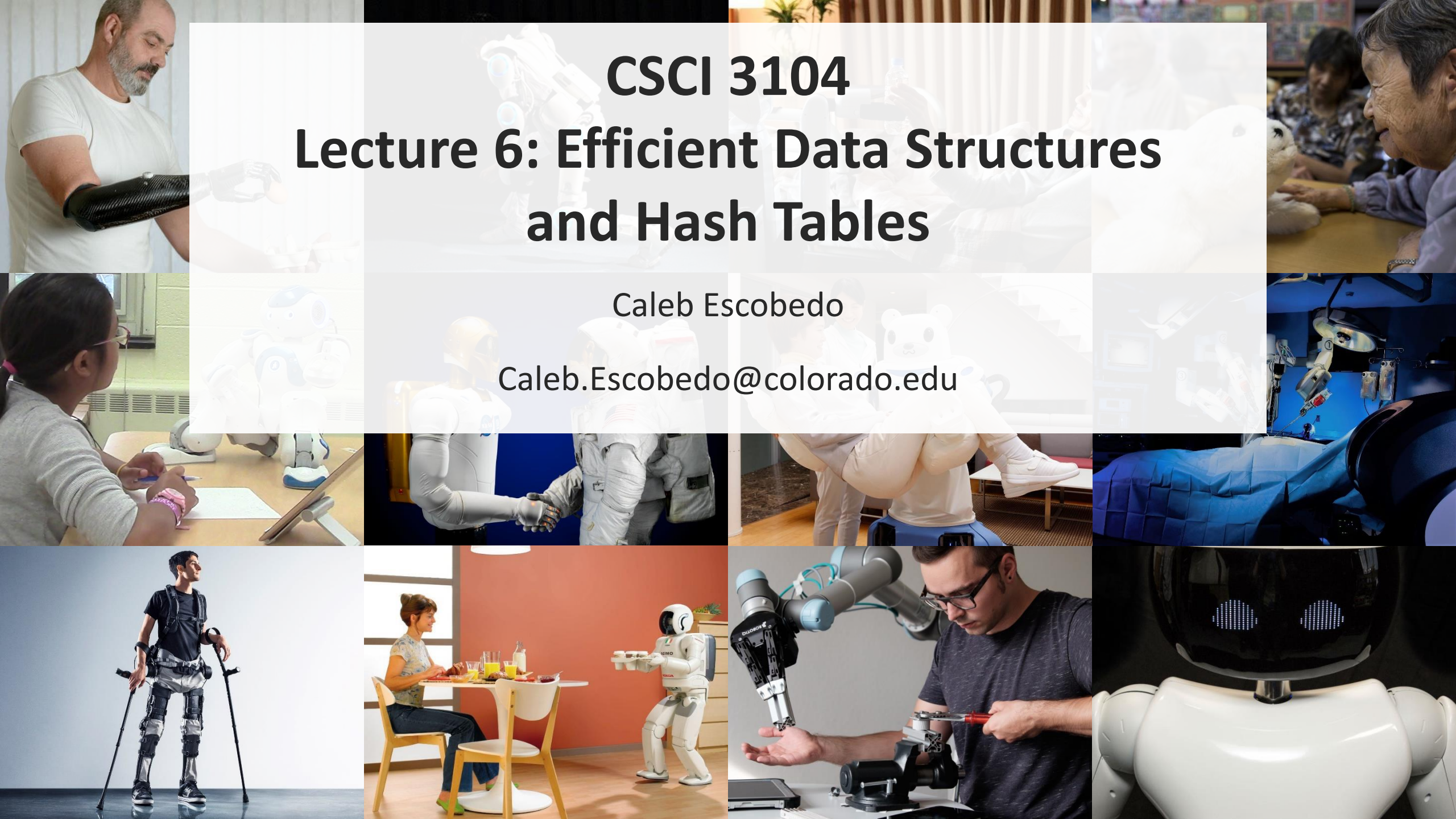


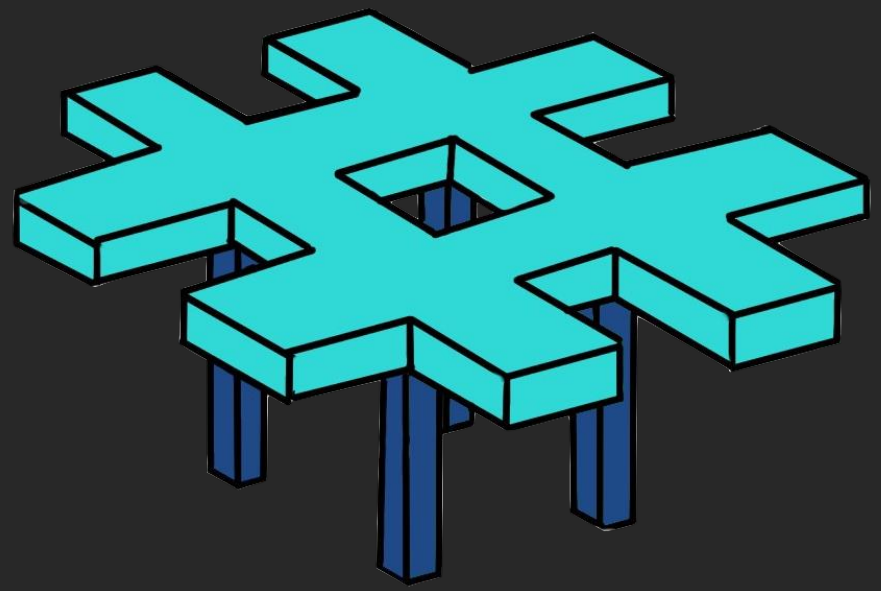
CSCI 3104
Lecture 6: Efficient Data Structures and Hash Tables

