Lecture 4: Linear Time Selection/Median

Michael Dinitz

September 10, 2020
601.433/633 Introduction to Algorithms
Announcements

- HW1 due now!
- HW2 due in 3 lectures: Tuesday 9/22
- Office hours basically set: see course webpage
  http://www.cs.jhu.edu/~mdimitz/classes/IntroAlgorithms/Fall2020/index.html
  (particularly embedded Google calendar)
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: $(n/2)$nd smallest element.
- Selection: Given array $A$ of length $n$ and $k \in [1, n]$, find $k$’th smallest element.

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

$k = 1$:
Warmup

\[ k = 1: \text{Scan through array, keeping track of smallest. } O(n) \text{ time.} \]
Warmup

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

\[ k = \Omega(1) \text{ or } k = n - \Omega(1) : \text{keep track of } k \text{ smallest/largest. } O(n) \text{ time.} \]

Does this work when \( k = n - 2 \)?

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

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

[Diagram of an array with question marks]
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/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/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/largest. \( O(n) \) time.

Does this work when \( k = n/2 \)?

- Need to keep track of \( k/2 \) smallest.
Warmup

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

Does this work when \( k = n/2 \)?

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

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

Does this work when \( k = n/2 \)?

- Need to keep track of \( k/2 \) smallest.
- When scanning, see an element, need to determine if one of \( k \) smallest. If yes, remove previous max of the \( k/2 \) 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.

R-Quickselect(A, k):
1. If n = 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.
4.1 If |L| = k - 1: return p.
4.2 If |L| > k - 1: return R-Quickselect(L, k).
4.3 If |L| < k - 1: return R-Quickselect(G, k - |L| - 1).
(Randomized) Quickselect

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

R-Quickselect(A, k):

1. If n = 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:
(Randomized) Quickselect

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

R-Quickselect(A, k):

1. If \( n = 1 \), 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 \( p \).
(Randomized) Quickselect

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

R-Quickselect($A$, $k$):

1. If $n = 1$, 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 $\lvert L \rvert = k - 1$: return $p$.
    4.2 if $\lvert L \rvert > k - 1$: 
(Randomized) Quickselect

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

R-Quickselect(A, k):

1. If \( n = 1 \), 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).
(Randomized) Quickselect

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

R-Quickselect($A, k$):

1. If $n = 1$, 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($A, k$):

1. If $n = 1$, 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 $|L| < k - 1$: return R-Quickselect($G, k - |L| - 1$).
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($X, a$), the element we’re looking for is $a$’th smallest of $X$.

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

Intuition:

Random pivot should be "near middle", so splits array "approximately in half".

\[ T(n) = T\left(\frac{n}{2}\right) + cn \]

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

Splitting around pivot:

\[ n - 1 \] comparisons

Recurse on either \( L \) or \( G \)

\[ \max(T(L), T(G)) = T(\max(L, G)) \]

\( L \), \( G \) distributed uniformly among \([0, n-1]\).

\[ T(n) \leq (n-1) + \sum_{i=0}^{1} \frac{1}{n} T(\max(i, n-i-1)) \]

\[ \leq (n-1) + \sum_{i=0}^{n-1} \frac{1}{2} T\left(\frac{n}{2}\right) \]

\[ = (n-1) + \frac{1}{2} \cdot 2 \cdot T\left(\frac{n}{2}\right) \]

\[ = (n-1) + \frac{1}{2} T(n) \]

\[ \Rightarrow T(n) = 2(n-1) + \frac{1}{2} T(n) \]

\[ \Rightarrow T(n) = \frac{4}{3} (n-1) \]

\[ = \Theta(n) \]
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
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
  \[\Longrightarrow T(n) = T(n/2) + cn \Longrightarrow O(n) \text{ time}\]

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

- Splitting around pivot: \(n - 1\) comparisons
- Recurse on either \(L\) or \(G\) \(\Longrightarrow\) 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
  \[T(n) = T(n/2) + cn \implies O(n)\] 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 \([0, n - 1]\).
  \[T(n) \leq (n - 1) + \sum_{i=0}^{n-1} \frac{1}{n} T(\max(i, n - i - 1))\]

\[\leq (n - 1) + \sum_{i=0}^{n/2-1} \frac{1}{n} T(n - i - 1) + \sum_{i=n/2}^{n-1} \frac{1}{n} T(i) = (n - 1) + \sum_{i=n/2}^{n-1} \frac{1}{n} T(i)\]
Quickselect: Running Time II

