Lecture 4: Linear Time Selection/Median

Michael Dinitz

September 9, 2021
601.433/633 Introduction to Algorithms

Intro and Problem Definition

Last time: sorting in expected $\mathbf{O(n \log n)}$ time (randomized quicksort)

- Should already know (from Data Structures) deterministic $\mathbf{O(n \log n)}$ algorithms for sorting (mergesort, heapsort)

Today: two related problems

- Median: Given array $\mathbf{A}$ of length $\mathbf{n}$, find the median: $\lceil \mathbf{n/2} \rceil$nd smallest element.
- Selection: Given array $\mathbf{A}$ of length $\mathbf{n}$ and $\mathbf{k} \in [\mathbf{n}] = \{\mathbf{1, 2, \ldots, n}\}$, find $\mathbf{k}$'th smallest element.

Can solve both in $\mathbf{O(n \log n)}$ time via sorting. Faster?

Warmup

$\mathbf{k = 1}$:

Warmup

$\mathbf{k = 1}$: Scan through array, keeping track of smallest. $\mathbf{O(n)}$ time.

Warmup

$\mathbf{k = 1}$: Scan through array, keeping track of smallest. $\mathbf{O(n)}$ time.
$\mathbf{k = n}$:

Warmup

$\mathbf{k = 1}$: Scan through array, keeping track of smallest. $\mathbf{O(n)}$ time.
$\mathbf{k = n}$: Scan through array, keeping track of largest. $\mathbf{O(n)}$ time.

Warmup

$\mathbf{k} = \mathbf{1}$: Scan through array, keeping track of smallest. $\mathbf{O(n)}$ time.
$\mathbf{k} = \mathbf{n}$: Scan through array, keeping track of largest. $\mathbf{O(n)}$ time.
$\mathbf{k} = \mathbf{O(1)}$ or $\mathbf{k} = \mathbf{n} - \mathbf{O(1)}$:

Warmup

$\mathbf{k} = \mathbf{1}$: Scan through array, keeping track of smallest. $\mathbf{O(n)}$ time.

$\mathbf{k} = \mathbf{n}$: Scan through array, keeping track of largest. $\mathbf{O(n)}$ time.

$\mathbf{k} = \mathbf{O(1)}$ or $\mathbf{k} = \mathbf{n} - \mathbf{O(1)}$: keep track of $\mathbf{k}$ smallest/largest. $\mathbf{O(n)}$ time.

$n - k$

Warmup

$\mathbf{k} = \mathbf{1}$: Scan through array, keeping track of smallest. $\mathbf{O(n)}$ time.
$\mathbf{k} = \mathbf{n}$: Scan through array, keeping track of largest. $\mathbf{O(n)}$ time.
$\mathbf{k} = \mathbf{O(1)}$ or $\mathbf{k} = \mathbf{n} - \mathbf{O(1)}$: keep track of $\mathbf{k}$ smallest/largest. $\mathbf{O(n)}$ time.

Does this work when $\mathbf{k} = \mathbf{n/2}$?

Warmup

$\mathbf{k} = \mathbf{1}$: Scan through array, keeping track of smallest. $\mathbf{O(n)}$ time.
$\mathbf{k} = \mathbf{n}$: Scan through array, keeping track of largest. $\mathbf{O(n)}$ time.
$\mathbf{k} = \mathbf{O(1)}$ or $\mathbf{k} = \mathbf{n} - \mathbf{O(1)}$: keep track of $\mathbf{k}$ smallest/largest. $\mathbf{O(n)}$ time.

Does this work when $\mathbf{k} = \mathbf{n/2}$? $\approx \frac{n}{2}$

- Need to keep track of $\mathbf{k}\cancel{\mathbf{/2}}$ smallest.

