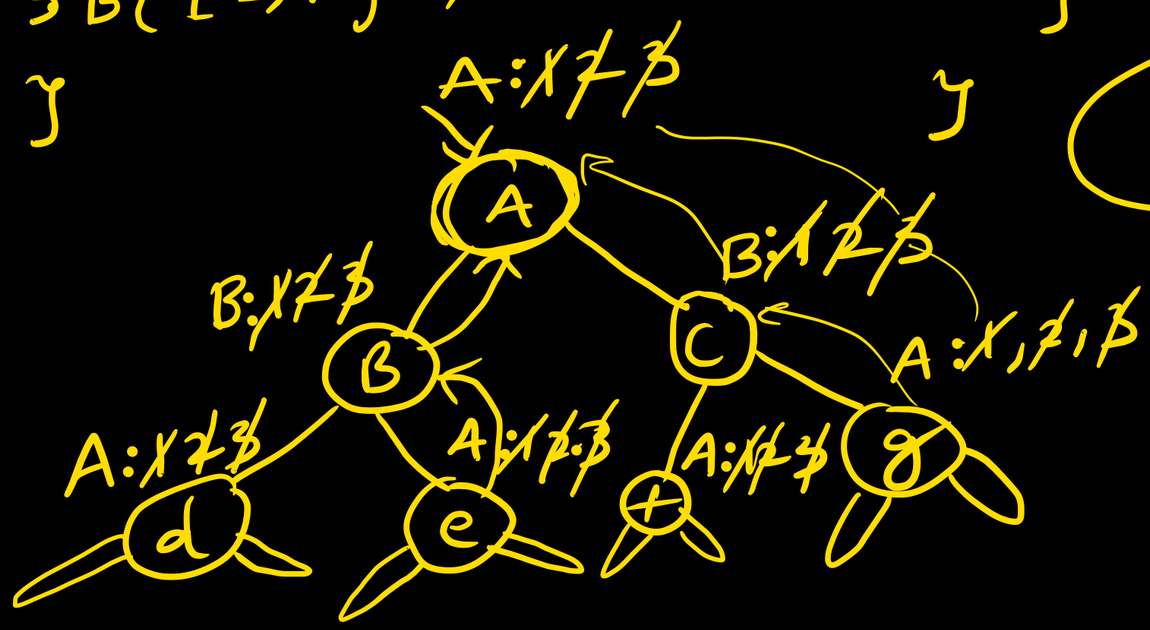
```
Void A( Struct node *t)
{   if (t)
    {
      1. Pf ("%c", t->data)
      2. B (t->left)
      3. B (t->Right)
    }
}
```

Similar to preorder.

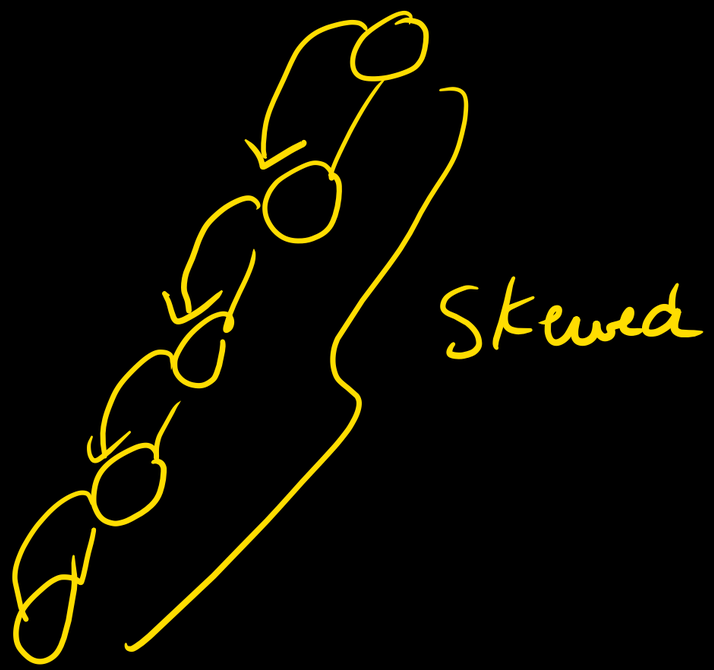
```
Void B(Struct node *t)
{   if (t)
    {
      1. A (t->left)
      2. Pf ("%c", data)
      3. A (t->right)
    }
}
```

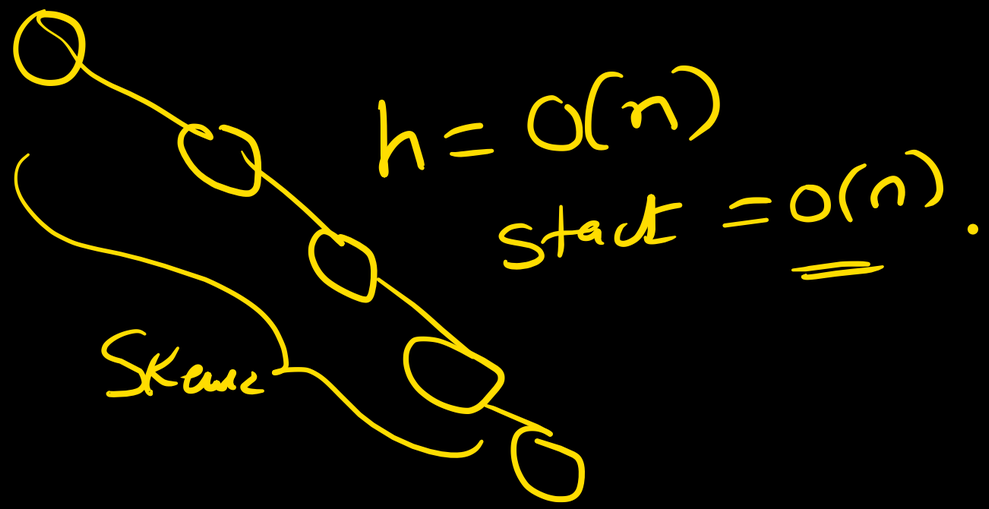
Similar to inorder.

