Chapter A5: Finding the Median

The median of a collection is the middle value (in sorted order). For example, the collection \{3, 1, 6, 2, 8, 4, 9\} has a median of 4: There are three values smaller than 4 and three values larger. The answer to a median question when there is an even number of values is the smaller of the two middle values. (There are other possible definitions but this is convenient.)

We want an algorithm for finding the median of a collection of \(n\) values. One possibility would be to sort the list. But one hopes for a faster method. One can try recursion or iteration, but at first glance they don’t seem to work.

Blum, Floyd, Pratt, Rivest, and Tarjan provided a linear-time algorithm: it uses \(O(n)\) comparisons. It is a bit hairy at places.

A5.1 A Generalization

One approach to the median problem is to define a generalization: the rank selection problem. The input is a collection of \(n\) numbers and a rank \(k\). The output must be the value that is the \(k\)th smallest value. For example, for the above collection, 4 has rank 4 and 8 has rank 6. The minimum always has rank 1.

Thus the median of a collection with \(n\) values has rank \(\lceil n/2 \rceil\) (round up): it’s \(n/2\) if \(n\) is even and \((n + 1)/2\) if \(n\) is odd.

A5.2 Using Median for Rank Selection

Okay. We do not know how to find the median. But I now argue that if (and that’s a big if) we have an algorithm for median, then we can derive an algorithm for rank-selection:

We use the median algorithm to break the list in half. And keep only the relevant half.

How much work does this take?
**RankSelect** \((A:\text{valuelist}, k:\text{integer})\)

R1: Calculate the median \(m\) of the list \(A\).

R2: Determine the set \(A_{<}\) of elements smaller than the median.
   Determine the set \(A_{>}\) of elements larger than the median.

R3: If \(k < \lfloor n/2 \rfloor\) then return \(\text{RankSelect}(A_{<}, k)\)
   If \(k > \lceil n/2 \rceil\) then return \(\text{RankSelect}(A_{>}, k - \lfloor n/2 \rfloor)\)

Else return \(m\)

Suppose that the median procedure takes \(cn\) comparisons where \(c\) is some constant.
(That is, suppose we have a linear-time median algorithm.) The number of comparisons
for Step R1 the first time is \(cn\). The number for Step R2 the first time is \(n\). The total
for the first two steps is therefore \(n(1 + c)\) comparisons.

The second time we have a list half the size. So our work is half that of the original. The
number of comparisons for Step R1 the second time is \(cn/2\). The number for Step R2
the second time is \(n/2\). The total for the first two steps is \(n(1 + c)/2\) comparisons.

Hence the total number of comparisons is
\[
\frac{n(1 + c)}{2} + \frac{n(1 + c)}{4} + \frac{n(1 + c)}{8} + \frac{n(1 + c)}{16} + \frac{n(1 + c)}{32} + \ldots
\]

If this sum continues forever, by formula for geometric series it would add up to
\[2n(1 + c).\]

So the total work is at most \(2(1 + c)n\). This is linear! (The beauty of the geometric
series.)

**A5.3 The Pseudo-Median**

We still don’t have a median algorithm. There are two ideas needed:
(1) that the above algorithm is still efficient if one uses only an *approximation* to the
median; and
(2) one can actually find an approximation to the median.

We define a *pseudo-median* as any value such that at least 30% of the values lie above
the pseudo-median and at least 30% of the values lie below the pseudo-median. (The
value 30% is chosen to make the presentation easier.)

Suppose we have an algorithm that finds a pseudo-median. Then we can use it for rank
selection in exactly the same way as above. (And hence we can find the median!)
\textbf{RankSelect} (A:valuelist, k:integer)

- \textbf{R1}: Calculate a pseudo-median \( m \) of the list \( A \).
- \textbf{R2}: Determine the set \( A_\lessdot \) of elements smaller than \( m \).
  - Determine the set \( A_\gtrdot \) of elements larger than \( m \).
  - Calculate the rank \( r \) of \( m \) in \( A \).
- \textbf{R3}: If \( k < r \) then return \text{RankSelect}(A_\lessdot, k)
  - If \( k > r \) then return \text{RankSelect}(A_\gtrdot, k - r)
  - Else return \( m \)

Hence we have a function \text{RankSelect} which uses as a subroutine the function \text{Pseudo-Median}.

\subsection*{A5.4 Finding a Pseudo-Median}

How do we find a pseudo-median? What we do is:

\emph{take a sample of the data and find the median of the sample.}

This algorithm is bizarre but beautiful: we now have a function \text{Pseudo-Median} which uses as a subroutine the function \text{RankSelect}.

\textbf{Pseudo-Median} (B:valuelist)

- \textbf{P1}: Break up the list \( B \) into quintets (groups of 5).
- \textbf{P2}: Calculate the median in each quintet (by brute-force). Call the result the \textbf{representative} of the quintet. There are \( n/5 \) representatives.
- \textbf{P3}: Return the median of the set of representatives:
  - namely \text{RankSelect}(reps, \( n/10 \))

Does this produce a pseudo-median?

Well, consider the resultant value \( m_r \) which is the median of the representatives. This value is bigger than \( n/10 \) representatives. And for each representative that \( m_r \) is bigger than, \( m_r \) is bigger than two other members of the quintet. Thus there are at least \( 3n/10 \) values which \( m_r \) is guaranteed to be bigger than. And similarly there are \( 3n/10 \) values which \( m_r \) is guaranteed to be smaller than. Hence \( m_r \) is guaranteed to be a pseudo-median.
Note the intricate recursion: RankSelect calls PseudoMedian which calls RankSelect and so on. The reason why this works is that the problem size is getting smaller at each stage. (When implementing this algorithm, one must be very careful not to get an infinite loop.)

A5.5 Analysis

The calculation of the median in a quintet takes at most 10 comparisons. (Since sorting the quintet takes this many comparisons!) So the total number of comparisons for Step P2 is $2n$.

Now, let us return to the RankSelect algorithm modified to use PseudoMedian. Let $g(n)$ be the number of comparisons needed by RankSelect on a list of size $n$ in the worst case. Then the number of comparisons for the three steps are:

Step R1: $2n + g(n/5)$
Step R2: $n$
Step R3: $g(7n/10)$

This gives us the recurrence:

$$g(n) = g(n/5) + g(7n/10) + 3n$$

which has solution $g(n) = 30n$. (Plug it in and check!)

Exercises

1. Implement the median-finding algorithm.

2. Run your program on random lists of length $10^i$ for $i = 0, 1, \ldots$ Comment on the results.

3. Why quintets? Use either theory or a modification of your code to compare triplets with quintets.