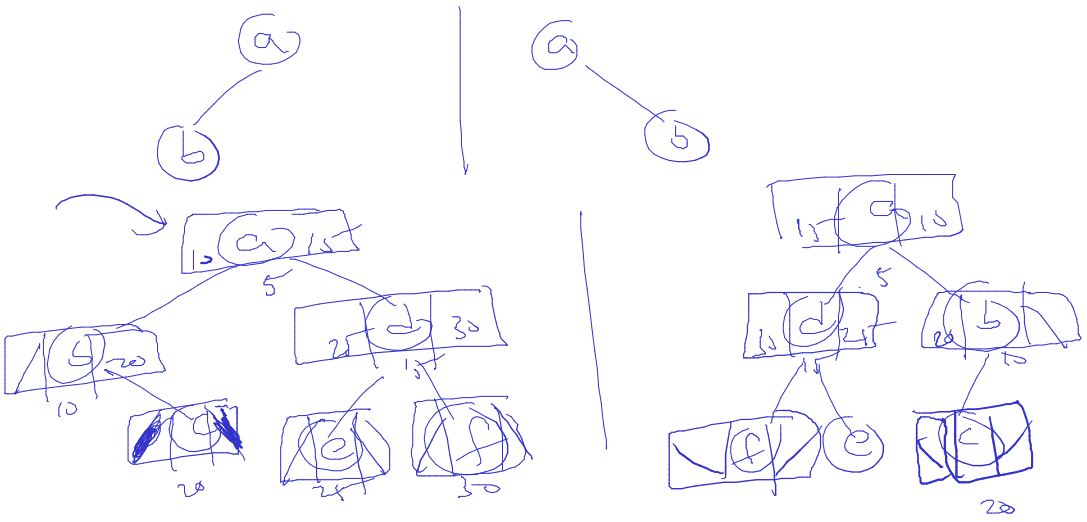
Mirror Image of a BT



```
void MI( &struct BTNode *t )
{
  if (t)
  {
    (1) MI (t->Lc);
    (2) MI (t->Rc);
    (3) SWAP(t->Lc, t->Rc);
  }
}
```

2M

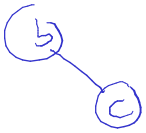
Children are getting swapped with

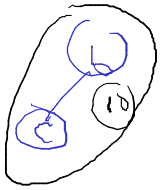
postorder style = c, b, e, f, d, a

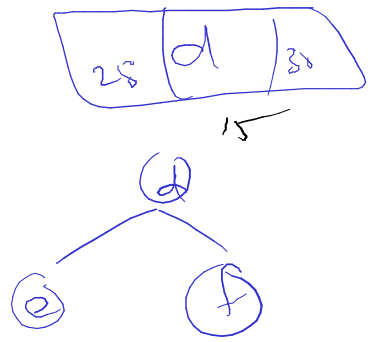
Tracing

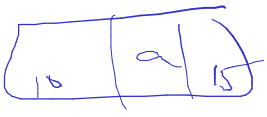
3 marks

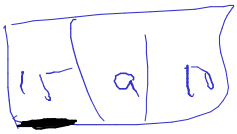
Before Swap	After Swap
	
	
	
	







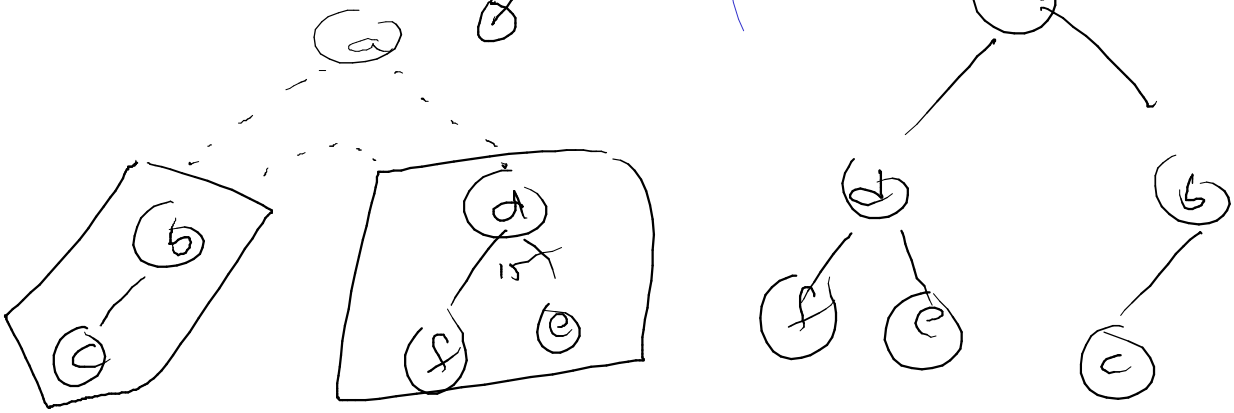




Before

Last Swap

After



= Max Depth = $\max\{H(LST), H(RST)\} + 1$

Height = No of Levels.