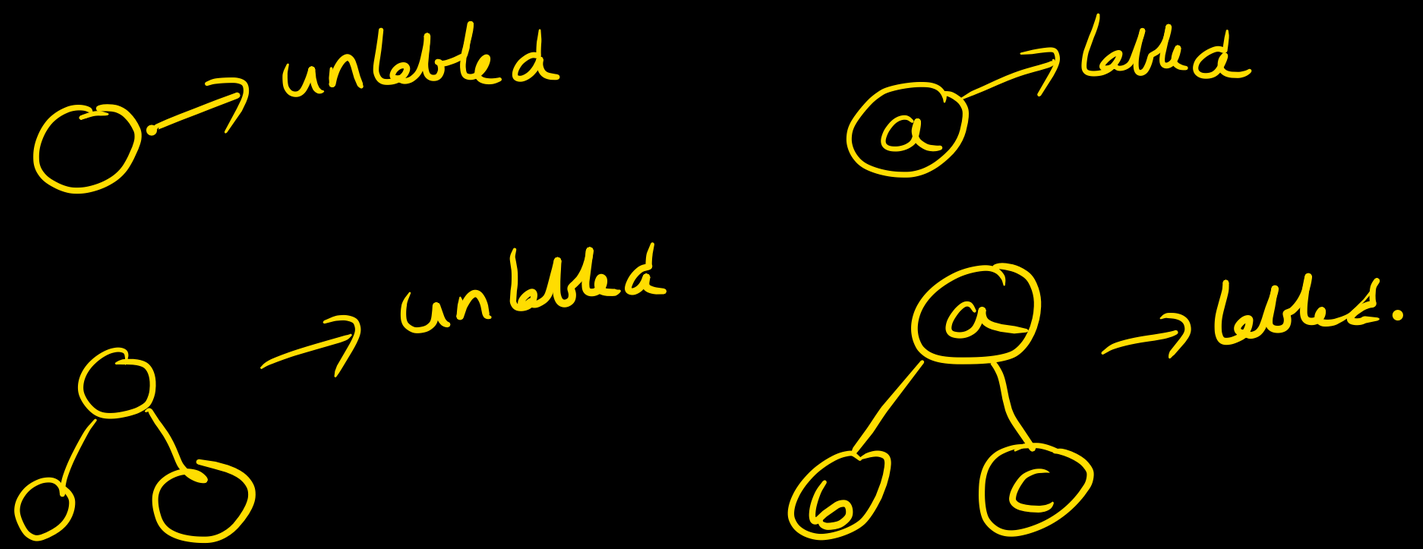
AdBefCg



	TC	SC
Pre :	O(n)	O(n) ✓
Inol :	O(n)	O(n) ✓
post :	O(n)	O(n) ✓





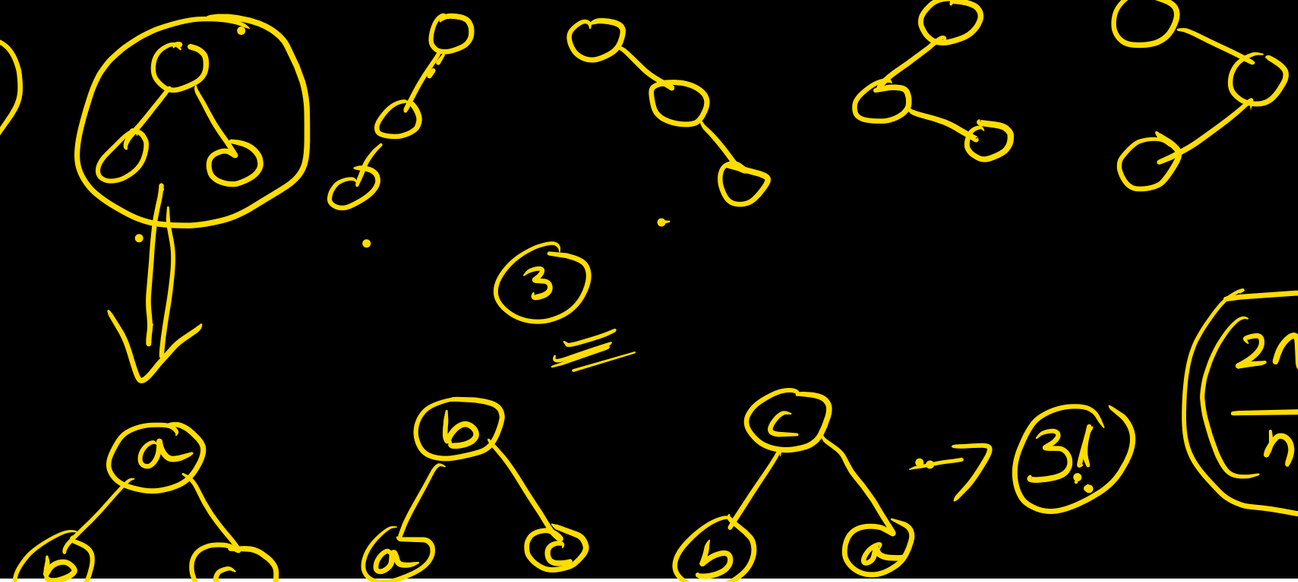

unlabled
labled
unlabled
a
b
c
labled.

Number of unlabled binary trees:

1)

2)

3)



$\left(\frac{2nCn}{n+1}\right)$

3

→ 3!

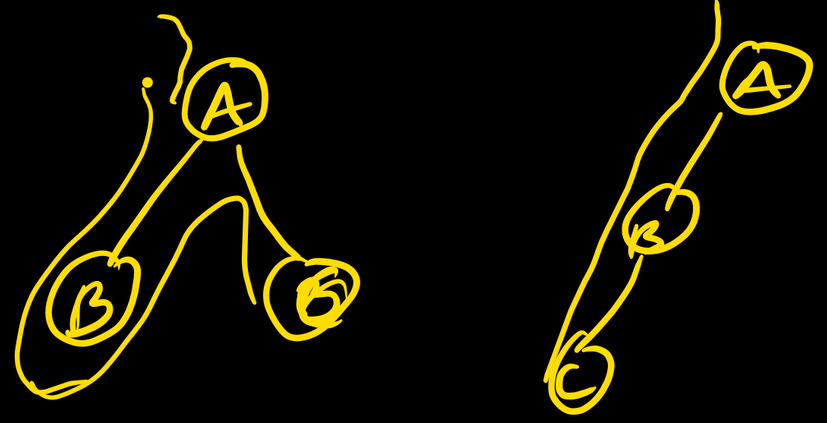
labled =

$\left(\frac{2nCn}{n+1}\right) * n!$

Geeta 2007 – III year B.tech. – 86 – 50

IV – 13

Construct a BT Given a pre order → uniquely



ABC . ABC.

Given a pre order we cannot construct a BT uniquely

" post " " " " "

" in order " " " " "

Given { In and pre, In and post } $\Rightarrow$ uniquely.

Given { pre and post } $\Rightarrow$ We cannot construct a BT uniquely.

IN 1 2 3 4 5 6 7 8

Pre 5 3 1 2 4 6 8 7

Root

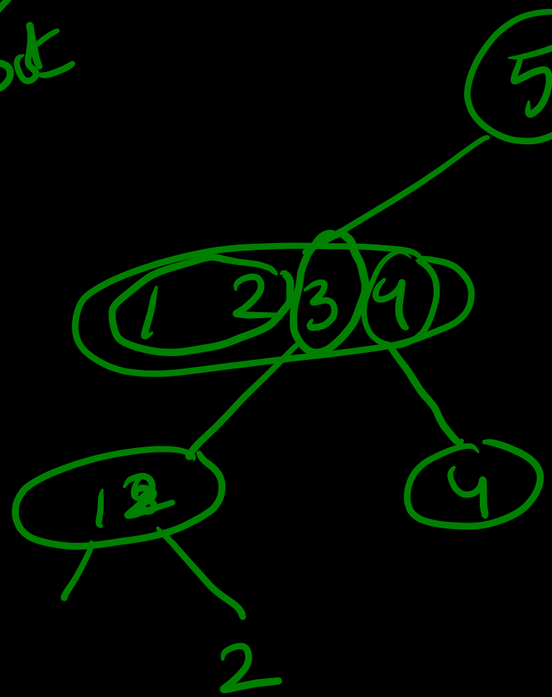
In: Left Root Right

Pre: Root left Right

In: 1 2 3 4 | 5 | 6 7 8 (Left, RST)

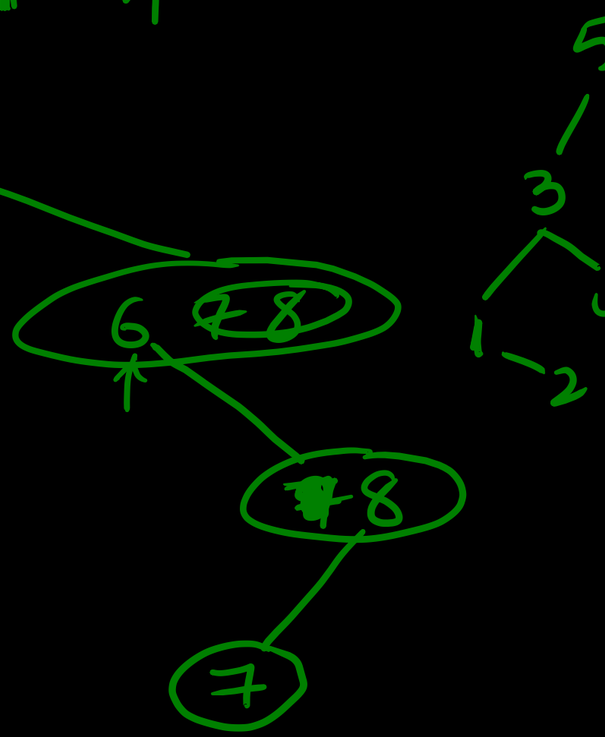
Pre: 5 | 3 | 1 2 4 | 6 8 7

5 = root



In: Left Root Right.

