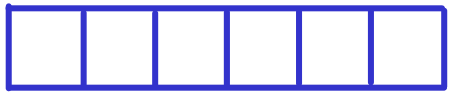
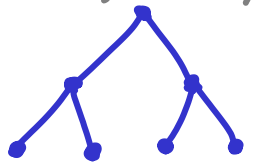
- SEARCH
- INSERT
- DELETE

} The 3 main operations that we perform on data structures

How fast can we do these?

---

	array	linked list	BST (RB, AVL, etc)
			
• SEARCH	$O(n)$ → [$O(\log n)$ sorted] ↳ by key $O(1)$ ↳ by index	$O(n)$	$O(\log n)$
• INSERT	$O(n)$ } [$O(1)$ amortized if not sorted]	$O(1)$ [$O(n)$ sorted]	$O(\log n)$
• DELETE	$O(n)$ } search + $O(1)$	$O(n)$ search + $O(1)$	$O(\log n)$

basic HASHING

- SEARCH
- INSERT
- DELETE

} O(1) expected with assumptions

not "expected worst-case",
it's just average time.

For some hashing methods, some operations can be O(1) worst-case.

The simplest form of hashing → Direct access table

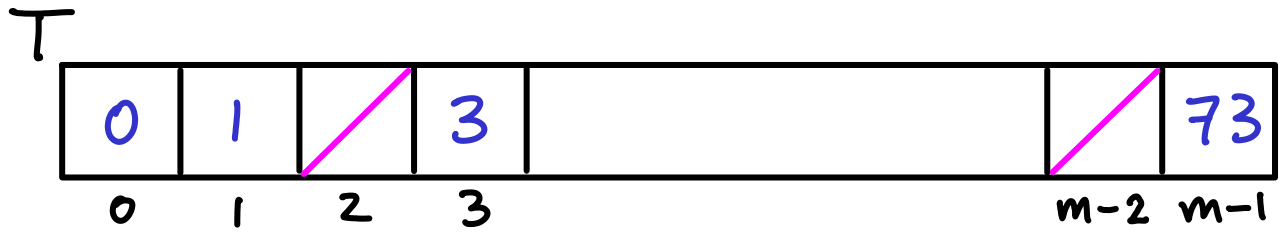
Assume keys are distinct

and come from a small set of possible values, U

Universe

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

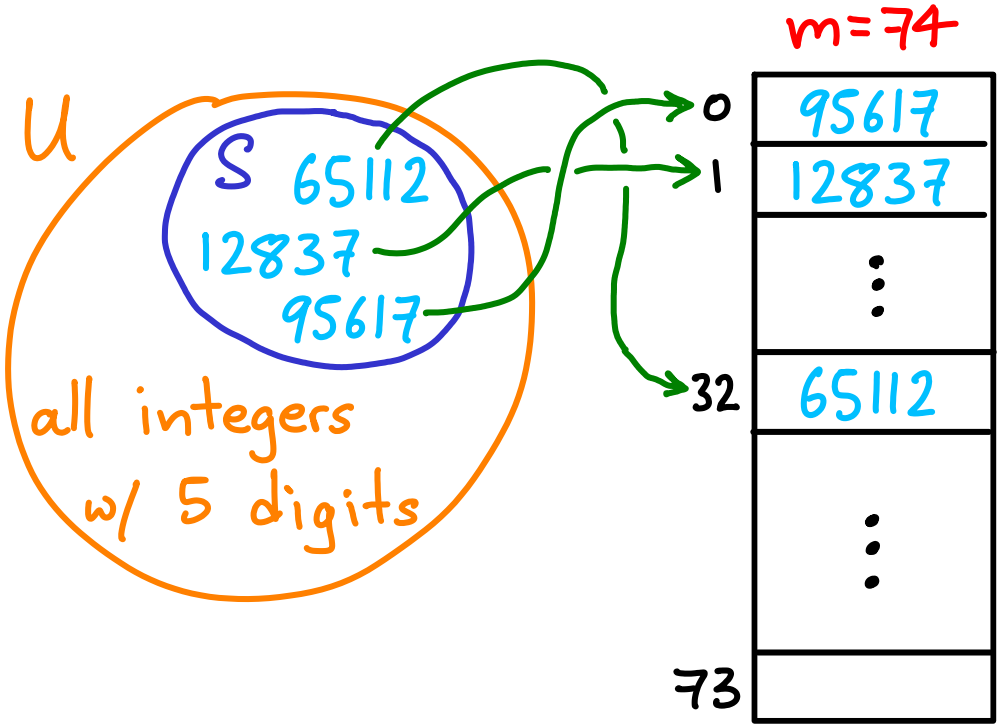
$m = 74$



search(T, 3) = 3 // insert(T, k) → T[k] = k // delete(T, k) → T[k] = ∅

Often U is larger than the available space, m

but we only need to deal with a subset S of U, where $|S| \leq m$



h: hash function

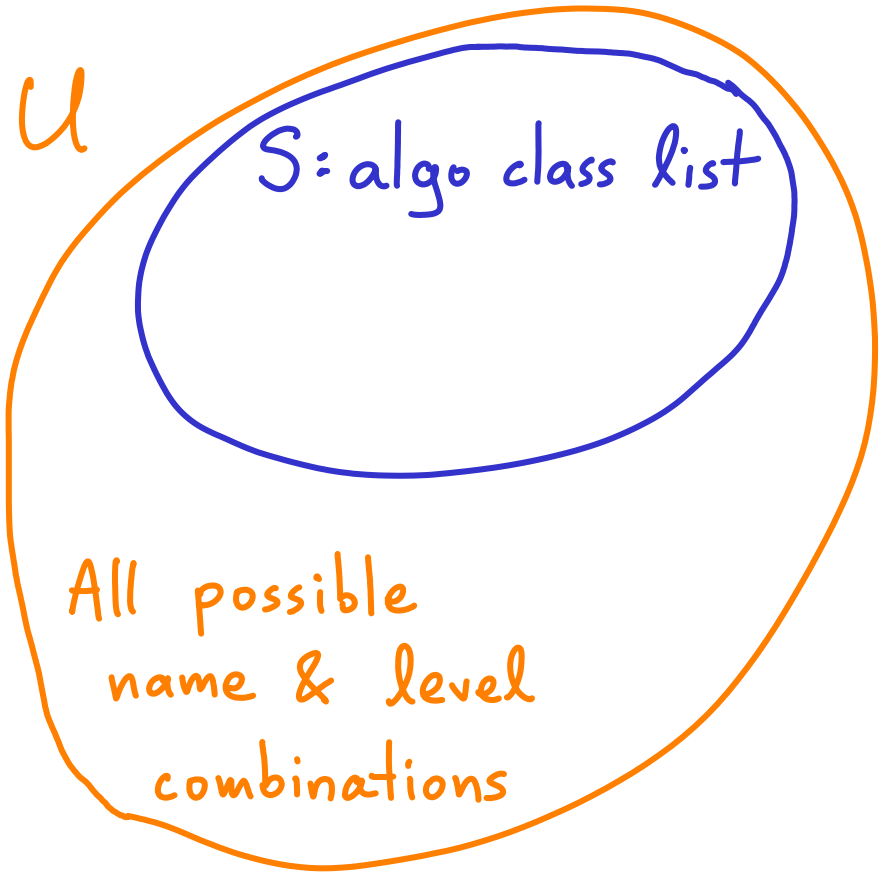
maps keys to T

$h(95617) = 0$

$h(12837) = 1$

$h(65112) = 32$

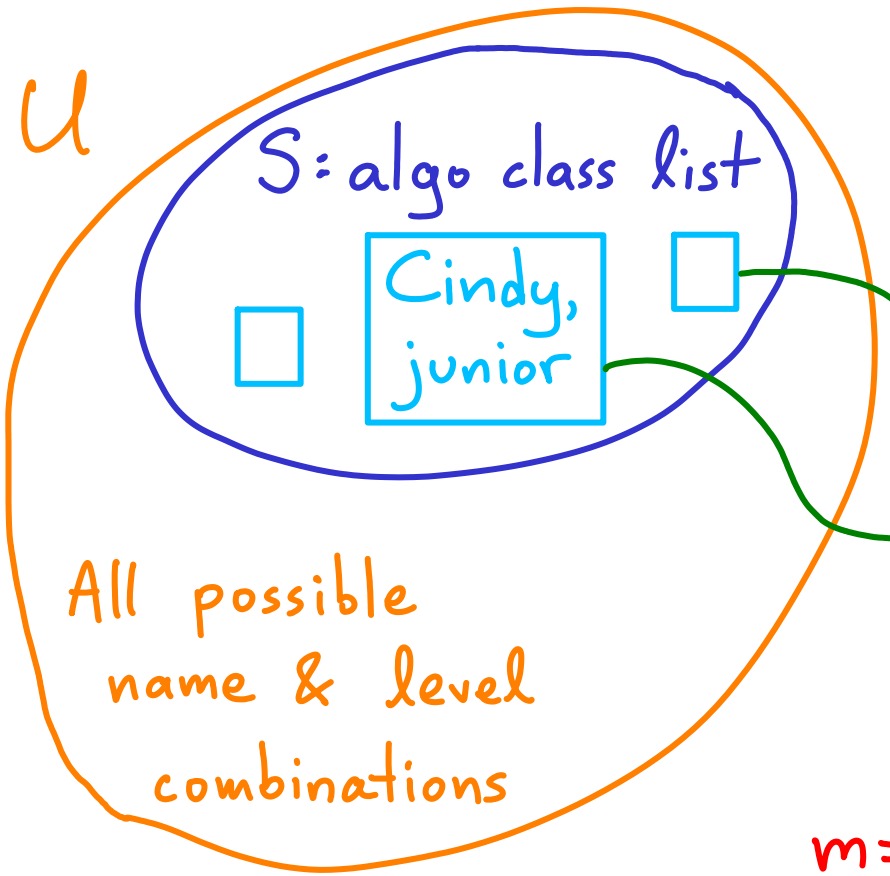
