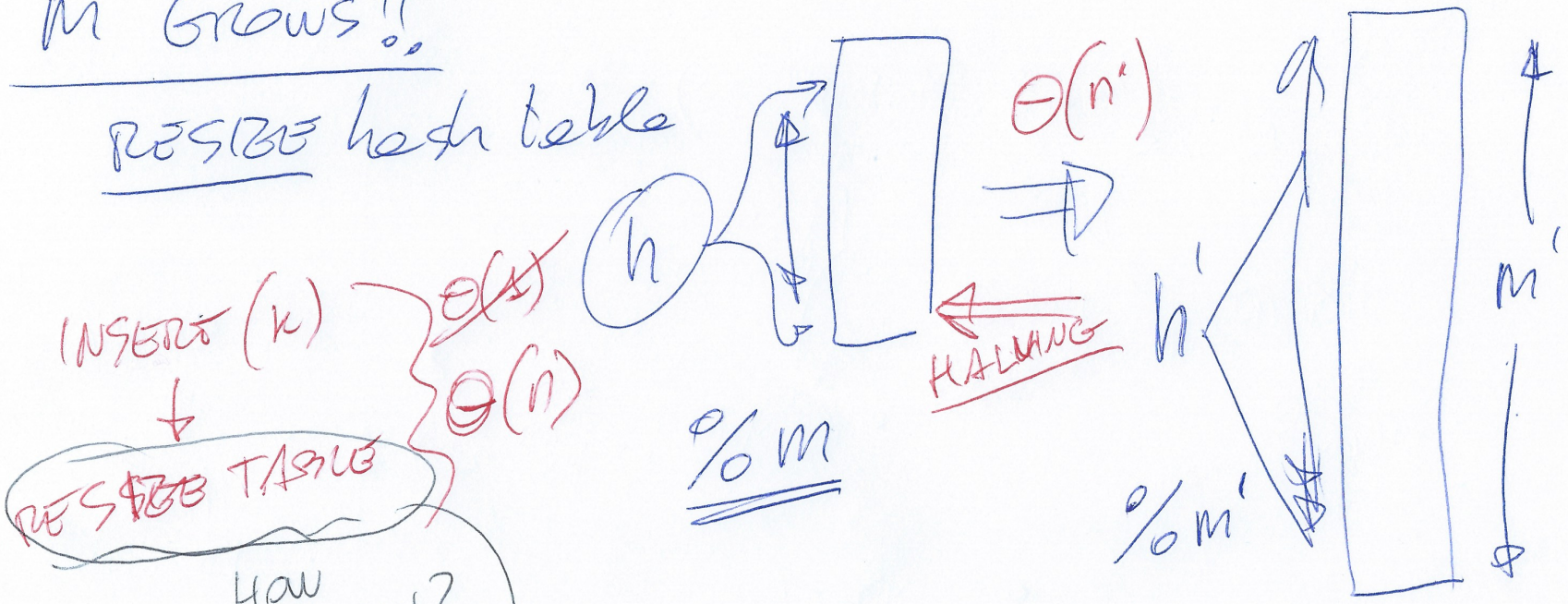
(1) HASHING → DIRECT ACCESS TO TABLE

$h(k)$ ≈ SHUFFLES

M GROWS!!

RESIZE hash table



INCREASE BY 1 ? TOO OFTEN

INCREASE BY Δ HAPPENS EACH Δ OPS

DOUBLE SIZE OK FOR NEXT Θ(n) OPERATIONS

② AMORTIZED COMPLEXITY

$$\sum_{i=0}^{n} T(op_i) = \Theta(\sim)$$

TOTAL COMPLEXITY OF THIS SET OF OPERATIONS

$\Rightarrow$ AM. COMPLEXITY $\Theta(\,)$ / #OPS.

n INSERTIONS (DOUBLING WHEN NEEDED)

$\longrightarrow$ $1 \cdot \Theta(n)$ (INSERTING + DOUBLING 1 TIME)