PRE: Root left Right.



DP:

DAC.

Dynamic

programming

optimization on the go

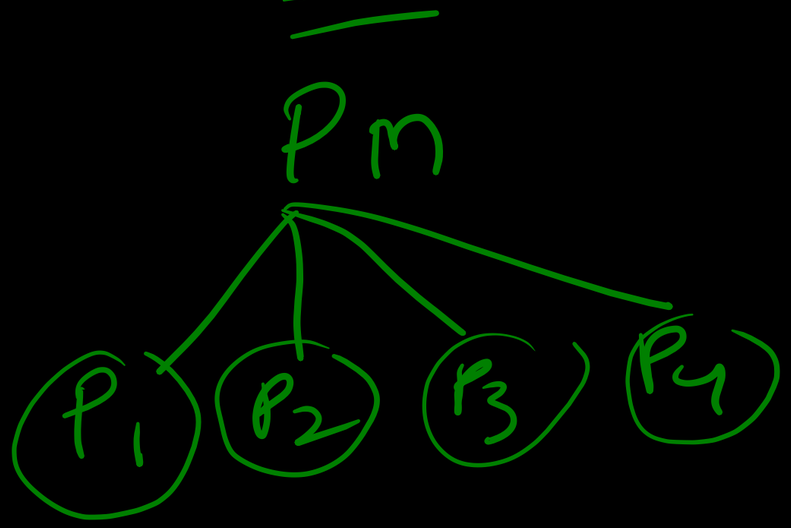
Table.

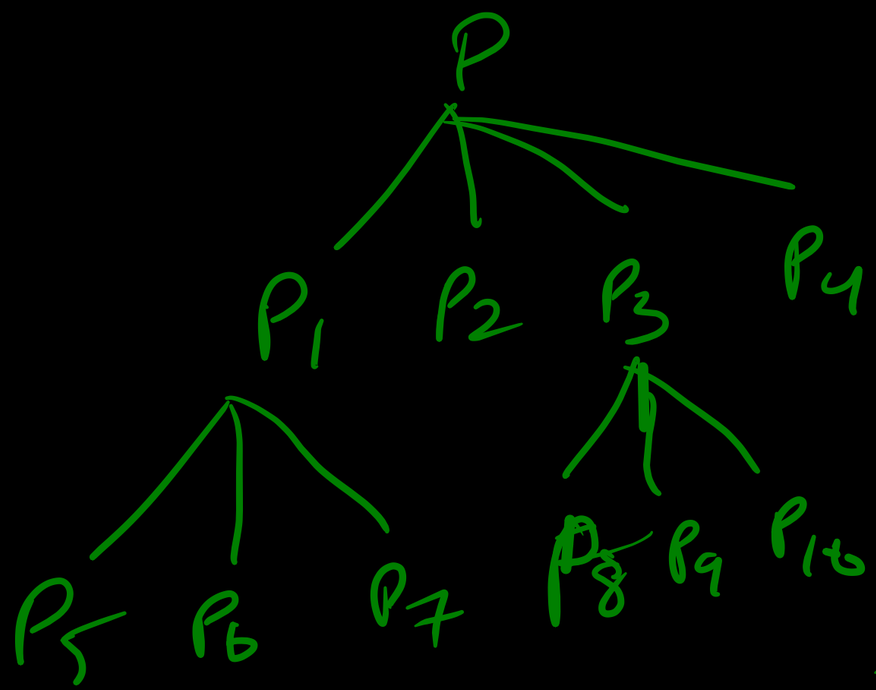
Knapsack problem

Yes d no

Subset sum.

DAC





DAC

↓

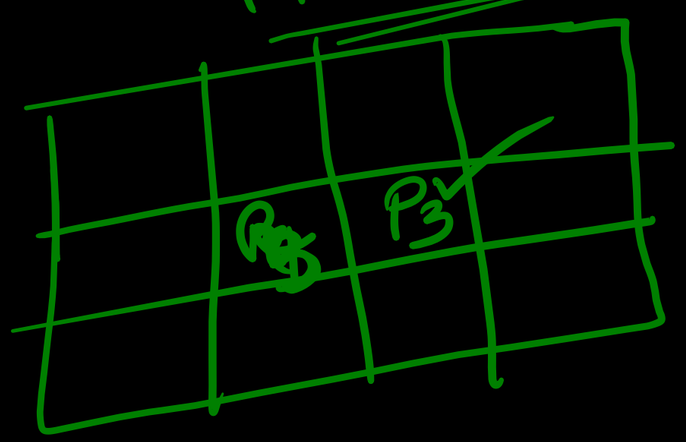
all of the problems are unique.

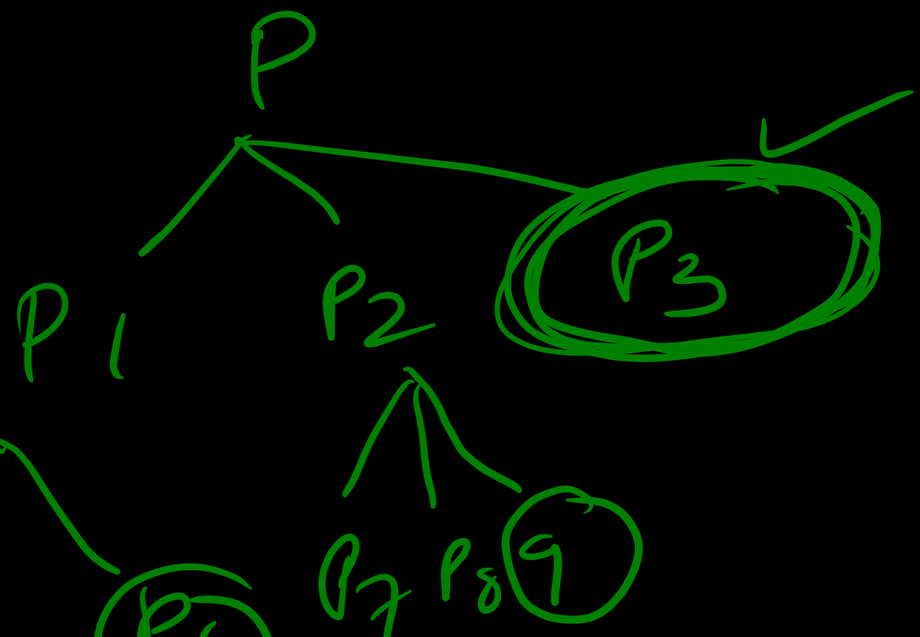
MS(1, 10)

