Linear-time Median

Def: Median of elements $A = a_1, a_2, ..., a_n$ is the $[n/2]$-th smallest element in $A$.

How to find median?

- sort the elements, output the elem. at $[n/2]$-th position
- running time? $O(n \log n)$
**Linear-time Median**

**Def:** Median of elements $A=a_1, a_2, \ldots, a_n$ is the $(n/2)$-th smallest element in $A$.

**How to find median?**

- sort the elements, output the elem. at $(n/2)$-th position
  - running time: $\Theta(n \log n)$
- we will see a faster algorithm
  - will solve a more general problem:

  $\text{SELECT} (A, k)$: returns the $k$-th smallest element in $A$
Linear-time Median

Idea: Suppose \( A = \{22, 5, 10, 11, 23, 15, 9, 8, 2, 0, 4, 20, 25, 1, 29, 24, 3, 12, 28, 14, 27, 19, 17, 21, 18, 6, 7, 13, 16, 26\} \)

1. If \( n \) (number of elements) \( \leq 5 \), then sort \( A \) using bubblesort and return the \( k \)-th smallest elem.
2. Split \( A \) into 5 tuples.
3. Find the median of each tuple and add all these elements to \( B \).
4. Find the median of the medians: \( x = \text{SELECT}(B, \frac{n}{5}) \).
5. Move \( x \) so that elements \( \leq x \) go left, then go elem. = \( x \), then elem. \( > x \) for us.
6. If \( k < j \) then return \( \text{SELECT}(\text{ArcaR}[:j], k) \).
7. If \( k > j \) then return \( \text{SELECT}(\text{ArcaR}[j+1:], n) \), rearranged \( A \):
8. If \( k=j \) then return \( x \).

Note: change the pseudo code to reflect multiple possible \( x \)'s
Linear-time Median

SELECT (A, k)
1. split A into n/5 groups of five elements
2. let b\textsubscript{i} be the median of the i-th group
3. let B = [b\textsubscript{1}, b\textsubscript{2}, ..., b\textsubscript{n/5}]
4. medianB = SELECT (B, B.length/2)
5. rearrange A so that all elements smaller than medianB come before medianB, all elements larger than medianB come after medianB, and elements equal to medianB are next to medianB
6. j = position of medianB in rearranged A (if more medianB’s, then take the closest position to n/2)
7. if (k < j) return SELECT (A[1...j-1], k )
8. if (k = j) return medianB
9. if (k > j) return SELECT (A[j+1...n], k-j )
Linear-time Median

Running the algorithm:

Sorting only for the analysis
(the algorithm does not sort B)

When making a recursive call on elements larger than \( x \), there are at most \( \frac{3}{4}N \) such elements.

Same for the recursive call on elements smaller than \( x \).
Linear-time Median

Running the algorithm:

Rearrange columns so that medianB in the “middle.”

Recurrence:

\[ T(n) \leq T\left(\frac{n}{3}\right) + T\left(\frac{3n}{4}\right) + cn \quad \forall n > 5 \]

\[ T(n) \leq c \quad \forall n \leq 5 \]
Linear-time Median

Recurrence: \( T(n) \leq T(n/5) + T(3n/4) + cn \) if \( n > 5 \)
\( T(n) \leq c \) if \( n < 6 \)

Claim: There exists a constant \( d \) such that \( T(n) \leq dn \).

**Base Case:** \( n \leq 5 \)
want to show \( T(n) \leq dn \)
know that \( T(n) \leq c \leq d \cdot n \)
for \( d \geq c \)

**Inductive Case:** \( n > 5 \)
want to show \( T(n) \leq dn \)
by IH we know \( T(m) \leq dm \) \( \forall m < n \)
know: \( T(n) \leq T(n/5) + T(3n/4) + cn \) (IH)
\( \leq d \cdot \frac{n}{5} + d \cdot \frac{3n}{4} + cn = \)
\( = \left( \frac{19}{20} d + c \right) \cdot n \)
want \( \leq d \cdot n \)

choose \( d \) so that this holds: \( \frac{19}{20} d + c \leq d \)

Just showed that \( T(n) \leq 20c \cdot n \)
\( = O(n) \)
Randomized Linear-time Median

Idea:
Instead of finding medianB, take a random element from A.

SELECT-RAND (A, k)
1. $x = a_i$ where $i$ = a random number from $\{1, \ldots, n\}$
2. rearrange A so that all elements smaller than $x$ come before $x$, all elements larger than $x$ come after $x$, and elements equal to $x$ are next to $x$
3. $j =$ position of $x$ in rearranged A (if more $x$’s, then take the closest position to n/2)
4. if ($k < j$) return SELECT-RAND ( A[1…j-1], k )
5. if ($k = j$) return medianB
6. if ($k > j$) return SELECT-RAND ( A[j+1…n], k-j)
**Randomized Linear-time Median**

**Worst case running time:** \( O(n^2) \).

**SELECT-RAND** \((A, k)\)

1. \( x = a_i \) where \( i = \) a random number from \( \{1, \ldots, n\} \)
2. rearrange \( A \) so that all elements smaller than \( x \) come before \( x \), all elements larger than \( x \) come after \( x \), and elements equal to \( x \) are next to \( x \)
3. \( j = \) position of \( x \) in rearranged \( A \) (if more \( x \)'s, then take the closest position to \( n/2 \))
4. if \( k < j \) return SELECT-RAND \((A[1\ldots j-1], k)\)
5. if \( k = j \) return medianB
6. if \( k > j \) return SELECT-RAND \((A[j+1\ldots n], k-j)\)
Randomized Linear-time Median

Worst case running time: $O(n^2)$.

Claim: Expected running time is $O(n)$. 
Master Theorem

Let \( a \geq 1 \) and \( b > 1 \) be constants, \( f(n) \) be a function and for positive integers we have a recurrence for \( T \) of the form

\[
T(n) = a \cdot T(n/b) + f(n),
\]

where \( n/b \) is rounded either way.

Then,

\[
O(n^{\log_a b - e})
\]

- If \( f(n) = O(n^{\log_a b - e}) \) for some constant \( e > 0 \), then
  \[
  T(n) = \Theta(n^{\log_a b}).
  \]

\[
\Omega(n^{\log_a b + e})
\]

- If \( f(n) = \Omega(n^{\log_a b + e}) \), then
  \[
  T(n) = \Omega(n^{\log_a b \log n}).
  \]

- If \( f(n) = \Omega(n^{\log_a b + e}) \) for some constant \( e > 0 \), and if \( af(n/b) \leq cf(n) \) for some constant \( c < 1 \) (and all sufficiently large \( n \)), then
  \[
  2f(\frac{n}{4}) = 2 \cdot c \cdot \frac{n}{4} \leq c_2 \cdot n \quad \checkmark
  \]
  \[
  T(n) = \Theta(f(n)).
  \]