Selection

CS 4231, Fall 2017

Mihalis Yannakakis

Selection (Order Statistics)

- Input: Set A of n numbers (or more generally, elements from an ordered domain), number i, $1 \le i \le n$
- Output: i-th smallest (rank i) element of A

- $i=1$: minimum
- $i=n$: maximum
- $i=(n+1)/2$: median

Straightforward solution

- Sort A
- Output i-th element
- $\Theta(n\log n)$

- Can we do better?

- Min, Max: $\Theta(n)$ easy

- General rank i ?

Divide and Conquer Selection of i-th smallest element

- Divide: Partition the input array A of elements with respect to a pivot element x into two parts:

$\leq x$	x	$> x$

- Conquer & Combine:

  Check if |left part| $\geq$ i , = i-1, or $<$ i-1,

  and accordingly recurse on left part,

  return x, or recurse on right part

Example

Select 6th smallest element in array

3	8	4	2	9	7	5

Partition the array w.r.t pivot 5

3	4	2	5	9	7	8

Select recursively the 2nd smallest element in right part:

9	7	8

Randomized Selection

- RANDOMIZED-SELECT(A,p,r,i)

  Input: array A, indices p≤r, number i≤r-p+1

  if p = r **then return** A[p]

  q = RANDOMIZED-PARTITION(A,p,r)

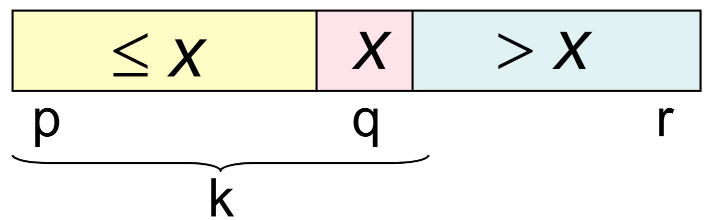
  k = q-p+1

      if i=k **then return** A[q]

      else if i<k

        then return RANDOMIZED-SELECT(A,p,q-1,i)

      else return RANDOMIZED-SELECT(A,q+1,r,i-k)



Main Call: RANDOMIZED-SELECT(A,1,n,i)

Analysis

- Worst Case: Unbalanced partition

  $T(n) = T(n-1)+\Theta(n) \Rightarrow T(n) = \Theta(n^2)$

- Lucky Case: Balanced partition ½ : ½

  $T(n) = T(n/2) + \Theta(n)$

  Master theorem: a=1, b=2, f(n)=n, $n^{\log_b a} = 1$

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

Expected Time Analysis

- Assume distinct elements
- Let T(n)=expected running time
- Partition (k, n-k-1) with probability 1/n

  for each k=0,…,n-1
- At worst we'll recurse on the larger part

$$
\begin{aligned}
T(n) &\le \sum_{k=0}^{n-1} \frac{1}{n} T(\max(k, n-k-1)) + \Theta(n) \\
&= \frac{1}{n} \sum_{k=0}^{\lfloor n/2 \rfloor - 1} T(n-k-1) + \frac{1}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} T(k) + \Theta(n) \\
&\le \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} T(k) + \Theta(n)
\end{aligned}
$$

Expected Time Analysis

$$T(n) \le \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} T(k) + an$$

Solution: $T(n) \le cn$

Proof:
(Induction step)

$$T(n) \le \frac{2c}{n} \cdot \sum_{k=\lfloor n/2 \rfloor}^{n-1} k + an$$

$$= \frac{2c}{n} \cdot \frac{(\lfloor n/2 \rfloor + n - 1)(n - \lfloor n/2 \rfloor)}{2} + an$$