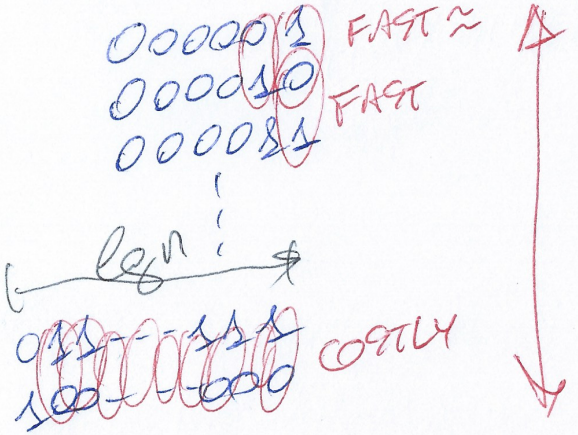
$+$

$n \cdot \Theta(1)$ (INSERTING (NO DOUBLING) n TIMES)

$$\frac{\Theta(n)}{n} = \Theta(1)$$

AMORTIZED

~~WORST~~ CASE (EXPECTED)



TOTAL

HOW MANY BIT FLIPS IN N INCREMENTS?

$\rightsquigarrow$ SHOW $\Theta(1)$ AMORTIZED BIT FLIPS EACH INCREMENT

3. HASHING DESTROYS ORDERING

- NO RANGE SEARCH

FIND ALL k S.T. $k_1 \leq k \leq k_2$

