

SUMMARY OF ALGORITHMS

❑ ADT – Abstract Data Type

CLRS: 10

❑ 1-d : Lists, Queue, Stack

❑ Array implementation

❑ Execution Stacks

❑ Heap (aka Priority Queue)

CRLS: 16

❑ Binary Trees

See *appendix B.5 Trees*

❑ Traversals (pre-, in-, post-order)

❑ BST

CRLS: 12

❑ AVL and Red/Black

CRLS: 13

❑ Huffman Encoding

CRLS: 16.3

I-D ADT'S: ARRAYS, QUEUES, STACKS & LINKED LISTS.

- **Abstract Data Types (ADT):** data type (class) with ops (methods).
 - ◆ **Examples:** Int. (0,1,...,Maxint). All 2 by 2 real matrices. IEEE floats, etc.
 - ◆ **The implementation is not part of the ADT!**

- **Queue (or FIFO) is a list with methods:**
 - ◆ **Enqueue(item) & item = Dequeue** *(relative to *front/*back respectively)*

- **STACK (or LIFO) is a list with methods:**
 - ◆ **push(item) & item = pop()** *(relative to *TOP)*

- **Linked List is a list with methods:**
 - ◆ **insert(item) & delete()** *(relative to *current)*
 - ◆ *current->next and current->last moves current*

STACK IS FUNDAMENTAL

- Reverse Polish: 6 5 2 3 + 8 * + 3 + *

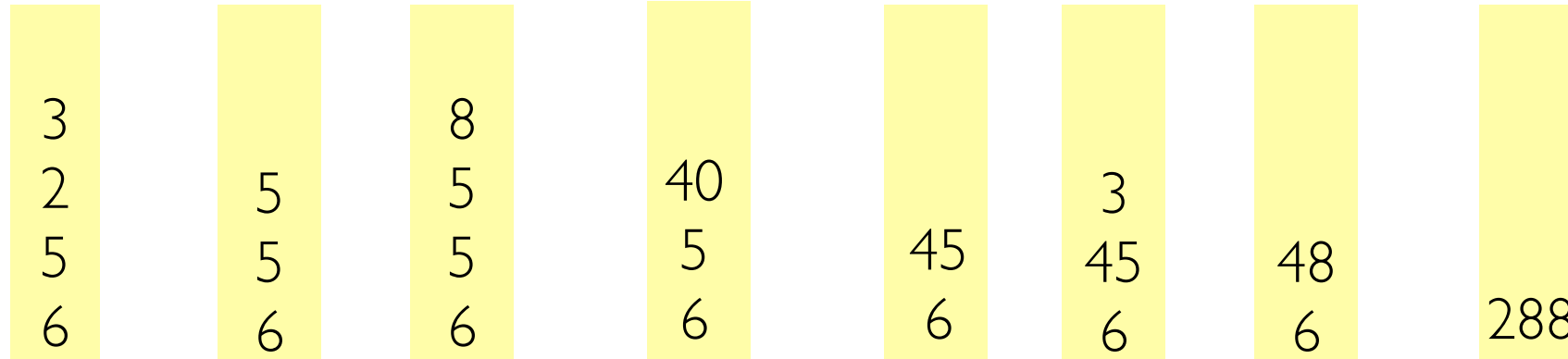
+

*

+

+

*



- See also conversion: infix → postfix

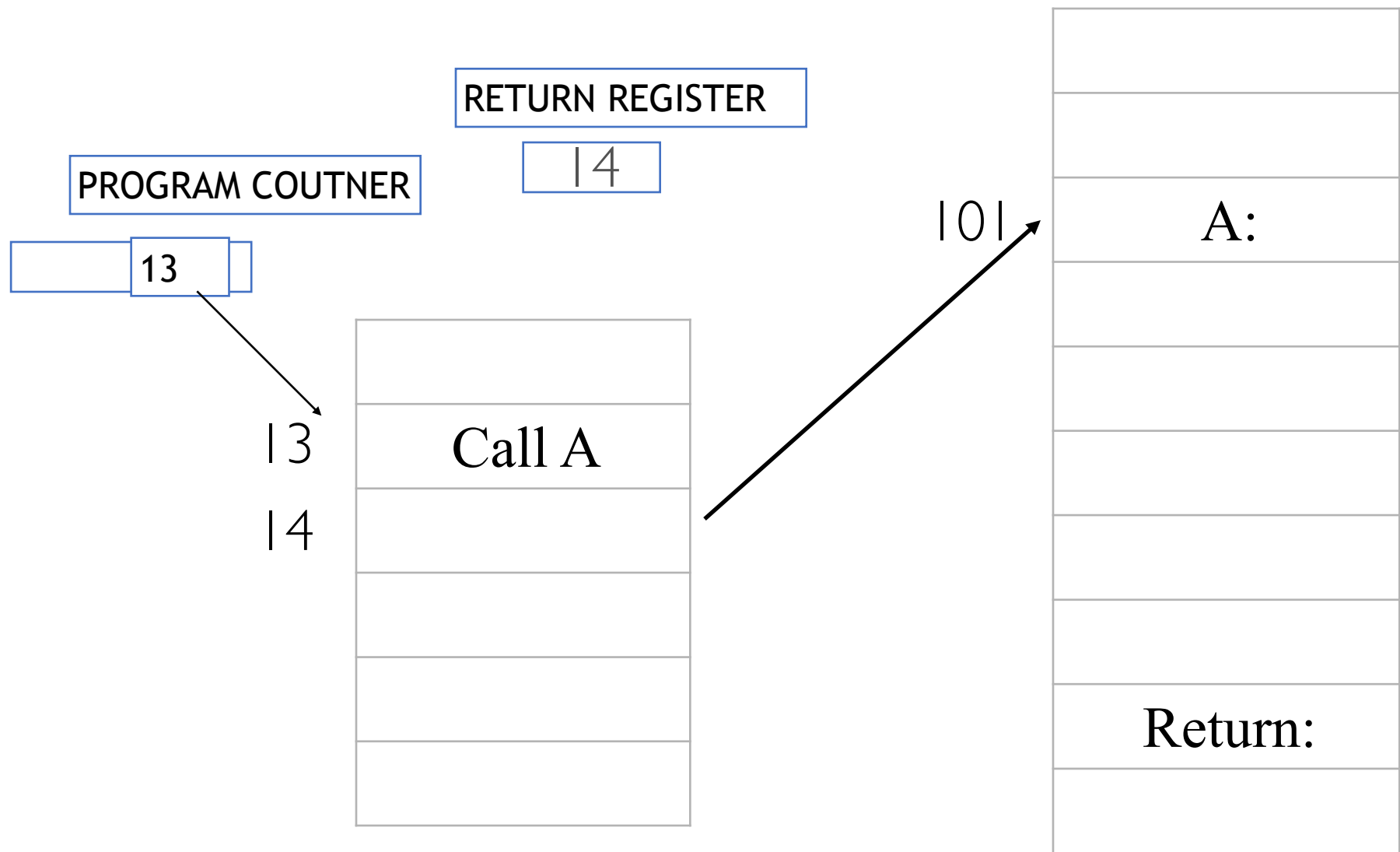
$$(6 * (5 + ((2+3)*8 + 3))) = 6 \ 5 \ 2 \ 3 \ + \ 8 \ * \ + \ 3 \ + \ *$$

- Execution Stacks for Function Calls:
 - Fixed return register
 - First line of subroutine (nested)
 - Execution stack (recursive)

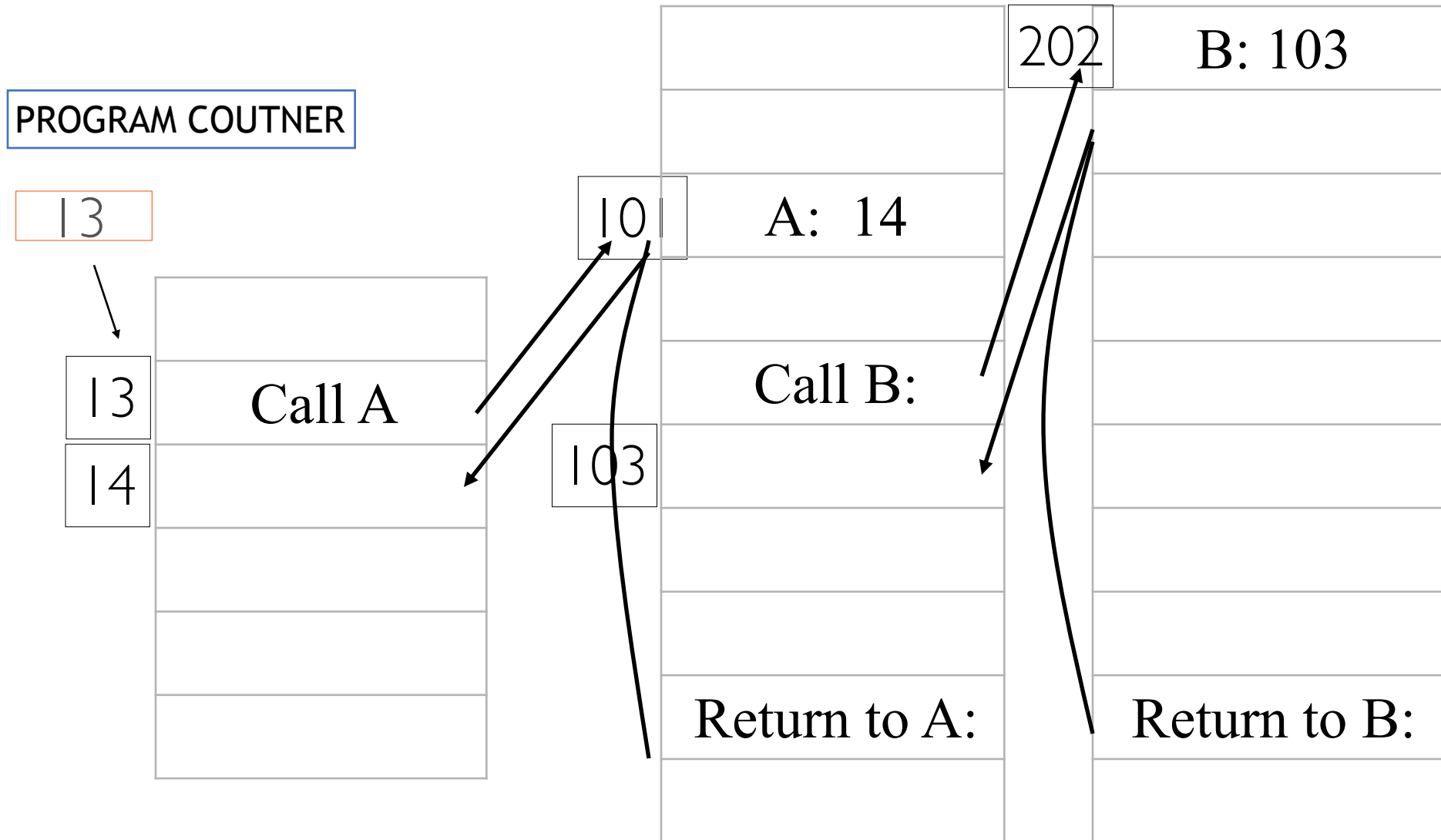
FORTRAN'S EVOLUTION OF THE SUBROUTINE CALL