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

Guess and check: $T(n) \leq 4n$. 
Quickselect: Running Time II

Want to solve recurrence relation

\[ T(n) \leq (n - 1) + \frac{2}{n} \sum_{i=n/2}^{n-1} T(i) \]

Guess and check: \( T(n) \leq 4n \).

\[
\begin{align*}
T(n) &\leq (n - 1) + \frac{2}{n} \sum_{i=n/2}^{n-1} 4i \\
&= (n - 1) + 4 \cdot \frac{2}{n} \left( \sum_{i=1}^{n-1} i + \sum_{i=1}^{n/2-1} i \right) \\
&= (n - 1) + 4 \cdot \frac{2}{n} \left( \frac{n(n - 1)}{2} - \frac{(n/2)(n/2 - 1)}{2} \right) \\
&\leq (n - 1) + 4 \left( (n - 1) - \frac{n/2 - 1}{2} \right) \\
&\leq (n - 1) + 4 \left( \frac{3n}{4} \right) \leq 4n.
\end{align*}
\]
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 \( n/5 \) 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 $7n/10$ when $p$ is median of medians.
Median-of-Medians is good pivot

**Theorem**

\(|L| \text{ and } |G| \text{ are both at most } \frac{7n}{10} \text{ when } p \text{ is median of medians.}\)

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

\[
\begin{array}{cccc}
- & - & m & - \\
\end{array}
\]
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 \(B\):

- 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 $B$:

- 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 \(B\):

- 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 \(m > p\) (dot of mom)
Median-of-Medians is good pivot

**Theorem**

\[ |L| \text{ and } |G| \text{ are both at most } \frac{7n}{10} \text{ when } p \text{ is median of medians.} \]

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

\[
\begin{array}{cccc}
- & - & m & - & -
\end{array}
\]

- 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 \( p \), \( \frac{n}{10} \) groups have \( m < p \) and \( \frac{n}{10} \) 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**

\(|L| \text{ and } |G| \text{ are both at most } \frac{7n}{10} \text{ when } p \text{ is median of medians.}\\
\)

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

- 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 \(m > p\)

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

\(|G| \geq \frac{n}{10} \cdot 3 = \frac{3n}{10} \implies |L| \leq \frac{7n}{10}\\
\)
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 \( \frac{n}{5} \) elements (median of each group). Want to find median.

What problem is this? Median / Selection!

Recursion! Use same algorithm on array of medians.
Algorithm due to Blum-Pratt-Floyd-Rivest-Tarjan.

BPFRT

1. Group \( A \) into \( n \) groups of 5, and let \( A' \) be an array of size \( n \times 5 \) containing the median of each group.

2. Let \( p = BPFRT(A', n \times 5) \), i.e., recursively find the median \( p \) of \( A' \) (the median-of-the-medians).

3. Split \( A \) using \( p \) as a pivot into \( L \) and \( G \).

4. Recurse on the appropriate piece:
   4.1 if \( |L| = k - 1 \) then return \( p \).
   4.2 if \( |L| > k - 1 \) then return \( BPFRT(L, k) \).
   4.3 if \( |L| < k - 1 \) then return \( BPFRT(G, k - |L| - 1) \).
Algorithm due to Blum-Pratt-Floyd-Rivest-Tarjan.

\textbf{BPFRT}(A, k)
Algorithm due to Blum-Pratt-Floyd-Rivest-Tarjan.

BPFRT($A, k$)

1. Group $A$ into $n/5$ groups of 5, and let $A'$ be an array of size $n/5$ containing the median of each group.

2. Let $p = \text{BPFRT}(A', n/10)$, i.e., recursively find the median $p$ of $A'$ (the median-of-the-medians).
Algorithm due to Blum-Pratt-Floyd-Rivest-Tarjan.

BPFRT($A, k$)

1. Group $A$ into $n/5$ groups of 5, and let $A'$ be an array of size $n/5$ containing the median of each group.

2. Let $p = \text{BPFRT}(A', n/10)$, i.e., recursively find the median $p$ of $A'$ (the median-of-the-medians).


4. Recurse on the appropriate piece:
   
   4.1 if $|L| = k - 1$ then return $p$.
   
   4.2 if $|L| > k - 1$ then return $\text{BPFRT}(L, k)$.
   
   4.3 if $|L| < k - 1$ then return $\text{BPFRT}(G, k - |L| - 1)$.
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
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

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

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

\[
T(n) = 2T(n/2) + cn \quad \Rightarrow \quad T(n) = \Theta(n \log n)
\]