“Notes” slides from before lecture

CSE 21, Winter 2017, Section A00

Lecture 17 Notes

Class URL: http://vlsicad.ucsd.edu/courses/cse21-w17/
Notes March 13 (1)

- **This week:** Days 25 (24), 26 of posted slides
  - Element distinctness, randomized selection \(\rightarrow\) pulling recursion, randomization, conditional probability, algorithm design all together
  - Review (Wednesday)

- **HW8 is due on Wednesday.**

- **Final exam reviews:**
  - Saturday 1-3pm York 2722
  - Sunday 1-3pm Center 101

- **FINAL EXAM WILL BE IN CENTER 101 (entire class) !!!**
  - Two pieces of paper allowed

- Please fill out CAPEs – thank you

- Any logistic, other issues?
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?

- "Sort" then traverse sorted list to see if adjacent elements are equal
- "Hash"
- bucket/radix sort in $\Theta(n)$ if $n$'s are bounded
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

*no extra memory used if “in-place” sorting algorithm is used.*
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 $A = a_1, a_2, \ldots, a_n$,

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 $A = a_1, a_2, \ldots, a_n$,

**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

*depends on value of max list element*
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.

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.
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 it is 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. $\checkmark$ 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 $n$, $\forall a_i \in A$
4. If $M[ h(a_i) ] = 1$ then return "Found repeat"
5. Else $M[ h(a_i) ] := 1$
6. Return "Distinct elements"

\[ h = \text{"mod 10"} \]
\[ m = 10 \]
\[ \text{input} = 27, 19, 36, 24, 17 \]

$\text{\textcircled{7}}$ $\text{\textcircled{9}}$ $\text{\textcircled{c}}$ $\text{\textcircled{4}}$ $\text{\textcircled{7}}$
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. \( O(m) \)
2. Pick a hash function \( h \) from all positive integers to 1,\ldots,m.
3. For \( i = 1 \) to \( n \),
4. \( \rightarrow \) If \( M[ h(a_i) ] = 1 \) then return "Found repeat"
5. \( \rightarrow \) 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 $M[1,..,m]$ 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

$\Theta(m)$, indep of $n$
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"

But this algorithm might make a mistake!!!

\[
\begin{align*}
53 & \equiv 3 \pmod{10} \\
43 & \equiv 3 \pmod{10}
\end{align*}
\]
but \( 53 \neq 43 \)
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"

Currently, fail this goal

43, ..., 43

$\times \ 43, \ldots, 53$
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?
Where is the connection?

Think of elements in the array = people

Days of the year = memory locations

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

collisions mean that two people share the same birthday.

\( \Pr(\text{collision}) \)
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?

$$Pr = (1 - \frac{2}{m})$$

- prob that $3^{rd}$ object causes no collision given no collision before

$$Pr = 1 - \frac{(i-1)}{m}$$

$$\text{Prob(no collisions)} = ?$$

- $n=1 \Rightarrow P = 1$
- $n=2 \Rightarrow P = (1 - \frac{1}{n})$
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 \(1 - x \leq e^{-x}\),

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

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

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

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

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

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

$$p > 1 - \frac{{n \choose 2}}{m}$$

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

$\text{HashDistinctness}(A, m)$
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.
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 $M[1,\ldots,m]$ 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 $M[h(a_i)]$
3. 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 $a_i$ 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?

$O(m)$ memory locations in $M$

$O(n) = \text{total size of linked lists}$
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 $a_i$ 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

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

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 \( a_i \) 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 + size of list M[ 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 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

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)]
4. Return "Distinct elements"

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

= $O(n + \text{total # collisions})$
Collisions depend on choice of hash function

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

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

So \( h \) is a function that chooses a random number in \( \{1,2,\ldots,m\} \) 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:** $O(n + \sum_{i=1}^{n} \# \text{ collisions between pairs } a_i \text{ and } a_j, \text{ where } j<i )$

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

random variable!
Element Distinctness: WHEN

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

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

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

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

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

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

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

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

$C(n,2) \cdot \frac{1}{m}$
Element Distinctness: WHEN

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

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

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:

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

$X_{i,j} = 1$ 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:

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

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

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

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

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

CSE21 Winter 2017, Day 24 (B00), Day 16 (A00)

March 13, 2017

http://vlsicad.ucsd.edu/courses/cse21-w17
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 $i^{th}$ smallest element in the list.
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 list.

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 list.

What algorithm would you choose in general?

\[ i = \frac{n}{2} \quad ? \quad \Theta(n^2) \times \]
\[ i = 7 \quad ? \]
\[ 7 \cdot n \]
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 list.

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 list.