- Function Call & Return
- Version 1 -- Return Register
 - ◆ no nesting
- Version 2 --- Return to top of Function
 - ◆ nesting but no recursion
- Version 3 --- The stack frame AT LAST!
 - ◆ Call yourself (recursion)

VERSION I



VERSION 2



VERSION 3

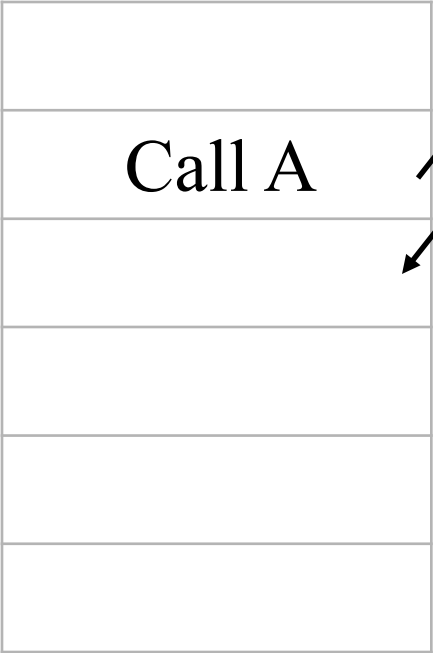
STACK

PROGRAM COUNTER

13

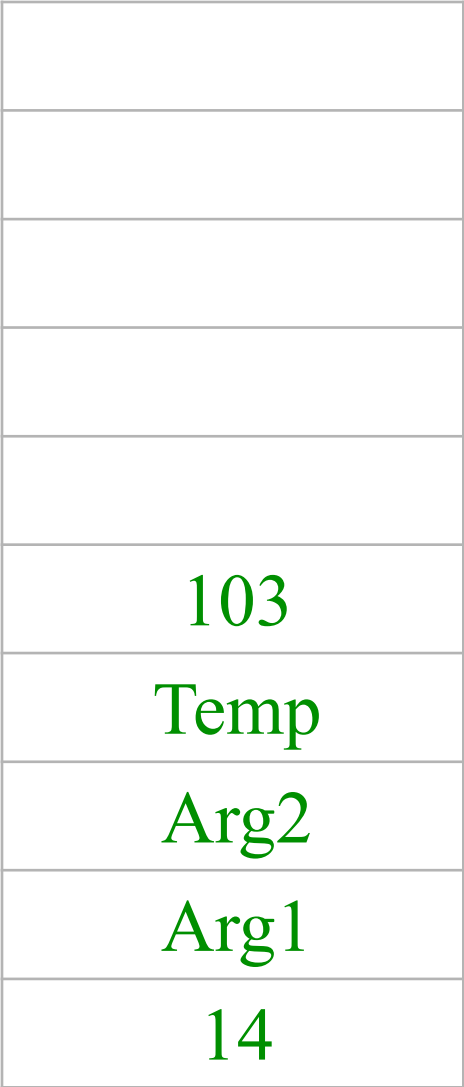
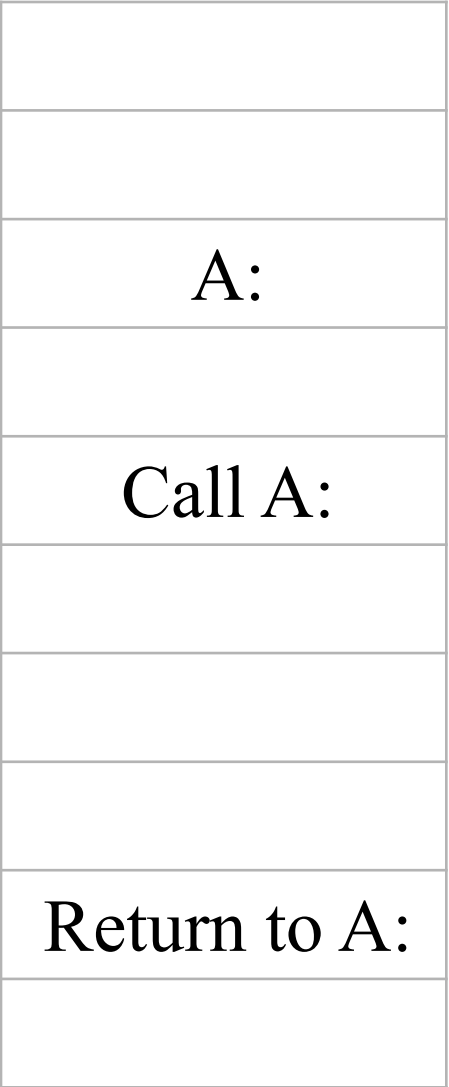
13

14



101

103



Priority Heaps

- Basic Heap ADT:

- ◆ *Data is $a[i]$, $i = 1, \dots, N$ (ROOT = 1, LABEL 0)*
- ◆ *Methods: Insert (key), Delete(key), DeleteMin, Build and Sort*

- Q: When is a tree an array? A: **complete** tree

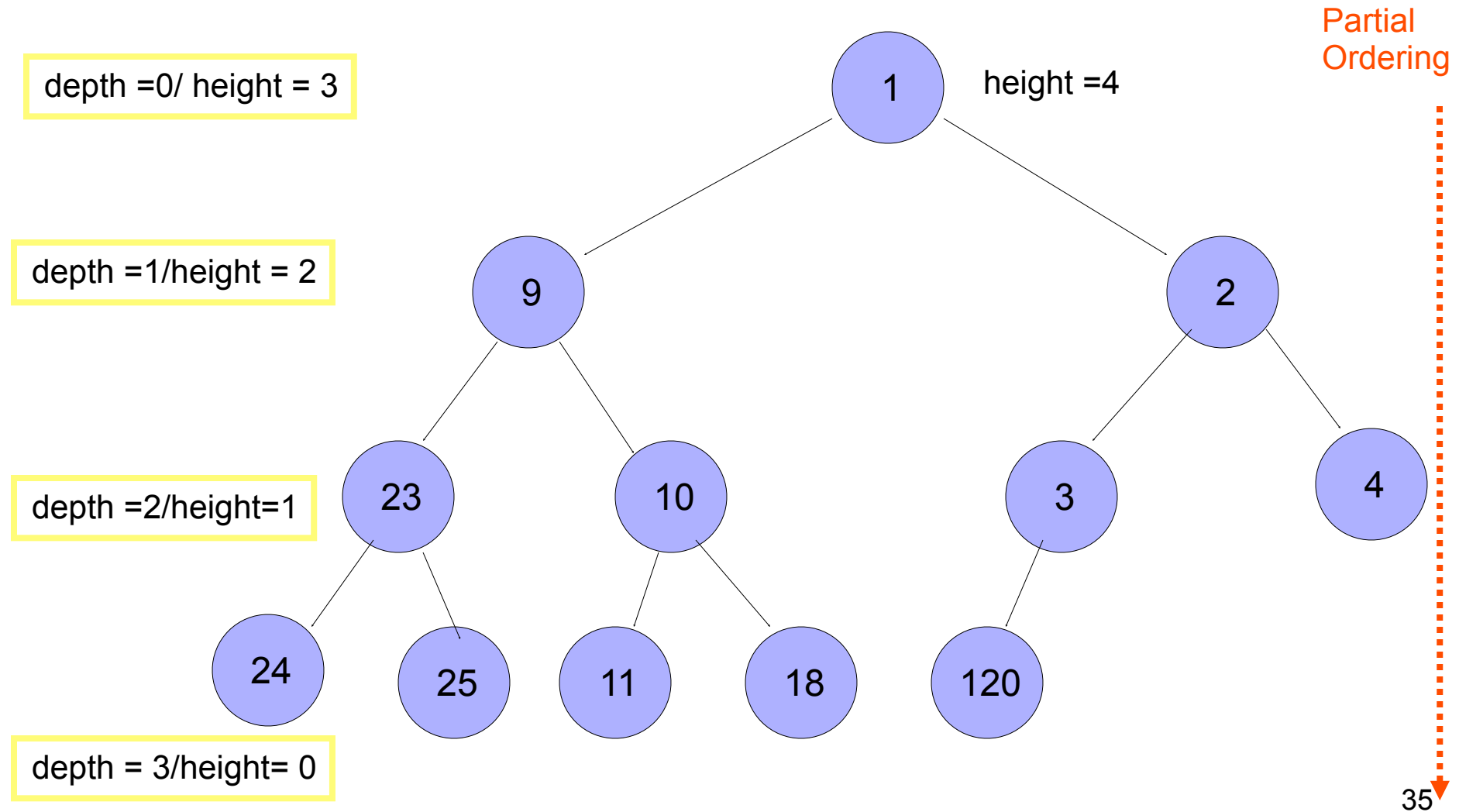
- ◆ *Parent $a[i] \rightarrow a[2i] = \text{left child} \ \& \ a[2i+1] = \text{right child}$*
- ◆ *Child $a[j]$: $\rightarrow a[j/2] = \text{parent}$ (integer division).*

