L27

4102

4.28.2016

abhi shelat

FINAL EXAM = 4 problem.

= optional.

= all the material

– Wed May 4th

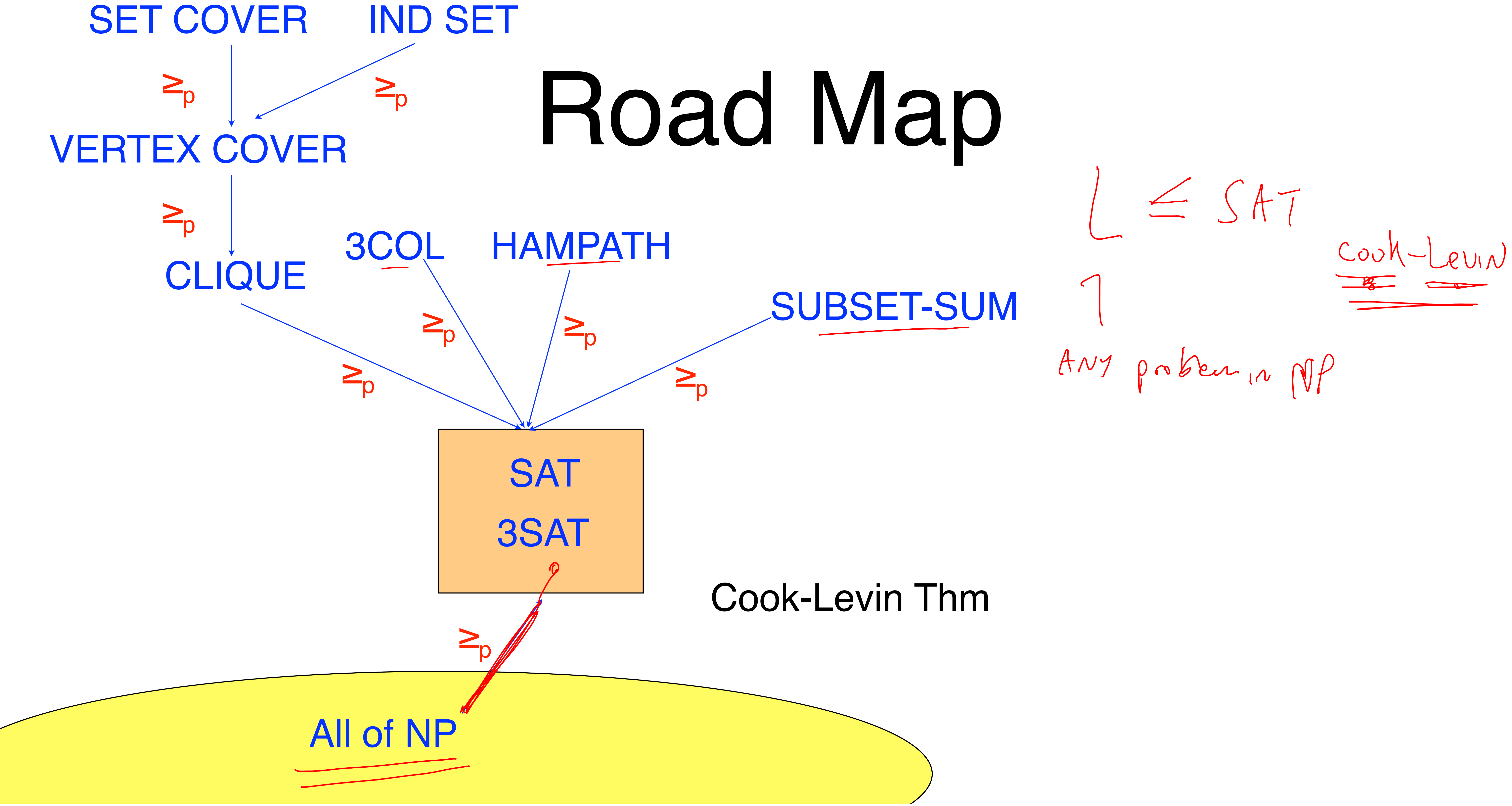
May 9th → May 11th

(32.3)

(2) ⤵ 60 – 30 – 10

↑ hw; ↑ mid + final; (classp

Road Map

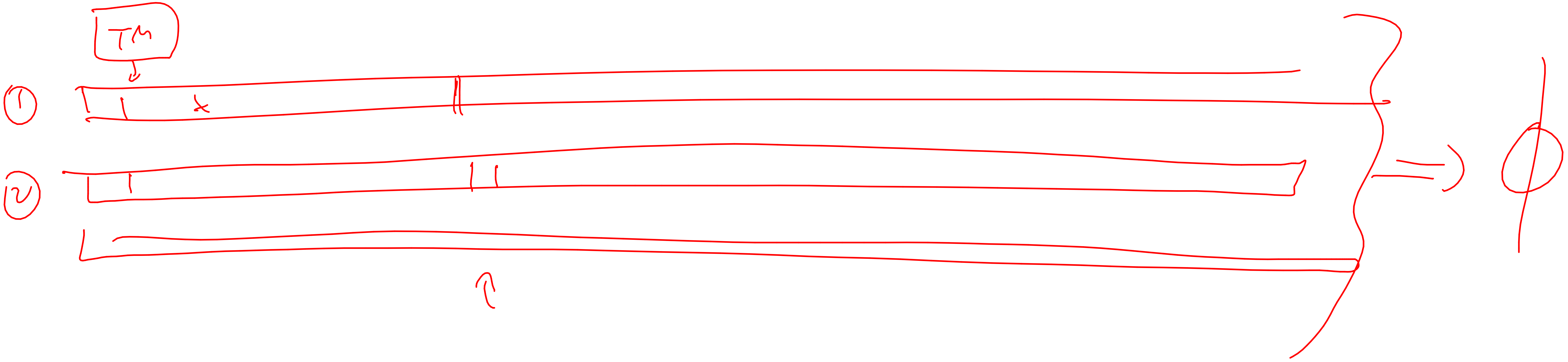

SET COVER
IND SET
≥p
≥p
VERTEX COVER
≥p
3COL
HAMPATH
CLIQUE
SUBSET-SUM
≥p
≥p
≥p
≥p
SAT
3SAT
Cook-Levin Thm
≥p
All of NP
L ≤ SAT
Cook-Levin
Any problem in NP