MS(1,5)

MS(6,10)

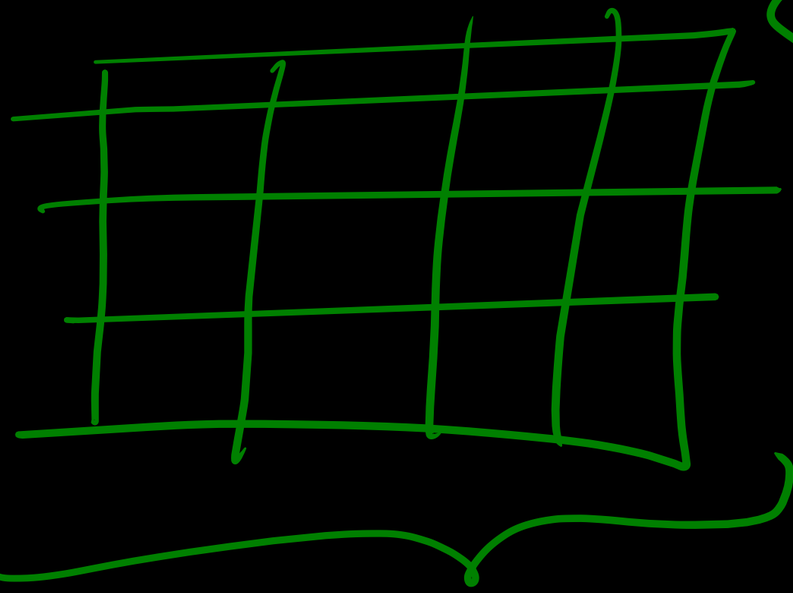
TOP Down approach





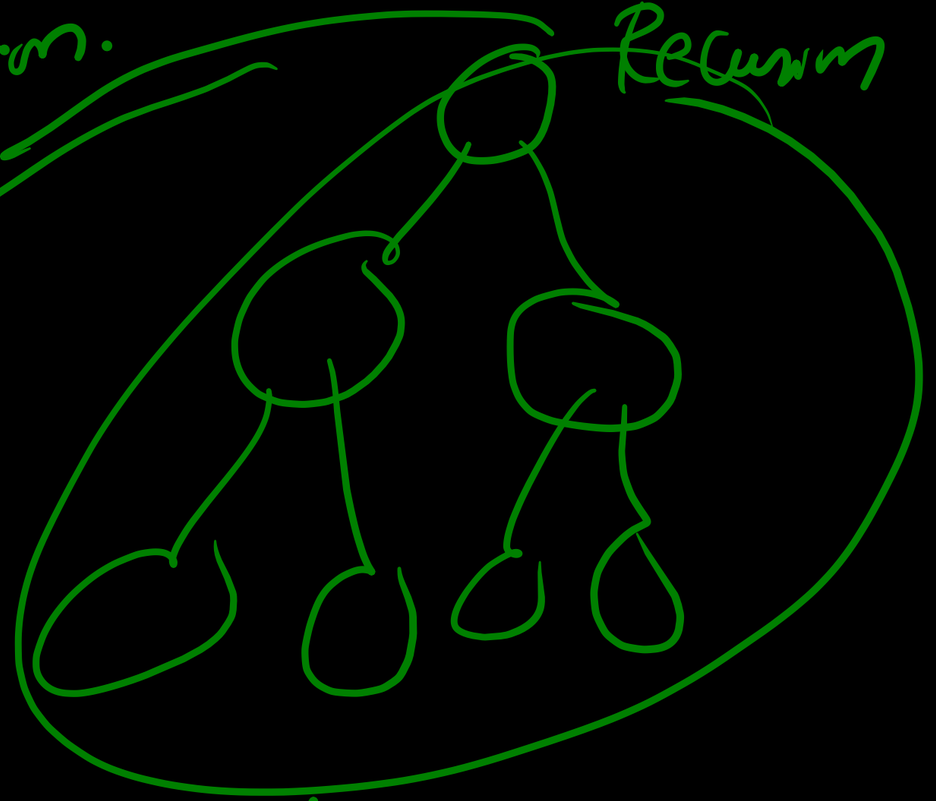
overlapping subproblem.
we will not solve them again

Bottom up dp.

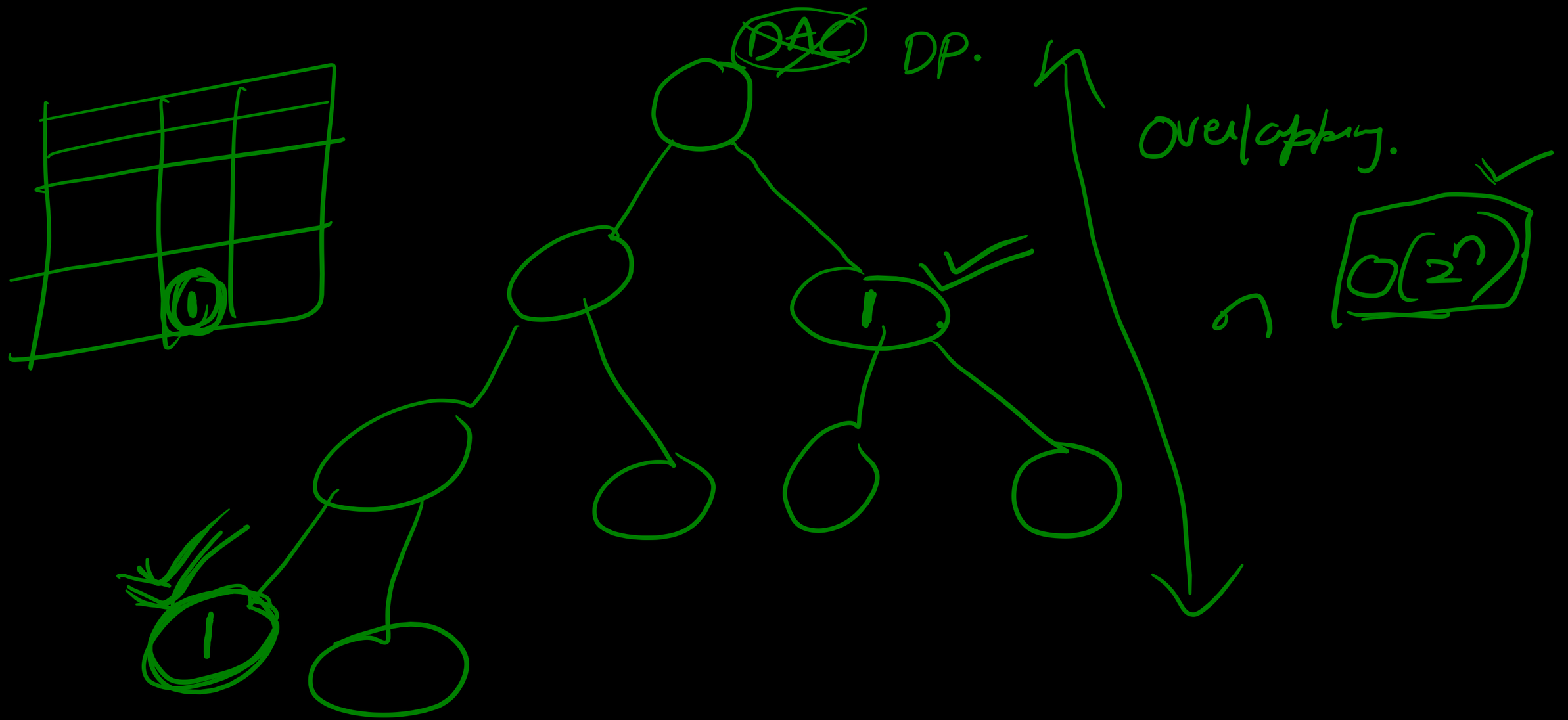


iteration.

