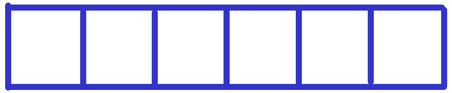
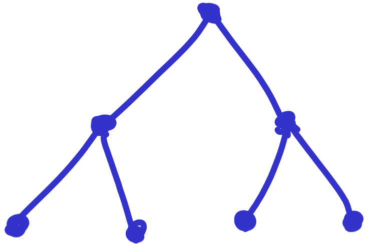
The 3 main operations on data structures:

SEARCH
INSERT
DELETE

How fast can we do these?





SEARCH	$O(n)$ [$O(\log n)$ sorted]	$O(n)$	
INSERT	$O(n)$ (if maintaining sorted)	$O(1)$	$O(\log n)$
DELETE	$O(n)$ (if we don't want gaps)	$O(n)$ O(1) + search	

basic

HASHING

The 3 main operations on data structures:

SEARCH
INSERT
DELETE

$O(1)$ expected with assumptions

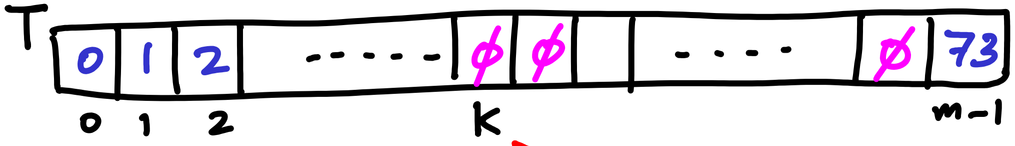
- Not "expected worst-case", just "average".
- For some methods, some ops can be $O(1)$ worst-case.

HASHING

Direct access table : good when keys are distinct & come from a small distribution $\mathcal{U}$.

e.g. $\mathcal{U} = \{0, 1, 2, \ldots, m-1\}$

Say $m=74$. Use an array:

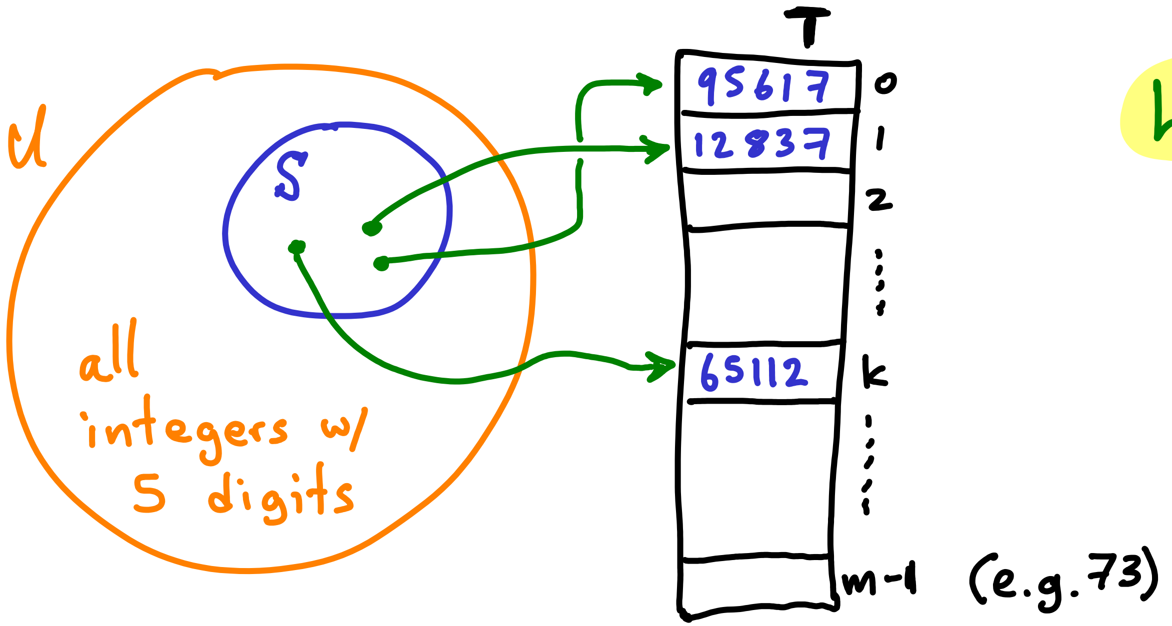


$\text{search}(T, 2) = 2$ // $\text{insert}(T, k) \rightarrow T[k] = k$ // $\text{delete}(T, k) \rightarrow T[k] = \phi$