Caleb Escobedo

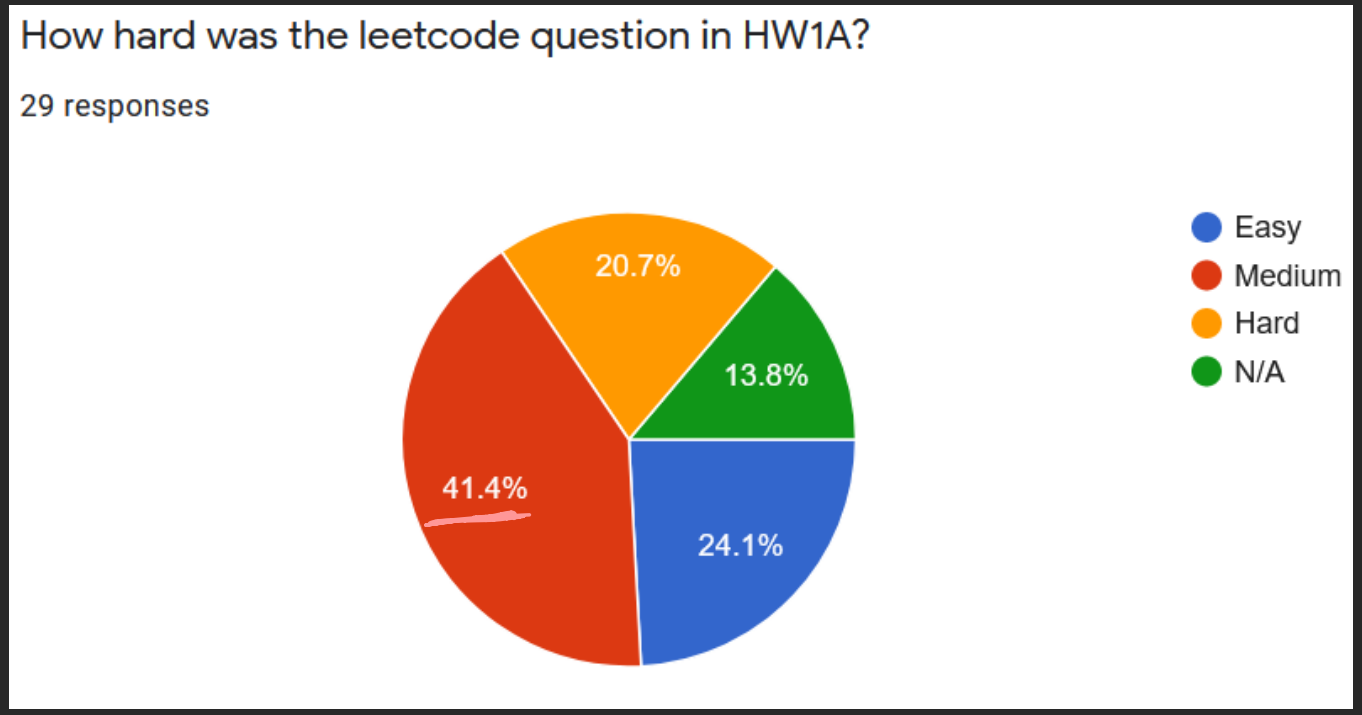
Caleb.Escobedo@colorado.edu

Lecture Outline

- Poll review/Administrative
- Efficient Data Structures
- Hash Table
  - Introduction
  - Perfect Hash
  - Chained Hash
- Birthday Paradox
- 4 Example Problems for HW2A



Poll Review!

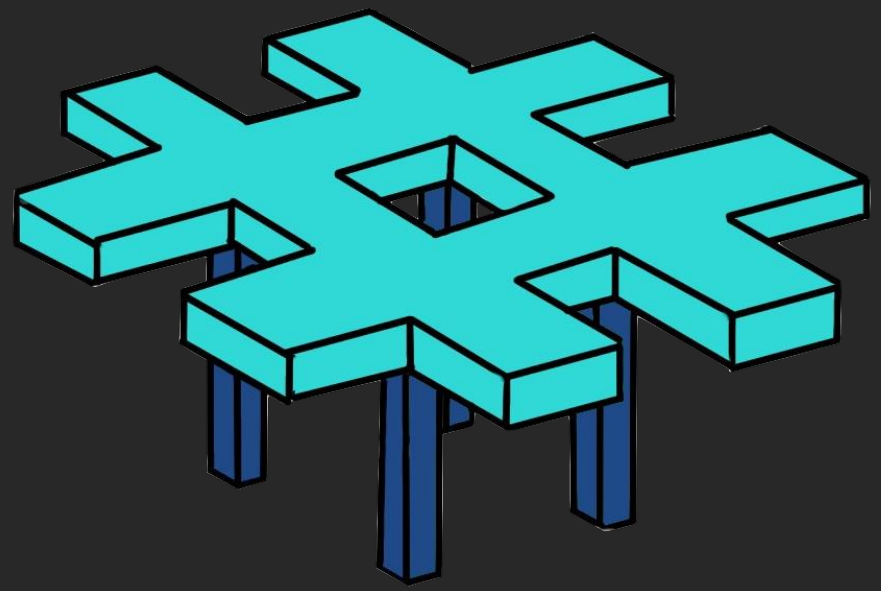

How hard was the leetcode question in HW1A?
29 responses
20.7%
13.8%
41.4%
24.1%
Easy
Medium
Hard
N/A

Administrative

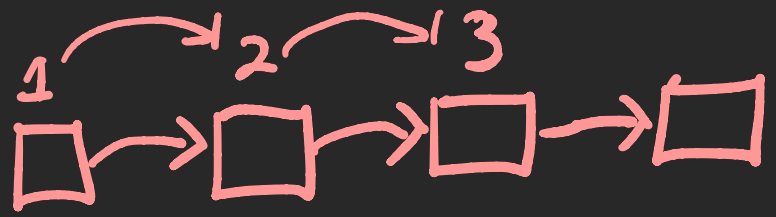
- HW0 grading not means to be harsh – refunding points for extra information
- Great job turning in HW1A – more students then HW0
- Exams will be similar in difficulty to homework
- You will get some credit for extra credit *if* you do it 2.5pt
  - Do it first if you don't know how to optimize
- More practice with Master Theorem – 3 problems today at end of class

Lecture Outline

- ~~Poll review/Administrative~~
- Efficient Data Structures
- Hash Table
  - Introduction
  - Perfect Hash
  - Chained Hash
- Birthday Paradox
- 4 Example Problems for HW2A

