Hashing (chp. 5)

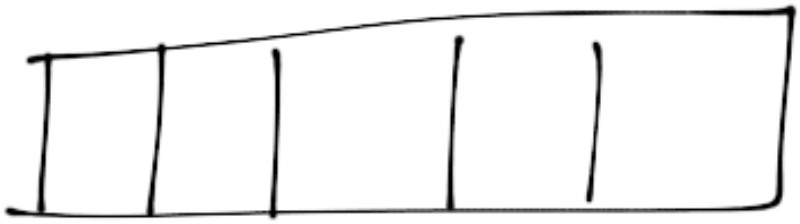
- Data structure is the Hash Table
- Basic array but one that supports $\Theta(1)$ insertion, deletion, find (avg. constant time)

- given key (data element, e.g. a number) compute its index into the array via a hash function

---

- e.g.

hash table of size 5



0 1 2 3 4 — index

given any number,

e.g. 19203, 32762, etc

simple hash function is

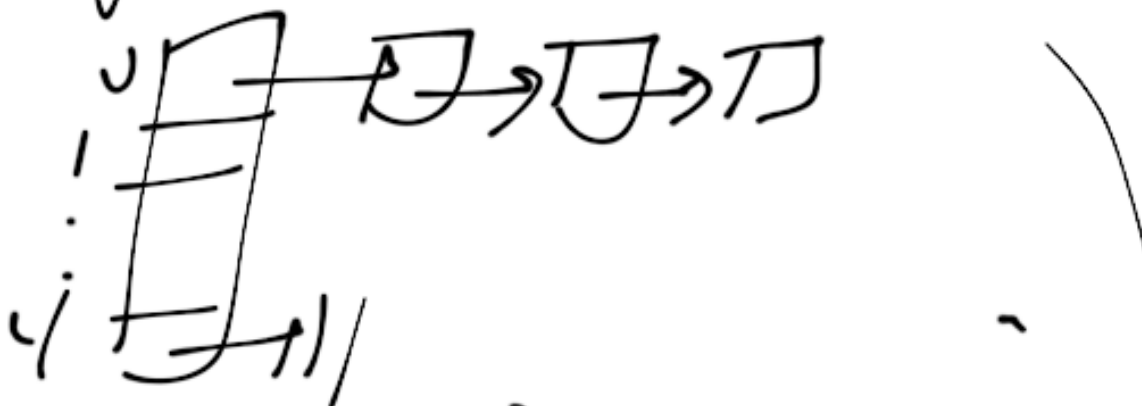
% (mod)

any number % 5

will give index ∈ [0, 4]

- two major concerns:
  - if no. of elements
    >> table size
    ⇒ take care of collisions

    augment array to be
    array of lists



- performance depends on size of hash table & good hash function

- 2nd major concern is the hash function, e.g., if 25 and all my numbers are multiple of 5, always store it [0]

- a good candidate for hash table size i, function is the prime number, e.g.