All $\Theta(1)$

If U is larger than our available storage, m

but we are working with a subset S of U, where $|S| \le m$



h : hash function

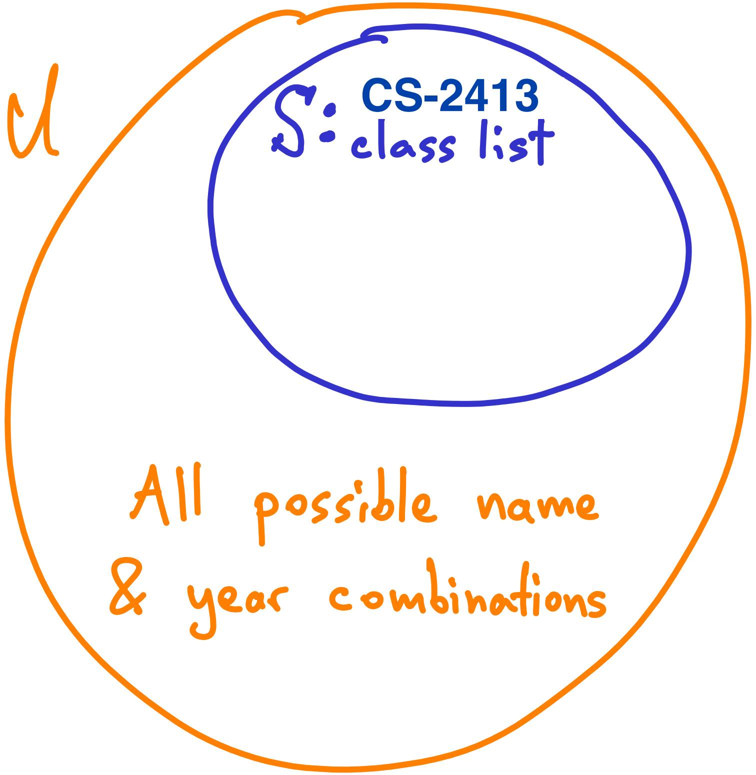
maps keys to T.