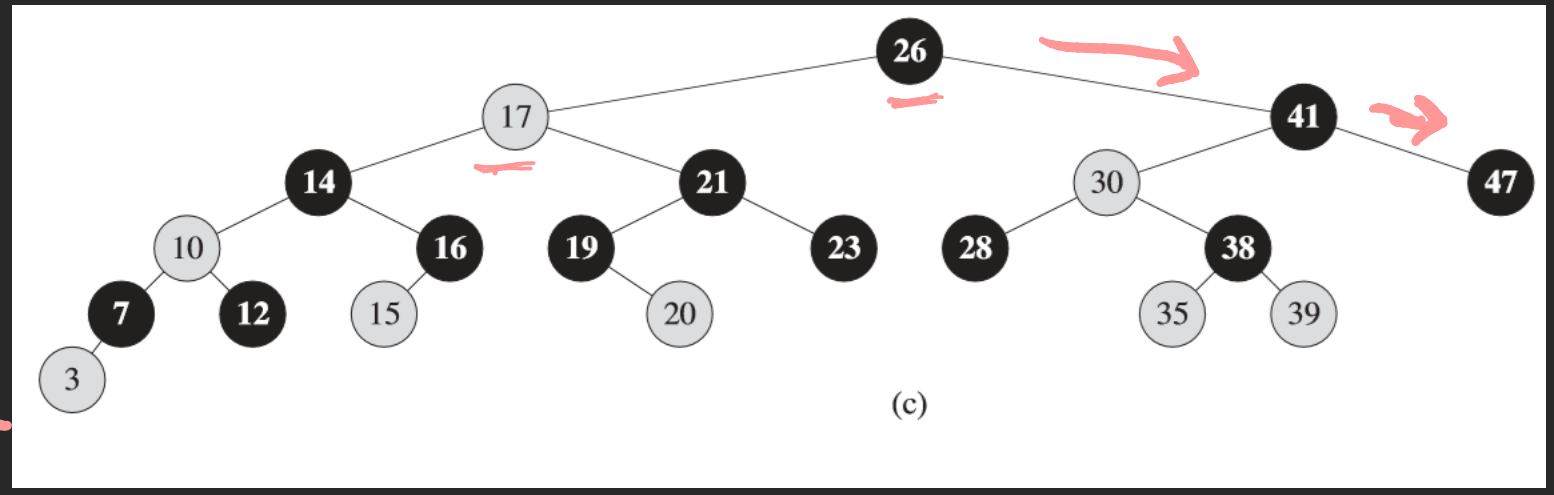


Efficient Data Structures



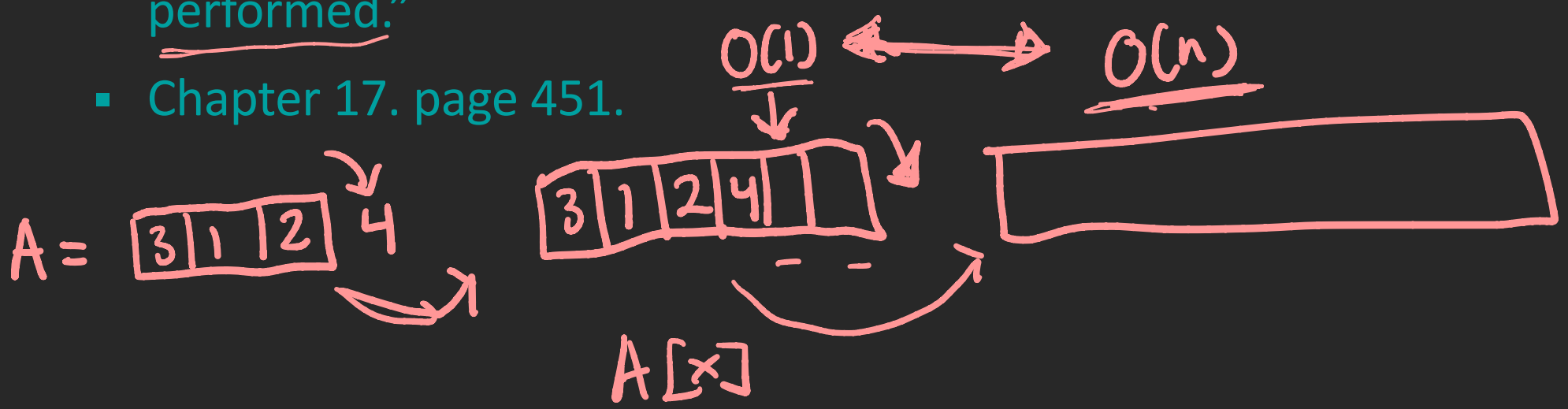
- Processing data efficiently depends on storing and accessing it efficiently
- Efficient data structures
  - Red-black trees – Ch. 13
  - AVL trees
  - Splay tress

Log(n)



Efficient Data Structures

- Just like algorithms, some data structures can be less or more efficient than others
- How will we analyze data structures:
  - Amortized analysis – "…the required time to perform a sequence of data-structure operations is averaged over all the operations performed."
  - Chapter 17. page 451.



Efficient Data Structures

- We use an *interface* to represent out *abstract data type* (ADT), like an *application programming interface* (API).

ADT

std::stack

```
template <class T, class Container = deque<T> > class stack;
```

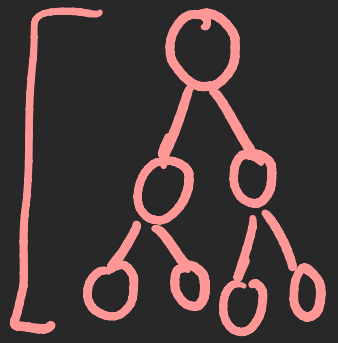
Member functions

(constructor)	Construct stack (public member function)
empty	Test whether container is empty (public member function)
size	Return size (public member function)
top	Access next element (public member function)
push	Insert element (public member function)
emplace C++11	Construct and insert element (public member function)
pop	Remove top element (public member function)
swap C++11	Swap contents (public member function)

Google Spreadsheets Python API v4

```python
import gspread

gc = gspread.service_account()

# Open a sheet from a spreadsheet in one go
wks = gc.open("Where is the money Lebowski?").sheet1

# Update a range of cells using the top left corner address
wks.update('A1', [[1, 2], [3, 4]])

# Or update a single cell
wks.update('B42', "it's down there somewhere, let me take another look.")

# Format the header
wks.format('A1:B1', {'textFormat': {'bold': True}})
```

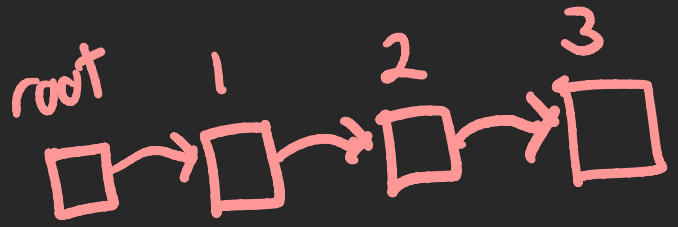
Efficient Data Structures



- Example: a self-balancing binary search tree (SBBST) can be used to implement a *dictionary* ADT