- Build Heap is $O(N)$ by bottom up, DeleteMin is $O(\log(N))$



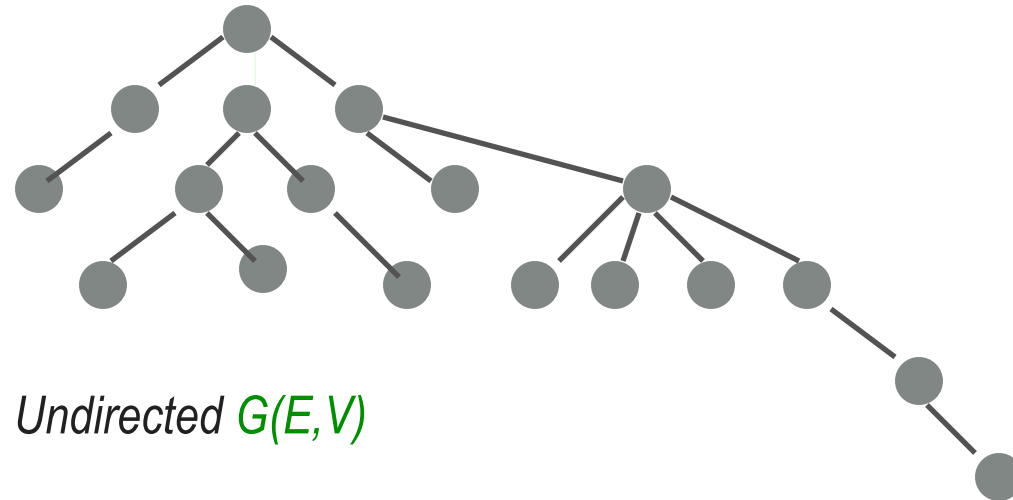
- ◆ *Heap sort by deleting min over and over is $O(N \log(N))$.*

Min Heap Order



INTRODUCTION TO TREES

- *Trees: inheritance, partial ordering, execution graphs,*
- *A tree is a special kind of Graph $G(E,V)$*
- *E = “edges/arcs” connecting V = “vertices/nodes”*



- *A tree is **Connected**, **Acyclic**, **Undirected** $G(E,V)$*
- *Binary Tree has 0,1,2 children (i.e. nodes have 1,2,3 edges)*

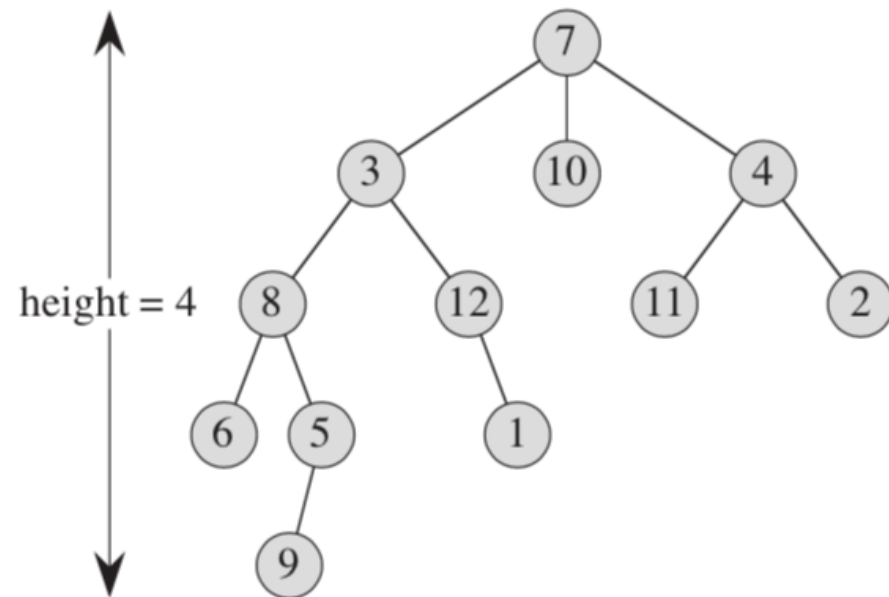
DEFINITIONS FOR BINARY TREES

- ◆ Full Tree: 0, 2 children,
- ◆ Complete Tree: Consecutive nodes (aka Heap),
- ◆ Perfect Tree: Complete and full last row.
- Full Tree Theorem: # of leaves: $L(N) = (N+1)/2$ for N nodes
- Perfect Tree with H levels (height or depth)
- Nodes in Perfect k -way tree : $N(H) = (k^{H+1}-1)/(k-1) \Rightarrow 2^{H+1} - 1$
- Execution Tree
- Traversals: in-, pre-, post-order.

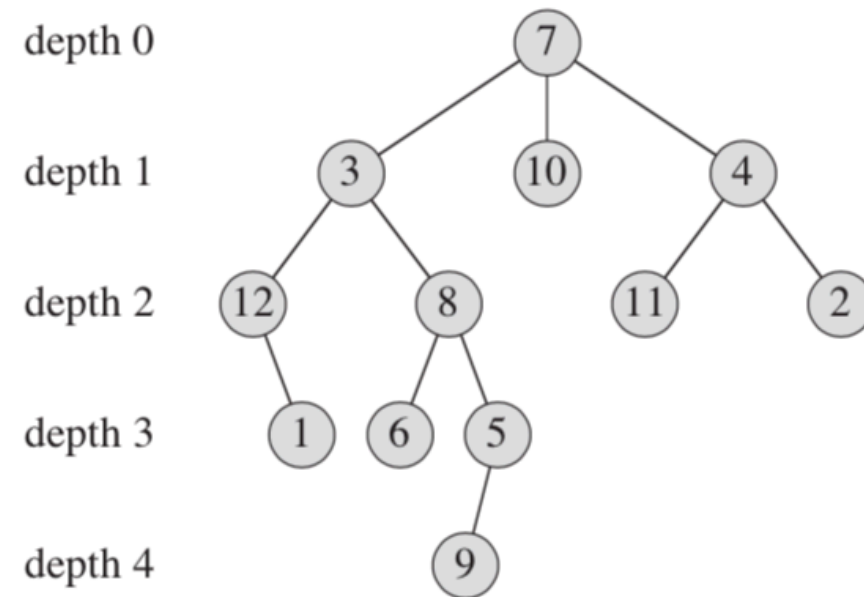
Height vs Depth of “nodes”

B.5 Trees

1177



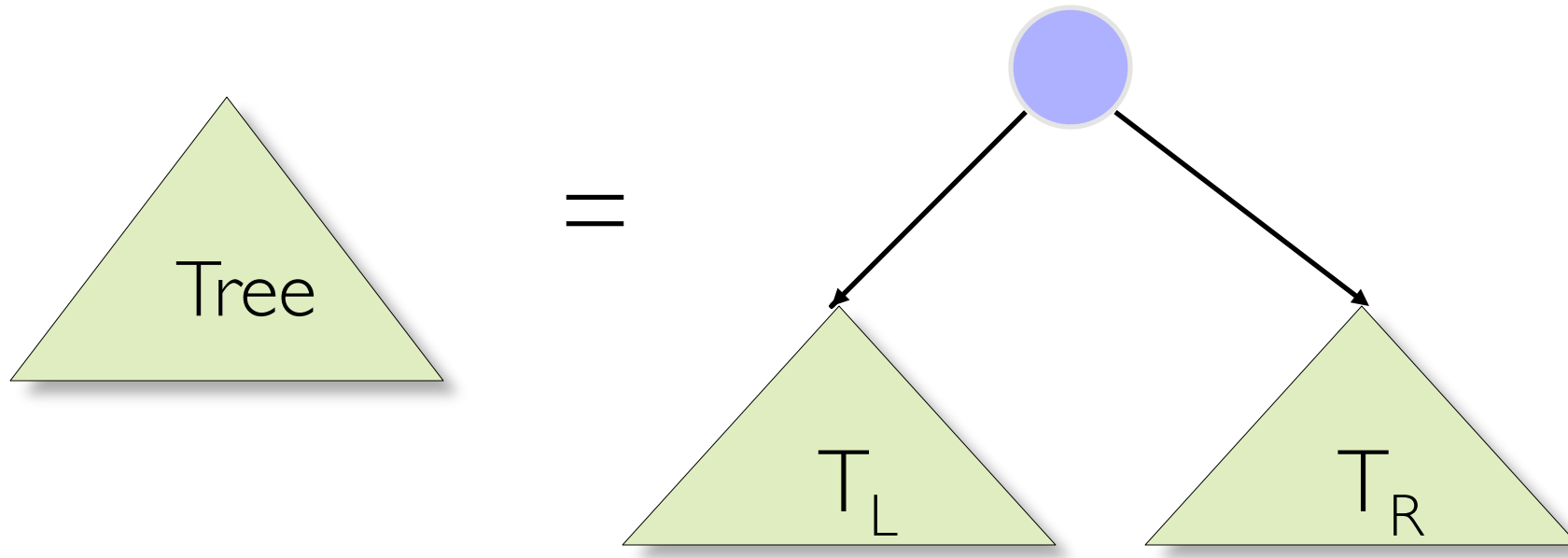
(a)



