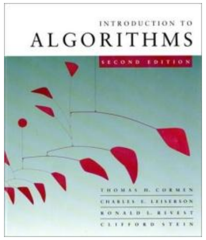
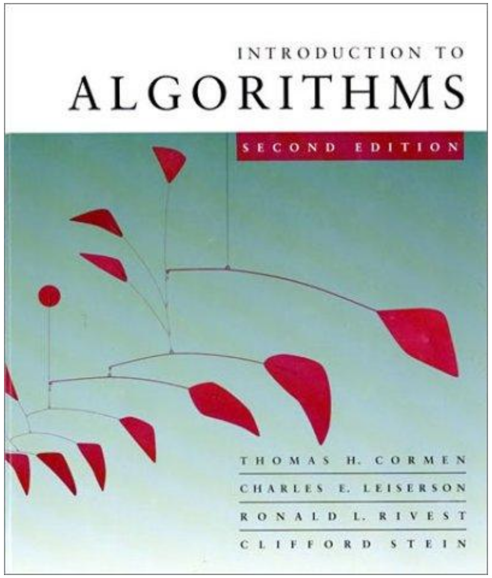


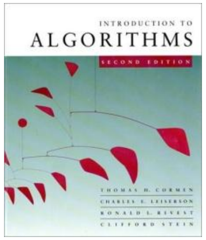
CS 5633 – Spring 2012



Order Statistics

Carola Wenk

Slides courtesy of Charles Leiserson with small changes by Carola Wenk



Order statistics

Select the ith smallest of n elements (the element with ***rank*** i).

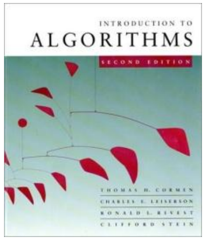
- $i = 1$: ***minimum***;
- $i = n$: ***maximum***;
- $i = \lfloor (n+1)/2 \rfloor$ or $\lceil (n+1)/2 \rceil$: ***median***.

Naive algorithm: Sort and index ith element.

Worst-case running time $= \Theta(n \log n + 1)$

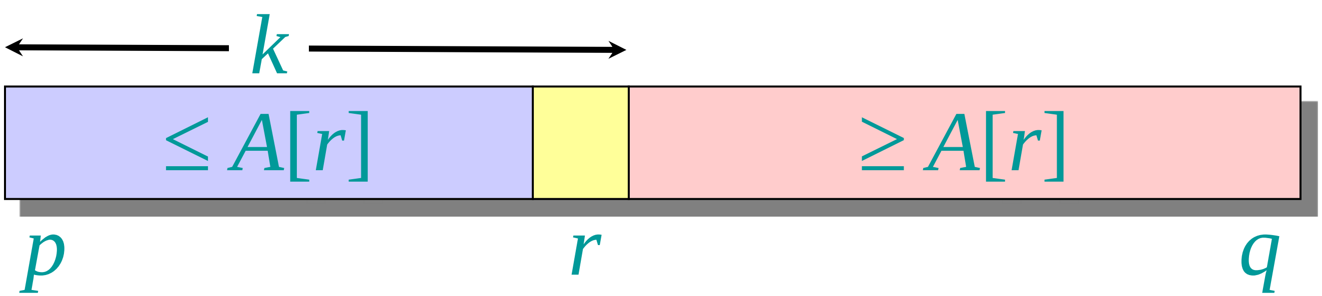
$= \Theta(n \log n)$,

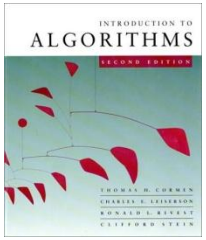
using merge sort or heapsort (*not* quicksort).



Randomized divide-and-conquer algorithm

RAND-SELECT(A, p, q, i) ⊳ i-th smallest of $A[p \,..\, q]$
  if $p = q$ **then return** $A[p]$
  $r \leftarrow$ RAND-PARTITION(A, p, q)
  $k \leftarrow r - p + 1$ ⊳ $k = \text{rank}(A[r])$
  if $i = k$ **then return** $A[r]$
  if $i < k$
    then return RAND-SELECT$(A, p, r - 1, i)$
    else return RAND-SELECT$(A, r + 1, q, i - k)$





Example

Select the $i = 7$th smallest:

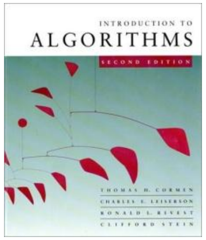
6	10	13	5	8	3	2	11	$i = 7$

pivot

Partition:

2	5	3	6	8	13	10	11	$k = 4$

Select the $7 - 4 = 3$rd smallest recursively.



Intuition for analysis

(All our analyses today assume that all elements are distinct.)

for RAND-PARTITION

Lucky:

$T(n) = T(9n/10) + dn$

$= \Theta(n)$

$n^{\log_{10/9} 1} = n^0 = 1$

CASE 3

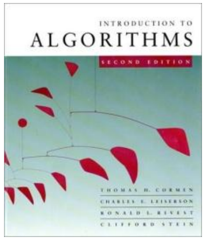
Unlucky:

$T(n) = T(n-1) + dn$

$= \Theta(n^2)$

arithmetic series

Worse than sorting!



