SUMMARY OF ALGORITHMS

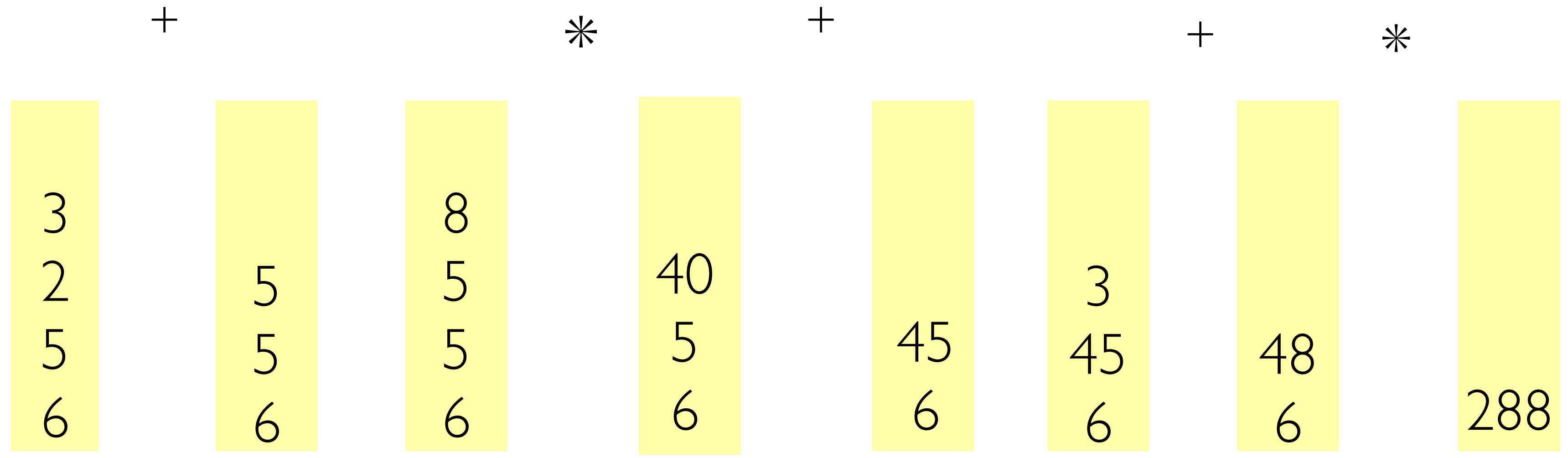
- ADT – Abstract Data Type **CLRS: 10**
  - 1-d : Lists, Queue, Stack
  - Array implementation
  - Execution Stacks
- Heap (aka Priority Queue) **CRLS: 16**
- Binary Trees **See appendix B.5 Trees)**
  - Traversals (pre-, in-, post-order)
  - BST **CRLS: 12**
  - AVL **CRLS: 13**
  - Huffman Encoding **CRLS: 16.3**

1-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 convertion: infix ➔ postfix

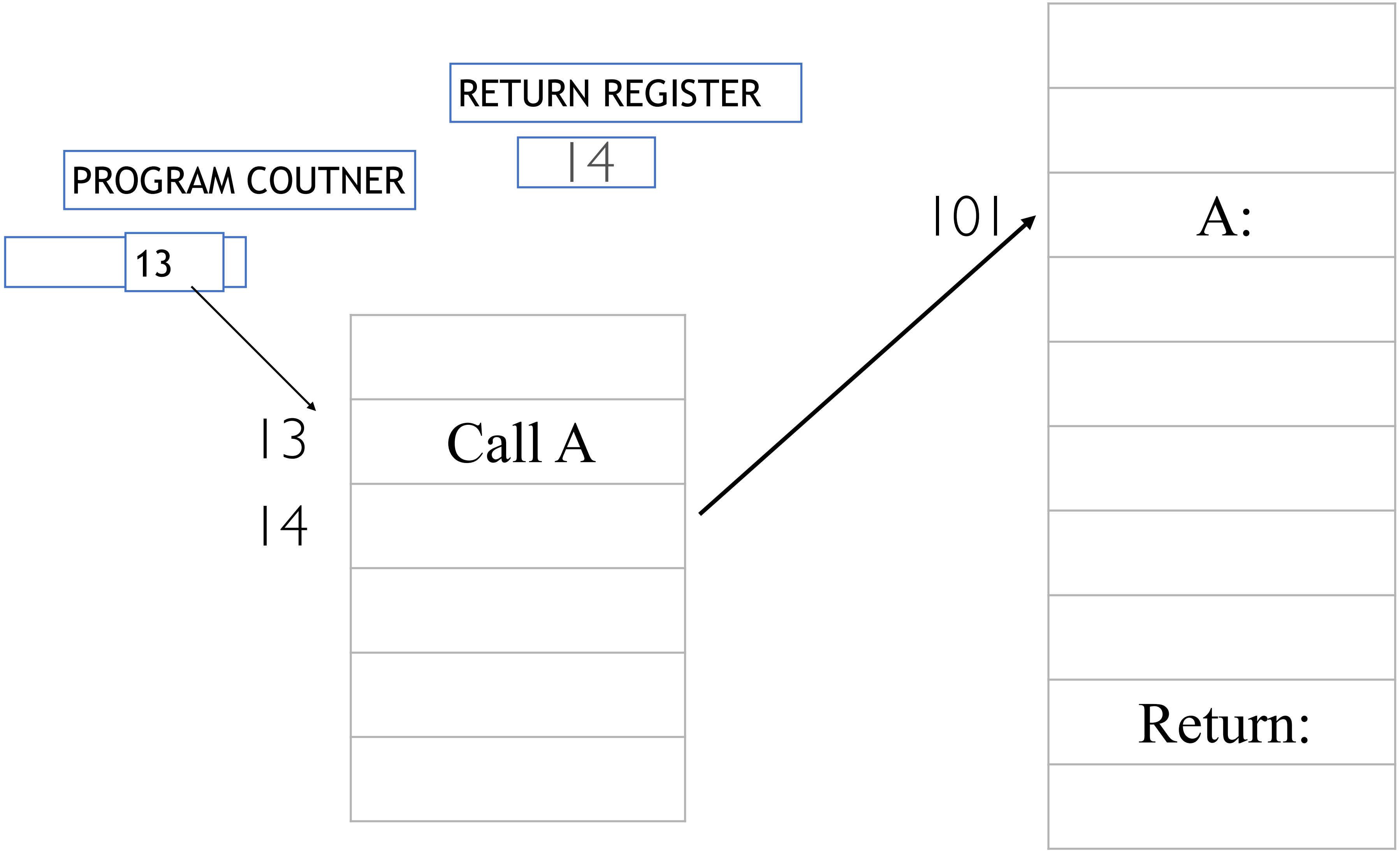
(6 * (5 + ((2+3)*8 + 3))) = 6 5 2 3 + 8 * + 3 + *

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

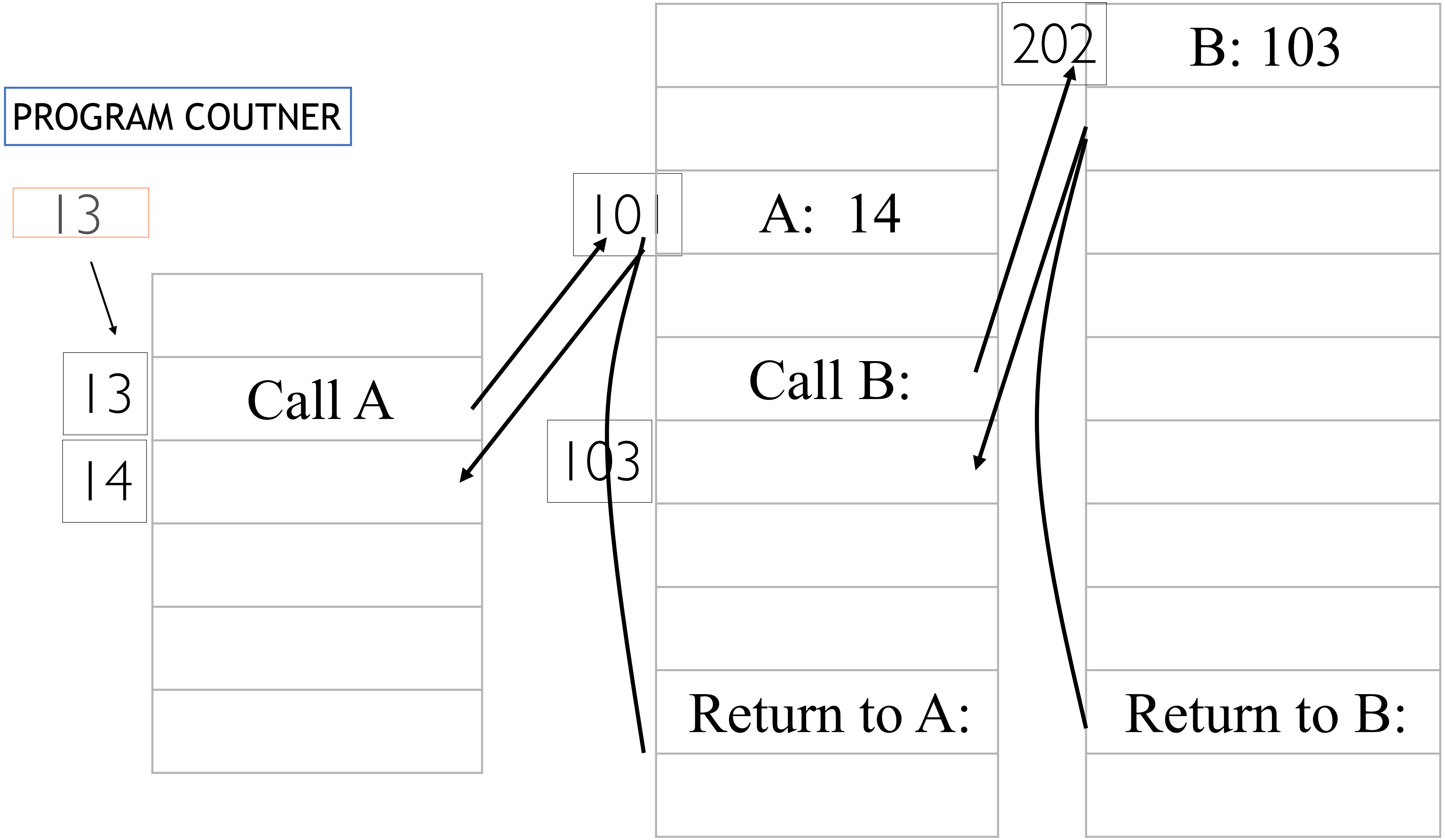
FORTRAN'S EVOLUTION OF THE SUBROUTINE CALL

