

Design and Analysis  
of Algorithms I

# Linear-Time Selection

---

## Randomized Selection (Analysis)

# Running Time of RSelect

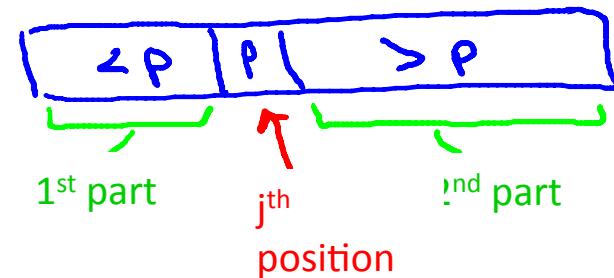
Rselect Theorem : for every input array of length  $n$ , the average running time of Rselect is  $O(n)$

- holds for every input [no assumptions on data]
- “average” is over random pivot choices made by the algorithm

# Randomized Selection

Rselect (array A, length n, order statistic i)

- 0) if  $n = 1$  return  $A[1]$
- 1) Choose pivot  $p$  from  $A$  uniformly at random
- 2) Partition  $A$  around  $p$   
let  $j$  = new index of  $p$
- 3) If  $j = i$ , return  $p$
- 4) If  $j > i$ , return Rselect( $1^{\text{st}}$  part of  $A$ ,  $j-1$ ,  $i$ )
- 5) [if  $j < i$ ] return Rselect ( $2^{\text{nd}}$  part of  $A$ ,  $n-j$ ,  $i-j$ )



# Proof I: Tracking Progress via Phases

Note : Rselect uses  $\leq cn$  operations outside of recursive call [ for some constant  $c > 0$  ] [from partitioning]

Notation : Rselect is in phase  $j$  if current array size between  $(\frac{3}{4})^{j+1} \cdot n$  and  $(\frac{3}{4})^j \cdot n$

- $X_j$  = number of recursive calls during phase  $j$

$$\text{Note : running time of RSelect} \leq \sum_{\text{phases } j} X_j \cdot c \cdot (\frac{3}{4})^j \cdot n$$

# of phase  $j$  subproblems

<= array size during phase  $j$

Work per phase  $j$  subproblem

Tim Roughgarden

# Proof II: Reduction to Coin Flipping

$X_j = \# \text{ of recursive calls during phase } j$  → Size between  $(\frac{3}{4})^{j+1} \cdot n$  and  $(\frac{3}{4})^j \cdot n$

Note : if Rselect chooses a pivot giving a 25 – 75 split (or better) then current phase ends !  
(new subarray length at most 75 % of old length)



Recall : probability of 25-75 split or better is 50%

So :  $E[X_j] \leq$  expected number of times you need to flip a fair coin  
to get one “heads”  
(heads ~ good pivot, tails ~ bad pivot)

Tim Roughgarden

# Proof III: Coin Flipping Analysis

Let  $N$  = number of coin flips until you get heads.  
( a “geometric random variable” )

Note :  $E[N] = 1 + (1/2)*E[N]$

1<sup>st</sup> coin flip      Probability of tails      # of further coin flips needed in this case

Solution :  $E[N] = 2$       (Recall  $E[X_j] \leq E[N]$ )

# Putting It All Together

Expected  
running time of  
RSelect

$$\leq E[cn \sum_{\text{phase } j} \left(\frac{3}{4}\right)^j X_j] \quad (*)$$

$$= cn \sum_{\text{phase } j} \left(\frac{3}{4}\right)^j E[X_j] \quad [\text{LIN EXP}]$$