Example: look up this semester's ALGO students
using only their first name & academic level



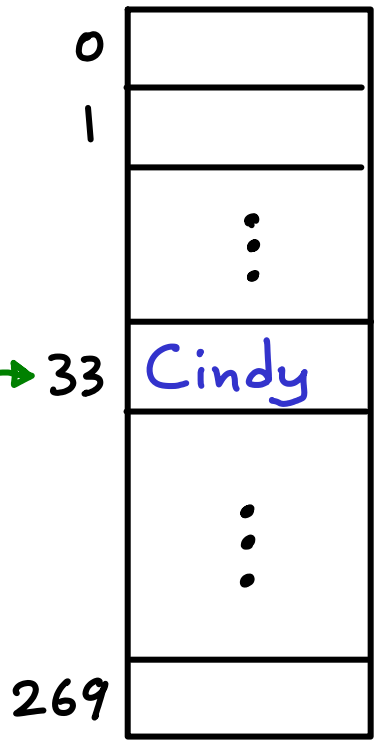
- take 1st letter of name, map to number → N = {1...26}
- map level similarly: junior=3, PhD=7, etc → L = {0...9}

$h(student) = h(L,N) = 10 \cdot N + L$

└ unique per pair N, L



h: 30+3 → 33



m=270

PROBLEMS?

(1) some permanently empty slots

(2) collisions

Could use more of the given info to design a more complicated h( )

↳ might reduce collisions

But that involves costly processing

and will need to be repeated if S changes (next semester)

We want to use a simple h( ) and deal with collisions