$L \leq SAT$ $\qquad R_L(x, ??) = 1 \iff \phi \in SAT$

$L \in NP$ is that there exists an efficient checker

$R_L(x, w) = 1$ if $x \in L$. $\qquad \exists w$ s.t. $R(x, w) = 1$

TURING MACHINE



a vertex cover of a graph is a set $C \subseteq V$
such that $\forall\ (x, y) \in E$
either $x \in C$ or $y \in C$

VC: $\{ (G, K)$: graph G has a vertex cover of size K $\}$

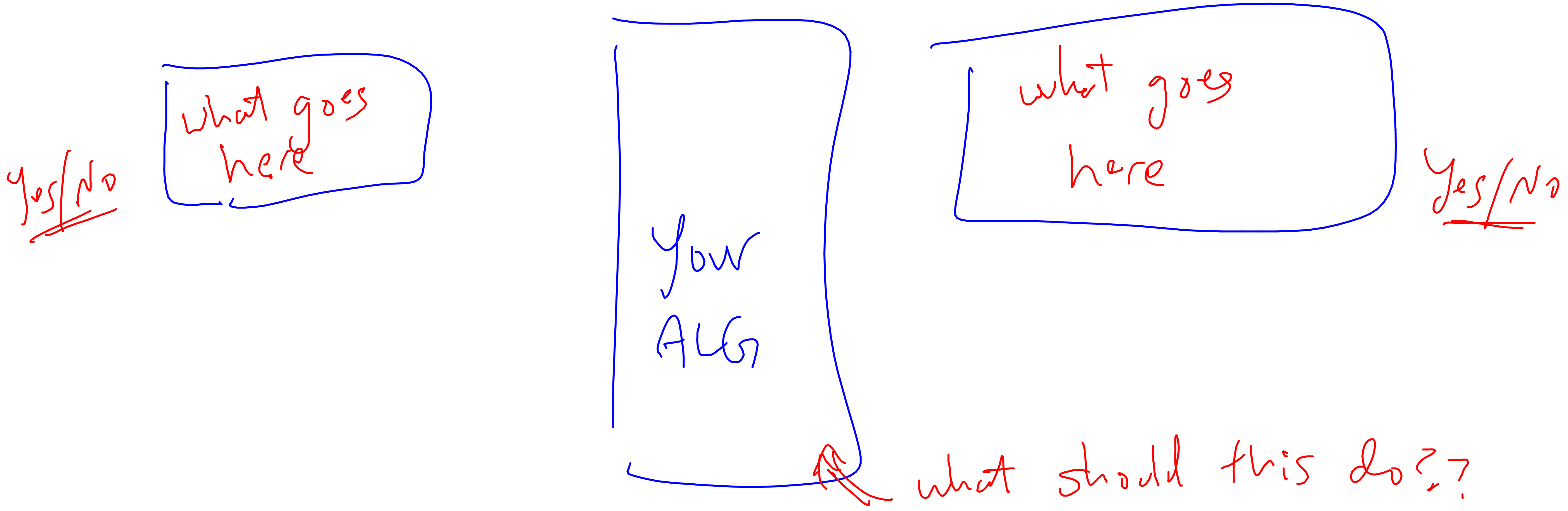
→ problem: given a pair (G, K), does $(G, K) \in VC$??

answer: yes or NO.

Andrew problem: (G, K), does graph G satisfy Andrew's wish w/ K libraries??

① Show that ANDREW is in NP.
it is easy to verify that (G, K) ∈ ANDREW

② 3SAT ≤ ANDREW or VC ≤ ANDREW



SAT is in NP.

$\phi \in SAT$ if $\exists$ an assignment A s.t. $\phi_A = TRUE$.

$R_{SAT}(x, A)$: ① Apply the assignment A to the variables

② check if the formula is true.

Dictionary

data structure

insert(·) –

search(·) –

delete(·) –

findnext(i) ⟶ it finds the lexographically next element in the Dict w.r.t. i.

e.g findnext(17) finds the smallest element in D that is > 17.

Dictionary

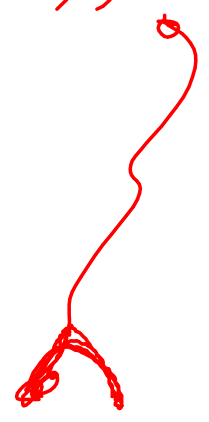
standard solution: bal bin tree   if $|D| = n$

insert → $O(\log(n))$

delete $O(\log(n))$

lookup → $O(\log(n))$

findnext $O(\log(n))$



Dictionary

standard solution: hashtable

- insert
- delete
- lookup
- findnext

could be done in $\Theta(1)$, very tricky

open addressing $\rightarrow O(\log n / \log\log n)$

findnext $\rightarrow \Theta(n)$

Dictionary

new constraint: keys belong to limited range:

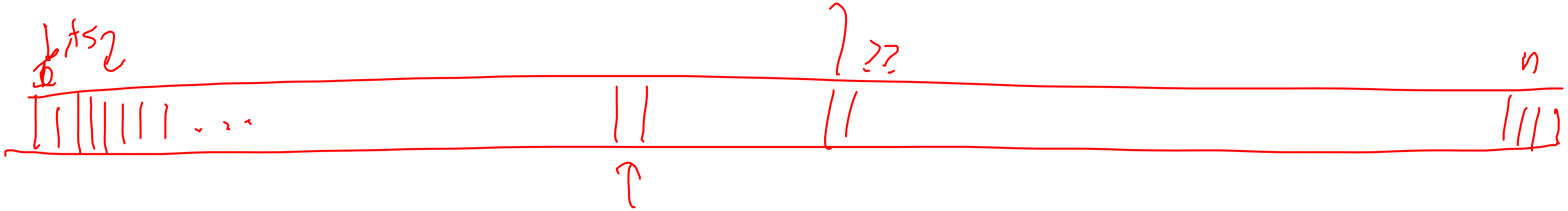
$$\{1, \ldots, n\}$$

64-bit keys

128-uid.

- insert — $\Theta(1)$
- delete — $\Theta(1)$
- lookup — $\Theta(1)$
- findnext — $\Theta(n)$

bit vector



insert - set to 1

Can we do better than O(n) findnext?

bitvector.

