Lecture 5: Linear Time Selection/Median

Michael Dinitz

September 9, 2025

601.433/633 Introduction to Algorithms

Intro and Problem Definition

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

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

Today: two related problems

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

Intro and Problem Definition

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

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

Today: two related problems

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

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

Warmup

$\boldsymbol{k = 1}$:

Warmup

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

Warmup

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

Warmup

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

Warmup

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

Warmup

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

Warmup

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

Does this work when $k = n/2$?

Warmup

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

Does this work when $k = n/2$?

- Need to keep track of $n/2$ smallest.

Warmup

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

Does this work when $k = n/2$?

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

Warmup

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

Does this work when $k = n/2$?

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

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

(Randomized) Quickselect

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

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

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

(Randomized) Quickselect

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

R-Quickselect(A, k):

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

(Randomized) Quickselect

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

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

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

(Randomized) Quickselect

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

R-Quickselect(A, k):

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

(Randomized) Quickselect

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

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

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

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($\boldsymbol{X}, \boldsymbol{a}$), the element we're looking for is $\boldsymbol{a}$'th smallest of $\boldsymbol{X}$.

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

Quickselect: Running Time

Intuition:

Quickselect: Running Time

Intuition:

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

Quickselect: Running Time

Intuition:

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

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

Quickselect: Running Time

Intuition:

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

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

- Splitting around pivot: $n - 1$ comparisons

Quickselect: Running Time

Intuition:

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

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

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

Quickselect: Running Time

Intuition:

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

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

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

Quickselect: Running Time

Intuition:

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

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

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

$$
\begin{aligned}
T(n) &\le (n-1) + \sum_{i=0}^{n-1} \frac{1}{n} T(\max(i, n-i-1)) \\
&\le (n-1) + \sum_{i=0}^{n/2-1} \frac{1}{n} T(n-i-1) + \sum_{i=n/2}^{n-1} \frac{1}{n} T(i) = (n-1) + \frac{2}{n} \sum_{i=n/2}^{n-1} T(i)
\end{aligned}
$$

Quickselect: Running Time II

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

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

Quickselect: Running Time II

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

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

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

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?

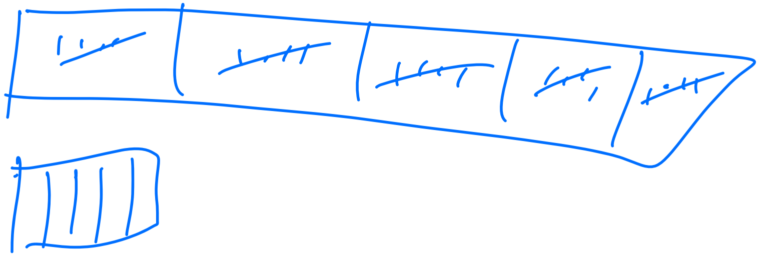
Deterministic Version

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

Want to claim: p is a good pivot, and can find p efficiently.

Median-of-Medians is good pivot

Theorem

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

Median-of-Medians is good pivot

Theorem

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

Let B be a group (of 5 elements), m median of B:

–	–	m	–	–

Median-of-Medians is good pivot

Theorem

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

Let B be a group (of 5 elements), m median of B:

–	–	m	–	–

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

Median-of-Medians is good pivot

Theorem

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

Let B be a group (of 5 elements), m median of B:

–	–	m	–	–

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

Median-of-Medians is good pivot

Theorem

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

Let B be a group (of 5 elements), m median of B:

–	–	m	–	–

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

By definition of p, $n/10$ groups have $m < p$ and $n/10$ have $m > p$

Median-of-Medians is good pivot

Theorem

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

Let B be a group (of 5 elements), m median of B:

–	–	m	–	–

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

By definition of p, $n/10$ groups have $m < p$ and $n/10$ have $m > p$

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

Median-of-Medians is good pivot

Theorem

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

Let B be a group (of 5 elements), m median of B:

–	–	m	–	–

- If $m < p$: at least three elements of B (m and two smaller) are in L 2
- If $m > p$: at least three elements of B (m and two larger) are in G 2

By definition of p, $n/10$ groups have $m < p$ and $n/10$ have $m > p$

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

$$|L| \geq \frac{n}{6} \cdot 2 = \frac{n}{3}$$

$$\Rightarrow |G| \leq \frac{2n}{3}$$

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

$$|L| \leq \frac{2n}{3}$$

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 $\boldsymbol{n/5}$ 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($\boldsymbol{A}, \boldsymbol{k}$)

BPFRT

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

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

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

BPFRT

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

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

1. Group $\boldsymbol{A}$ into $\boldsymbol{n/5}$ groups of $\mathbf{5}$, and let $\boldsymbol{A'}$ be an array of size $\boldsymbol{n/5}$ containing the median of each group.
2. Let $\boldsymbol{p} = \text{BPFRT}(\boldsymbol{A'}, \boldsymbol{n/10})$, i.e., recursively find the median $\boldsymbol{p}$ of $\boldsymbol{A'}$ (the median-of-the-medians).
3. Split $\boldsymbol{A}$ using $\boldsymbol{p}$ as a pivot into $\boldsymbol{L}$ and $\boldsymbol{G}$.
4. Recurse on the appropriate piece:
   4.1 if $|\boldsymbol{L}| = \boldsymbol{k} - \mathbf{1}$ then return $\boldsymbol{p}$.
   4.2 if $|\boldsymbol{L}| > \boldsymbol{k} - \mathbf{1}$ then return BPFRT($\boldsymbol{L}, \boldsymbol{k}$).
   4.3 if $|\boldsymbol{L}| < \boldsymbol{k} - \mathbf{1}$ then return BPFRT($\boldsymbol{G}, \boldsymbol{k} - |\boldsymbol{L}| - \mathbf{1}$).

$|A'|/2$ $O(n)$ $T(n/5)$ $T(n/3)$ $O(n)$ $T(\frac{7n}{10})$ $T(2n/3)$

BPFRT Analysis

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

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

BPFRT Analysis

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

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

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

$$T(2n/3) + T(n/3) + cn$$

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

BPFRT Analysis

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

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

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

Guess $T(n) \leq 10cn$:

$$T(n) \leq 10c(7n/10) + 10c(n/5) + cn = 9cn + cn = 10cn$$

Deterministic $O(n \log n)$ Quicksort

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

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.

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

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

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