- 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)

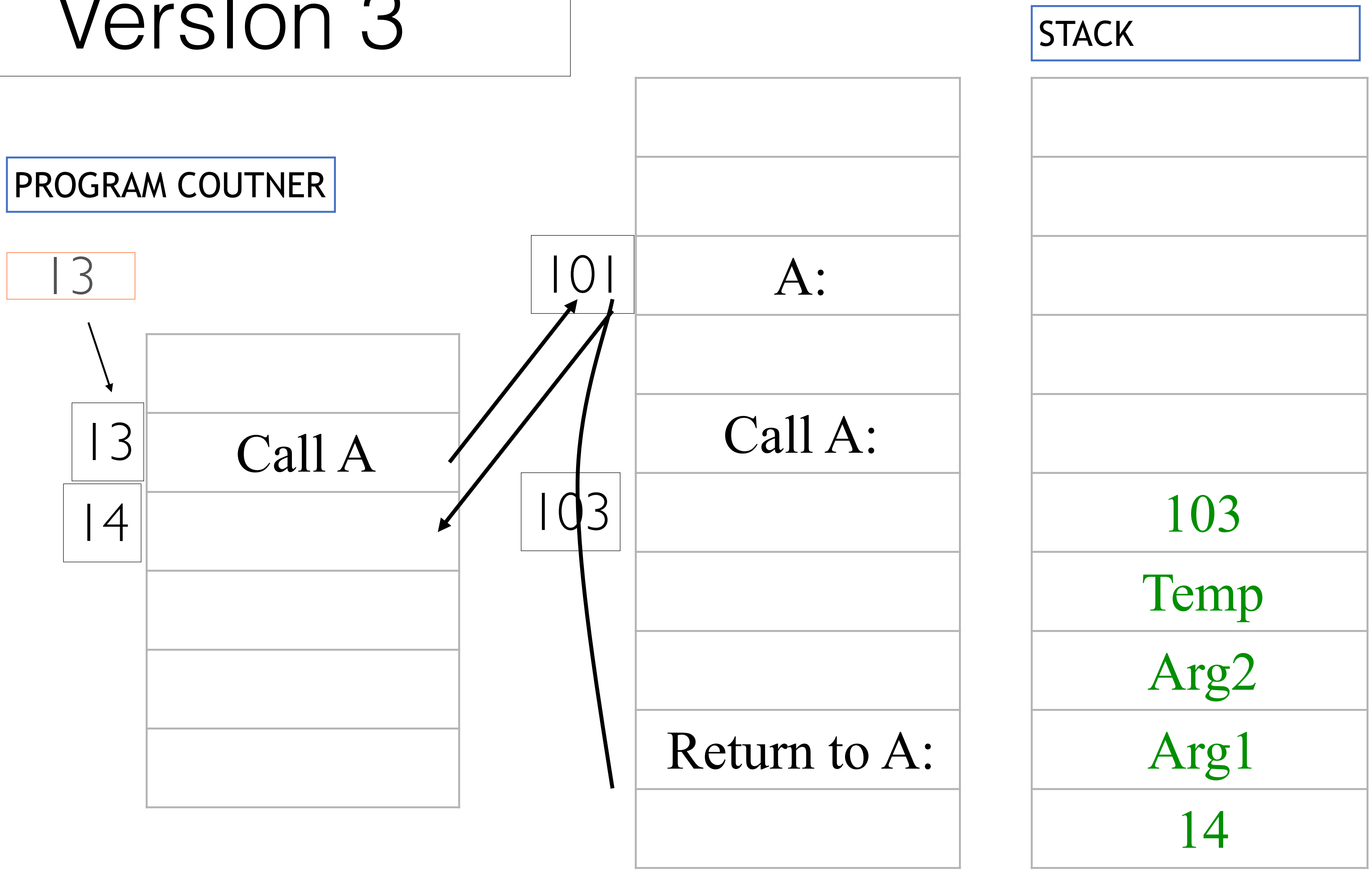
VERSION 1


RETURN REGISTER
14
PROGRAM COUTNER
13
101
A:
13
Call A
14
Return:

VERSION 2


PROGRAM COUTNER
13
13
Call A
14
101
A: 14
Call B:
103
Return to A:
202
B: 103
Return to B:

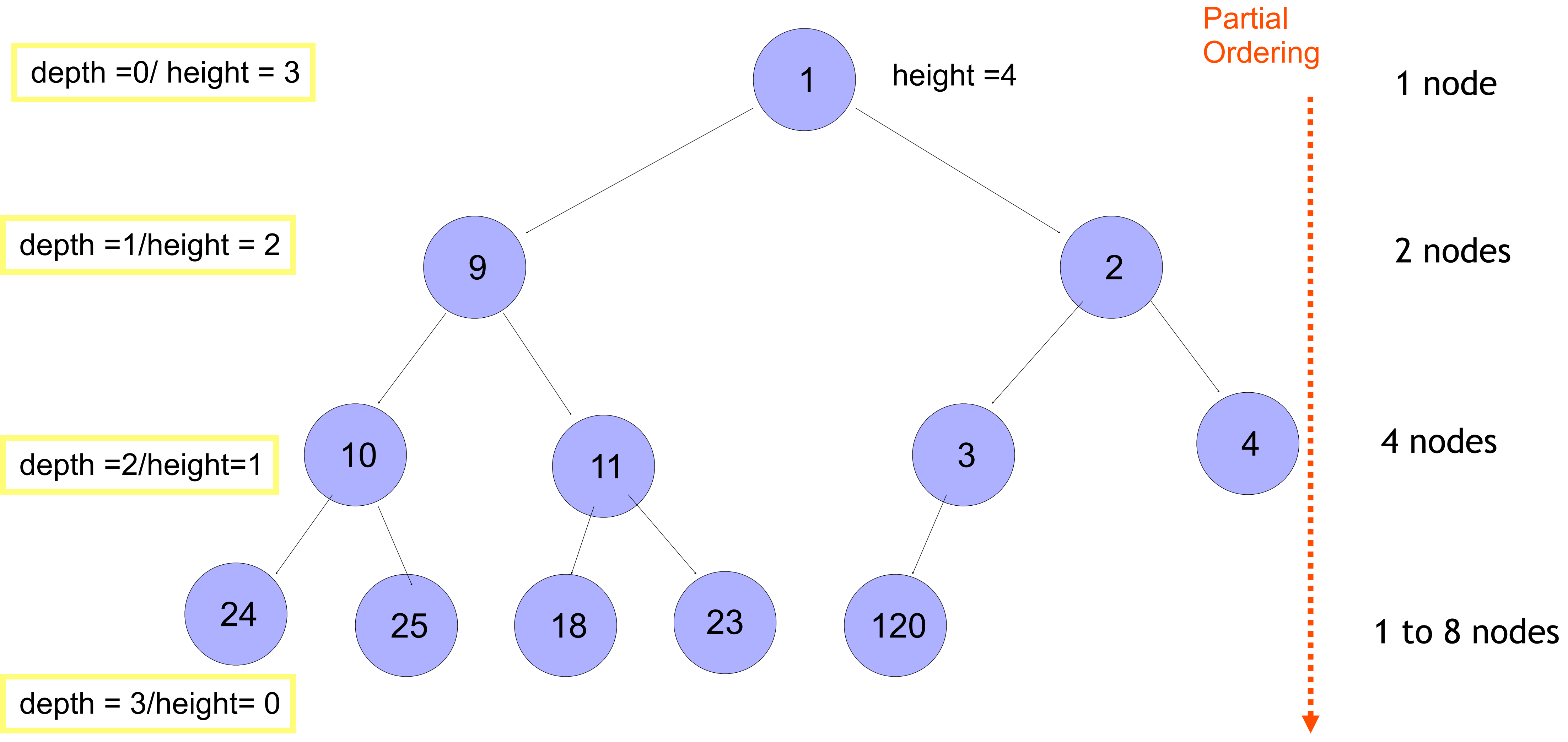
VersIon 3


STACK
PROGRAM COUTNER
13
101
A:
13
Call A
Call A:
14
103
103
Temp
Arg2
Return to A:
Arg1
14