SBBST H	behavior	$T(n)$
`Add(x)`	add x to H	$O(\log n)$
`Find(x)`	determine whether x is in H	$O(\log n)$
`Remove(x)`	remove x from H	$O(\log n)$

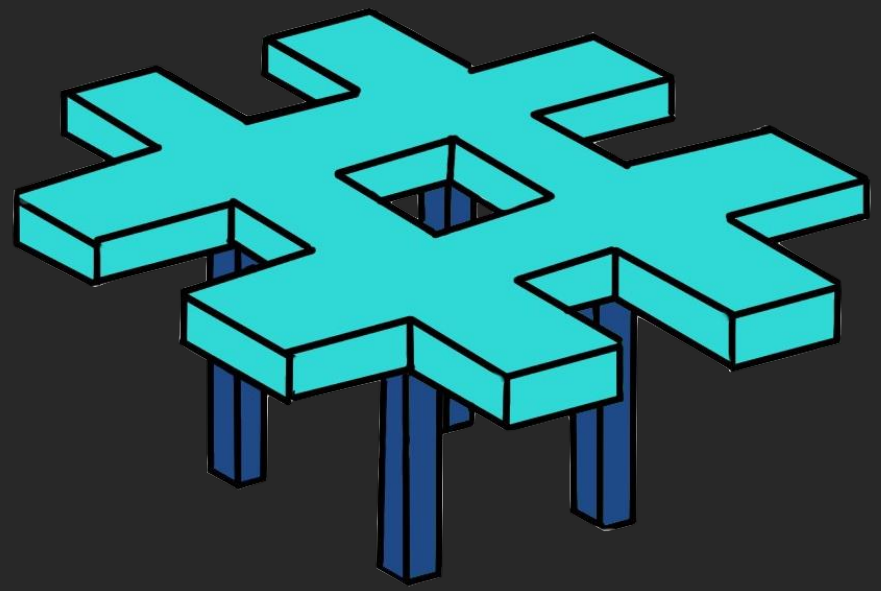
- Example: linked list implementation of a *dictionary* ADT



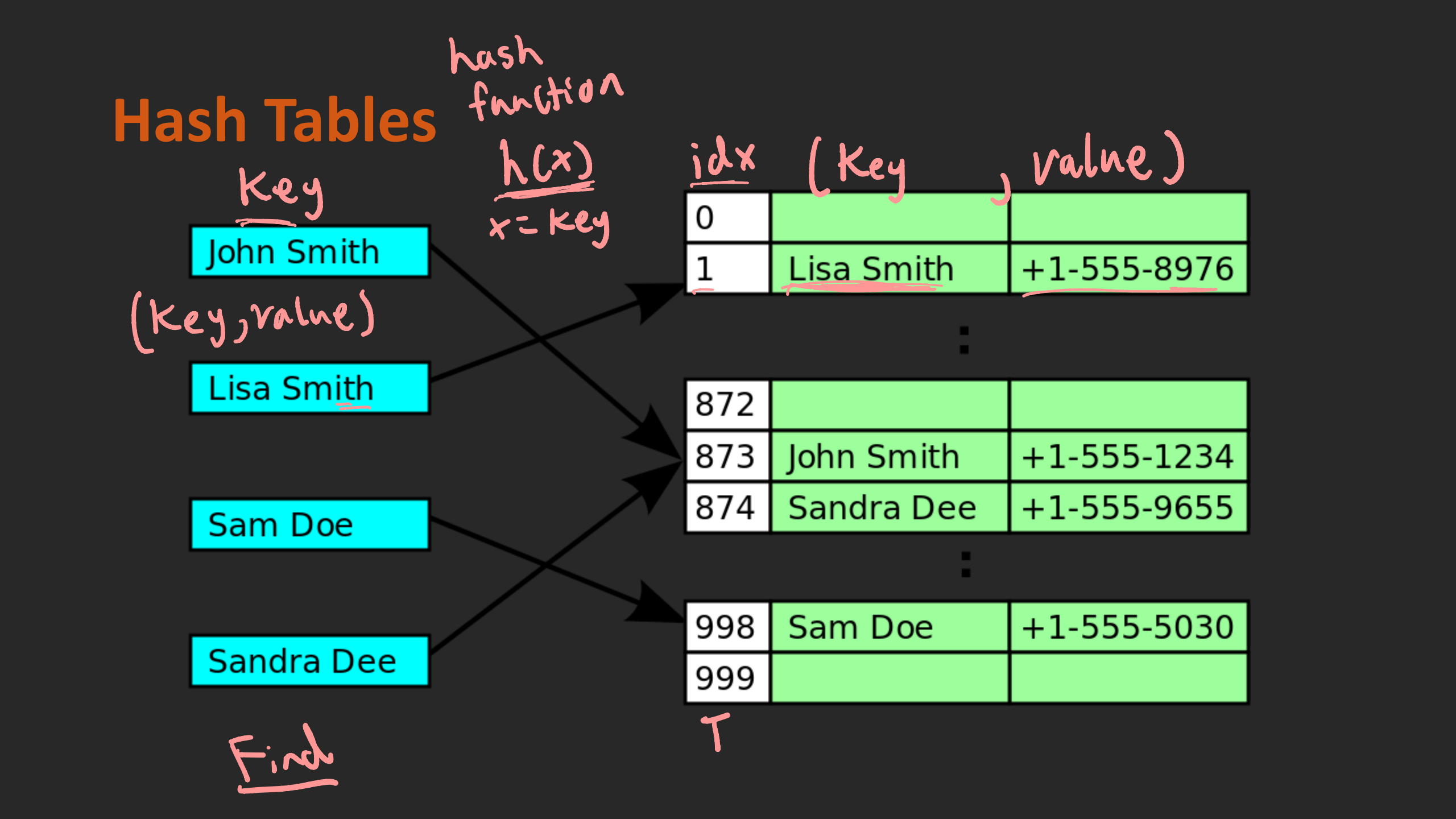
linked list H	behavior	$T(n)$
`Add(x)`	add x to H	$O(1)$
`Find(x)`	determine whether x is in H	$O(n)$
`Remove(x)`	remove x from H	$O(n)$

Lecture Outline

- ~~Poll review/Administrative~~
- ~~Efficient Data Structures~~ ✕
- Hash Table
  - Introduction
  - Perfect Hash
  - Chained Hash
- Birthday Paradox
- 4 Example Problems for HW2A