$$= E[\# \text{ of coin flips } N] = 2$$

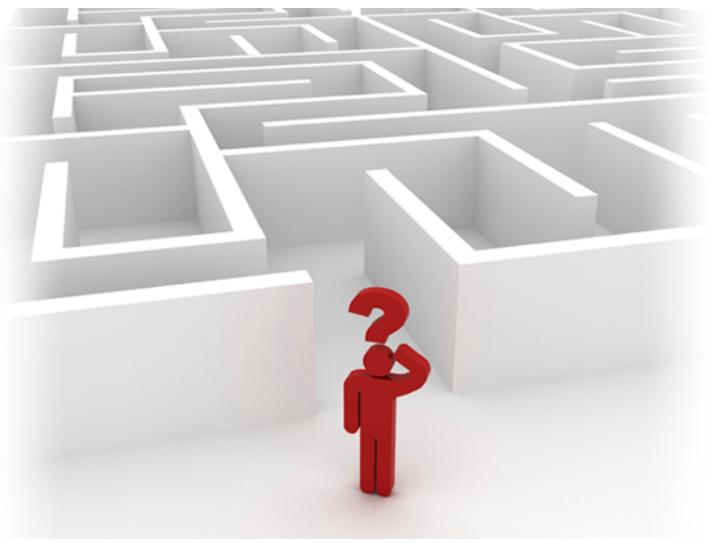
$$\leq 2cn \sum_{\text{phase } j} \left(\frac{3}{4}\right)^j$$

geometric sum,  
 $\leq 1/(1-3/4) = 4$

$$\leq 8cn = O(n)$$

**Q.E.D.**

Tim Roughgarden



Design and Analysis  
of Algorithms I

# Linear-Time Selection

---

## Randomized Selection (Algorithm)

# Prerequisites

Watch this after:

- QuickSort - Partitioning around a pivot
- QuickSort – Choosing a good pivot
- Probability Review, Part I

# The Problem

Input : array A with n **distinct** numbers and a number

For simplicity

Output :  $i^{\text{th}}$  order statistic (i.e.,  $i^{\text{th}}$  smallest element of input array)

Example : median.

(  $i = (n+1)/2$  for n odd,  
 $i = n/2$  for n even )



3<sup>rd</sup> order statistic

Solve one problem using the solution of another problem

## Reduction to Sorting

O(nlog(n)) algorithm

- 1) Apply MergeSort
- 2) return  $i^{\text{th}}$  element of sorted array

Fact : can't sort any faster [ see optional video ]

Next :  $O(n)$  time (randomized) by modifying Quick Sort.

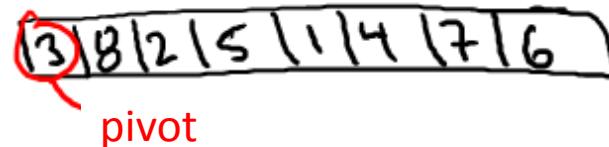
Optional Video :  $O(n)$  time deterministic algorithm.

-- pivot = “median of medians” (warning : not practical )

# Partitioning Around a Pivot

Key Idea : partition array around a pivot element.

-Pick element of array



-Rearrange array so that

- Left of pivot => less than pivot

- Right of pivot => greater than pivot



Note : puts pivot in its “rightful position”.

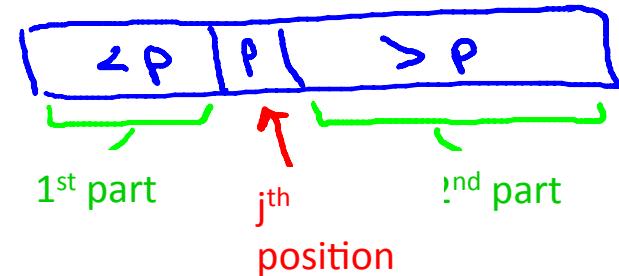
Suppose we are looking for the 5<sup>th</sup> order statistic in an input array of length 10. We partition the array, and the pivot winds up in the third position of the partitioned array. On which side of the pivot do we recurse, and what order statistic should we look for?

- The 3rd order statistic on the left side of the pivot.
- The 2nd order statistic on the right side of the pivot.
- The 5th order statistic on the right side of the pivot.
- Not enough information to answer question – we might need to recurse on the left or the right side of the pivot.

# Randomized Selection

Rselect (array A, length n, order statistic i)

- 0) if  $n = 1$  return  $A[1]$
- 1) Choose pivot  $p$  from  $A$  uniformly at random
- 2) Partition  $A$  around  $p$   
let  $j$  = new index of  $p$
- 3) If  $j = i$ , return  $p$
- 4) If  $j > i$ , return Rselect( $1^{\text{st}}$  part of  $A$ ,  $j-1$ ,  $i$ )
- 5) [if  $j < i$ ] return Rselect ( $2^{\text{nd}}$  part of  $A$ ,  $n-j$ ,  $i-j$ )



# Properties of RSelect

Claim : Rselect is correct (guaranteed to output  $i$ th order statistic)

Proof : by induction. [like in optional QuickSort video]

Running Time ? : depends on “quality” of the chosen pivots.