Heaps = Array disguised as Tree

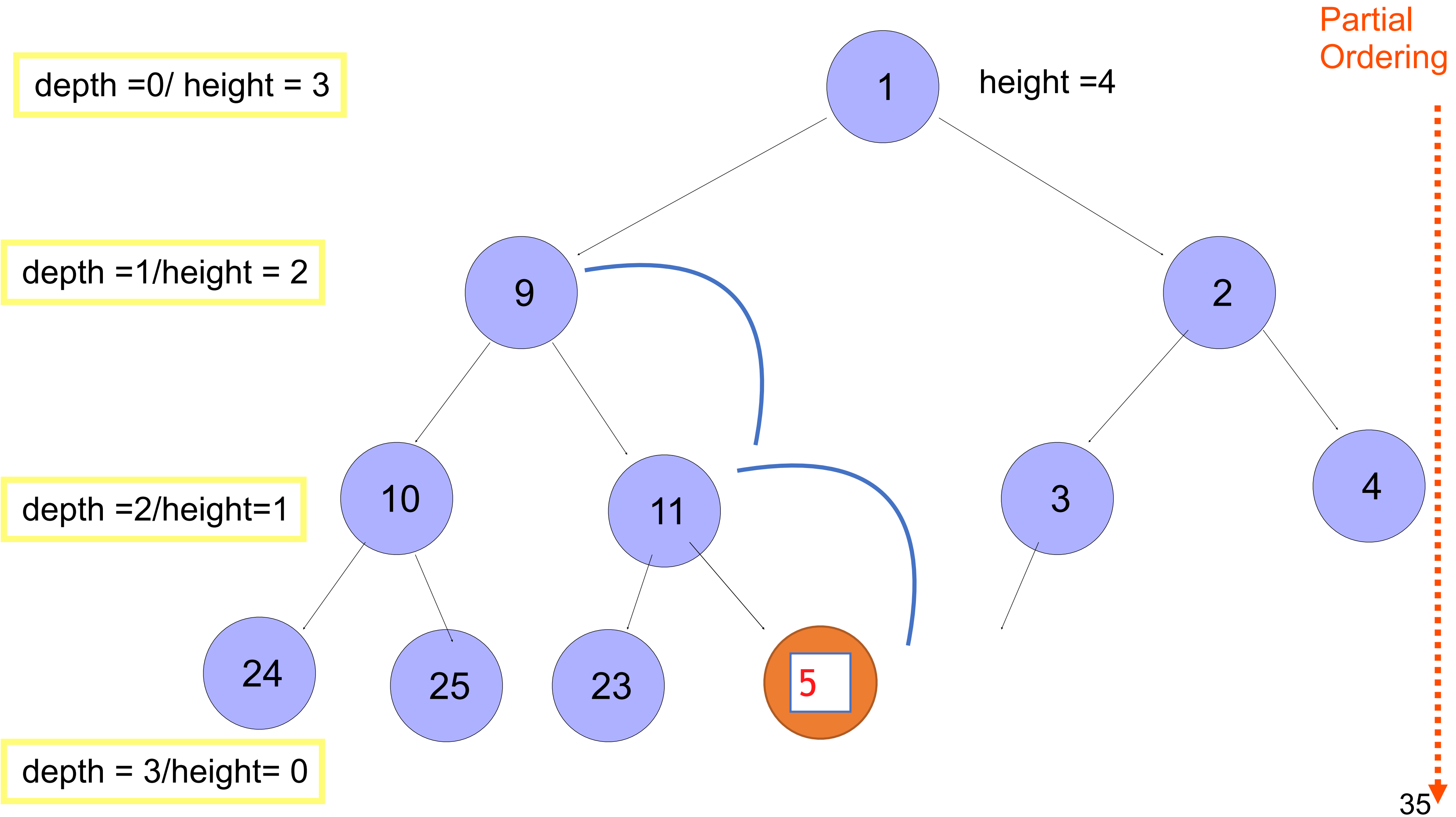
- Basic Heap ADT:
  - *Data is a[i], i = 1,...,N (ROOT = 1, LABEL 0)*
  - *Methods: Insert (key), Delete(key), DeleteMin, Build and Sort*
- Q: When is a tree an array? A: complete tree
  - *Parent a[i] ➔ a[2i] = left child & a[2i+1] = right child*
  - *Child a[j]: ➔ a[j/2] = parent (integer division).*
- 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



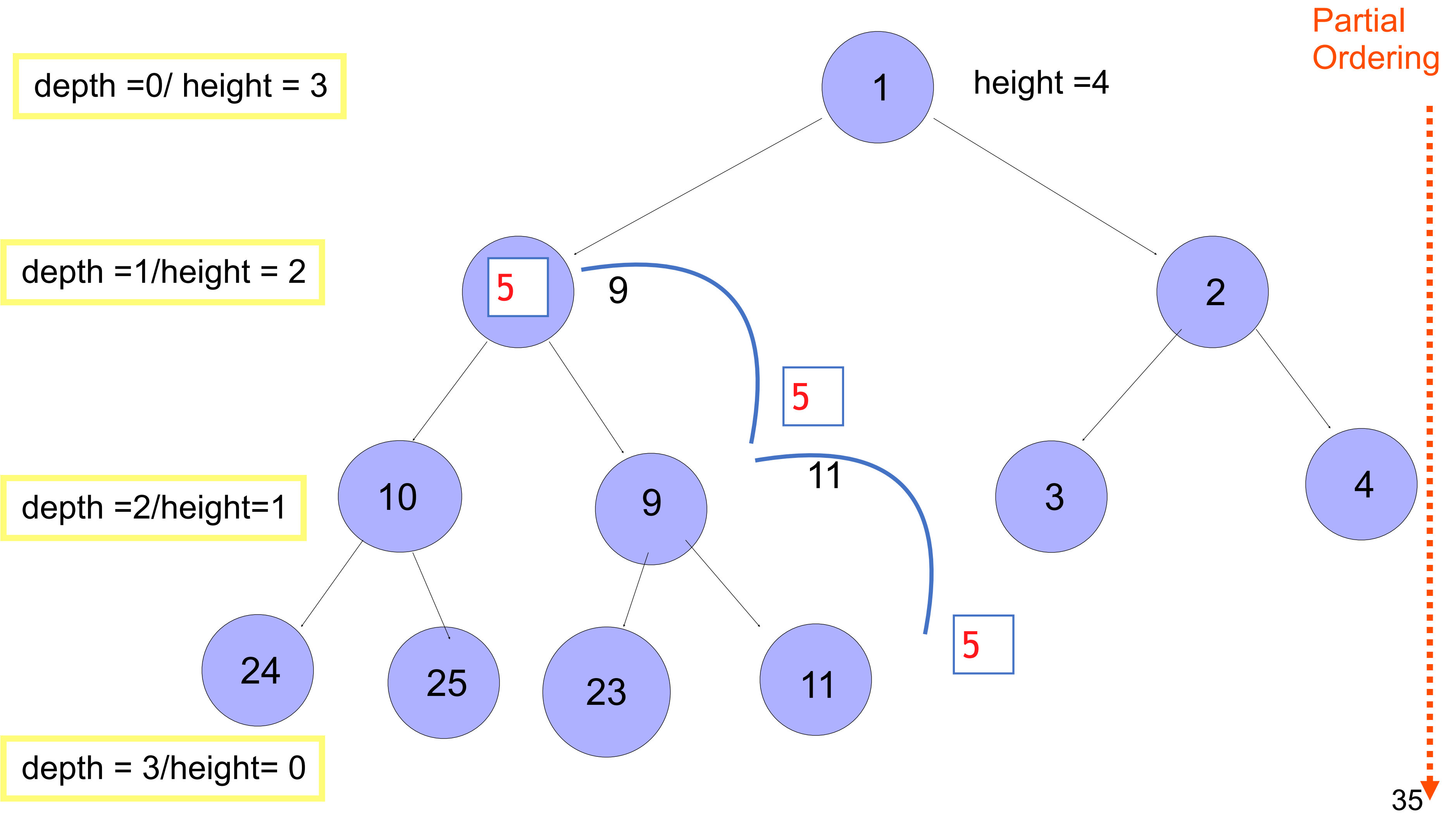
MAX NUMBER OF NODES N = 1 + 2 + 4 + ... + 2^H = 2^{H+1} - 1 and Total height H = O(Log2(N))

Insert 5: Insertion sort on path to root

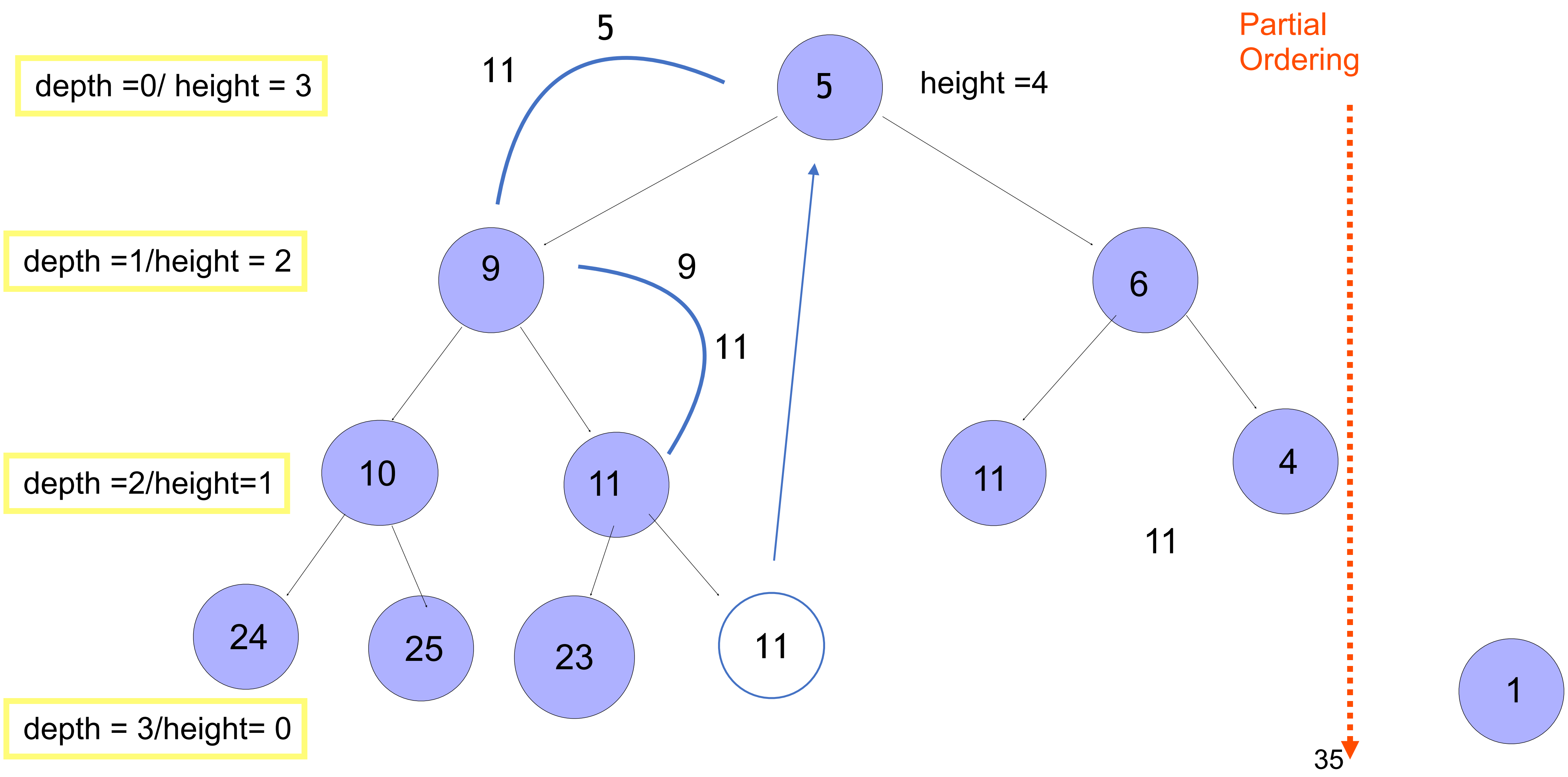


Build by One by One Insert O(N log(N))