Recursion



Recursive function ⇒ table.


DAC
DP.
overlapping.
O(2^n)

→ { ~~(Code)~~ Algo → what does this algo do.
they will say the algo → Fill in the blanks. }

able to write algo → Fill in the blanks
trace the algo and find out what it does.

(1) write algo. ✓

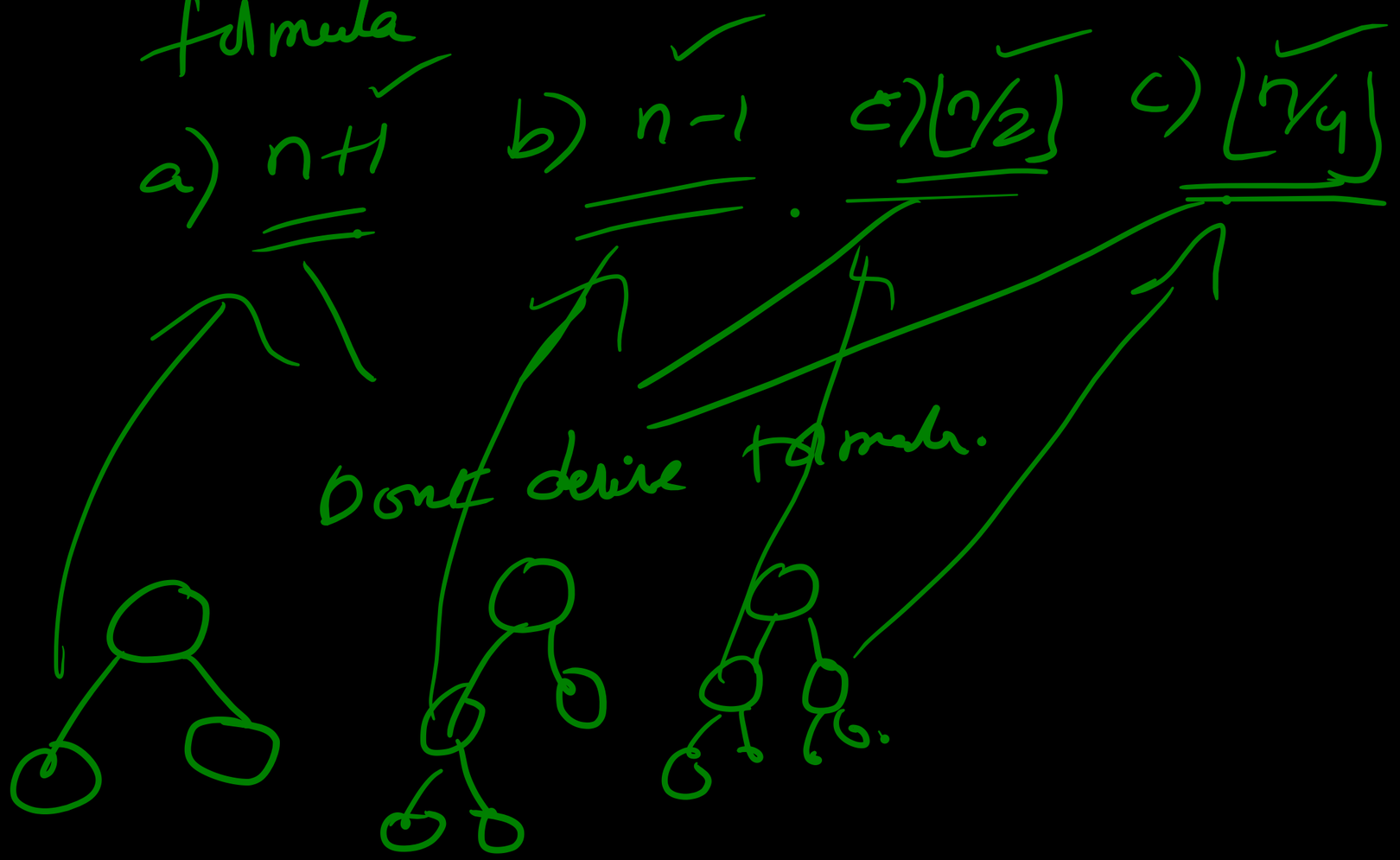
(2) ~~Oracle~~ trace them ✓

Gate:

formula

a) $n+1$ b) $n-1$ c) $\lfloor n/2 \rfloor$ c) $\lfloor n/4 \rfloor$

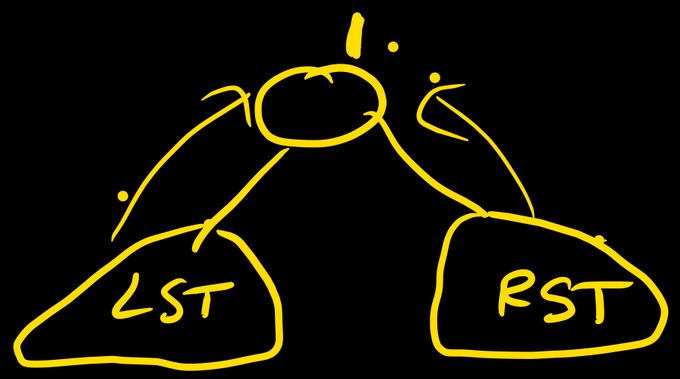
Don't derive formula.



Write an algo to count no of nodes in a BT.

Recursive Questions :→ RA → Trace. 99% recursive 1%

$$NN(T) = \begin{cases} 1 + NN(LST) + NN(RST); & \text{if } T != \text{null} \\ 0; & T \text{ is null} \end{cases}$$

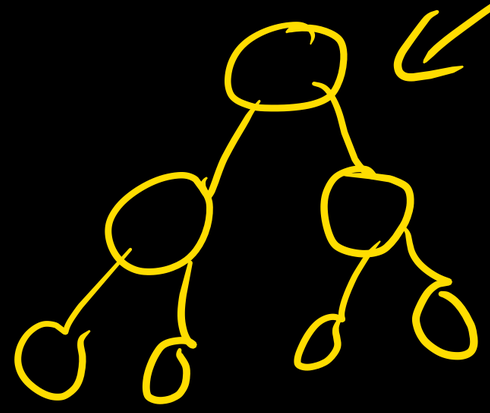


```
int NN( struct node *t)
{
  if(t)
  {
    return 1 + NN(t->left) + NN(t->right));
  }
  else
  return 0
}
```

