Day 25 – Randomize Algorithms

CSE21 Fall 2018

December 3, 2018

https://sites.google.com/ucsd.edu/cse21fa18/

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?

Sort → $O(n \log n)$

Linear Search → $O(n)$

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?

$\Theta(n.\log(n))$ time

$\Theta(n)$ time

$\Theta(n.\log n)$

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

- 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

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

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, $1 <= i <= n$,

RandSelect(A,i)

1. If $n=1$ return a_1 (Base case)
2. Initialize lists S and B.
3. Pick integer j uniformly at random from 1 to n. → pivot
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 $s >= i$, return RandSelect(S, i).
10. If $s < i$, return RandSelect(B, i-(s+1)).

separate the elements into S, B

compare |S| to i

Algorithm will incorporate both randomness and recursion!

Selection Problem: WHEN

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

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 >= 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 looki is the first in list.
B. When elemen ing for is i^{th} in li
C. W t we're looking for middle of the list.
D. hen element we're looking for is last in list.
E. None of the above.

Performance depends on more than the input!

(Hoare algorithm)

Selection Problem: WHEN

Given list of distinct integers $A = a_1, a_2, \ldots, a_n$ and integer i, 1 <= i <= 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 >= i, return RandSelect(S, i).
10. If s < i, return RandSelect(B, i-(s+1)).

const. time

$\Theta(n)$

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

Recurrence equation … unravelling … $\Theta(n)$

Selection Problem: WHEN

Recurrence equation … unravelling … $\Theta(n)$ - how?

$$n + \frac{n}{2} + \frac{n}{4} + \frac{n}{8} + \frac{n}{16} + \cdots = 2n$$

first pass through the algorithm

on averge for 2nd pass

on averge for 3rd pass

$\Theta(n)$

$$\sum_{i=0}^{\infty} \frac{1}{2^i} = 2$$

Situation so far:

- Sort then search takes worst-case $\Theta(n \log n)$
- Randomized selection takes worst-case expected time $\Theta(n)$

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 <= i < j <= n such that $a_i = a_j$.

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

compare all pairs of elements $\to \Theta(n^2)$

sort then "linear search" $\to \Theta(n \cdot \log n)$

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

How much memory does it require?

no additional memory

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

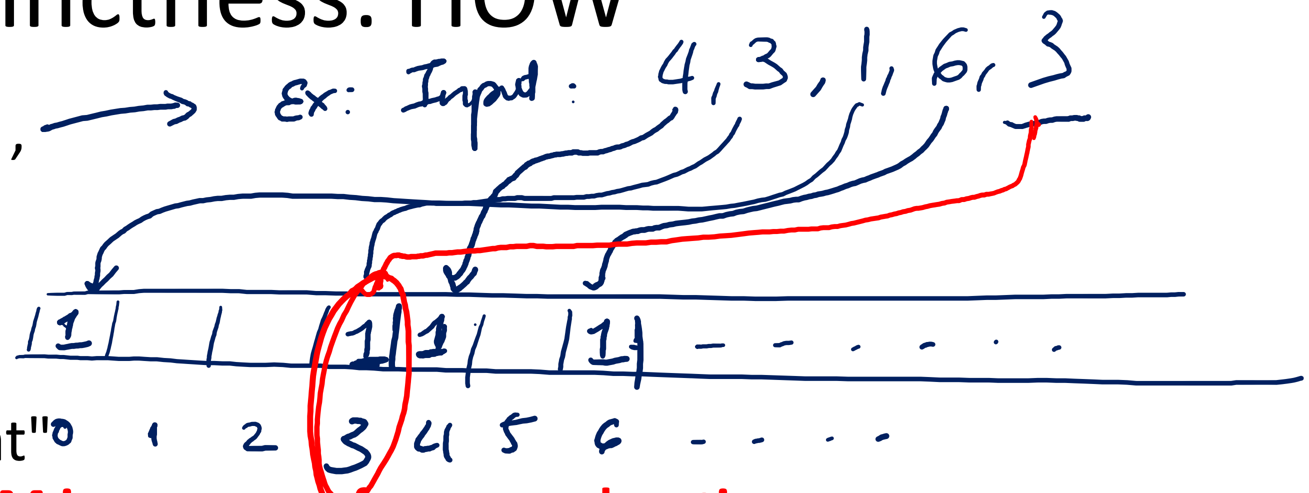
Element Distinctness: HOW

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 $1 <= i < j <= n$ 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"

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

Memory here is decided by the maximum value in your list.

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"

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

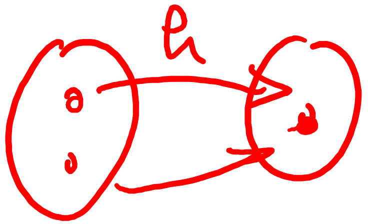
Element Distinctness: HOW

To simulate having more memory locations: use **Virtual Memory**.