- NO SUCCESSOR SEARCH
- PREDECESSOR
- MAX
- MIN

SHOULD COMPUTE $h(k)$ FOR EACH POSSIBLE $k_1 \leq k \leq k_2$

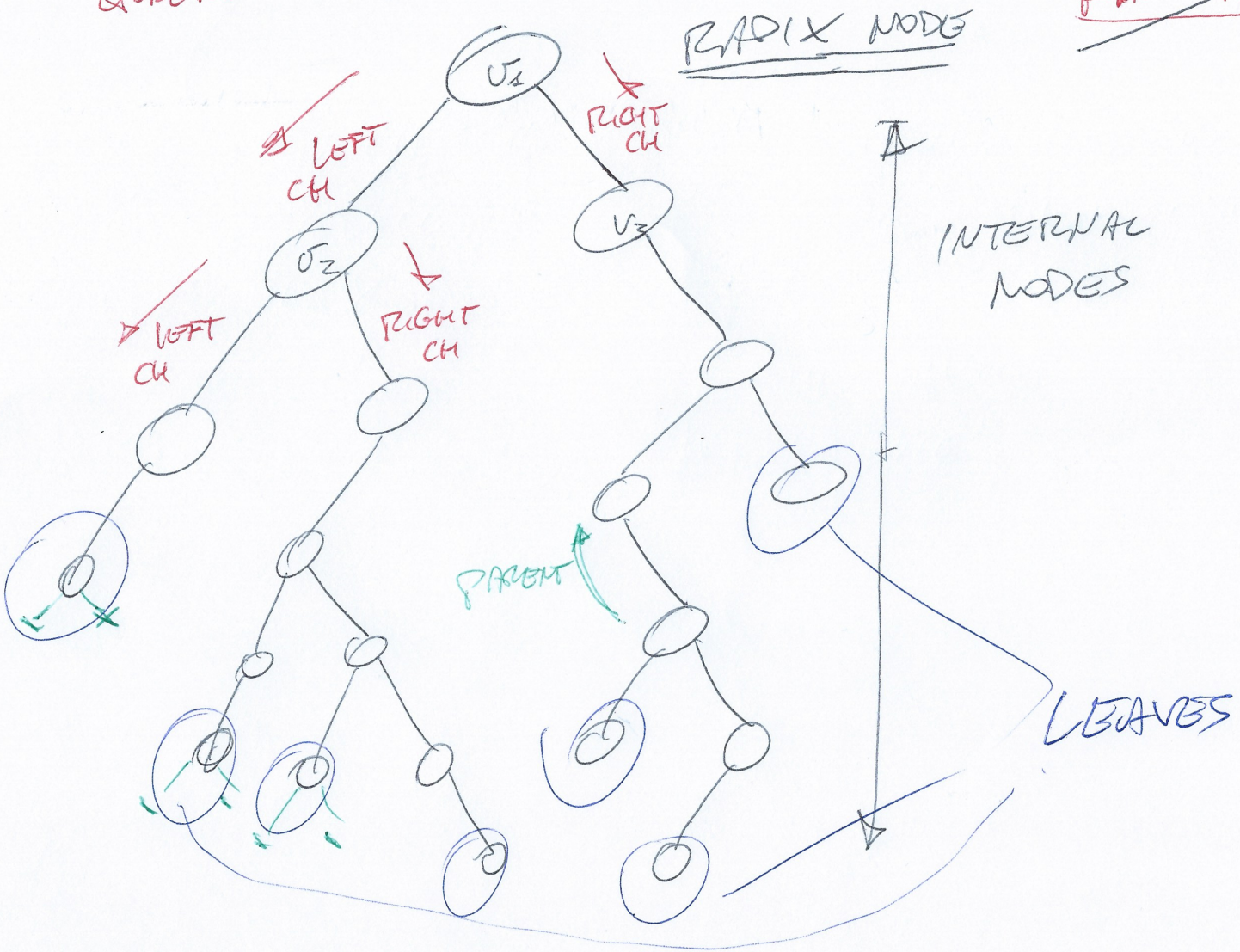
→ SCAN THE WHOLE HASH TABLE

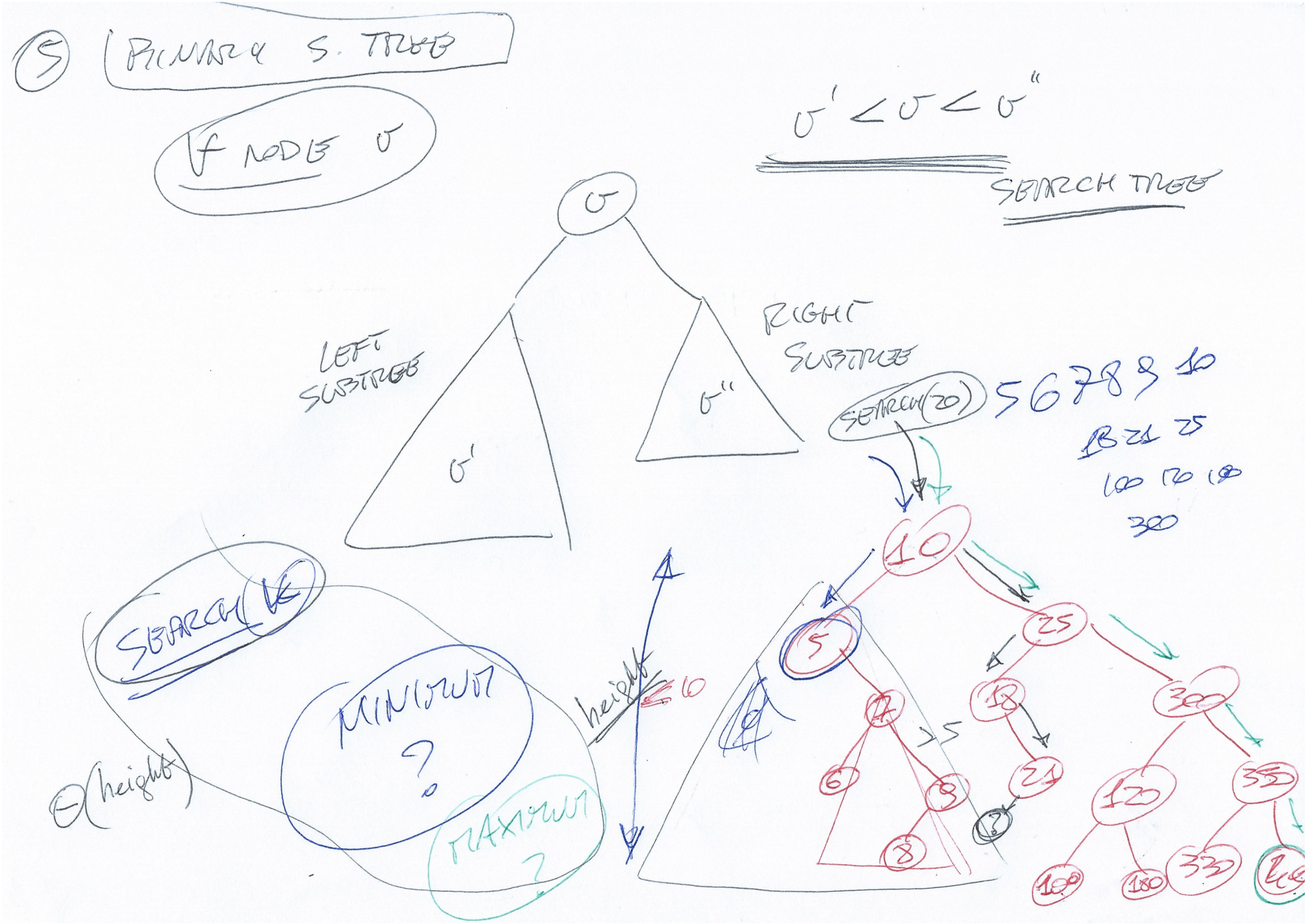
(BINARY) SEARCH TREES

KEYS FROM A TOTALLY ORDERED SET

$k_1 < k_2$?

NUMBERS OR STRINGS OR FINGERPRINTS




5
BINARY S. TREES
