## Probability: Randomized algorithms, Hash functions

<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>Tu, Th 11am-12:20pm</td>
<td>Center 119</td>
</tr>
<tr>
<td>Lecture B</td>
<td>Tiefenbruck</td>
<td>Tu, Th 9:30am-10:50am</td>
<td>Center 119</td>
</tr>
<tr>
<td>Lecture C</td>
<td>Minnes</td>
<td>Tu, Th 3:30pm-4:50pm</td>
<td>WLH 2005</td>
</tr>
</tbody>
</table>

[http://cseweb.ucsd.edu/classes/fa15/cse21-abc/](http://cseweb.ucsd.edu/classes/fa15/cse21-abc/)

Dec 1, 2015
Review

Sample space  
Outcomes  
Events

Probability distribution
  * Uniform  
  * Binomial  
  * Geometric

Random variable  
Expectation  
Concentration  
Variance

Independence  
Linearity of Expectation

Conditional Probability  
Conditional Expectation  
Bayes' Rule

Rosen Chapter 7
Review

Sample space  Outcomes  Events

Probability distribution
* Uniform  * Binomial  * Geometric

Random variable  Expectation  Concentration  Variance

Independence  Linearity of Expectation

Conditional Probability  Conditional Expectation  Bayes' Rule

Today: Applications in algorithms

Rosen Chapter 7
Selection Problem: WHAT

Given list of distinct integers $a_1, a_2, \ldots, a_n$ and integer $i$, $1 \leq i \leq n$,

find the $i^{\text{th}}$ smallest element in the array.
Selection Problem: HOW

Given list of distinct integers $a_1, a_2, \ldots, a_n$ and integer $i$, $1 \leq i \leq n$,

find the $i^{th}$ smallest element in the array.

What algorithm would you choose if $i=1$?
Selection Problem: HOW

Given list of distinct integers $a_1, a_2, \ldots, a_n$ and integer $i$, $1 \leq i \leq n$,
find the $i^{th}$ smallest element in the array.

What algorithm would you choose in general?
Selection Problem: HOW

Given list of distinct integers \( a_1, a_2, \ldots, a_n \) and integer \( i, 1 \leq i \leq n \),
find the \( i^{th} \) smallest element in the array.

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

Algorithm: first sort list and then step through to find \( i^{th} \) smallest. What's its runtime?

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

Given list of distinct integers $a_1, a_2, \ldots, a_n$ and integer $i$, $1 \leq i \leq n$,
find the $i^{th}$ smallest element in the array.

*What algorithm would you choose in general? Different strategy …*

Pick random list element called “pivot.”

Partition list into those smaller than pivot, those bigger than pivot.

Using $i$ and size of partition sets, determine in which set to continue looking.
Given list of distinct integers $a_1, a_2, \ldots, a_n$ and integer $i$, $1 \leq i \leq n$, find the $i^{th}$ smallest element in the array.

Pick random list element called “pivot.”
Partition list into those smaller than pivot, those bigger than pivot.
Using $i$ and size of partition sets, determine in which set to continue looking.

ex. \begin{align*} 17, 42, 3, 8, 19, 21, 2 & \quad \text{i = 3} \end{align*}
Selection Problem: HOW

Given list of distinct integers \( a_1, a_2, \ldots, a_n \) and integer \( i, 1 \leq i \leq n \), find the \( i^{th} \) smallest element in the array.

Pick random list element called “pivot.”
Partition list into those smaller than pivot, those bigger than pivot.
Using \( i \) and size of partition sets, determine in which set to continue looking.

ex. \( 17, 42, 3, 8, 19, 21, 2 \) \hspace{1cm} i = 3 \hspace{1cm} \text{Random pivot: 17}
Given list of distinct integers $a_1, a_2, \ldots, a_n$ and integer $i$, $1 \leq i \leq n$, find the $i^{th}$ smallest element in the array.

Pick random list element called “pivot.”
Partition list into those smaller than pivot, those bigger than pivot.
Using $i$ and size of partition sets, determine in which set to continue looking.

ex. 17, 42, 3, 8, 19, 21, 2  \hspace{1cm} i = 3  \hspace{1cm} \text{Random pivot: 17}

Smaller than 17: 3, 8, 2  \hspace{1cm} \text{Bigger than 17: 42, 19, 21}
Selection Problem: HOW

Given list of distinct integers $a_1, a_2, \ldots, a_n$ and integer $i$, $1 \leq i \leq n$, find the $i^{th}$ smallest element in the array.