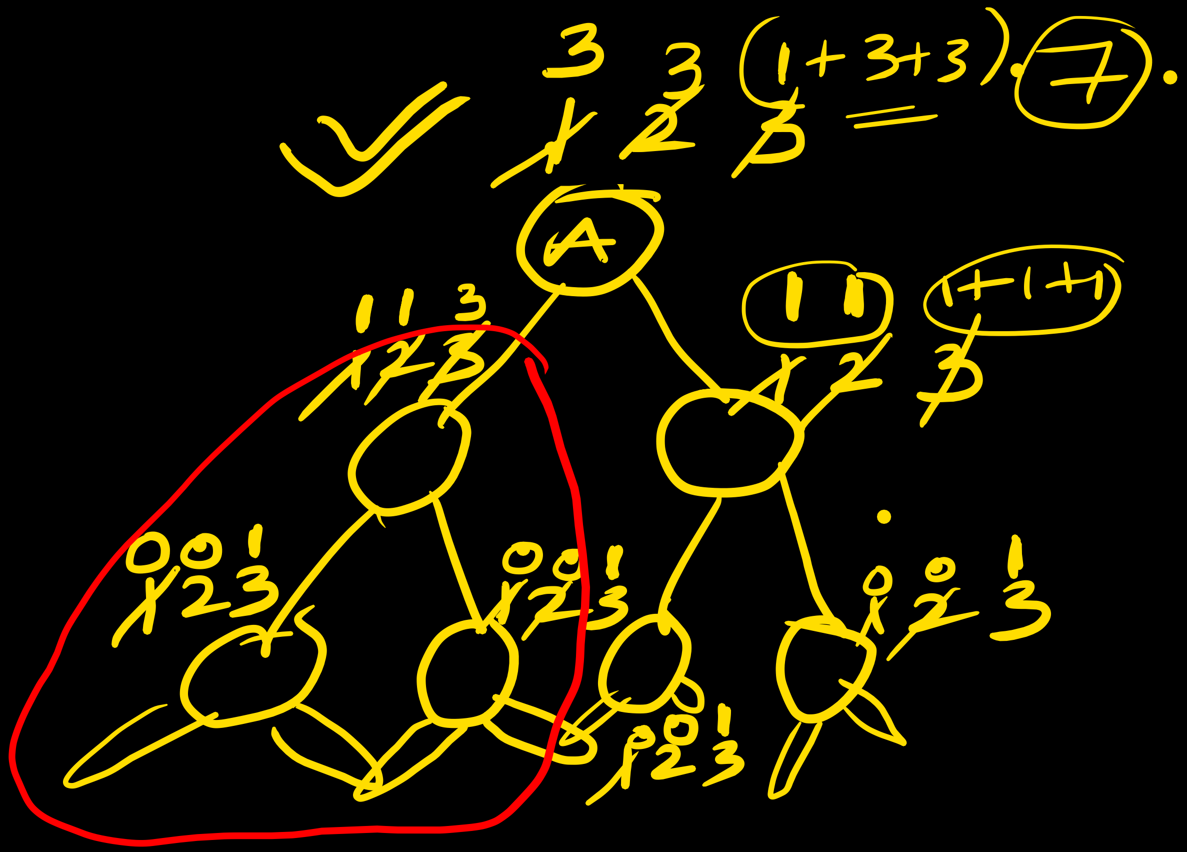
Trace

Random algo ⇒ what does it do

Trace the algo.



```
int   NN( struct node *t)
{
    if (t)
    {
      1 l = NN(t -> left)
      2 r = NN(t -> Right)
      3 return (1 + l + r)
    }
    else
      return 0
}
```



a) _____

b) _____

c) _____

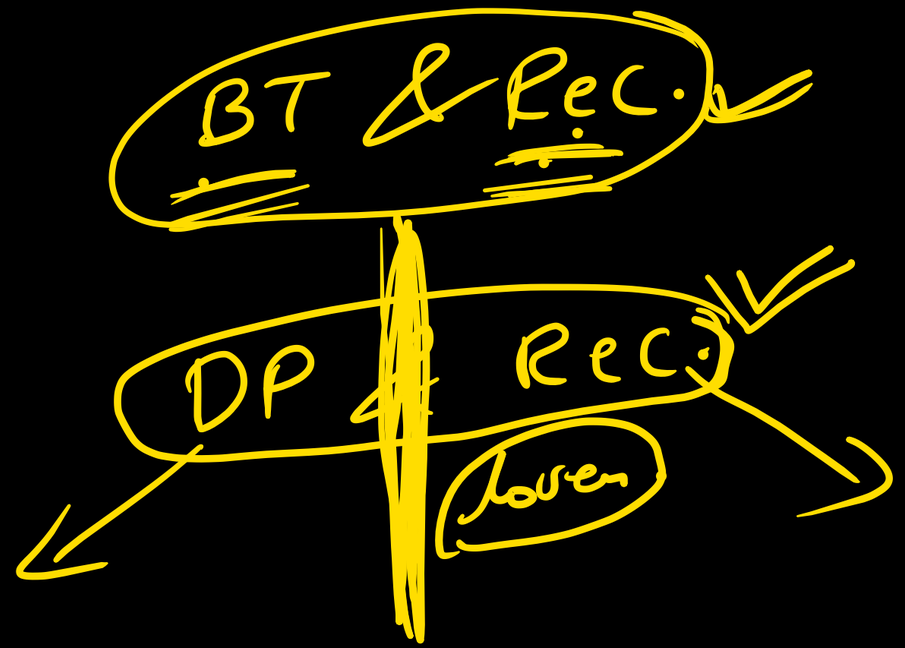
d) _____

_____ Count the nodes _____ Leave _____ Non leaf _____ to count the full nodes.

RE

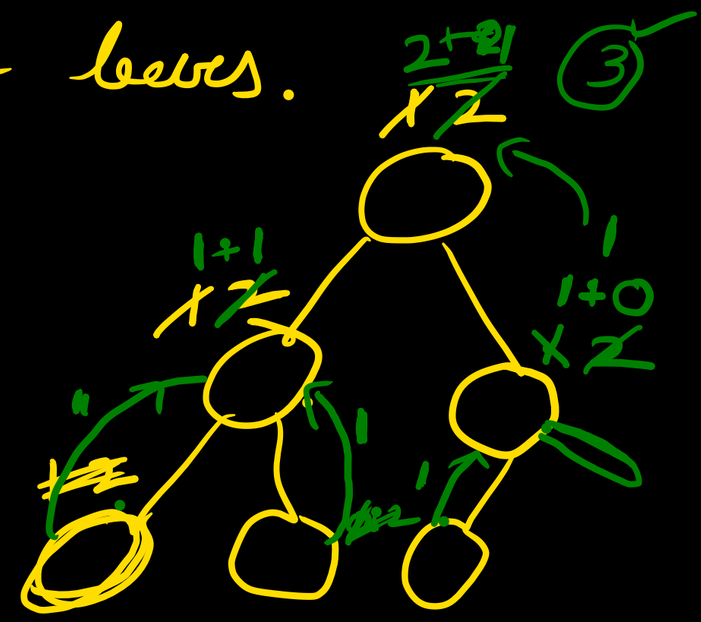
Recursive tree.

loops.



memoized recurse.

To write an algo to find the no of leaves.