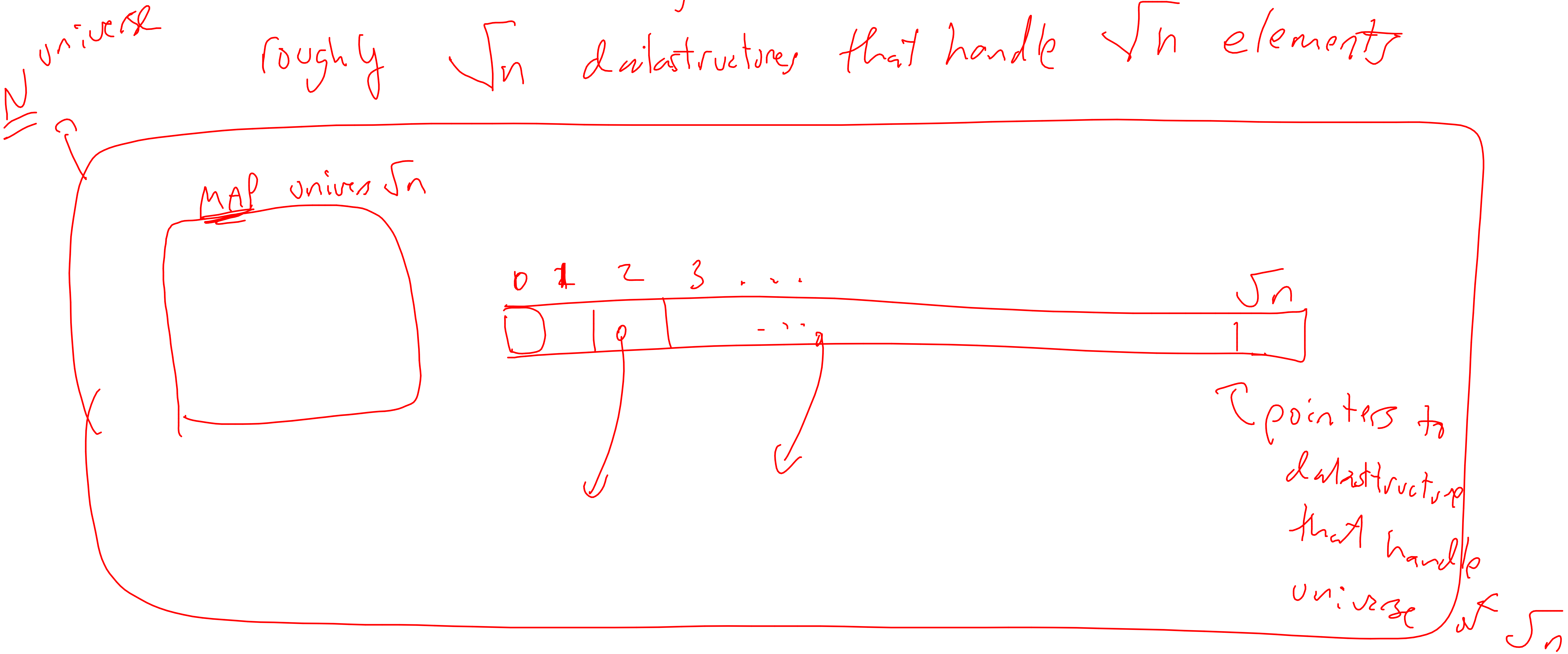
1, 1, 1, n

lg lg lg lg … lg n

loglog(n) for all operations

van emde Boas Q

the big idea: a datastructure for n elements consists of roughly √n datastructures that handle √n elements



van emde Boas Q

$VEB_{(n)}$

VEB queue

VEB$_{(n)}$

sz, min, max

VEB queue

$\text{VEB}_{(n)}$

sz, min, max

base case: 1 bit queue.

normal case:

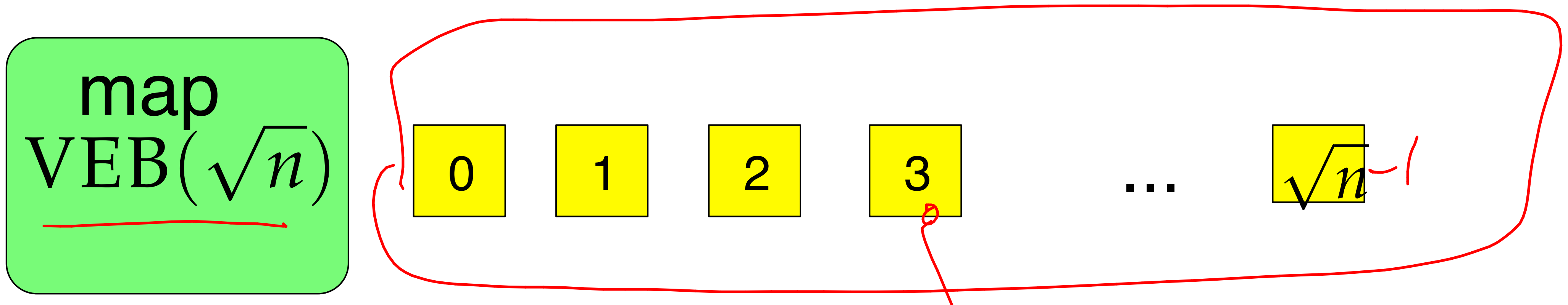
VEB queue

VEB(n) {

sz, min, max

base case: 1 bit queue. N=2, then bit vector.