Pick random list element called “pivot.”
Partition list into those smaller than pivot, those bigger than pivot.
Using $i$ and size of partition sets, determine in which set to continue looking.

ex. 17, 42, 3, 8, 19, 21, 2  \hspace{1cm} i = 3  \hspace{1cm} \text{Random pivot: 17}

Smaller than 17: 3, 8, 2
Bigger than 17: 42, 19, 21

Has 3 elements so third smallest must be in this set
Selection Problem: HOW

Given list of distinct integers $a_1, a_2, \ldots, a_n$ and integer $i$, $1 \leq i \leq n$, find the $i$th smallest element in the array.

Pick random list element called “pivot.”
Partition list into those smaller than pivot, those bigger than pivot.
Using $i$ and size of partition sets, determine in which set to continue looking.

ex. $17, 42, 3, 8, 19, 21, 2$ $i = 3$ Random pivot: 17
New list: 3, 8, 2 $i = 3$
Selection Problem: HOW

Given list of distinct integers $a_1, a_2, ..., a_n$ and integer $i$, $1 \leq i \leq n$, find the $i^{th}$ smallest element in the array.

Pick random list element called “pivot.”
Partition list into those smaller than pivot, those bigger than pivot.
Using $i$ and size of partition sets, determine in which set to continue looking.

ex. $17, 42, 3, 8, 19, 21, 2$ $i = 3$ Random pivot: 17
New list: $3, 8, 2$ $i = 3$ Random pivot: 8
Selection Problem: HOW

Given list of distinct integers $a_1, a_2, \ldots, a_n$ and integer $i$, $1 \leq i \leq n$, find the $i^{th}$ smallest element in the array.

Pick random list element called “pivot.”
Partition list into those smaller than pivot, those bigger than pivot.
Using $i$ and size of partition sets, determine in which set to continue looking.

ex. \underline{17, 42, 3, 8, 19, 21, 2} \quad i = 3 \quad \text{Random pivot: 17} \\
New list: 3, 8, 2 \quad i = 3 \quad \text{Random pivot: 8}

Smaller than 8: 3, 2 \quad Bigger than 8:
Selection Problem: HOW

Given list of distinct integers $a_1, a_2, \ldots, a_n$ and integer $i$, $1 \leq i \leq n$, find the $i^{th}$ smallest element in the array.

Pick random list element called “pivot.”
Partition list into those smaller than pivot, those bigger than pivot.
Using $i$ and size of partition sets, determine in which set to continue looking.

ex. $17, 42, 3, 8, 19, 21, 2$ \hspace{1cm} $i = 3$ \hspace{1cm} Random pivot: 17
New list: 3, 8, 2 \hspace{1cm} $i = 3$ \hspace{1cm} Random pivot: 8

Smaller than 8: 3, 2 \hspace{1cm} Bigger than 8:

Has 2 elements so third smallest must be "next" element, i.e. 8
Selection Problem: HOW

Given list of distinct integers \(a_1, a_2, \ldots, a_n\) and integer \(i, 1 \leq i \leq n\), find the \(i^{th}\) smallest element in the array.

Pick random list element called “pivot.”
Partition list into those smaller than pivot, those bigger than pivot.
Using \(i\) and size of partition sets, determine in which set to continue looking.

ex. \[17, 42, 3, 8, 19, 21, 2\] \(i = 3\) \(\text{Random pivot: 17}\)
New list: \(3, 8, 2\) \(i = 3\) \(\text{Random pivot: 8}\)
Smaller than 8: \(3, 2\) Bigger than 8:

Return 8 compare to original list: \(17, 42, 3, 8, 19, 21, 2\)
Selection Problem: HOW

Given list of distinct integers $A = a_1, a_2, \ldots, a_n$ and integer $i$, $1 \leq i \leq n$,

Algorithm will incorporate both randomness and recursion!
Selection Problem: HOW

Given list of distinct integers $A = a_1, a_2, \ldots, a_n$ and integer $i$, $1 \leq i \leq n$, RandSelect($A,i$)

1. If $n=1$ return $a_1$

What are we doing in this first line?