FIND NODE v
v' < v < v"
SEARCH TREE
v
LEFT SUBTREE
v'
v"
RIGHT SUBTREE
SEARCH(20)
5 6 7 8 9 10
15 25
100 120 180
300
SEARCH(k)
MINIMUM ?
MAXIMUM ?
Θ(height)
height ≤ 10
10
5
25
15
300
7
>5
6
9
21
8
?
12
35
100
180
330

BINARY TREE VISIT

SEARCH

- IN-ORDER VISIT

VISIT (n) ← NODE

- if n is NIL

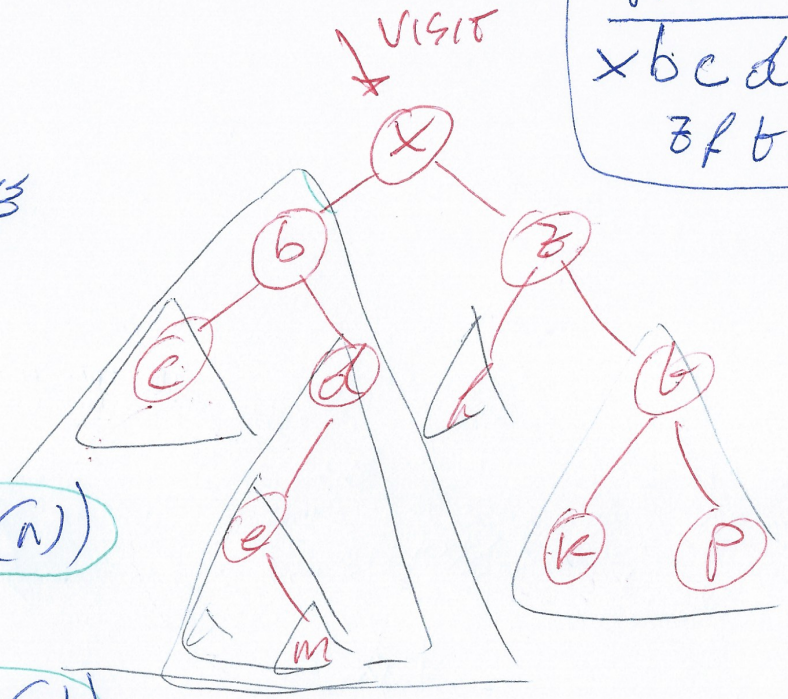
else

VISIT (LEFT CHILD(n))

