

# HASHING

The 3 main operations on data structures:

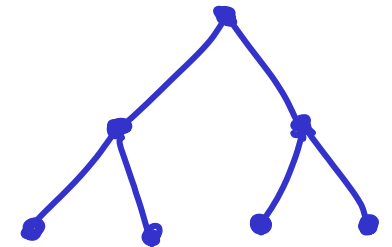
SEARCH  
INSERT  
DELETE

How fast can we do these?

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 }  $O(1)$   
DELETE } expected with assumptions

# 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  $U$ .

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

# HASHING

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

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

Say  $m=74$ .

Use an array:  $T$

|   |   |   |     |        |        |     |        |       |
|---|---|---|-----|--------|--------|-----|--------|-------|
| 0 | 1 | 2 | ... | $\phi$ | $\phi$ | ... | $\phi$ | 73    |
| 0 | 1 | 2 |     | $k$    |        |     |        | $m-1$ |

# HASHING

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

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

Say  $m=74$ . Use an array:  $T$

|   |   |   |     |        |        |     |        |     |
|---|---|---|-----|--------|--------|-----|--------|-----|
| 0 | 1 | 2 | ... | $\phi$ | $\phi$ | ... | $\phi$ | 73  |
| 0 | 1 | 2 |     | k      |        |     |        | m-1 |

$\text{search}(T, 2) = 2$





# HASHING

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

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

Say  $m=74$ . Use an array: 

|   |   |   |       |   |   |  |  |  |       |   |     |
|---|---|---|-------|---|---|--|--|--|-------|---|-----|
| 0 | 1 | 2 | ----- | k | ϕ |  |  |  | ----- | ϕ | 73  |
| 0 | 1 | 2 |       | k |   |  |  |  |       |   | m-1 |

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

# HASHING

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

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

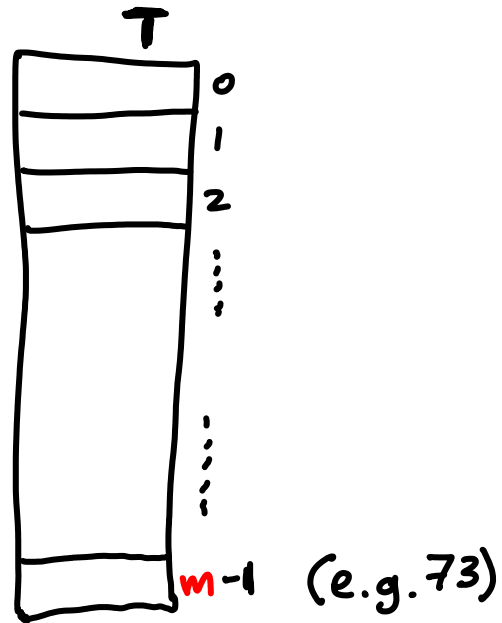
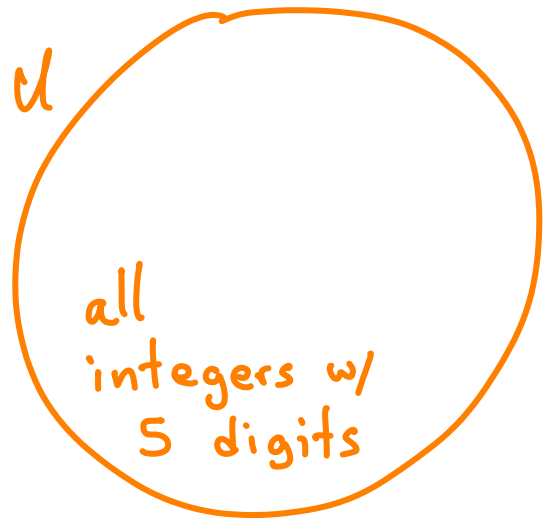
Say  $m=74$ . Use an array: 

|   |   |   |     |        |        |     |        |    |
|---|---|---|-----|--------|--------|-----|--------|----|
| 0 | 1 | 2 | ... | $\phi$ | $\phi$ | ... | $\phi$ | 73 |
| 0 | 1 | 2 |     | $k$    |        |     | $m-1$  |    |

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

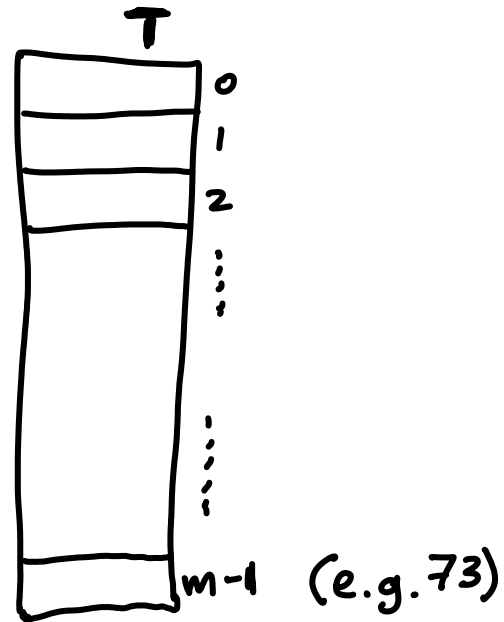
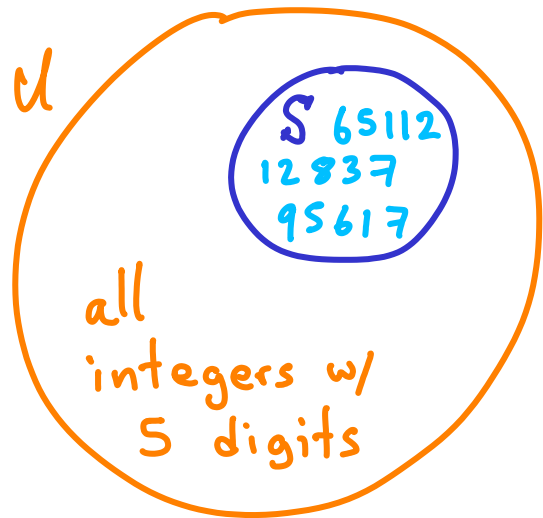
All  $\Theta(1)$

If  $U$  is larger than our available storage,  $m$



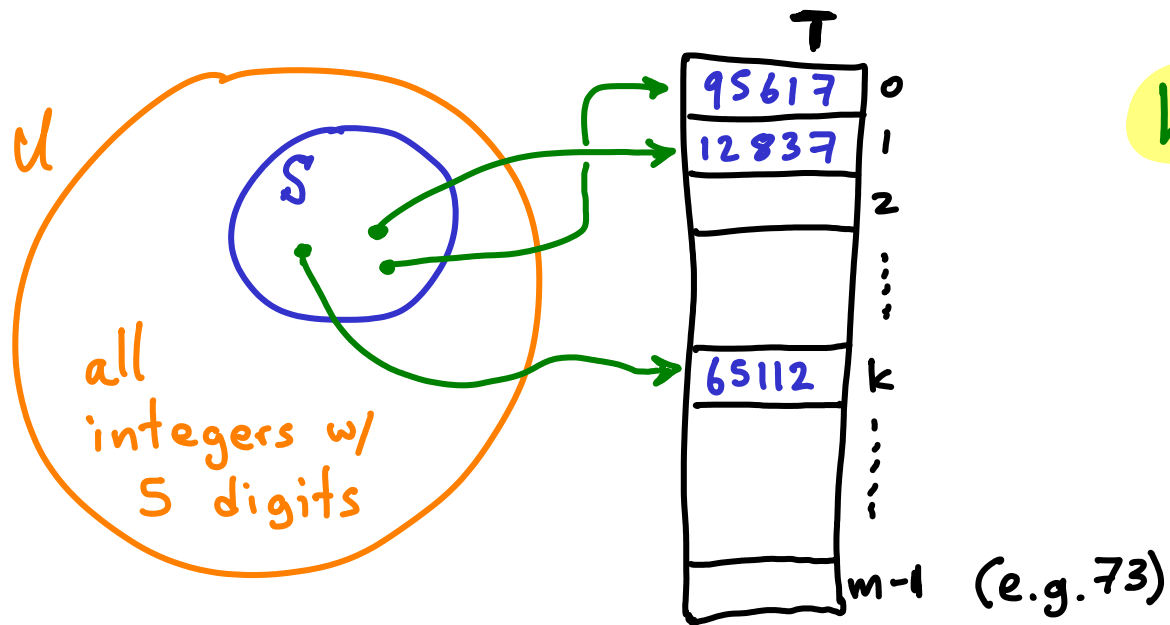
If  $U$  is larger than our available storage,  $m$