A. Establishing the base case of the recursion.
B. Establishing the induction step.
C. Randomly picking a pivot.
D. Randomly returning a list element.
E. None of the above.
Selection Problem: HOW

Given list of distinct integers $A = a_1, a_2, \ldots, a_n$ and integer $i$, $1 \leq i \leq n$, 
$\text{RandSelect}(A, i)$

1. If $n=1$ return $a_1$
2. Initialize lists $S$ and $B$.
3. Pick integer $j$ uniformly at random from 1 to $n$.
4. For each index $k$ from 1 to $n$ (except $j$):
   5. if $a_k < a_j$, add $a_k$ to the list $S$.
   6. if $a_k > a_j$, add $a_k$ to the list $B$. 
Selection Problem: HOW

Given list of distinct integers $A = a_1, a_2, \ldots, a_n$ and integer $i$, $1 \leq i \leq n$, $\text{RandSelect}(A,i)$

1. If $n=1$ return $a_1$
2. Initialize lists $S$ and $B$.
3. Pick integer $j$ uniformly at random from 1 to $n$.
4. For each index $k$ from 1 to $n$ (except $j$):
   5. if $a_k < a_j$, add $a_k$ to the list $S$.
   6. if $a_k > a_j$, add $a_k$ to the list $B$.
7. Let $s$ be the size of $S$.
8. If $s = i-1$, return $a_j$. 
Selection Problem: HOW

Given list of distinct integers \( A = a_1, a_2, \ldots, a_n \) and integer \( i, 1 \leq i \leq n \),

\[ \text{RandSelect}(A,i) \]

1. If \( n=1 \) return \( a_1 \)
2. Initialize lists \( S \) and \( B \).
3. Pick integer \( j \) uniformly at random from 1 to \( n \).
4. For each index \( k \) from 1 to \( n \) (except \( j \)):
   5. if \( a_k < a_j \), add \( a_k \) to the list \( S \).
   6. if \( a_k > a_j \), add \( a_k \) to the list \( B \).
7. Let \( s \) be the size of \( S \).
8. If \( s = i-1 \), return \( a_j \).
9. If \( s \geq i \), return \( \text{RandSelect}(S, i) \).
10. If \( s < i \), return \( \text{RandSelect}(B, \_??\_?) \).

What's the right way to fill in this blank?
A. \( i \)
B. \( s \)
C. \( i+s \)
D. \( i-(s+1) \)
E. None of the above.
Selection Problem: WHEN

Given list of distinct integers $A = a_1, a_2, \ldots, a_n$ and integer $i$, $1 \leq i \leq n$, RandSelect($A$, $i$)

1. If $n=1$ return $a_1$
2. Initialize lists $S$ and $B$.
3. Pick integer $j$ uniformly at random from 1 to $n$.
4. For each index $k$ from 1 to $n$ (except $j$):
   5. if $a_k < a_j$, add $a_k$ to the list $S$.
   6. if $a_k > a_j$, add $a_k$ to the list $B$.
7. Let $s$ be the size of $S$.
8. If $s = i-1$, return $a_j$.
9. If $s \geq i$, return RandSelect($S$, $i$).
10. If $s < i$, return RandSelect($B$, $i-(s+1)$).

What input gives the best-case performance of this algorithm?
A. When element we're looking for is the first in list.
B. When element we're looking for is $i^{th}$ in list.
C. When element we're looking for is in the middle of the list.
D. When element we're looking for is last in list.
E. None of the above.
Selection Problem: WHEN

Given list of distinct integers $A = a_1, a_2, \ldots, a_n$ and integer $i$, $1 \leq i \leq n$,

$\text{RandSelect}(A, i)$

1. If $n=1$ return $a_1$
2. Initialize lists $S$ and $B$.
3. Pick integer $j$ uniformly at random from $1$ to $n$.
4. For each index $k$ from $1$ to $n$ (except $j$):
   5. if $a_k < a_j$, add $a_k$ to the list $S$.
   6. if $a_k > a_j$, add $a_k$ to the list $B$.
7. Let $s$ be the size of $S$.
8. If $s = i-1$, return $a_j$.
9. If $s \geq i$, return $\text{RandSelect}(S, i)$.
10. If $s < i$, return $\text{RandSelect}(B, i-(s+1))$.

Performance depends on more than the input!
Selection Problem: WHEN

Given list of distinct integers \( A = a_1, a_2, \ldots, a_n \) and integer \( i, 1 \leq i \leq n \),

\[ \text{RandSelect}(A, i) \]

1. If \( n=1 \) return \( a_1 \)
2. Initialize lists \( S \) and \( B \).
3. **Pick integer \( j \) uniformly at random from 1 to \( n \).**
4. For each index \( k \) from 1 to \( n \) (except \( j \)):
5. \hspace{1cm} if \( a_k < a_j \), add \( a_k \) to the list \( S \).
6. \hspace{1cm} if \( a_k > a_j \), add \( a_k \) to the list \( B \).
7. Let \( s \) be the size of \( S \).
8. **If \( s = i-1 \), return \( a_j \).**
9. If \( s \geq i \), return \( \text{RandSelect}(S, i) \).
10. If \( s < i \), return \( \text{RandSelect}(B, i-(s+1)) \).