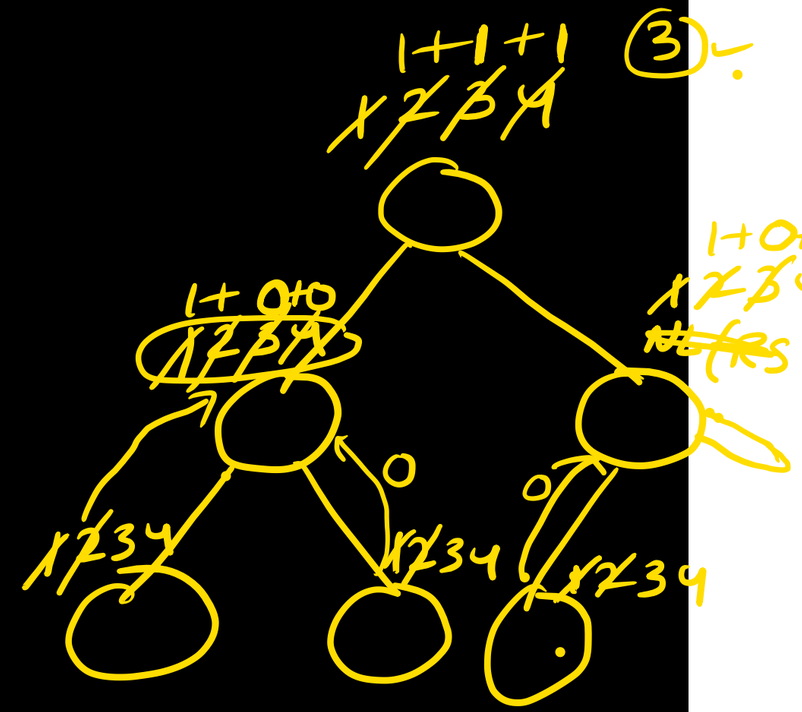



```
NL(Struct node *t)
{
if (t==NULL) return 0;
if (t->left ==NULL && t->right ==NULL)
return 1;
else return ( NL(t->left) + NL(t->right));
}
```

Write an algo to find no of non leaves

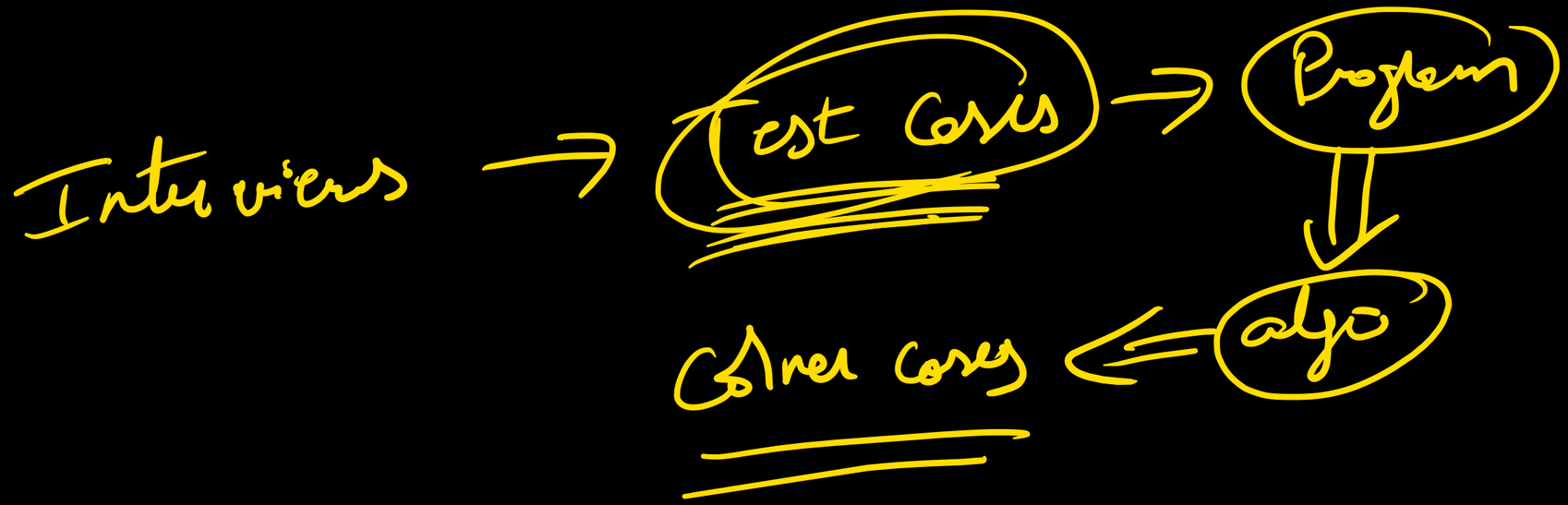
$$NL(T) = \begin{cases} 0 & \text{if } T \text{ is a leaf.} \\ 1 + \underbrace{NL(LST)} + \underbrace{NL(RST)} & \text{otherwise.} \end{cases}$$

```
int NL(struct node *t)
{
① if (t==null) return 0;
② if (t->left==null && t->right==null)
    return 0;
else return (1 + NL(t->left) + NL(t->right))
                  ③              ④
```

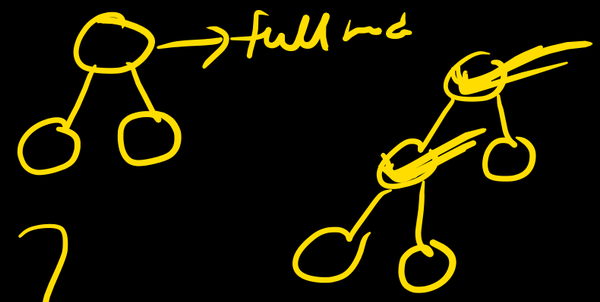


Sln write the algo.

other cases and dry run.



In a binary tree if a node has two children, then it is called full node.



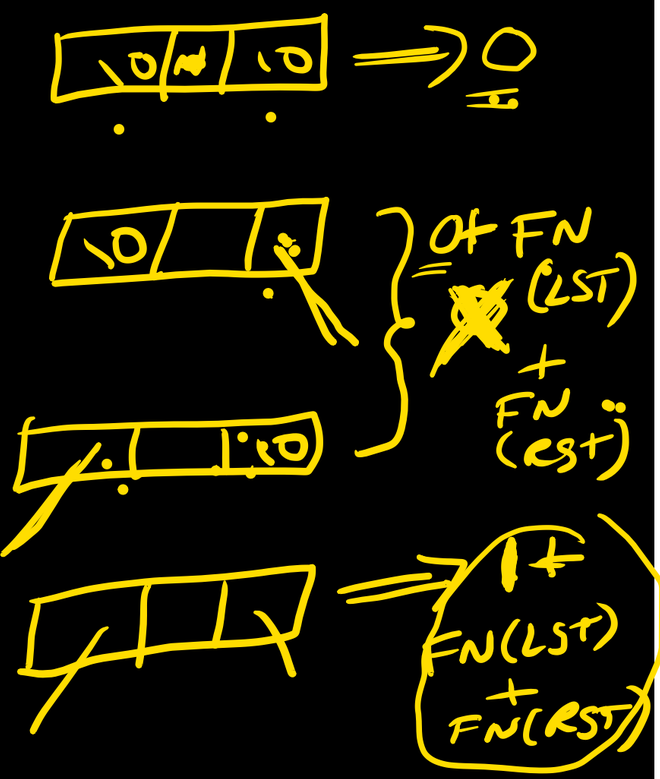
level: { Complete tree ⇔ multiple definitions
Full tree ⇔ multiple definitions
Strict tree ⇔ multiple definitions } Many books have many def.