= From root to the leaf Node

Two ways

Root is @ Level 1

L=1 → @

H = 1

Root is @ Level 0

@ ← Level 0

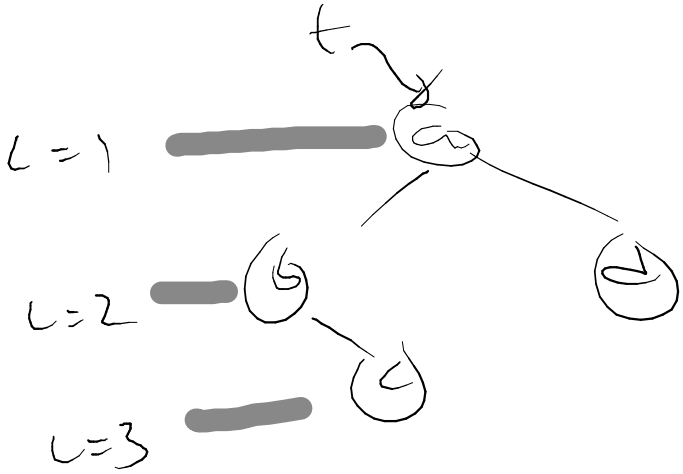
H = 0

Empty tree

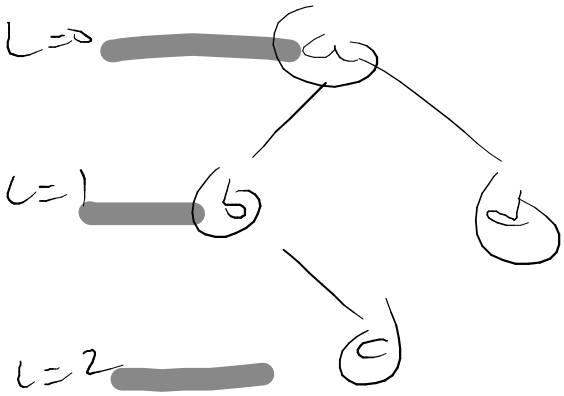
Height = 0

Empty tree

Height = -1

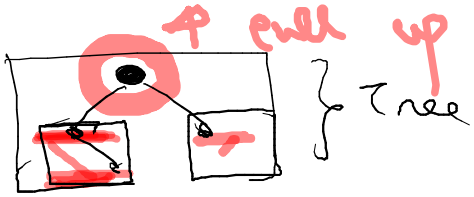


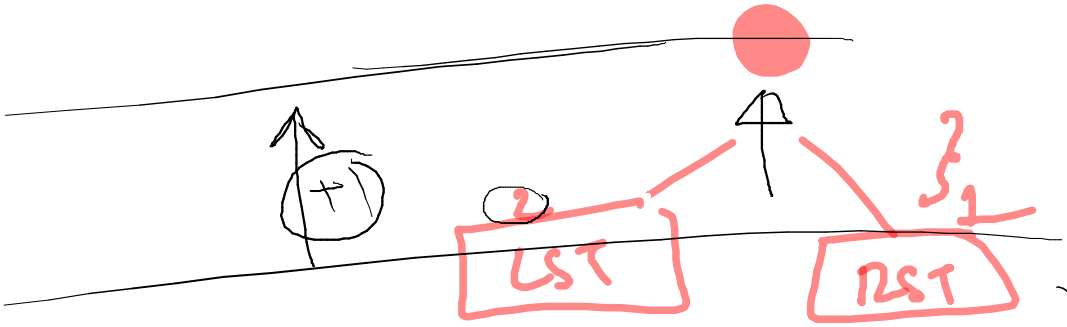
H(t) = 3



H(t) = 2

Logic for Height

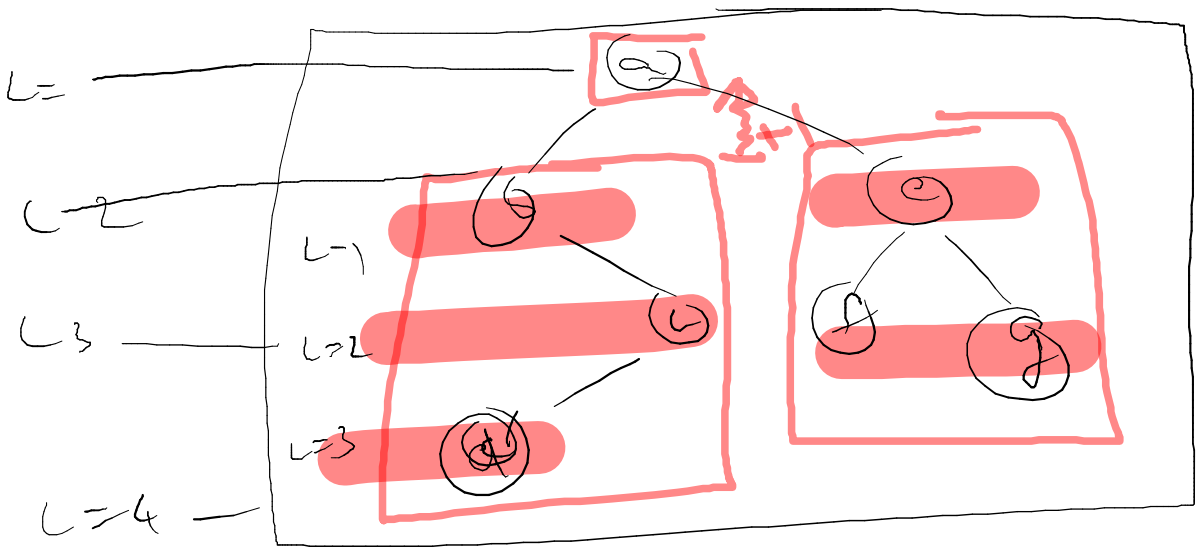


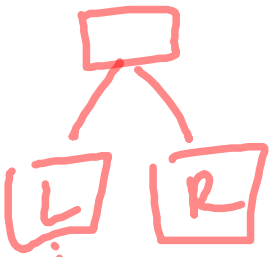


max(H(LST) , H(RST)) + 1

max (2 , 1) + 1

2 + 1 = 3





max(H(L), H(R) + 1)

= max(3, 2) + 1

3 + 1 = 4

```
int Height( struct BTNode * t )
{
    if (!t)
        return -1;          // Empty tree
    int xL = Height(t->L);
    int xR = Height(t->R);
    if xL > xR
        return (xL+1);
    else
        return (xR+1);
}
```

Root is @ Level 0.

