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?

Linear search $(a_1, \ldots, a_n, j a_i)$ for each $i$ $O(n^2)$

Sort it. And look at neighbors.
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 \( A = a_1, a_2, \ldots, a_n \),

\textbf{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

M is an array of memory locations
This is memory location indexed by \( a_i \)

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.
Virtual Memory Applications

Not just in this algorithm but in many computational settings we want to simulate a huge space of possible memory locations but where we are only going to access a small fraction.

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. (9 digits) $10^9$ memory locations.

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.

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 $M[1,\ldots,m]$ 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 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 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

$\text{HashDistinctness}(A, m)$
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 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

$\Theta(m)$
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"

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

You could have
\[
\begin{align*}
  a_i \neq a_j \\
  h(a_i) = h(a_j)
\end{align*}
\]
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 $M[1,..,m]$ 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
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"

When is our algorithm correct with high probability in the ideal hash model?
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?
Where is the connection?

Think of elements in the array = people

Days of the year = memory locations

h(person)=birthday

collisions mean that two people share the same birthday.
General Birthday Paradox-type Phenomena

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

\[
1 - \frac{0}{m}, \quad 1 - \frac{1}{m}, \quad 1 - \frac{2}{m}, \quad 1 - \frac{3}{m} \quad \cdots
\]

Probability the ith object causes no collisions is \(1-(i-1)/m\)

Given that the previous objects are in distinct locations.
Using conditional probabilities, the probability there is no collisions is \[ [1(1-1/m)(1-2/m)\ldots(1-(n-1)/m)] \]

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

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

\( n = 150 \)
\( m = 365 \)
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. \( m \gg \binom{n}{2} \approx \frac{n^2}{2} \).

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

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

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

\[ p > 1 - \frac{\binom{n}{2}}{m} \]

So if \( m \) is large then \( p \) is close to 1.
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"

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

$$\textbf{ChainHashDistinctness}(A, m)$$
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. \hspace{1cm} For each element $j$ in $M[ h(a_i) ]$,
5. \hspace{1cm} If $a_j = a_i$ then return "Found repeat"
6. \hspace{1cm} 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

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 positive 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 \( a_i \) 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

\textbf{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.  \(\Theta(1)\)
3. For \(i = 1\) to \(n\),
4. \hspace{1cm} For each element \(j\) in M[ \(h(a_i)\) ],
5. \hspace{1cm} If \(a_j = a_i\) then return "Found repeat"
6. \hspace{1cm} 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 \( a_i \):
\[
O( 1 + \text{size of list } M[ h(a_i) ] ) \\
= O( 1 + \# j<i \text{ with } h(a_j)=h(a_i) )
\]
Element Distinctness: WHEN

\textbf{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"

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

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

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

How can I find the total number of collisions?

Doesn’t it depend on the random assignment to memory locations?
Element Distinctness: WHEN

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

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

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

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

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

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

LINEARITY OF EXPECTATION!
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?

$$\text{total number of collisions} = \sum_{(i,j) \ j<i} X_{i,j}$$

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

$X_{i,j} = 0$ otherwise
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:

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

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 = \( \sum_{(i,j):j<i} X_{i,j} \)**

\( \mathbb{E}(X) = \mathbb{P}(X=1) \)

What's \( \mathbb{E}(X_{i,j})? = \mathbb{P}(X_{i,j} = 1) \)

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:

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

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

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

We want $m = O(n)$

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

**Note:** This is much better than our original approach using sorting.

- **Expected runtime is** $O(n)$
- **Memory is** $O(n)$.
Announcements

Final Exam
A00: Wednesday 3/22 7pm
B00: Friday 3/24 11:30am

HW 8
Due Wednesday
11:59pm

OHs, 1-1s
Now is the time!

Two Final Exam Review Sessions
Saturday 3/18 1-3pm
Sunday 3/19 1-3pm
Locations TBD
Final Exam PPs posted
TTK has been started