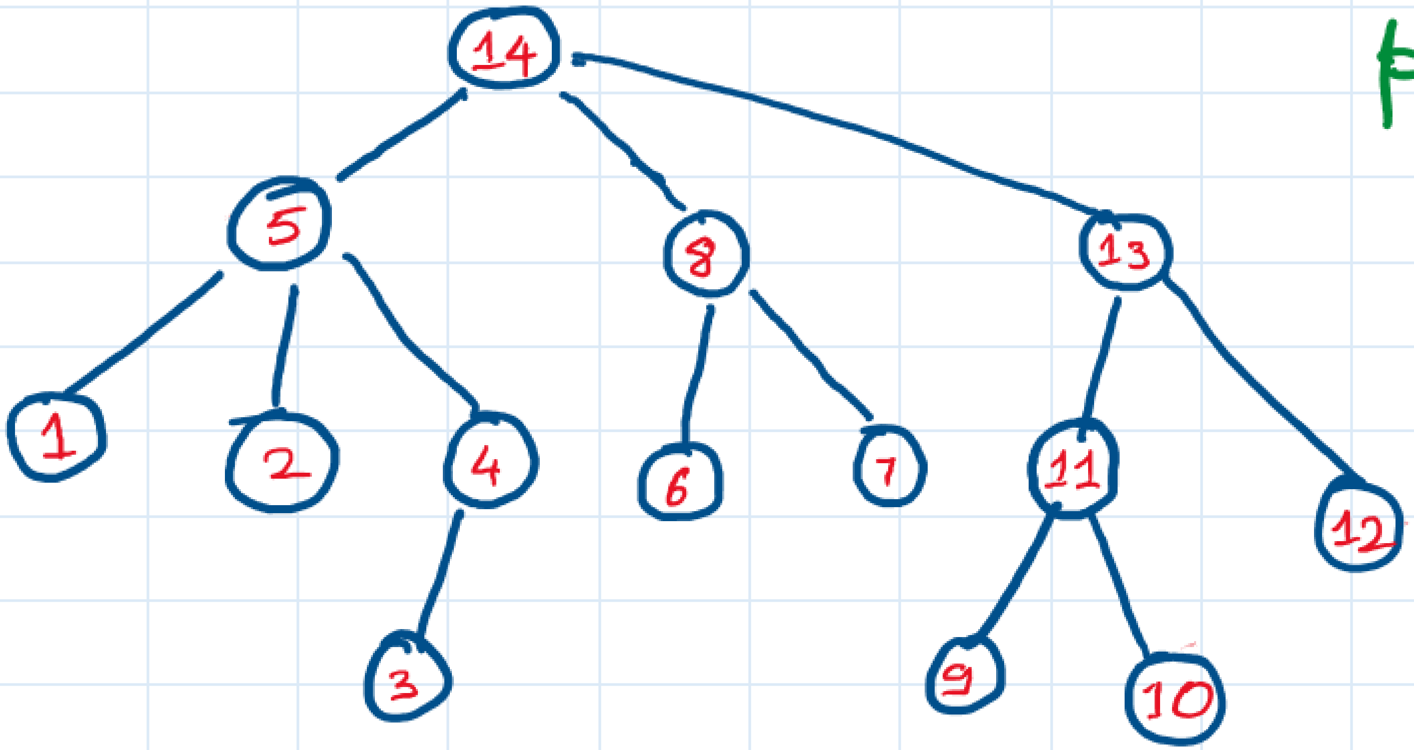
COL106 - Data Structures and Algorithms

POST ORDER TRAVERSAL OF TREE T

- TRAVERSE SUBTREES ROOTED AT ITS CHILDREN
  - In ORDERED TREES MAINTAIN ORDER.
- VISIT ROOT OF T




```
post order (p) :
    for each c in
            children(p)
        post order(c)
    visit (p);
```

RUNNING TIME OF PRE-/POST-ORDER

AT EACH TREENODE,

THE NONRECURSIVE PART OF THE ALGORITHM NEEDS $O(c_p+1)$ WORK.