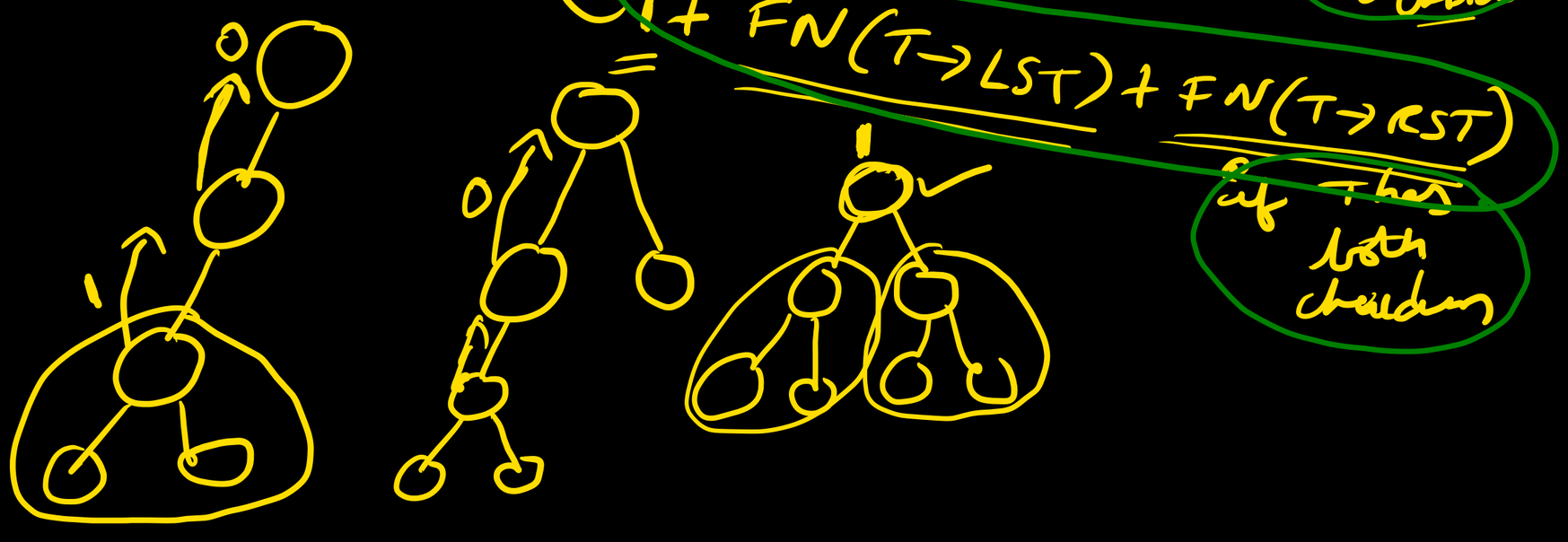
1002
2011
3020

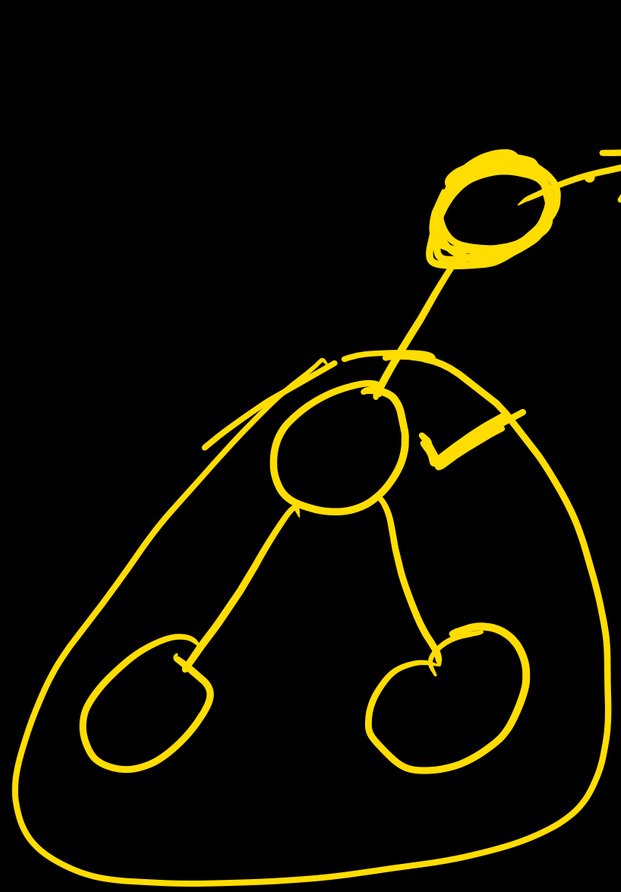
In the exam they will give you def.

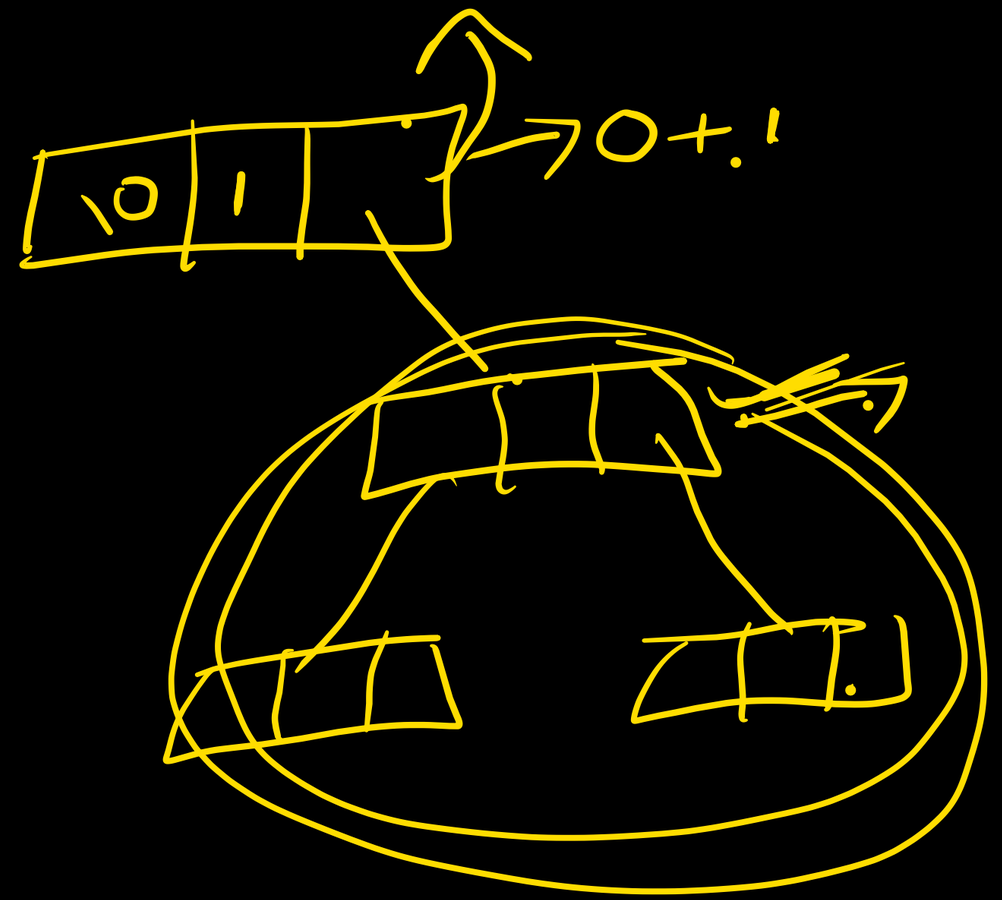
Write an algorithm to find no of full nodes.

$$FN(T) = \begin{cases} 0 \; ; \; T = \text{Null} \\ 0 \; ; \; T \text{ is a leaf} \\ 0 + FN(T \to LST) + FN(RST) & \text{if } T \text{ has only one child} \\ 1 + FN(T \to LST) + FN(T \to RST) & \text{if } T \text{ has both children} \end{cases}$$






0+1


10
1
0+1

```
int FN(struct node *t)
{
1  if (!t) return 0;
2  if (!t->left && !t->right)
       return 0;
   return (FN(t->left) + FN(t->right)
       + (t->left && t->right) ? 1 : 0);
```

not Null   not Null

Null for not Null