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



If  $U$  is larger than our available storage,  $m$

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



$h$ : hash function

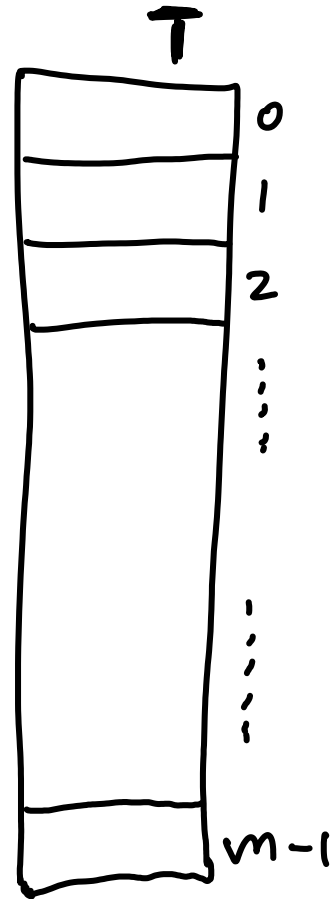
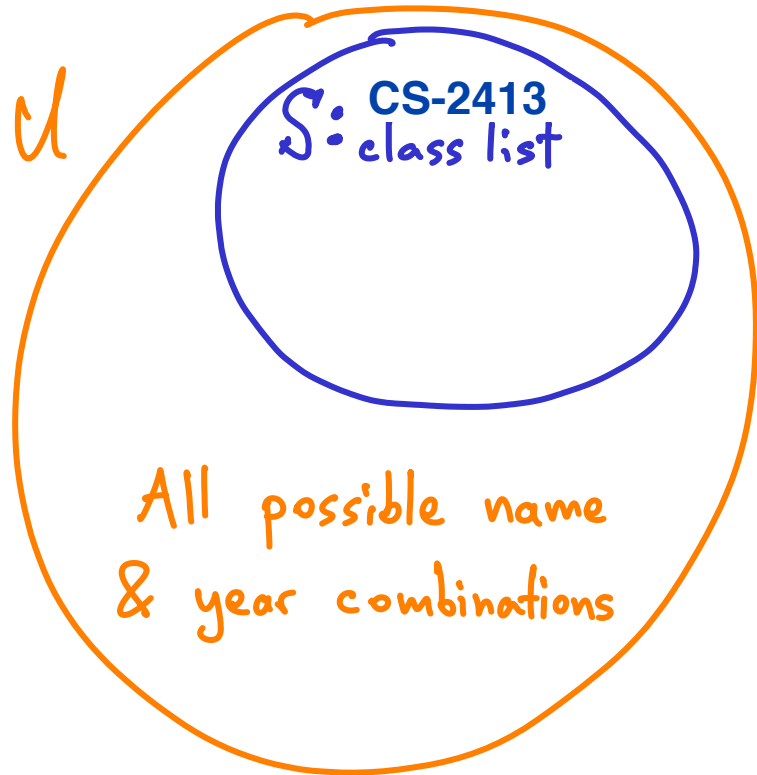
maps keys to  $T$ .

$$h(95617) = 0$$

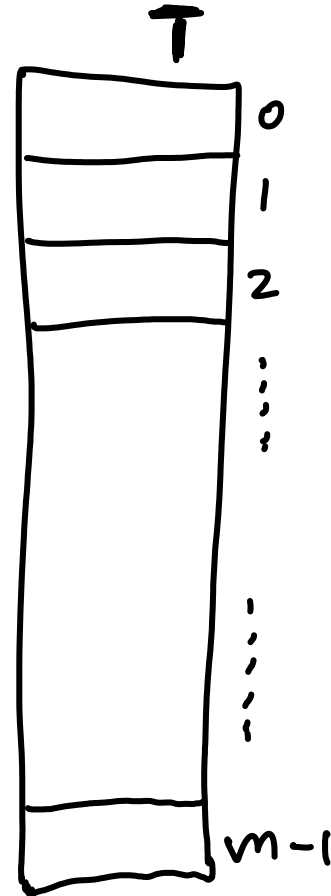
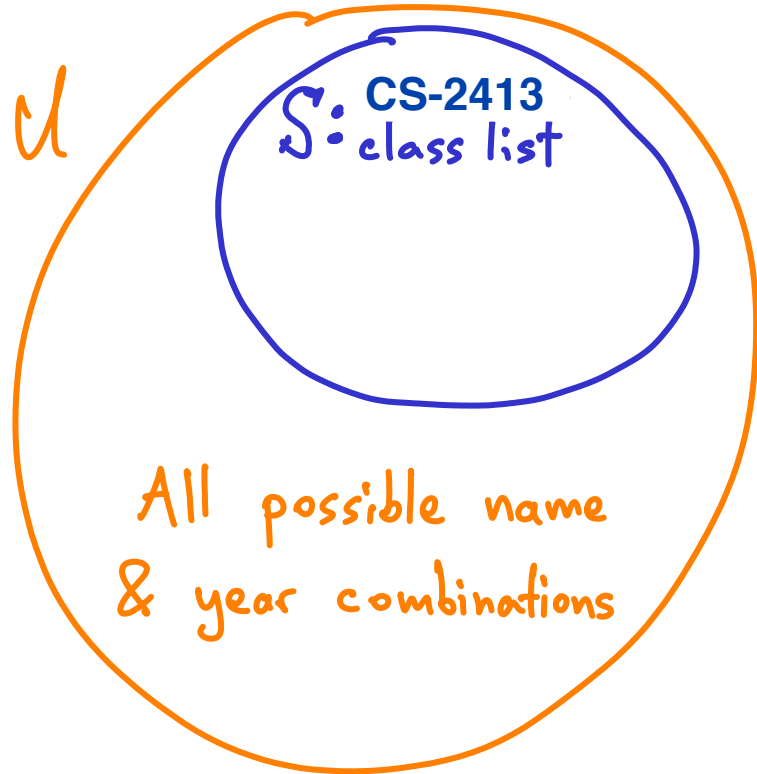
$$h(12837) = 1$$

$$h(65112) = k$$

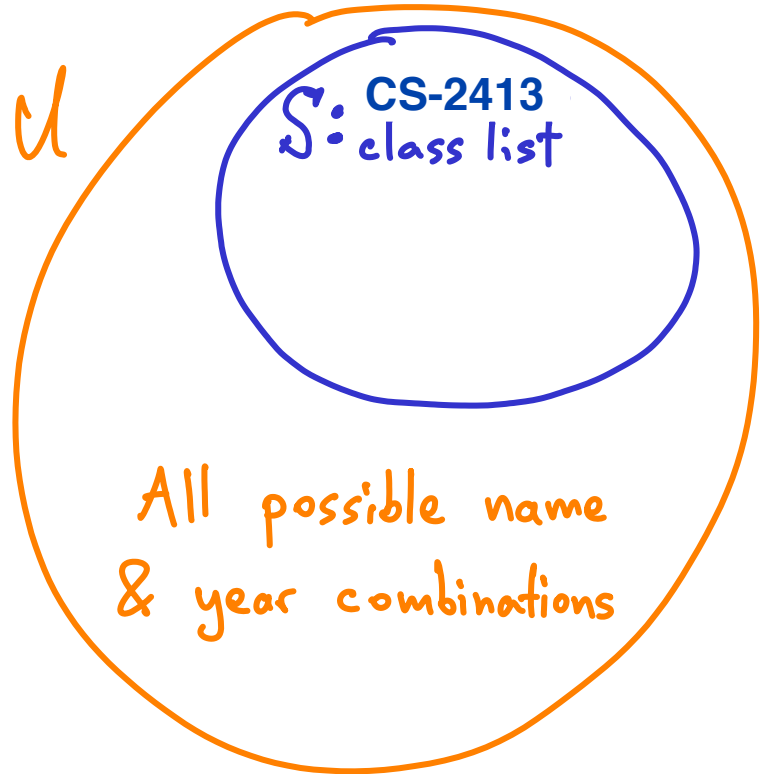
Example: look up this semester's CS-2413 students  
using name & academic level (year)



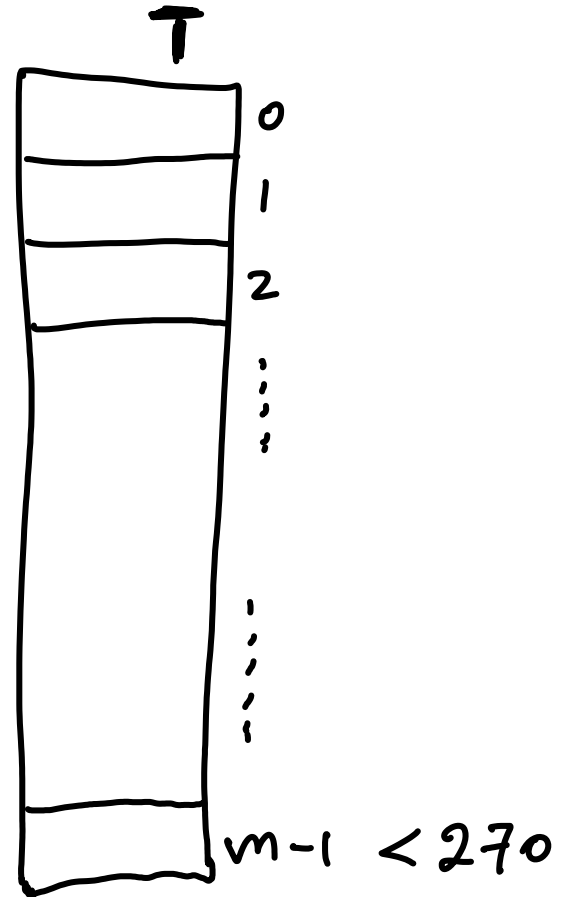
- take first letter of name, map to number  $\longrightarrow L = \{1 \dots 26\}$
- map year similarly: sophomore = 2, junior = 3, etc  $\longrightarrow Y = \{0 \dots 9\}$