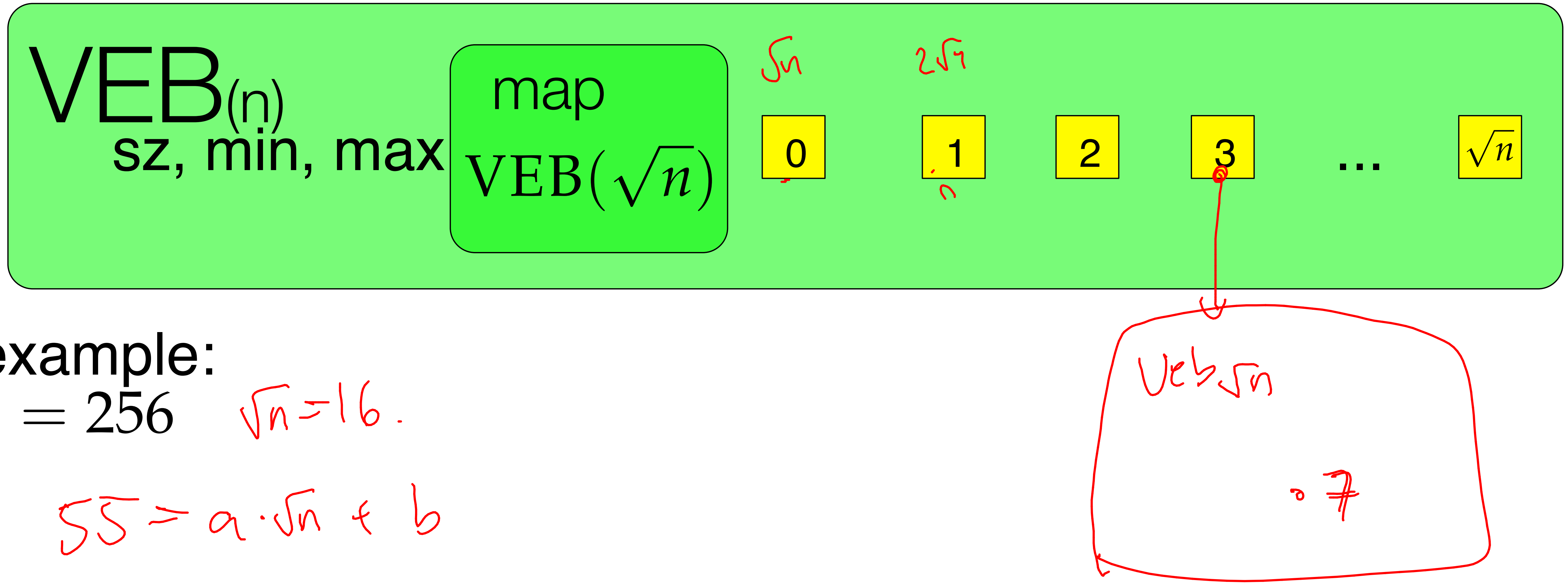
normal case:



Keep track of each pointer that is NOT NULL

pointers to VEB $\sqrt{n}$

}

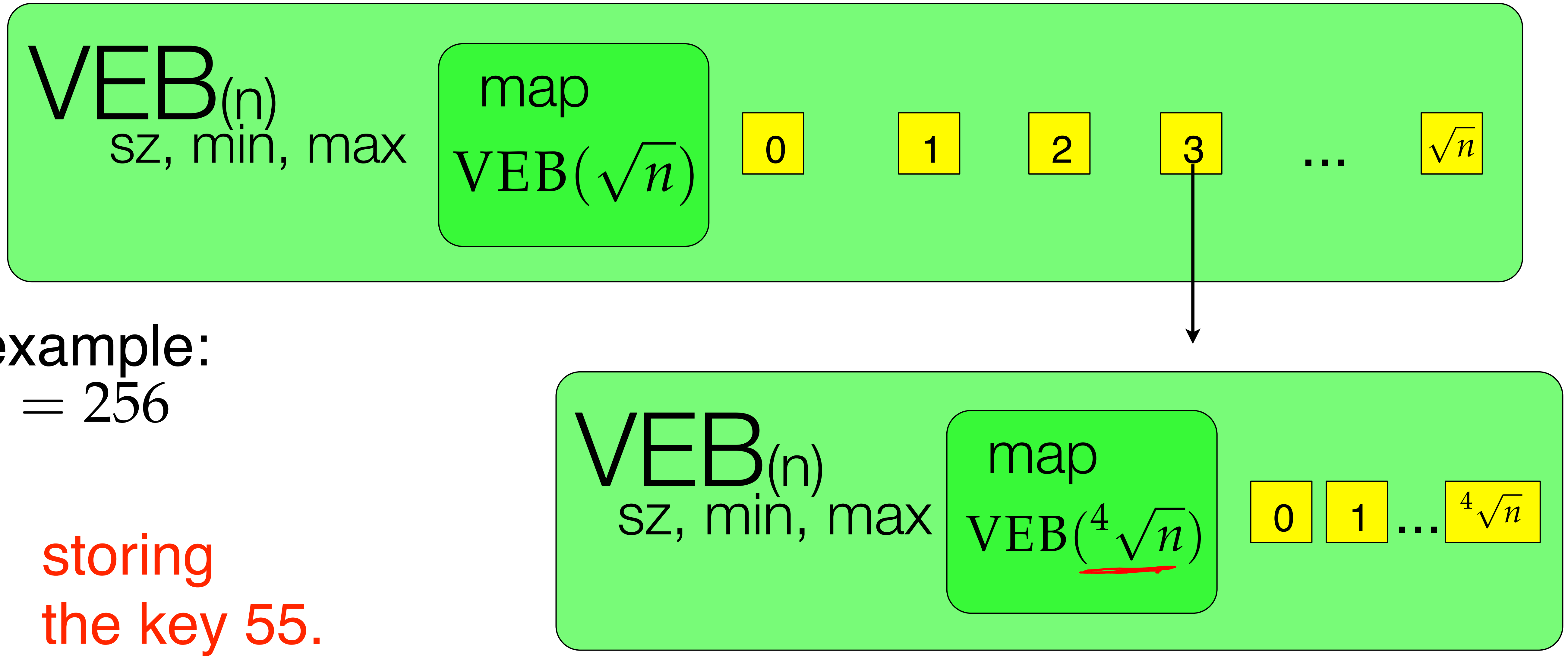


example:

$n = 256$ $\sqrt{n} = 16$.