$$\le \frac{2c}{n} \cdot \frac{(3n/2)(n/2)}{2} + an$$

$$\le (3c/4 + a)n$$

$$\le cn \quad \text{provided } 3c/4 + a \le c, \text{ i.e., } 4a \le c$$

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

Randomized Selection algorithm

- Practical, fast, in-place
- Linear expected time

  (if equal elements use 3-way partition)
- Quadratic worst case time

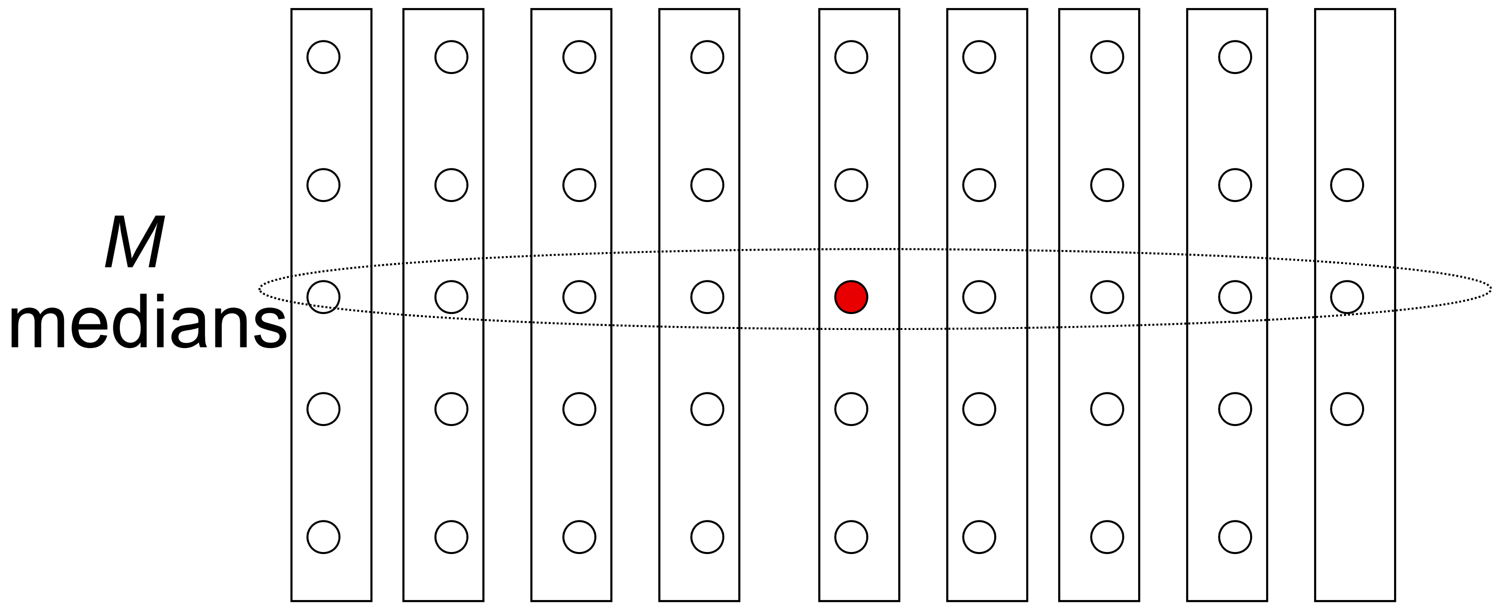
- Is there a deterministic algorithm that runs in worst case linear time?
- Yes

Deterministic Selection Algorithm

- Divide the given set A of elements into groups of 5
- Find the median in each group directly
  $\rightarrow$ subset M
- Recursively select the median *x* of M
- Partition A with respect to pivot *x*
- Recurse on the appropriate part