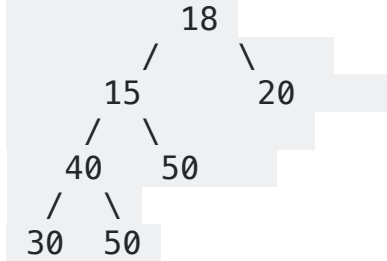
(b)

BINARY TREE: RECURSIVE DEFINITION

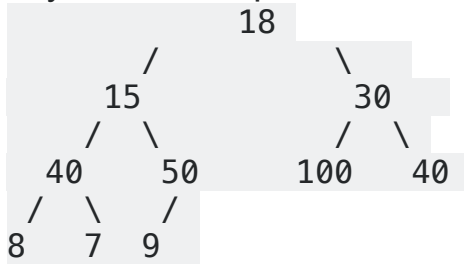
- A binary tree is null or a single node with a Right and Left Child that is a binary tree!
(Useful for organizing recursive algorithms on binary trees.)



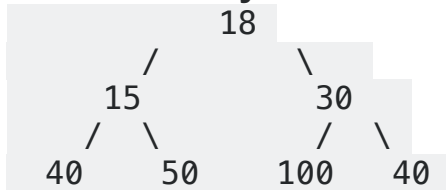
Full Binary Tree: A Binary Tree is full if every node has 0 or 2 children. Following are examples of a full binary tree.



Complete Binary Tree: A Binary Tree is complete Binary Tree if all levels are completely filled except possibly the last level and the last level has as many nodes as possible.



Perfect Binary Tree: A Binary tree is Perfect Binary Tree in which all internal nodes have two children and all leaves are at same level.

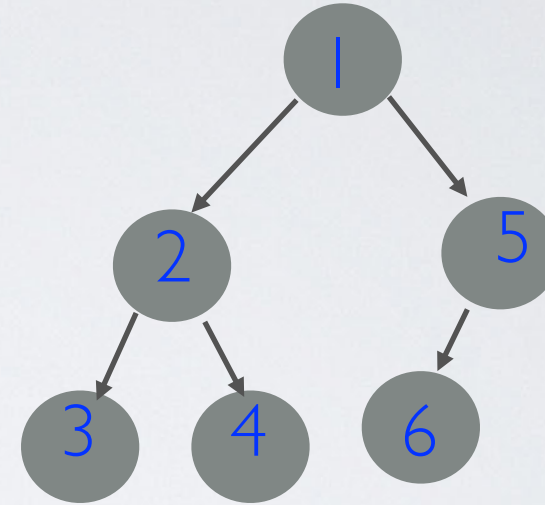


TREE TRAVERSALS

■ Preorder: Print [Tree]{
 Print root;
 Print Tree[LeftTree];
 Print Tree:[RightTree];
 }

■ Inorder: Print [Tree]{
 Print Tree[LeftTree];
 Print root;
 Print Tree:[RightTree]
 }

■ Postorder: Print [Tree]{
 Print Tree[LeftTree];
 Print Tree:[RightTree]
 Print root;
 }

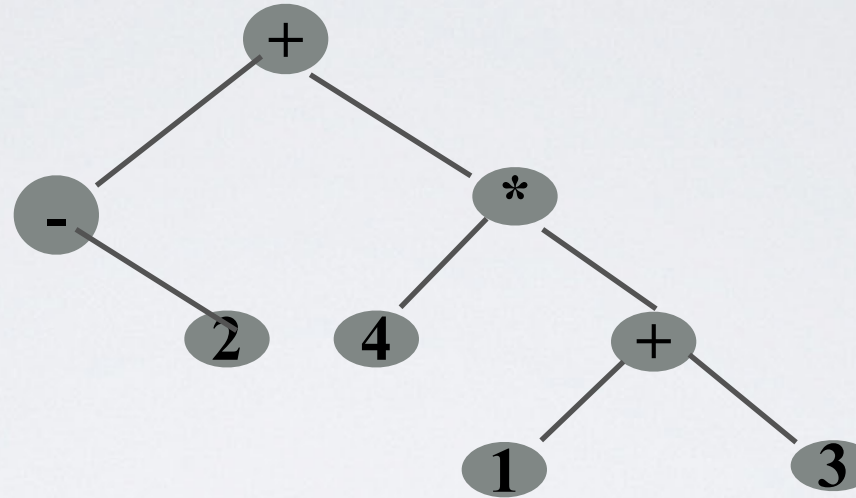


Pre: 1→2→3→4→5→6

In: 3→2→4→1→6→5 sort on BST

Post: 3→4→2→6→5→1

Expression Trees



Preorder: + - 2 * 4 + 1 3 (Lisp, Scheme) (+ (- 2) (* 4 (+ 1 3)))

In order: -2 + 4 * (1 + 3) (C, C++, Java) Standard precedence

Postorder: 2 - 4 1 3 + * + (HP calculator, PS, Forth)

Binary Search Tree: left \leq root < right

- in order traversal gives sorted list
- easy to search

see https://en.wikipedia.org/wiki/Binary_expression_tree

DIMENSIONS OF A PERFECT TREE

- Perfect Tree (all levels filled) with H levels:

(Height: $H = \log_k(N)$ for k -array tree)

- # nodes: $N(H) = 1 + k + k^2 + k^3 + \dots + k^H = (k^{H+1}-1)/(k-1)$

(binary tree: $N = 1 + 2 + 2^2 + 2^3 + \dots + 2^H = 2^{H+1} - 1$)

- total Depth: $T_D(N) = k \, dN/dk = (H+1)k^{H+1}/(k-1) - k(k^{H+1}-1)/(k-1)^2$

→ (binary tree) $2(H+1)2^H - 2(2^{H+1}-1) = (H-1)N + H + 1$

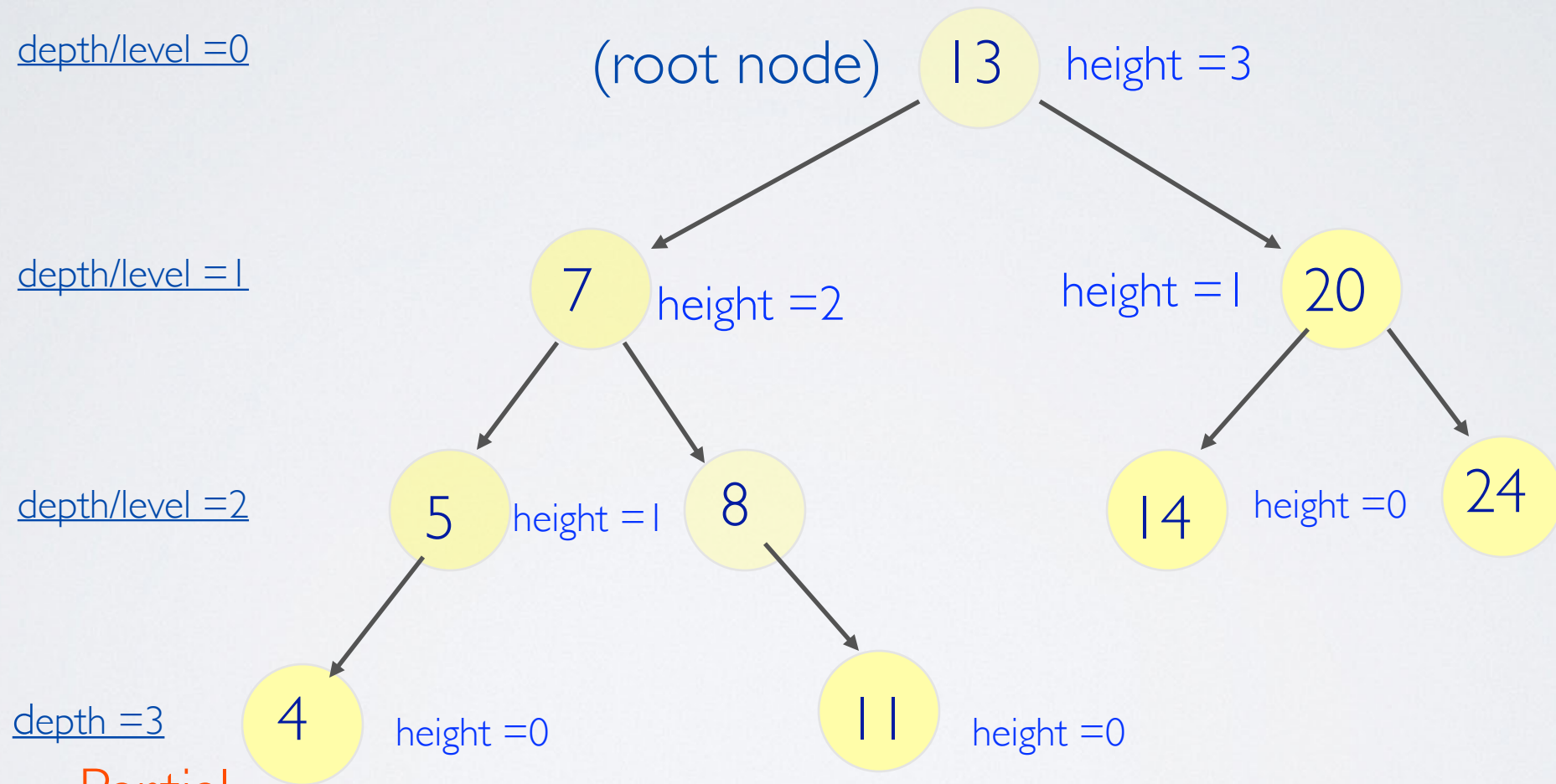
- total Height: $T_H(N) + T_D(N) = H N$ (each $h + d = H$)

$$T_H = H N - T_D = N - H - 1 \quad (\text{binary tree})$$

SEARCH TREES

- *BST tree Recursive definition*
 - ◆ *Insertion and Deletion*
- *AVL tree balance:*
 - ◆ *Insertions: single (zig-zig) and double (zig-zag) rotations.*
 - ◆ *Lazy Deletion*
- *Red/Black Tree*

