree of height 'h' can have $2^{h+1} - 1$ nodes.

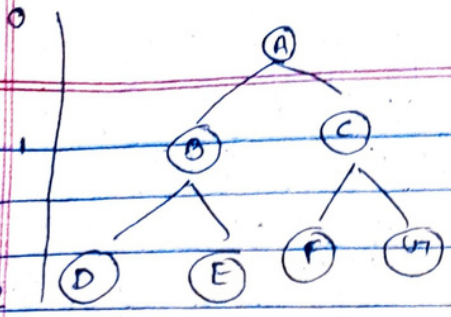
Here h = 2

max^m nodes -

$n = 2^{h+1} - 1$

$= 2^{2+1} - 1 = 8 - 1 = 7$

(So, it is a full binary tree)

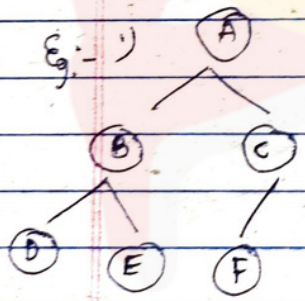


* Complete binary tree -

If a binary tree is represented in an array, then they should not be any blank space in b/w the element.

→ If there are blank space, it's not a complete binary tree.

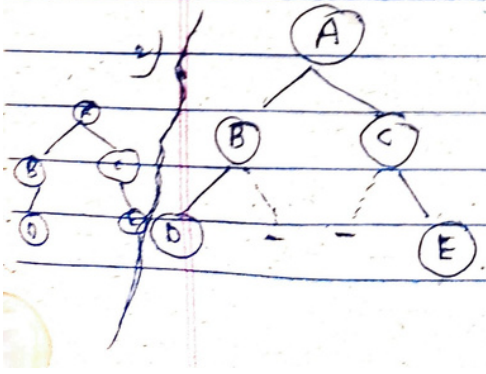
Eg:- 1)



T	A	B	C	D	E	F		

→ Complete binary tree

2)



T	A	B	C	D	-	-	E

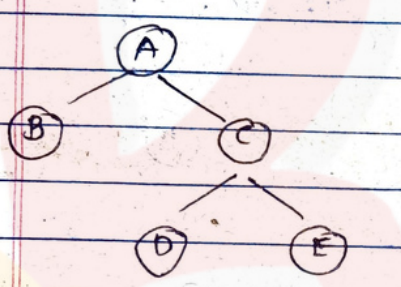
→ Not complete binary tree

Book definition of complete binary tree -

A complete binary tree of height 'h' will be a full binary tree up to a 'h-1' height and at the last level, the element will be filled from left to right without skipping any elements.

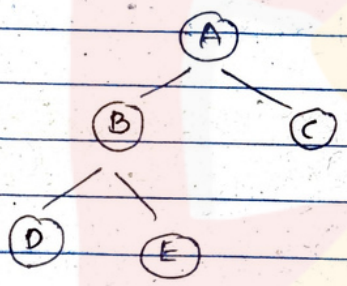
* A full binary tree is always a complete binary tree but for a complete binary tree it is not necessary.

* Strict v/s complete binary tree -



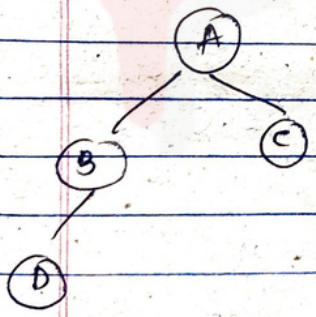
T | A | B | C | - | - | D | E |

strict - ✓
complete - X



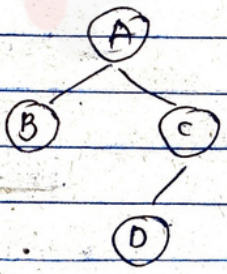
T | A | B | C | D | E | | |

strict - ✓
complete - ✓



T | A | B | C | D | | |

strict - X
complete - ✓



T | A | B | C | - | D |

strict - X
complete - X

Tree traversal

Visiting all the nodes:

* In binary tree, these are following methods are used to traversal -

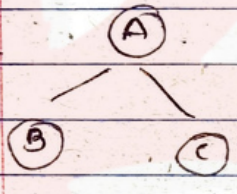
i) Preorder : visite(node), Preorder (left subtree), Preorder (Rightsubtree)

ii) Inorder : Inorder (left), visit (node), Inorder (right)

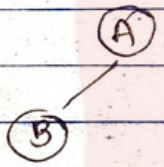
iii) Postorder : Postorder (left), Postorder (right), visit (node)

iv) level order : Level by level.