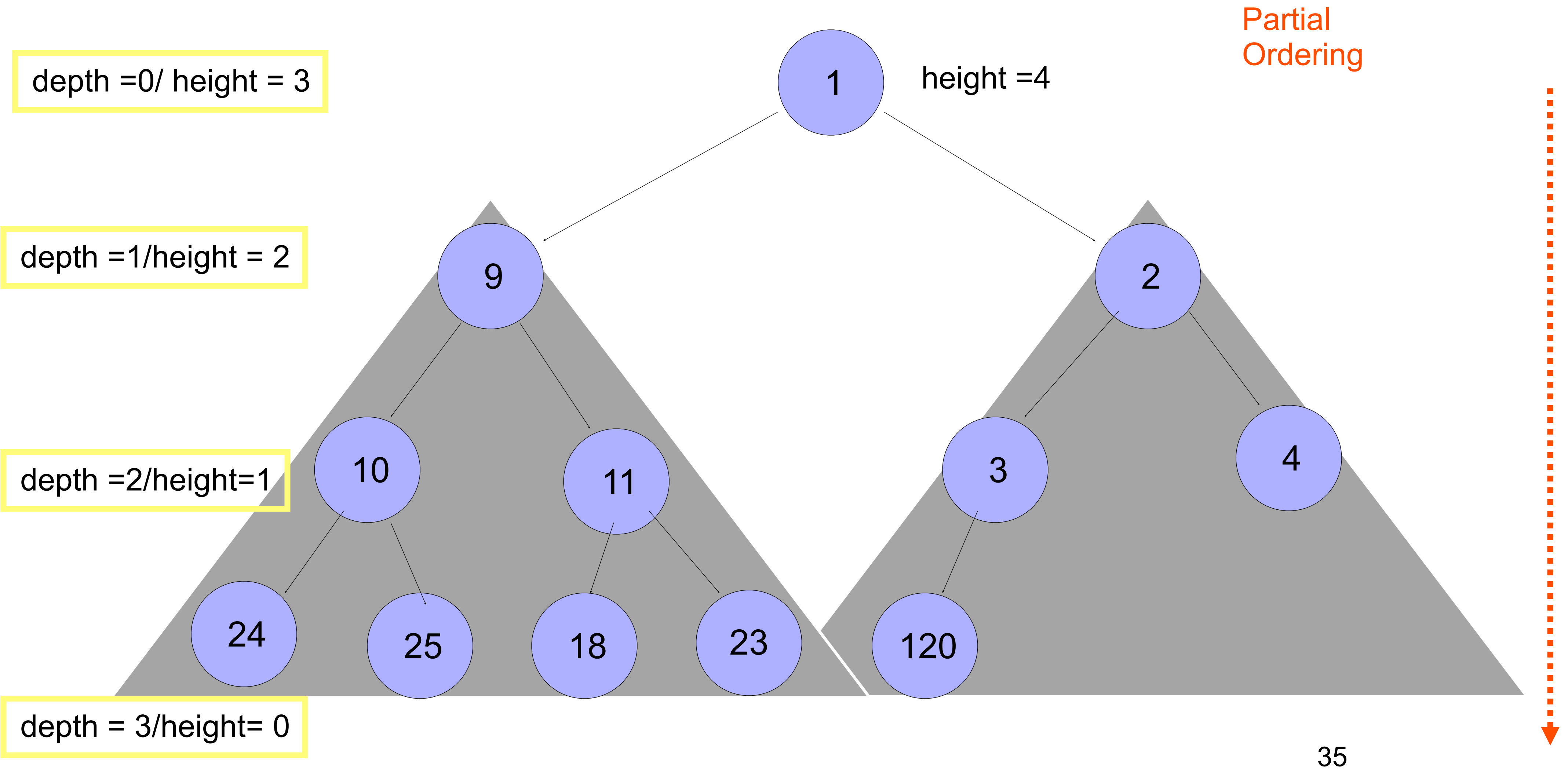
Insert 5: Insertion sort on path to root



Delete 1: Push down 11 to min child

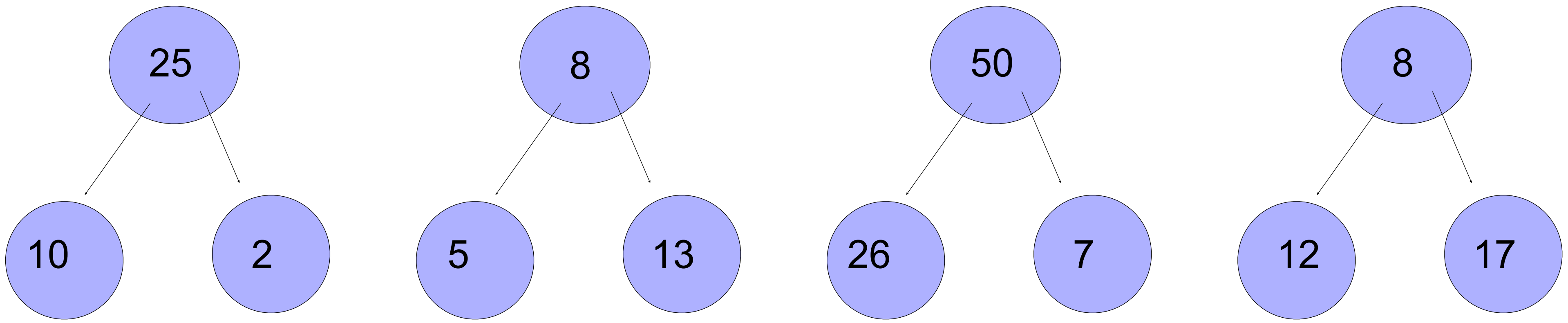


Recursive: Heap = Root + Left and Right Heap



Bottom up Heapify in O(N)

- Take Array as is and them heapify the bottom at most 2^(H-1) pairs



$$T(N) \leq 2^{H-1} + 2 * 2^{H-2} + 3 * 2^{H-3} + \cdots + (H-1) * 1 = 2^H \sum_{n=1}^{H-1} n(1/2)^n$$

Do the sum for large total height H by for x = 1/2

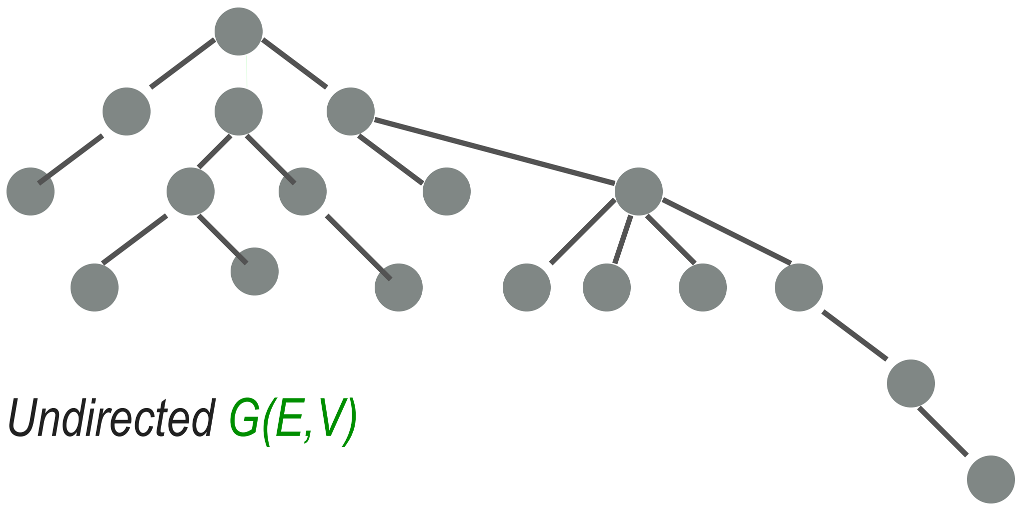
$$\sum_{n=0}^{H-1} n x^n = x \frac{d}{dx} \sum_{n=0}^{H-1} x^n = x \frac{d}{dx} \frac{1 - x^H}{1 - x} \simeq 1/(1-x)^2 = 4$$

or Solve the Recursive relation with $N = 2^H$

$T(H) = 2T(H-1) + c_0 H$ with the guess $T = c_1\, 2^H + c_2\, H$

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"*



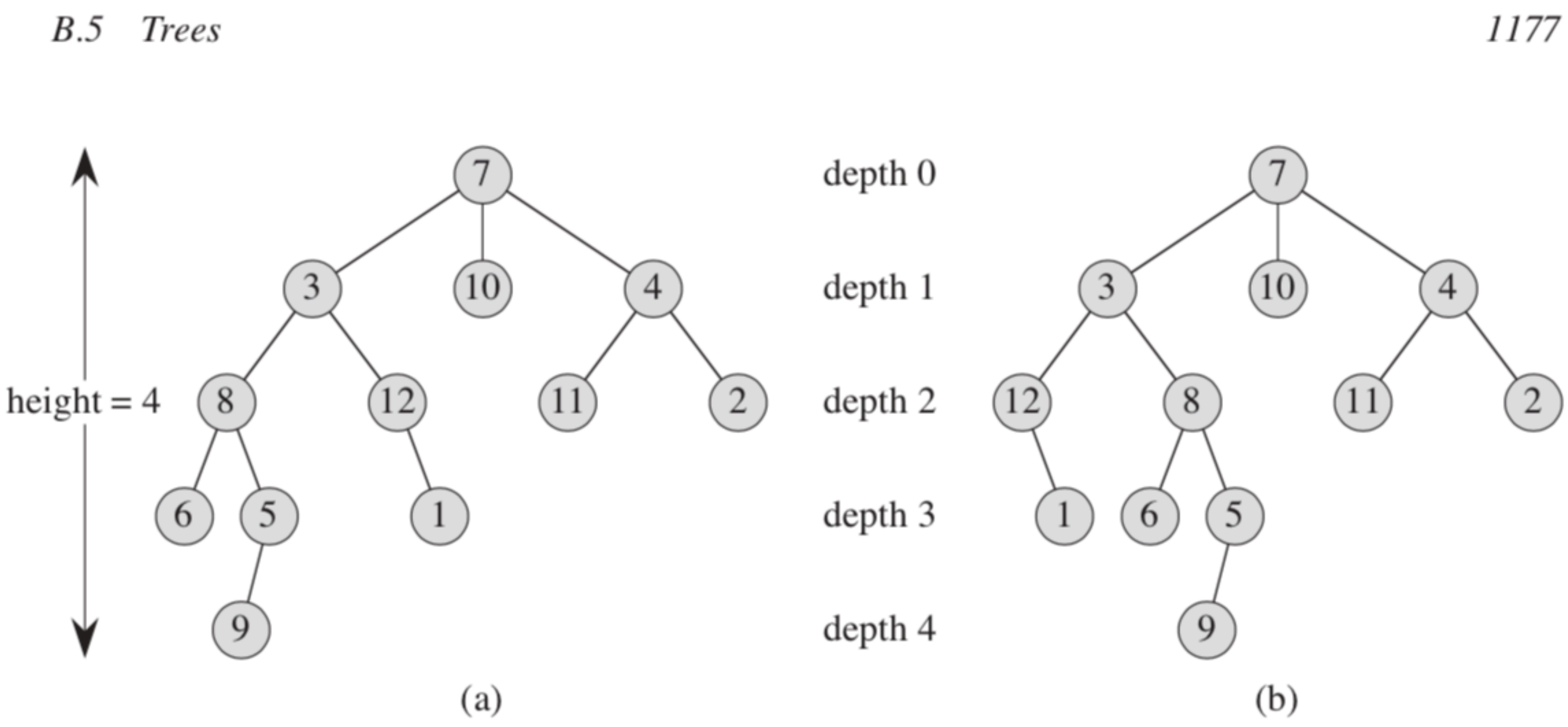
- *A tree is Connected, Acyclic, Undirected G(E,V)*
- *Binary Tree has 0,1,2 children (i.e. nodes have 1,2,3 edges)*

