CS3000

5/21 - Weds :)

Admin

- exam #1 tmw
- HW2 due 5/22 9pm
- HW3 out tmw, due 5/30 9pm

Agenda

1. Linked List
2. Hash Table
3. Collisions / Run-Time
4. Exam Prep

1. Linked Lists

Can we do better?

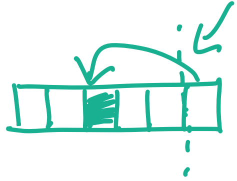
1. D&C
2. data structure
   - trees (pointers)
   - heap
   - stacks (underlying array)
   - queues
3. ...

plain array A, length n

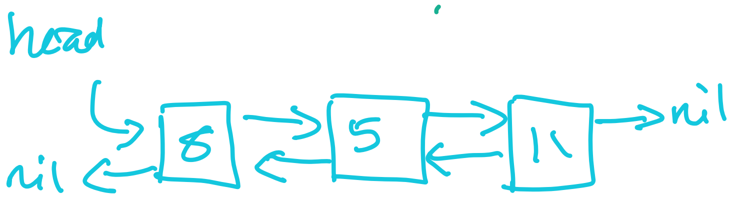
starting point for run-time

w.c.

	unsorted	sorted
find	$\Theta(n)$	$\Theta(\lg n)$
insert	$\Theta(1)$	$\Theta(n)$
remove	$\Theta(1), \Theta(n)$	$\Theta(n)$



linked list:
- array vibes
- no positions
- traverse using pointers



(assume unsorted)

each node n:
- n.key
- n.next (could be nil)
- n.prev (could be nil)

Find — basically linear search

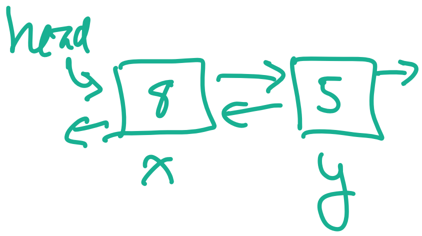
- no positions
- not sorted

- visit every node in order
- compare n.key to key

$\Theta(n)$

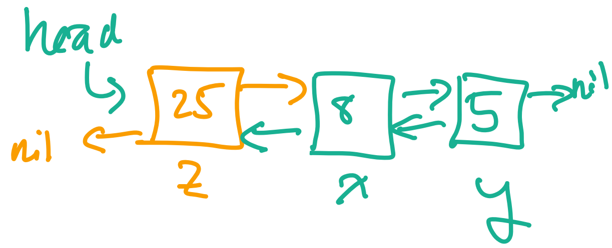
insert

new node z



updates:

- head points to z
- z.prev is NIL
- z.next is x
- x.prev is z

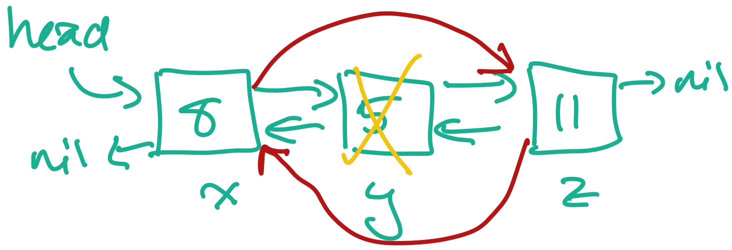


$\Theta(1)$

(and can grow arbitrarily)

remove, once found

- used find to retrieve node to remove
- or, given the node



ex) remove y

x.next to y.next

z.prev to y.prev

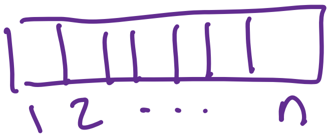
(deallocate y)

$\Theta(1)$

+ maintains original order!

2. Hash Table

↳ underlying array
- array uses
- positions



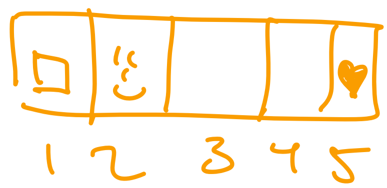
(regular array) unsorted
- order of inserts = order in array

⟨5, 7, 3, 1, ...⟩

position is determined by the $\boxed{\text{key}}$,
<u>not</u> the order in which we insert

key	position
☺	2
♥	5
□	1

insert(♥)
insert(□)
insert(☺)

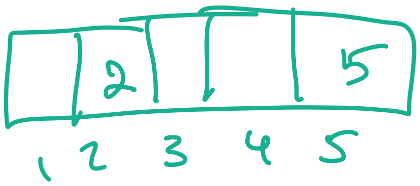


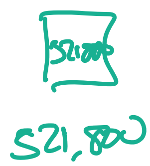
Can this be faster than a regular array?