PRINT n

VISIT (RIGHT CHILD(n))

SORTED KEYS



PRE-ORDER

x b c d e m z k p

IN-ORDER

c b e m d x k z

LEFT RIGHT

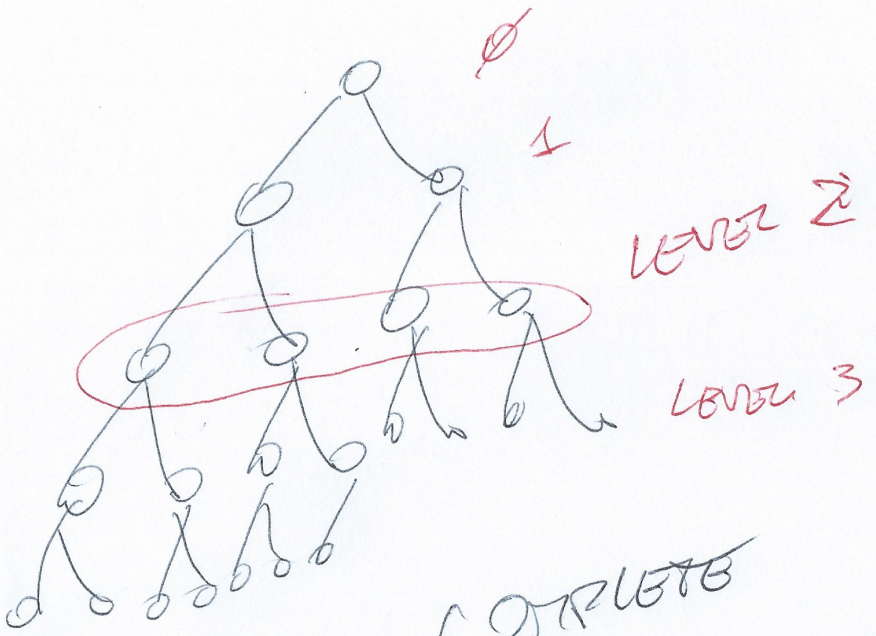
- PRE-ORDER VISIT (n)
  - ~~VISIT~~ (n)
  - PRINT
  - VISIT (LEFT (n))
  - VISIT (RIGHT (n))

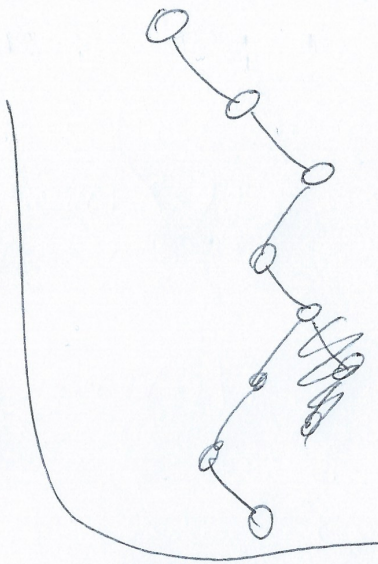
- POST ORDER VISIT (n)
  - VISIT (LEFT (n))
  - VISIT (RIGHT (n))
  - PRINT (n)

7) HEIGHT BINARY SEARCH TREE

$\log_2 n \leq h \leq n$

COMPLEXITY OF SEARCH, MIN, MAX





PATH (IS A TREE)