The above function is with root is @ Level 1

If you want to have root is @ Level zero then simply change first step as

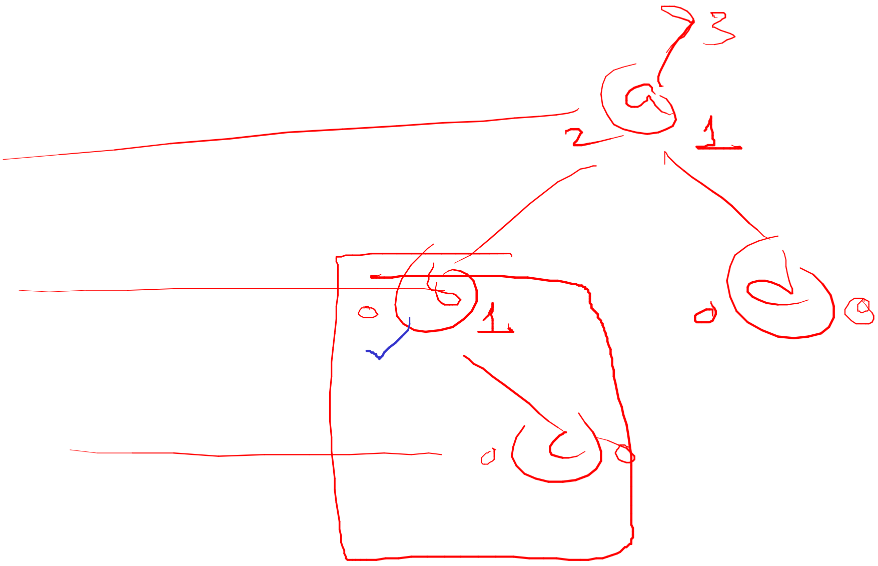
```
if (!t)
    return -1;
```

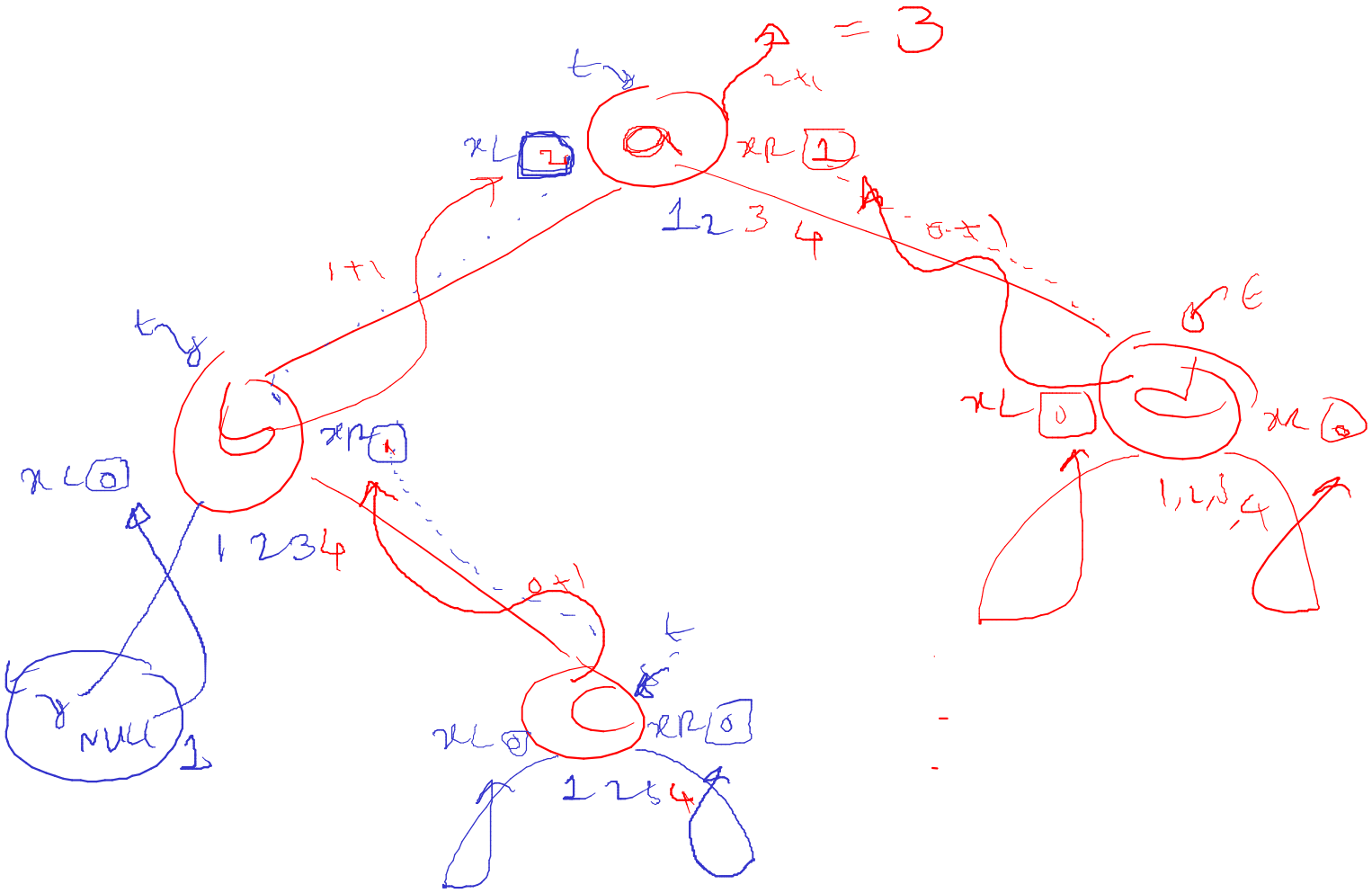
```
int Height( struct BTNode * t )
{
① if (!t)
      return 0;        } Empty tree

② int xL = Height(t->LC);
③ int xR = Height(t->RC);

④ if xL > xR
      return (xL+1);
   else
      return (xR+1);
}
```