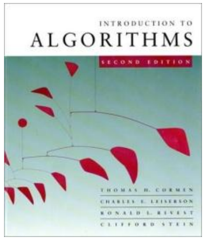
Analysis of expected time

The analysis follows that of randomized quicksort, but it's a little different.

Let $T(n)$ = the random variable for the running time of RAND-SELECT on an input of size n, assuming random numbers are independent.

For $k = 0, 1, \ldots, n–1$, define the ***indicator random variable***

$$X_k = \begin{cases} 1 & \text{if PARTITION generates a } k : n-k-1 \text{ split,} \\ 0 & \text{otherwise.} \end{cases}$$



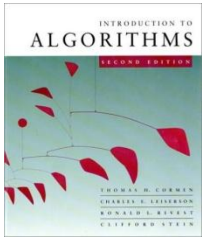
Analysis (continued)

To obtain an upper bound, assume that the i th element always falls in the larger side of the partition:

$$
T(n) = \begin{cases} T(\max\{0, n-1\}) + dn & \text{if } 0 : n-1 \text{ split,} \\ T(\max\{1, n-2\}) + dn & \text{if } 1 : n-2 \text{ split,} \\ \quad\vdots & \\ T(\max\{n-1, 0\}) + dn & \text{if } n-1 : 0 \text{ split,} \end{cases}
$$

$$
= \sum_{k=0}^{n-1} X_k \left(T(\max\{k, n-k-1\}) + dn \right) .
$$

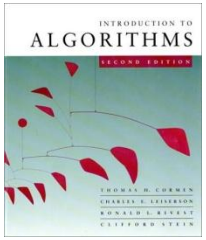
$$
\le 2 \sum_{k=\lfloor n/2 \rfloor}^{n-1} X_k \left(T(k) + dn \right)
$$



Calculating expectation

$$E[T(n)] = E\left[2 \sum_{k=\lfloor n/2 \rfloor}^{n-1} X_k \left(T(k) + dn \right) \right]$$

Take expectations of both sides.

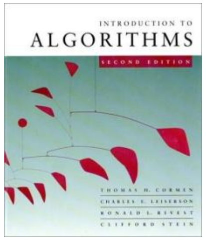


Calculating expectation

$$E[T(n)] = E\left[2 \sum_{k=\lfloor n/2 \rfloor}^{n-1} X_k \left(T(k) + dn \right) \right]$$

$$= 2 \sum_{k=\lfloor n/2 \rfloor}^{n-1} E\left[X_k \left(T(k) + dn \right) \right]$$

Linearity of expectation.



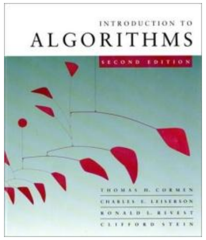
Calculating expectation

$$E[T(n)] = E\left[2\sum_{k=\lfloor n/2 \rfloor}^{n-1} X_k\left(T(k)+dn\right)\right]$$

$$= 2\sum_{k=\lfloor n/2 \rfloor}^{n-1} E\left[X_k\left(T(k)+dn\right)\right]$$

$$= 2\sum_{k=\lfloor n/2 \rfloor}^{n-1} E\left[X_k\right] \cdot E\left[T(k)+dn\right]$$

Independence of X_k from other random choices.



Calculating expectation

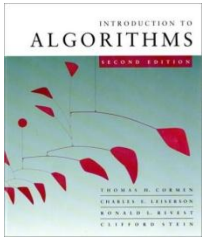
$$E[T(n)] = E\left[2 \sum_{k=\lfloor n/2 \rfloor}^{n-1} X_k \left(T(k) + dn \right) \right]$$

$$= 2 \sum_{k=\lfloor n/2 \rfloor}^{n-1} E\left[X_k \left(T(k) + dn \right) \right]$$

$$= 2 \sum_{k=\lfloor n/2 \rfloor}^{n-1} E\left[X_k \right] \cdot E\left[T(k) + dn \right]$$

$$= \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} E\left[T(k) \right] + \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} dn$$

Linearity of expectation; $E[X_k] = 1/n$.



Calculating expectation

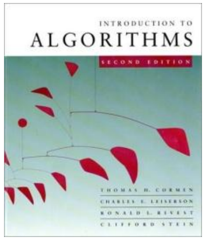
$$E[T(n)] = E\left[2 \sum_{k=\lfloor n/2 \rfloor}^{n-1} X_k \left(T(k) + dn \right) \right]$$

$$= 2 \sum_{k=\lfloor n/2 \rfloor}^{n-1} E\left[X_k \left(T(k) + dn \right) \right]$$

$$= 2 \sum_{k=\lfloor n/2 \rfloor}^{n-1} E[X_k] \cdot E[T(k) + dn]$$

$$= \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} E[T(k)] + \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} dn$$

$$= \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} E[T(k)] + dn$$



Hairy recurrence

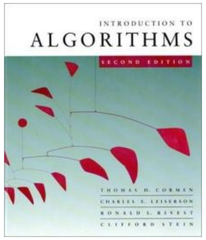
(But not quite as hairy as the quicksort one.)

$$E[T(n)] = \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} E[T(k)] + dn$$

Prove: $E[T(n)] \le cn$ for constant $c > 0$.

- The constant c can be chosen large enough so that $E[T(n)] \le cn$ for the base cases.