What is the running time of the RSelect algorithm if pivots are always chosen in the worst possible way?

- $\theta(n)$
- $\theta(n \log n)$
- $\theta(n^2)$
- $\theta(2^n)$

Example :

-- suppose  $i = n/2$   
-- suppose choose pivot = minimum  
every time  
 $\Rightarrow \Omega(n)$  time in each of  $\Omega(n)$  recursive calls

# Running Time of RSelect?

Running Time ? : depends on which pivots get chosen.  
(could be as bad as  $\theta(n^2)$  )

Key : find pivot giving “balanced” split.

Best pivot: the median ! (but this is circular)

⇒ Would get recurrence  $T(n) \leq T(n/2) + O(n)$

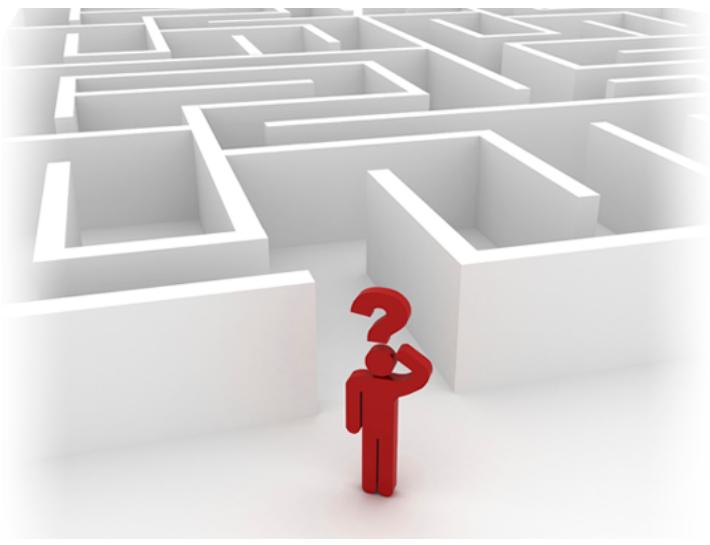
⇒  $T(n) = O(n)$  [ case 2 of Master Method ]

Hope : random pivot is “pretty good” “often enough”

# Running Time of RSelect

Rselect Theorem : for every input array of length  $n$ , the average running time of Rselect is  $O(n)$

- holds for every input [no assumptions on data]
- “average” is over random pivot choices made by the algorithm



Design and Analysis  
of Algorithms I

# Linear-Time Selection

---

## Deterministic Selection (Algorithm)

# The Problem

Input : array A with n **distinct** numbers and a number

For simplicity

Output :  $i^{\text{th}}$  order statistic (i.e.,  $i^{\text{th}}$  smallest element of input array)

Example : median.

(  $i = (n+1)/2$  for n odd,  
 $i = n/2$  for n even )

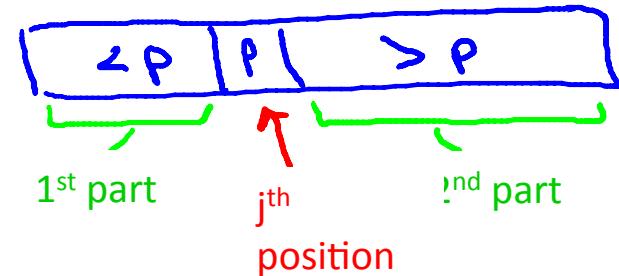


3<sup>rd</sup> order statistic

# Randomized Selection

Rselect (array A, length n, order statistic i)

- 0) if  $n = 1$  return  $A[1]$
- 1) Choose pivot  $p$  from  $A$  uniformly at random
- 2) Partition  $A$  around  $p$   
let  $j$  = new index of  $p$
- 3) If  $j = i$ , return  $p$
- 4) If  $j > i$ , return Rselect( $1^{\text{st}}$  part of  $A$ ,  $j-1$ ,  $i$ )
- 5) [if  $j < i$ ] return Rselect ( $2^{\text{nd}}$  part of  $A$ ,  $n-j$ ,  $i-j$ )



# Guaranteeing a Good Pivot

Recall : “best” pivot = the median !     (seems circular!)

