The 3 main operations on data structures:

- **SEARCH**: \( O(n) \) \( \text{[O(logn) sorted]} \)
- **INSERT**: \( O(n) \) (if maintaining sorted)
- **DELETE**: \( O(n) \) (if we don't want gaps)

How fast can we do these? (as seen so far in class)

- **SEARCH**
  - Basic: \( O(n) \)
  - Sorted: \( O(n) \) [as \( O(logn) \) is more efficient]
- **INSERT**
  - Basic: \( O(n) \)
  - Sorted: \( O(1) \) + search
- **DELETE**
  - Basic: \( O(n) \)
  - Sorted: \( O(n) \) [as \( O(logn) \) is more efficient]
basic Hashing

The 3 main operations on data structures:

```plaintext
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.
```
Direct access table: good when keys are distinct & come from a small distribution \( U \).

\[ U = \{0, 1, 2, \ldots, m-1\} \]

Say \( m = 74 \). Use an array: \( T \)

- search \((T, 2) = 2\)  
- insert \((T, k) \rightarrow T[k] = k\)  
- delete \((T, k) \rightarrow T[k] = \emptyset\)

\[ \text{All } \Theta(1) \]
If $U$ is larger than our available storage, $m$, but we are working with a subset $S$ of $U$, where $|S| \leq m$. 

$U$ represents the set of all integers with $S$ digits. 

$S$ represents a subset of $U$. 

$h$: hash function 

Maps keys to $T$. 

- $h(95617) = 0$ 
- $h(12837) = 1$ 
- $h(65112) = k$
Example: look up this semester's COMP160 students using name & academic level (year)

U

S: COMP 160 class list

All possible name & year combinations
\( h \) takes the first letter of a name, maps to a number \( L = \{1 \ldots 26\} \), maps year similarly (sophomore = 2, junior = 3, etc) \( Y = \{0 \ldots 9\} \), and \( h(\text{student}) = 10 \cdot L + Y \) is unique for any value in \( \{L, Y\} \).

\( S: \text{COMP 160 class list} \)

- Chao, junior

\( h = 30 + 3 \)

\( T \)

0 1 2 \ldots

Chao 33

\( m-1 < 270 \)

**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** = $O(n)$

---

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 = $\frac{1}{m}$

Average # keys per slot = $\frac{n}{m} = \alpha = \text{"load factor"}$

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) = \(\Theta(1+\alpha)\)

1) \(h(\text{key}) \rightarrow \text{slot}\# \rightarrow \text{Assume } h() \text{ takes } \Theta(1) \text{ to evaluate}\)
2) \(\text{scan list} \rightarrow \text{Expect to scan } \geq \text{ half of a list}\)
Choosing Hashing Functions depending on keys and $m$.

Objective: get uniform distribution of keys to slots - always

Ex: $h(k) = k \mod m \}$ If $S = \text{integers}$ then it's fine.

"Division method" $\}$ ... but if $S = m \times i$ for $i = 1, 2, 3$ etc \underline{FAIL}

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

(similar to sorting algo not working well on pre-sorted data)

"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 $k \mod m = k \mod 2^r$ keeps only last $r$ digits

$r=6 : k = 1011000111011010$

$h$ 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)

\[
h(k) = (A \cdot k) \mod 2^w \text{ right-shifted by } w-r
\]

\[
\text{\( \rightarrow \) some odd integer in } [2^{w-r} \ldots 2^w-1] \text{ \( \rightarrow \) \( w \)-bit \# with leading 1.}
\]

heuristic: pick \( A \) not close to any power of 2

ex: \( m = 2^3 : r = 3 \)

\[
\begin{align*}
A &= 1011001 \\
A \cdot k &= 1001010 \underbrace{0110011}_{w-r} \\
&\text{remains after mod } 2^7
\end{align*}
\]

\[
h(1101011) = 011
\]

If we had \( A = 2^{w-1} \) \( \rightarrow \) \( A \cdot k = 110101100000000 \)

\[
\text{or, } A = 2^5 \quad \rightarrow \quad A \cdot k = 001101011000000
\]

heuristic provides some "randomness" to the process.