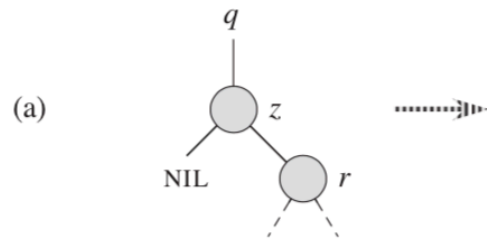
BINARY SEARCH TREES



Partial
Ordering

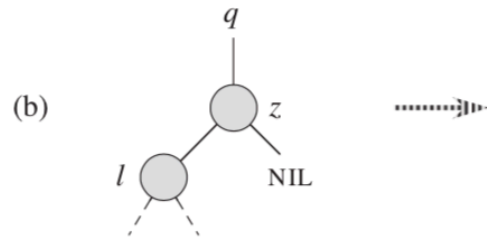
BINARY SEARCH TREE: BST

1. BST is a Binary Tree with keys stored in each node.
2. The key (K_0) in each node is: greater or equal to all keys in T_L , the Left subtree ($K_{\text{left}} \leq K_0$) less than all keys in T_R , the Right subtree ($K_0 < K_{\text{Right}}$)
3. The BST defines a partial ordered set --- as you move down to the left/right the keys decrease/increase.
4. Insert new K_{new} push down to subtree Left/Right if $K_{\text{new}} \leq / > K_0$.
5. Delete K_0 and replace by SMALLEST key in T_R , the Right subtree.

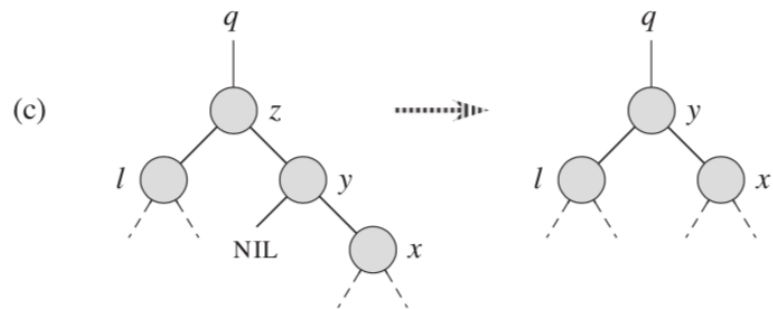


Right Only Child

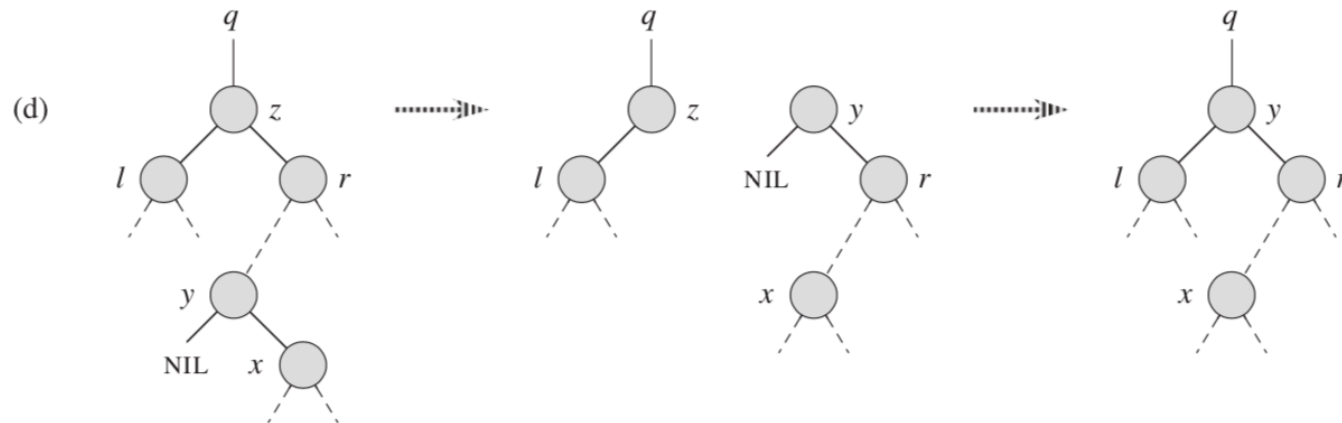
BST DELETE



Left Only Child



R & L Twins



Right to Left Rotation

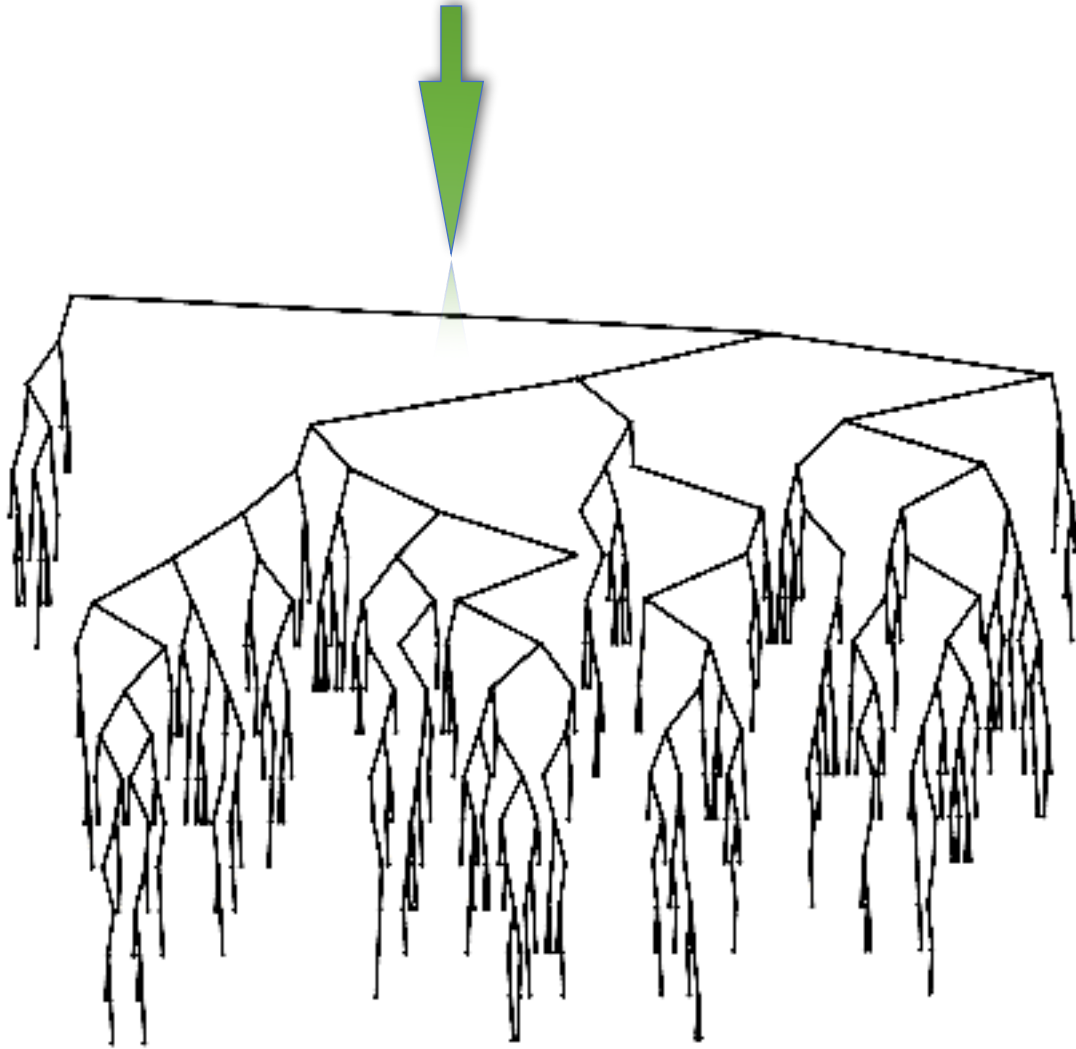
AVERAGE TOTAL DEPTH OF BST

$$\begin{aligned} \blacksquare T_D(N) &= \frac{2}{N}[T_D(0) + T_D(1) + T_D(2) + \cdots + T_D(N-1)] + c(N-1) \\ T_D(x) &\simeq \frac{2}{x} \int_0^x T_D(x) + c(x-1) \\ xT_D(x) &\simeq 2 \int_0^x T_D(x) + c(x^2 - x) \\ &\Rightarrow T_D(x) + x \frac{dT_D(x)}{dx} = 2T_D(x) + c(2x-1) \\ \frac{dT_D(x)}{dx} &\simeq T_D(x)/x + 2c \\ &\Rightarrow T_D(x) = 2cx \log(x) \end{aligned}$$