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

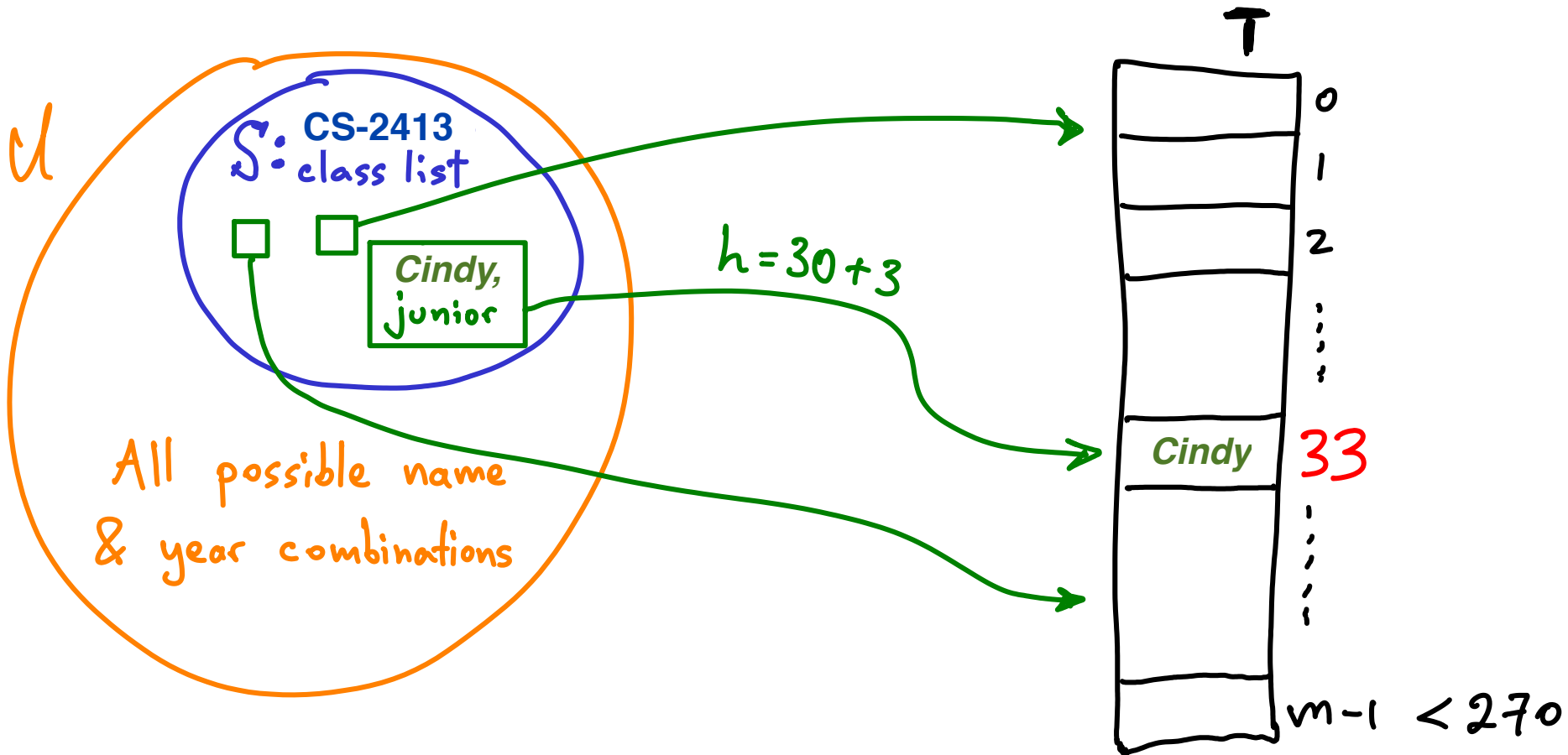


All possible name  
& year combinations

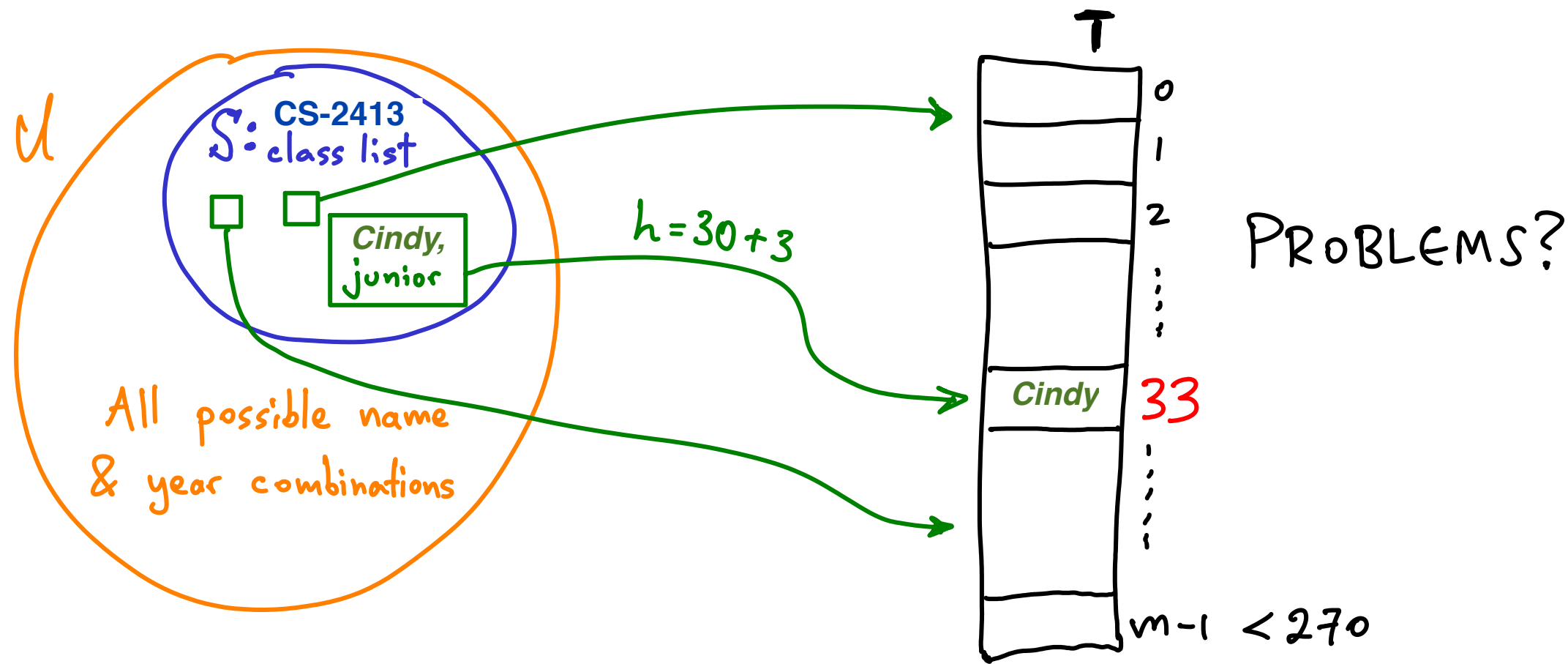




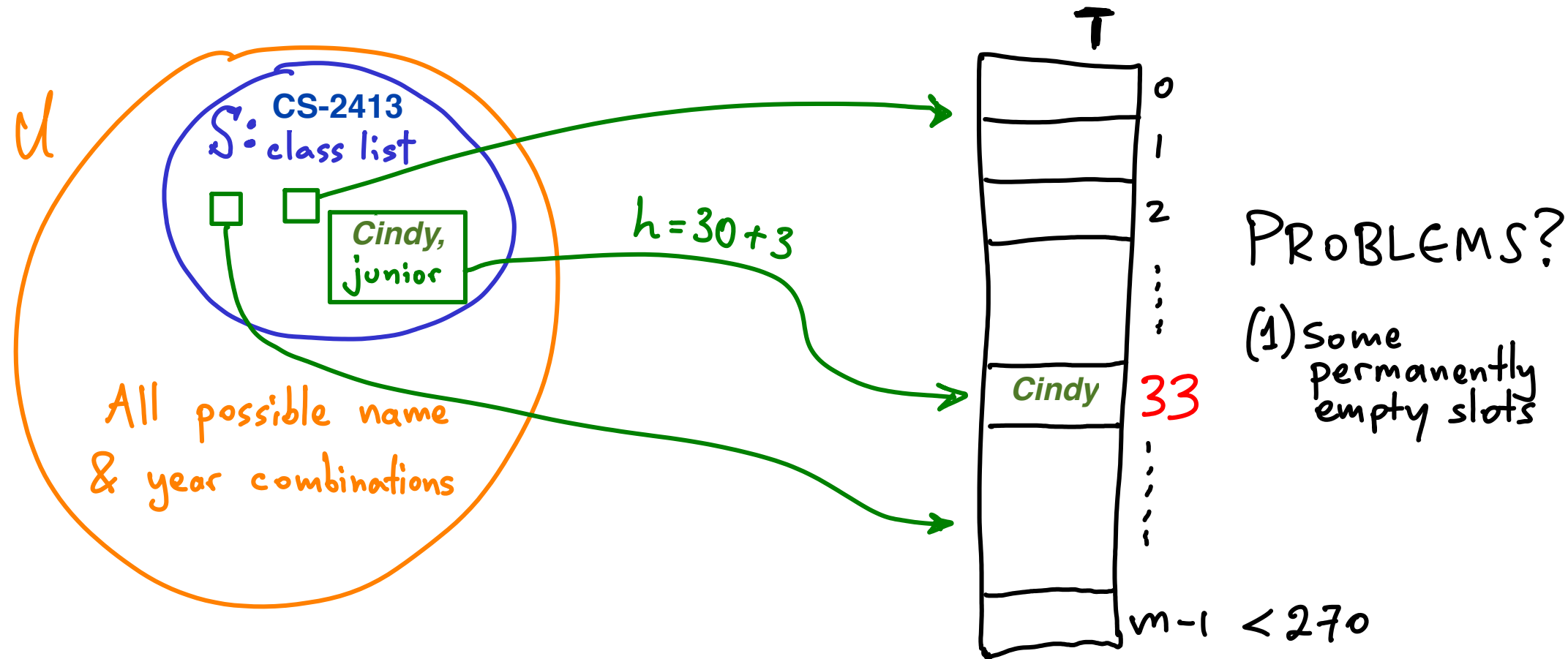
- $h$
- take first letter of name, map to number  $\rightarrow L = \{1 \dots 26\}$
  - map year similarly: sophomore = 2, junior = 3, etc  $\rightarrow Y = \{0 \dots 9\}$
  - $h(\text{student}) = 10 \cdot L + Y \rightarrow$  unique for any value in  $\{L, Y\}$