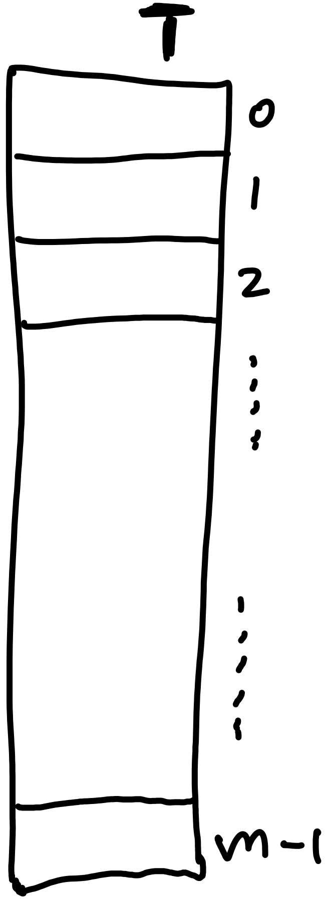
$h(95617) = 0$

$h(12837) = 1$

$h(65112) = k$

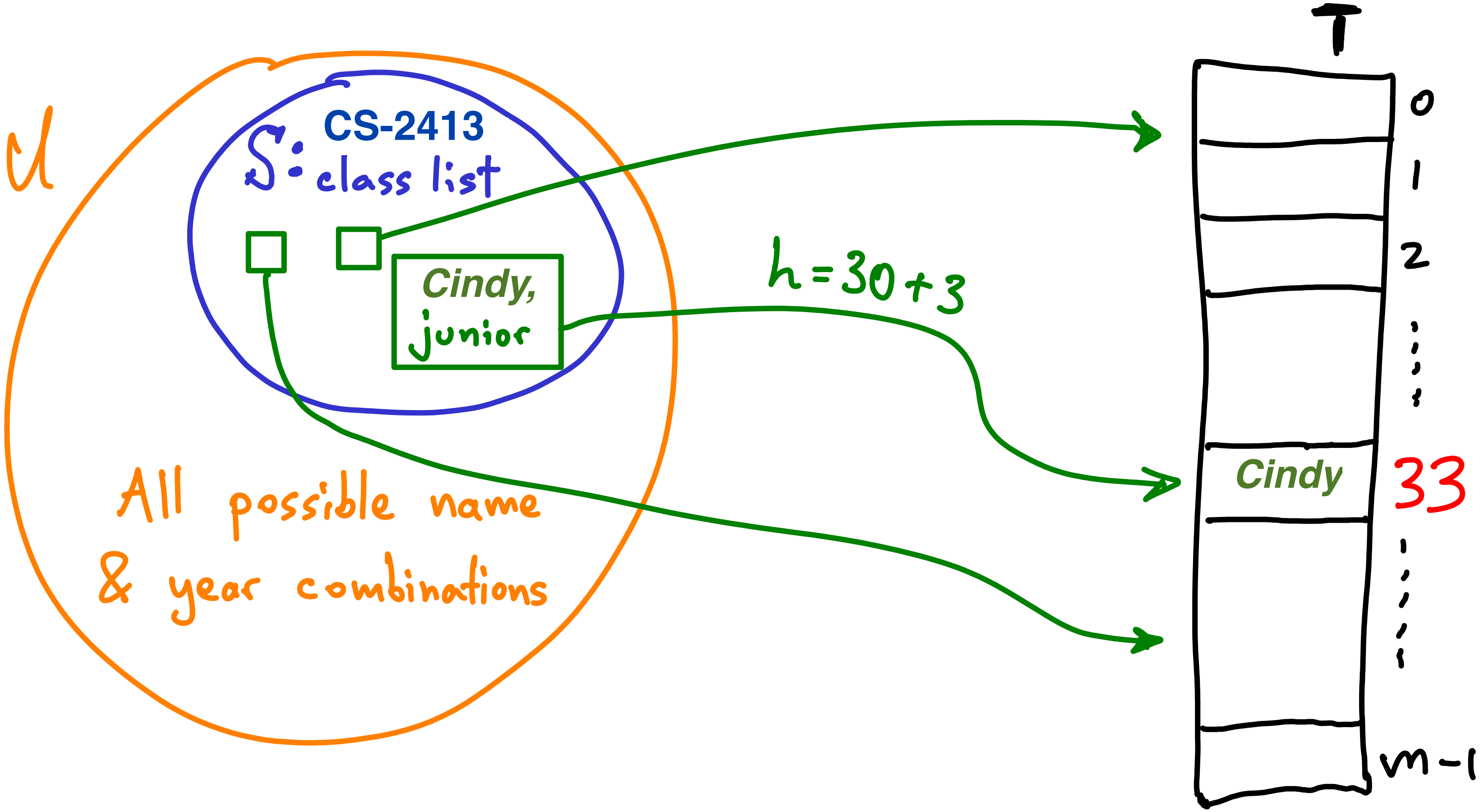
Example: look up this semester's **CS-2413** students using name & academic level (year)





h:

- take first letter of name, map to number $\longrightarrow L = \{1 \ldots 26\}$
- map year similarly: sophomore = 2, junior = 3, etc $\longrightarrow Y = \{0 \ldots 9\}$
- $h(\text{student}) = 10 \cdot L + Y$ → unique for any value in $\{L, Y\}$