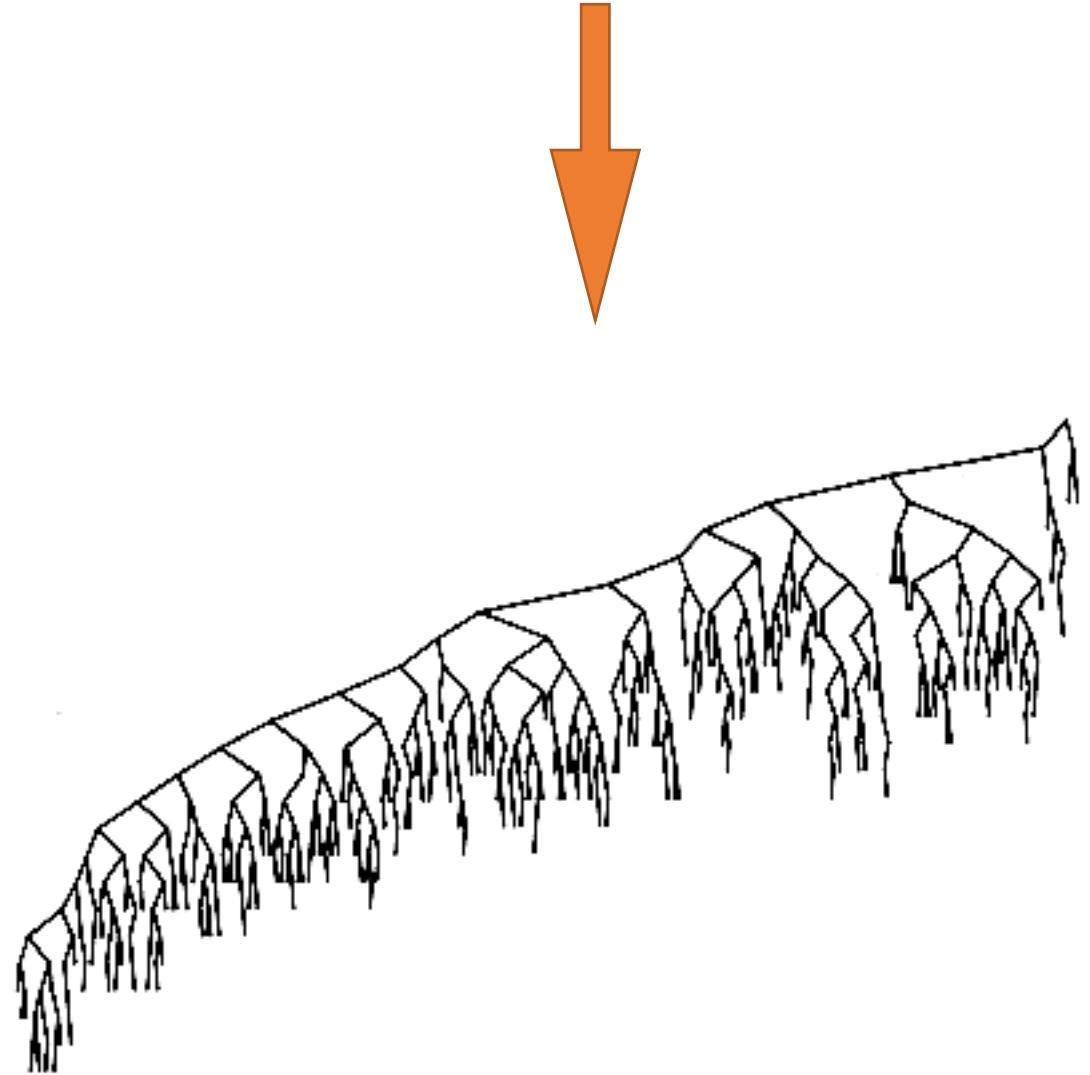
SAME AS QUICK SORT!

◆ Solution: $T_D(N) = \Theta(N \log(N))$

Average BST vs After $O(N^2)$ insert/delete



Depth $O(\log N)$



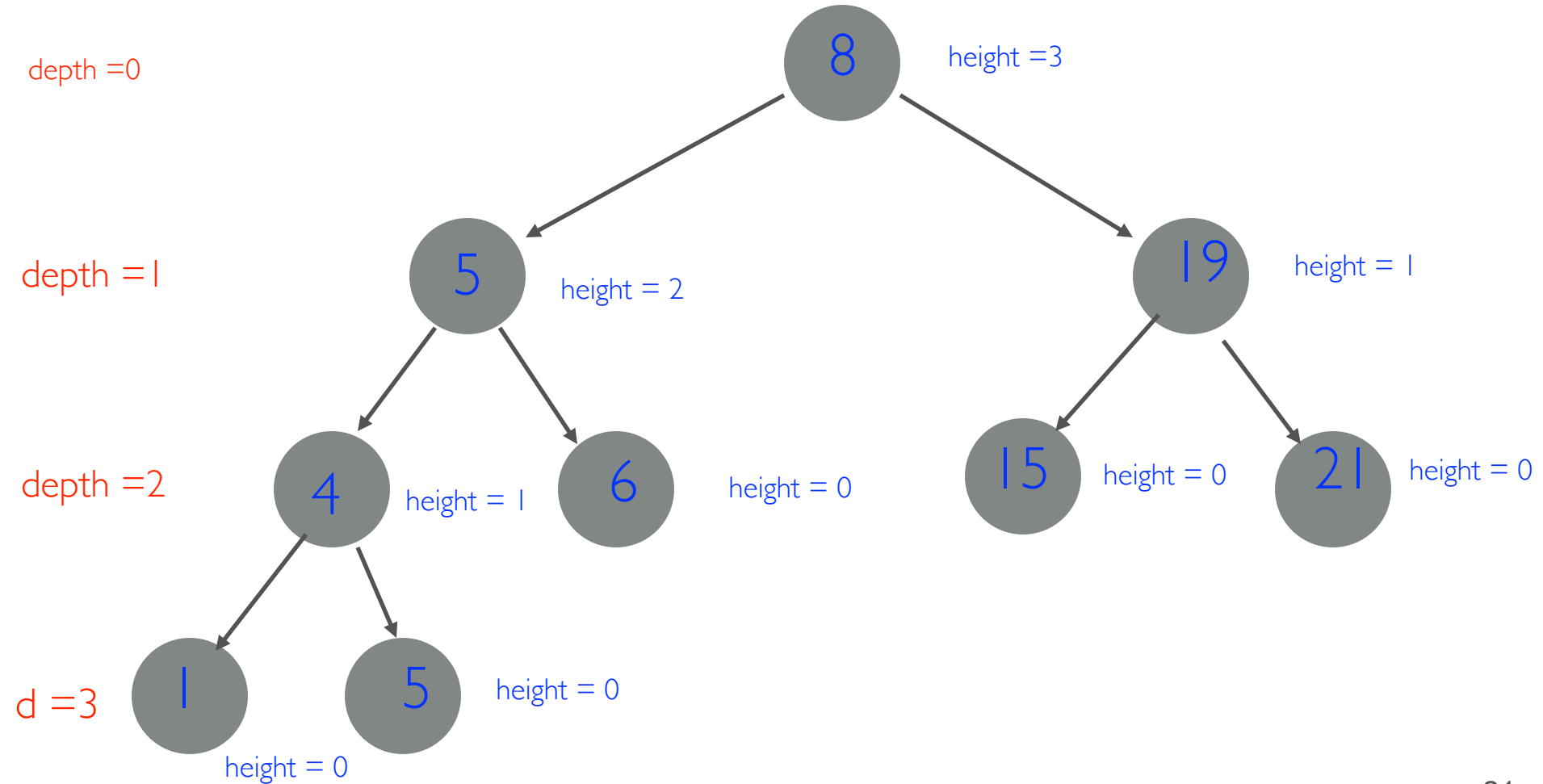
Depth $O(N)$

RELATIONS: BOOLEAN VALUED MATRIX $R[A,B]$

- Set: $S = \{a,b,c,\dots\}$
- Relation $(a,b) \in S \times S$: $a R b$ is True?
- Properties:
 - ◆ Reflexive: $a R a$ is True
 - ◆ Anti-symmetric: $a R b$ and $b R a \rightarrow a = b$
 - ◆ Transitive: $a R b$ and $b R c \rightarrow a R c$
 - ◆ Total Ordering: $a R b$ or $b R a$ (inclusive or)
 - ◆ Self dual: $a R b \leftrightarrow b R a$
 - ◆ Transpose: $a R b \leftrightarrow b R^T a$
- RAT is partial ordering: e.g. descendants in a tree!

(e.g. \leq is total ordering for int but $g(N) = O(f(N))$ is partial ordering!)

