- level order
- left / right view
- vertical order
- top / bottom view

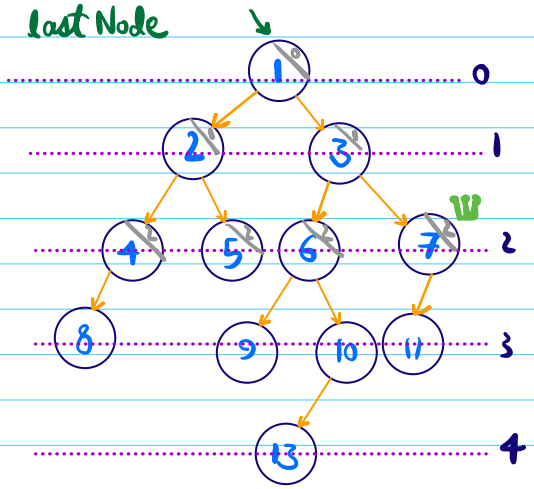
- bin tree types
- reveiw

Level order

level by level
left to right

1, "\n"
2, 3, "\n"
4, 5, 6, 7, "\n"
8, 9, 10, 11, "\n"
13 "\n"

output's



Queue

Fifo

output: 1 2 3 4 5 6 7 8 9 10 11 13

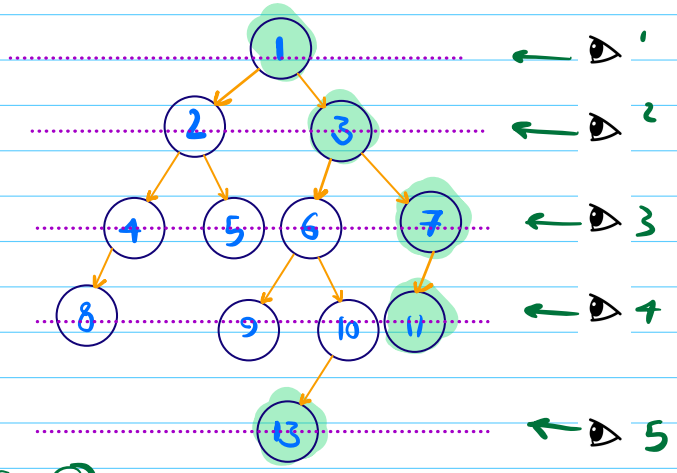
```
void levelOrder(Tnode r){
    if(r==null) ret;
    //Q initialize
    q.enq(r);
    TNode last=r
    while(!q.empty()){
        n=q.deq();
        print(n.val);
        if(n.left!=null) q.enq(n.left)
        if(n.right!=null) q.enq(n.right)
        if(n==last){
            print("\n")
            last = q.rear();
        }
    }
}
```

enq == enQueue
deq == deQueue

TC: O(n)
SC: O(n)

left right

P1 Right view of tree?

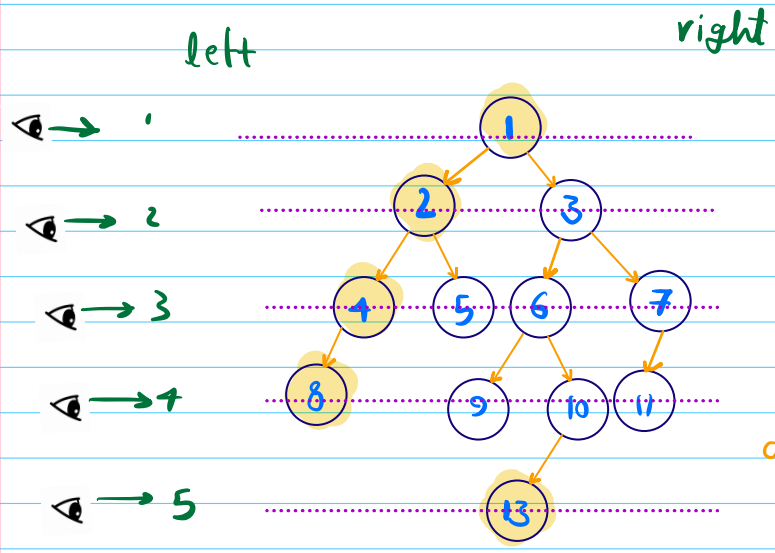


output ①, ③, ⑦, ⑪, ⑬

```
void levelOrder(Tnode r){
    if(r==null) ret;
    //Q initialize
    q.enq(r);
TNode ← last=r
    while(!q.empty()){
        n=q.deq();
        if(n.left!=null) q.enq(n.left)
        if(n.right!=null) q.enq(n.right)
        if(n==last){
            print(n.val);
            last = q.rear();
        }
    }
}
```

enq == enQueue
deq == deQueue

P2 left view of tree?

