<table>
<thead>
<tr>
<th>Lecture</th>
<th>Instructor</th>
<th>Time</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lecture A</td>
<td>Tiefenbruck</td>
<td>MWF 9-9:50am</td>
<td>Center 212</td>
</tr>
<tr>
<td>Lecture B</td>
<td>Jones</td>
<td>MWF 2-2:50pm</td>
<td>Center 214</td>
</tr>
<tr>
<td>Lecture C</td>
<td>Tiefenbruck</td>
<td>MWF 11-11:50am</td>
<td>Center 212</td>
</tr>
</tbody>
</table>

[http://cseweb.ucsd.edu/classes/wi16/cse21-abc/](http://cseweb.ucsd.edu/classes/wi16/cse21-abc/)

March 9, 2016
Element Distinctness: WHAT

Given list of positive integers \( a_1, a_2, \ldots, a_n \) decide whether all the numbers are distinct or whether there is a repetition, i.e. two positions \( i, j \) with \( 1 \leq i < j \leq n \) such that \( a_i = a_j \).

*What algorithm would you choose in general?*
Element Distinctness: HOW

Given list of positive integers $a_1, a_2, \ldots, a_n$ decide whether all the numbers are distinct or whether there is a repetition, i.e. two positions $i, j$ with $1 \leq i < j \leq n$ such that $a_i = a_j$.

**What algorithm would you choose in general? Can sorting help?**

Algorithm: first sort list and then step through to find duplicates. What's its runtime?

A. $\Theta(1)$
B. $\Theta(n)$
C. $\Theta(n \log n)$
D. $\Theta(n^2)$
E. None of the above
Element Distinctness: HOW

Given list of positive integers $a_1, a_2, \ldots, a_n$ decide whether all the numbers are distinct or whether there is a repetition, i.e. two positions $i, j$ with $1 \leq i < j \leq n$ such that $a_i = a_j$.

What algorithm would you choose in general? Can sorting help?

Algorithm: first sort list and then step through to find duplicates. How much memory does it require?

A. $\Theta(1)$
B. $\Theta(n)$
C. $\Theta(n \log n)$
D. $\Theta(n^2)$
E. None of the above
Element Distinctness: HOW

Given list of positive integers $a_1, a_2, \ldots, a_n$ decide whether all the numbers are distinct or whether there is a repetition, i.e. two positions $i, j$ with $1 \leq i < j \leq n$ such that $a_i = a_j$.

What algorithm would you choose in general? What if we had unlimited memory?
Given list of positive integers $A = a_1, a_2, \ldots, a_n$,

**UnlimitedMemoryDistinctness**($A$)
1. For $i = 1$ to $n$,
2. If $M[a_i] = 1$ then return "Found repeat"
3. Else $M[a_i] := 1$
4. Return "Distinct elements"

What's the runtime of this algorithm?
A. $\Theta(1)$  
B. $\Theta(n)$  
C. $\Theta(n \log n)$  
D. $\Theta(n^2)$  
E. None of the above
Element Distinctness: HOW

Given list of positive integers $A = a_1, a_2, \ldots, a_n$,

$$\text{UnlimitedMemoryDistinctness}(A)$$
1. For $i = 1$ to $n$,
2. If $M[a_i] = 1$ then return "Found repeat"
3. Else $M[a_i] := 1$
4. Return "Distinct elements"

What's the runtime of this algorithm?
A. $\Theta(1)$
B. $\Theta(n)$
C. $\Theta(n \log n)$
D. $\Theta(n^2)$
E. None of the above

What's the memory use of this algorithm?
A. $\Theta(1)$
B. $\Theta(n)$
C. $\Theta(n \log n)$
D. $\Theta(n^2)$
E. None of the above
Element Distinctness: HOW

To simulate having more memory locations: use Virtual Memory.

Define hash function

\[ h: \{ \text{desired memory locations} \} \rightarrow \{ \text{actual memory locations} \} \]

- Typically we want more memory than we have, so \( h \) is not one-to-one.
- How to implement \( h \)?
  - CSE 12, CSE 100.
- Here, let's use hash functions in an algorithm for Element Distinctness.
Element Distinctness: HOW

Given list of positive integers \( A = a_1, a_2, \ldots, a_n \), and \( m \) memory locations available

\[ \text{HashDistinctness}(A, m) \]

1. Initialize array \( M[1,\ldots,m] \) to all 0s.
2. Pick a hash function \( h \) from all positive integers to \( 1,\ldots,m \).
3. For \( i = 1 \) to \( n \),
4. If \( M[ h(a_i) ] = 1 \) then return "Found repeat"
5. Else \( M[ h(a_i) ] := 1 \)
6. Return "Distinct elements"
Given list of positive integers \( A = a_1, a_2, \ldots, a_n \), and \( m \) memory locations available