(RECALL THAT WE ASSUMED "VISIT" IS $O(1)$)

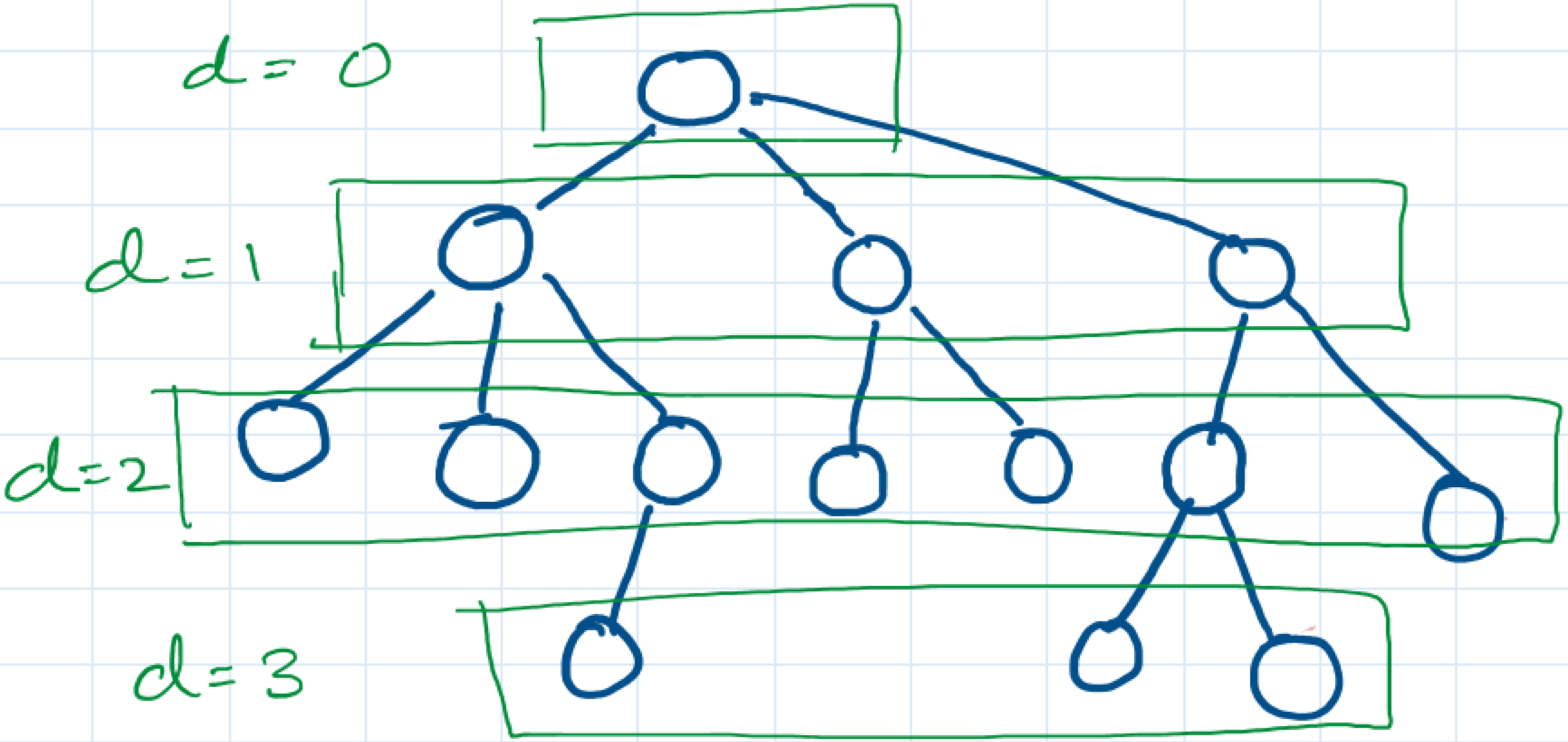
WE ALREADY SAW THAT $\sum c_p = n-1$

$\Rightarrow$ OVERALL RUNNING TIME $= O(n)$

Can we do better?

