The 3 main operations on data structures:

\{ \text{SEARCH, INSERT, DELETE} \} \quad O(1)

expected with assumptions
The 3 main operations on data structures:

- **SEARCH**
- **INSERT**
- **DELETE**

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

E.g. \( U = \{0, 1, 2, \ldots, m-1\} \)

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

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73
```

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

All \( \Theta(1) \)
If $U$ is larger than our available storage, this is much worse.

$U$: all possible keys
$S$: subset of $U$: the keys being used.

Integers with $S$ digits
If \( U \) is larger than our available storage, this is much worse.

\[ T = \begin{bmatrix} 79123 & 65112 & \cdots & \phi & \phi & 12837 & \cdots & 95617 \end{bmatrix} \]

\( U \): all possible keys
\( S \): subset of \( U \): the keys being used.

\( h \): hash function
maps keys to \( T \).

\( h(12837) = 1 \)
\( h(65112) = k \)
I want to quickly access data for students. Student ID's are annoying. I remember names & years.

U

$S: \text{Comp 160 class list}$

All possible name & year combinations

T

0

1

2

\vdots

m-1
\[ h : \begin{cases} 
\text{take first letter of name, map to number } L = \{1 \ldots 26\} \\
\text{map year/level similarly: } Y = \text{sophomore} = 2, \text{ junior} = 3, \text{ etc.} \\
h(\text{student}) = 10 \cdot L + Y \rightarrow \text{unique for any value in } \{1, Y\} 
\end{cases} \]

\[ Y = \{0 \ldots 9\} \]

\[ h = 30 + 3 \]

All possible name & year combinations
We resume with the example of mapping class records to an array, based on the first letter of a student's first name, and the year they are in.

There is a problem if several students map to the same array slot.

This is unavoidable if the array is smaller than the number of students. Even if it is bigger though, the function is so simple that there can be "collisions". I could be more careful and design a hash function that will use much more information (full name + year + ID + ???) so that every student maps to a unique slot. But, that is a lot of work, requires looking at the input distribution very carefully, and besides I'd have to do it all over next year. Instead the simple function can be used year after year, as long as I can confidently deal with collisions some way.
take first letter of name, map to number \( L = \{1 \ldots 26\} \)

\[
h : \begin{cases} 
  \text{map year/level similarly: } Y = \text{sophomore} = 2, \text{ junior} = 3, \text{ etc.} \\
  h(\text{student}) = 10 \cdot L + Y \rightarrow \text{unique for any value in \( \{L, Y\} \)}
\end{cases}
\]

All possible name & year combinations

\( S \): COMP 160 class list

\( h = 30 + 3 \)

cyrus, junior

PROBLEMS?

(1) Some permanently empty slots
(2) Other \( h = 33 \)?
What if many keys map to same slot?

\[ h(65112) = k \]
\[ h(2315) = k \]
\[ h(89) = k \]

\[ T(k) = ? \]
What if many keys map to same slot?

Make a linked list.

Insert = $\Theta(1)$

Search/Delete* = $\Theta(n)$

*O(1) if not accessing by key (i.e., via search)

$|S| = n$
What if many keys map to the same slot?
Make a linked list.

- Insert = $\Theta(i)$
- Search/Delete = $\Theta(n)$

Must be consistent for each key.

$|S| = n$

$S$ into $T$ evenly

Collisions are unavoidable

$n > m$
What if many keys map to same slot? Make a linked list.

- Insert = $\Theta(1)$
- Search/Delete = $\Theta(n)$

Must be consistent for each key

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

- Probability two given keys collide = $\frac{1}{m}$
- Average # keys per slot $= \frac{n}{m} = \infty$ = "load factor"

(simple uniform hashing)

(even though your function is deterministic, the more random it appears to be, the more this analysis is applicable)

Average size of linked list.
$|S| = n$

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