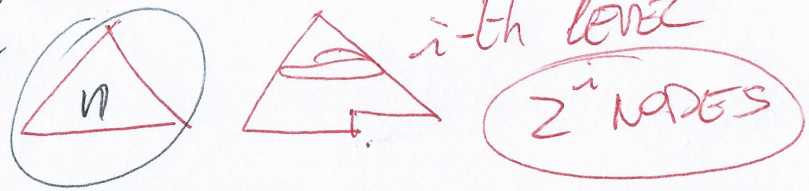
elements level $i = 2^i$

$$\sum_{i=0}^{h} 2^i = 2^{h+1} - 1 = n$$

$h \approx \log_2 n$

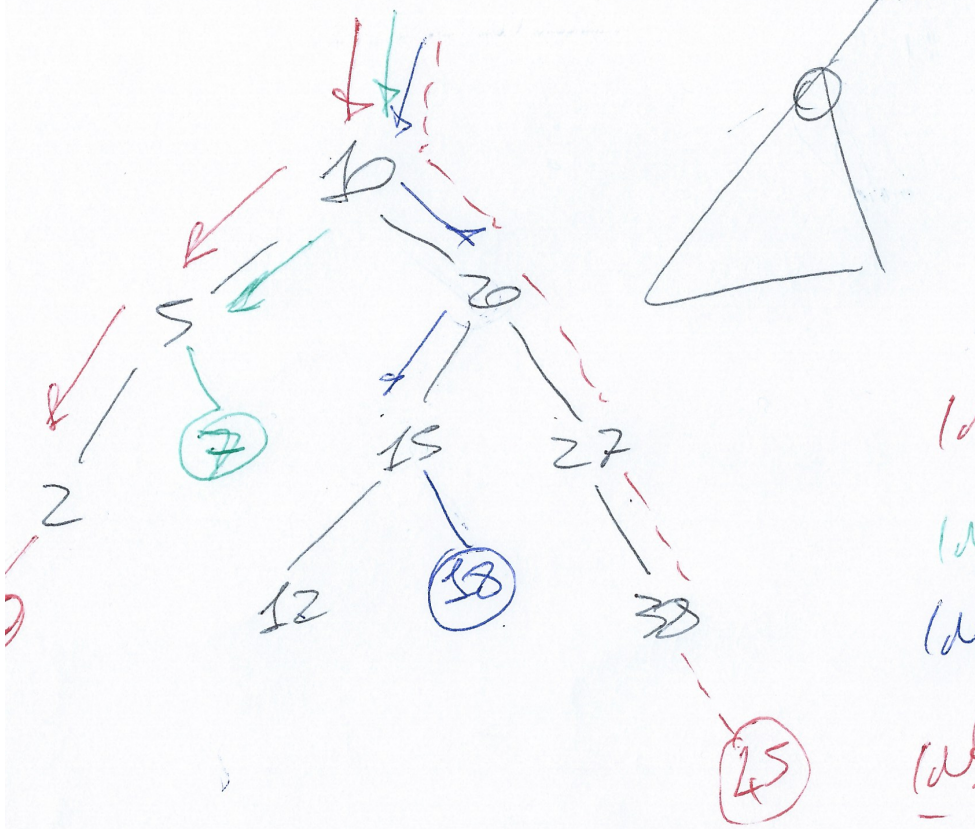
COMPLETE B-TREE

$\approx \log_2 n$



(8) INSERT IN A BST

BUILDS A NEW LEAF



INSERT(k, x)

$k < x$

⇓

INSERT(k, LEFT(x))

INSERT(1)

INSERT(7)

INSERT(18)