1, 3, 5, 7, 11, 13, 17,

19, 23, 29, 31, 37,

41, 43, 47, 53, 59,

... 10,007

- text gives good hash function for string keys

$$h = \sum_{i=0}^{KeySize-1} key[KeySize-i-1] * 37^i$$

via Horner's rule

$$h_k = K_0 + 37K_1 + 37^2 K_2$$

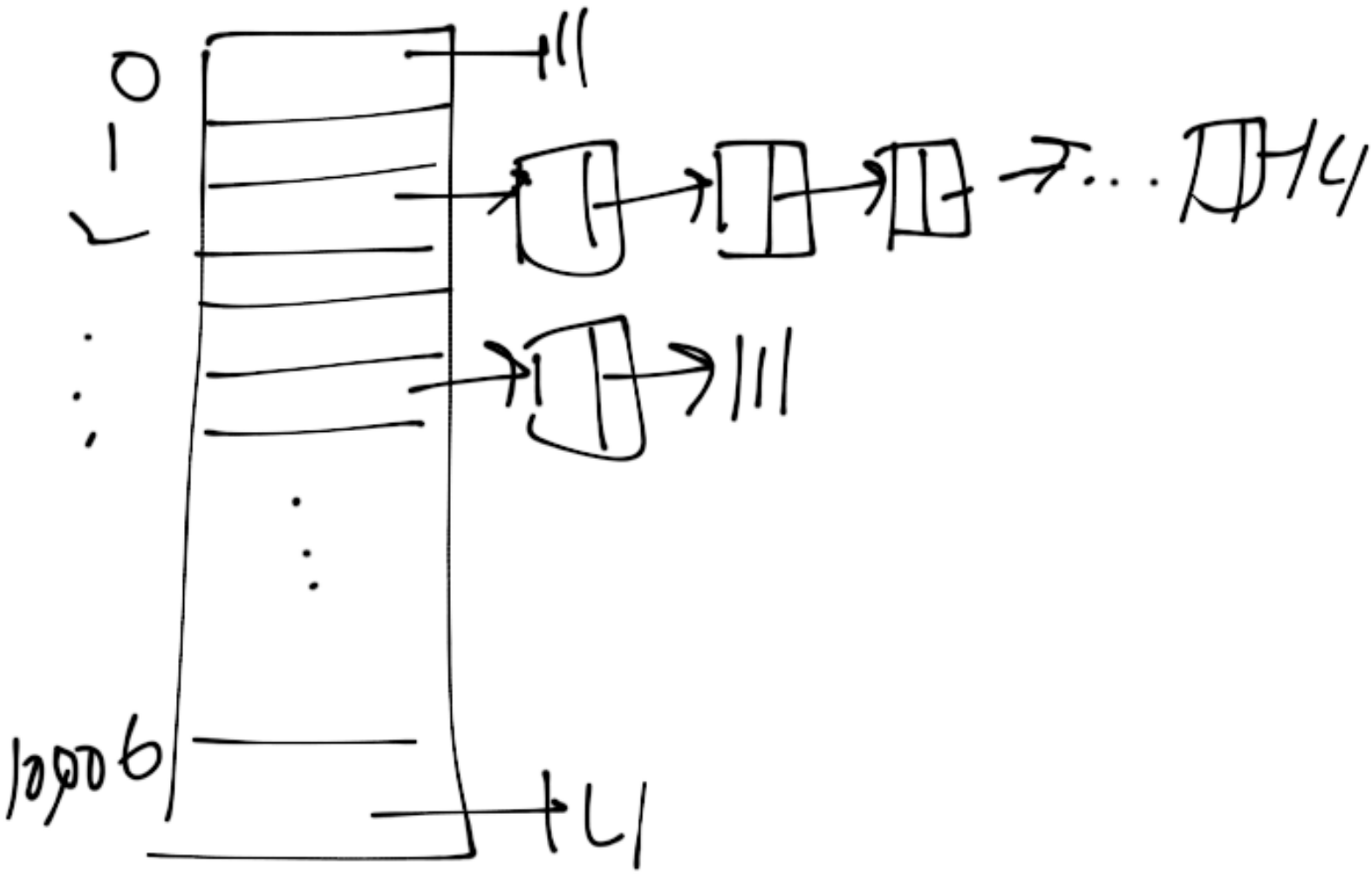
$$= ((K_2)37 + K_1)37 + K_0$$

e.g.

```
h = 0
for (int i = 0; i < key.length(); i++)
    h = 37 * h + key[i];
h = h % tableSize
if (h < 0) h += tableSize;
```

implementing the hash table
(doesn't seem to exist
in C++ STL)

```
class hash_t
{
    private:
    std::vector<std::list<T>>
                elements;
};
```