```
search(key) : h(key) -> slot# ;
```

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

$\frac{n}{m} = \alpha = \text{"load factor"}$

average size of linked list.

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

expect to search $\frac{1}{2}$ list

see CLRS (overcomplicated?)
Choosing Hashing Functions depending on keys and m.

Objective: get uniform distribution of keys to slots - always

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

“Division method” \( \) ... but if \( S = m \cdot i \) for \( i = 1, 2, 3 \) etc FALL

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)

So, at least choose $m$ : prime & not close to power of 2
"Multiplication Method"

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

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

1000000   1111111

OK, and ending in a 1 too if odd

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

$w = 7$

$A = 1011001$

$k = 1101011$
"Multiplication Method"

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

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

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

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

ex: $m=2^3 \div r=3$  
$A=1011001$  
$k=1101011$  
$\Rightarrow A \cdot k = 10010100110011$

$h(1101011) = 011$

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

or, $A=2^5 \rightarrow A \cdot k = 001101011000000$
Resolving collisions w/ Open Addressing assuming \( n \leq m \)

The point is to avoid auxiliary linked lists.

Instead, create a probe sequence as a function of key value. (\( \rightarrow \) permutation of slots to try.)

works because table is large

one reason: use as much space as possible in advance for the table itself

ex: \( h(64) \) → 9, 2, 4, 8, 1, 3, 11, 7, 10, 5, 6.

Insert(64):
- Try \( T[9] \): full
- Try \( T[2] \): full
- Try \( T[4] \): full
- Try \( T[8] \): ok

Search(64) follows same sequence.
Would return "not found" after 4 attempts.
ex: $h(64) \rightarrow 9, 2, 4, 8, 1, 3, 11, 7, 10, 5, 6.$

Really this is $h(k, i)$

- $h(64, 1) = 9$
- $h(64, 2) = 2$
- $h(64, 3) = 4$

\[\text{etc.}\]
ex: $h(64) \rightarrow 9, 2, 4, 8, 1, 3, 11, 7, 10, 5, 6.$

Really this is $h(k, i)$  

$h(64, 1) = 9 / h(64, 2) = 2 / h(64, 3) = 4 / \ldots$

Notice I didn’t mention ‘Delete.’

Delete(64):  

$h(64, 1) = 9$, occupied by 2014 so…

$h(64, 2) = 2$, occupied by 43 so…

$h(64, 3) = 4$, occupied by 78 so…

$h(64, 4) = 8$, found 64, DELETE IT.

OK?
**ex:** $h(64) \rightarrow 9, 2, 4, 8, 1, 3, 11, 7, 10, 5, 6.$

Really this is $h(k, i)$

$h(64, 1) = 9$ / $h(64, 2) = 2$ / $h(64, 3) = 4$ / etc.

Notice I didn’t mention Delete.

Delete(64): $h(64, 1) = 9$, occupied by 2014 so...

$h(64, 2) = 2$, occupied by 43 so...

$h(64, 3) = 4$, occupied by 78 so...

$h(64, 4) = 8$, found 64, DELETE IT.

But what if $h(103) \rightarrow 4, 8, 2, 7, 1, 10, 11, 3, 5, 9, 6$ ?

As mentioned in class, suppose you inserted 64, then inserted 103, then deleted 64, then searched for 103.
ex: \(h(64) \rightarrow 9, 2, 4, 8, 1, 3, 11, 7, 10, 5, 6.\)

Really this is \(h(k, i)\)

\(h(64, 1) = 9 \quad h(64, 2) = 2 \quad h(64, 3) = 4 \quad \text{etc.}\)

Notice I didn’t mention Delete.

\[\begin{array}{|c|c|}
\hline
1 & 36 \\
2 & 43 \\
3 & 78 \\
4 & 5 \\
5 & 103 \\
6 & 64 \\
7 & 2014 \\
\hline
\end{array}\]

\(\text{Delete}(64):\)

\(h(64, 1) = 9, \text{ occupied by 2014 so...}\)
\(h(64, 2) = 2, \text{ occupied by 43 so...}\)
\(h(64, 3) = 4, \text{ occupied by 78 so...}\)
\(h(64, 4) = 8, \text{ found 64, DELETE IT.}\)

But what if \(h(103) \rightarrow 4, 8, 2, 7, 1, 10, 11, 3, 5, 9, 6?\)

\(\text{Search}(103): h(103, 1) = 4, \text{ occupied by 78}\)
\(h(103, 2) = 8, \text{ empty: declare 103 not in T.}\)
ex: $h(64) \rightarrow 9, 2, 4, 8, 1, 3, 11, 7, 10, 5, 6$.

Really this is $h(k, i)$, where $h(64, 1) = 9$, $h(64, 2) = 2$, $h(64, 3) = 4$, etc.

Notice I didn't mention Delete.

Delete(64): $h(64, 1) = 9$, occupied by 2014 so...
$h(64, 2) = 2$, occupied by 43 so...
$h(64, 3) = 4$, occupied by 78 so...
$h(64, 4) = 8$, found 64, DELETE IT.

But what if $h(103) \rightarrow 4, 8, 2, 7, 1, 10, 11, 3, 5, 9, 6$?

Search(103): $h(103, 1) = 4$, occupied by 78
$h(103, 2) = 8$, empty: declare 103 not in T.

Could use special "deleted" markers, but search time increases.
Typical probing sequences

Linear probing: \( h(k, i) = (h(k, 0) + i) \mod m \) \( \sim h(k) \) and wrap around.
... tends to generate clusters.

Double hashing: \( h(k, i) = (h_1(k) + i \cdot h_2(k)) \mod m \)

Heuristic: choose \( m = 2^r \) \& \( h_2(k) : \text{odd} \).
otherwise \( \frac{1}{2} \text{-table empty} \)
Typical probing sequences

Linear probing: \( h(k,i) = (h(k,0) + i) \mod m \) \( \sim h(k) \) and wrap around.

...tends to generate clusters.

Double hashing: \( h(k,i) = \left( h_1(k) + i \cdot h_2(k) \right) \mod m \)

"random" offset

Heuristic: choose \( m = 2^r \) & \( h_2(k) \text{ odd} \).

\( \frac{1}{2} \) table empty otherwise.

See also quadratic probing in CLRS

\( \Rightarrow \) still only generates \( m \) probe sequences

Double hashing generates \( O(m^2) \) probe sequences: better