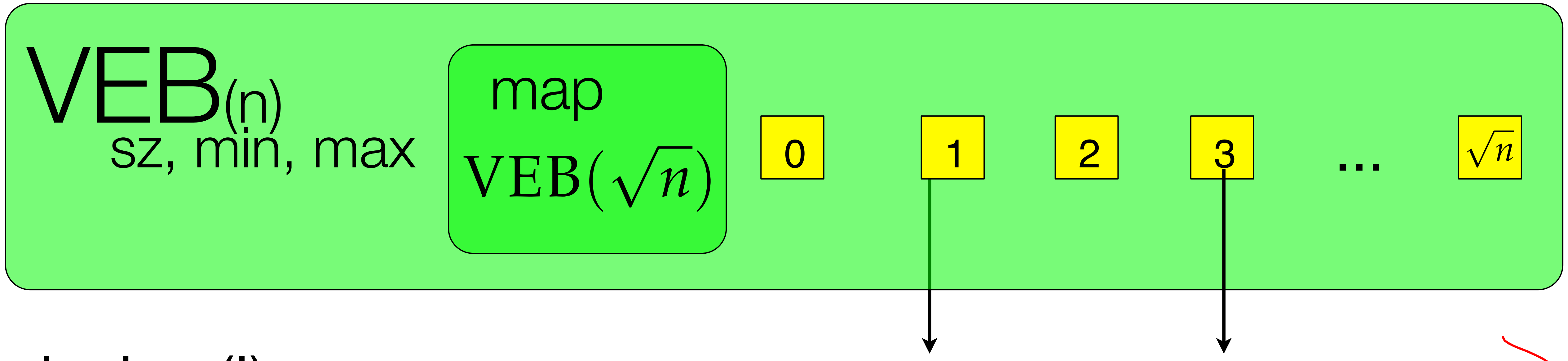
$55 = a \cdot \sqrt{n} + b$

$= 3(16) + 7$



example:
$n = 256$

storing
the key 55.



(key)

(key, value)

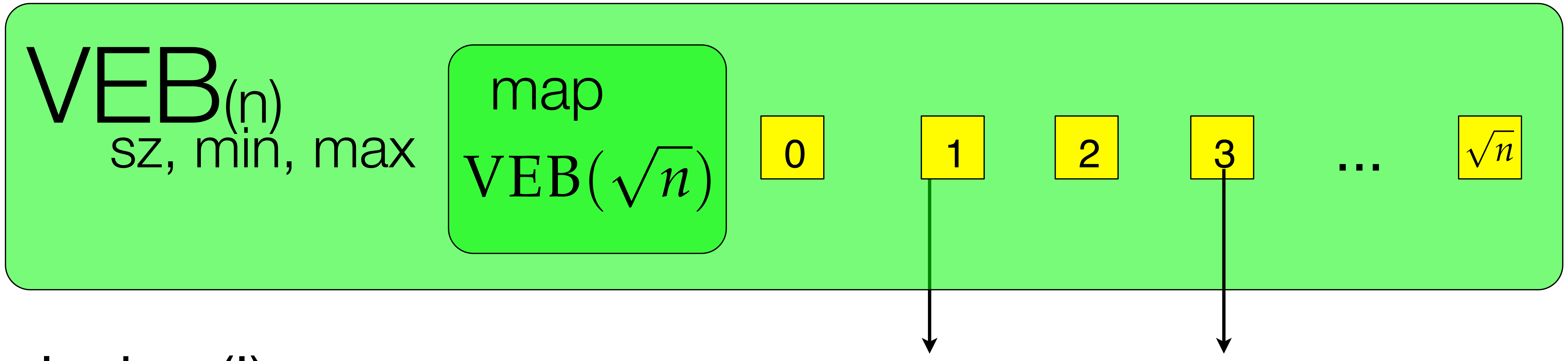
lookup(i) :

1. write $i = a \cdot \sqrt{n} + b$ $a < \sqrt{n}$ $b \le \sqrt{n}$.

   if (a is null) return false

   else return: a.lookup(b)

Running time $L(n) = L(\sqrt{n}) + 2 = \Theta(\log \log n)$



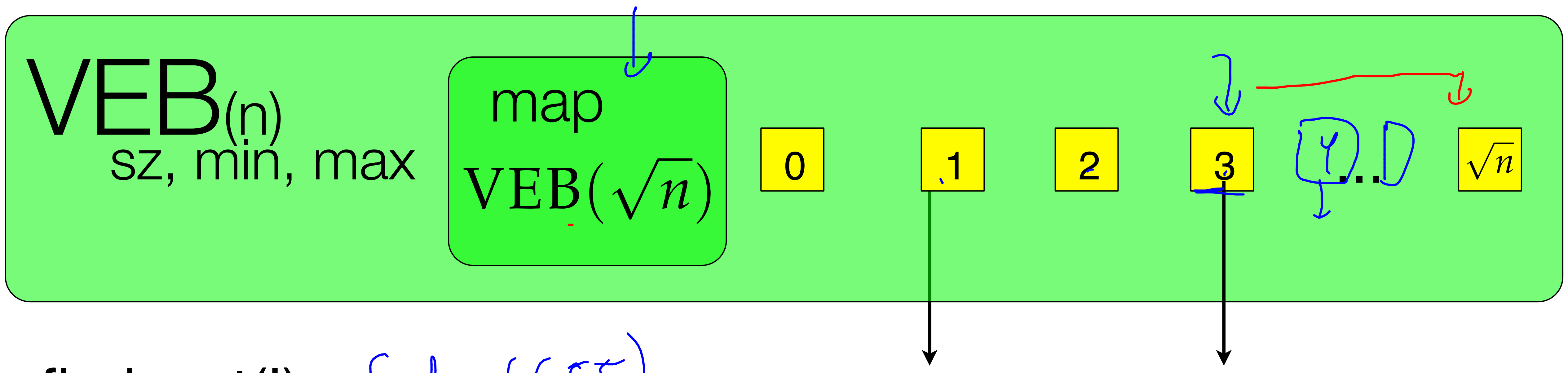
lookup(i)

write $i = a\sqrt{n} + b$

a = null ??

if size = 0 or a.size = 0 then return false

else return a.lookup(b)



findnext(i) findnext(55)