DEFINITIONS FOR BINARY TREES

- Full Tree: 0, 2 children,
- Complete Tree: Consecutive nodes (aka Heap),
- Perfect Tree: Compete and full last row.

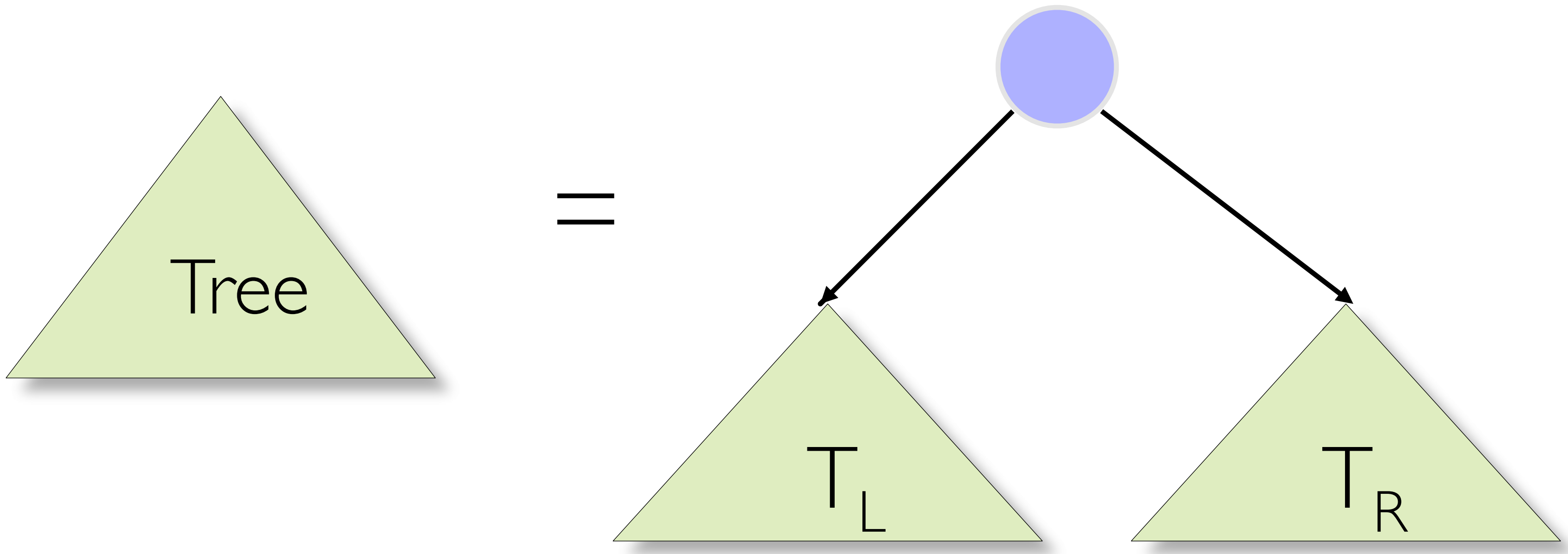
- Full Tree Theorem: # of leaves: $L(N) = (N+1)/2$ for N nodes
- Perfect Tree with H levels (height or depth)
- Nodes in Perfect k-way tree : $N(H) = (k^{H+1}-1)/(k-1) \rightarrow 2^{H+1} - 1$
- Execution Tree
- Traversals: in-, pre-,post-order.

Height vs Depth of "nodes"



BINARY TREE: RECURSIVE DEFINITION

- A binary trees is null or a singe 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.

```
         18
       /    \
     15      20
    /  \
   40   50
  /  \
 30  50
```

Complete Binary Tree: A Binary Tree is complete Binary Tree if all levels are completely filled except possibly the last level and the las
keys as left as possible.

```
            18
       /         \
     15           30
    /  \         /  \
  40    50     100   40
 /  \   /
8   7  9
```

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.

```
           18
       /       \
     15         30
    /  \        /  \
  40    50    100   40
```

TREE TRAVERSALS

- **Preorder:**

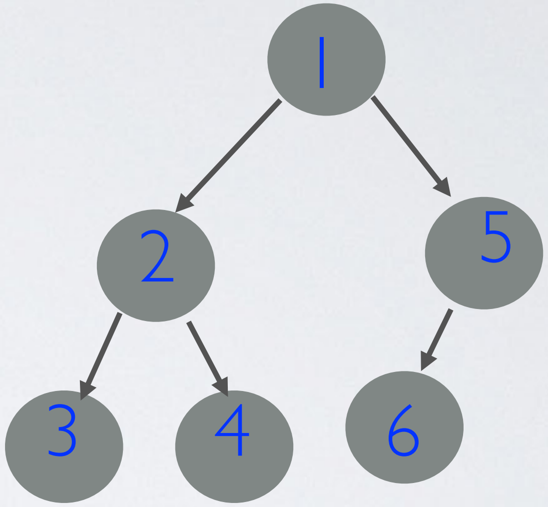
```
Print [Tree]{
        Print root;
        Print Tree[LeftTree];
        Print Tree:[RightTree];
}
```

- **Inorder:**

```
Print [Tree]{
        Print  Tree[LeftTree];
        Print root;
        Print Tree:[RightTree]
}
```

- **Postorder:**

```
Print [Tree]{
Print  Tree[LeftTree];
Print Tree:[RightTree]
Print root;
}
```

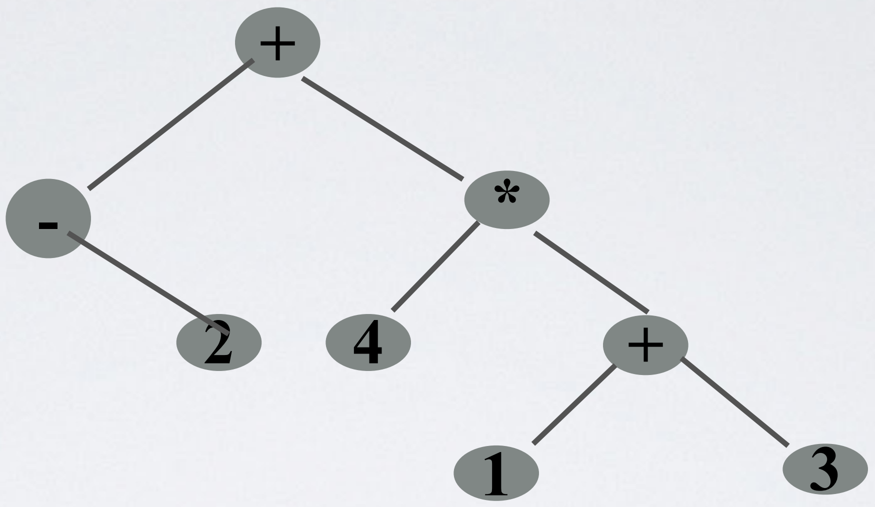


Pre: 1➔2➔3➔4➔5➔6

In: 3➔2➔4➔1➔6➔5 sort on BST

Post: 3➔4➔2➔6➔5➔1

Expression Trees



<u>Preorder</u>: + - 2 * 4 + 1 3 (Lisp, Scheme) (+ (- 2) (* 4 (+ 1 3)))

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

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

<u>Binary Search Tree</u>: left <= 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: $N(H) = 1 + k + k^2 + k^3 + .. + k^H = (k^{H+1}-1)/(k-1)$

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

- 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) $2\,(H+1)\ 2^H - 2\,(2^{H+1} - 1) = (H-1)\,N + H + 1$