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

Pick random list element called the “pivot”

Partition list into those Smaller than pivot, those Bigger than pivot

Using \( i \) and size of partition sets \( S \) and \( B \), 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 list.

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} n = 7$
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 list.

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

$S = 2, 3, 8$  $p = 17$  $B = 42, 19, 21$
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 list.

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. $17, 42, 3, 8, 19, 21, 2$\hfill i = 3 \hfill \text{Random pivot: 17}

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

$|S| = 3$ \hfill $p = 17$
look in $S$ with $i = 3$
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 list.

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}

\textbf{Smaller than 17: 3, 8, 2} \hspace{1cm} \textbf{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 list.

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: $3, 8, 2$ $i = 3$

$S = 2, 3$ $p = 8$ $B = |S| = s$ $s + 1 = i \Rightarrow$ DONE!
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 list.

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

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 list.

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 list.

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:

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 $i$, $1 \leq i \leq n$,
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 $a_1$

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 $B$.

"pivot about $p = a_j$ process..."

\[ S \quad a_j \quad B \]

\[ 2 \quad 3 \quad 8 \]

// index of random pivot element

in its proper position!!!
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. \( \text{if } a_k < a_j, \text{ add } a_k \text{ to the list } S. \)
6. \( \text{if } a_k > a_j, \text{ add } a_k \text{ to the list } B. \)
7. \( \text{Let } s \text{ be the size of } S. \)
8. \( \text{If } s = i-1, \text{ return } a_j. \)
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 $\text{RandSelect}(B, i-(s+1))$.

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 $i$, $1 \leq i \leq n$, 
$\text{RandSelect}(A, i)$

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 $i$, $1 \leq i \leq n$, 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($B$, $i-(s+1)$).

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 $i^{\text{th}}$ 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**

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

== expected time on worst input of size n

worst case over all inputs of size n

average runtime incorporating random choices in the algorithm
Selection Problem: WHEN

**Situation so far:**

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

Randomized selection takes worst-case expected time $\Theta(n)$
Randomized Selection (1)

We can’t easily find a “magical”, “balanced” pivot $p$, so we choose $p$ **randomly** from the array $A$

Worst case: $\Omega(n^2)$ but highly unlikely

$T(n) = \text{expected time for Select on array of size } n$

Suppose we could guarantee a “good” pivot, e.g., such that $n/4 \leq |S|, |B| \leq 3n/4$

$\Rightarrow$ subproblem size $\leq 3n/4$

Then, $T(n) \leq T(3n/4) + O(n)$

... Does this give $T(n) = O(n)$?
Suppose we can guarantee a “good” pivot s.t. $n/4 \leq |S|, |B| \leq 3n/4$

$\Rightarrow$ subproblem size $\leq 3n/4$

Let $R(n) \equiv$ expected number of pivot operations before array is reduced to $\leq 3n/4$ elements

$$T(n) \leq T(3n/4) + O(n \times R(n))$$

$R(n) =$ find balanced pivot element
$O(n) =$ perform pivot, make subproblem

$T(3n/4) =$ upper bound on time needed to solve subproblem
**Randomized Selection (3)**

**Suppose** we can guarantee a “good” pivot s.t. \( n/4 \leq |S|, |B| \leq 3n/4 \)

\[ \Rightarrow \text{subproblem size} \leq 3n/4 \]

Let \( R(n) \equiv \text{expected number of split operations before array is reduced to} \leq 3n/4 \text{ elements} \)

\[ \Rightarrow T(n) \leq T(3n/4) + O(n \times R(n)) \]

\( R(n) = \text{find balanced pivot element} \)

\( O(n) = \text{perform pivot, make subproblem} \)

\( T(3n/4) = \text{solve subproblem} \)

**Fact:** \( R(n) \leq 2 \implies T(n) \leq T(3n/4) + O(n) = O(n) \)

(*** Why is \( R(n) \leq 2 \) ?)
Analysis of Randomized Selection Complexity

Suppose: \( T(n) \leq dn + T(3n/4) \)

Claim: \( T(n) \leq kn \) for some \( k \) would follow

Proof strategy: **Constructive Induction** ("Substitution")

**Strong Induction Hypothesis:** \( T(m) \leq km \) for all \( m \leq n-1 \)

Induction Step: \( T(n) \leq dn + k(3n/4) \)
\[ = (d + 3k/4)n \]
which we want to be equivalent to \( T(n) \leq kn \)

But this will be true as long as we pick a value of \( k \geq 4d \)
Announcements

**Final Exam**
- A00: Wednesday 3/22 7pm
  LOCATION = CENTER 101
- 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 York 2722
- Sunday 3/19 1-3pm Center 101

Final Exam PPs posted
TTK has been started