Hash Tables


hash function
h(x)
x = key
Key
John Smith
Lisa Smith
Sam Doe
Sandra Dee
(Key, value)
Find
idx
(Key, value)
0
1
Lisa Smith
+1-555-8976
872
873
John Smith
+1-555-1234
874
Sandra Dee
+1-555-9655
998
Sam Doe
+1-555-5030
999
T

Hash Tables

- Can also be used to implement a *dictionary* interface

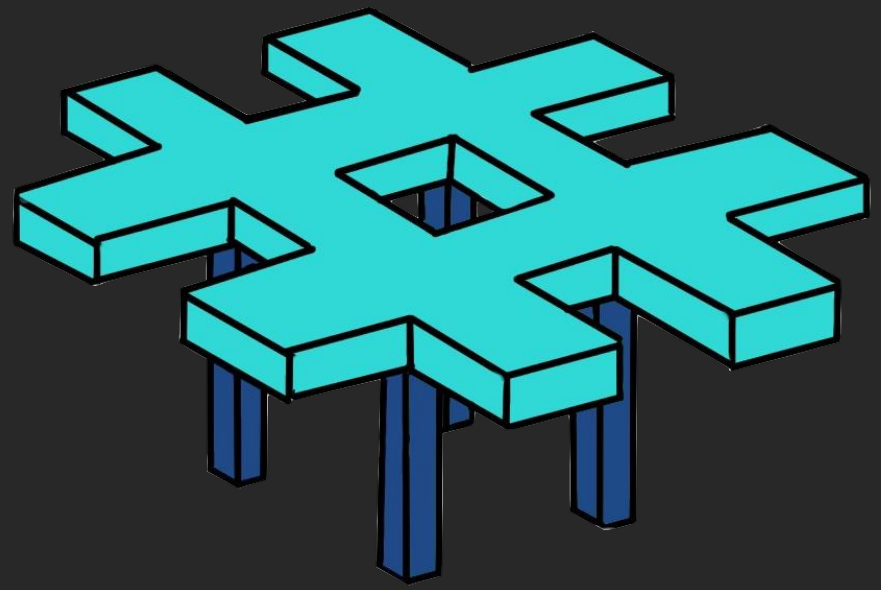
hash table H	behavior	expected	worst case
`Add(x)`	add x to H	$O(1+\alpha)$	$\Theta(n)$
`Find(x)`	determine whether x is in H	$O(1+\alpha)$	$\Theta(n)$
`Remove(x)`	remove x from H	$O(1+\alpha)$	$\Theta(n)$

- Use probabilities to decouple input from functionality
  - Similar to Randomized Quicksort

- This means we need to look at the average and worst-case behaviors

Lecture Outline

- ~~Poll review/Administrative~~
- ~~Efficient Data Structures~~
- Hash Table
  - ~~Introduction~~
    - Perfect Hash
    - Chained Hash
- Birthday Paradox
- 4 Example Problems for HW2A



Perfect Hash

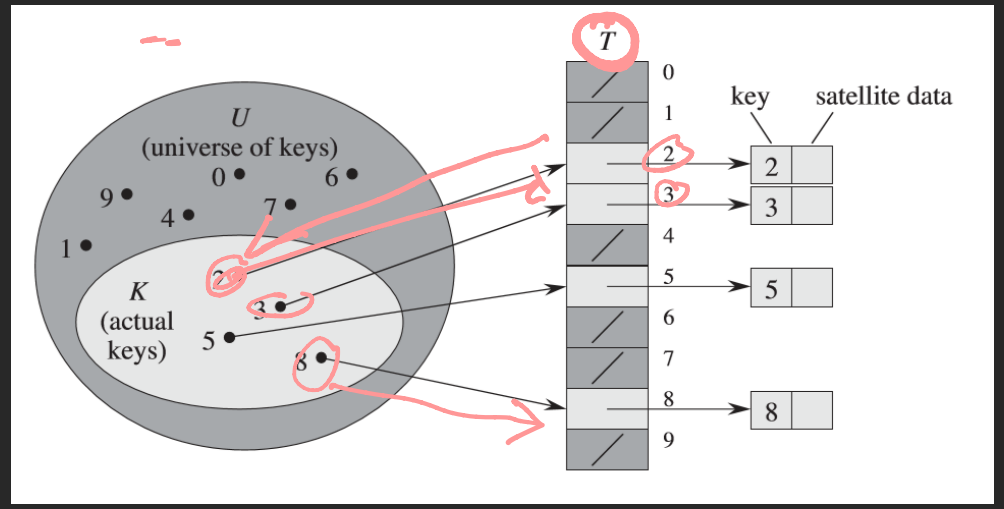
$|U| = T.len$

time $= O(1)$

- Assume we have a perfect hash function
  - No two elements are assigned to the same "key" or location