Warmup

$\mathbf{k} = \mathbf{1}$: Scan through array, keeping track of smallest. $\mathbf{O(n)}$ time.
$\mathbf{k} = \mathbf{n}$: Scan through array, keeping track of largest. $\mathbf{O(n)}$ time.
$\mathbf{k} = \mathbf{O(1)}$ or $\mathbf{k} = \mathbf{n} - \mathbf{O(1)}$: keep track of $\mathbf{k}$ smallest/largest. $\mathbf{O(n)}$ time.

Does this work when $\mathbf{k} = \mathbf{n/2}$?

- Need to keep track of $\mathbf{k/2}$ smallest.
- When scanning, see an element, need to determine if one of $\mathbf{k}$ smallest. If yes, remove previous max of the $\cancel{\mathbf{k}}\mathbf{/2}$ we've been keeping track of.
  - Not easy to do! Foreshadow: would need to use a *heap*. $\mathbf{\Theta(\log n)}$-worst case time.

Warmup

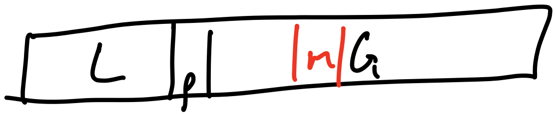
$\mathbf{k} = \mathbf{1}$: Scan through array, keeping track of smallest. $\mathbf{O(n)}$ time.
$\mathbf{k} = \mathbf{n}$: Scan through array, keeping track of largest. $\mathbf{O(n)}$ time.
$\mathbf{k} = \mathbf{O(1)}$ or $\mathbf{k} = \mathbf{n} - \mathbf{O(1)}$: keep track of $\mathbf{k}$ smallest/largest. $\mathbf{O(n)}$ time.

Does this work when $\mathbf{k} = \mathbf{n/2}$?

- Need to keep track of $\mathbf{k/2}$ smallest.
- When scanning, see an element, need to determine if one of $\mathbf{k}$ smallest. If yes, remove previous max of the $\mathbf{k/2}$ we've been keeping track of.
  - Not easy to do! Foreshadow: would need to use a *heap*. $\mathbf{\Theta(\log n)}$-worst case time.
- $\mathbf{\Theta(n \log n)}$ worst-case time.

(Randomized) Quickselect

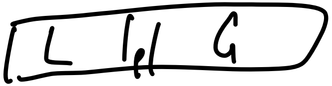
Main idea: (Randomized) Quicksort, but only recurse on side with element we're looking for.



(Randomized) Quickselect

Main idea: (Randomized) Quicksort, but only recurse on side with element we're looking for.

R-Quickselect($\mathbf{A}, \mathbf{k}$):



1. If $|\mathbf{A}| = \mathbf{1}$, return the element.
2. Pick a pivot element $\mathbf{p}$ uniformly at random from $\mathbf{A}$.
3. Compare each element of $\mathbf{A}$ to $\mathbf{p}$, creating subarrays $\mathbf{L}$ of elements less than $\mathbf{p}$ and $\mathbf{G}$ of elements greater than $\mathbf{p}$.
4. 4.1 If $|\mathbf{L}| = \mathbf{k} - \mathbf{1}$:

(Randomized) Quickselect

Main idea: (Randomized) Quicksort, but only recurse on side with element we're looking for.

R-Quickselect($\mathbf{A}, \mathbf{k}$):

1. If $|\mathbf{A}| = \mathbf{1}$, return the element.
2. Pick a pivot element $\mathbf{p}$ uniformly at random from $\mathbf{A}$.
3. Compare each element of $\mathbf{A}$ to $\mathbf{p}$, creating subarrays $\mathbf{L}$ of elements less than $\mathbf{p}$ and $\mathbf{G}$ of elements greater than $\mathbf{p}$.
4. 4.1 If $|\mathbf{L}| = \mathbf{k} - \mathbf{1}$: return $\mathbf{p}$.

(Randomized) Quickselect

Main idea: (Randomized) Quicksort, but only recurse on side with element we're looking for.