Define **hash function**

h: { desired memory locations } → { actual memory locations }

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



Element Distinctness: HOW

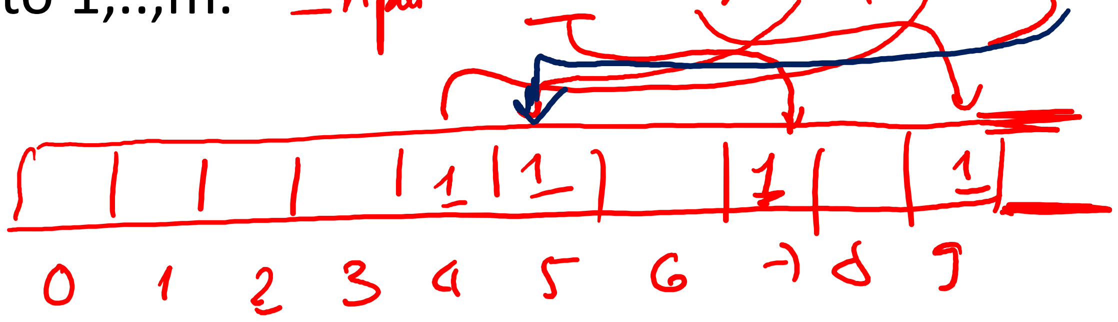
Given list of positive integers A = $a_1, a_2, \ldots, a_n$, and m memory locations available

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"

Ex: h is mod 10

Input: 27, 19, 35, 24, 35



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,..,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"

· If n < m. → fine.

· If n > m ?

linear?

~~infinite?~~

memory use depends $\boxed{m}$

What's the runtime of this algorithm?

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

What's the memory use of this algorithm?

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

Element Distinctness: WHY

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

Input: 27, 19, 17, ...

h = mod 10

1 | 1

27 19

17

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?

Correctness: Goal is

If there is a repetition, algorithm finds it ✓

If there is no repetition, algorithm reports "Distinct elements" ✗

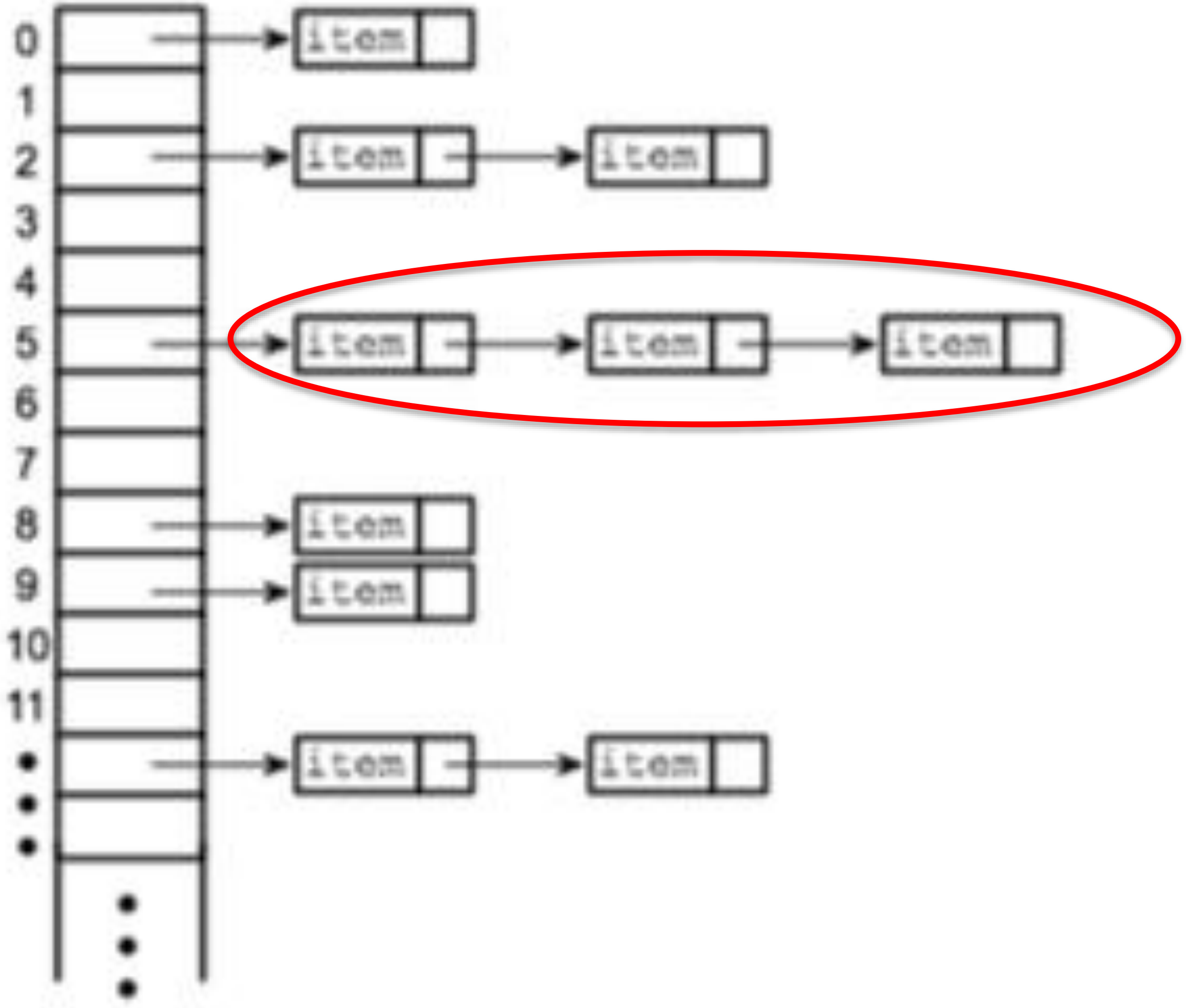
Hash Collisions

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: 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" ✓