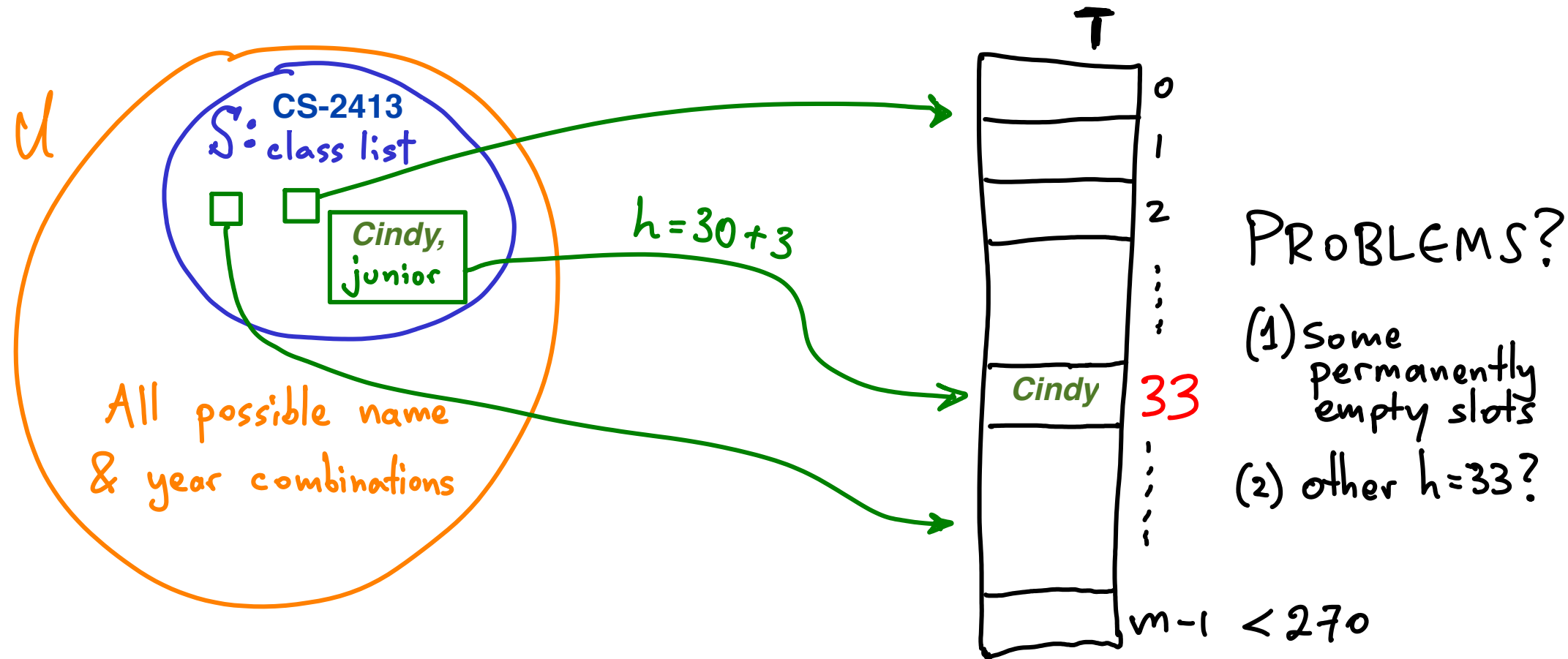
- $h$
- take first letter of name, map to number  $\rightarrow L = \{1 \dots 26\}$
  - map year similarly: sophomore = 2, junior = 3, etc  $\rightarrow Y = \{0 \dots 9\}$
  - $h(\text{student}) = 10 \cdot L + Y \rightarrow$  unique for any value in  $\{L, Y\}$



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



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



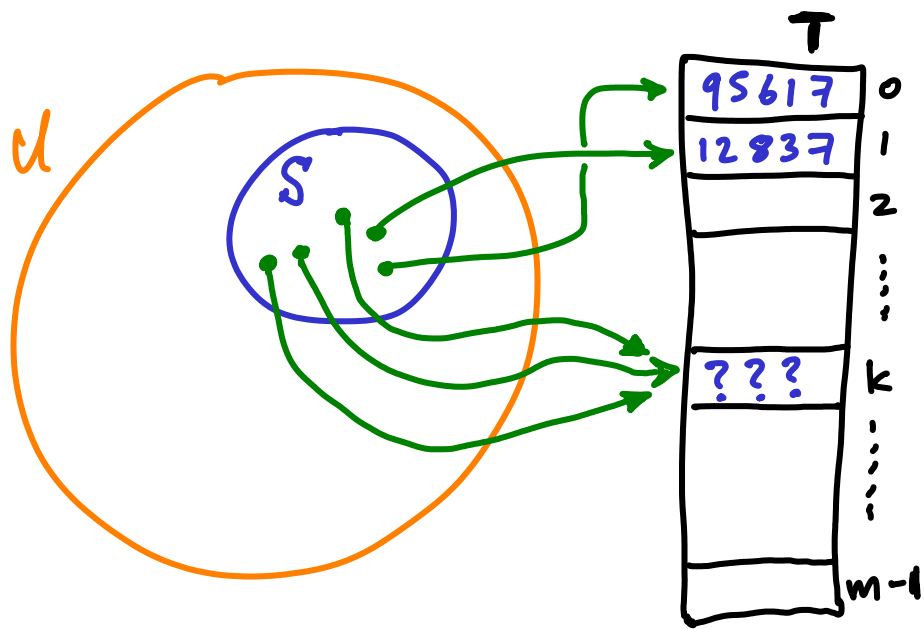
- Could use more of the given info to design a more complicated  $h()$ 
  - ↳ might minimize collisions

- Could use more of the given info to design a more complicated  $h()$ 
  - ↳ might minimize collisions
- But that involves costly processing

- 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)

- 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



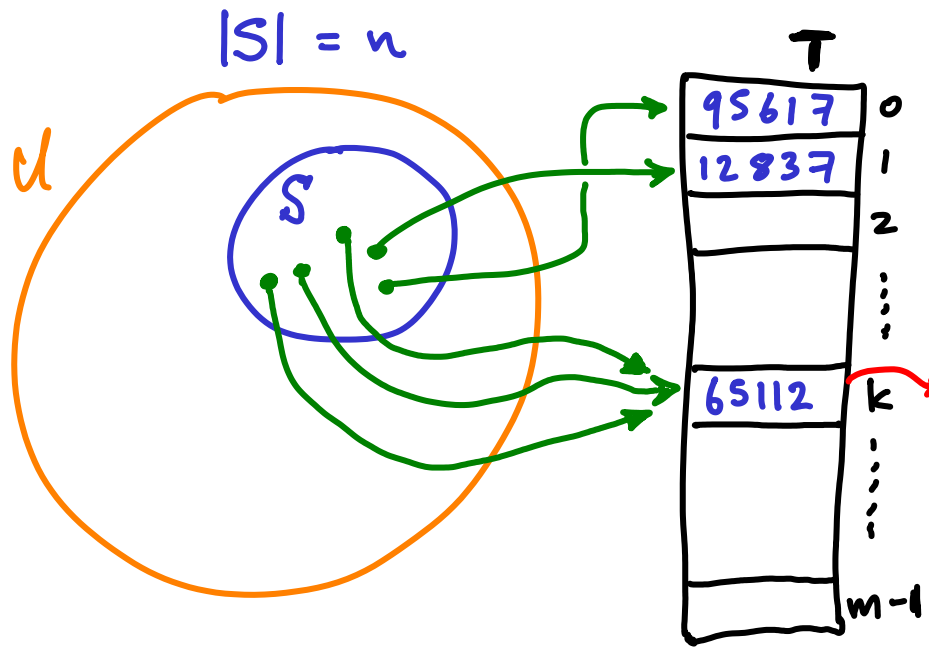


What if many keys map to same slot?

$$h(65112) = k$$

$$h(2315) = k$$

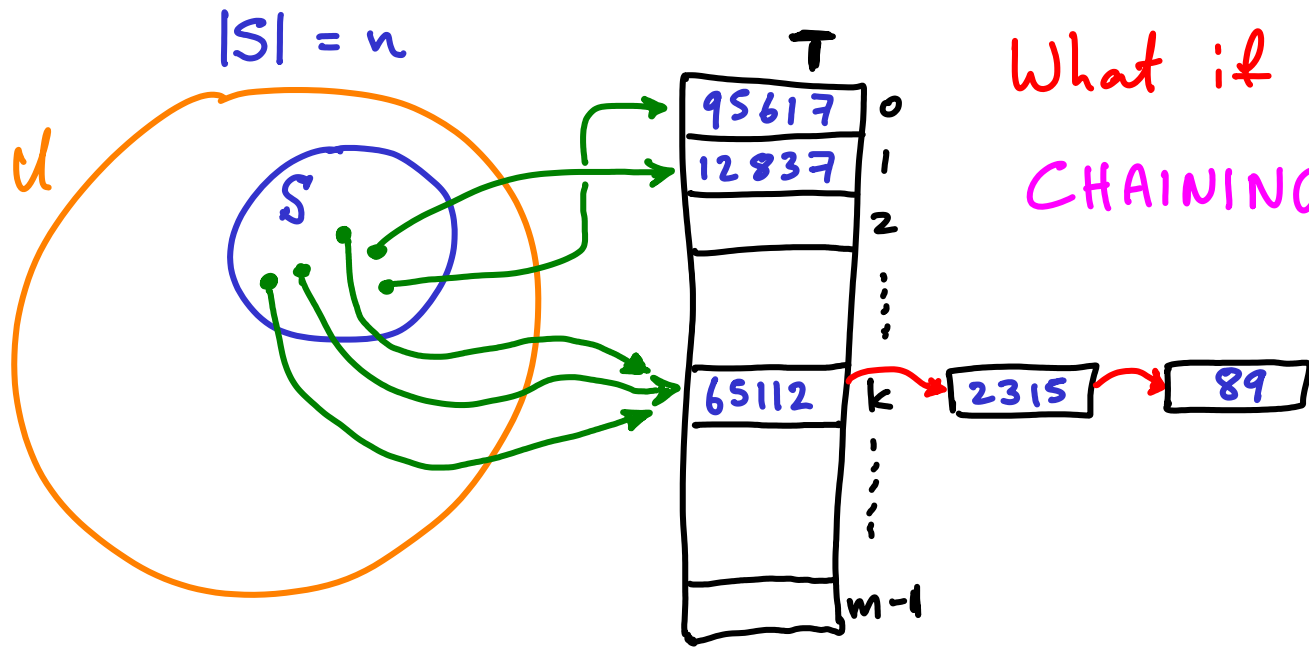
$$h(89) = k$$



What if many keys map to same slot?