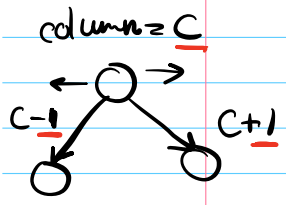


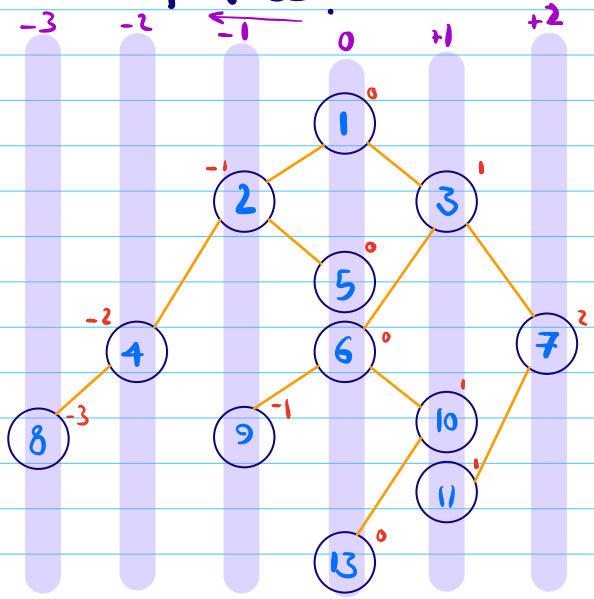
output: 1,2,4,8,13

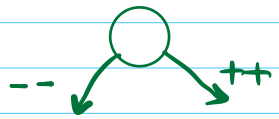
idea1: add right child then left child assignments

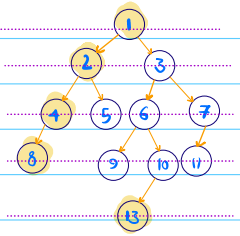
idea2: print the front of queue pass crown accordingly

P3 Vertical order of tree? →









*lower level comes first

* left nodes came first if both nodes same level

output 8, 4, 2, 9, 1, 5, 6, 13, 3, 10, 11, 7

NC Pair<TNode, int> (node, col)

Queue <3,H> <4,-2> <5,0> ... Q<NC>

<1,0> <2,-1>

Step 1: populate HM assign columns

```
void levelOrder(TNode r){
if(r == null) ret;
// HM<int, List<TNode>>
// Q initialize
q.enq(r, 0);
while(!q.empty()){
    nc = q.deq();
    n = nc.Node
    if(n.left != null) q.enq(<n.left, nc.col-1>)
    if(n.right != null) q.enq(<n.right, nc.col+1>)
    // check if key exist
    HM[nc.col].add(n)
}
}
```

enq == enQueue
deq == deQueue

<n.left, nc.col-1>

HM.add(nc.col, new list)

TNode ←

n

TC: O(n+n)
SC: O(n)

break

Step 2:

#Cols

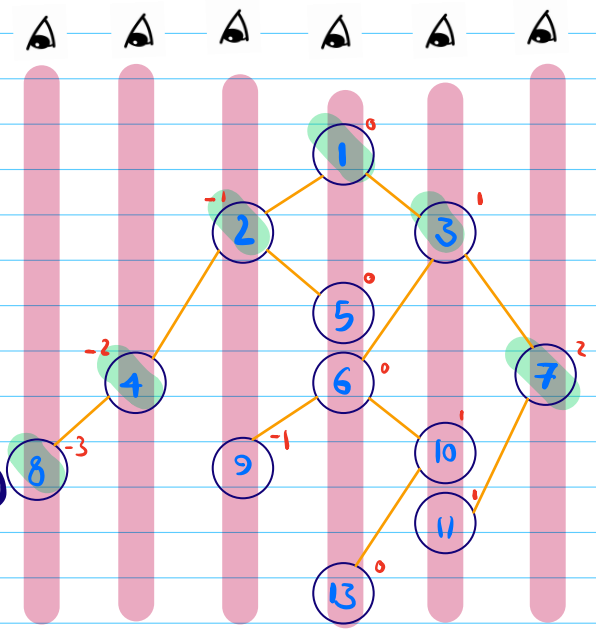
```
ks = HM.keySet →
minCol = MIN(ks)
maxCol = MAX(ks)
for(int i = minCol; i <= maxCol; i++){
    if(col i exist in HM)
    for(n in HM[i])){
        print(n.Val)
    }
}
```