R-Quickselect($\mathbf{A}, \mathbf{k}$):

1. If $|\mathbf{A}| = \mathbf{1}$, return the element.
2. Pick a pivot element $\mathbf{p}$ uniformly at random from $\mathbf{A}$.
3. Compare each element of $\mathbf{A}$ to $\mathbf{p}$, creating subarrays $\mathbf{L}$ of elements less than $\mathbf{p}$ and $\mathbf{G}$ of elements greater than $\mathbf{p}$.
4. 4.1 If $|\mathbf{L}| = \mathbf{k} - \mathbf{1}$: return $\mathbf{p}$.
   4.2 if $|\mathbf{L}| > \mathbf{k} - \mathbf{1}$:

(Randomized) Quickselect

Main idea: (Randomized) Quicksort, but only recurse on side with element we're looking for.

R-Quickselect$(\mathbf{A}, \mathbf{k})$:

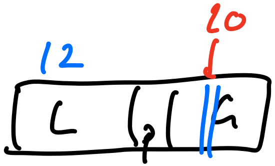
1. If $|\mathbf{A}| = \mathbf{1}$, return the element.
2. Pick a pivot element $\mathbf{p}$ uniformly at random from $\mathbf{A}$.
3. Compare each element of $\mathbf{A}$ to $\mathbf{p}$, creating subarrays $\mathbf{L}$ of elements less than $\mathbf{p}$ and $\mathbf{G}$ of elements greater than $\mathbf{p}$.
4. 4.1 If $|\mathbf{L}| = \mathbf{k} - \mathbf{1}$: return $\mathbf{p}$.
   4.2 if $|\mathbf{L}| > \mathbf{k} - \mathbf{1}$: return R-Quickselect$(\mathbf{L}, \mathbf{k})$.

(Randomized) Quickselect

Main idea: (Randomized) Quicksort, but only recurse on side with element we're looking for.

R-Quickselect($\mathbf{A}, \mathbf{k}$):

1. If $|\mathbf{A}| = \mathbf{1}$, return the element.
2. Pick a pivot element $\mathbf{p}$ uniformly at random from $\mathbf{A}$.
3. Compare each element of $\mathbf{A}$ to $\mathbf{p}$, creating subarrays $\mathbf{L}$ of elements less than $\mathbf{p}$ and $\mathbf{G}$ of elements greater than $\mathbf{p}$.
4. 4.1 If $|\mathbf{L}| = \mathbf{k} - \mathbf{1}$: return $\mathbf{p}$.
   4.2 if $|\mathbf{L}| > \mathbf{k} - \mathbf{1}$: return R-Quickselect($\mathbf{L}, \mathbf{k}$).
   4.3 If $|\mathbf{L}| < \mathbf{k} - \mathbf{1}$:



R-Quickselect(G, k) BAD

(Randomized) Quickselect

Main idea: (Randomized) Quicksort, but only recurse on side with element we're looking for.

R-Quickselect$(\mathbf{A}, \mathbf{k})$:

1. If $|\mathbf{A}| = \mathbf{1}$, return the element.
2. Pick a pivot element $\mathbf{p}$ uniformly at random from $\mathbf{A}$.
3. Compare each element of $\mathbf{A}$ to $\mathbf{p}$, creating subarrays $\mathbf{L}$ of elements less than $\mathbf{p}$ and $\mathbf{G}$ of elements greater than $\mathbf{p}$.
4. 4.1 If $|\mathbf{L}| = \mathbf{k} - \mathbf{1}$: return $\mathbf{p}$.
   4.2 if $|\mathbf{L}| > \mathbf{k} - \mathbf{1}$: return R-Quickselect$(\mathbf{L}, \mathbf{k})$.
   4.3 If $|\mathbf{L}| < \mathbf{k} - \mathbf{1}$: return R-Quickselect$(\mathbf{G}, \mathbf{k} - |\mathbf{L}| - \mathbf{1})$.

Quickselect: Correctness

Sketch here: good exercise to do at home!

Quickselect: Correctness

Sketch here: good exercise to do at home!