CHAINING: Make a linked list.

Worst-case time for  
search, insert, delete?

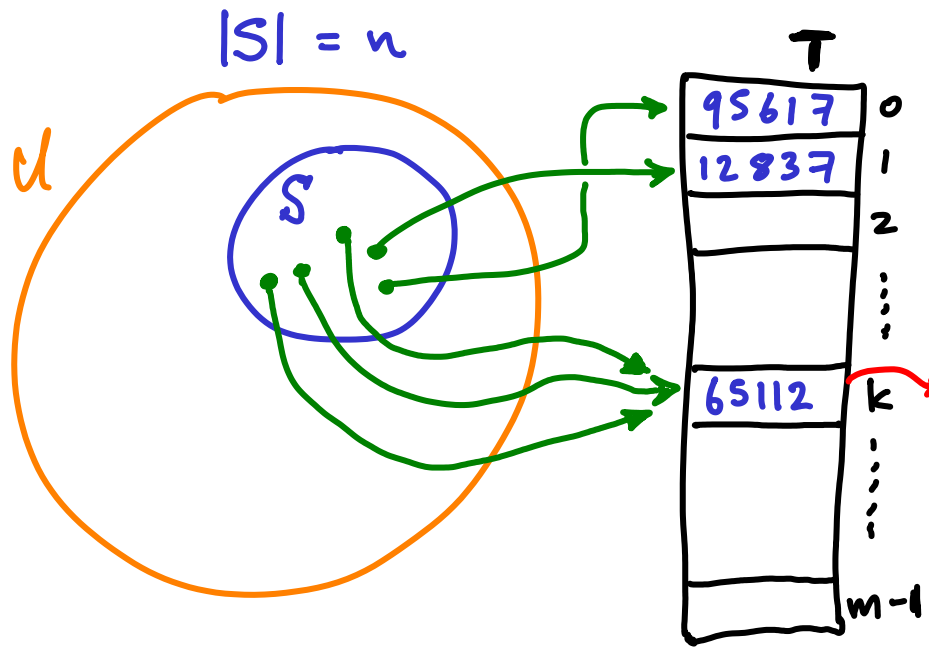


What if many keys map to same slot?

CHAINING: Make a linked list.

Insert =  $\Theta(1)$

Search/Delete =  $O(n)$   
list size



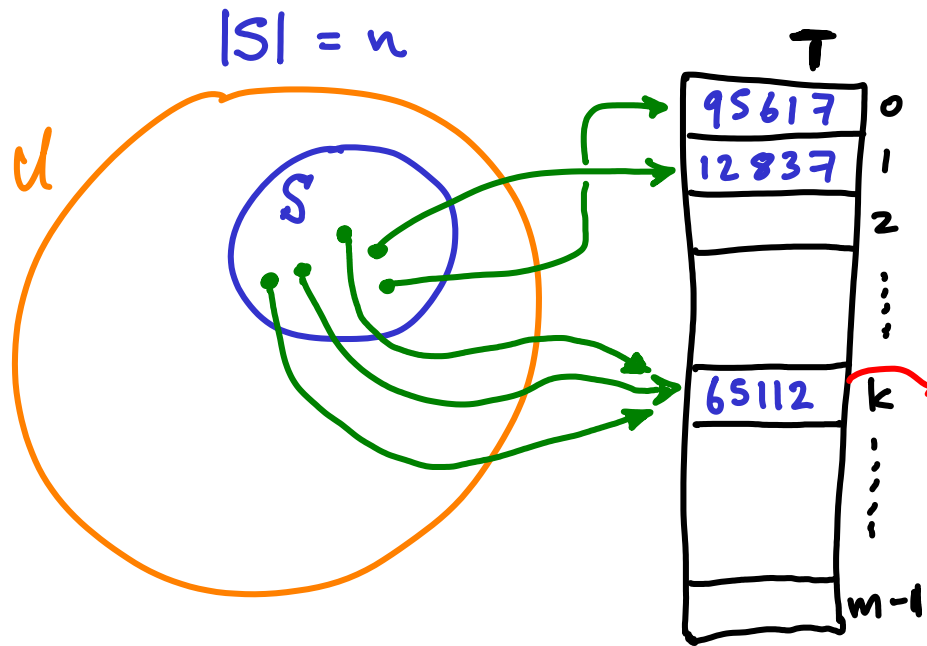
What if many keys map to same slot?

CHAINING: Make a linked list.

Insert =  $\Theta(1)$

Search/Delete =  $O(n)$   
list size

Must be consistent for each key



What if many keys map to same slot?

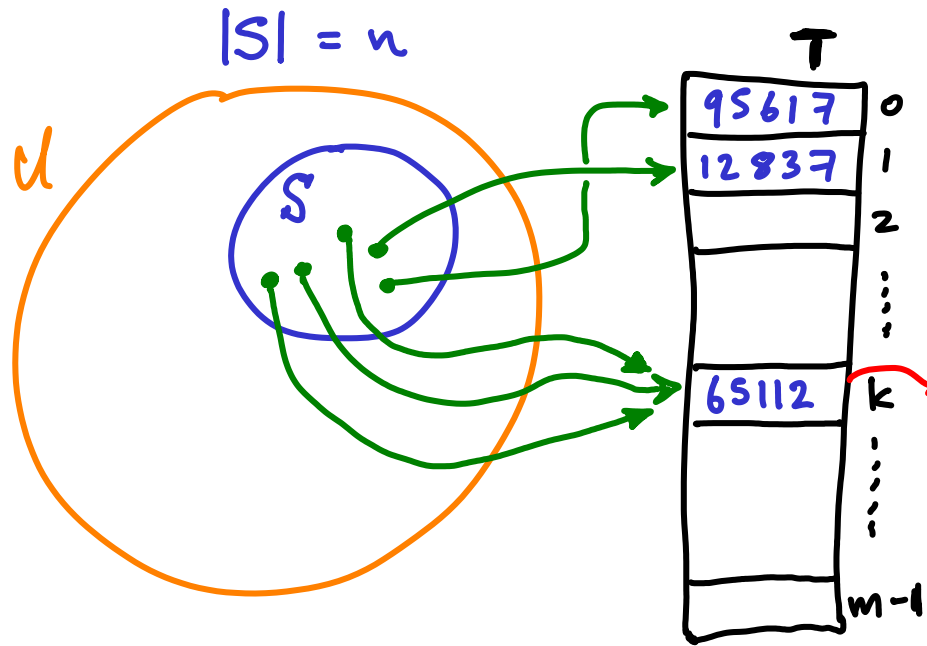
CHAINING: Make a linked list.

Insert =  $\Theta(1)$

Search/Delete =  $O(n)$   
list size

Must be consistent for each key

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



What if many keys map to same slot?

CHAINING: Make a linked list.

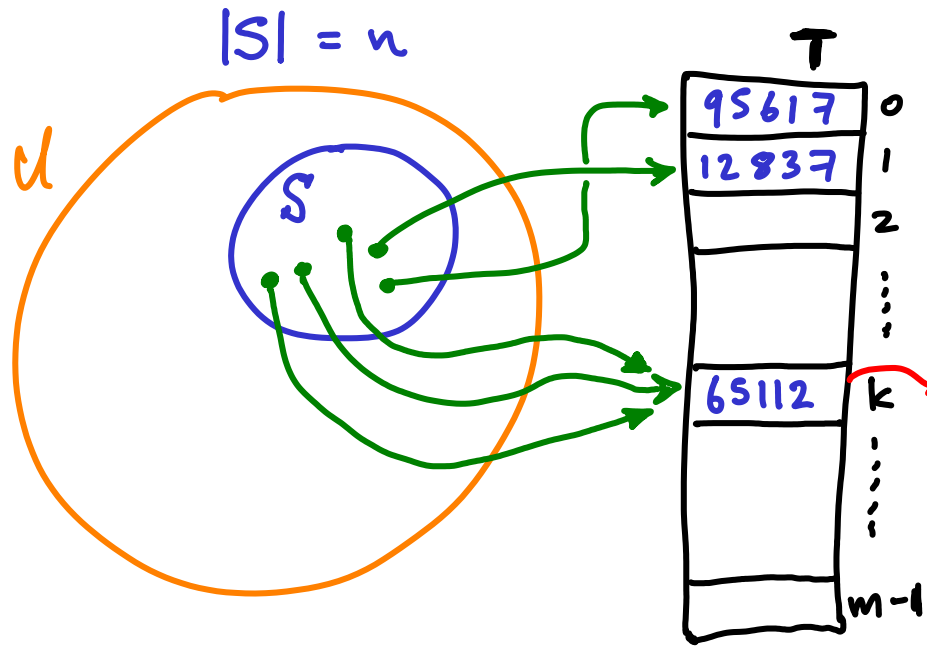
Insert =  $\Theta(1)$

Search/Delete =  $O(n)$   
list size

Must be consistent for each key

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

$n > m$ : COLLISIONS are  
unavoidable



What if many keys map to same slot?