- Analysis is tricky:
  - need to consider average case $\Theta()$

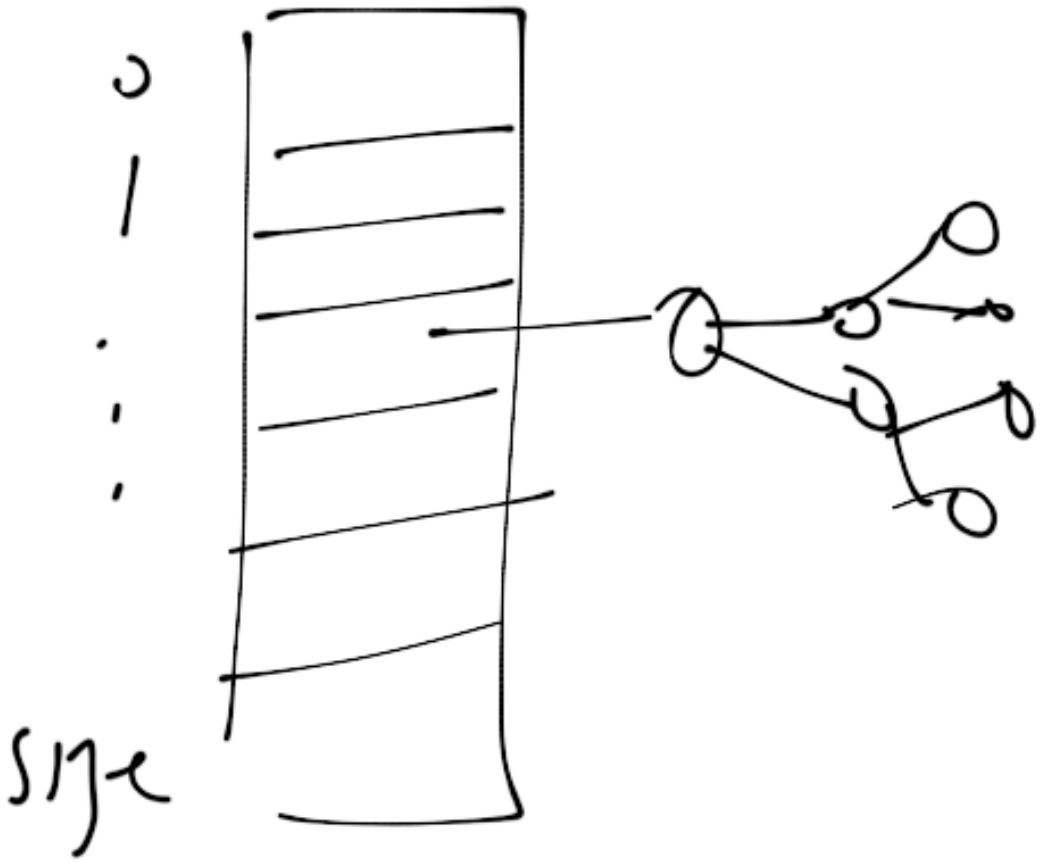
e.g. $\Theta(1)$ + $\Theta(\frac{n}{size})$??

hash fn search

- other approaches:

  hash table of hash tables

  hash table of trees



— average-case analysis ≡ amortized analysis

Midterm

Q1: Copy const: new object

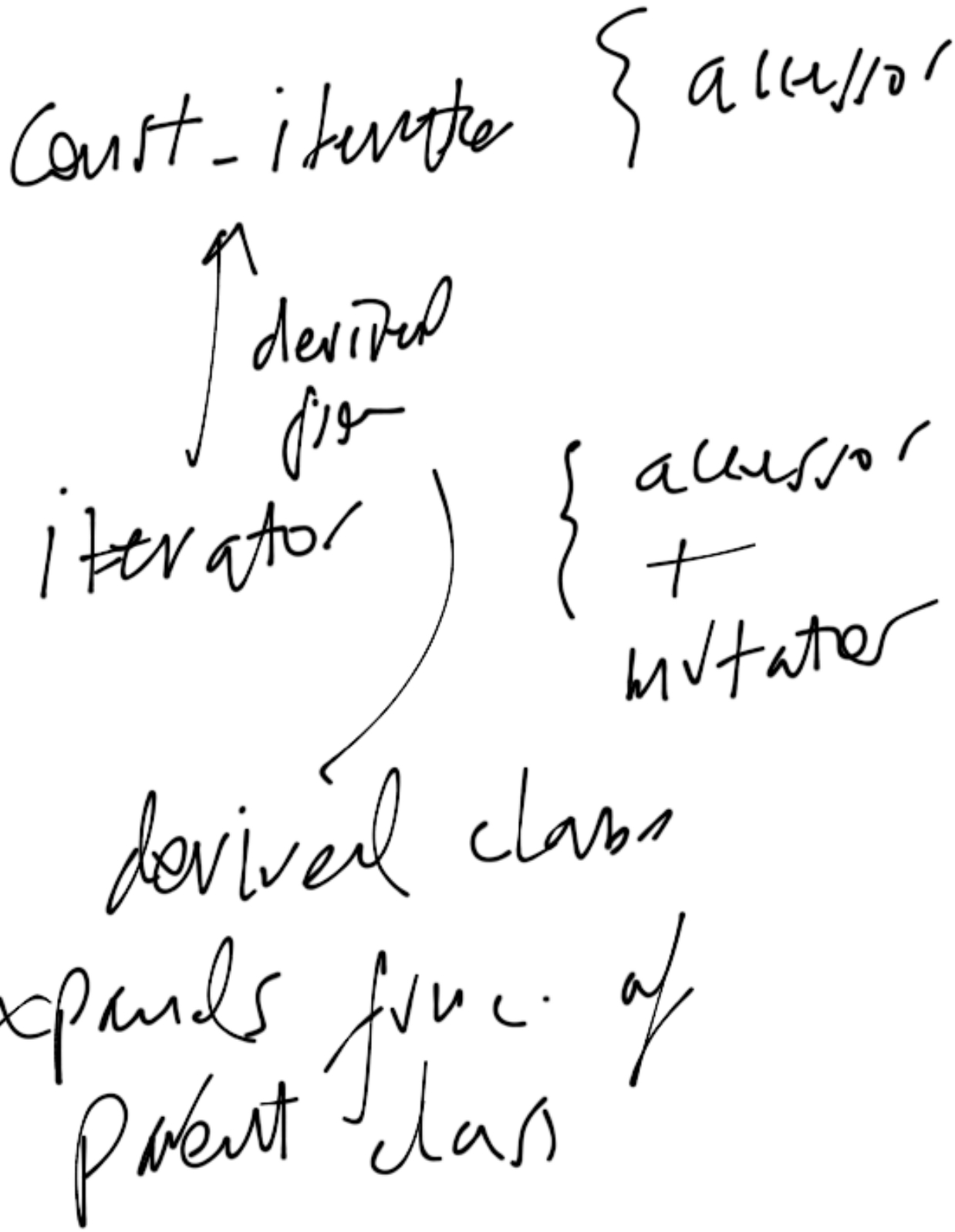
Copy ass.: Copies from
existing to existing
obj;

Q2:

A3.operator=(

A1.operator+(A2));

Q1 :


const_iterator
accessor
derived
for
iterator
accessor
+
mutator
derived class
expands func. of
parent class

Q4:

$O(T(n)) = f(n)$ |

$\exists\, c, n_0 : T(n) \leq c F(n),$

$\forall\, n \geq n_0$

upper bound

(worst case)

Q5. (see §7.9)

$\Theta(T(n))$ = avg. running time

$\Omega(T(n))$ = min. run time (lower bound)

if $\Theta = \Omega$, an algo. performs optimally

if $O = \Omega$, alg is optimal

Q6

Alg-1:

optimal als on
average, but
dependents under
some condition
e.g., binary search tree

Alg 2:

e.g., insertion sort

Alg. 3

Impossible alg:

average case better than

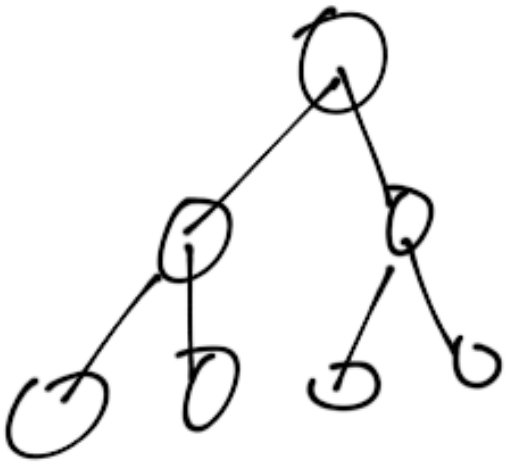
best case

Alg 4:

optimal

AVL

Q7



level	# nodes
0	1
1	2
2	4
⋮	⋮
i	2^i nodes per level

if n levels

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

leaves 2^n

Q8.