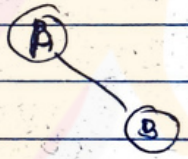
Ex:-



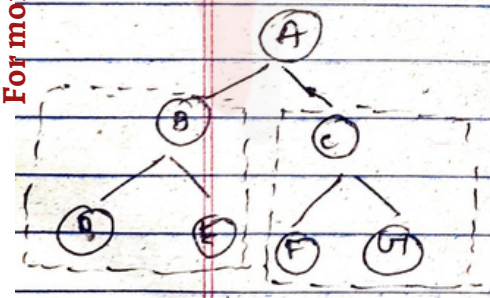
Pre - A, B, C
In - B, A, C
post - B, C, A
Level - A, B, C



Pre - A, B
In - B, A.
Post - B, A
Level - A, B



Pre - A, B
In - A, B
Post - B, A
Level - A, B



→ Level - A, B, C, D, E, F, G

→ Pre - A, ( ), ( )

A, (B, D, E), (C, F, G)

A B D E C F G

→ Inorder - ( ), A, ( )

(D, B, E), A (F, C, G)

D, B, E, A, F, C, G

→ Post - ( ), ( ), A

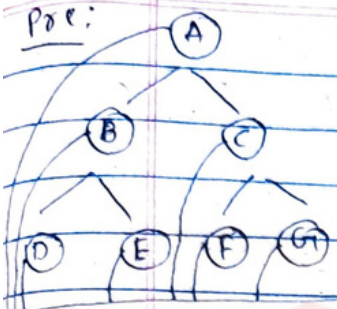
(D, E, B), (F, G, C), A

D E B F G C A

* Other methods for traversal in binary tree

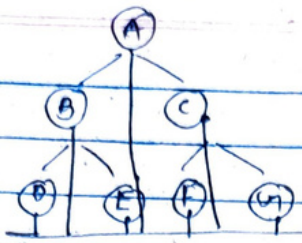
Method - 1

Pre:



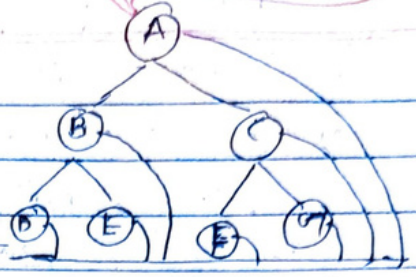
A, B, D, E, C, F, G

In order



D, B, E, A, F, C, G

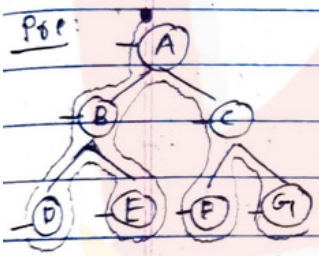
Post



D, E, B, F, G, C, A

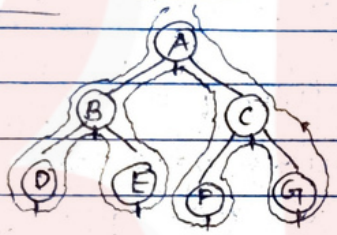
Method - 02

Pre:

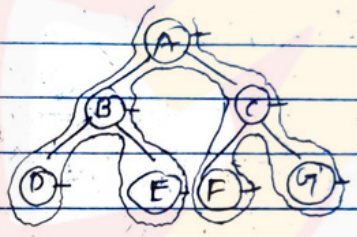


A, B, D, E, C, F, G

Post order



D, B, E, A, F, C, G



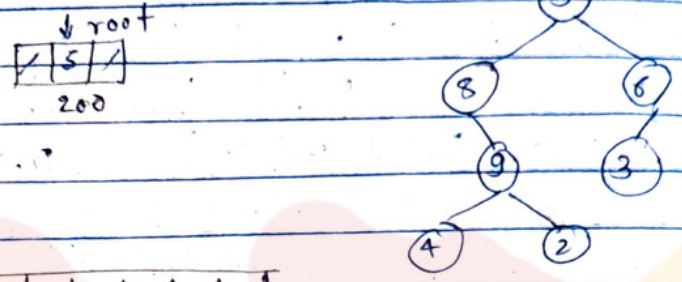
D, E, B, F, G, C, A

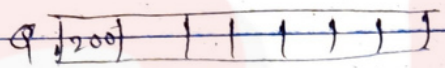
* 1st element of preorder and last element of Post order is always same.

12/06/22

* Creating binary tree -

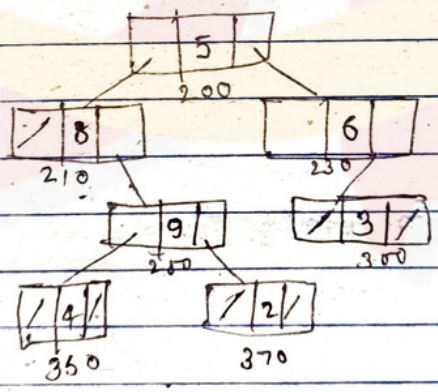
1) Take a queue
2) Creat a root node
3) Insert the address of root in the queue

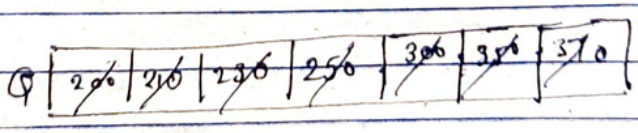




After that

We took the address from queue and make pointer on that previous node and asked there left child or not, if yes, create it and insert the address of it in queue, similar for right tree.





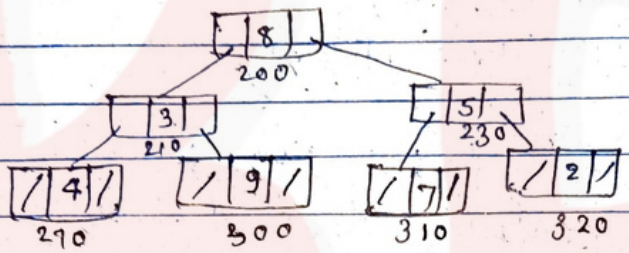
When queue became empty means tree is generated.

```
Void create()
{ Node *p, *t;
  Queue q;
  int x;
  printf("Enter root value");
  scanf("%d", &x);
  root = malloc(...);
  root->data = x;
  root->lchild = root->rchild = 0;
  enqueue(root);
  while(!isEmpty(q))
    {
      p = dequeue(&q);
      printf("Enter left child");
      scanf("%d", &x);
      if(x != -1)
      {  t = malloc(...);
         t->data = x;
         t->lchild = t->rchild = 0;
         p->lchild = t;
      } enqueue(t);
}
```