AVL: BST WITH $|H_L - H_R| = 0, 1$



FIBONACCI SERIES & RABBITS

- 2 rabbit beget 2 rabbit every month after one month!
 - ◆ $F_k = (F_{k-1} - F_{k-2}) + 2 F_{k-2} = F_{k-1} + F_{k-2}$
 - ◆ with $F_0 = F_1 = 1$
- 1, 1, 2, 3, 5, 8, 13, 21, Fibonacci: $F_k = F_{k-1} + F_{k-2}$
- Bad Recursion: $\Omega(C^N)$ with $C = 1.61...$
 - ◆ Exponential $T(N) = T(N-1) + T(N-2) + 1$
 - ◆ $T(N) > F_N > (2)^{N/2}$
- Iteration: $\Theta(N)$
- Math $\Theta(1)$! Try $F_k = x^k$ find two homogeneous solutions to characteristic equation.

$$F_k = \frac{1}{\sqrt{5}} \left[\left(\frac{1 + \sqrt{5}}{2} \right)^{k+1} - \left(\frac{1 - \sqrt{5}}{2} \right)^{k+1} \right]$$

$$\frac{1 + \sqrt{5}}{2} \simeq 1.61803$$

WORST CASE HEIGHT $H(N)$ FOR AVL

- Minimum # of Nodes (see Fig 4.33):

$$N(H) = N(H-1) + N(H-2) + 1 > N(H-1) + N(H-2)$$

- Almost Fibonacci: $F_k = F_{k-1} + F_{k-2}$

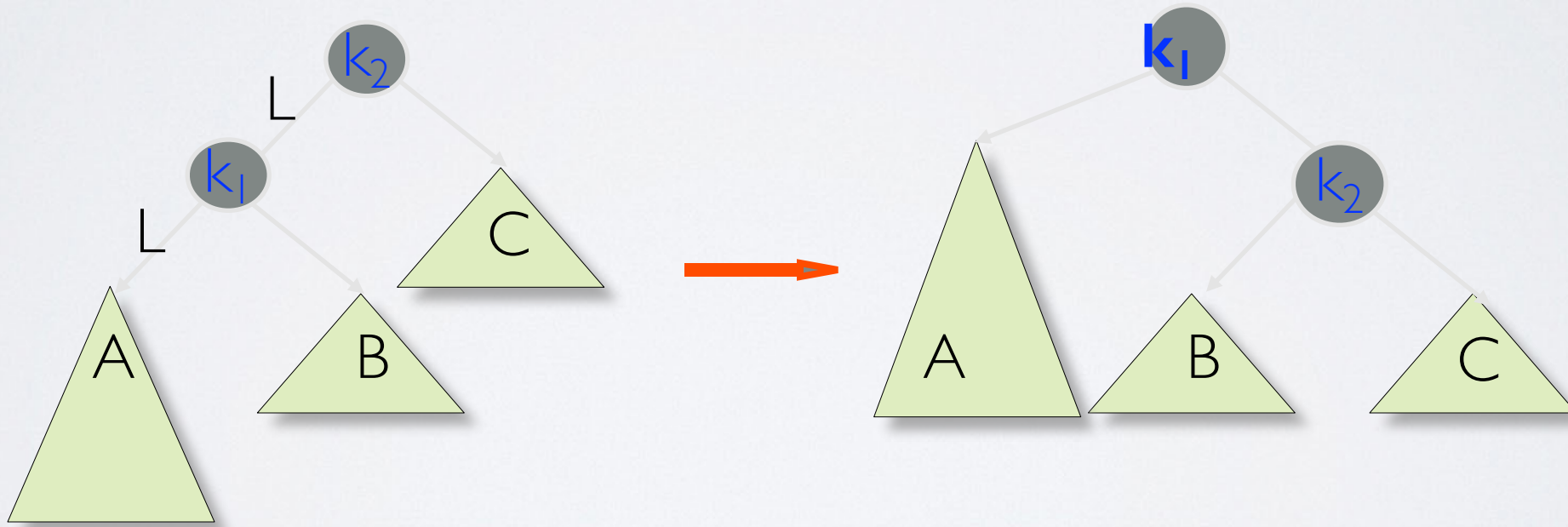
◆ So $N(H) > F_H \sim c^H$ with $c = (1 + 5^{1/2})/2 = 1.618034$

◆ Or $H < \log(N)/\log(c) \quad 1.440420 \log_2(N) = 2.078 \ln(N)$
 $= 4.784 \log_{10}(N)$

(Better estimate: $H = 1.44 \log_2(N+2) - 0.328$

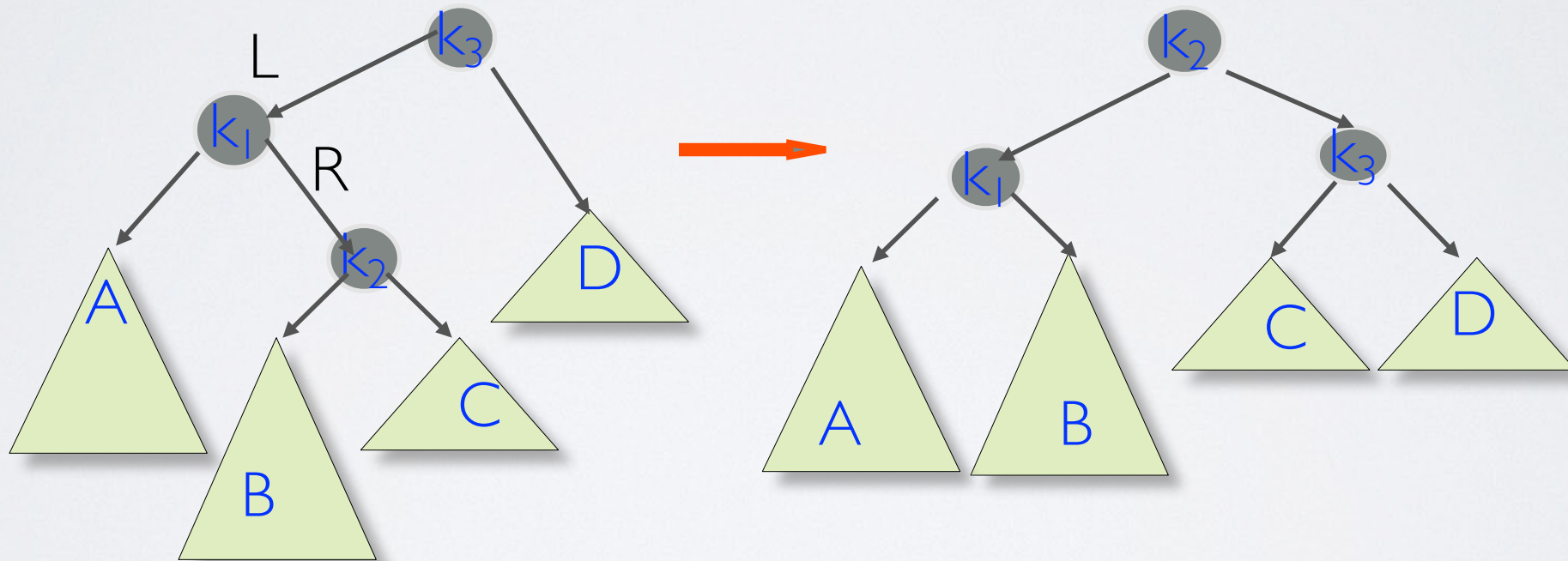
ZIG-ZIG INSERTION FOR LL OR RR:

- Insert New Key along path going **L**eft and **L**eft again into A:
- This cause violation of AVL balance.
- k_2 is lowest node failing AVL balance.
- Single rotation of $k_1 \rightarrow k_2$ restores AVL balance



ZIG-ZAG INSERTION FOR LR

- Insert New Key along path going *Left* and then *Right* into B:
- This cause violation of AVL balance.
- k_3 is lowest node failing AVL balance.
- Double rotation of $k_1 \rightarrow k_2 \rightarrow k_3$ restores AVL balance



HUFFMAN CODING

- ❑ Place all letters at leaves of a binary tree
 - ❑ The code is path (i.e. address) of each leaf.
 - ❑ Binary code for each letter: e.g. "a" = 01001, "b" = 101, ..

$$\text{ext. depth} = \sum_i w_i d_i, \quad \text{average code length} = \frac{\sum_i w_i d_i}{\sum_i w_i} = \sum_i p_i d_i$$

