### Lecture A
- **Tiefenbruck**
- **MWF 9-9:50am**
- **Center 212**

### Lecture B
- **Jones**
- **MWF 2-2:50pm**
- **Center 214**

### Lecture C
- **Tiefenbruck**
- **MWF 11-11:50am**
- **Center 212**

http://cseweb.ucsd.edu/classes/wi16/cse21-abc/

March 7, 2016
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.
Selection Problem: HOW

Given list of distinct integers $a_1, a_2, \ldots, a_n$ and integer $i$, $1 <= i <= n$,
find the $i^{\text{th}}$ 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.
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. \hspace{1cm} 17, 42, 3, 8, 19, 21, 2 \hspace{1cm} i = 3
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
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

Smaller than 17: 3, 8, 2
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 $i = 3$ 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. \[17, 42, 3, 8, 19, 21, 2\] \hspace{1cm} i = 3 \hspace{1cm} \text{Random pivot: 17}

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

New list: 3, 8, 2 $i = 3$ Random pivot: 8

Smaller than 8: 3, 2

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

New list: 3, 8, 2 \hspace{1cm} i = 3 \hspace{1cm} \text{Random pivot: 8}
Smaller than 8: 3, 2 \hspace{1cm} \text{Bigger than 8:}

Return 8 \hspace{1cm} \text{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 $i$, $1 \leq i \leq n$,
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 \( a_1 \)

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 $B$. 
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.
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 $\text{RandSelect}(B, i-(s+1))$.

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. 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 $\text{RandSelect}(B, i-(s+1))$.

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 😊
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?*
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
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

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 \textit{Virtual Memory}.

Define \textbf{hash function} \begin{align*}
    h: \{ \text{desired memory locations} \} & \to \{ \text{actual memory locations} \} \\
\end{align*}

- Typically we want more memory than we have, so $h$ is \textbf{not one-to-one}.
- How to implement $h$?
  - CSE 12, CSE 100.
- Here, let's use hash functions in an algorithm for Element Distinctness.
Virtual Memory Applications

For example, suppose you have a company of 5,000 employees and each is identified by their SSN. You want to be able to access employee records by their SSN.

You don’t want to keep a table of all possible SSN’s so we’ll use a virtual memory data structure to emulate having that huge table.

Can you think of any other examples?
Ideally, we could use a very unpredictable function called a hash function to assign random physical locations to each virtual location.

Later we will discuss how to actually implement such hash functions. But for now assume that we have a function $h$ so that for every virtual location $v$, $h(v)$ is uniformly and randomly chosen among the physical locations.

We call such an $h$ an ideal hash function if its computable in constant time.
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 $M[1,..,m]$ to all 0s.
2. Pick a hash function $h$ from all positive integers to $1,..,m$.
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. 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
Element Distinctness: HOW

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. 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
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"

But this algorithm might make a mistake!!!
When?
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

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"  Hash Collisions
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 $M[1, \ldots, m]$ 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"

When is our algorithm correct with high probability in the ideal hash model?
Birthday paradox

One example where people are misled is the *birthday paradox*.

**What is the chance of two people in a group sharing the same birthday?**

Example: What is the chance that, in a group of 30 people, two share the same birthday?
Days of the year = memory locations

Where is the connection?

\( h(\text{person}) = \text{birthday} \)

collisions mean that two people share the same birthday.
We have \( n \) objects and \( m \) places. We are putting each object at random into one of the places. What is the probability that 2 objects occupy the same place?
Calculating the general rule
Calculating the general rule

Probability the first object causes no collisions is 1
Calculating the general rule

Probability the second object causes no collisions is $1 - \frac{1}{m}$
Calculating the general rule

Probability the third object causes no collisions is \( \frac{m-2}{m} = 1 - \frac{2}{m} \)
Calculating the general rule

Probability the $i$th object causes no collisions is $1 - (i-1)/m$
Using conditional probabilities, the probability there is no collisions is \[ [1(1-1/m)(1-2/m)\ldots(1-(n-1)/m)] \]

\[ p = \prod_{i=1}^{n} \left(1 - \frac{i-1}{m}\right) \]

Then using the fact that \( 1 - x \leq e^{-x} \),

\[ p \leq \prod_{i=1}^{n} e^{-\frac{i-1}{m}} = e^{-\sum_{i=1}^{n} \frac{i-1}{m}} = e^{-\frac{n}{2m}} \]
Conditional Probabilities

\[ p \leq \prod_{i=1}^{n} e^{-\frac{i-1}{m}} = e^{-\sum_{i=1}^{n} \frac{i-1}{m}} = e^{-\frac{\binom{n}{2}}{m}} \]

We want \( p \) to be close to 1 so \( \frac{\binom{n}{2}}{m} \) should be small, i.e. \( m \gg \binom{n}{2} \approx \frac{n^2}{2} \).

For the birthday problem, this is when the number of people is about \( \sqrt{2(365)} \approx 27 \)

In the element distinctness algorithm, we need the number of memory locations to be at least \( \Omega(n^2) \).
Conditional Probabilities

\[
p \leq \prod_{i=1}^{n} e^{-\frac{i-1}{m}} = e^{-\sum_{i=1}^{n} \frac{i-1}{m}} = e^{-\frac{(n)}{m}}
\]

On the other hand, it is possible to show that if \( m >> n^2 \) then there are no collisions with high probability. i.e.

\[
p > 1 - \frac{(n)}{m}
\]

So if \( m \) is large then \( p \) is close to 1.
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 $M[1,\ldots,m]$ 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 this means about this algorithm is that we can get time to be $O(n)$ at the expense of using $O(n^2)$ memory. Since we need to initialize the memory, this doesn’t seem worthwhile because sorting uses less memory and slightly more time. So what can we do?
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 positive 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 positive 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"
Element Distinctness: MEMORY

Given list of positive 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?
Element Distinctness: MEMORY

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

**ChainHashDistinctness**($A$, $m$)
1. Initialize array $M[1, \ldots, m]$ 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 $i$ 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) ] \( \Theta(1) \)
7. Return "Distinct elements"
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"

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:
\[ O(1 + \text{size of list } M[ h(a_i) ]) = O(1 + \# j<i \text{ with } h(a_j)=h(a_i)) \]
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"

Worst case is when we don't find $a_i$:

\[
\begin{align*}
O(1 + \text{size of list } M[ h(a_i) ] ) = O(1 + \# j<i \text{ with } h(a_j)=h(a_i) )
\end{align*}
\]

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

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

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

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

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

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. $C(n,2)$
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 \( O(n) \).