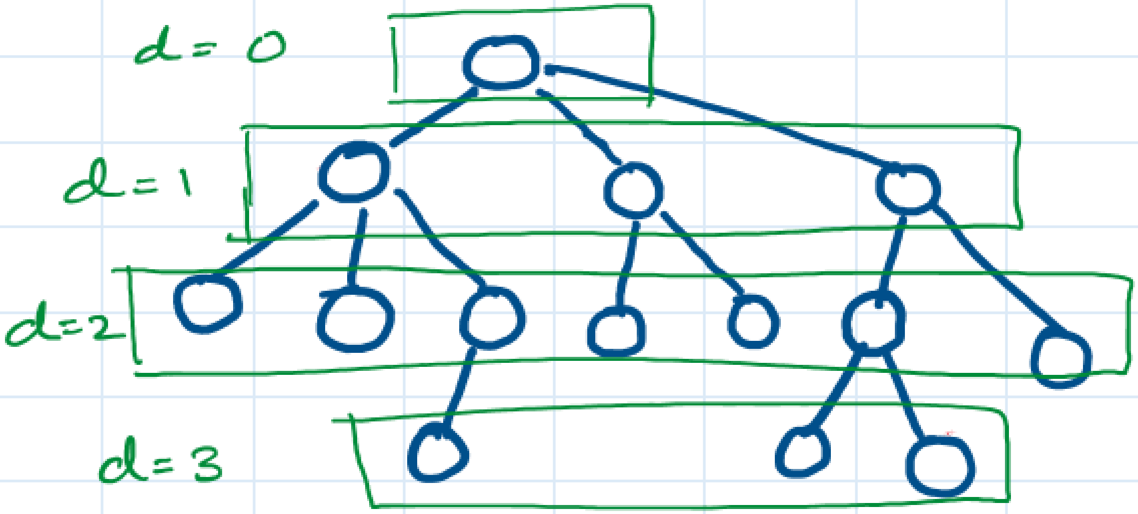
BREADTH-FIRST TRAVERSAL

VISIT ALL NODES AT DEPTH d BEFORE VISITING NODES AT d+1



BREADTH - FIRST TRAVERSAL

VISIT ALL NODES AT DEPTH d BEFORE VISITING NODES AT d+1



QUEUE NODES AT EACH LEVEL.

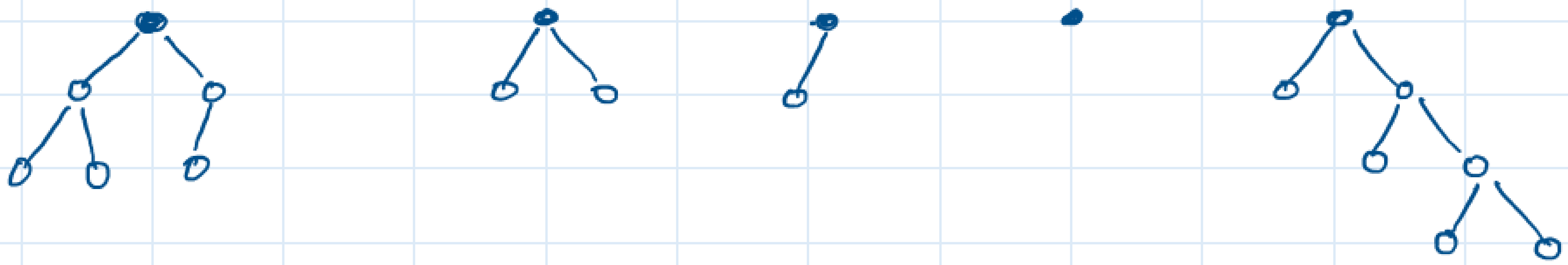
```
breadthFirst():
    Q ← EMPTY QUEUE.
    Q.enqueue(root);
    while Q not Empty
        p = Q.dequeue()
        visit p
        for each children(p)
            Q.enqueue(c)
```

What is the complexity?

BINARY TREES

- ORDERED TREES
- EACH NODE HAS **AT MOST** TWO CHILDREN : **LEFT, RIGHT**
- LEFT PRECEDES RIGHT.

A PROPER BINARY TREE IS THE ONE WHERE EVERY NODE HAS **0 OR 2** CHILDREN



BINARY TREES ARE VERY POPULAR -

BINARY TREE ADT & APPLN.