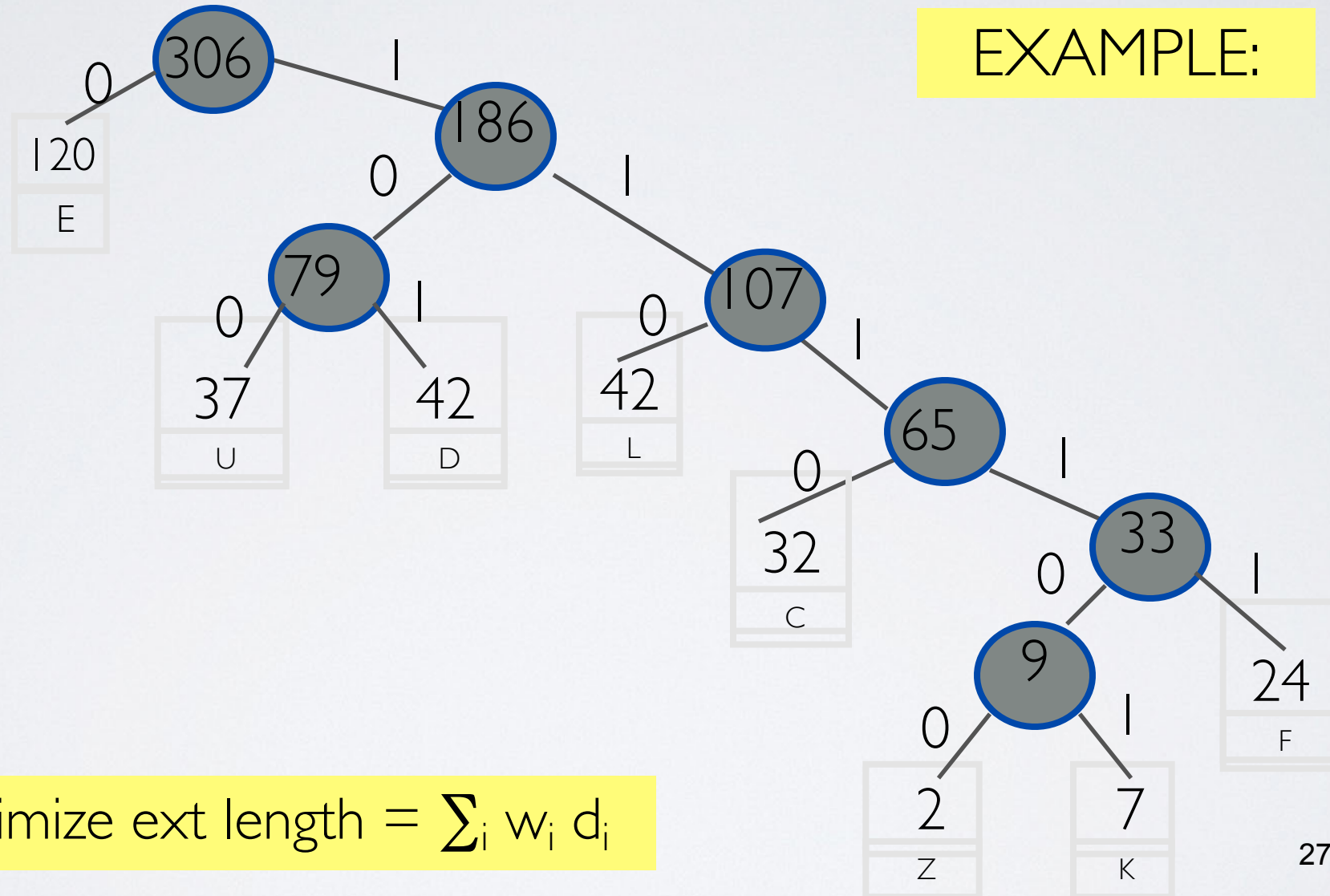
Build the Huffman tree:

- ❑ Sort symbol list: $w_1 < w_2 < \dots < w_N$
- ❑ Remove w_1 and w_2 and place as left and right children of parent $w_{(12)}$
- ❑ Place $w_{(12)} = w_1 + w_2$ in symbol list and Repeat

W_i →

2	7	24	32	37	42	42	120
Z	K	F	C	U	D	L	E

EXAMPLE:



Minimize ext length = $\sum_i w_i d_i$

RESULTING CODE: $AVERAGE\ BITS/CHAR = 785/306 = 2.565$

	Letter	Weight	Code	Bits	Count
■	C	32	1110	4	128
■	D	42	101	3	126
■	E	120	0	1	120
■	F	24	11111	5	120
■	K	7	111101	6	42
■	L	42	110	3	126
■	U	37	100	3	111
■	Z	2	111100	6	12
Total:					306
					785

PROOF BY INDUCTION

- *Base case $N=2$ has minimum with $d_1 = d_2 = 1$*
- *Two smallest weights w_1 & w_2 are at max depth*
- *(If not swap with any other is smaller: See identity next)*
 - ◆ *Can swap to give same parent $w_{12} = w_1 + w_2$*
- *Hence prove for N :*
- *$\text{Min}[(d_{12} + 1) (w_1 + w_2) + w_3 d_3 + \dots + w_N d_N]$ over all trees T*
$$= (w_1 + w_2) + \text{Min}[d_{12} w_{12} + w_3 d_3 + \dots + w_N d_N]$$

“SCHWARTZ” PARING INEQUALITY!

Need for Huffman and Many Opt Algorithms

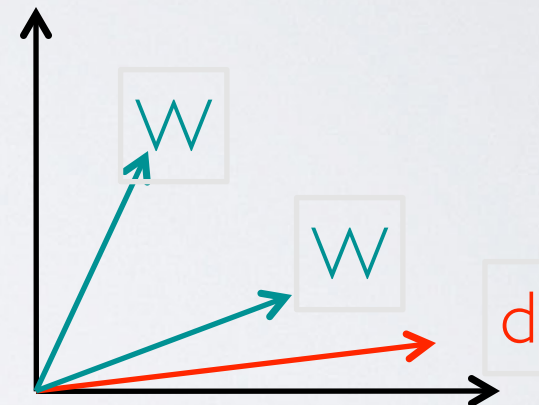
Prove: (more parallel is larger!)

$$w_S d_S + w_L d_L > w_L d_S + w_S d_L$$

because $(w_L - w_S)(d_L - d_S) > 0$

scalar product is larger

when w and d are more nearly parallel!



MORE OPTIMIZATION

- Object Function and elementary move

- Sorting $S = \text{MIN}_{\pi} \sum_i I * a[\pi(I)]$

- swap minimize $|a[I] - a[J]|$ if out of order

- Continuum vs Discrete:

- Bisection : $\text{Log}(N)$ vs error $\Rightarrow \text{error}/2$

- Find zero: $f(x) = 0$ or (continuous)

- Find key $f[I] = (a[I] - \text{key})^2 = 0$ ($a[I]$ sorted)

- Newton's, Secant, Regula falsi

- & Dictionary method (linear extrapolation)

- $\text{Log}(\text{Log}(N))$ vs error $\Rightarrow (\text{error})^\phi$

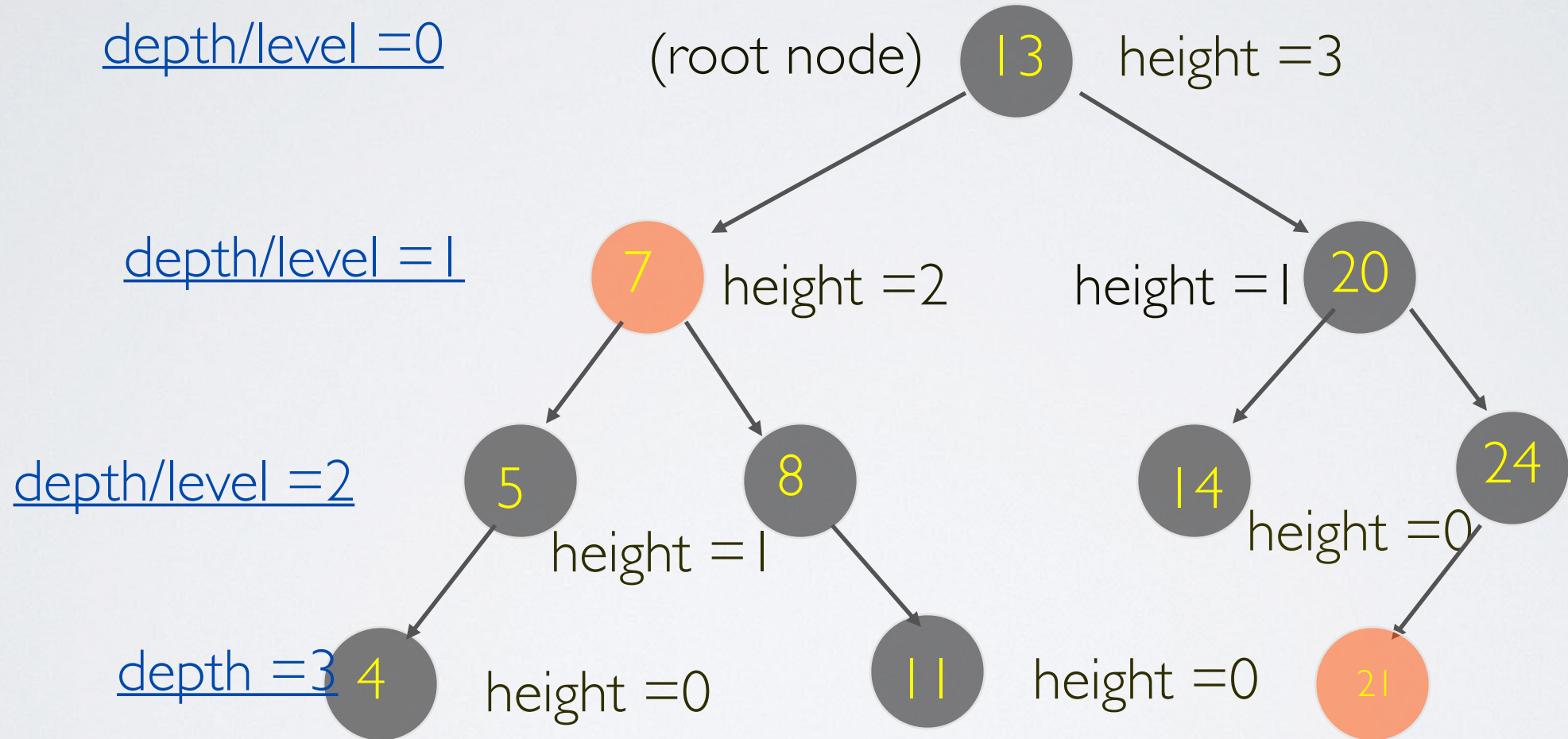
ϕ is 2 for Newton and the golden ratio 1.618 for secant.

RED-BLACK TREES

1. *Every node is red or black*
2. *The root is black*
3. *Every nil node (leaf) is black*
4. *A red node must have black children*
5. *Every path to a nil pointer must have same number of black nodes*

Insertion: Place as new leaf in BST must color red (due #5) If parent is red it violates #4. Must do rotations to fix it!

RED-BLACK TREES



Partial Ordering →