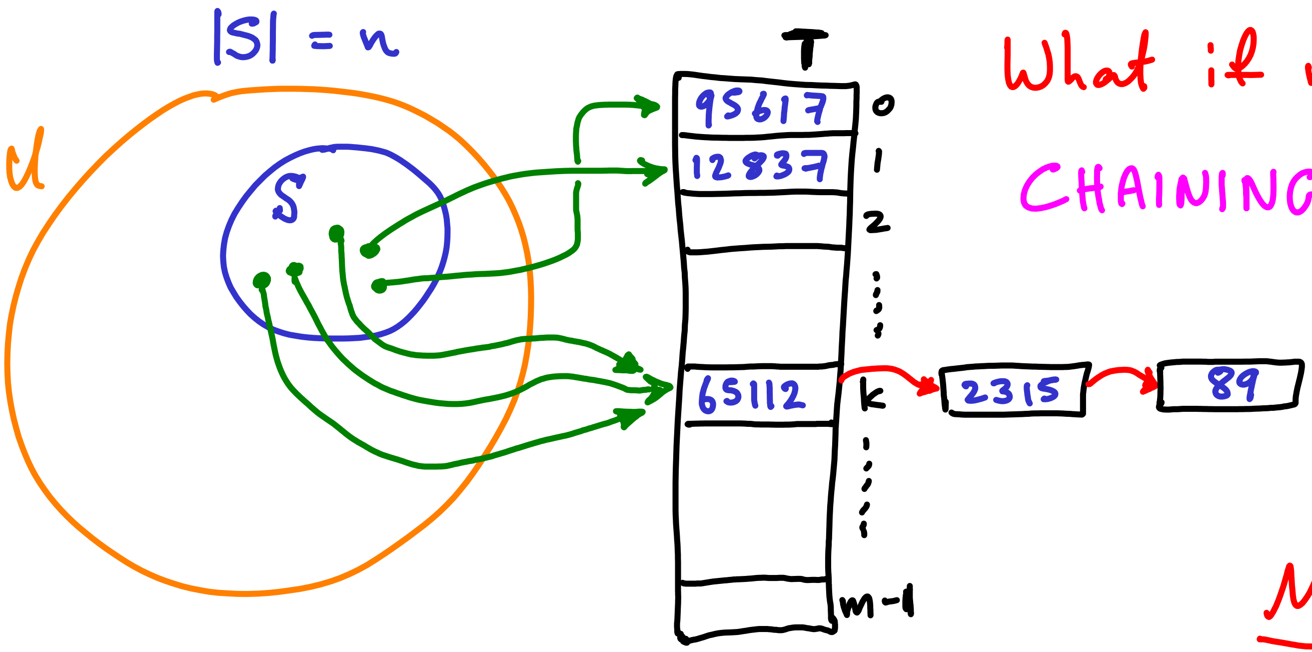


PROBLEMS?

(1) Some permanently empty slots

(2) other h=33?

- Could use more of the given info to design a more complicated h()
  - might minimize collisions
- But that involves costly processing
  and will need to be repeated if S changes (e.g. next semester)
- We want to keep a simple h() and deal with collisions



What if many keys map to same slot?

CHAINING: Make a linked list.