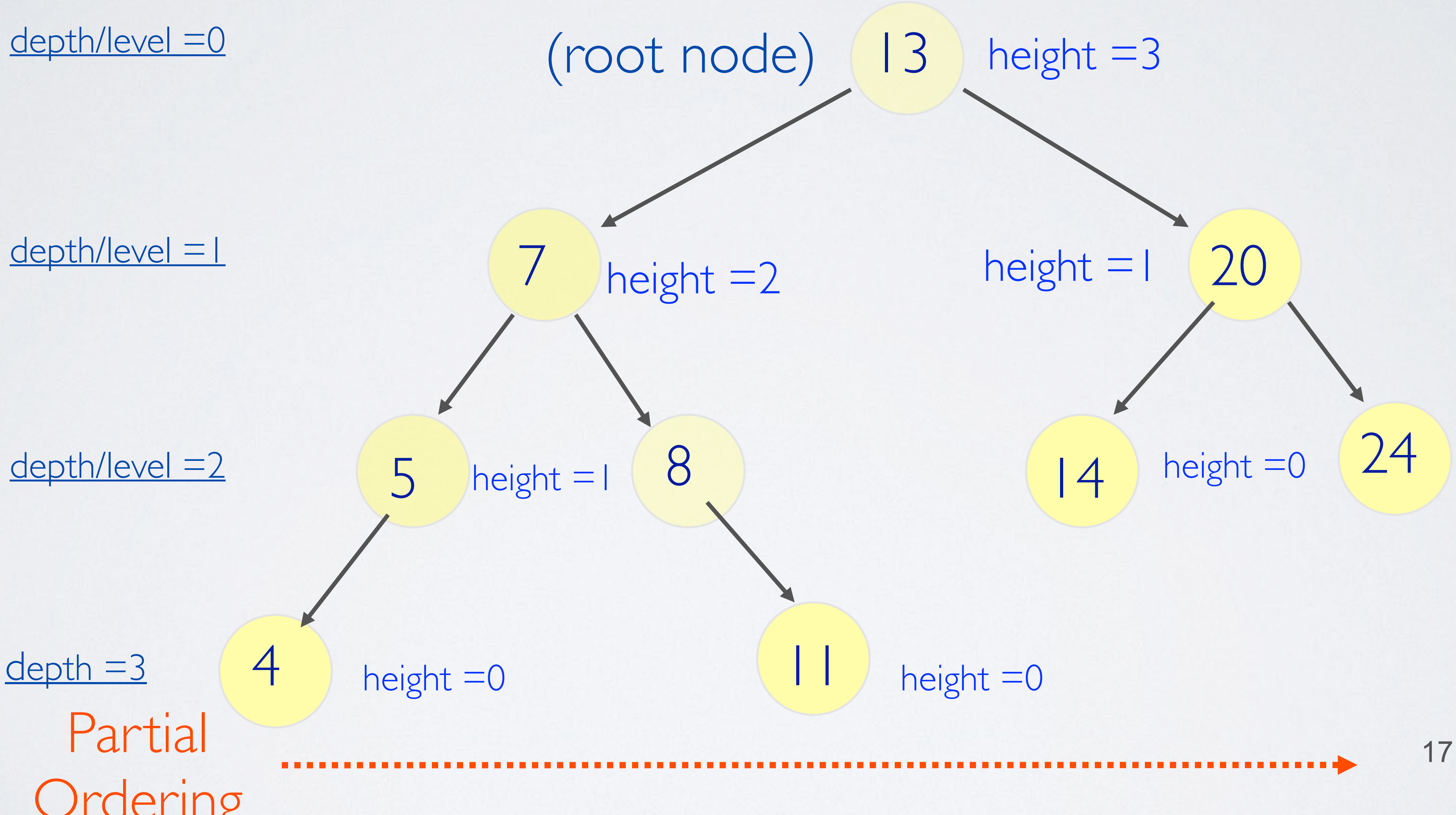
- total Height : $T_H(N) + T_D(N) = H\,N$ (each $h + d = H$)

  $T_H = H\,N - T_D = N - H - 1$ (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



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 ($K_{left} <= K_0$) less than all keys in T_R, the Right subtree ($K_0 < K_{Right}$)

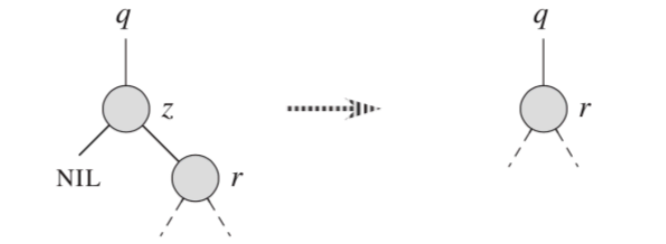
3. The BST defines a partial ordered set --- as you move down to the left/right the keys decrease/increase.

4. Insert new K_{new} push down to subtree Left/Right if K_{new} <=/> K_0.

5. Delete K_0 and replace by SMALLEST key in T_R, the Right subtree.

BST DELETE Z

(a)

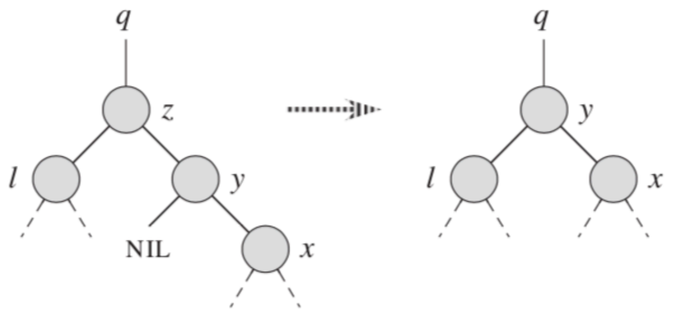


Right Only Child

(b)

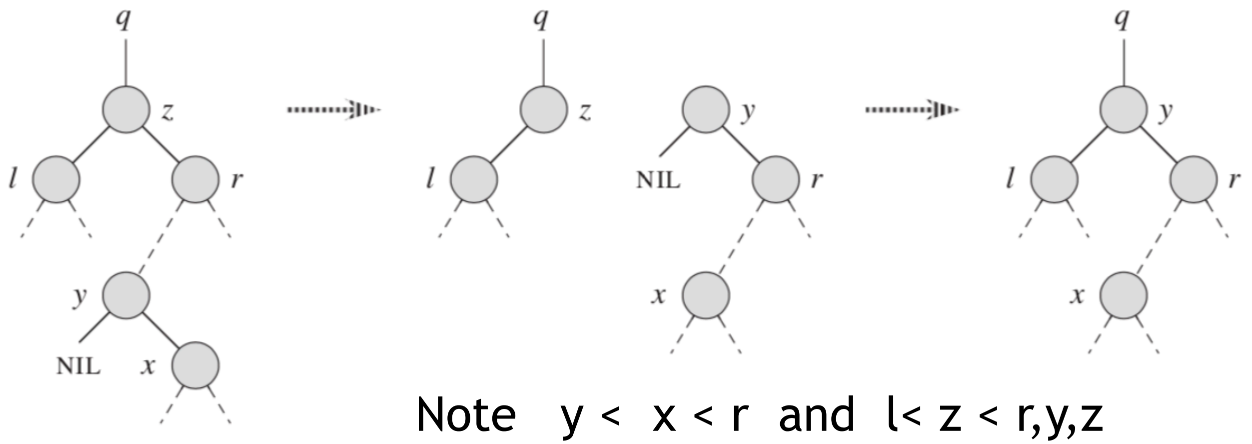
Left Only Child

(c)



R & L Twins

(d)



Right to Left Rotation

Note y < x < r and l< z < r,y,z

AVERAGE TOTAL DEPTH OF BST

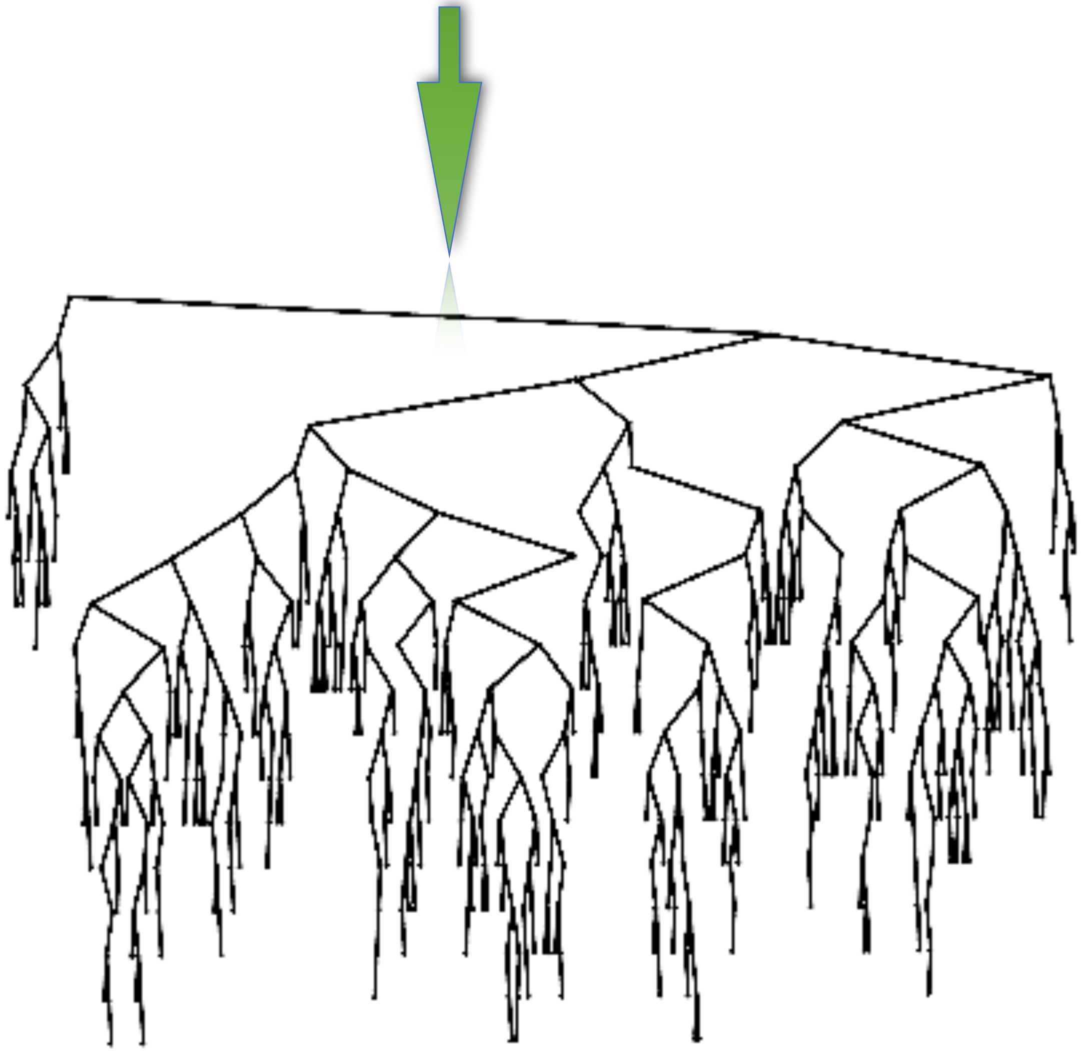
$$\begin{aligned}
T_D(N) &= \frac{2}{N}[T_D(0) + T_D(1) + T_D(2) + \cdots + T_D(N-1)] + c(N-1) \\
T_D(x) &\simeq \frac{2}{x}\int_0^x T_D(x) + c(x-1) \\
xT_D(x) &\simeq 2\int_0^x T_D(x) + c(x^2 - x) \\
&\Rightarrow T_D(x) + x\frac{dT_D(x)}{dx} = 2T_D(x) + c(2x-1) \\
\frac{dT_D(x)}{dx} &\simeq T_D(x)/x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

SAME AS QUICK SORT!