Goal : find pivot guaranteed to be pretty good.

Key Idea : use “median of medians”!

# A Deterministic ChoosePivot

ChoosePivot(A,n)

- logically break A into  $n/5$  groups of size 5 each
- sort each group (e.g., using Merge Sort)
- copy  $n/5$  medians (i.e., middle element of each sorted group) into new array C
- recursively compute median of C (!)
- return this as pivot

# The DSelect Algorithm

DSelect(array A, length n, order statistic i)

1. Break A into groups of 5, sort each group
2. C = the  $n/5$  “middle elements”
3.  $p = \text{DSelect}(C, n/5, n/10)$  [recursively computes median of C]
4. Partition A around p
5. If  $j = i$  return p
6. If  $j < i$  return DSelect( $1^{\text{st}}$  part of A,  $j-1$ ,  $i$ )
7. [else if  $j > i$ ] return DSelect( $2^{\text{nd}}$  part of A,  $n-j$ ,  $i-j$ )

ChoosePivot

Same as  
before

How many recursive calls does DSelect make?

0

1

2

3

# Running Time of DSelect

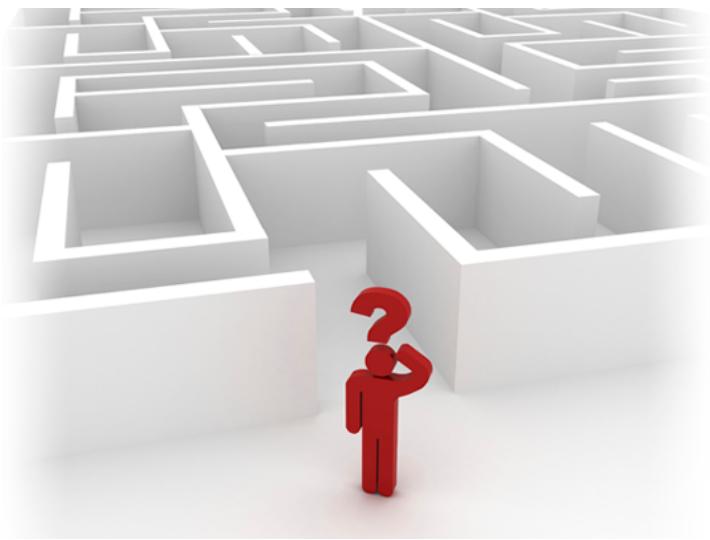
Dselect Theorem : for every input array of length  $n$ ,  
Dselect runs in  $O(n)$  time.

Warning : not as good as Rselect in practice

- 1) Worse constraints
- 2) not-in-place

History : from 1973

Blum – Floyd – Pratt – Rivest – Tarjan  
(‘95)    (‘78)                         (‘02)    (‘86)



Design and Analysis  
of Algorithms I

# Linear-Time Selection

---

## Deterministic Selection (Analysis)

# The DSelect Algorithm

DSelect(array A, length n, order statistic i)

1. Break A into groups of 5, sort each group

2. C = the  $n/5$  “middle elements”

3.  $p = \text{DSelect}(C, n/5, n/10)$  [recursively computes median of C]

4. Partition A around p

5. If  $j = i$  return p

6. If  $j < i$  return DSelect( $1^{\text{st}}$  part of A,  $j-1$ ,  $i$ )

7. [else if  $j > i$ ] return DSelect( $2^{\text{nd}}$  part of A,  $n-j$ ,  $i-j$ )

Choose  
Pivot

Same as  
before

What is the asymptotic running time of step 1 of the DSelect algorithm?