\( \text{HashDistinctness}(A, m) \)
1. Initialize array \( M[1,..,m] \) to all 0s.
2. Pick a hash function \( h \) from all positive integers to 1,..,m.
3. For \( i = 1 \) to \( n \),
4. If \( M[ h(a_i) ] = 1 \) then return "Found repeat"
5. Else \( M[ h(a_i) ] := 1 \)
6. Return "Distinct elements"

What's the runtime of this algorithm?
A. \( \Theta(1) \)
B. \( \Theta(n) \)
C. \( \Theta(n \log n) \)
D. \( \Theta(n^2) \)
E. None of the above
Given list of positive integers $A = a_1, a_2, \ldots, a_n$, and $m$ memory locations available

**HashDistinctness**(A, m)
1. Initialize array $M[1,\ldots,m]$ to all 0s.
2. Pick a hash function $h$ from all positive integers to 1,\ldots,m.
3. For $i = 1$ to $n$,
4. If $M[h(a_i)] = 1$ then return "Found repeat"
5. Else $M[h(a_i)] := 1$
6. Return "Distinct elements"

What's the memory use of this algorithm?
A. $\Theta(1)$
B. $\Theta(n)$
C. $\Theta(n \log n)$
D. $\Theta(n^2)$
E. None of the above
Element Distinctness: WHY

Given list of positive integers $A = a_1, a_2, \ldots, a_n$, and $m$ memory locations available

\textbf{HashDistinctness}(A, m)
1. Initialize array $M[1, \ldots, m]$ to all 0s.
2. Pick a hash function $h$ from all positive integers to $1, \ldots, m$.
3. For $i = 1$ to $n$,
4. \hspace{1em} If $M[h(a_i)] = 1$ then return "Found repeat"
5. \hspace{1em} Else $M[h(a_i)] := 1$
6. \hspace{1em} Return "Distinct elements"

\textit{But this algorithm might make a mistake!!! When?}
Element Distinctness: WHY

Given list of positive integers $A = a_1, a_2, \ldots, a_n$, and $m$ memory locations available

HashDistinctness($A$, $m$)
1. Initialize array $M[1,..,m]$ to all 0s.
2. Pick a hash function $h$ from all positive integers to 1,..,$m$.
3. For $i = 1$ to $n$,
4. If $M[ h(a_i) ] = 1$ then return "Found repeat"
5. Else $M[ h(a_i) ] := 1$
6. Return "Distinct elements"

Correctness: Goal is
If there is a repetition, algorithm finds it
If there is no repetition, algorithm reports "Distinct elements"
Element Distinctness: WHY

Given list of positive integers \( A = a_1, a_2, \ldots, a_n \), and \( m \) memory locations available

\[\text{HashDistinctness}(A, m)\]

1. Initialize array \( M[1,\ldots,m] \) to all 0s.
2. Pick a hash function \( h \) from all positive integers to \( 1,\ldots,m \).
3. For \( i = 1 \) to \( n \),
4. If \( M[ h(a_i) ] = 1 \) then return "Found repeat"
5. Else \( M[ h(a_i) ] := 1 \)
6. Return "Distinct elements"

Correctness: Goal is
If there is a repetition, algorithm finds it ✓
If there is no repetition, algorithm reports "Distinct elements" ✗ Hash Collisions
Resolving collisions with chaining

Hash Table

Each memory location holds a pointer to a linked list, initially empty.

Each linked list records the items that map to that memory location.

Collision means there is more than one item in this linked list.
Given list of positive integers $A = a_1, a_2, \ldots, a_n$, and $m$ memory locations available

$\text{ChainHashDistinctness}(A, m)$
1. Initialize array $M[1,\ldots,m]$ to null lists.
2. Pick a hash function $h$ from all positive integers to $1,\ldots,m$.
3. For $i = 1$ to $n$,
4. For each element $j$ in $M[ h(a_i) ]$,
5. If $a_j = a_i$ then return "Found repeat"
6. Append $a_i$ to the tail of the list $M[ h(a_i) ]$
7. Return "Distinct elements"
Element Distinctness: WHY

Given list of positive integers $A = a_1, a_2, \ldots, a_n$, and $m$ memory locations available

$\text{ChainHashDistinctness}(A, m)$
1. Initialize array $M[1,\ldots,m]$ to null lists.
2. Pick a hash function $h$ from all positive integers to $1,\ldots,m$.
3. For $i = 1$ to $n$,
4. For each element $j$ in $M[ h(a_i) ]$,
5. If $a_j = a_i$ then return "Found repeat"
6. Append $a_i$ to the tail of the list $M[ h(a_i) ]$
7. Return "Distinct elements"