Insert = $\Theta(1)$

Search/Delete = $\underbrace{O(n)}_{\text{list size}}$

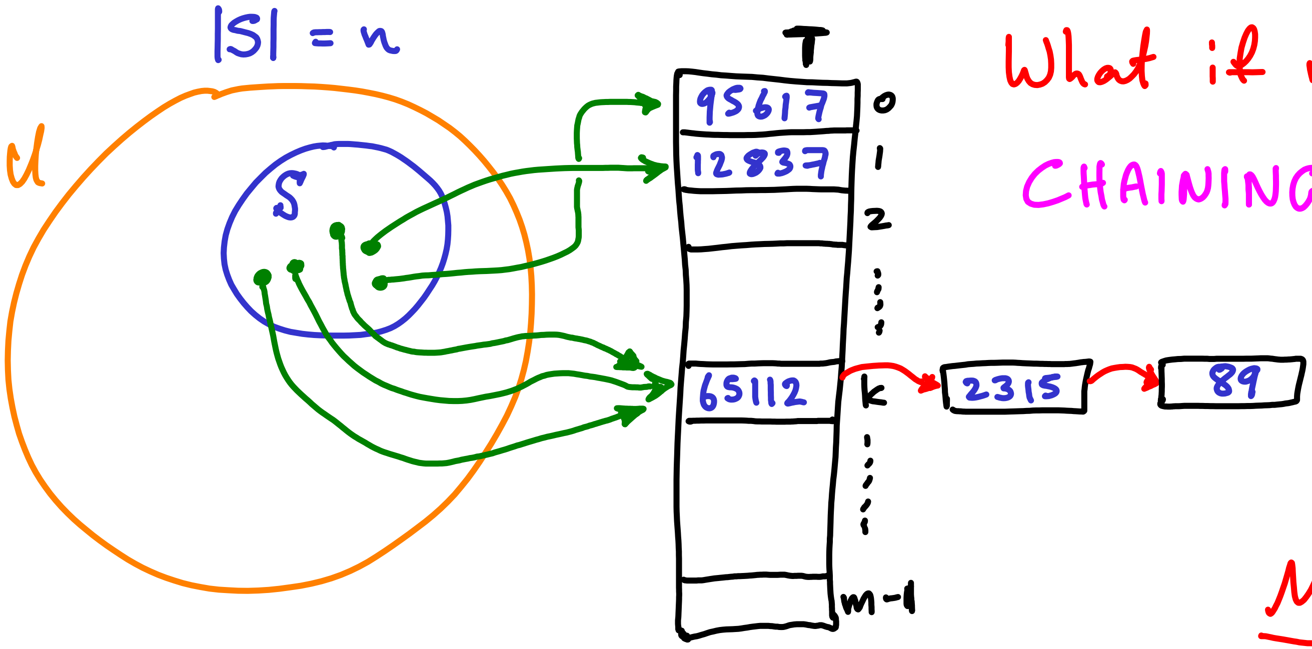
Must be consistent for each key

If CHAINING, we don't need $n<m$.

$n>m$: COLLISIONS are unavoidable

minimize collisions by spreading S into T evenly

↳ want random-looking h() yet consistent / deterministic



What if many keys map to same slot?

CHAINING: Make a linked list.