- IT EXTENDS TREE ADT WITH

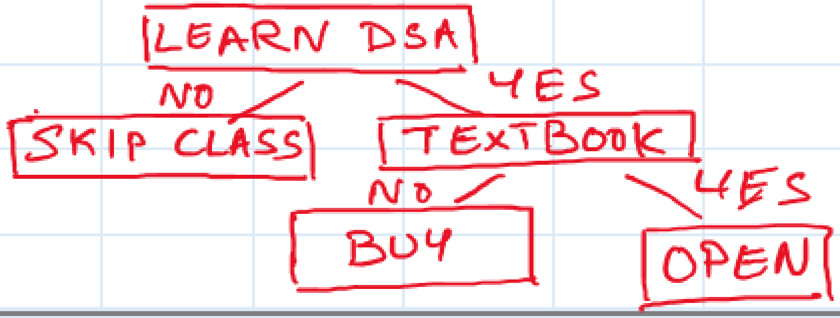
left(p) : RETURNS REF TO LEFT CHILD NODE (OR NULL)

right(p) : RETURNS REF TO RIGHT CHILD NODE (OR NULL)

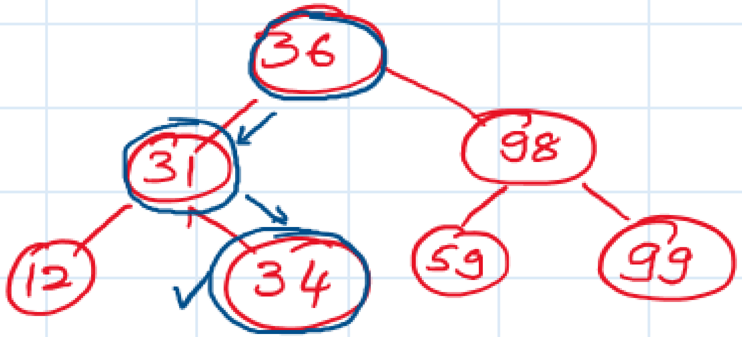
sibling(p) : RETURNS REF TO SIBLING NODE (OR NULL)