CHAINING: Make a linked list.

Insert =  $\Theta(1)$

Search/Delete =  $O(n)$   
list size

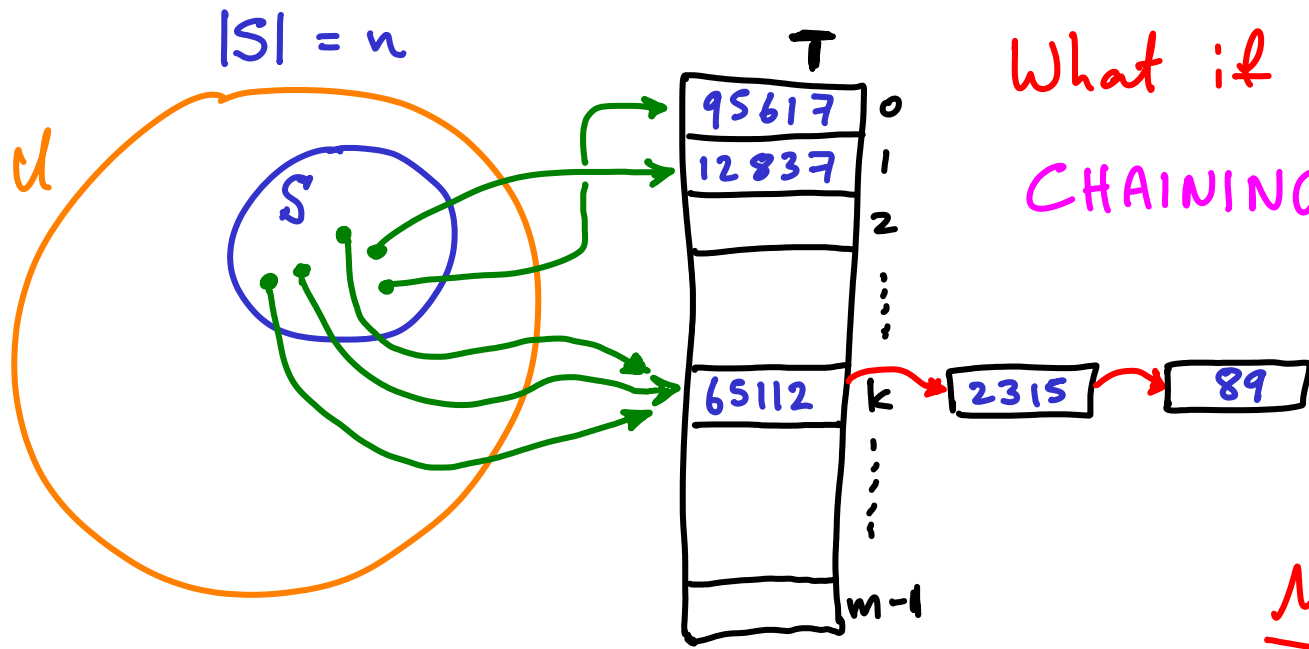
Must be consistent for each key

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 =  $\Theta(1)$

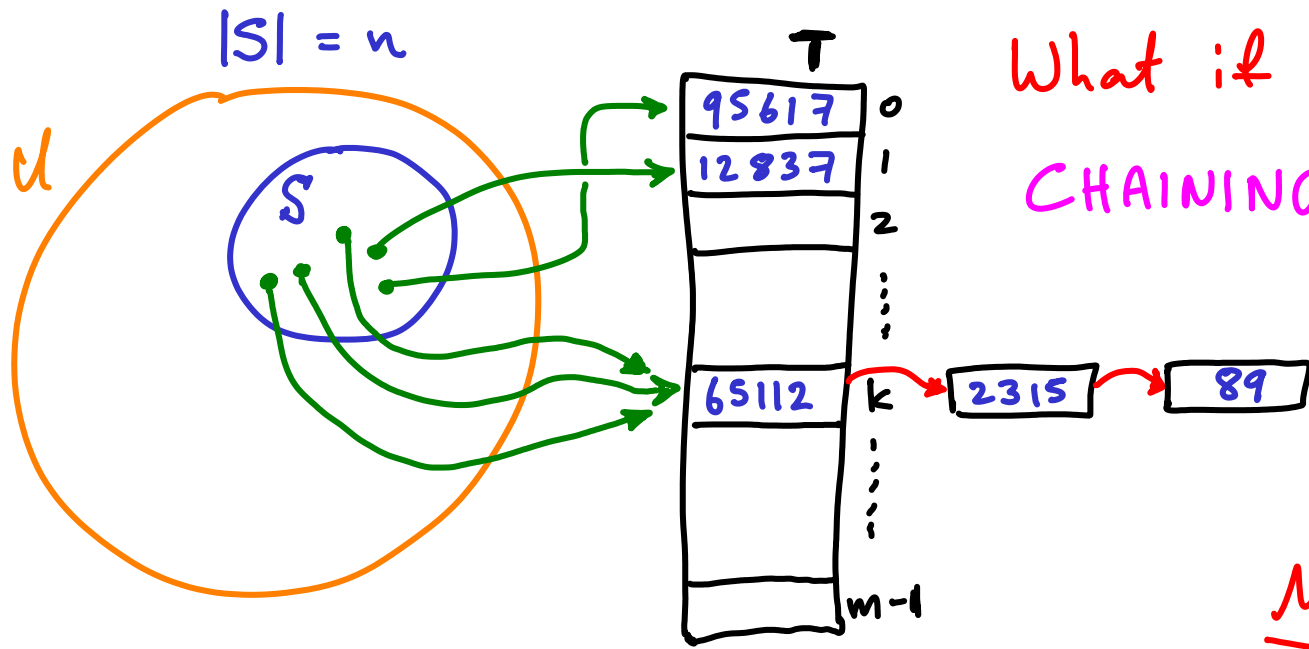
Search/Delete =  $O(n)$   
list size

Must be consistent for each key

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

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





What if many keys map to same slot?

CHAINING: Make a linked list.

Insert =  $\Theta(1)$

Search/Delete =  $O(n)$   
list size

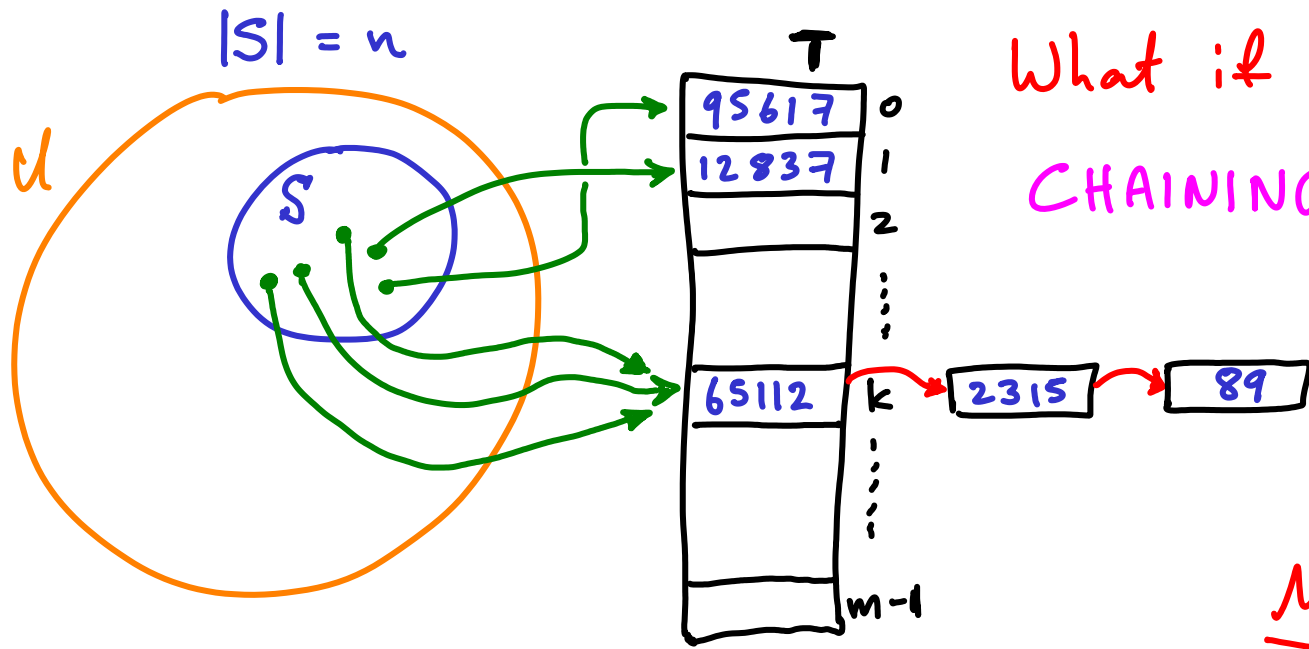
Must be consistent for each key

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

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

Average # keys per slot =  $\frac{n}{m}$

average size of linked list.



What if many keys map to same slot?

CHAINING: Make a linked list.