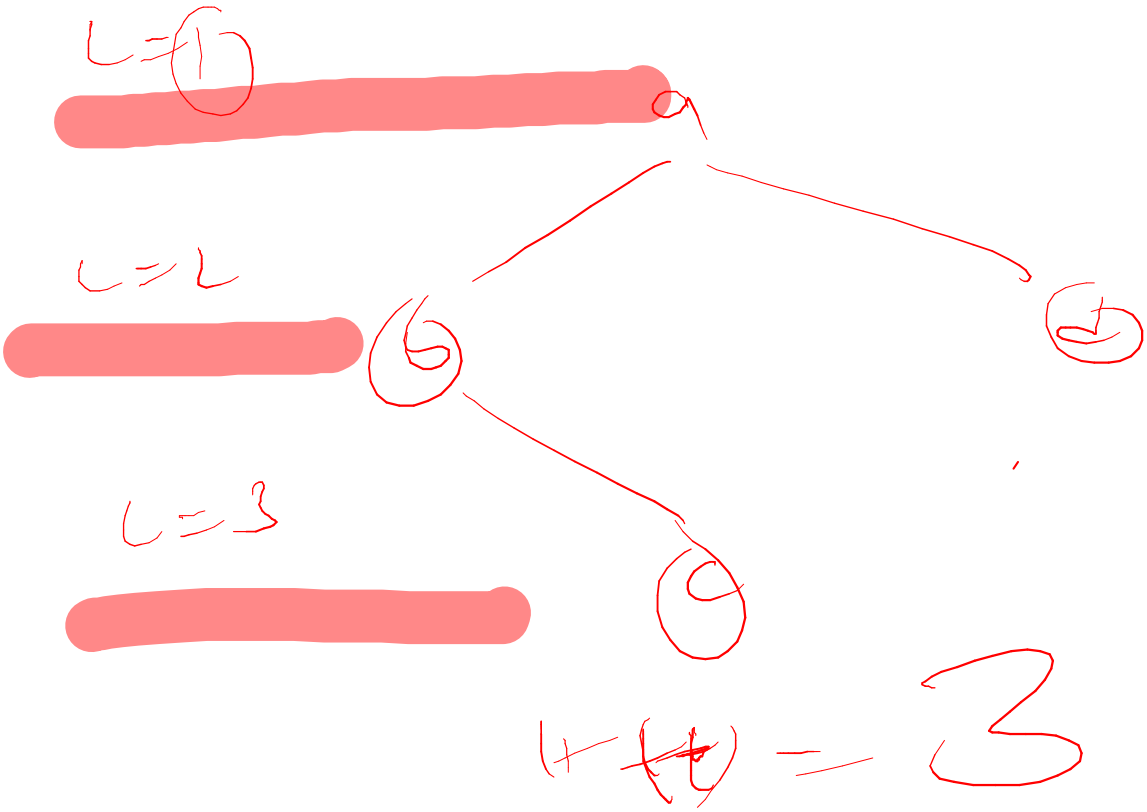
root is @ Level 1

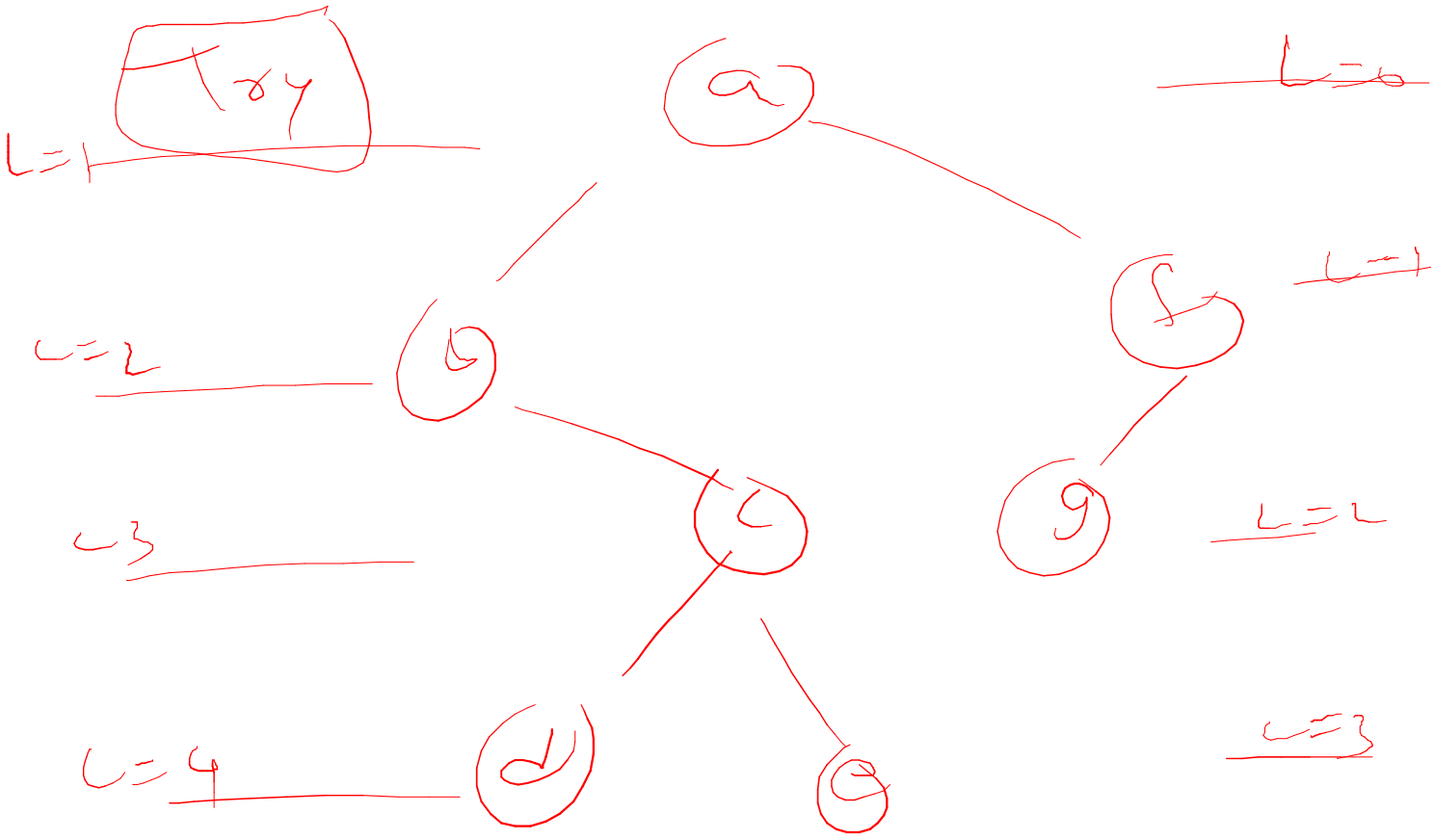
t = t->LC

Check




= 3
xL 2
xR 1
1 2 3 4
0+1
1+1
xL 0
xR 1
1 2 3 4
NULL
1
xL 0
xR 0
1 2 3 4
xL 0
xR 0
1,2,3,4