Soln. A

$O(\lg n) \times n$

$= O(n \lg n)$

+

$n \times O(\lg n)$

$= O(n \lg n)$

$O(n \lg n)$

Which better

Soln. B

$O(1) \times n$

$= O(n)$

+

$O(n) \times 1$

$= O(n)$

+

$O(n \lg n)$

$O(n \lg n)$

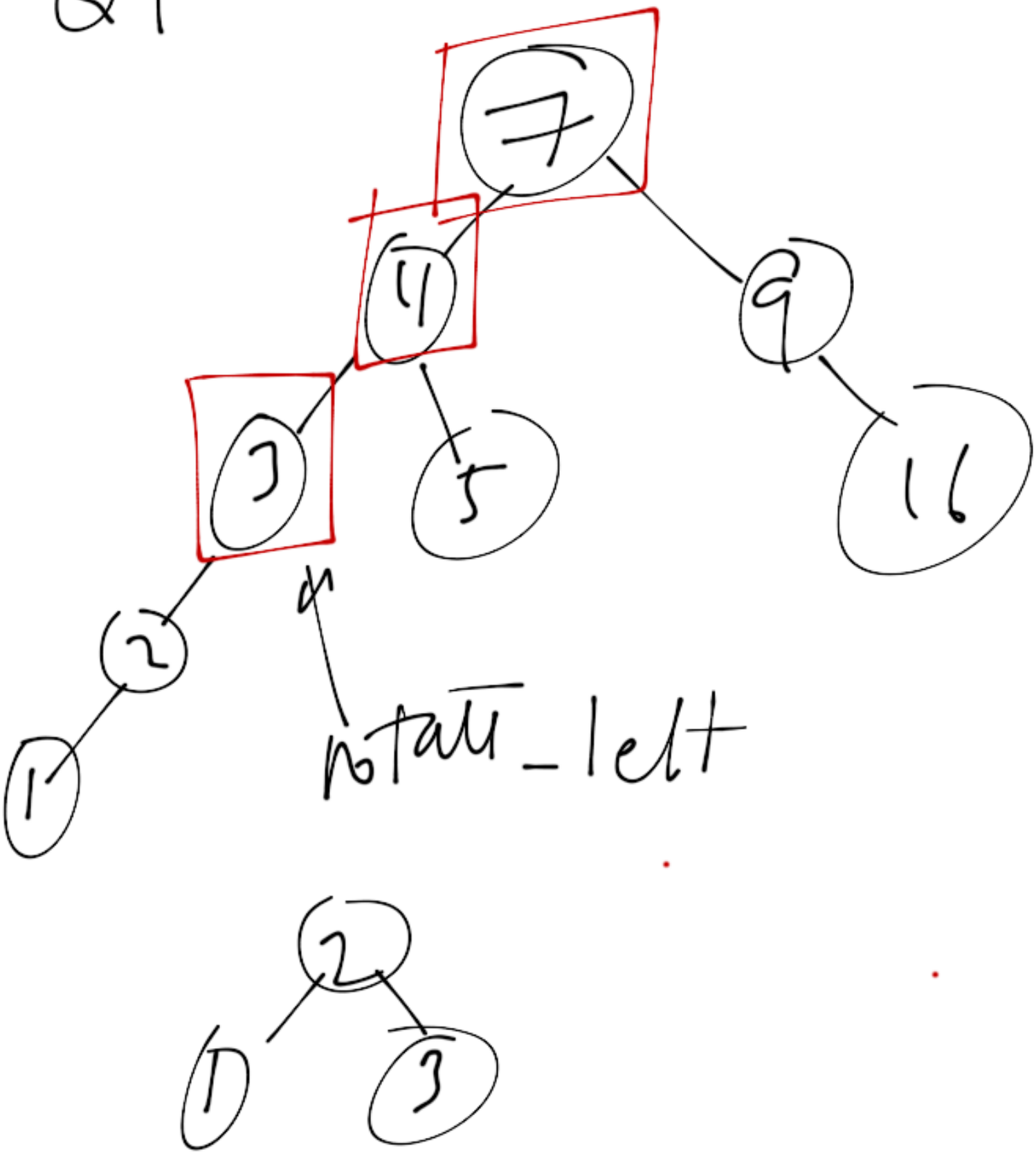
Stat. B can hit

better because

$O(2n) < O(n \log n)$

in the intermediate steps

Q9:



min: 5

max: 98

avg: 65.7 ± 21.8