Insert =  $\Theta(1)$

Search/Delete =  $O(n)$   
list size

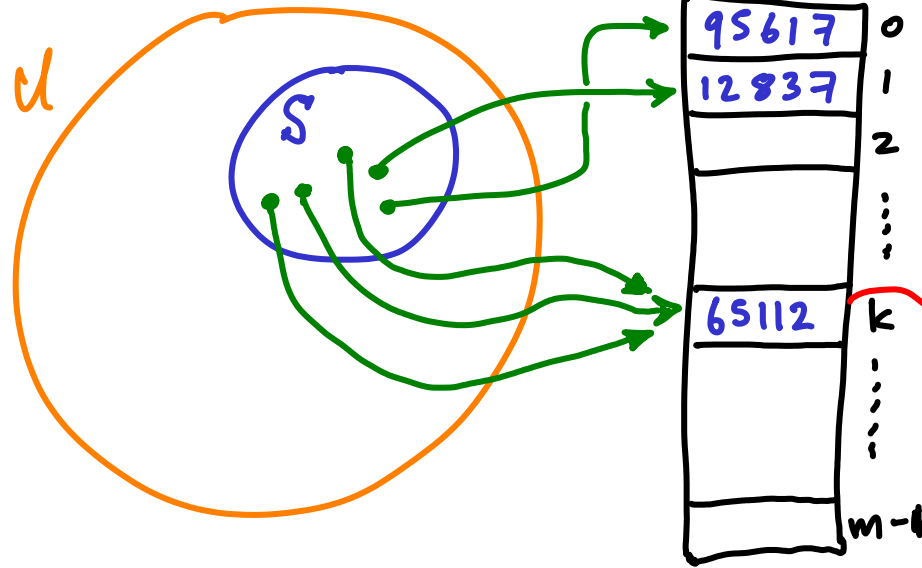
Must be consistent for each key

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

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

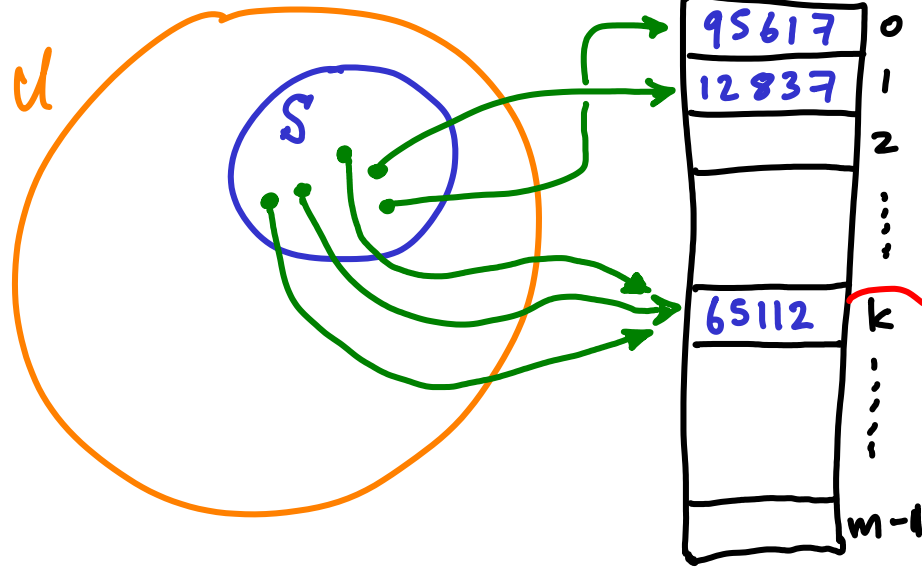
Average # keys per slot =  $\frac{n}{m} = \alpha$  = "load factor"  
average size of linked list.

$$|S| = n$$



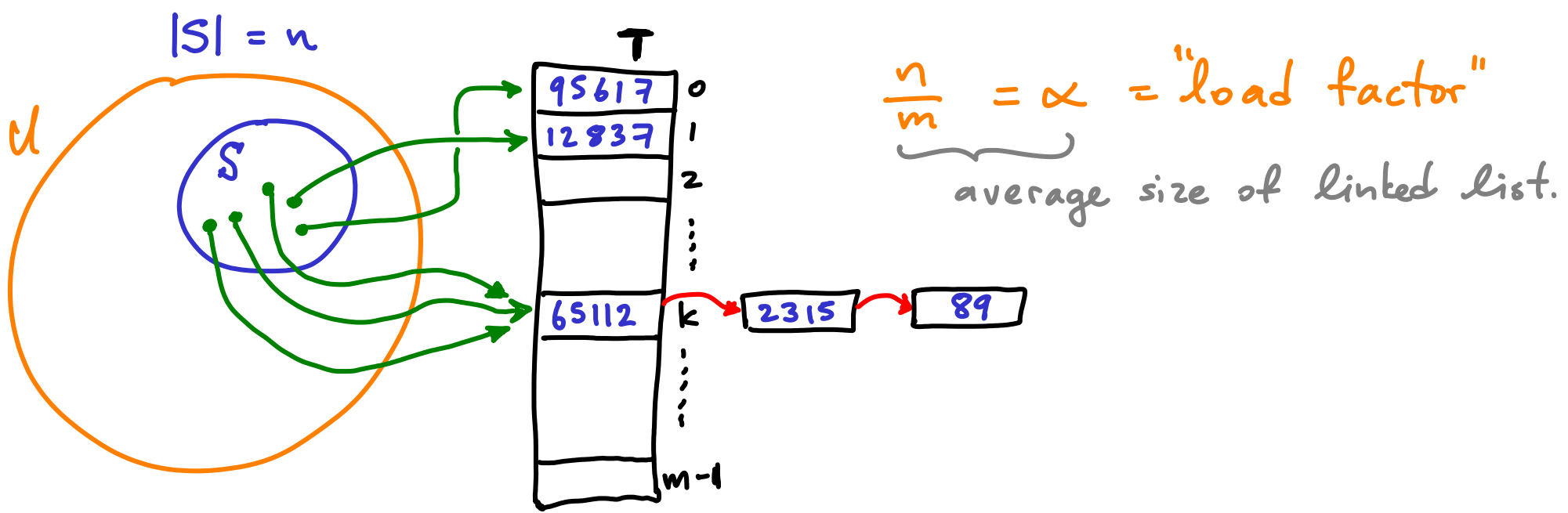
$\frac{n}{m} = \alpha = \text{"load factor"}$   
average size of linked list.

$$|S| = n$$



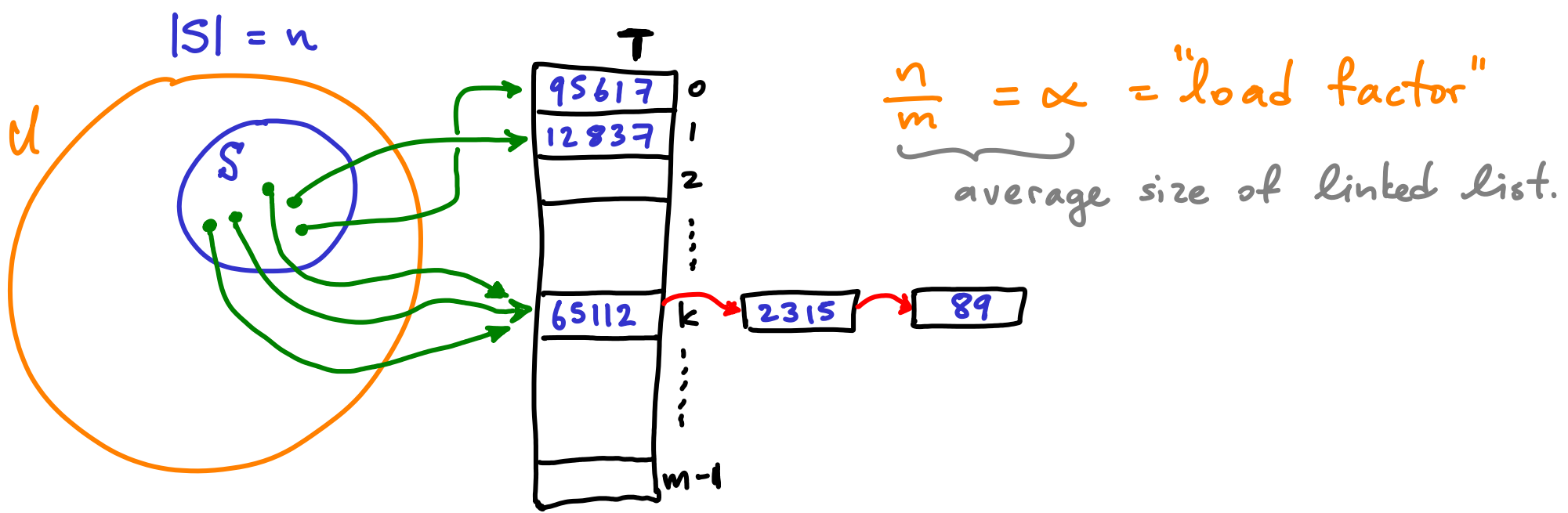
$\frac{n}{m} = \alpha = \text{"load factor"}$   
average size of linked list.