Minimum time if we happen to pick pivot which is the \( i^{th} \) smallest list element.

In this case, what's the runtime?

A. \( \Theta(1) \)
B. \( \Theta(n) \)
C. \( \Theta(n \log n) \)
D. \( \Theta(n^2) \)
E. None of the above
Selection Problem: WHEN

How can we give a time analysis for an algorithm that is allowed to pick and then use random numbers?

$T(x)$: a random variable that represents the runtime of the algorithm on input $x$

Compute the **worst-case expected time**

$$ET(n) = \max_{x, |x| \leq n} E(T(x))$$

- worst case over all inputs of size $n$
- average runtime incorporating random choices in the algorithm
How can we give a time analysis for an algorithm that is allowed to pick and then use random numbers?

$T(x)$: a random variable that represents the runtime of the algorithm on input $x$

Compute the **worst-case expected time**

$$ET(n) = \max_{x, |x| \leq n} E(T(x))$$

Recurrence equation … unravelling …

$\Theta(n)$
Selection Problem: WHEN

**Situation so far:**

Sort then search takes worst-case $\Theta(n \log n)$

Randomized selection takes worst-case expected time $\Theta(n)$
Selection Problem: WHEN

**Situation so far:**

Sort then search takes worst-case $\Theta(n \log n)$

Randomized selection takes worst-case expected time $\Theta(n)$

*How do we implement randomized algorithms?*

*Are there deterministic algorithms that perform as well?*

For selection problem: Blum et al, yes!

In general: open 😊
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 $a_i = a_j$. 
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 \textit{repetition}, i.e. two positions $i, j$ with $1 \leq i < j \leq n$ such that $a_i = a_j$.

\textit{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
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?
Element Distinctness: HOW

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
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"

$M$ is an array of memory locations
This is memory location indexed by $a_i$

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
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 distinct 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"
Element Distinctness: HOW

Given list of distinct 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,..,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
Element Distinctness: HOW

Given list of distinct 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"

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 distinct 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"

But this algorithm might make a mistake!!!
When?
Given list of distinct 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"

**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 distinct 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. \quad If $M[h(a_i)] = 1$ then return "Found repeat"
5. \quad Else $M[h(a_i)] := 1$
6. Return "Distinct elements"

\textbf{Correctness: Goal is}
If there is a repetition, algorithm finds it
If there is no repetition, algorithm reports "Distinct elements"

\textbf{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.
Element Distinctness: HOW

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

$\text{ChainHashDistinctness}(A, m)$
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 $i$ to the tail of the list $M[ h(a_i) ]$
7. Return "Distinct elements"
Element Distinctness: WHY

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

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

**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 $i$ to the tail of the list $M [ h(a_i) ]$
7. Return "Distinct elements"

What's the memory use of this algorithm?
Given list of distinct integers \( A = a_1, a_2, \ldots, a_n \), and \( m \) memory locations available

\[ \text{ChainHashDistinctness}(A, m) \]
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 \( i \) to the tail of the list \( M[ h(a_i) ] \)
7. Return "Distinct elements"

**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. \[\Theta(1)\]
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 \(i\) 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 \( M[ h(a_i) ] \),
   If \( a_j = a_i \) then return "Found repeat"
5. Append \( i \) to the tail of the list \( M[ h(a_i) ] \)
6. 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 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 + size of list M[ h(a_i) ] )
= O( 1 + # j<i with h(a_j)=h(a_i) )
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 \( i \) to the tail of the list \( M[ h(a_i) ] \)
7. Return "Distinct elements"

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

\[ = O(n + \text{total } \# \text{collisions}) \]
Element Distinctness: WHEN

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

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

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

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

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

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

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

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

**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:

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

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:

\( X_{i,j} = 1 \) if \( h(a_i = a_j) \) and \( X_{i,j} = 0 \) 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:

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

**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. $\binom{n}{2}$
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 $h(a_i = a_j)$ 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

**Final exam:** Saturday, December 5, 11:30-2:30. That’s 4 days from today.

- Lecture B00 (9:30am, Tiefenbruck) - Final exam in Peterson 110.
- Lecture A00 (11am, Tiefenbruck) - Final exam in Peterson 108.
- Lecture C00 (3:30pm, Minnes) - Final exam in Warren Lecture Hall 2001.
- See seating charts on course website.

**Review sessions:**
- Tuesday, Dec 1, 8pm-10pm in Solis 107
- Wednesday, Dec 2, 8pm-10pm in York 2722