The 3 main operations on data structures:

- **SEARCH**
  - \(O(n)\) [\(O(\log n)\) sorted]

- **INSERT**
  - \(O(n)\) (if maintaining sorted)

- **DELETE**
  - \(O(n)\) (if we don't want gaps)

How fast can we do these?

- **SEARCH**
  - \(O(\log n)\)

- **INSERT**
  - \(O(1)\)

- **DELETE**
  - \(O(n)\) \(O(1) + \text{search}\)
The 3 main operations on data structures:

\[
\text{SEARCH} \quad \text{INSERT} \quad \text{DELETE}
\]

\[O(1)\] 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., \( U = \{0, 1, 2, ..., m-1\} \)

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

\[
\begin{array}{ccccccccccccc}
0 & 1 & 2 & \cdots & \phi & \phi & \cdots & \phi & 73 \\
0 & 1 & 2 & k & \cdots & & & m-1
\end{array}
\]

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

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

$h$: 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)

$U$

$S$: class list

All possible name & year combinations

$T$

0

1

2

... 

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

**U**

- \( S \):
  - class list
  - Cindy, junior

**All possible name & year combinations**

**T**

- \( h = 30 + 3 \)
- Cindy

**PROBLEMS?**

1. Some permanently empty slots
2. Other \( h = 33 \)?

\( m - 1 < 270 \)
• Could use more of the given info to design a more complicated \( h() \)
  \[ \Rightarrow \] 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
If CHAINING, we don’t need \( n < m \).

\( n > m \): **COLLISIONS are unavoidable**

If CHAINING, we don’t need \( n < m \).

\( n > m \): **COLLISIONS are unavoidable**

minimize collisions by spreading \( S \) into \( T \) evenly

\( \rightarrow \) 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 = $\frac{n}{m} = \alpha = \text{“load factor”}$
$|S| = n$

$\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 \hspace{1cm} \rightarrow$ Expect to scan $\gg$ half of a list
Choosing Hashing Functions depending on keys and m.

Objective: get uniform distribution of keys to slots - always

Ex: \( h(k) = k \mod m \) \( S = \) 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 \( k \mod m = k \mod 2^r \) keeps only last \( r \) digits.

\[
r=6 : \quad k = 10110001111111010
\]

\( h \) 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)

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

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

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

ex: \( m = 2^3 : r = 3 \quad w = 7 \)

\[
A = 1011001 \quad \Rightarrow A \cdot k = 1001010 \quad \underbrace{0110011}_{\text{remains after } \mod 2^7}
\]

\[
h(1101011) = 011
\]

If we had \( A = 2^{w-1} \) \( \Rightarrow A \cdot k = 1101011\underbrace{0000000}_{\text{some "randomness" to the process.}} \)

or, \( A = 2^5 \) \( \Rightarrow A \cdot k = 0011010\underbrace{1100000}_{\text{same}} \)