CS3000

6/14- Weds

Admin

- Exam tom.
- 6/20 → Rec 3-5 due 9pm

  Second chance HW due 9pm
- 6/20 → Recitation 5

  Short HW4 out, due 6/22
- 6/21 → last lecture (form)
- 6/22 → Exam question make-up day

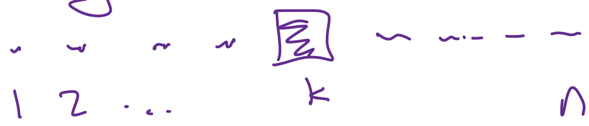
Agenda

1. Selection Problem
2. Exam Qs

1. Selection Problem

- given an unsorted array A of length n
- want: k^{th} smallest element in A

If array were sorted...



- How can we solve this problem?

  ↳ breadth, not depth (don't worry about randomization yet)

- sort, return A[k] $\Theta(n \lg n)$
- make a min heap, extract min k times $\Theta(n \lg n + k \lg n)$
- Find the min k times $\Theta(kn)$
- Selection sort up to position k $\Theta(kn)$
- Quicksort partition, pivot in right position ? it depends
- counter to track what we've seen so far
- sleep sort to find min on subarrays
- pick random element, compare to everything else to see what position it would be in

Goal:

- introduce randomization
- expected run-time better (?) than a known w.c.

RAND SELECT

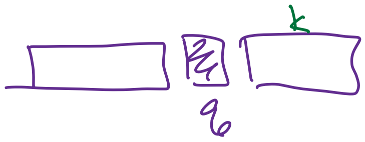
- shuffle input (any permutation equally likely)
- Partition on leftmost element A[p]
- q = final position of pivot
- is q == k? Done! Return A[q]
- is q < k?
  - throw away left, recurse on right
- is q > k?
  - throw away right, recurse left

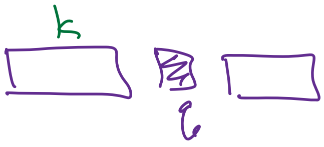
Options for randomizing

today → 1. shuffle input

2. choose random element → instead of next

yesterday





ex ⟨11, 10, 8, 13, 9, 3⟩ ↳ pivot

n = 6

Want: 2nd smallest element

partition

10, 8, 9, 3, $\boxed{11}$ 13
5

q > k
(looks unbalanced/unlucky)

throw away $\boxed{11, 13}$

10, 8, 9, 3
↳ pivot

partition

8, 9, 3, $\boxed{10}$
4

q > k
(v. unbalanced/unlucky)

throw away $\boxed{10}$

8, 9, 3

↳ pivot

partition

$3, \underset{2}{\boxed{8}}, 9$

q == k

return A[2] = 8

Overall expected run-time

- similar to Rand. Quicksort
- should be better than actual Quicksort
- hopefully better than w.c. run-times above
  ↳ time to beat $\Theta(n \lg n)$
- balanced partition is 1/4 n to 3/4 n

RandSelect
Randomized Quicksort } Las Vegas Algorithms

- guarantee correctness
- expected run-time

Monte Carlo Algorithms

- expected correctness
- guaranteed run time

} RandSelect that runs m times, instead of "until we find it"

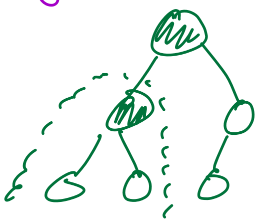
2. Exam Qs

- 6/15 in class until 11:30
  - ↳ 90 minute exam
- 8.5x11 inch cheat sheet, one side
- extra scratch paper
- Greedy, Heapsort, Am. Analysis, Graphs

Qs....

Heapify

- almost perfect min heap
- root might be wrong
- bubble down to correct position
- heap stored as array



- root is at $A[i]$

compare root to left $A[2i]$

and right $A[2i+1]$

swap with smaller

recursively heapify rest of heap

Amortized Analysis

- traditional: in a sequence of n operations
  assume w.c. n times

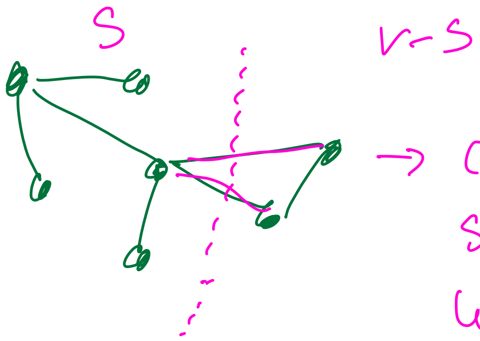
- amortized: in a sequence of n operations
  count the # times w.c. happens
  count the # times b.c. happens

$T(n) = \#w.c. * cost + \#b.c. * cost$

$\lg n$ w.c.

$n - \lg n$ b.c $\approx$ n b.c.

Cut: partition vertices of a graph



→ cross the cut

safe: in eventual MST

light: min edge crossing the cut