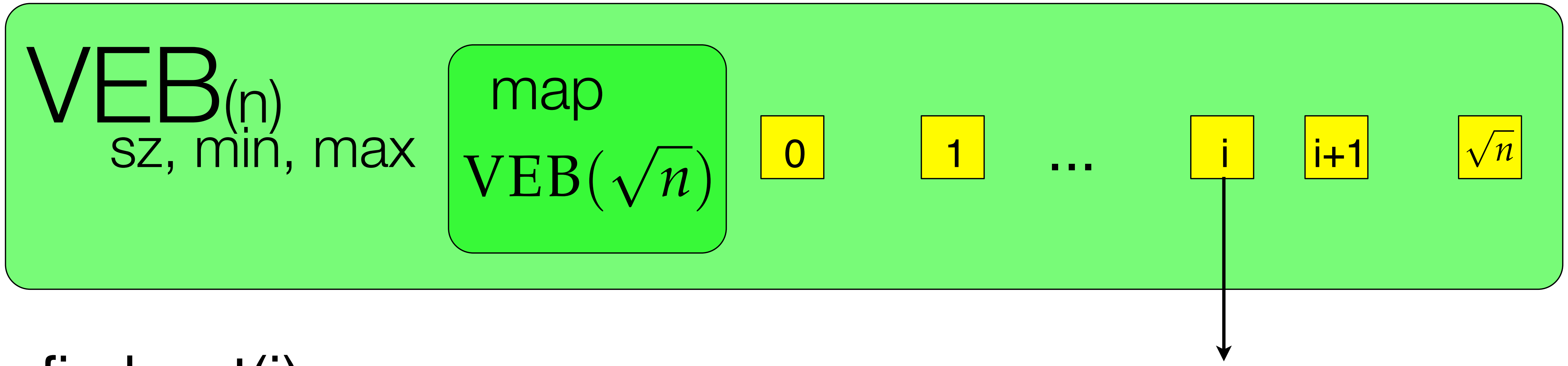
idea: $55 = 3 \cdot 16 + 7$

use max to distinguish → Case 1 if bucket 3 has the next value

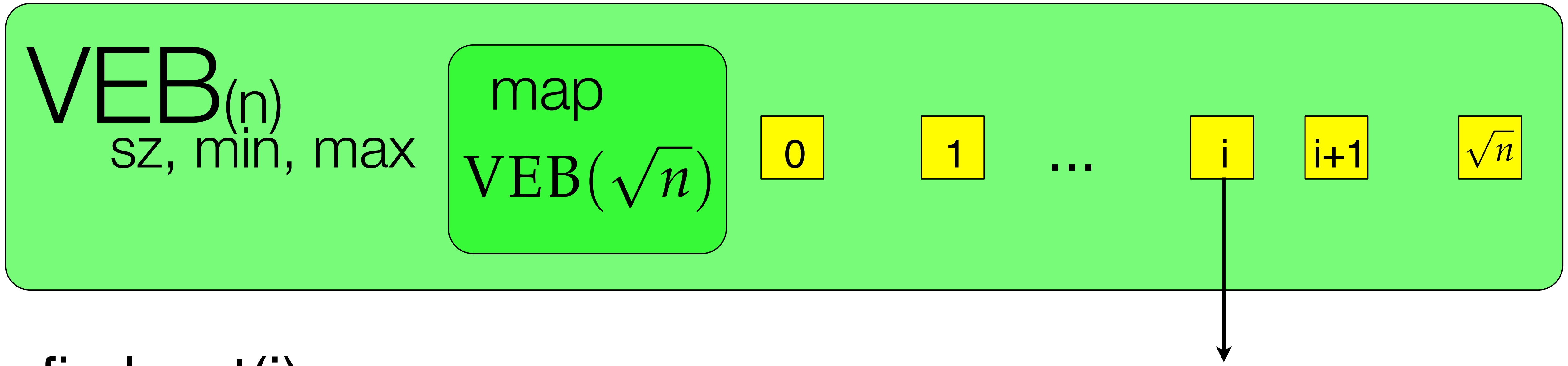
findnext(7) on bucket 3.

→ Case 2 bucket 3 does not.

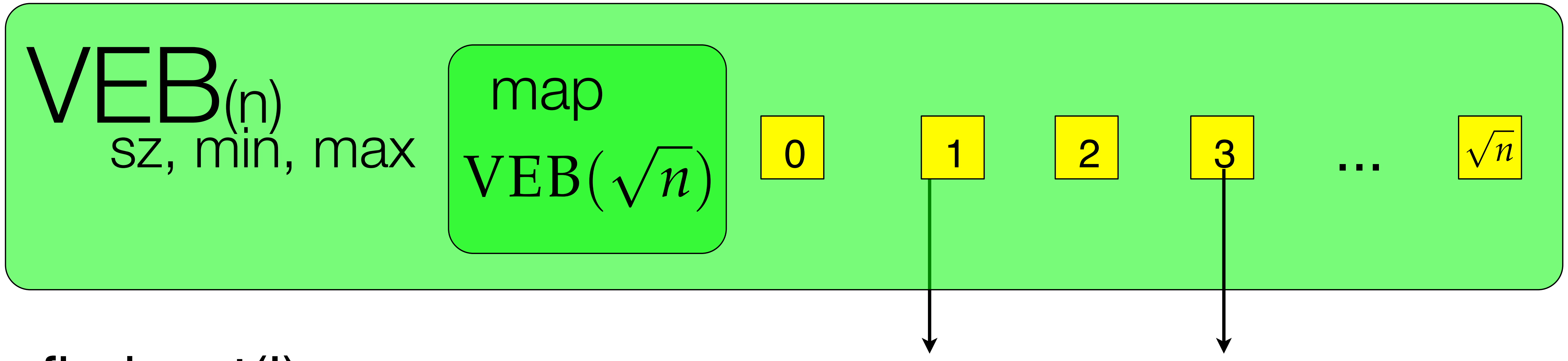
use map to findnext(3), and then return b.min.