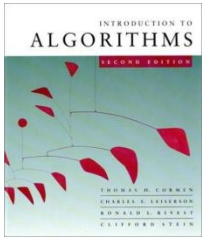
Use fact: $\sum_{k=\lfloor n/2 \rfloor}^{n-1} k \le \frac{3}{8} n^2$ (exercise).



Substitution method

$$E\left[T(n)\right] \le \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} ck + dn$$

Substitute inductive hypothesis.

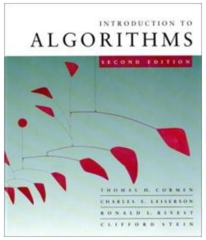


Substitution method

$$E[T(n)] \le \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} ck + dn$$

$$\le \frac{2c}{n}\left(\frac{3}{8}n^2\right) + dn$$

Use fact.



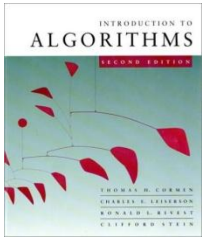
Substitution method

$$E[T(n)] \le \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} ck + dn$$

$$\le \frac{2c}{n}\left(\frac{3}{8}n^2\right) + dn$$

$$= cn - \left(\frac{cn}{4} - dn\right)$$

Express as ***desired – residual***.



Substitution method

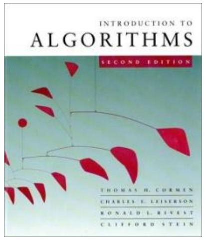
$$E[T(n)] \le \frac{2}{n}\sum_{k=\lfloor n/2 \rfloor}^{n-1} ck + dn$$

$$\le \frac{2c}{n}\left(\frac{3}{8}n^2\right) + dn$$

$$= cn - \left(\frac{cn}{4} - dn\right)$$

$$\le cn,$$

if $c \ge 4d$.



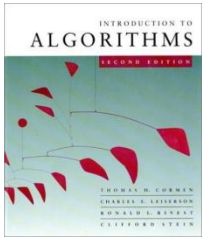
Summary of randomized order-statistic selection

- Works fast: linear expected time.
- Excellent algorithm in practice.
- But, the worst case is ***very*** bad: $\Theta(n^2)$.

Q. Is there an algorithm that runs in linear time in the worst case?

A. Yes, due to Blum, Floyd, Pratt, Rivest, and Tarjan [1973].

IDEA: Generate a good pivot recursively.

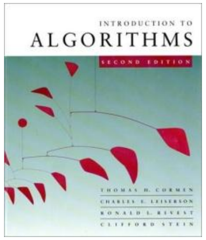


Worst-case linear-time order statistics

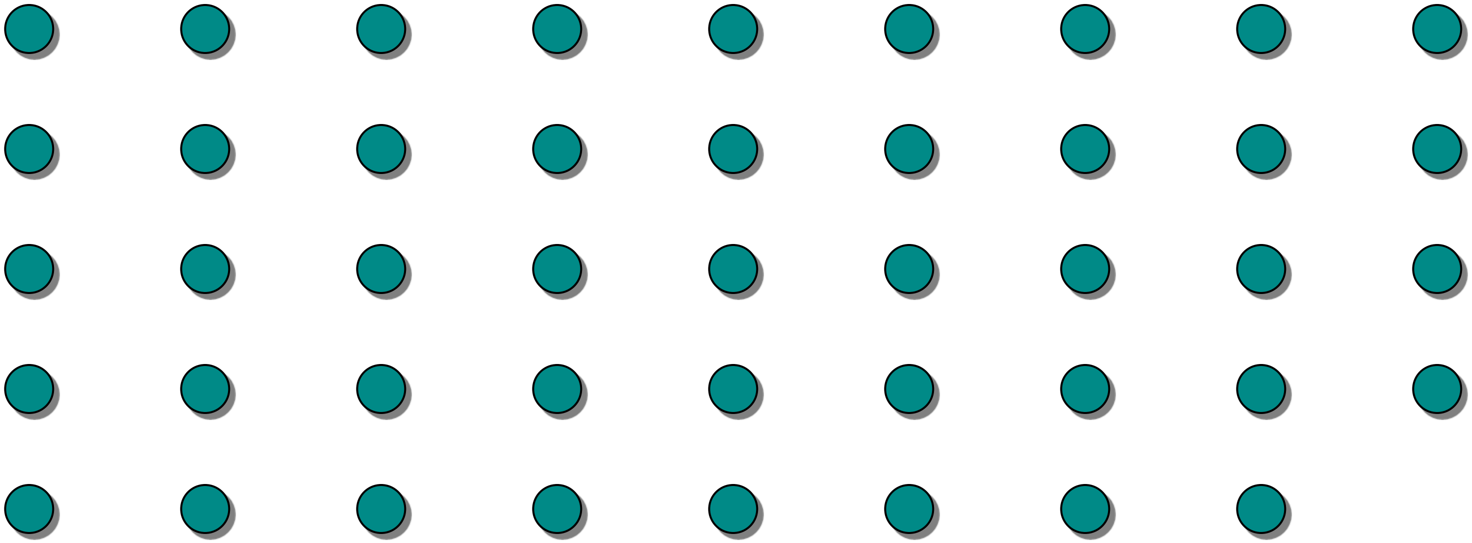
SELECT(i, n)