* do keys have to be distinct?

simplifying assumption: keys are ints ≥ 1
- map key to position (hash function)
  ↳ key = position?

insert(5)
insert(2)
insert(521,000)



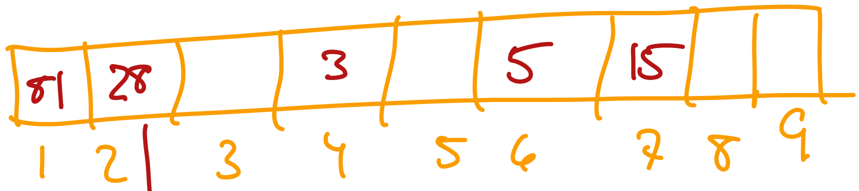


↳ exceeds capacity of array
or, little used capacity

Problem to solve #1: key → position gets positions that are too big!
- still have ints ≥ 1 keys
- restrict range of possible values: $\boxed{\text{mod}}$

ex hash function $h(k) = k \bmod 9 + 1$

insert:	pos:
5	6
28	2
81	1
3	4
15	7
19	9



Something already there!

By intro'ing mod
- solve problem #1
- now, problem #2!

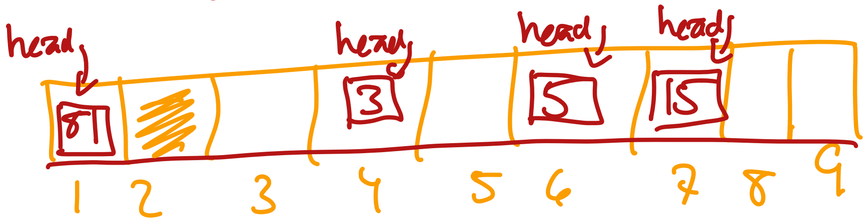
Collision

3. Collision + Run-Time

↳ hash function should minimize collisions, but can't avoid them!

- Has to put multiple things at same position in an array?

Linked List at each position in array
- "chaining"
- element first to its position, create a one-element linked list
- in case of collision, add to existing Linked List at that position



Assumptions:
- keys don't need to be pos. ints
- hash function is $\Theta(1)$

insert

- hash key $\Theta(1)$
- create/add to linked list $\Theta(1)$

find

- hash key $\Theta(1)$
- find in linked list $\Theta(n)$

worst case: all n elements hashed to the same position :(

- not luck
- bad hash function!

$\Theta(m)$ where m is length of linked list

remove

- hash key $\Theta(1)$
- remove it from the linked list
  - ↳ once found: $\Theta(1)$

what makes a good hash function?

- don't put everything in same position
- restricts the range
- predictable/deterministic
- don't waste space / use all available positions
- all positions are equally utilized
- work on different data types

real life: w/ a good hash function!

$\boxed{\text{expected } \Theta(1) \text{ for find :)}}$

HashTable IRL:

$\Theta(1)$ insert, remove, find :)

4. Exam #1

Logistics

- 11:40-1:20 5122 (be on time!)
- take your time!
- don't write on backs of pages
- pseudocode on there for speific questions
- 8.5x11 inch cheat sheet, one side only
- no phones, calcs, devices, notes etc.
- time displayed on screen

Questions

- prove correctness
- iterative run-time
- recursive run-time

given pseudocode,
- compute T(n)
- (solve T(n))
- give bound

given T(n),
- solve and
- give bound

- no master method
- use iteration method to solve recurrences
- put recursive sequence in closed form
- given pseudocode, what does it do?
- no space complexity :)
- write pseudocode, would be straight forward
- growth of functions (O, Ω)
- what does this data structure look like?

  stacks, queues, trees - simple

arrays, heaps — more involved

- given an algorithm we know, what does it do?
- High level: linear search, binary search, insertion sort, mergesort, quicksort, heapsort