Prove by induction ("loop invariant") that on any call to R-Quickselect($\mathbf{X}, \mathbf{a}$), the element we're looking for is $\mathbf{a}$'th smallest of $\mathbf{X}$.

- Base case: first call to R-Quickselect($\mathbf{A}, \mathbf{k}$). Correct by definition.
- Inductive case: suppose was true for call R-Quickselect($\mathbf{Y}, \mathbf{b}$).
  - If we return element: correct
  - If we recurse on $\mathbf{L}$: correct
  - If we recurse on $\mathbf{G}$: correct

Quickselect: Running Time

Intuition:

Quickselect: Running Time

Intuition:

- Random pivot should be "near middle", so splits array "approximately in half".
- $\mathbf{O(\log n)}$ recursive calls, but each one on an array of half the size
  $\implies \mathbf{T(n) = T(n/2) + cn} \implies \mathbf{O(n)}$ time

Quickselect: Running Time

Intuition:

- Random pivot should be "near middle", so splits array "approximately in half".
- $\mathbf{O(\log n)}$ recursive calls, but each one on an array of half the size
  $\implies \mathbf{T(n) = T(n/2) + cn} \implies \mathbf{O(n)}$ time

Formalize this. Let $\mathbf{T(n)}$ be expected # comparisons on array of size $\mathbf{n}$.

Quickselect: Running Time

Intuition:

- Random pivot should be "near middle", so splits array "approximately in half".
- $\mathbf{O(\log n)}$ recursive calls, but each one on an array of half the size $\implies \mathbf{T(n) = T(n/2) + cn} \implies \mathbf{O(n)}$ time

Formalize this. Let $\mathbf{T(n)}$ be expected # comparisons on array of size $\mathbf{n}$.

- Splitting around pivot: $\mathbf{n-1}$ comparisons

Quickselect: Running Time

Intuition:

- Random pivot should be "near middle", so splits array "approximately in half".
- $\mathbf{O(\log n)}$ recursive calls, but each one on an array of half the size $\implies \mathbf{T(n) = T(n/2) + cn} \implies \mathbf{O(n)}$ time

Formalize this. Let $\mathbf{T(n)}$ be expected # comparisons on array of size $\mathbf{n}$.

- Splitting around pivot: $\mathbf{n-1}$ comparisons
- Recurse on either $\mathbf{L}$ or $\mathbf{G} \implies$ recursion costs at most $\mathbf{\max(T(|L|), T(|G|)) = T(\max(|L|, |G|))}$.

Quickselect: Running Time

Intuition:

- Random pivot should be "near middle", so splits array "approximately in half".
- $\mathbf{O(\log n)}$ recursive calls, but each one on an array of half the size $\implies \mathbf{T(n) = T(n/2) + cn} \implies \mathbf{O(n)}$ time

Formalize this. Let $\mathbf{T(n)}$ be expected # comparisons on array of size $\mathbf{n}$.

- Splitting around pivot: $\mathbf{n - 1}$ comparisons
- Recurse on either $\mathbf{L}$ or $\mathbf{G} \implies$ recursion costs at most $\mathbf{max(T(|L|), T(|G|)) = T(max(|L|, |G|))}$.
- $\mathbf{|L|, |G|}$ distributed uniformly among $[\mathbf{0, n - 1}]$.

Quickselect: Running Time

Intuition:

- Random pivot should be "near middle", so splits array "approximately in half".
- $\mathbf{O(\log n)}$ recursive calls, but each one on an array of half the size $\implies \mathbf{T(n) = T(n/2) + cn} \implies \mathbf{O(n)}$ time

Formalize this. Let $\mathbf{T(n)}$ be expected # comparisons on array of size $\mathbf{n}$.