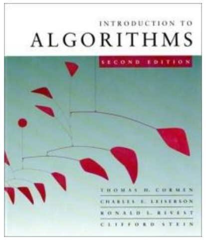
1. Divide the n elements into groups of 5. Find the median of each 5-element group by rote.
2. Recursively SELECT the median x of the $\lfloor n/5 \rfloor$ group medians to be the pivot.
3. Partition around the pivot x. Let $k = \text{rank}(x)$.
4. **if** $i = k$ **then return** x
   elseif $i < k$
   then recursively SELECT the ith smallest element in the lower part
   else recursively SELECT the $(i–k)$th smallest element in the upper part

Same as RAND-SELECT (steps 3–4)

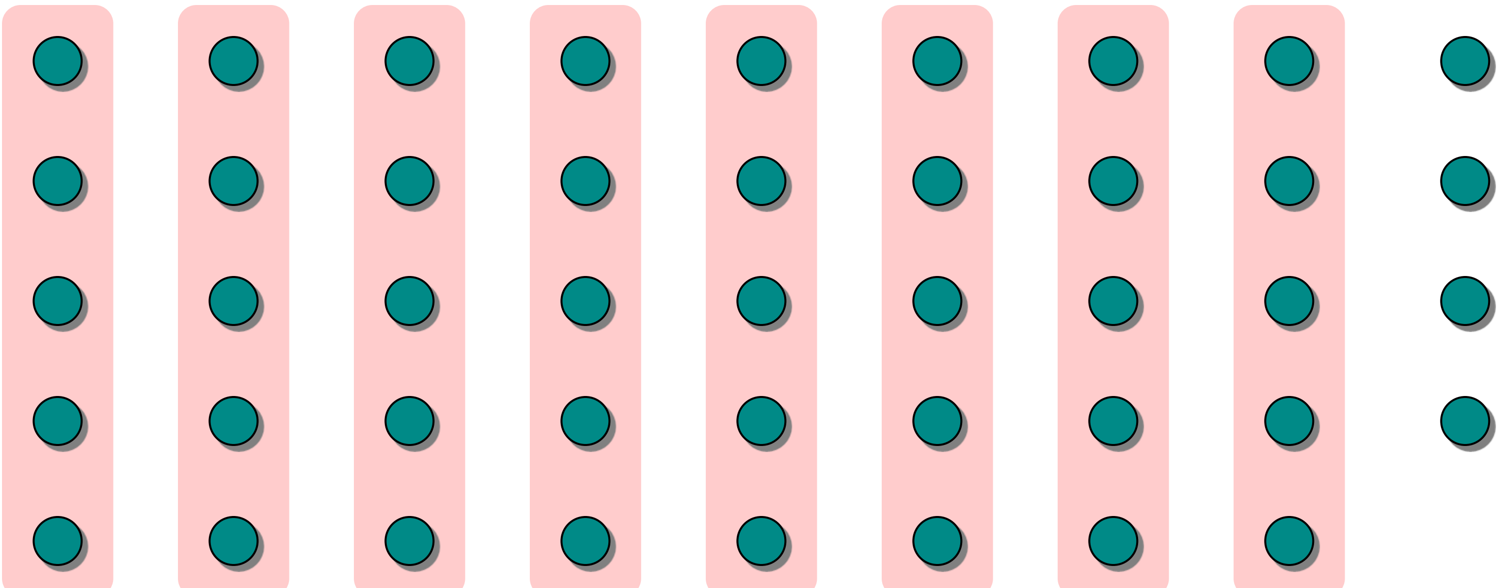


Choosing the pivot

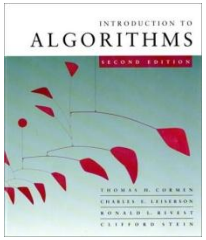




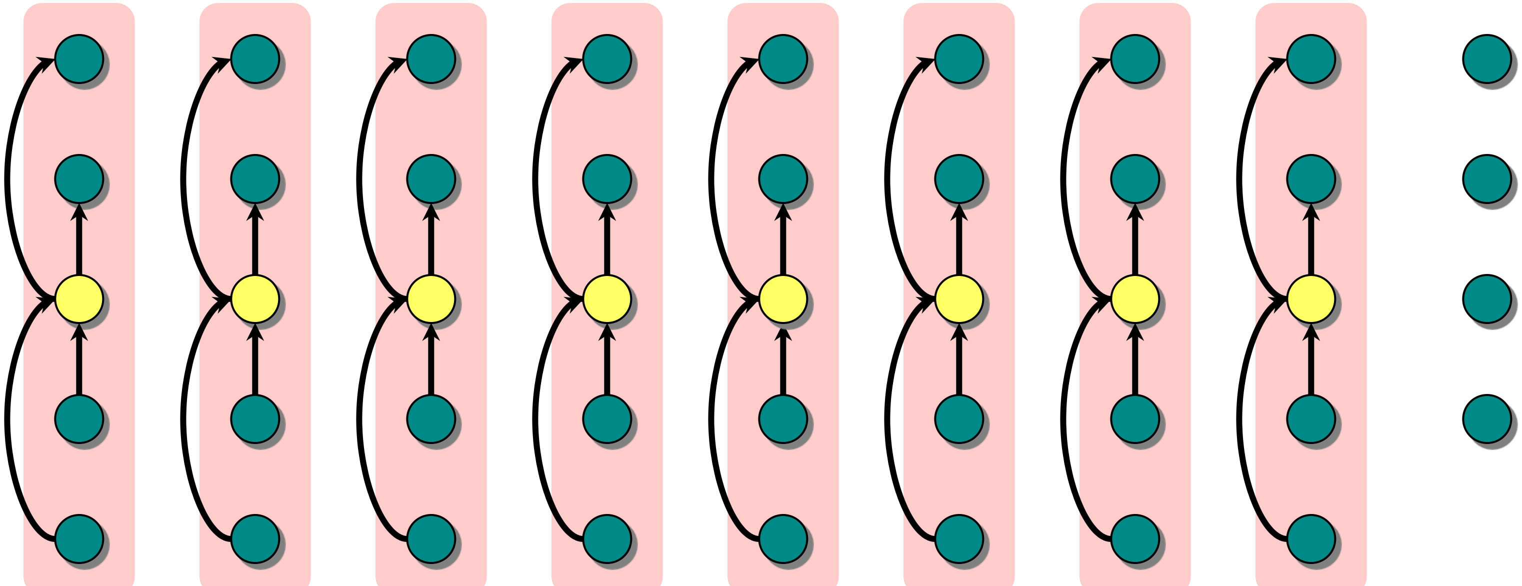