Expected time of search (and delete)



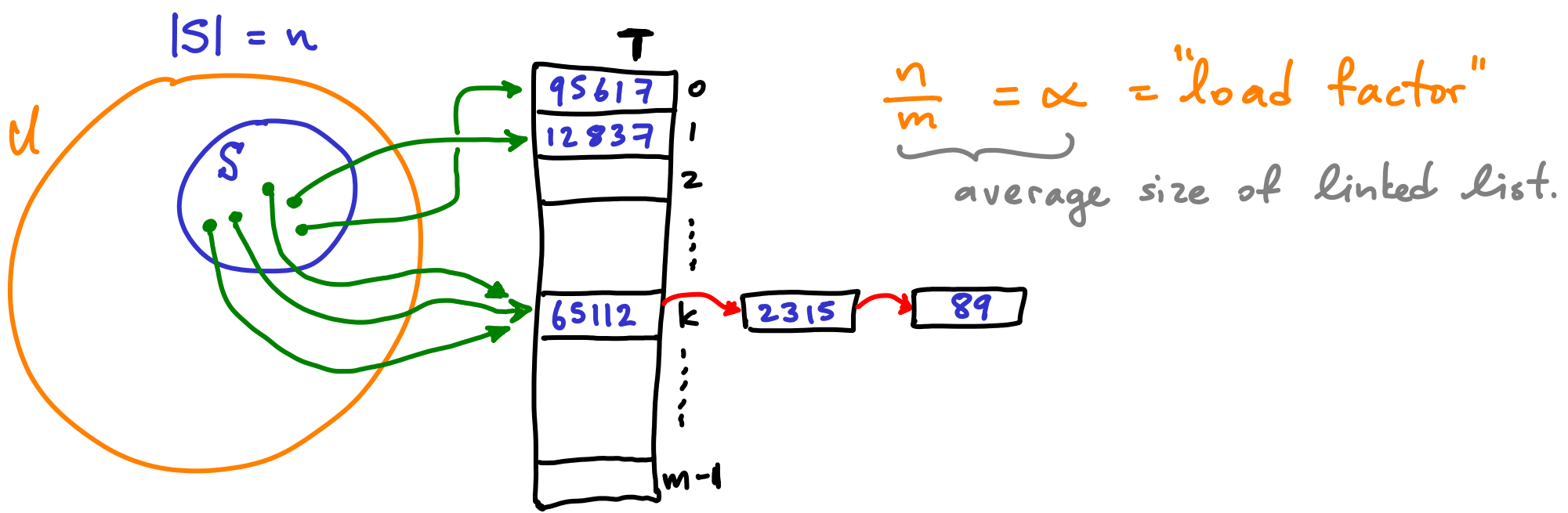
Expected time of search (and delete)

1)  $h(\text{key}) \rightarrow \text{slot\#} \rightarrow$  Assume  $h()$  takes  $\Theta(1)$  to evaluate



## Expected time of search (and delete)

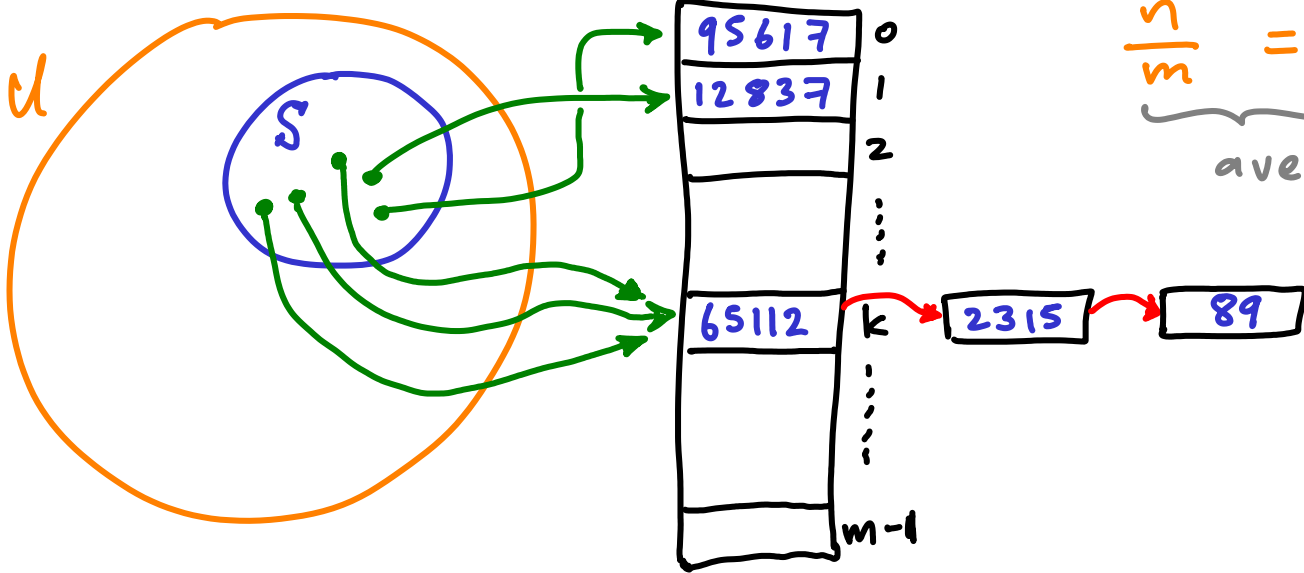
- 1)  $h(\text{key}) \rightarrow \text{slot \#}$   $\rightarrow$  Assume  $h()$  takes  $\Theta(1)$  to evaluate
- 2) scan list  $\rightarrow$  Expect to scan  $\geq$  half of a list



Expected time of search (and delete) =  $\Theta(1 + \alpha)$

- 1)  $h(\text{key}) \rightarrow \text{slot \#}$   $\rightarrow$  Assume  $h()$  takes  $\Theta(1)$  to evaluate
- 2) scan list  $\rightarrow$  Expect to scan  $\geq$  half of a list

$$|S| = n$$



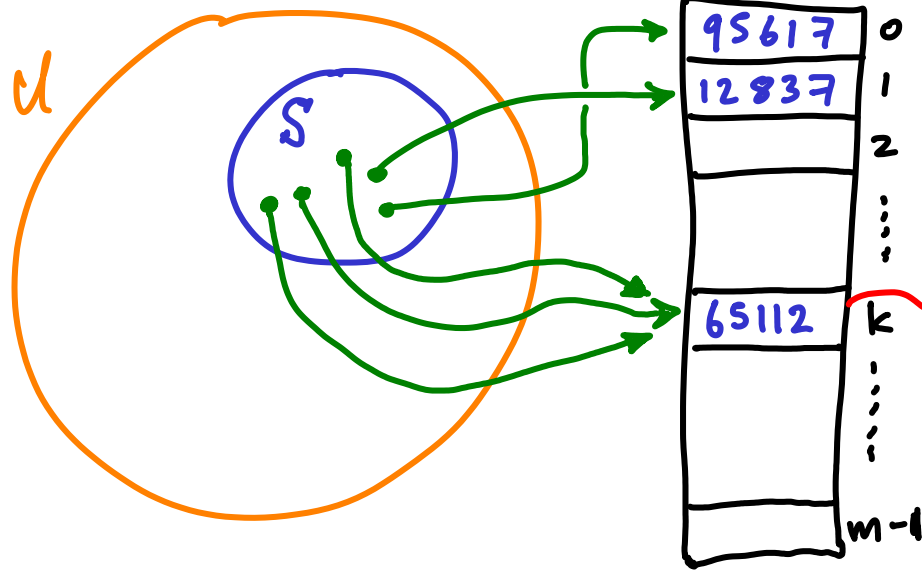
$\frac{n}{m} = \alpha = \text{"load factor"}$   
average size of linked list.

Expected time of search (and delete) =  $\Theta(1 + \alpha)$   
great if  $\alpha = \Theta(1)$

- 1)  $h(\text{key}) \rightarrow \text{slot \#}$   $\rightarrow$  Assume  $h()$  takes  $\Theta(1)$  to evaluate
- 2) scan list  $\rightarrow$  Expect to scan  $\geq$  half of a list



$$|S| = n$$



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

This analysis assumes simple uniform hashing condition

great if  $\alpha = \Theta(1)$

Expected time of search (and delete) =  $\Theta(1 + \alpha)$

1)  $h(\text{key}) \rightarrow \text{slot \#}$   $\rightarrow$  Assume  $h()$  takes  $\Theta(1)$  to evaluate

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

CHOOSING HASHING FUNCTIONS depending on keys and  $m$ .

CHOOSING HASHING FUNCTIONS depending on keys and  $m$ .

Objective: get uniform distribution of keys to slots - always

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"

How good is this?

## CHOOSING HASHING FUNCTIONS depending on keys and $m$ .

Objective: get uniform distribution of keys to slots - always

Ex:  $h(k) = k \bmod m$  } If  $S = \text{integers}$  then it's fine.  
"Division method" } ...but if  $S = m \cdot i$  for  $i = 1, 2, 3$  etc FAIL

## CHOOSING HASHING FUNCTIONS depending on keys and $m$ .

Objective: get uniform distribution of keys to slots - always

Ex:  $h(k) = k \bmod m$  } If  $S = \text{integers}$  then it's fine.  
"Division method" } ...but if  $S = m \cdot i$  for  $i = 1, 2, 3$  etc FAIL

We don't want any specific input pattern to affect uniformity.

## CHOOSING HASHING FUNCTIONS depending on keys and $m$ .

Objective: get uniform distribution of keys to slots - always

Ex:  $h(k) = k \bmod m$  } If  $S = \text{integers}$  then it's fine.  
"Division method" } ...but if  $S = m \cdot i$  for  $i = 1, 2, 3$  etc FAIL

We don't want any 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  $k \bmod m = ?$  (does what?)

$r=6$  :  $k = 1011000111011010$



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

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

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)

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)

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

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

heuristic: pick  $A$  not close to any power of 2

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

$A = 1011001$   
 $k = 1101011$

$$\left. \begin{array}{l} A = 1011001 \\ k = 1101011 \end{array} \right\} A \cdot k = 1001010 \underbrace{0110011}_{\substack{\text{remains after} \\ \text{mod } 2^7}}$$

$$h(1101011) = 011$$

$$\begin{array}{l} \text{If we had } A = \underbrace{2^{w-1}}_{1000000} \rightarrow A \cdot k = 1101011 \underline{0000000} \\ \text{or, } A = 2^5 \rightarrow A \cdot k = 0011010 \underline{1100000} \end{array}$$

heuristic provides some "randomness" to the process.