List<TNode>

P4 top veiw of tree

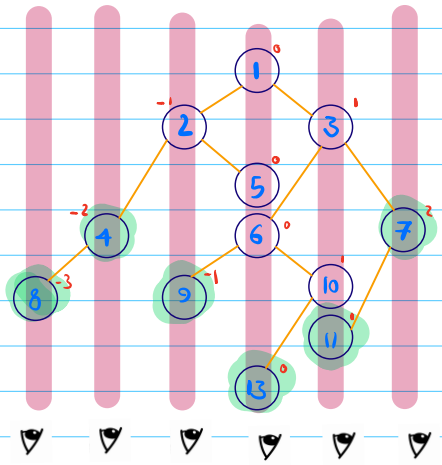
8, 4, 2, 1, 3, 7

Step 2:
```
#Cols
KS = HM.keySet →
minCol = MIN(KS)
maxCol = MAX(KS)
for(int i=minCol; i<=maxCol; i++){
    if(Col i exist in HM)
        print(HM[i].first().val)
    }
```



idea:

use last page code, print only first item of list

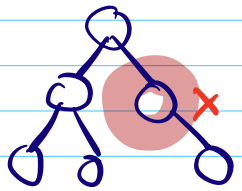


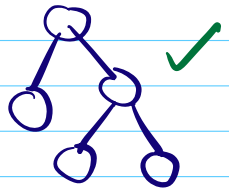
P5 bottom veiw of tree → assinment

types of bin trees
- proper
- complete
- perfect

```
isProper(Tnode n){
  if(n==null) ret true
  if(n.left!=null && n.right==null) ret false
  if(n.right!=null && n.left==null) ret false
  ret isProper(n.left) && isProper(n.right)
}
```

- proper either 0 or 2 children



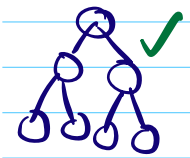


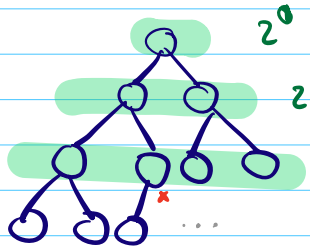
hints level order ← isComplete?

- Complete all levels are completly filled except than maybe the last level +

if last level not full, it should left to right in order

Quiz





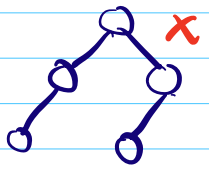
2^0 ✓

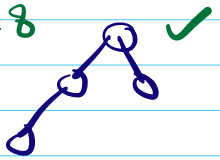
2^1

2^2

$2^3 = 8$

Complete but not proper

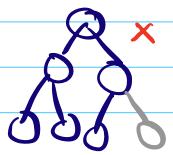


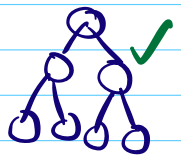


- Perfect all levels are completly filled

Quiz perfect ⇒ proper

Quiz



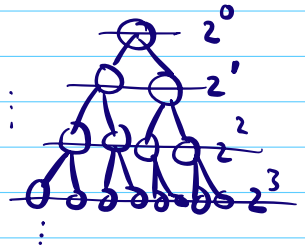


isPerfect? ← (#nodes == $2^{\text{height}} - 1$)

isPerfect? ↓ Count nodes with any traversal and check

P6 Find the height of a perfect bin. tree with n nodes

known ← n



$n \propto h$

$H+1$?

$n = 2^0 + 2^1 + 2^2 + \cdots + 2^H$

2x 2x

$q = 2$

$a = 2^0 = 1$

$\frac{a(q^n - 1)}{q - 1}$

$H = l - 1$

$n = \frac{1(2^{H+1} - 1)}{2 - 1}$

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

Last session P2 dry run

from last session

node vals are distinct

P2 Construct a tree from given inorder & Post order

0 1 2 3 4 5 6

LVRS 4, 2, 7, 5, 1, 3, 6

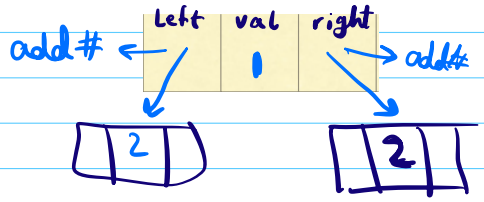
LRVS 4, 7, 5, 2, 6, 3, 1

hint

① last node of LRV is root

② inorder
... root ...
size of L subtree | index | size of R subtree

Tnode




```
tNode tree(in[], post[], sin, ein, spost, epost){
    if( sin > ein || spost > epost){
        ret null
    }
    root = new tNode(post[epost])
    rIndex = find_index_of_root_in_inorder(in[], sin, ein, data)
    lSize = rIndex - sin   // l subtree  sin → rIndex-1
    rSize = ein - rIndex   // r subtree  rIndex+1 → ein
    root.left = tree(in, post, sin, rIndex-1, spost, spost+lsize-1)
    root.right = tree(in, post, rIndex+1, ein, spost+lsize, epost-1)
    ret root
}
```