Choosing the pivot



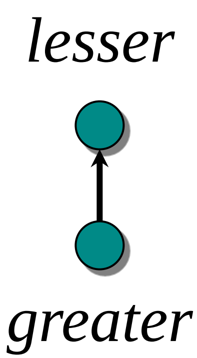
1. Divide the n elements into groups of 5.

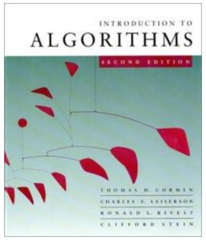


Choosing the pivot

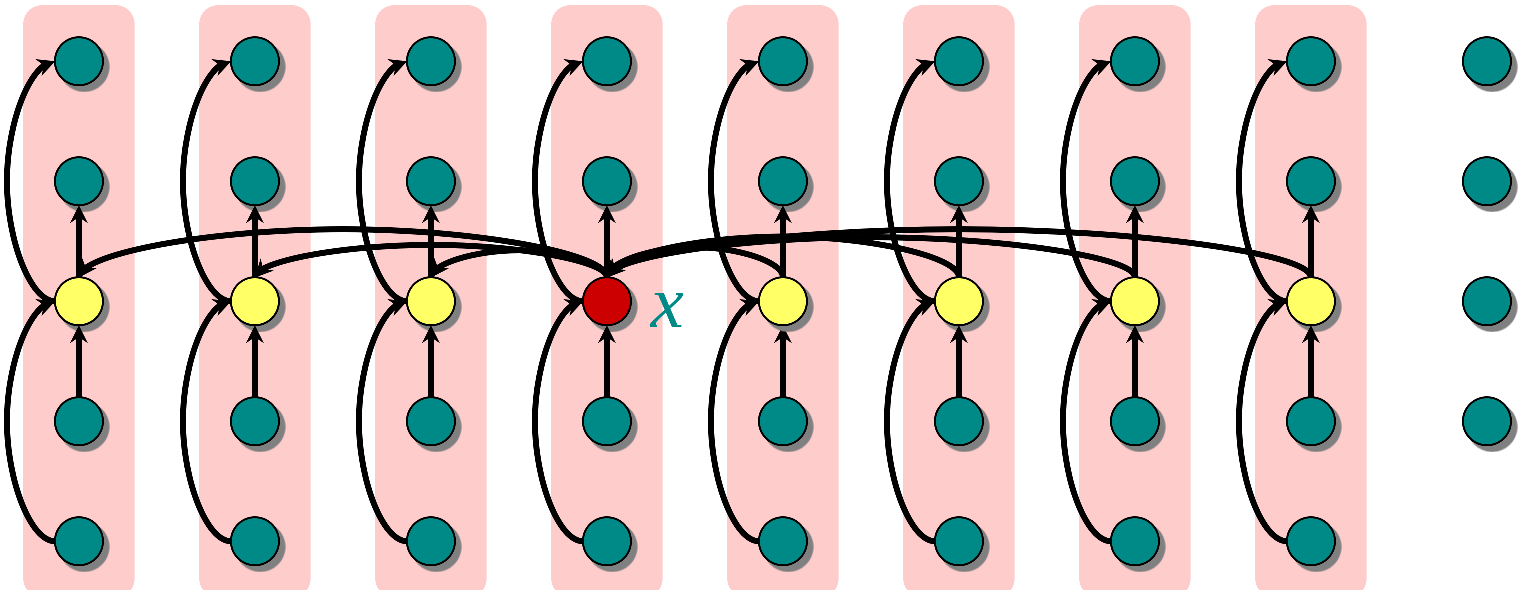


1. Divide the n elements into groups of 5. Find the median of each 5-element group by rote.

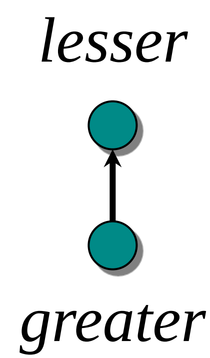


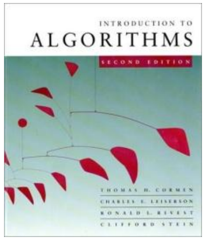


Choosing the pivot



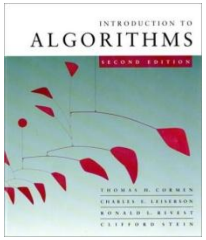
1. Divide the n elements into groups of 5. Find the median of each 5-element group by rote.
2. Recursively SELECT the median x of the $\lfloor n/5 \rfloor$ group medians to be the pivot.