- Splitting around pivot: $\mathbf{n-1}$ comparisons
- Recurse on either $\mathbf{L}$ or $\mathbf{G} \implies$ recursion costs at most $\mathbf{\max(T(|L|), T(|G|)) = T(\max(|L|, |G|))}$.
- $\mathbf{|L|, |G|}$ distributed uniformly among $\mathbf{[0, n-1]}$.

def of E[·]

$$\mathbf{T(n) \le (n-1) + \sum_{i=0}^{n-1} \frac{1}{n} T(\max(i, n-i-1))}$$

$$\mathbf{\le (n-1) + \sum_{i=0}^{n/2-1} \frac{1}{n} T(n-i-1) + \sum_{i=n/2}^{n-1} \frac{1}{n} T(i) = (n-1) + \frac{2}{n} \sum_{i=n/2}^{n-1} T(i)}$$

Quickselect: Running Time II

Want to solve recurrence relation $\mathbf{T(n) \leq (n-1) + \frac{2}{n}\sum_{i=n/2}^{n-1} T(i)}$.

Guess and check: $\mathbf{T(n) \leq 4n}$.

Quickselect: Running Time II

Want to solve recurrence relation $\mathbf{T(n) \le (n-1) + \frac{2}{n}\sum_{i=n/2}^{n-1} T(i)}$.

Guess and check: $\mathbf{T(n) \le 4n}$.

induction : $T(i) \le 4 \cdot i \quad \forall i < n$

$$
\begin{aligned}
T(n) &\le (n-1) + \frac{2}{n}\sum_{i=n/2}^{n-1} 4i = (n-1) + 4 \cdot \frac{2}{n}\sum_{i=n/2}^{n-1} i \\
&= (n-1) + 4 \cdot \frac{2}{n}\left(\sum_{i=1}^{n-1} i - \sum_{i=1}^{n/2-1} i\right) \\
&= (n-1) + 4 \cdot \frac{2}{n}\left(\frac{\cancel{n}(n-1)}{\cancel{2}} - \frac{\cancel{(n/2)}(n/2-1)}{2}\right) \\
&\le (n-1) + 4 \cdot \left((n-1) - \frac{n/2-1}{2}\right) \\
&\le (n-1) + 4\left(\frac{3n}{4}\right) \le 4n.
\end{aligned}
$$

Deterministic Version

Intuition:

- Randomization worked because it got us a "reasonably good" pivot.
- Simple deterministic pivot (first element, last element, etc.) bad because might not split array well.
- Deterministically find a pivot that's "close" to the middle?

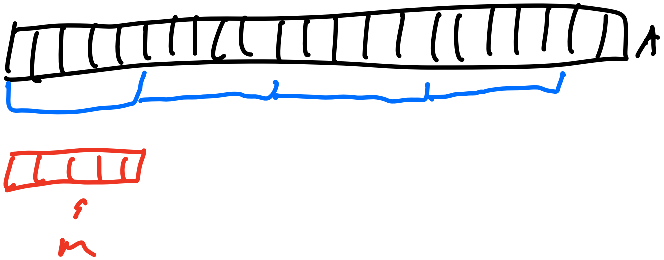
Deterministic Version

Intuition:

- Randomization worked because it got us a “reasonably good” pivot.
- Simple deterministic pivot (first element, last element, etc.) bad because might not split array well.
- Deterministically find a pivot that’s “close” to the middle?

Median-of-medians:

- Split **A** into **n/5** groups of **5** elements each.
- Compute median of each group.
- Let **p** be the median of the **n/5** medians



Deterministic Version

Intuition:

- Randomization worked because it got us a "reasonably good" pivot.
- Simple deterministic pivot (first element, last element, etc.) bad because might not split array well.
- Deterministically find a pivot that's "close" to the middle?

Median-of-medians:

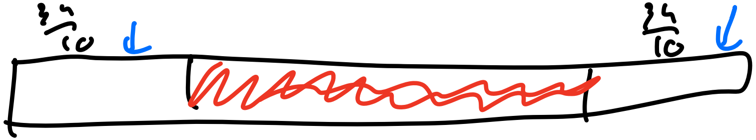
- Split $\mathbf{A}$ into $\mathbf{n/5}$ groups of $\mathbf{5}$ elements each.
- Compute median of each group.
- Let $\mathbf{p}$ be the median of the $\mathbf{n/5}$ medians

Want to claim: $\mathbf{p}$ is a good pivot, and can find $\mathbf{p}$ efficiently.