findnext(i)



findnext(i)



findnext(i)

write $i = a\sqrt{n} + b$

<base case if size is zero>

if a.max $> b$ then $O(1)$

return a.findnext(b) $F(\sqrt{n})$

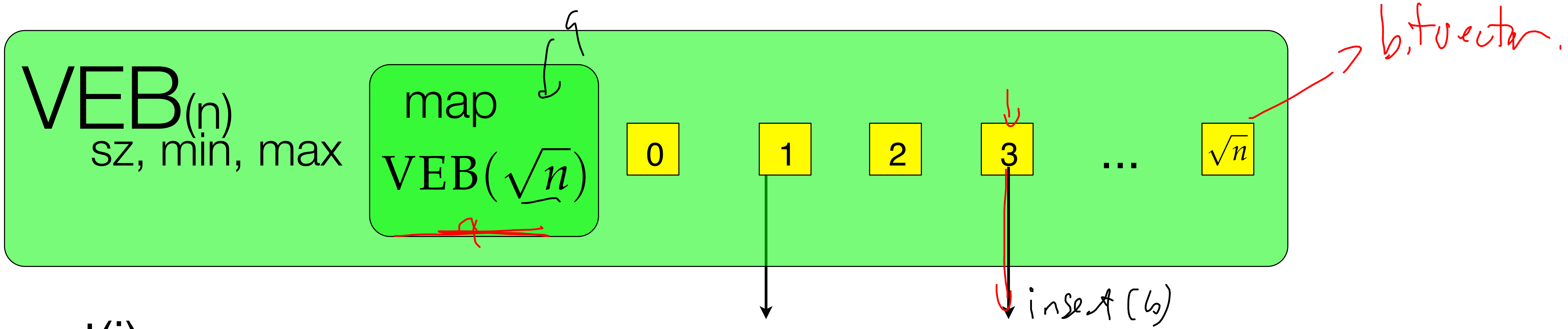
else

return map.findnext(a) .min

$F(\sqrt{n})$ $\Theta(1)$

$F(n) = F(\sqrt{n}) + O(1)$

$= \Theta(\log\log n)$



insert(i)

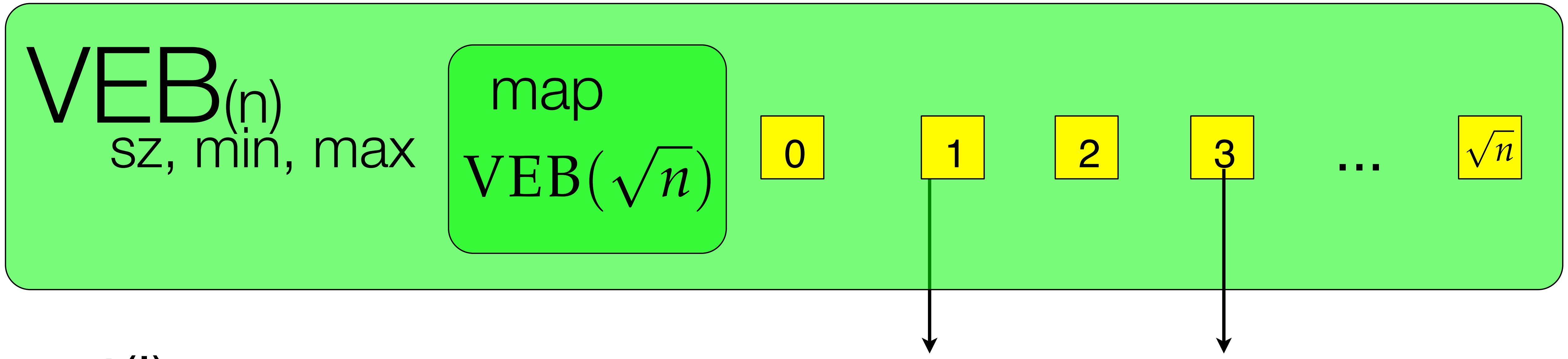
write $i = a\sqrt{n} + b$ $O(1)$

- map.insert(a) →
- a.insert(b)

$$I(N) = I(\sqrt{N}) + I(\sqrt{n}) + O(1)$$

$$= 2I(\sqrt{N}) + O(1)$$

$$\Theta(\log n)$$

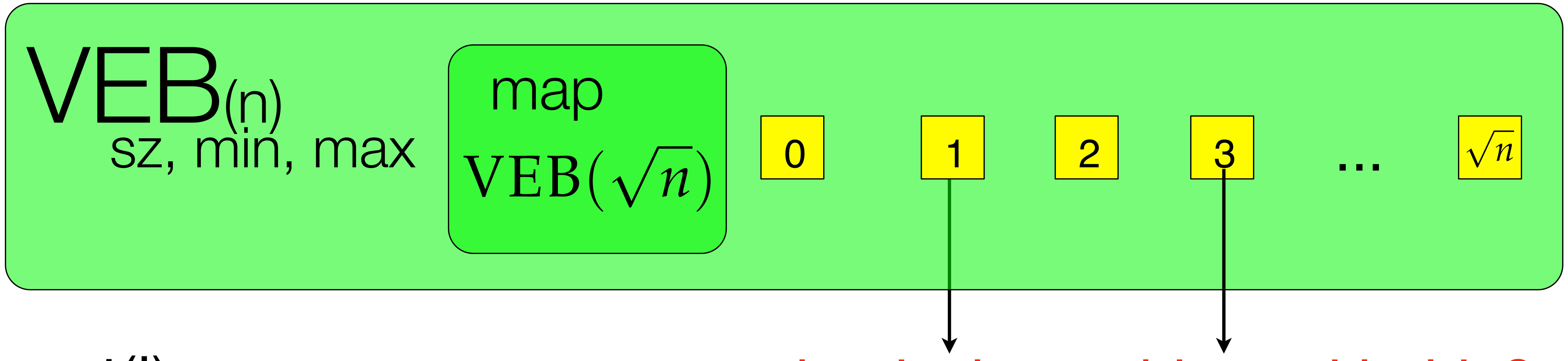


insert(i)

write $i = a\sqrt{n} + b$

a.insert(b)

map.insert(a)



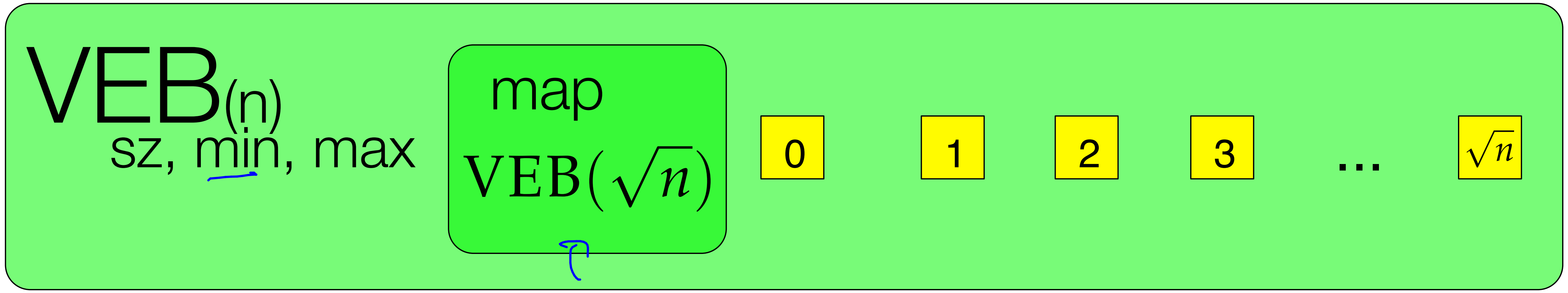
what is the problem with this?