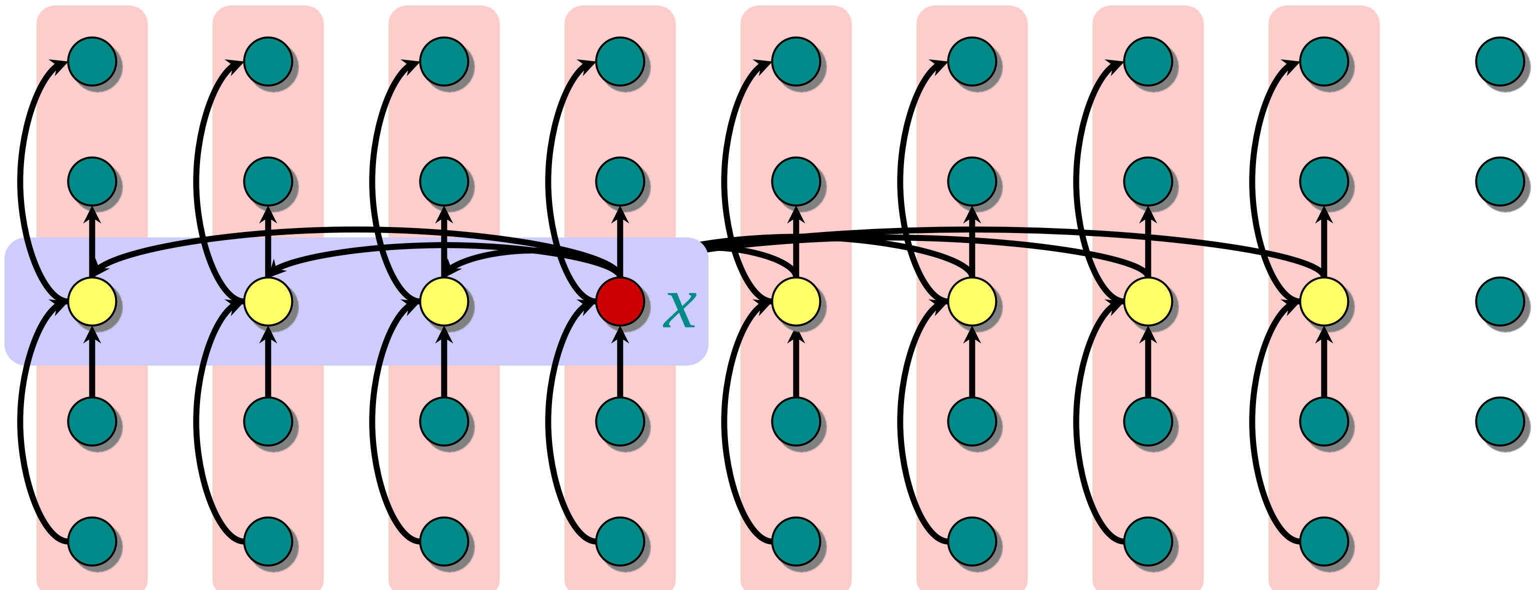


Developing the recurrence

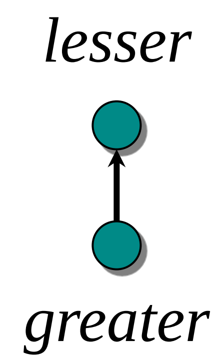
$T(n)$	SELECT(i, n)
$\Theta(n)$	1. Divide the n elements into groups of 5. Find the median of each 5-element group by rote.
$T(n/5)$	2. Recursively SELECT the median x of the $\lfloor n/5 \rfloor$ group medians to be the pivot.
$\Theta(n)$	3. Partition around the pivot x. Let $k = \text{rank}(x)$.
$T(\ ?\ )$	4. **if** $i = k$ **then return** x **elseif** $i < k$ **then** recursively SELECT the ith smallest element in the lower part **else** recursively SELECT the $(i–k)$th smallest element in the upper part

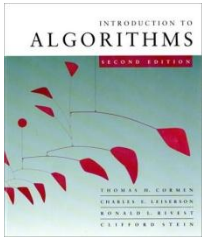


Analysis (Assume all elements are distinct.)