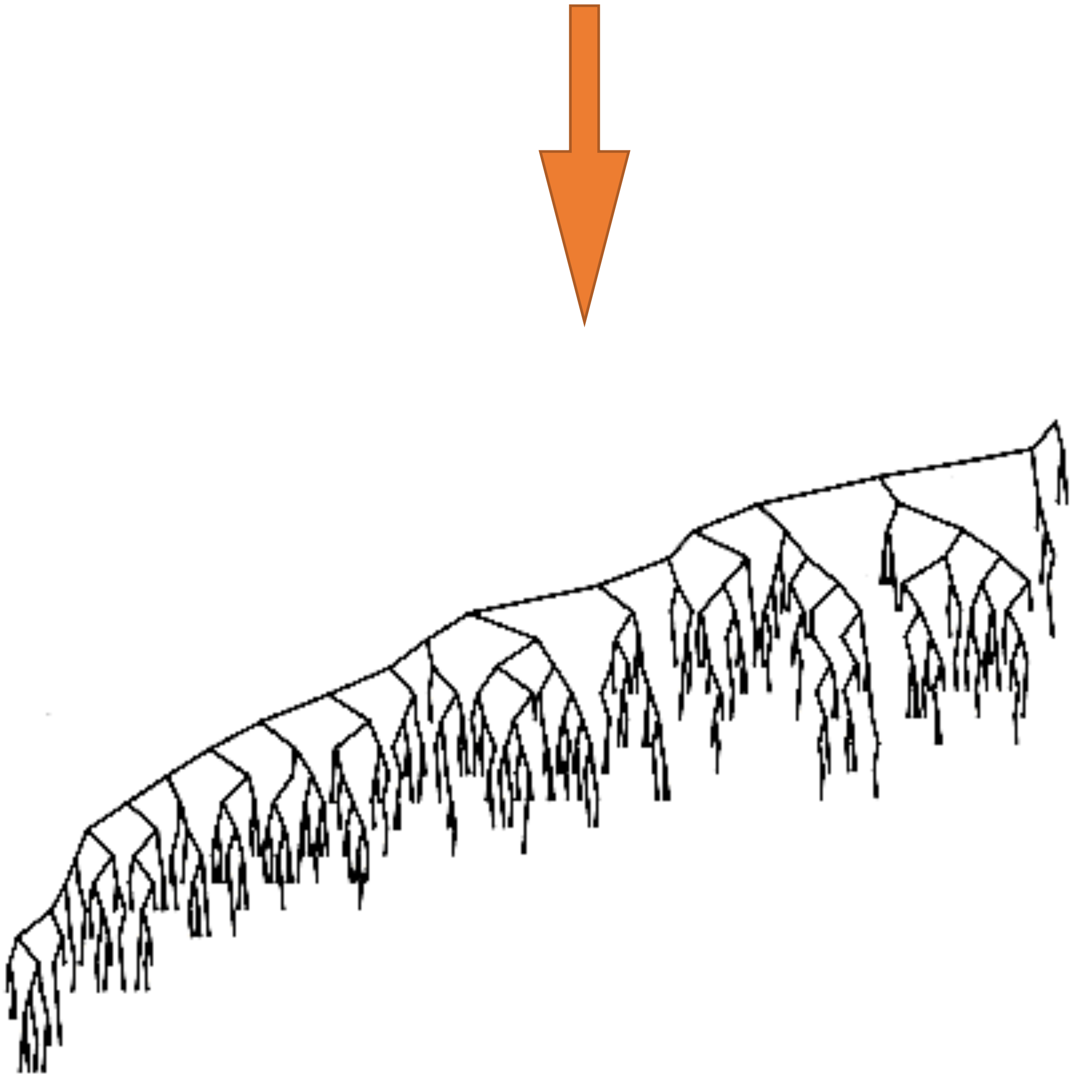
- Solution: $T_D(N) = \Theta(N \log(N))$

See Average of Quick Sort Sec 7.7.5 (p 278)

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



Depth O(log N)



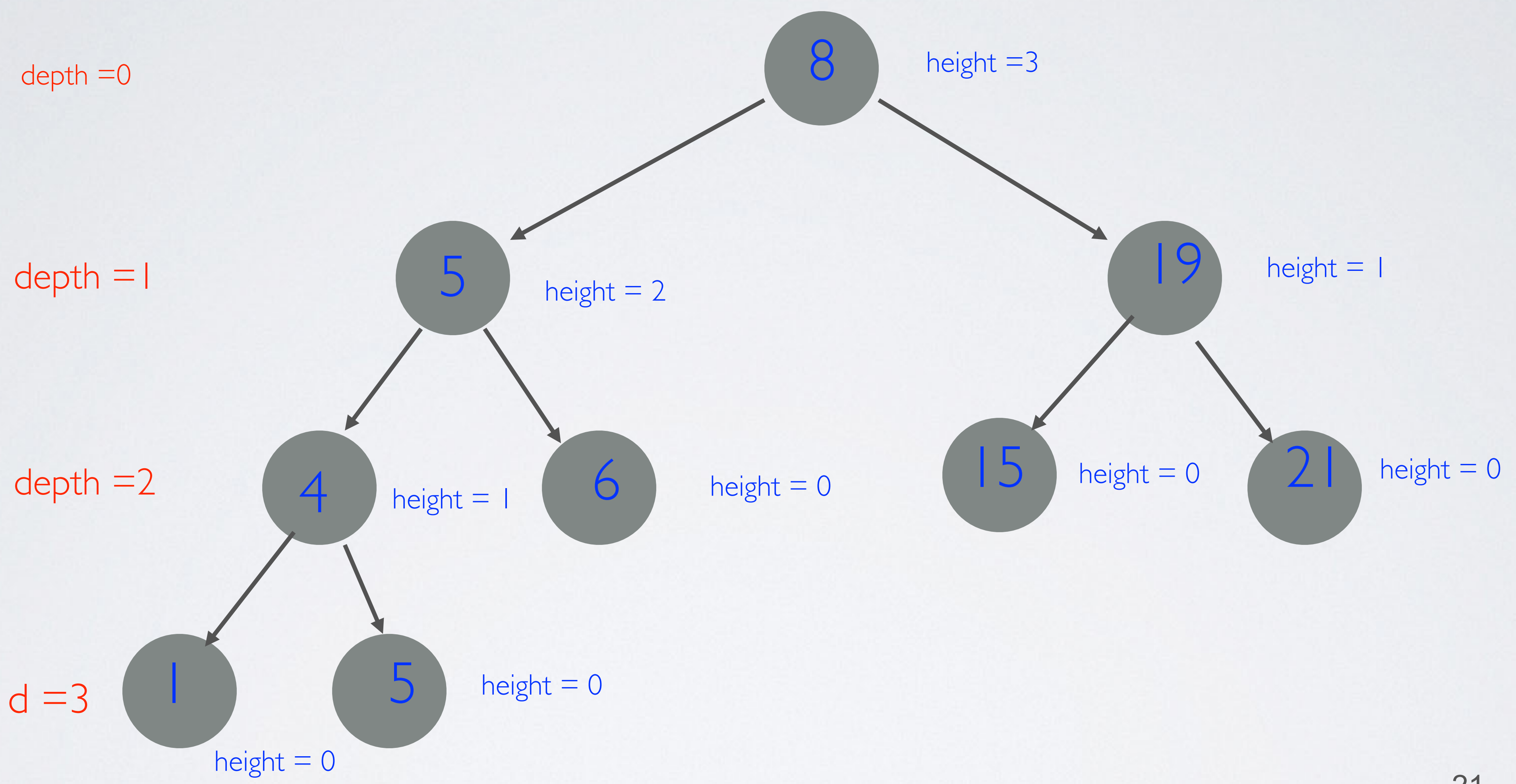
Depth O(N)

RELATIONS: BOOLEAN VALUED MATRIX R[A,B]

- Set: S = {a,b,c,....}
- Relation (a,b) 2 S x S: a R b is True?
- Properties:
  - Reflexive: a R a is True
  - Anti-symmetric: a R b and b R a ➔ a = b
  - Transitive: a R b and b R c ➔ a R c
  - Total Ordering: a R b or b R a (inclusive or)
  - Self dual: a R b ←➔ b R a
  - Transpose: a R b ←➔ b R^T a
- RAT is partial ordering: e.g. descendants in a tree!

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

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



WORST CASE HEIGHT H(N) FOR AVL

- Minimum # of Nodes (see Fig 4.33):

$$N(H) = N(H\text{-}1) + N(H\text{-}2) + 1 > N(H\text{-}1) + N(H\text{-}2)$$

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

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

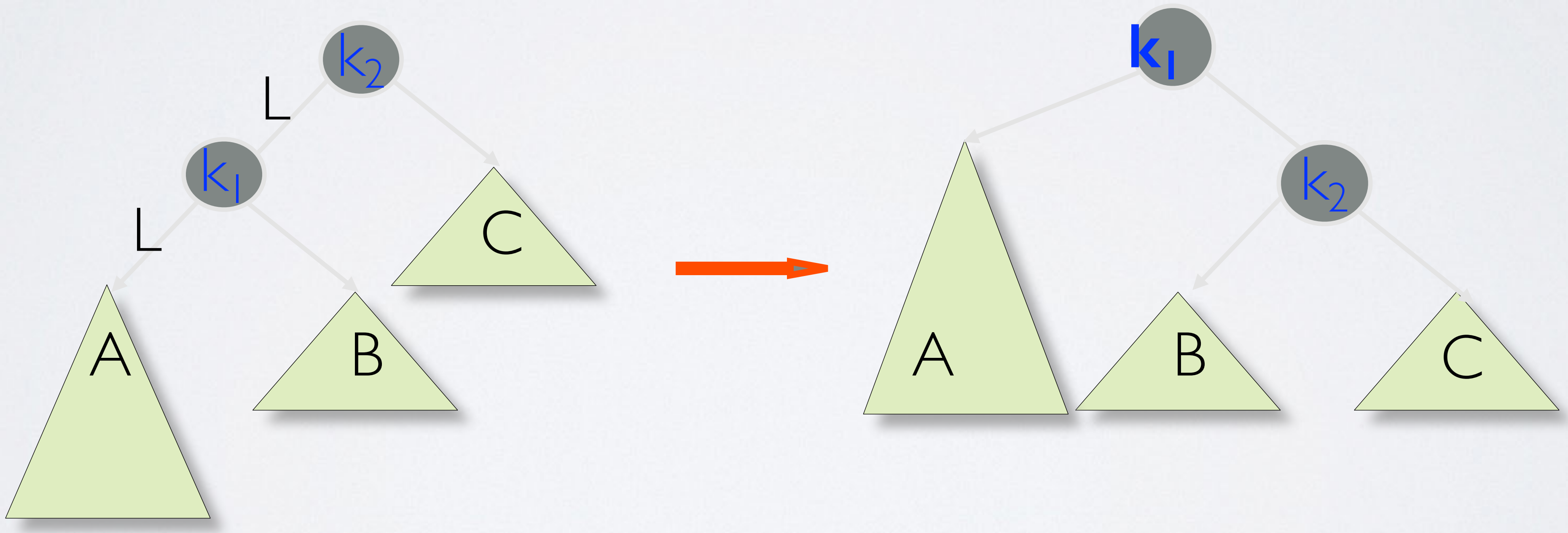
  - Or $H <$ $\log(N)/\log(c)$ $1.440420 \log_2(N) = 2.078 \ln(N)$

