Linear Time Bounds

for Median Computations

Manuel Blum
Robert Floyd
Vaughan Pratt
Ronald Rivest
Robert Tarjan
Given \( n \) numbers, how hard is it to find the \( i^{th} \) largest?

\[ i = 1 \Rightarrow \text{finding largest} \]

\[ i = \frac{n}{2} \Rightarrow \text{finding median} \]

Measure difficulty by the number of times we do a comparison of two numbers to see which is larger.

Consider maximum number of comparisons needed.
3 comparisons needed
Previous results

\[
\begin{align*}
 i &= 1 & n-1 & \text{needed} \\
 i &= \frac{n}{2} & \leq (n \log_2 n) & \text{needed} \\
 2 < i < n-1 & \quad ? & \left( \geq n-1 \leq n \log_2 n \right)
\end{align*}
\]

Our results

No more than \( 6n \) comparisons ever needed.

Also, \( \frac{3n}{2} - 3 \) necessary,

i.e. any algorithm can be forced to make at least \( \frac{3n}{2} - 3 \) comparisons.
Given:

A simple approach: (Hoare)

1. Take a guess (pick one at random).
2. How good is it? (Compare it to all others).
3. If it's OK, stop. If it's too big, throw it and all larger away. Too small...”

Start over.
DIVIDE
AND
CONQUER
ABOUT WHAT
YOU'D EXPECT
HAPPINESS IS GUESSING RIGHT!
LEFT HOLDING THE BAG!
IF AT FIRST you DON'T SUCCEED...
GETTING A GOOD GRIP ON THE PROBLEM
Essence of our procedure is making an intelligent guess: picking a non-extreme element.

**Method:**

1. Divide elements into \( \frac{n}{15} \) groups of 15 each.
2. Find median of each group. (by sorting them)
3. Let our guess be the median of the group medians (a recursive operation).
Algorithm

1. Group elements and sort them.
2. Find M, median of column medians, recursively.
3. Partition A and B about M by a binary search technique.
4. Stop if M is i-th largest, else discard either L or G (no more).
5. Merge remaining short columns into columns of size 15.

Go to step 2.
Let $M$ be median of medians:

- Columns with medians $\leq M$
- Columns with medians $\geq M$

Groups displayed as sorted columns with largest elements on top.

**Thm.** $M$ is less than, as well as greater than, $n/4$ elements.
Analysis:

\[ f(n) \text{ is upper bound on number of comparisons used, for all } c. \]

\[ f(n) \leq \frac{42n}{15} + g(n) \quad \text{(Sorting)} \]

\[ g(n) \text{ is upper bound on main loop.} \]

\[ g(n) \leq f\left(\frac{n}{15}\right) \quad \text{(To find } M) \]

\[ + \frac{n}{15} \cdot 0.3 \quad \text{(Partition A,B)} \]

\[ + \frac{n}{30} \cdot \frac{7}{15} \cdot 17 \quad \text{(Merge step)} \]

\[ + g\left(\frac{11n}{15}\right) \quad \text{(Return to top)} \]

Thm. \[ f(n) \leq 6\frac{1}{3}n \]

With some more work, \[ f(n) \leq 5.73n \]
A lower bound

\[ f(i, n) = \text{comparisons used by the optimal algorithm.} \]

\[ f^*(i, n) = \text{comparisons used by optimal algorithm, when given additional information now and then.} \]

\[ f(i, n) \geq f^*(i, n) \]

The additional information will be very special, allowing us to calculate \( f^*(i, n) \) easily:

any element compared against a second time will turn out to be either maximal or minimal of those remaining, and so can be discarded.
\[ f^*(i, n) \geq n + \min (i, n - i + 1) + \lceil \log_2(n) \rceil - 4 \]