insert(i)

write $i = a\sqrt{n} + b$

a.insert(b)

map.insert(a)



insert(i)

write $i = a\sqrt{n} + b$

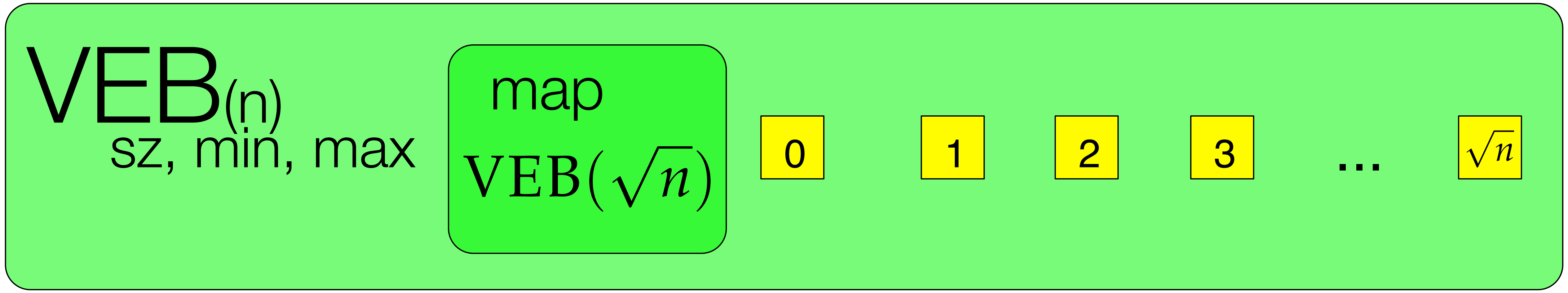
a.insert(b)

map.insert(a)

what is the problem with this?

how can we get around the problem of inserting twice?

answer: LAZY inserts. how many times do we need to insert into MAP?

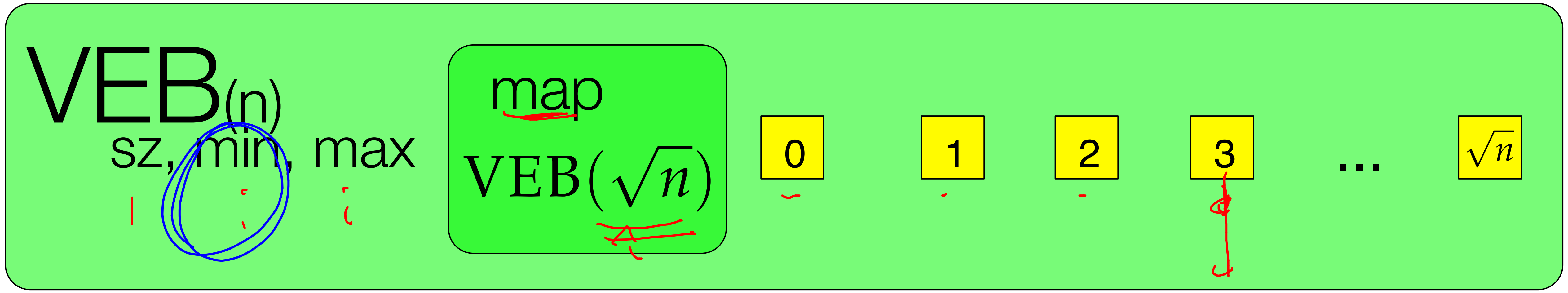


insert(i)

write $i = a\sqrt{n} + b$

if sz==0 then

else



$I(n) = I(\sqrt{n}) + \Theta(1)$

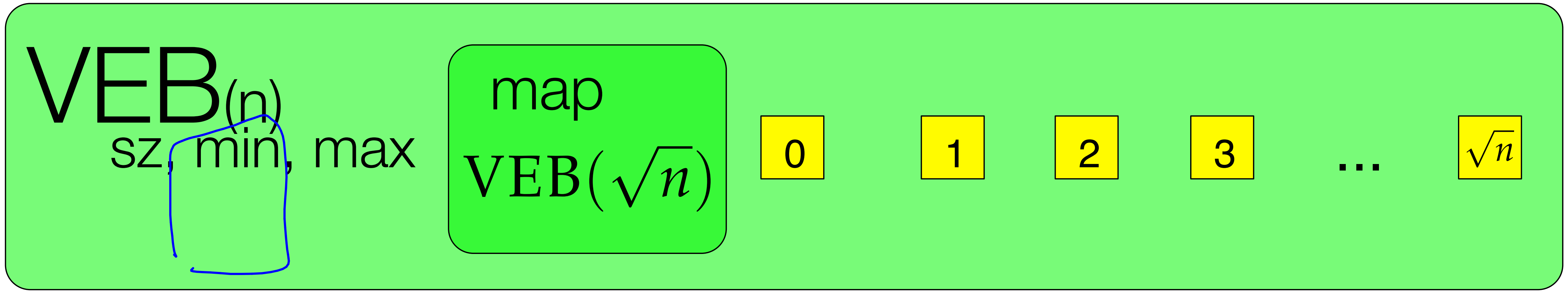
```
insert(i)
  if sz==0 then update sz=1,min=max=i
  else
     if min>i swap(i,min)
     write  i = a√n + b
     if [a].sz==0  then  map.insert(a)
     [a].insert(b)
     update sz, min, max
```

first insert $O(1)$

size=1
min=max=

if a was empty $I(\sqrt{n})$ $\Theta(1)$

else $I(\sqrt{n})$



lookup(i)

write $i = a\sqrt{n} + b$

if size==0 return false

if i==min return true

else return a.lookup(b)