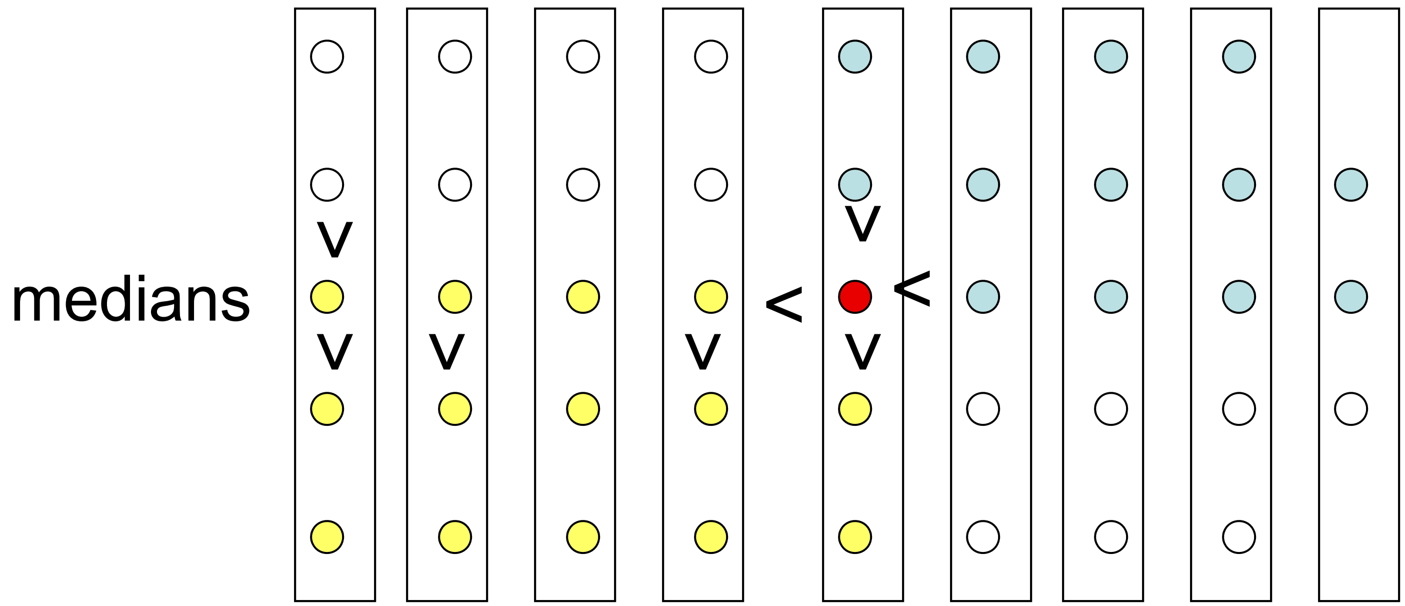
Key property: Partition wrt *x* not too unbalanced

Choice of pivot x



$\lceil n/5 \rceil$ Groups

Choice of pivot x



$\lceil n/5 \rceil$ Groups

At least 1/2 groups have median $\leq x$
from each such group (but one) 3 elements $\leq x$
$\Rightarrow$ at least $(3n/10)$-2 elements are $\leq x$

Similarly, at least $(3n/10)$-2 elements are $\geq x$

Analysis

$$T(n) \le T\left(\left\lceil \frac{n}{5} \right\rceil\right) + T\left(\frac{7n}{10} + 2\right) + \Theta(n)$$

Solution: $T(n) = O(n)$

Proof (by substitution): $T(n) \le cn$

$$\begin{aligned}
T(n) &\le T(\lceil n/5 \rceil) + T(7n/10 + 2) + \Theta(n) \\
&\le c(n/5 + 1) + c(7n/10 + 2) + an \\
&= n[c(1/5 + 7/10) + a] + 3c \\
&\le cn
\end{aligned}$$

provided $c(1/5 + 7/10) + a < c$, i.e., $10a < c$

and $n(c/10 - a) \ge 3c$, i.e., $n \ge 3c / (c/10 - a)$

For example, take $n_0 = 60$, $c = \max(20a, \{T(i)/i \mid i \le 60\})$

Deterministic vs. Randomized Selection

- Deterministic runs always in O(n) time
- But in practice, randomized algorithm is faster because hidden constant of deterministic algorithm in O(n) is large