$$\forall_{x_i, x_j \in U} \; h(x_i) \neq h(x_j)$$

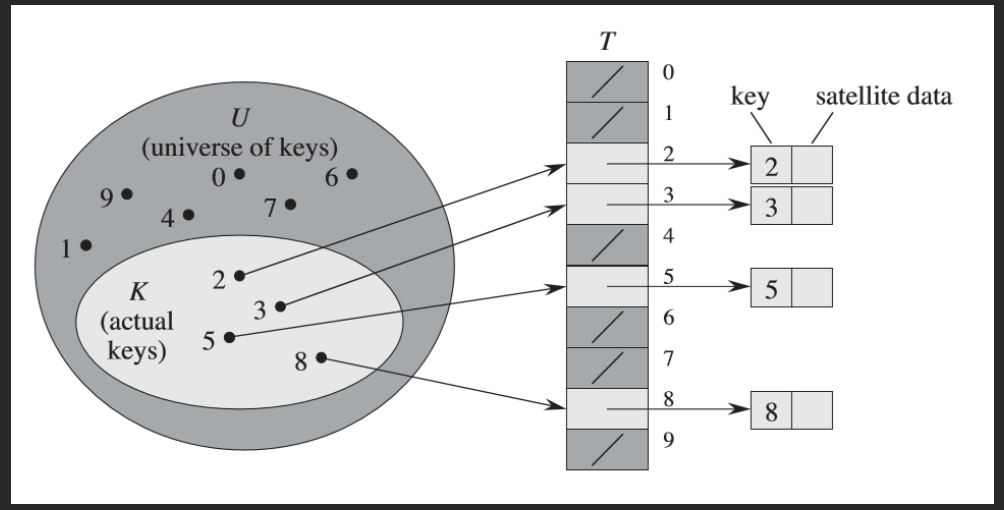
$$h(x) \text{ is } \textit{bijective}$$



Perfect Hash

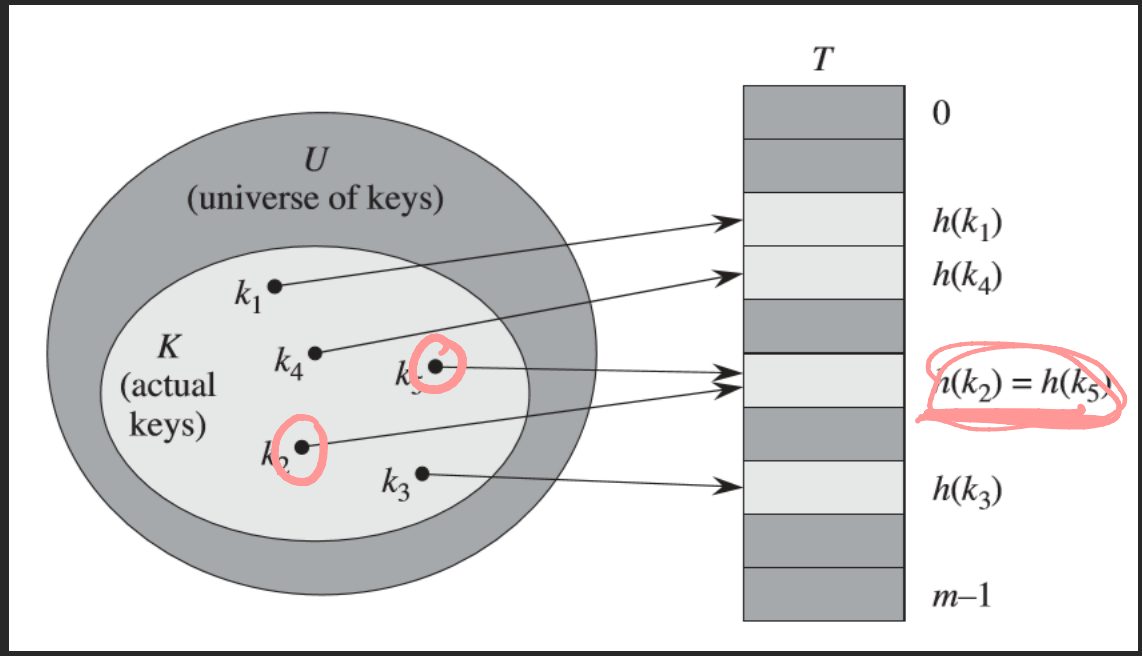
- We can use an array *A*[0…n-1] to store our elements

```
PerfectHash-Add(A,x,v)    { A[h(x)] = v    }
PerfectHash-Find(A,x)     { return A[h(x)] }
PerfectHash-Remove(A,x)   { A[h(x)] = NULL }
```



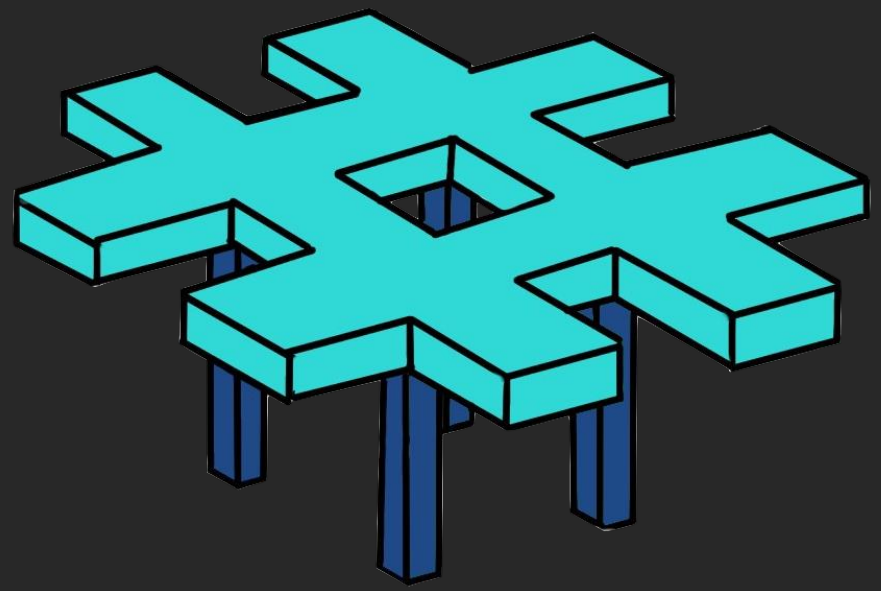
Practical Hash

- Make the has table smaller than the universe of possible elements
- Problem: in order to fit m items into $k < m$ location we need to assignee their values to the same location!



Lecture Outline

- ~~Poll review/Administrative~~
- ~~Efficient Data Structures~~
- Hash Table
  - ~~Introduction~~
  - ~~Perfect Hash~~
  - Chained Hash
- Birthday Paradox
- 4 Example Problems for HW2A



Chained Hash

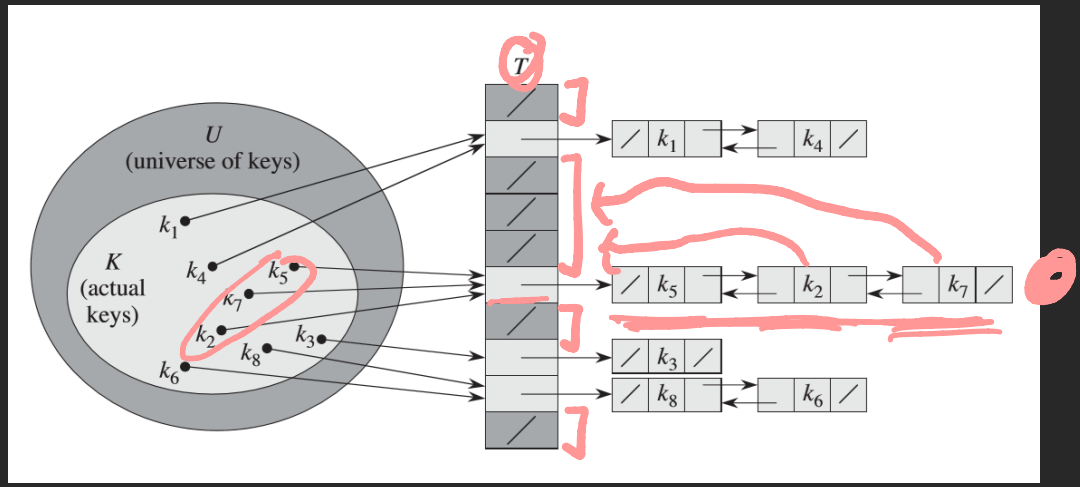
- Array *A* has *k* elements each of which is a linked list
- What happens if we have $h(x_i) = h(x_j) = k$

```
ChainedHash-Add(A,x,v)    { A[h(x)].prepend([x,v]) }
ChainedHash-Find(A,x)     { A[h(x)].search(x)      }
ChainedHash-Remove(A,x)   { A[h(x)].remove(x)      }
```

hash function
h(x)

