Median Selection

**Definition of median**

Let \( A = a_1 \cdots a_n \) be a sequence of integers.

The median of \( A \) is a value \( v \) such that

\[
|\{ i : a_i < v \}| \leq n/2
\]

and

\[
|\{ i : a_i > v \}| \leq n/2
\]

That is, if \( b_1 \cdots b_n \) is \( A \) sorted in ascending order, then the median is \( b_{n/2} \).

**Algorithmic problem**

We want to compute the median using only comparisons.

More generally, given \( k \) we would like to find a value \( a \) such that

\[
|\{ i : a_i < v \}| \leq k
\]

and

\[
|\{ i : a_i > v \}| \leq n - k
\]

**A \( O(n \log n) \) solution**

- Sort \( A \) and return the value in the \( \lfloor n/2 \rfloor \)-th (respectively, \( k \)-th) position.

This requires time \( O(n \log n) \).

**A procedure inspired by quicksort**

Assume elements are distinct for the moment.

ChoosePivot() is a procedure that decides which value to partition around.

Partition() does the partition and returns the index where the pivot has been placed.

```plaintext
Select (A[1], \ldots, A[n], k)
begin
  v \leftarrow \text{ChoosePivot} (A[1], \ldots, A[n]);
  i \leftarrow \text{Partition} (A[1], \ldots, A[n], v);
  if i = k then return v
  else if i < k then Select(A[1], \ldots, A[i], k)
  else Select(A[i + 1], \ldots, A[n], k - i)
end
```
Remember Quicksort

QuickSort\( (A[1], \ldots, A[n]) \)
begin
  if\( n = 1 \) then halt;
  \( v \leftarrow \text{ChoosePivot}(A[1], \ldots, n) \);
  \( i \leftarrow \text{Partition}(A[1], \ldots, n, v) \);
  QuickSort\( (A[1], \ldots, A[i - 1]) \);
  QuickSort\( (A[i + 1], \ldots, A[n]) \)
end

Implementing ChoosePivot()

• Choose always the first element.
  There can be cases where the selection procedure takes \( O(n^2) \)
  time. Similar problem with Quicksort. Like for Quicksort, the
  average case is better.

• Choose a random element in the array.
  Average time for Select is \( O(n) \). Average time for QuickSort
  is \( O(n \log n) \). Will do analysis next time.

The median of medians

Divide the vector into \( n/5 \) subsequences of 5 consecutive
elements each.

Find the median in each sequence. Let \( m_1, \ldots, m_{n/5} \) be these
medians. Find recursively the median of these medians, let it
be \( mm \). This will be the pivot.

Implementing Partition in \( O(n) \) Time

Partition\( (A[1], \ldots, A[n], v) \)
begin
  \( i \leftarrow 1; \ j \leftarrow n; \)
  while true do begin
    repeat \( (i \leftarrow i + 1) \) until \( A[i] \geq v \);
    repeat \( (j \leftarrow j - 1) \) until \( A[j] \leq v \);
    if \( (i < j) \) then swap \( A[i] \) and \( A[j] \)
    else return \( i \)
  end
end

• Choose an element that is guaranteed to be bigger than
  \( \geq 30\% \) of the elements and smaller than \( \geq 30\% \) of the
  elements.

Worst case Select \( O(n) \). Worst case QuickSort \( O(n \log n) \).
How to implement?

ChoosePivotBFPRT\( (A[1], \ldots, A[n]) \)
begin
  for \( i = 1 \) to \( \frac{n}{5} \) do
    let \( m_i \) be the median of \( A[5i - 4], A[5i - 3], \ldots, A[5i] \);
  \( mm = \text{Select}(m_1, \ldots, m_{n/5}, n/10) \);
  return \( mm \)
end
Analysis

Consider ChoosePivotBFPRT\((A[1], \ldots, A[n])\).

Call “intermediate medians” the values \(m_1, \ldots, m_n\).

There are \(n/10\) intermediate medians \(\leq mm\). For each one, there are two elements smaller than them. Thus there are \(.3n\) elements \(< mm\)

Likewise, there are \(.3n\) elements \(\geq mm\).

\[\text{Running time is} \quad T(n) \leq T\left(\lceil n/5 \rceil\right) + T(.7n + 6) + O(n)\]

that still solves to \(T(n) = O(n)\).

If there are repeated elements

We can reduce to the case of no repetitions by considering the median of the array \(a'_1 \cdots a'_n\), where \(a'_i = (n + 1)a_i + i\).

The order is preserved and there are no repetitions.

Alternatively, one has to refine the algorithm and the analysis (see CLR).

Taking into account \([\cdot] \text{ and } [\cdot]\)

In the general case, \(n\) may not be divisible by 5.

We have to solve \([n/5]\) median subproblems (the last one may involve less than 5 elements), and then find the median of these intermediate medians, which takes time \(T([n/5])\).

The median-of-medians is bigger than at least

\(3([1/2[n/5]] - 2) \geq .3n - 6\)

elements in the array; and smaller than at least that many ones.

Why 5?

In general, the recursion

\[T(n) \leq T(\alpha n) + T(\beta n) + cn , \quad T(1) = c'\]

solves to \(T(n) = O(n)\) if \(\alpha + \beta < 1\).

While a recursion

\[T(n) \leq T(\alpha n) + T(\beta n) + cn , \quad T(1) = c'\]

with \(\alpha + \beta \geq 1\) typically yields \(T(n) = \Omega(n \log n)\).
3 does not work

Dividing the array in groups of 3 elements, we would spend $T(n/3)$ time in finding the median-of-medians.

Then, even if the size of the vector is a multiple of 3, we can only guarantee that the median-of-medians is larger than $n/3$ elements and smaller than $n/3$.

So we may recurse to a sub-array with $n-2n/3$ elements. The recursion is

$$T(n) \leq T(n/3) + T(2n/3) + O(n)$$

No good!

Lower bounds

We need to make at least $n/2$ comparisons just to read all the elements.

More involved argument: we need $\geq n - 1$ comparisons.

Much more involved argument: we need $\geq 2n - o(n)$ comparisons. Bent and John (1985)

Exceedingly complicated: we need $\geq (2 + 2^{-30})n - o(n)$ comparisons, Dor and Zwick (1997).

Better (?) algorithms

The median-of-medians algorithm is by Blum, Floyd, Pratt, Rivest, Tarjan (1973).

An algorithm that makes $5n + o(n)$ comparisons is due Schönhage, Pippenger, Paterson (1976).

Same people, same year, an algorithm that makes $3n + o(n)$ comparisons, Schönhage, Pippenger, Paterson (1976).

Quite recently: an algorithm that makes $2.95n + o(n)$ comparisons, due to Dor and Zwick (1995).