- MANY USES

DECISION TREES

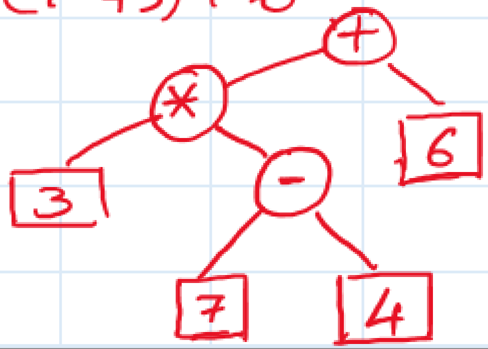


SEARCH TREES

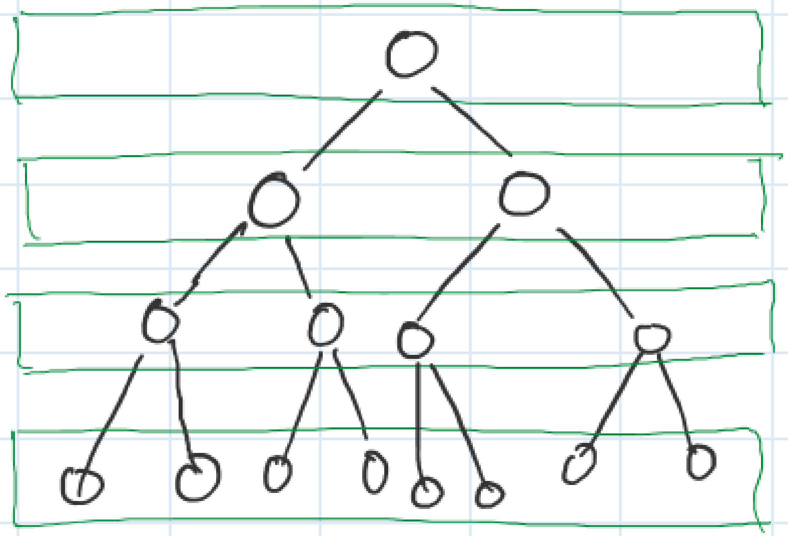


EXPRESSION TREES

$(3*(7-4))+6$



PROPERTIES OF BINARY TREES



	MAX # OF NODES
LEVEL 0	1
LEVEL 1	2
LEVEL 2	4
LEVEL 3	8
	⋮

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

$$1 \leq e \leq 2^h$$

$$e = i + 1$$

LET n = # of nodes

e = # of leaf nodes

i = # of internal -"-

h = the height of tree

$$h \geq \log_2 e$$

$$h \geq \log_2(n+1) - 1$$

$e = l + 1$ IN A PROPER BINARY TREE

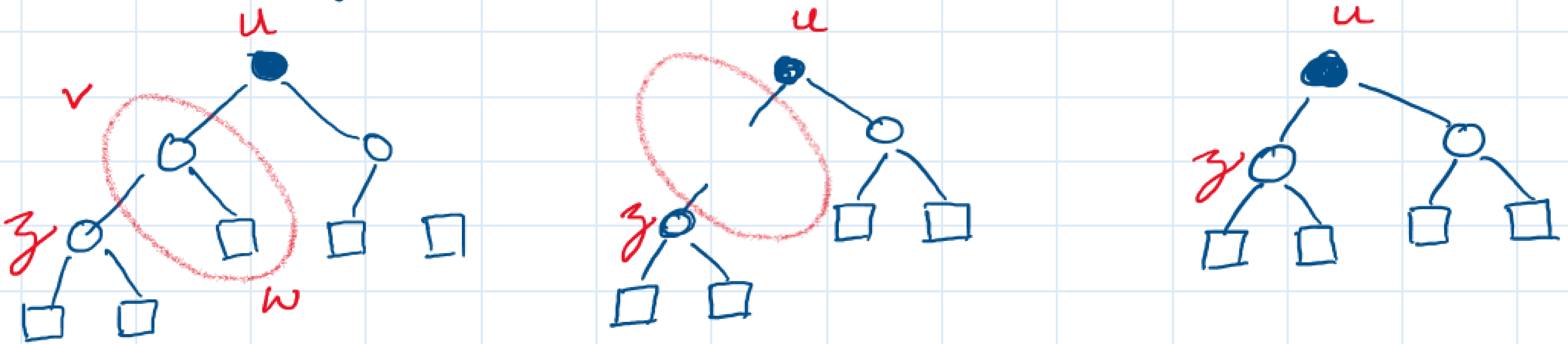
SPLIT TREE INTO TWO PILES
INTERNAL & EXTERNAL

CASE 1:

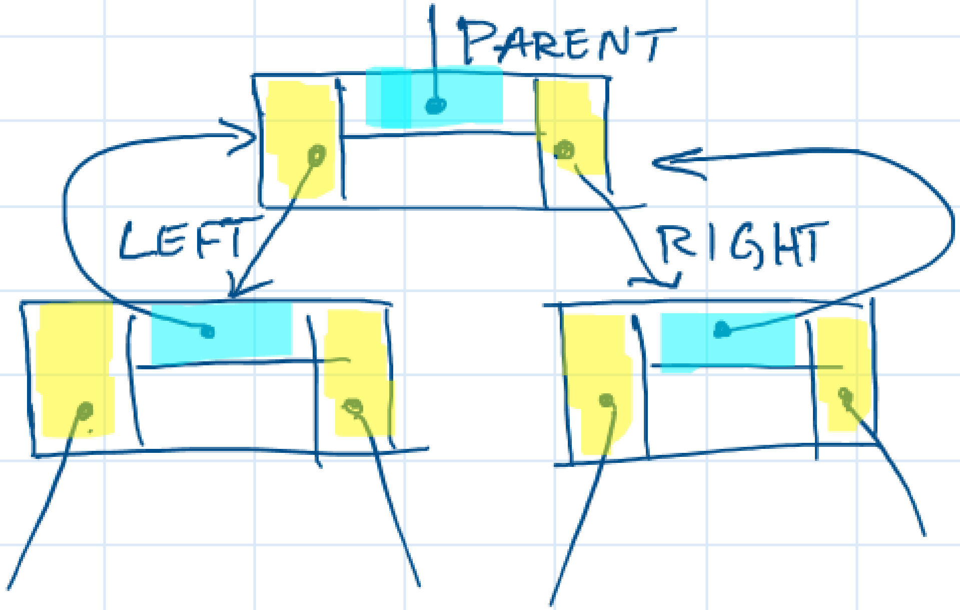
If tree T has only one node v
then put it in external pile

CASE 2:

Repeat the process of deleting a leaf node and its parent, reconnecting to get a full binary tree. At the end you are left with ...?



IMPLEMENTING BINARY TREES.



addRoot(e)
addLeft(p, e)
addRight(p, e)
set(p, e)
attach(p, T_1, T_2)
remove(p)