Chained Hash

- How do we best choose *h(x)* in order to the length of linked lists in our hash table



- *Best case*: If we store n items in a hash table with ℓ slots, the best we can do is have each bucket contain the same number of items $\alpha = n/\ell$; we call α the *load factor*.
- *Worst case*: $h(x)$ assigns each of n items to the same bucket, and search takes $\Theta(n)$ time.

∀: $h(x_i) = 10$

Chained Hash – *Uniform Hashing*

- *Best case*: If we store n items in a hash table with ℓ slots, the best we can do is have each bucket contain the same number of items $\alpha = n/\ell$; we call α the *load factor*.
- *Worst case*: $h(x)$ assigns each of n items to the same bucket, and search takes $\Theta(n)$ time.

forall x

hash fxn

slots
buckets

$$\forall_x \ \Pr(h(x) = k) = 1/\ell$$

Chained Hash – *Uniform Hashing*

$$\forall_x \ \Pr(h(x) = k) = 1/\ell$$

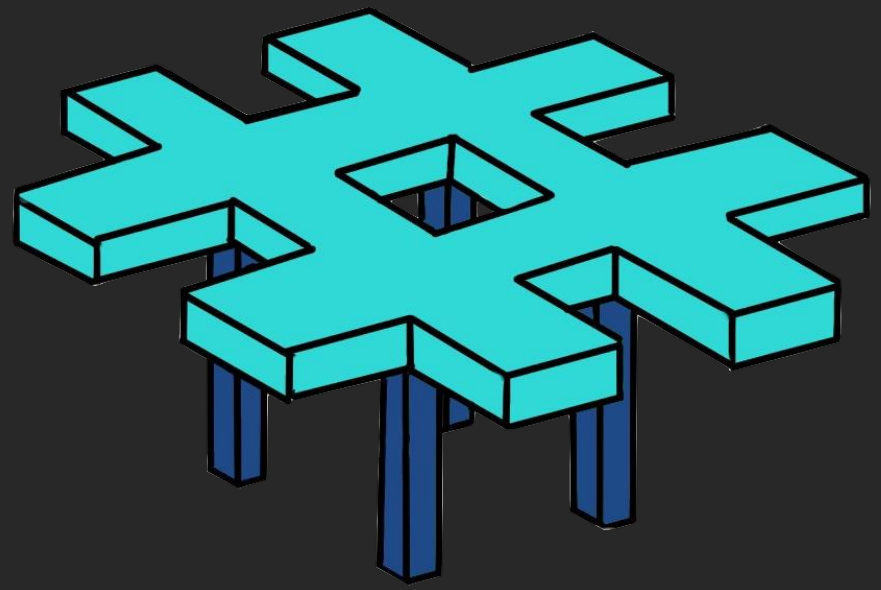
$$E(A[k].\text{length}) = \sum_{i=0}^{\ell-1} \frac{A[i].\text{length}}{\ell} = \frac{n}{\ell} = \alpha \ .$$

#of items

# of buckets

Lecture Outline

- ~~Poll review/Administrative~~
- ~~Efficient Data Structures~~
- ~~Hash Table~~
  - ~~Introduction~~
  - ~~Perfect Hash~~
  - ~~Chained Hash~~
- Birthday Paradox
- 4 Example Problems for HW2A



Sparse Case and Birthday Paradox!

$\alpha = 0$

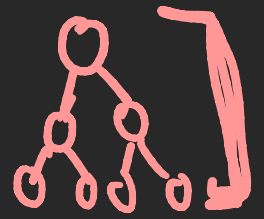
α is small

hash table H	behavior	expected	worst case
`Add(x)`	add x to H	$O(1+\alpha)$	$\Theta(n)$
`Find(x)`	determine whether x is in H	$O(1+\alpha)$	$\Theta(n)$
`Remove(x)`	remove x from H	$O(1+\alpha)$	$\Theta(n)$

- Relative to other implementation of the dictionary ADT, hash table becomes inefficient when:

$$\alpha = \Omega(\log n)$$

log(n)



Birthday Paradox!

- How large does n need to be before we start seeing buckets with multiple items?
- n people → items
- ℓ days a year → buckets
- What is the expected number of pairs of individuals that have the same birthday? In terms of n and ℓ

Birthday Paradox!

$$X_{ij} = \begin{cases} 1 & \text{if persons } i \text{ and } j \text{ have the same birthday} \\ 0 & \text{otherwise .} \end{cases}$$

$$E(X_{ij}) = \Pr(\text{persons } i \text{ and } j \text{ have the same birthday}) = 1/\ell$$

Birthday Paradox!

$\ell = 365$

$n = 28$

1.04

$$\binom{n}{2} = \frac{n!}{2 \cdot (n-2)!}$$

$$\frac{n \cdot (n-1) \cdot (n-2)!}{2 \cdot (n-2)!}$$

$$\frac{n(n-1)}{2} \cdot \frac{1}{\ell}$$

$$\begin{aligned} E(X) &= E\left(\sum_{ij} X_{ij}\right) \\ &= \sum_{ij} E(X_{ij}) \\ &= \sum_{ij} \frac{1}{\ell} \\ &= \binom{n}{2} \cdot \frac{1}{\ell} \\ &= \frac{n(n-1)}{2\ell} \,. \end{aligned}$$