- $\theta(1)$
- $\theta(\log n)$
- $\theta(n)$
- $\theta(n \log n)$

Note : sorting an array with 5 elements takes  
 $\leq 120$  operations

[ why 120 ? Take  $m = 5$  in our  $6m(\log_2 m + 1)$  bound for Merge Sort ]

$$6 * 5 * (\log_2 5 + 1) \leq 120$$

$\leq 3$

# of gaps      ops per group

So :  $\leq (n/5) * 120 = 24n = O(n)$  for all groups

# The DSelect Algorithm

- DSelect(array A, length n, order statistic i)  $\theta(n)$
1. Break A into groups of 5, sort each group  $\theta(n)$
  2. C = the  $n/5$  “middle elements”  $\theta(n)$
  3. p = DSelect(C,  $n/5$ ,  $n/10$ ) [recursively computes median of C]
  4. Partition A around p  $T\left(\frac{n}{5}\right)$
  5. If  $j = i$  return p  $\theta(n)$
  6. If  $j < i$  return DSelect(1<sup>st</sup> part of A,  $j-1$ , i)  $T(?)$
  7. [else if  $j > i$ ] return DSelect(2nd part of A,  $n-j$ ,  $i-j$ )

# Rough Recurrence

Let  $T(n)$  = maximum running time of Dselect on an input array of length  $n$ .

There is a constant  $c \geq 1$  such that :

1.  $T(1) = 1$
2.  $T(n) \leq c*n + T(n/5) + T(?)$

sorting the groups  
partition

recursive  
call in line 3

recursive call in  
line 6 or 7

# The Key Lemma

Key Lemma : 2<sup>nd</sup> recursive call (in line 6 or 7) guaranteed to be on an array of size  $\leq 7n/10$  (roughly)

Upshot : can replace “?” by “ $7n/10$ ”

Rough Proof : Let  $k = n/5 = \# \text{ of groups}$   
Let  $x_i = i^{\text{th}}$  smallest of the  $k$  “middle elements”  
[So pivot =  $x_{k/2}$ ]

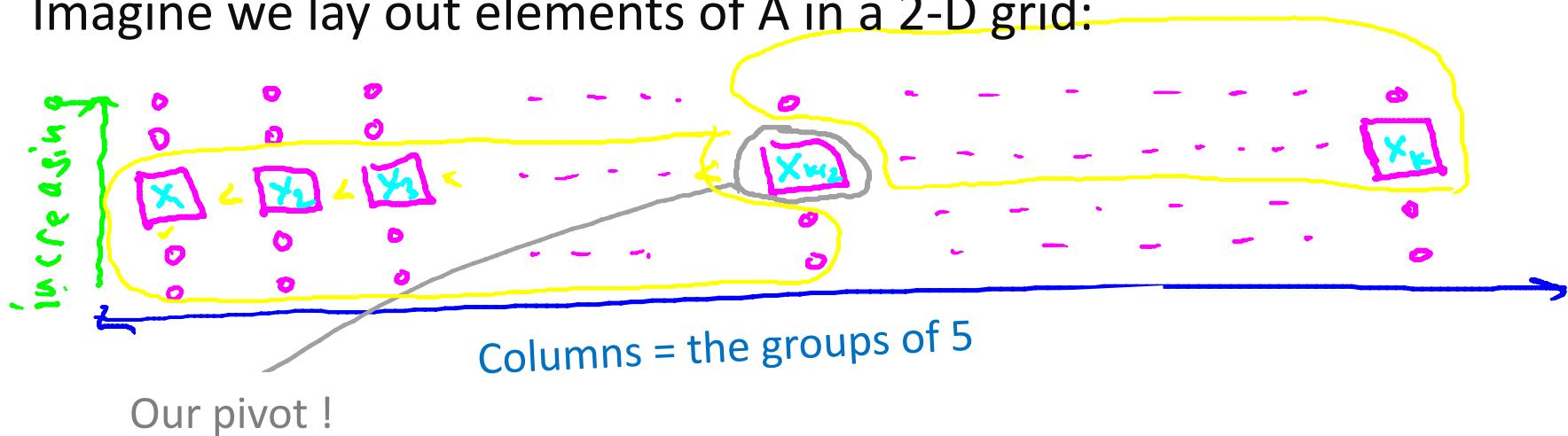
Goal :  $\geq 30\%$  of input array smaller than  $x_{k/2}$ ,  
 $\geq 30\%$  is bigger

Tim Roughgarden

# Rough Proof of Key Lemma

Thought Experiment :

Imagine we lay out elements of A in a 2-D grid:

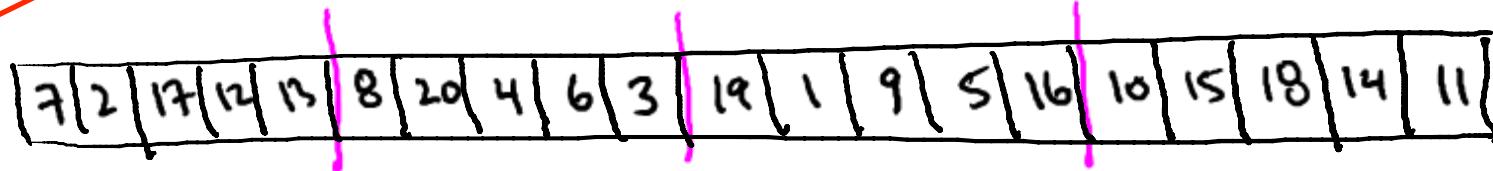


Key point :  $x_{k/2}$  bigger than 3 out of 5 (60%) of the elements in  
~ 50% of the groups

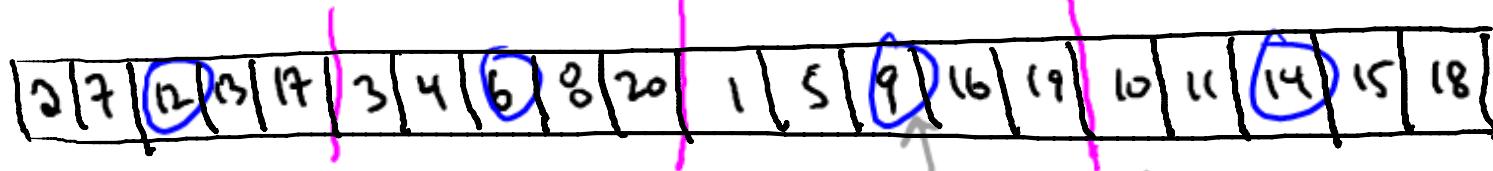
=> bigger than 30% of A (similarly, smaller than 30% of A)

# Example

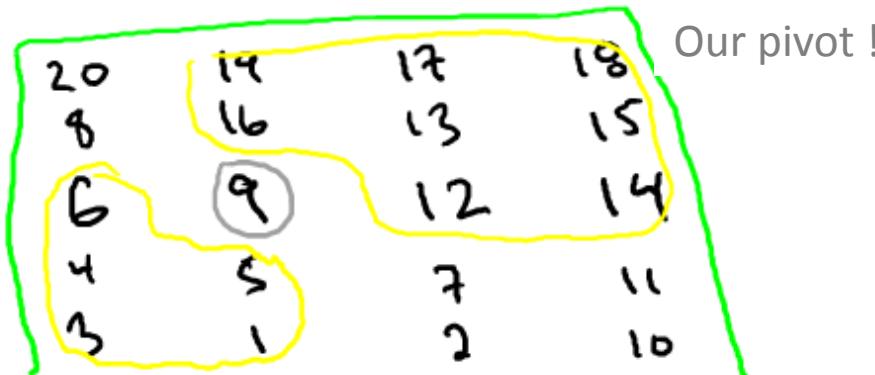
Input



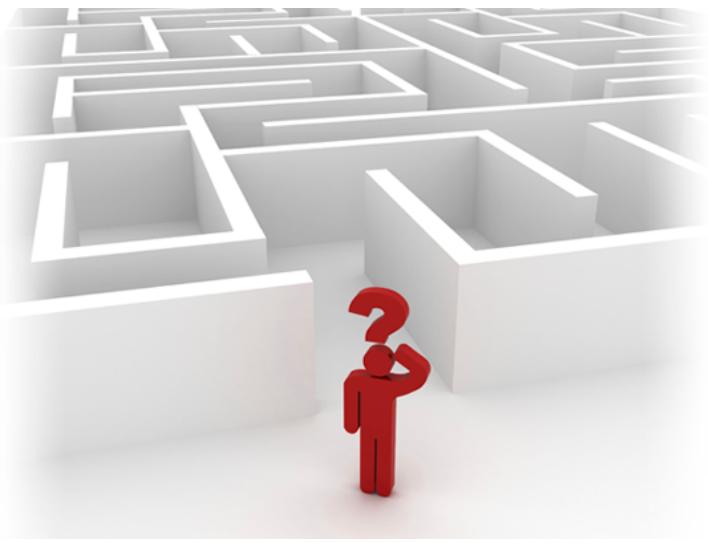
After  
sorting  
groups  
of 5



The  
grid :