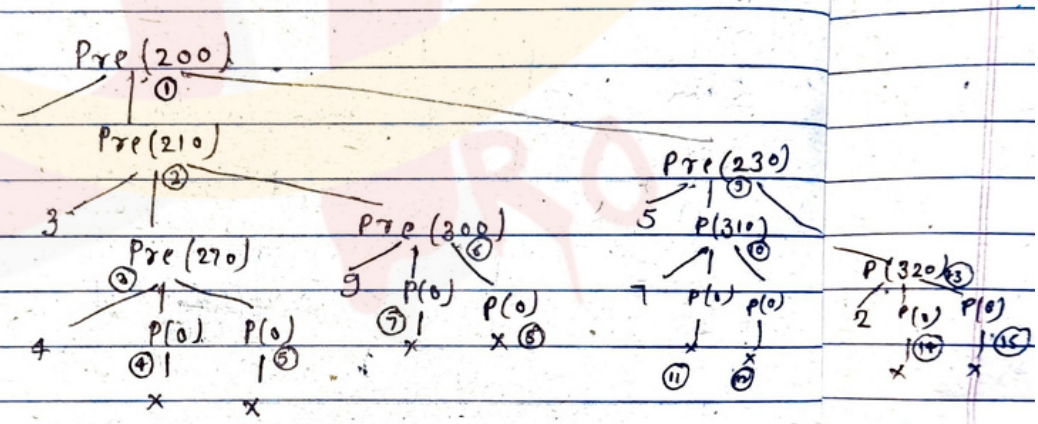
Same for right child

Recursive fn. for preorder traversal -

```
void Preorder(Node *t)
{
    if(t != NULL)
    {
        printf("%d", t->data);
        preorder(t->lchild);
        preorder(t->rchild);
    }
}
```



Tracing



* If there are 'n' nodes then total 'n+1' null pointers will be there.

Eg:- Here n = 7

& Null pointers = 8

Time - O(n)

* Total no. of ~~traversal~~ calls in this traversal -

$n + n + 1 = 2n + 1$

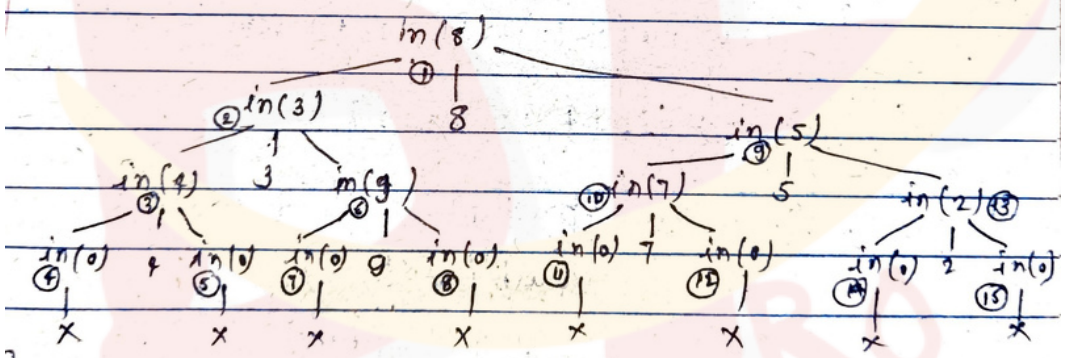
Here n = no. of nodes

* The size of the stack = h + 2

(In this case size of stack is 4 (1+2)

* ~~Recursive function for inorder traversal~~ -

```
void Inorder (Node *t)
{ if (t != NULL)
  { Inorder (t->lchild);
    printf ("%d", t->data);
    Inorder (t->rchild);
  }
}
```



O/P : 4, 3, 9, 8, 7, 5, 2

Total calls = 2n + 1

Size of stack = h + 2

(stack depends upon height of tree);

* Recursive fn. for postorder traversal

```
void Postorder (Node *t)
{ if (t != NULL)
  {
    Postorder(t->lchild);
    Postorder (t->rchild);
    printf ("%d", t->data);
  }
}
```

* Iterative fn. for preorder -

For this we need stack data structure.

```
void Preorder(Node *t)
{ struct stack st;
  while(t!=NULL || !isEmpty(st))
  {
    if(t != NULL)
    { printf("%d", t->data);
      push(&st, t);
      t = t->lchild;
    }
    else
    {
      t = pop(&st);
      t = t->rchild;
    }
  }
}
```

time - O(n)

* Stack depends upon the height of tree.

* Iterative fn. for inorder -

```
void Inorder (Node *t)
{ struct stack st;
while (t != NULL || isEmpty(st))
  {
    if (t != NULL)
    {
      push(&st, t);
      t = t->lchild;
    }
    else
    {
      t = POP (&st);
      printf ("%d", t->data);
      t = t->rchild;
    }
  }
}
```

time - O(n)