L=1
L=2
L=3
1+(1) = 3



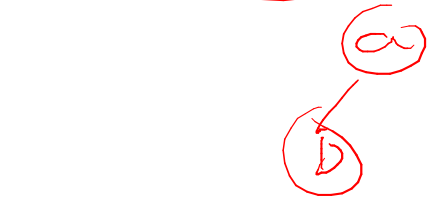
node no @ L=1

H=4

root is @ L=0

H=3

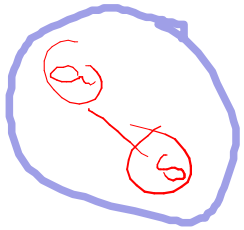
Logic for the following ⓐ



pre = ab

post = ba

IN = ba



pre = ab

post = ba

IN = ab

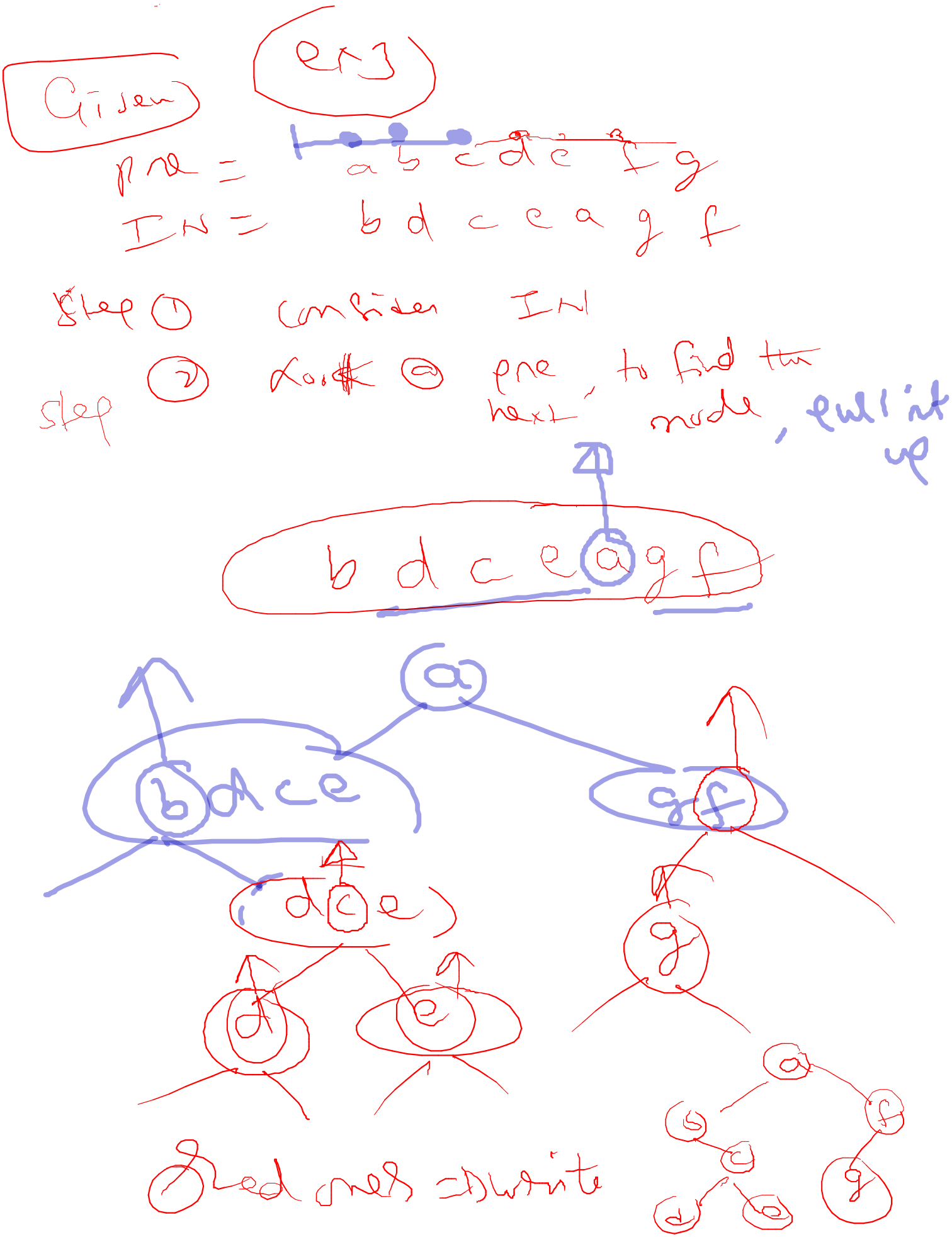
Logic

IN + pre

IN + post

(2M) (Q)

Generate binary tree whose preorder and inorder are given below:

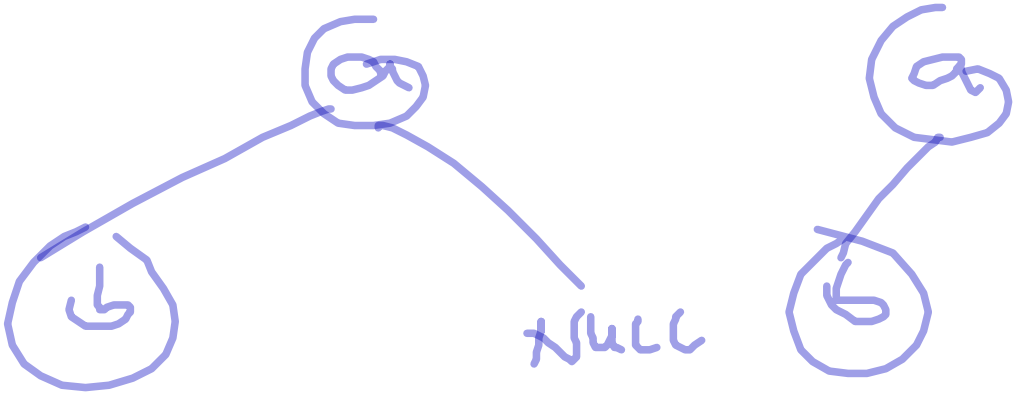


ex 1

pre = ba

IN = ba

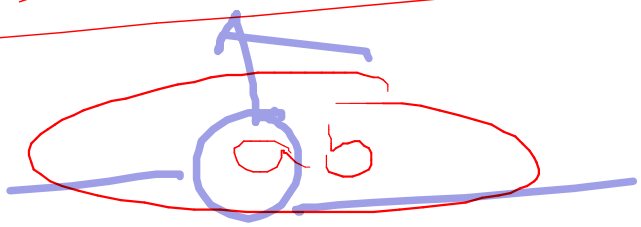
unique Tree

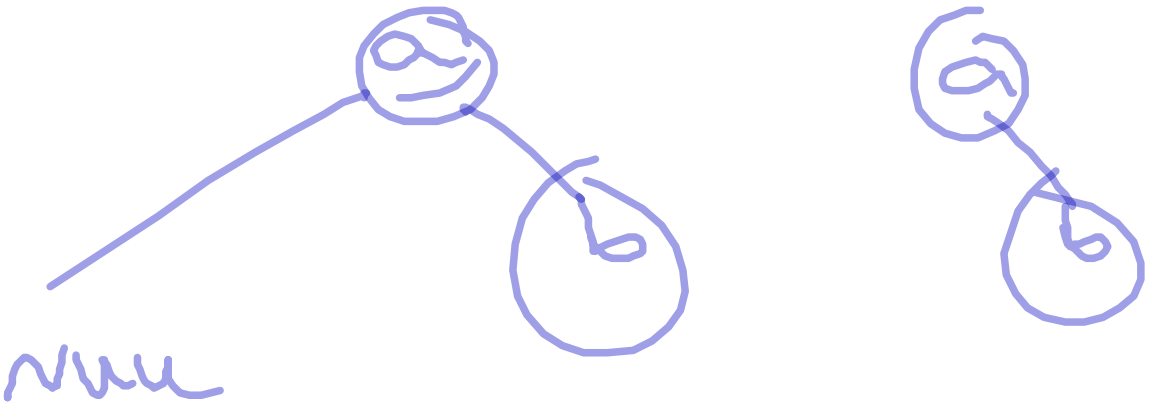


ex 2

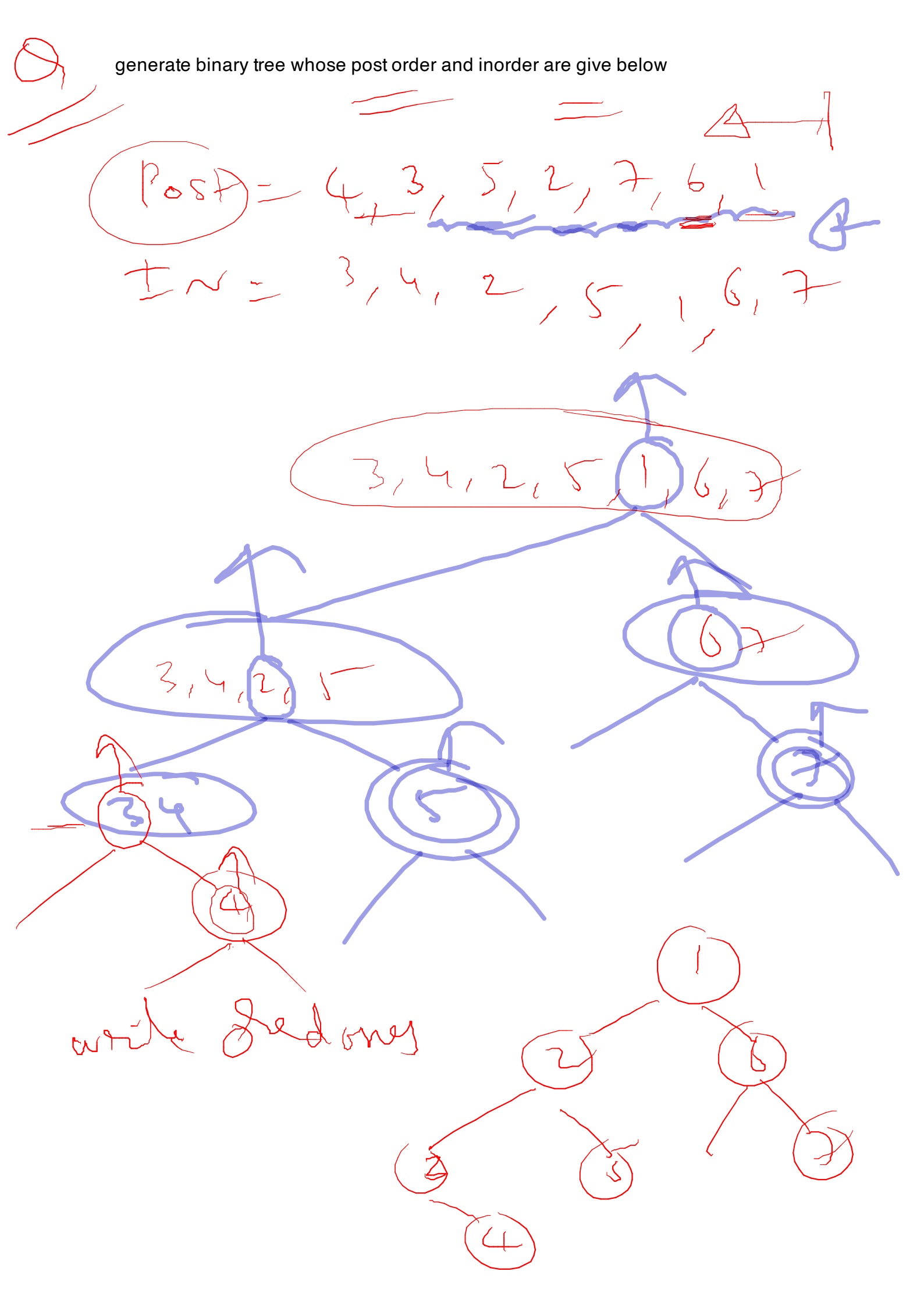
pre = ab

IN = ab



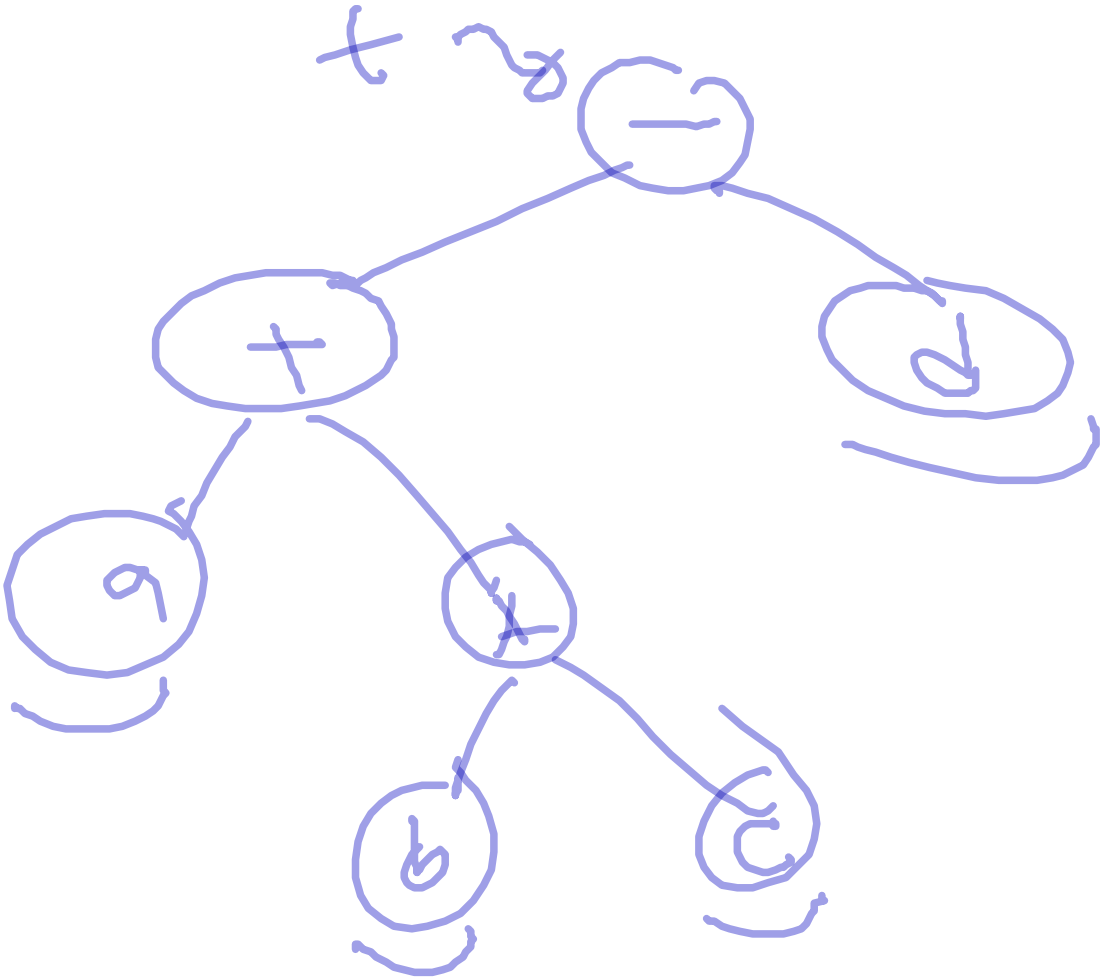