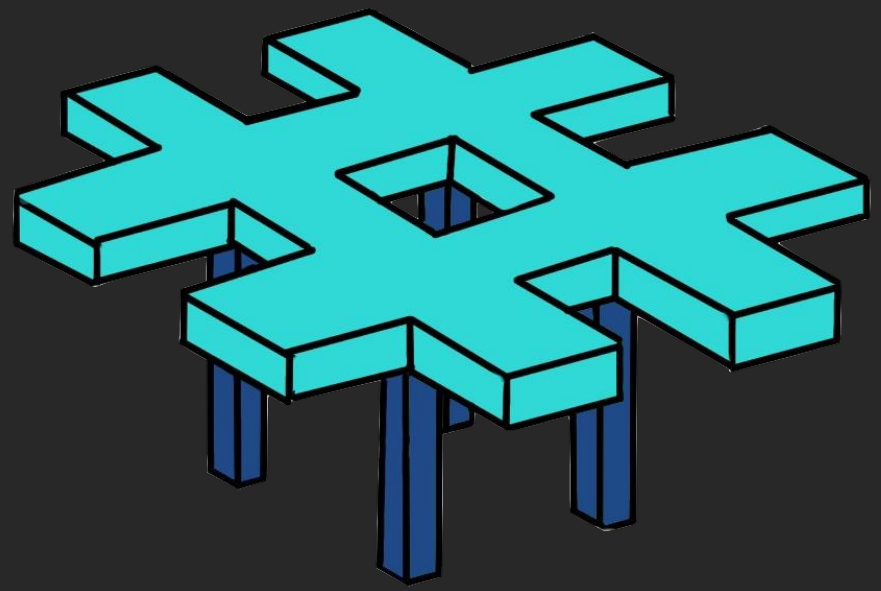
Closing Thoughts

- We will continue going over hash tables next class
- Homework 1B is due tomorrow night
- This will help you with homework 2B
  - Should be a depth first traversal of tree!!!

procedure	arguments	input	output	global array
first partition	$x = 3 \quad p = 0 \quad r = 5$	$A = [2, 6, 4, 1, 5, \mathbf{3}]$	$[2, 1 \mid \mathbf{3} \mid 6, 5, 4]$	$[2, 1, 3, 6, 5, 4]$
L QS, partition	$x = 1 \quad p = 0 \quad r = 1$	$A = [2, \mathbf{1}]$	$[\mathbf{1} \mid 2]$	$[1, 2, 3, 6, 5, 4]$
R QS, partition	$x = 4 \quad p = 3 \quad r = 5$	$A = [6, 5, \mathbf{4}]$	$[\mathbf{4} \mid 5, 6]$	$[1, 2, 3, 4, 5, 6]$
L QS, partition	do nothing			$[1, 2, 3, 4, 5, 6]$
R QS, partition	$x = 6 \quad p = 4 \quad r = 5$	$A = [5, \mathbf{6}]$	$[5 \mid \mathbf{6}]$	$[1, 2, 3, 4, 5, 6]$

Lecture Outline

- ~~Poll review/Administrative~~
- ~~Efficient Data Structures~~
- ~~Hash Table~~
  - ~~Introduction~~
  - ~~Perfect Hash~~
  - ~~Chained Hash~~
- ~~Birthday Paradox~~
- 4 Example Problems for HW2A



Example 1 unrolling

- $T(n) = 4T(n-1) + 4$, $T(1) = \Theta(1)$

$4(4T(n-2)+4)+4$

$4^2T(n-2)+4^2+4$

$4^3T(n-3)+4^3+4^2+4$

k

$\rightarrow 4^kT(n-k)+4^k+4^{k-1}+\dots+4$

$4^{n-1}T(1)+4^{n-1}+4^{n-2}+\dots+4$

$T(n-k) = T(1)$

$k = n-1$

$T(n-(n-1))$

$T(n-n+1) = T(1)$

$\rightarrow 4^n\left(4^{-1}T(1)+4^{-1}+4^{-2}+\dots+4^{-n+1}\right)$

C $\qquad \sum_{i=1}^{n-1}\frac{1}{4^i} < \frac{4}{3}$

$C\,4^n = \Theta(4^n)$

Example 2

- $T(n) = 3T(n/5) + (n^2)$

$a \quad b \quad f(n)$

$a f(\frac{n}{b}) \le c f(n)$

$a \cdot (\frac{n}{b})^2 \le cn^2$

$3 \cdot (\frac{n}{5})^2 \le cn^2$

$\frac{3}{25} n^2 \le cn^2$

$\frac{3}{25} \le c < 1$

$c = \frac{4}{25}$

Master theorem. Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function and let $T(n)$ be defined on the non-negative integers by the recurrence,

$$T(n) = a\,T(n/b) + f(n) \ ,$$

where we interpret n/b to mean either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then $T(n)$ can be bounded asymptotically as follows:

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \log n)$
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $a\,f(n/b) \leq c\,f(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$. □

1) $n^{\log_b a} = n^{\log_5 3}$

2) $f(n) = n^2$

3) what is the relation ship between $f(n)$ and $n^{\log_b a}$

$\log_5 5 = 1$

$\log_5 25 = 2$

$n^2 = n^{\log_5 (3+\epsilon)}$

$\epsilon = 22$

$T(n) = \Theta(f(n)) = \Theta(n^2)$

Example 3

- T(n) = 4T(n/3) + (1)

a b

Master theorem. Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function and let $T(n)$ be defined on the non-negative integers by the recurrence,

$$T(n) = a\,T(n/b) + f(n) \ ,$$

where we interpret n/b to mean either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then $T(n)$ can be bounded asymptotically as follows:

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \log n)$
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $a\,f(n/b) \leq c\,f(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$. □

$T(n) = \Theta(n^{\log_3 4})$

1) $n^{\log_b a} = n^{\log_3 4}$

2) $f(n) = 1$

3) $f(n) \;?\; n^{\log_b a}$

$\log_3 3 = 1$

$\log_3 9 = 2$

$\log_3 1 = 0$

$1 = n^{\log_3(4-\varepsilon)}$

$4 - \varepsilon = 1$

$\varepsilon = 3$

Example 4

- T(n) = 5T(n/5) + (n)

Master theorem. Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function and let $T(n)$ be defined on the non-negative integers by the recurrence,

$$T(n) = a\,T(n/b) + f(n) \ ,$$

where we interpret n/b to mean either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then $T(n)$ can be bounded asymptotically as follows:

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \log n)$
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $a\,f(n/b) \leq c\,f(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$. □

1) $n^{\log_5 a} = n^{\log_5 5} = n^1$

2) $f(n) = n$

3) Compare $f(n)$ and $n^{\log_5 a}$

$f(n) = n = n^{\log_5 5}$, Case 2

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