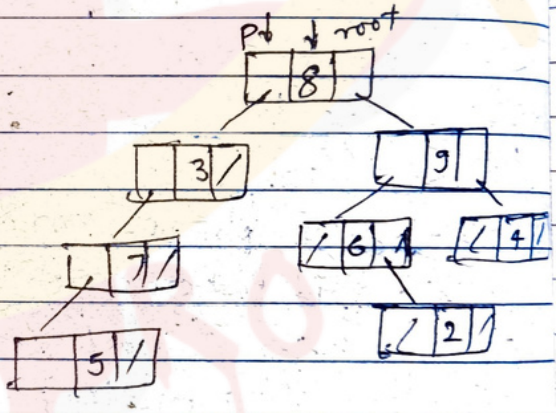
* Iterative fn. for Postorder -

```
void Postorder (Node *t)
{ struct stack st;
  long int temp;
  while(t != NULL || isEmpty(st))
  { if (t != NULL)
    {
      push (&st, t);
      t = t->lchild;
    }
    else
    {
      temp = Pop(&st);
```

```
if (temp > 0)
{ push(&st, --temp);
  t = ((Node *) temp) -> rchild;
else
{
  printf ("%d", ((Node*) temp) -> data);
  t = NULL;
}
}
}
}
```

15/06/22

* Level order Traversal :- Same as creating binary tree




```
void Levelorder(Node *P)
{
  Queue q;
  printf("%d", P->data);
  enqueue(&q, P);
  while (!isEmpty(q))
{
  P = dequeue(&q);
  if (P->lchild)
  { printf("%d", P->lchild->data);
    enqueue(&q, P->lchild);
  }
  if (P->rchild)
  { printf("%d", P->rchild->data);
    enqueue(&q, P->rchild);
  }
```

O/P = 8, 3, 9, 7, 6, 4, 5, 2

Can we generate tree from traversal -

→ If preorder, or postorder or inorder traversal are given then $\frac{2nCn}{n+1}$ tree can be generated.

In this case, we can not create a unique tree, so it is not possible.

→ ① Preorder
② Postorder
③ Inorder
] $\frac{2nCn}{n+1}$ tree

→ If Preorder, Postorder both are given] I.t tree can be generated → It is also not possible

✳ We can create if these traversal are given.

i) Post + Inorder

ii) Pre + Inorder

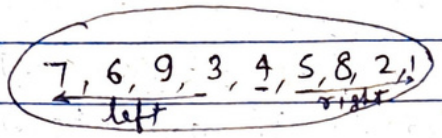
So, if two traversal are given and then in one of them should be inorder and other can be either postorder or preorder. In this case we can create a unique tree.

* Generating tree from traversal -

Preorder - 4, 7, 9, 6, 3, 2, 5, 8, 1

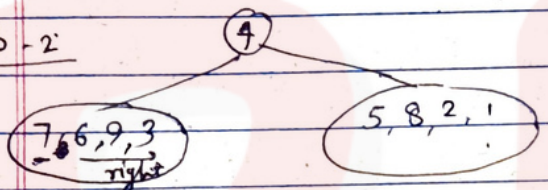
Inorder - 7, 6, 9, 3, 4, 5, 8, 2, 1

step-1



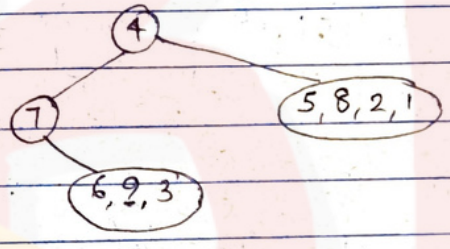
Searching and splitting

step-2

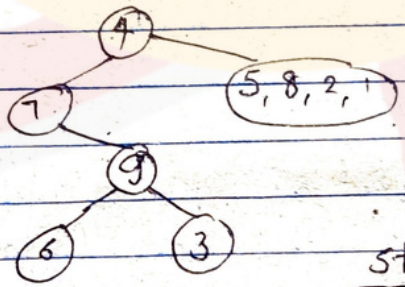


time - $O(n^2)$

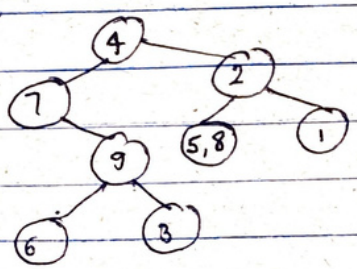
step-3



step-4



step-5



step-6