Median-of-Medians is good pivot

Theorem

$|\mathbf{L}|$ and $|\mathbf{G}|$ are both at most $\mathbf{7n/10}$ when $\mathbf{p}$ is median of medians.



Median-of-Medians is good pivot

Theorem

$|\mathbf{L}|$ *and* $|\mathbf{G}|$ *are both at most* $\mathbf{7n/10}$ *when* $\mathbf{p}$ *is median of medians.*

Let $\mathbf{B}$ be a group (of $\mathbf{5}$ elements), $\mathbf{m}$ median of $\mathbf{B}$:

–	–	**m**	–	–

Median-of-Medians is good pivot

Theorem

$|\mathbf{L}|$ *and* $|\mathbf{G}|$ *are both at most* $\mathbf{7n/10}$ *when* $\mathbf{p}$ *is median of medians.*

Let $\mathbf{B}$ be a group (of $\mathbf{5}$ elements), $\mathbf{m}$ median of $\mathbf{B}$:

–	–	**m**	–	–

- If $\mathbf{m} < \mathbf{p}$: at least three elements of $\mathbf{B}$ ($\mathbf{m}$ and two smaller) are in $\mathbf{L}$

Median-of-Medians is good pivot

Theorem

$|\mathbf{L}|$ *and* $|\mathbf{G}|$ *are both at most* $\mathbf{7n/10}$ *when* $\mathbf{p}$ *is median of medians.*

Let $\mathbf{B}$ be a group (of $\mathbf{5}$ elements), $\mathbf{m}$ median of $\mathbf{B}$:

–	–	**m**	–	–

- If $\mathbf{m} < \mathbf{p}$: at least three elements of $\mathbf{B}$ ($\mathbf{m}$ and two smaller) are in $\mathbf{L}$
- If $\mathbf{m} > \mathbf{p}$: at least three elements of $\mathbf{B}$ ($\mathbf{m}$ and two larger) are in $\mathbf{G}$

Median-of-Medians is good pivot

Theorem

$|\mathbf{L}|$ and $|\mathbf{G}|$ are both at most $\mathbf{7n/10}$ when $\mathbf{p}$ is median of medians.

Let $\mathbf{B}$ be a group (of $\mathbf{5}$ elements), $\mathbf{m}$ median of $\mathbf{B}$:

–	–	**m**	–	–

- If $\mathbf{m} < \mathbf{p}$: at least three elements of $\mathbf{B}$ ($\mathbf{m}$ and two smaller) are in $\mathbf{L}$
- If $\mathbf{m} > \mathbf{p}$: at least three elements of $\mathbf{B}$ ($\mathbf{m}$ and two larger) are in $\mathbf{G}$

By definition of $\mathbf{p}$, $\mathbf{n/10}$ groups have $\mathbf{m} < \mathbf{p}$ and $\mathbf{n/10}$ have $\mathbf{m} > \mathbf{p}$

Median-of-Medians is good pivot

Theorem

$|\mathbf{L}|$ *and* $|\mathbf{G}|$ *are both at most* $\mathbf{7n/10}$ *when* $\mathbf{p}$ *is median of medians.*

Let $\mathbf{B}$ be a group (of $\mathbf{5}$ elements), $\mathbf{m}$ median of $\mathbf{B}$:

–	–	**m**	–	–

- If $\mathbf{m} < \mathbf{p}$: at least three elements of $\mathbf{B}$ ($\mathbf{m}$ and two smaller) are in $\mathbf{L}$
- If $\mathbf{m} > \mathbf{p}$: at least three elements of $\mathbf{B}$ ($\mathbf{m}$ and two larger) are in $\mathbf{G}$