$$= 4.784 \log_{10}(N)$$

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

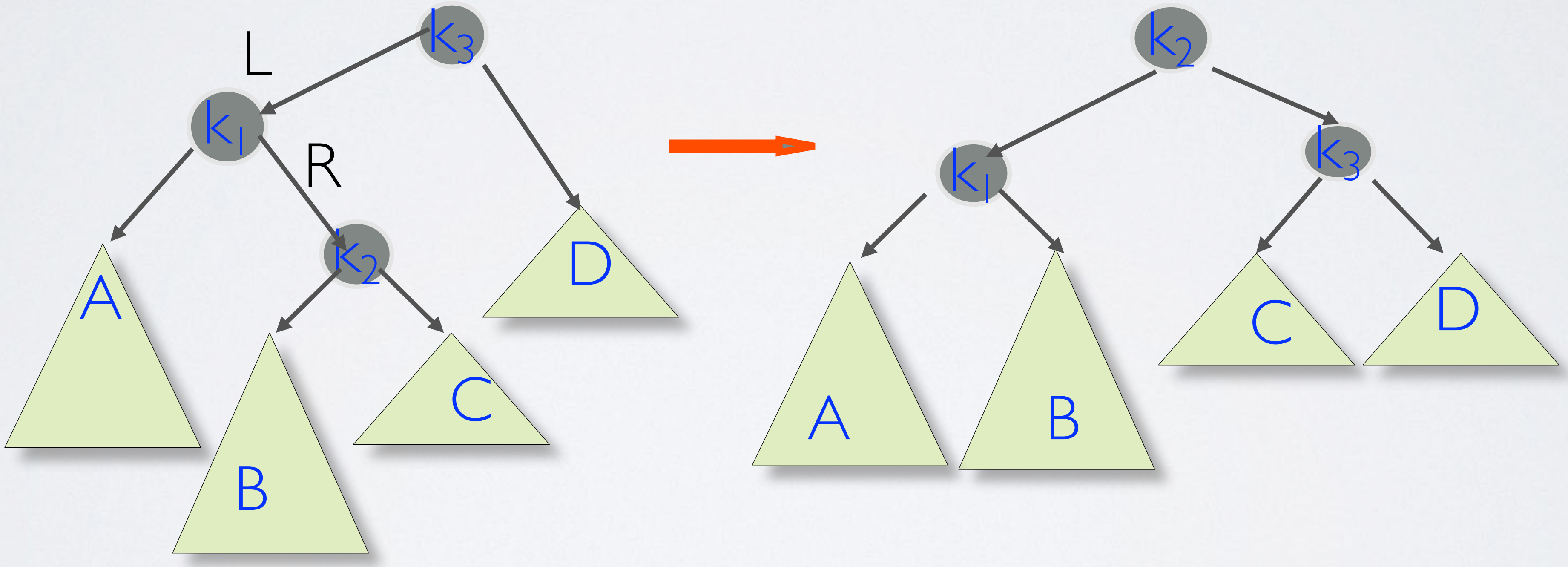
ZIG-ZIG INSERTION FOR LL OR RR:

- *Insert New Key along path going Left and Left again into A:*
- *This cause violation of AVL balance.*
- *k_2 is lowest node failing AVL balance.*
- *Single rotation of k_1 ➔ 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 blance*



HUFFMAN CODING

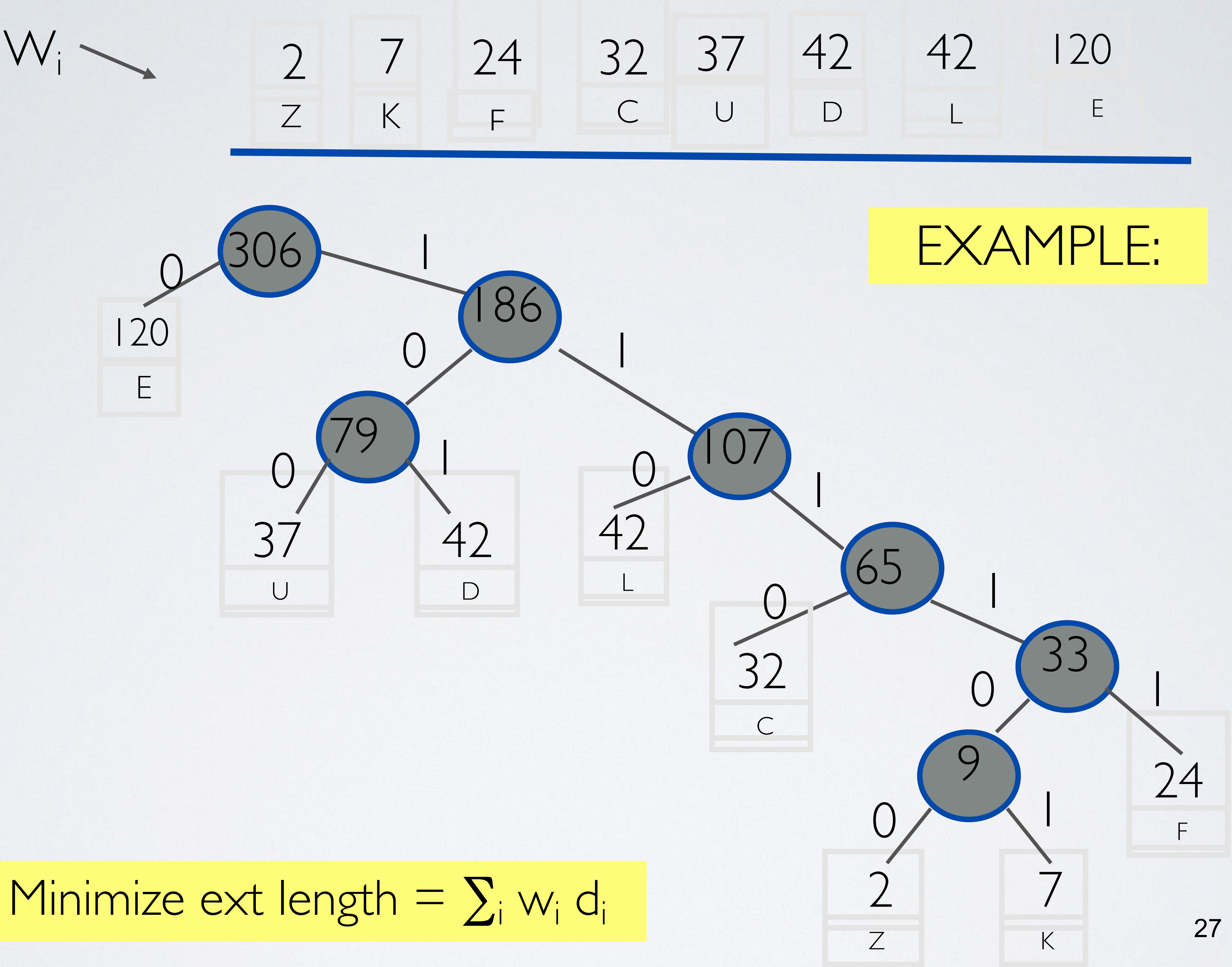
- Place all letters at leaves of a binary tree
  - The code is path (i.e. address) of each leaf.
  - Binary code for each letter: e.g. “a” = 01001, “b” = 101, ..

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

Build the Huffman tree:

- Sort symbol list: $w_1 < w_2 < \ldots < w_N$
- 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

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:*
- $Min[(d_{12} + 1)\,(w_1 + w_2) \;+ w_3\, d_3 + \ldots + w_N\, d_N]$ *over all trees T*

$$= (w_1 + w_2) + Min[d_{12} w_{12} + w_3\, d_3 + \ldots + \; w_N\, d_N]$$

"SCHWARTZ" PARING INEQUALITY!

Need for Huffman and Many Opt Algorithms

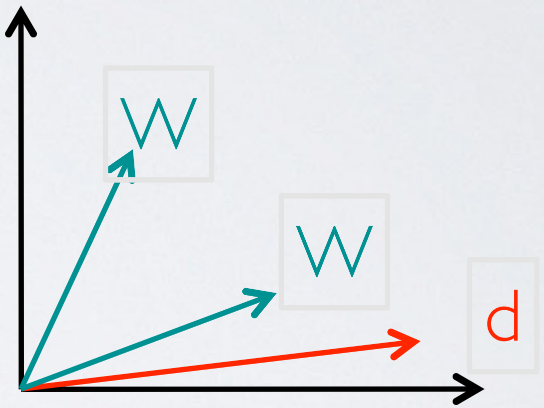
Prove: (more parallel is larger!)

$w_S\, d_S + w_L\, d_L > w_L\, d_S + w_S\, d_L$

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 $S = MIN_\pi \sum_I I * a[\pi(I)]$

    swap minimize I a[I] + J a[J] if out of order
- Continuum vs Discrete:
  - Bisection : Log(N) vs error => error/2
  - Find zero: ff(x) = 0 or (continuous)
  - Find key f[I] = (a[I] - key)^2 = 0 (a[I] sorted)
- Newton's, Secant, Regula falsi

  & Dictionary method (linear extrapolation)

  Log(Log(N)) vs error => $(\text{error})^\phi$

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