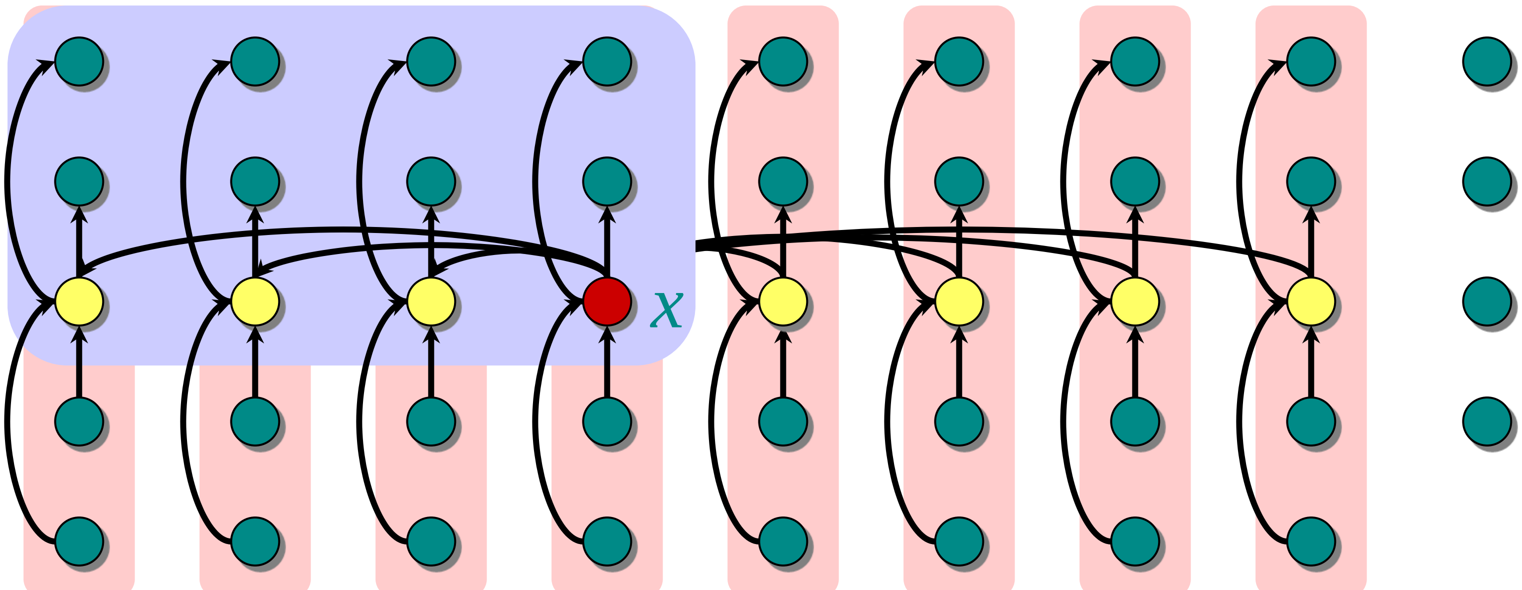


At least half the group medians are $\le x$, which is at least $\lfloor \lfloor n/5 \rfloor /2 \rfloor = \lfloor n/10 \rfloor$ group medians.



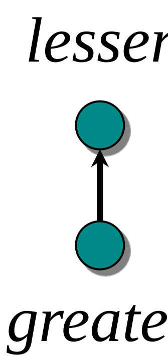


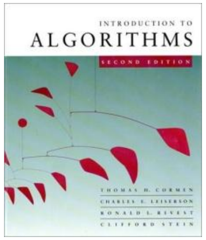
Analysis (Assume all elements are distinct.)



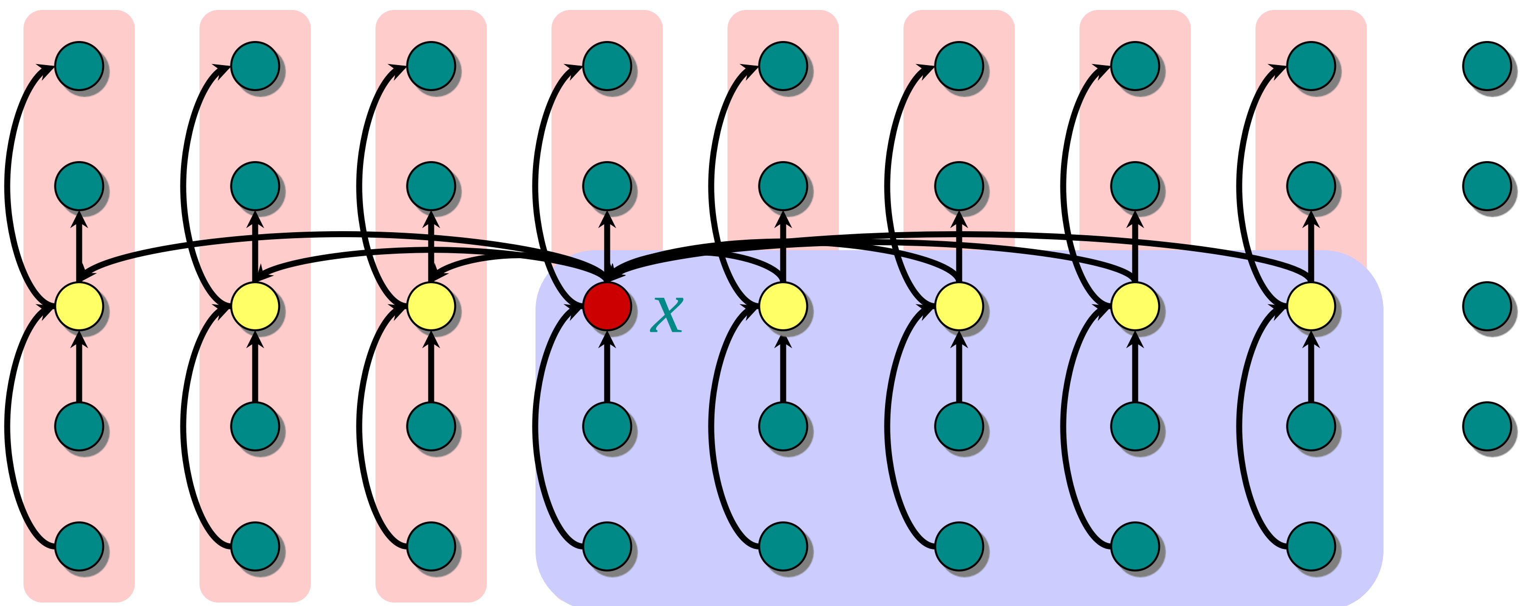
At least half the group medians are $\le x$, which is at least $\lfloor \lfloor n/5 \rfloor /2 \rfloor = \lfloor n/10 \rfloor$ group medians.