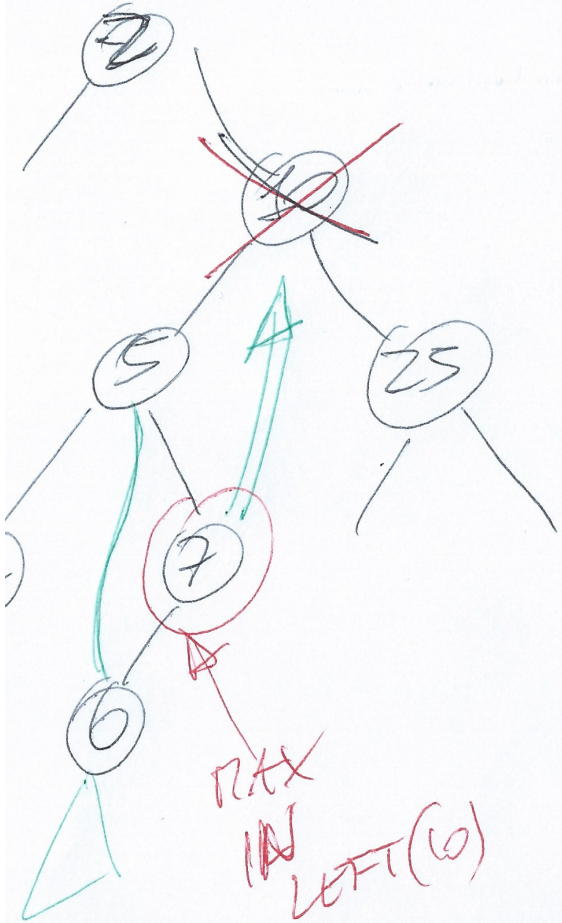
INSERT(45)

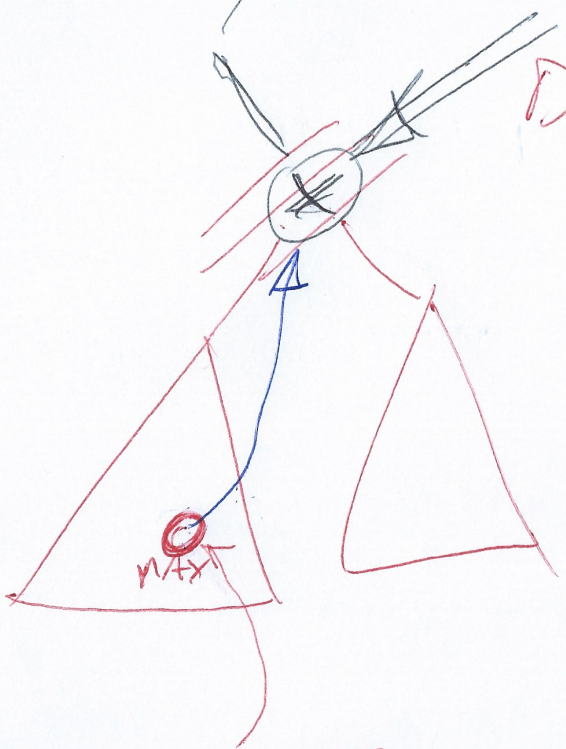
Θ(height)

(9) DELETE IN A BST

$O(height)$

DELETE(x)
- SEARCH(x)