By definition of $\mathbf{p}$, $\mathbf{n/10}$ groups have $\mathbf{m} < \mathbf{p}$ and $\mathbf{n/10}$ have $\mathbf{m} > \mathbf{p}$

$$|\mathbf{L}| \geq \frac{\mathbf{n}}{\mathbf{10}} \cdot \mathbf{3} = \frac{\mathbf{3n}}{\mathbf{10}} \implies |\mathbf{G}| \leq \frac{\mathbf{7n}}{\mathbf{10}}$$

groups with $m < p$

Median-of-Medians is good pivot

Theorem

$|\mathbf{L}|$ and $|\mathbf{G}|$ are both at most $\mathbf{7n/10}$ when $\mathbf{p}$ is median of medians.

Let $\mathbf{B}$ be a group (of $\mathbf{5}$ elements), $\mathbf{m}$ median of $\mathbf{B}$:

–	–	m	–	–

- If $\mathbf{m} < \mathbf{p}$: at least three elements of $\mathbf{B}$ ($\mathbf{m}$ and two smaller) are in $\mathbf{L}$
- If $\mathbf{m} > \mathbf{p}$: at least three elements of $\mathbf{B}$ ($\mathbf{m}$ and two larger) are in $\mathbf{G}$

By definition of $\mathbf{p}$, $\mathbf{n/10}$ groups have $\mathbf{m} < \mathbf{p}$ and $\mathbf{n/10}$ have $\mathbf{m} > \mathbf{p}$

$$|\mathbf{L}| \geq \frac{\mathbf{n}}{\mathbf{10}} \cdot \mathbf{3} = \frac{\mathbf{3n}}{\mathbf{10}} \implies |\mathbf{G}| \leq \frac{\mathbf{7n}}{\mathbf{10}}$$

$$|\mathbf{G}| \geq \frac{\mathbf{n}}{\mathbf{10}} \cdot \mathbf{3} = \frac{\mathbf{3n}}{\mathbf{10}} \implies |\mathbf{L}| \leq \frac{\mathbf{7n}}{\mathbf{10}}$$

Finding Median of Medians

Have **n/5** elements (median of each group). Want to find median.

What problem is this?

Finding Median of Medians

Have **n/5** elements (median of each group). Want to find median.

What problem is this? Median / Selection!

Finding Median of Medians

Have **n/5** elements (median of each group). Want to find median.

What problem is this? Median / Selection!

Recursion! Use same algorithm on array of medians.

BPFRT

Algorithm due to Blum-Pratt-Floyd-Rivest-Tarjan.

BPFRT

Algorithm due to Blum-Pratt-Floyd-Rivest-Tarjan.

BPFRT($\mathbf{A}, \mathbf{k}$)

BPFRT

Algorithm due to Blum-Pratt-Floyd-Rivest-Tarjan.

BPFRT$(\mathbf{A}, \mathbf{k})$

1. Group $\mathbf{A}$ into $\mathbf{n/5}$ groups of $\mathbf{5}$, and let $\mathbf{A'}$ be an array of size $\mathbf{n/5}$ containing the median of each group.
2. Let $\mathbf{p} =$ BPFRT$(\mathbf{A'}, \mathbf{n/10})$, i.e., recursively find the median $\mathbf{p}$ of $\mathbf{A'}$ (the median-of-the-medians).

BPFRT

Algorithm due to Blum-Pratt-Floyd-Rivest-Tarjan.