Q generate binary tree whose post order and inorder are give below



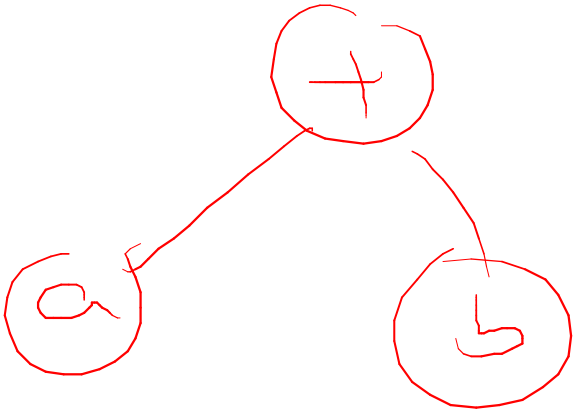
Generate B.T. (Expression)

a b c * + d −

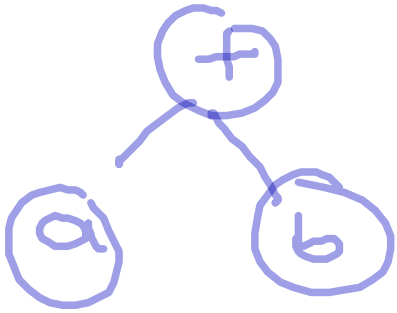


Expression trees

a + b



Logic



Preorder = Prefix = + a b