Correctness: Goal is
If there is a repetition, algorithm finds it
If there is no repetition, algorithm reports "Distinct elements"
Element Distinctness: MEMORY

Given list of positive integers \( A = a_1, a_2, \ldots, a_n \), and \( m \) memory locations available

\[
\text{ChainHashDistinctness}(A, m)
\]
1. Initialize array \( M[1,\ldots,m] \) to null lists.
2. Pick a hash function \( h \) from all positive integers to \( 1,\ldots,m \).
3. For \( i = 1 \) to \( n \),
4.   For each element \( j \) in \( M[ h(a_i) ] \),
5.     If \( a_j = a_i \) then return "Found repeat"
6.   Append \( a_i \) to the tail of the list \( M[ h(a_i) ] \)
7. Return "Distinct elements"

What's the memory use of this algorithm?
Element Distinctness: MEMORY

Given list of distinct integers $A = a_1, a_2, \ldots, a_n$, and $m$ memory locations available

\textbf{ChainHashDistinctness}(A, m)
1. Initialize array $M[1,\ldots,m]$ to null lists.
2. Pick a hash function $h$ from all positive integers to $1,\ldots,m$.
3. For $i = 1$ to $n$,
4. \quad For each element $j$ in $M[ h(a_i) ]$,
5. \quad \quad If $a_j = a_i$ then return "Found repeat"
6. \quad Append $a_i$ to the tail of the list $M[ h(a_i) ]$
7. Return "Distinct elements"

\textbf{What's the memory use of this algorithm?}
Size of $M$: $O(m)$. Total size of all the linked lists: $O(n)$. Total memory: $O(m+n)$. 
Element Distinctness: WHEN

ChainHashDistinctness(A, m)
1. Initialize array M[1,..,m] to null lists.
2. Pick a hash function \( h \) from all positive integers to 1,..,m.
3. For \( i = 1 \) to \( n \),
4. For each element \( j \) in M[ \( h(a_i) \) ],
5. If \( a_j = a_i \) then return "Found repeat"
6. Append \( a_i \) to the tail of the list M [ \( h(a_i) \) ]
7. Return "Distinct elements" \( \Theta(1) \)
Element Distinctness: WHEN

ChainHashDistinctness(A, m)
1. Initialize array M[1,..,m] to null lists.
2. Pick a hash function $h$ from all positive integers to 1,..,m.
3. For $i = 1$ to $n$,
4. For each element $j$ in M[ $h(a_i)$ ],
5. If $a_j = a_i$ then return "Found repeat"
6. Append $a_i$ to the tail of the list M [ $h(a_i)$ ]
7. Return "Distinct elements"

Worst case is when we don't find $a_i$: $O( 1 + \text{size of list } M[ h(a_i) ] )$
ChainHashDistinctness(A, m)
1. Initialize array M[1,..,m] to null lists.
2. Pick a hash function \( h \) from all positive integers to 1,..,m.
3. For \( i = 1 \) to \( n \),
4. For each element \( j \) in M[ \( h(a_i) \) ],
5. If \( a_j = a_i \) then return "Found repeat"
6. Append \( a_i \) to the tail of the list M [ \( h(a_i) \) ]
7. Return "Distinct elements"

Worst case is when we don't find \( a_i \):
\[
O(1 + \text{size of list } M[ h(a_i) ] )
= O(1 + \# j<i \text{ with } h(a_j)=h(a_i) )
\]
ChainHashDistinctness(A, m)
1. Initialize array M[1,..,m] to null lists.
2. Pick a hash function \( h \) from all positive integers to 1,..,m.
3. For \( i = 1 \) to \( n \),
4. For each element \( j \) in \( M[ h(a_i) ] \),
5. If \( a_j = a_i \) then return "Found repeat"
6. Append \( a_i \) to the tail of the list \( M[ h(a_i) ] \)
7. Return "Distinct elements"

Total time: \( O(n + \sum_{i=1}^{n} \text{ # collisions between pairs } a_i \text{ and } a_j, \text{ where } j<i ) \)

= \( O(n + \text{ total # collisions}) \)

Worst case is when we don't find \( a_i \): \( O( 1 + \text{ size of list } M[ h(a_i) ] ) \)
= \( O( 1 + \text{ # } j<i \text{ with } h(a_j)=h(a_i) ) \)
Collisions depend on choice of hash function

\[ h: \{ \text{desired memory locations} \} \rightarrow \{ \text{actual memory locations} \} \]

**Ideal hash function model**: each output in \{1, 2, …, m\} is equally likely.

So \( h \) is a function that chooses a random number in \{1, 2, …, m\} for each input \( a_i \).
Element Distinctness: WHEN