BPFRT($\mathbf{A}, \mathbf{k}$) Let $T(n)$ = running time on array of size n

1. Group $\mathbf{A}$ into $\mathbf{n/5}$ groups of $\mathbf{5}$, and let $\mathbf{A'}$ be an array of size $\mathbf{n/5}$ containing the median of each group. $O(n)$ $O(n)$
2. Let $\mathbf{p}$ = BPFRT$(\mathbf{A'}, \mathbf{n/10})$, i.e., recursively find the median $\mathbf{p}$ of $\mathbf{A'}$ (the median-of-the-medians). $T(n/5)$ $T(n/3)$
3. Split $\mathbf{A}$ using $\mathbf{p}$ as a pivot into $\mathbf{L}$ and $\mathbf{G}$. $O(n)$ $O(n)$
4. Recurse on the appropriate piece:
   - 4.1 if $|\mathbf{L}| = \mathbf{k} - \mathbf{1}$ then return $\mathbf{p}$.
   - 4.2 if $|\mathbf{L}| > \mathbf{k} - \mathbf{1}$ then return BPFRT$(\mathbf{L}, \mathbf{k})$.
   - 4.3 if $|\mathbf{L}| < \mathbf{k} - \mathbf{1}$ then return BPFRT$(\mathbf{G}, \mathbf{k} - |\mathbf{L}| - \mathbf{1})$. $\leq T\left(\frac{7n}{10}\right)$ $T\left(\frac{2n}{3}\right)$

BPFRT Analysis

Let $\mathbf{T(n)}$ be (worst-case) running time on $\mathbf{A}$ of size $\mathbf{n}$.

- Step 1: $\mathbf{O(n)}$ time
- Step 2: $\mathbf{T(n/5)}$ time
- Step 3: $\mathbf{O(n)}$ time
- Step 4: $\mathbf{T(7n/10)}$ time

BPFRT Analysis

Let $\mathbf{T(n)}$ be (worst-case) running time on $\mathbf{A}$ of size $\mathbf{n}$.

- Step 1: $\mathbf{O(n)}$ time
- Step 2: $\mathbf{T(n/5)}$ time
- Step 3: $\mathbf{O(n)}$ time
- Step 4: $\mathbf{T(7n/10)}$ time

$$\mathbf{T(n) \leq T(7n/10) + T(n/5) + cn}$$

$$T\left(\frac{2n}{3}\right) + T\left(\frac{n}{3}\right) + cn$$

$$= \Theta(n \log n)$$

BPFRT Analysis

Let $\mathbf{T(n)}$ be (worst-case) running time on $\mathbf{A}$ of size $\mathbf{n}$.

- Step 1: $\mathbf{O(n)}$ time
- Step 2: $\mathbf{T(n/5)}$ time
- Step 3: $\mathbf{O(n)}$ time
- Step 4: $\mathbf{T(7n/10)}$ time

$$\mathbf{T(n) \le T(7n/10) + T(n/5) + cn}$$

Guess $\mathbf{T(n) \le 10cn}$:

induction

$$\mathbf{T(n) \le 10c(7n/10) + 10c(n/5) + cn = 9cn + cn = 10cn}$$

Deterministic $\mathbf{O(n \log n)}$ Quicksort

Can use this to get *deterministic* $\mathbf{O(n \log n)}$-time Quicksort!

Deterministic $\mathbf{O(n \log n)}$ Quicksort

Can use this to get *deterministic* $\mathbf{O(n \log n)}$-time Quicksort!
Use BPFRT($\mathbf{A, n/2}$) to choose median as pivot.

Deterministic $\mathbf{O(n \log n)}$ Quicksort

Can use this to get *deterministic* $\mathbf{O(n \log n)}$-time Quicksort!
Use BPFRT($\mathbf{A, n/2}$) to choose median as pivot.

Let $\mathbf{T(n)}$ be time on input of size $\mathbf{n}$.

- BPFRT to find pivot takes $\mathbf{O(n)}$ time
- Splitting around pivot takes $\mathbf{O(n)}$ time
- Each recursive call takes $\mathbf{T(n/2)}$ time

Deterministic $O(n \log n)$ Quicksort

Can use this to get *deterministic* $O(n \log n)$-time Quicksort!
Use BPFRT($A, n/2$) to choose median as pivot.

Let $T(n)$ be time on input of size n.

- BPFRT to find pivot takes $O(n)$ time
- Splitting around pivot takes $O(n)$ time
- Each recursive call takes $T(n/2)$ time

$$T(n) = 2T(n/2) + cn \implies T(n) = \Theta(n \log n)$$