Insert $= \Theta(1)$

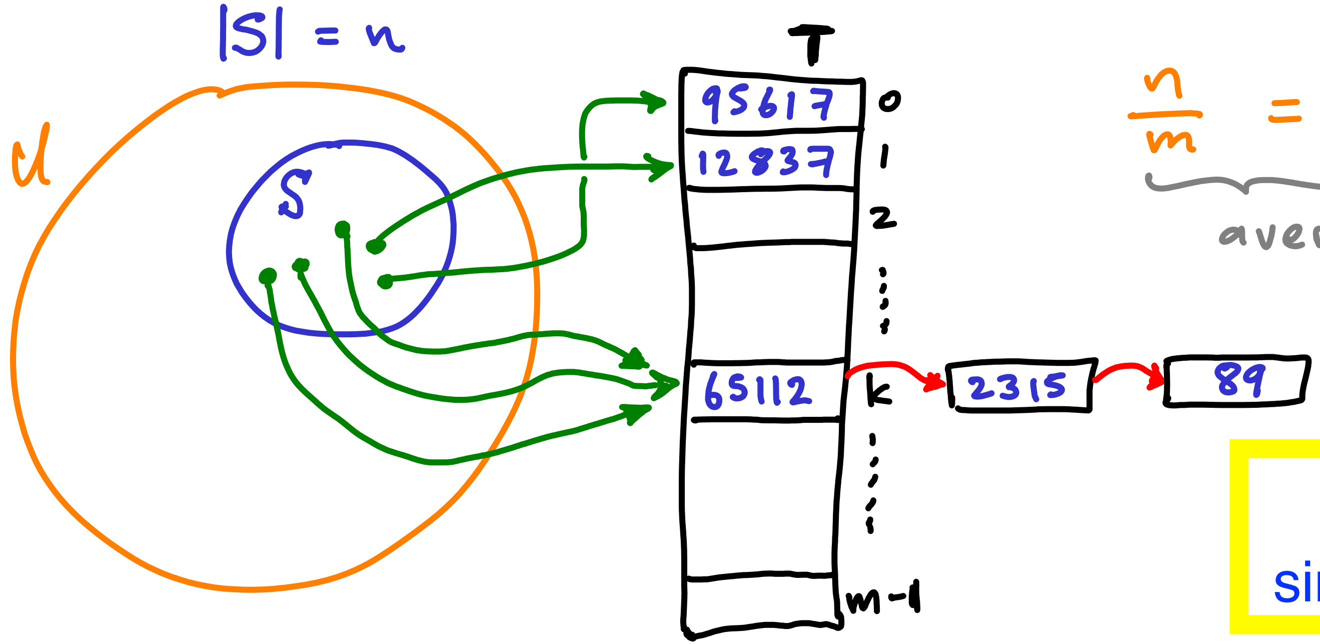
Search/Delete $= \underbrace{O(n)}_{\text{list size}}$

Must be consistent for each key

For a random h, every slot is equally likely : simple uniform hashing

Probability two given keys collide $= \frac{1}{m}$

Average # keys per slot $= \underbrace{\frac{n}{m} = \alpha}_{\text{average size of linked list.}}$ = "load factor"



$$\underbrace{\frac{n}{m} = \alpha}_{\text{average size of linked list.}} = \text{"load factor"}$$

This analysis assumes
simple uniform hashing condition

Expected time of search (and delete) $= \overbrace{\Theta(1+\alpha)}^{\text{great if } \alpha = \Theta(1)}$

1) h(key) → slot# → Assume h() takes $\Theta(1)$ to evaluate

2) scan list → Expect to scan $\geq$ half of a list

CHOOSING HASHING FUNCTIONS depending on keys and m.

Objective: get uniform distribution of keys to slots - always

Ex: $h(k) = k \bmod m$ "Division method"

If S = integers then its fine.

...but if $S = m \cdot i$ for $i = 1, 2, 3$ etc <u>**FAIL**</u>

We don't want <u>any</u> specific input pattern to affect uniformity.

"Fails" if m has a small divisor. e.g. for even m, if all keys are even, half of T: empty.

If $m = 2^r$ then $k \bmod m = k \bmod 2^r$ keeps only last r digits

$r=6$: $k = 1011000111\underline{011010}$

h depends on a small part of the input (key)

heuristic: choose m : prime & not close to power of 2

"MULTIPLICATION METHOD"

(Just an FYI. You don't need to know this)

Suppose $m = 2^r$, and we are using w-bit words (keys)

$h(k) = (A \cdot k) \bmod 2^w$ right-shifted by $w-r$

↳ some odd integer in $[2^{w-1} \ldots 2^w - 1]$ → w-bit # with leading 1.

heuristic: pick A not close to any power of 2

ex: $m = 2^3$: $r = 3$
$w = 7$

$$\left.\begin{matrix} A = 1011001 \\ k = 1101011 \end{matrix}\right\} A \cdot k = \underbrace{1001010\,011\,\overbrace{0011}^{w-r}}_{\text{remains after } \bmod 2^7}$$

$h(1101011) = 011$

If we had $A = 2^{w-1}$ → $A \cdot k = 1101011\underline{0000000}$
1000000

or, $A = 2^5$ → $A \cdot k = 0011010\underline{1100000}$

heuristic provides some "randomness" to the process.