Inorder = Infix = a + b

Postorder = Postfix = a b +

Logic =

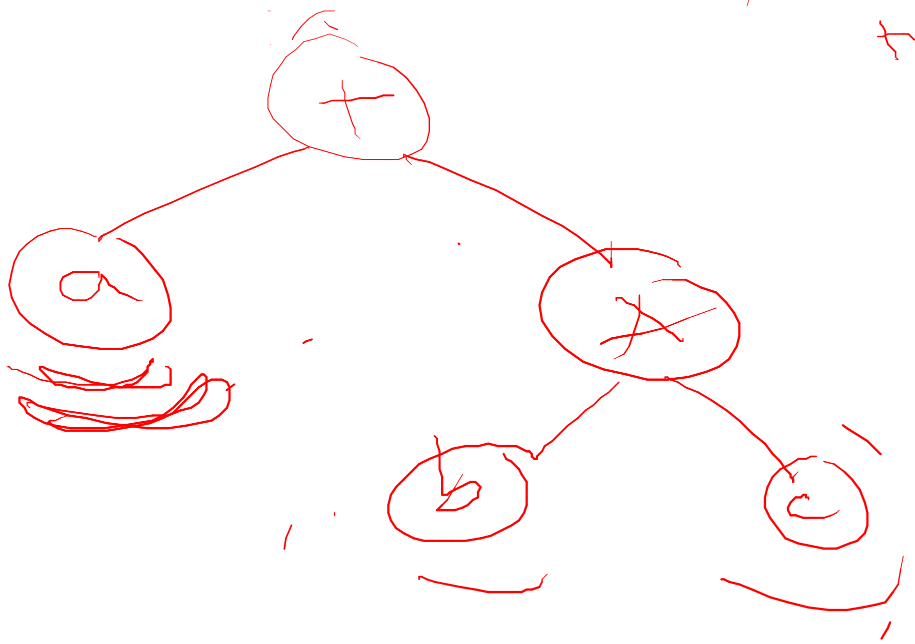
operator = Root / Parent Node

operands = Leaves

a + b × c

a + ○

pulling up = as per precedence/associativity rules



Pre = + a × b c

IN = a + b × c

Post = a b c × +

Post