index index

tree 0, len-1 inclusive

initial call

data

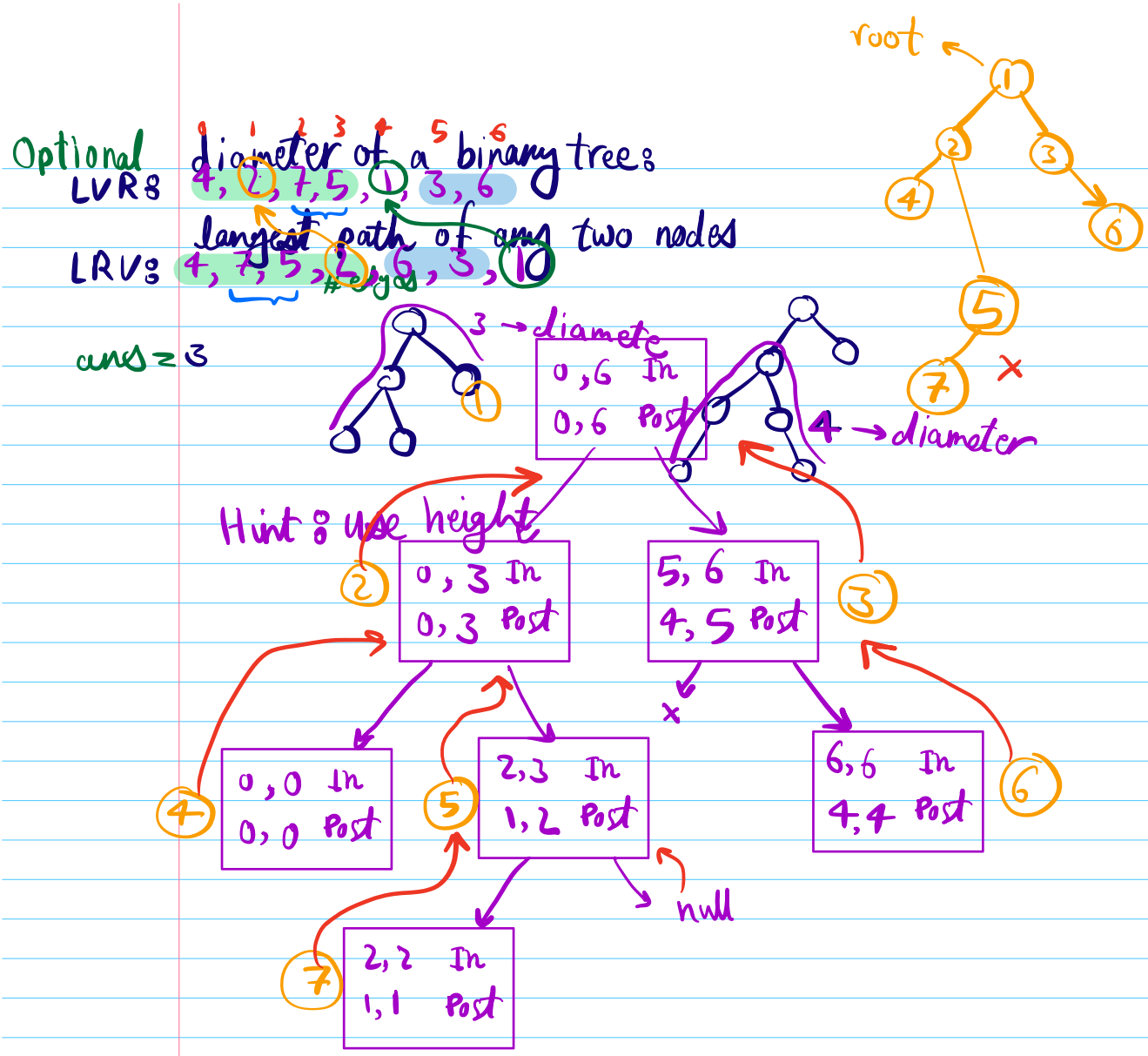
O(1) → 4

a-b+1

TC: O(n)

SC: O(H)

didn't use !!


Optional
diameter of a binary tree:
longest path of any two nodes
LVR: 4, 2, 7, 5, 1, 3, 6
LRV: 4, 7, 5, 2, 6, 3, 1
# edges
ans = 3
root
3 → diameter
4 → diameter
Hint: use height
0, 6 In
0, 6 Post
0, 3 In
0, 3 Post
5, 6 In
4, 5 Post
0, 0 In
0, 0 Post
2, 3 In
1, 2 Post
6, 6 In
4, 4 Post
2, 2 In
1, 1 Post
null
x
x