DELETE
- x IS A LEAF

DONE

- ELSE TAKE EITHER
  MIN(RIGHT(x))
  OR
  MAX(LEFT(x))
  AND PUT IT IN THE PLACE OF x

↓

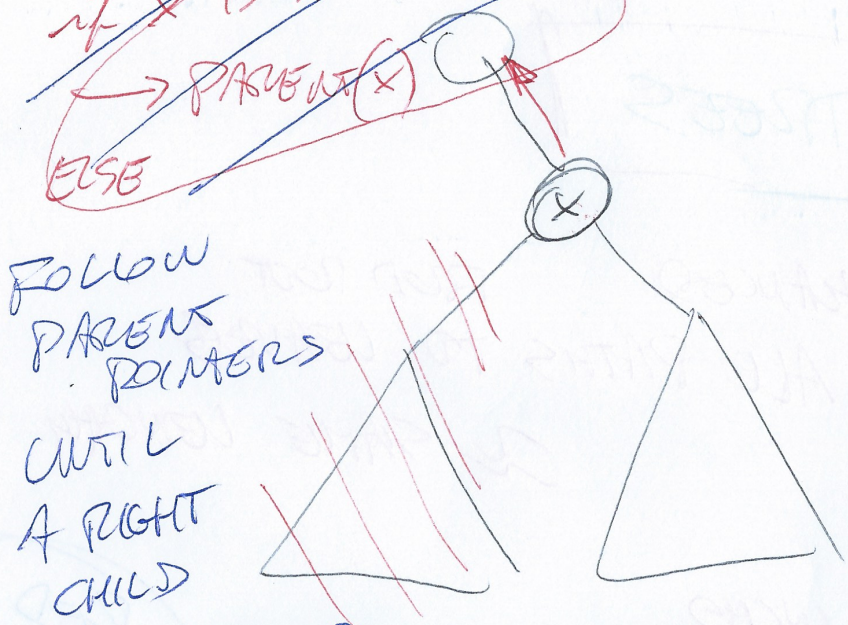
DELETION OF AN ITEM IN A SUBTREE HAS ONLY ONE CHILD (NO CHILD

(10) PREDECESSOR IN A BST

- MAX IN LEFT(x) if it exists

OR if x IS A RIGHT CHILD → PARENT(x)

ELSE

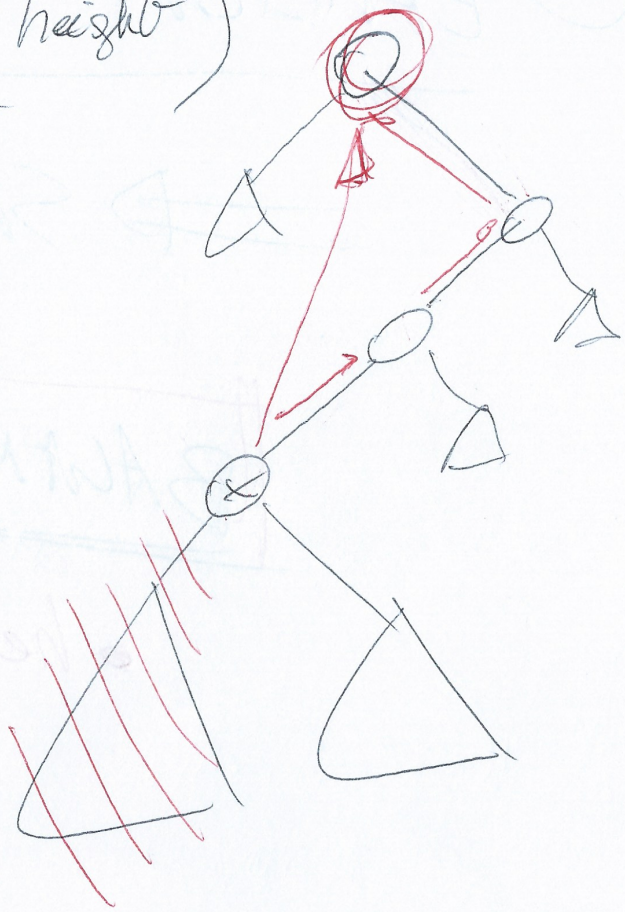


FOLLOW PARENT POINTERS UNTIL A RIGHT CHILD IS TRAVERSED

⇓

PARENT IS THE PREDECESSOR

Θ(height)



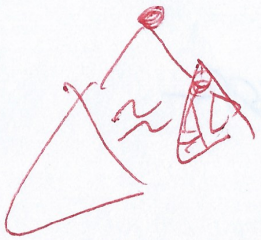
(11) EFFICIENT BST

⟹ SMALL HEIGHT $\Theta(\lg n)$

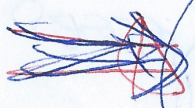
BALANCED TREES

- height BALANCED — FROM ROOT ALL PATHS TO LEAVES ≈ SAME LENGTH

- WEIGHT BALANCED



FOR ALL NODES



RED BLACK TREES

height $\geq 2 \lg_2 n$