- Therefore, at least $3\lfloor n/10 \rfloor$ elements are $\le x$.



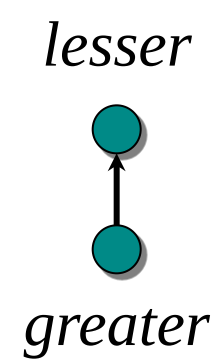


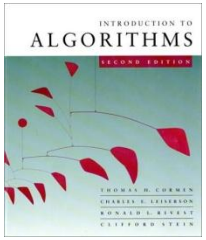
Analysis (Assume all elements are distinct.)



At least half the group medians are $\le x$, which is at least $\lfloor \lfloor n/5 \rfloor /2 \rfloor = \lfloor n/10 \rfloor$ group medians.

- Therefore, at least $3\lfloor n/10 \rfloor$ elements are $\le x$.
- Similarly, at least $3\lfloor n/10 \rfloor$ elements are $\ge x$.

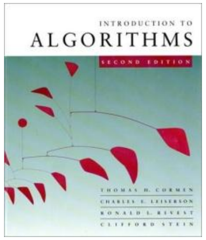




Analysis (Assume all elements are distinct.)

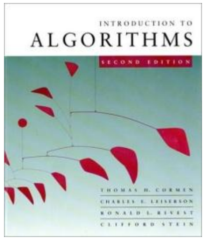
Need "at most" for worst-case runtime

- At least $3\lfloor n/10 \rfloor$ elements are $\leq x$
  $\Rightarrow$ at most $n\text{-}3\lfloor n/10 \rfloor$ elements are $\geq x$
- At least $3\lfloor n/10 \rfloor$ elements are $\geq x$
  $\Rightarrow$ at most $n\text{-}3\lfloor n/10 \rfloor$ elements are $\leq x$
- The recursive call to SELECT in Step 4 is executed recursively on $n\text{-}3\lfloor n/10 \rfloor$ elements.



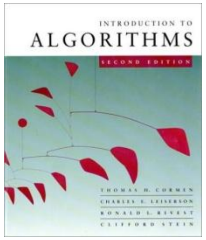
Analysis (Assume all elements are distinct.)

- Use fact that $\lfloor a/b \rfloor \geq ((a-(b-1))/b$ (page 51)
- $n-3\lfloor n/10 \rfloor \leq n-3\cdot(n-9)/10 = (10n - 3n + 27)/10$ $\leq 7n/10 + 3$
- The recursive call to SELECT in Step 4 is executed recursively on at most $7n/10+3$ elements.



Developing the recurrence

Cost	Step
$T(n)$	SELECT(i, n)
$\Theta(n)$	1. Divide the n elements into groups of 5. Find the median of each 5-element group by rote.
$T(n/5)$	2. Recursively SELECT the median x of the $\lfloor n/5 \rfloor$ group medians to be the pivot.
$\Theta(n)$	3. Partition around the pivot x. Let $k = \text{rank}(x)$.
$T(7n/10 +3)$	4. **if** $i = k$ **then return** x **elseif** $i < k$ **then** recursively SELECT the ith smallest element in the lower part **else** recursively SELECT the $(i–k)$th smallest element in the upper part



Solving the recurrence

$$T(n) = T\left(\frac{1}{5}n\right) + T\left(\frac{7}{10}n + 3\right) + dn$$

for $\Theta(n)$

Substitution: $T(n) \le c(\frac{1}{5}n - 3) + c(\frac{7}{10}n + 3 - 3) + dn$

$T(n) \le c(n - 3)$

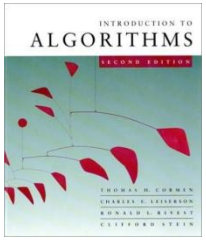
Technical trick. This shows that $T(n) \in O(n)$

$$\le \frac{9}{10}cn - 3c + dn$$

$$= c(n-3) - \frac{1}{10}cn + dn$$

$$\le c(n-3) ,$$

if c is chosen large enough, e.g., $c=10d$



Conclusions

- Since the work at each level of recursion is basically a constant fraction (9/10) smaller, the work per level is a geometric series dominated by the linear work at the root.
- In practice, this algorithm runs slowly, because the constant in front of n is large.
- The randomized algorithm is far more practical.

Exercise: *Try to divide into groups of 3 or 7.*