Tim Roughgarden



Design and Analysis  
of Algorithms I

# Linear-Time Selection

---

## Deterministic Selection (Analysis II)

# Rough Recurrence (Revisited)

Let  $T(n)$  = maximum running time of Dselect on an input array of length  $n$ .

There is a constant  $c \geq 1$  such that :

1.  $T(1) = 1$
2.  $T(n) \leq c*n + T(n/5) + T(\text{?})$

$\leq 7n/10$  by  
Key Lemma

sorting the groups      recursive      recursive call in  
partition                  call in line 3                  line 6 or 7

# Rough Recurrence (Revisited)

$$T(1) = 1, T(n) \leq cn + T(n/5) + T(7n/10)$$

Constant  $c \geq 1$

Note : different-sized subproblems => can't use Master Method!

Strategy : “hope and check”

Hope : there is some constant  $a$  [independent of  $n$ ]

Such that  $T(n) \leq an$  for all  $n \geq 1$

[if true, then  $T(n) = O(n)$  and algorithm is linear time ]

# Analysis of Rough Recurrence

Claim : Let  $a = 10c$

Then  $T(n) \leq an$  for all  $n \geq 1$

=> Dselect runs in  
 $O(n)$  time

$$T(1) = 1 ; T(n) \leq cn + T(n/5) + T(7n/10)$$

Constant  $c \geq 1$

Proof : by induction on  $n$

Base case :  $T(1) = 1 \leq a*1$  (since  $a \geq 1$ )

Inductive Step :  $[n > 1]$

Inductive Hypothesis :  $T(k) \leq ak \forall k < n$

We have  $T(n) \leq cn + T(n/5) + T(7n/10)$

$$\begin{aligned} &\stackrel{\text{GIVEN}}{\leq} cn + a(n/5) + a(7n/10) \\ &\stackrel{\text{IND HYP}}{=} n(c + 9a/10) = an \end{aligned}$$

Q.E.D.

Tim Roughgarden



Design and Analysis  
of Algorithms I

# Linear-Time Selection

---

An  $\Omega(n \log n)$   
Sorting Lower Bound

# A Sorting Lower Bound

Theorem : every “comparison-based” sorting algorithm has worst-case running time  $\Omega(n \log n)$

[ assume deterministic, but lower bound extends to randomized ]

Comparison-Based Sort : accesses input array elements only via comparisons ~ “general purpose sorting method”

Examples : Merge Sort, Quick Sort, Heap Sort

Non Examples : Bucket Sort, Counting Sort, Radix Sort

Three green arrows point from the text "Non Examples" to the three sorting methods listed. The first arrow points to "Bucket Sort" with the label "Good for data from distributions". The second arrow points to "Counting Sort" with the label "good for small integers". The third arrow points to "Radix Sort" with the label "good for medium-size integers".

Tim Roughgarden

# Proof Idea

Fix a comparison-based sorting method and an array length  $n$

⇒ Consider input arrays containing  $\{1, 2, 3, \dots, n\}$  in some order.

⇒  $n!$  such inputs

Suppose algorithm always makes  $\leq k$  comparisons to correctly sort these  $n!$  inputs.

=> Across all  $n!$  possible inputs, algorithm exhibits  $\leq 2^k$  distinct executions      i.e., resolution of the comparisons

# Proof Idea (con'd)

By the Pigeonhole Principle : if  $2^k < n!$ , execute identically on two distinct inputs => must get one of them incorrect.

So : Since method is correct,

$$\begin{aligned} 2^k &\geq n! \\ &\geq \left(\frac{n}{2}\right)^{\frac{n}{2}} \\ \Rightarrow k &\geq \frac{n}{2} \cdot \log_2 \frac{n}{2} = \Omega(n \log n) \end{aligned}$$