**Total time:** $O(n + \sum_{i=1}^{n} \# \text{collisions between pairs } a_i \text{ and } a_j, \text{ where } j<i )$

$= O(n + \text{total # collisions})$

*What's the expected total number of collisions?*
Element Distinctness: WHEN

**Total time:** $O(n + \sum_{i=1}^{n} \text{# collisions between pairs } a_i \text{ and } a_j, \text{ where } j<i )$

$=$ $O(n + \text{total # collisions})$

*What's the expected total number of collisions?*

For each pair $(i,j)$ with $j<i$, define:

$$X_{i,j} = 1 \text{ if } h(a_i)=h(a_j) \text{ and } X_{i,j}=0 \text{ otherwise.}$$

**Total # of collisions =** $\sum_{(i,j): j<i} X_{i,j}$
Element Distinctness: WHEN

Total time: $O(n + \sum_{i=1}^{n} \# \text{ collisions between pairs } a_i \text{ and } a_j, \text{ where } j<i)$

$= O(n + \text{total # collisions})$

What's the expected total number of collisions?

For each pair (i,j) with j<i, define:

$X_{i,j} = 1$ if $h(a_i)=h(a_j)$ and $X_{i,j}=0$ otherwise.

Total # of collisions $= \sum_{(i,j): j<i} X_{i,j}$

So by linearity of expectation: $E(\text{total # of collisions}) = \sum_{(i,j): j<i} E(X_{i,j})$
Element Distinctness: WHEN

**Total time:** $O(n + \sum_{i=1}^{n} \# \text{collisions between pairs } a_i \text{ and } a_j, \text{ where } j<i )$

$= O(n + \text{total # collisions})$

**What's the expected total number of collisions?**

For each pair (i,j) with j<i, define:

$X_{i,j} = 1 \text{ if } h(a_i)=h(a_j) \text{ and } X_{i,j}=0 \text{ otherwise.}$

Total # of collisions $= \sum_{(i,j): j<i} X_{i,j}$

What's $E(X_{i,j})$?

A. $1/n$
B. $1/m$
C. $1/n^2$
D. $1/m^2$
E. None of the above.
Element Distinctness: WHEN

**Total time:**\( O(n + \sum_{i=1}^{n} \# \text{collisions between pairs } a_i \text{ and } a_j, \text{ where } j<i) \)

\[ = O(n + \text{total # collisions}) \]

*What's the expected total number of collisions?*

For each pair \((i,j)\) with \(j<i\), define:

\[ X_{i,j} = 1 \text{ if } h(a_i) = h(a_j) \text{ and } X_{i,j} = 0 \text{ otherwise.} \]

**Total # of collisions =**\[ \sum_{(i,j): j<i} X_{i,j} \]

How many terms are in the sum? That is, how many pairs \((i,j)\) with \(j<i\) are there?

A. \(n\)
B. \(n^2\)
C. \(\text{C}(n,2)\)
D. \(n(n-1)\)
Element Distinctness: WHEN

Total time: \( O(n + \sum_{i=1}^{n} \# \text{collisions between pairs } a_i \text{ and } a_j, \text{ where } j<i) \)

\[ = O(n + \text{total \# collisions}) \]

What's the expected total number of collisions?

For each pair \((i,j)\) with \(j<i\), define: \(X_{i,j} = 1\) if \(h(a_i)=h(a_j)\) and \(X_{i,j}=0\) otherwise.

So by linearity of expectation:

\[ E(\text{total \# of collisions}) = \sum_{(i,j):j<i} E(X_{i,j}) = \binom{n}{2} \frac{1}{m} = O(n^2/m) \]
Element Distinctness: WHEN

**Total time:** $O(n + \sum_{i=1}^{n} \# \text{ collisions between pairs } a_i \text{ and } a_j, \text{ where } j<i)$

$$= O(n + \text{total \# collisions})$$

**Total expected time:** $O(n + n^2/m)$

In ideal hash model, as long as $m>n$ the total expected time is $O(n)$. 
Reminders

HW 8 due **Wednesday** at 11:59pm via Gradescope.

**Final exam:**

<table>
<thead>
<tr>
<th>Code</th>
<th>Date</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>A00</td>
<td>Wed, March 16</td>
<td>8:00am - 11:00am</td>
</tr>
<tr>
<td>B00</td>
<td>Mon, March 14</td>
<td>3:00pm - 6:00pm</td>
</tr>
<tr>
<td>C00</td>
<td>Mon, March 14</td>
<td>11:30am - 2:30pm</td>
</tr>
</tbody>
</table>

See website for practice final, review session details, seating charts. Review sessions are Thursday evening and Saturday at noon.