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 = 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^{st} part of A, j-1, i)
7. [else if j > i] return DSelect(2nd part of A, n-j, i-j)

Choose Pivot

Same as before

Ter
ver

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 <= 120 operations

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

$6*5*(\log_2 5+1) <= 120$

$\log_2 5$ <=3

\# of gaps

ops per group

So : <= (n/5)*120 = 24n = O(n) for all groups

The DSelect Algorithm

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

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] → $T(\frac{n}{5})$
4. Partition A around p → $\theta(n)$
5. If j = i return p
6. If j < i return DSelect(1st part of A, j-1, i)
7. [else if j > i] return DSelect(2nd part of A, n-j, i-j)

(steps 6–7) → $T(?)$

Tim Roughgarden

Rough Recurrence

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

There is a constant c >= 1 such that :

1. T(1) = 1
2. T(n) <= 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^{nd} recursive call (in line 6 or 7) guaranteed to be on an array of size <= 7n/10 (roughly)

Upshot : can replace "?" by "7n/10"

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

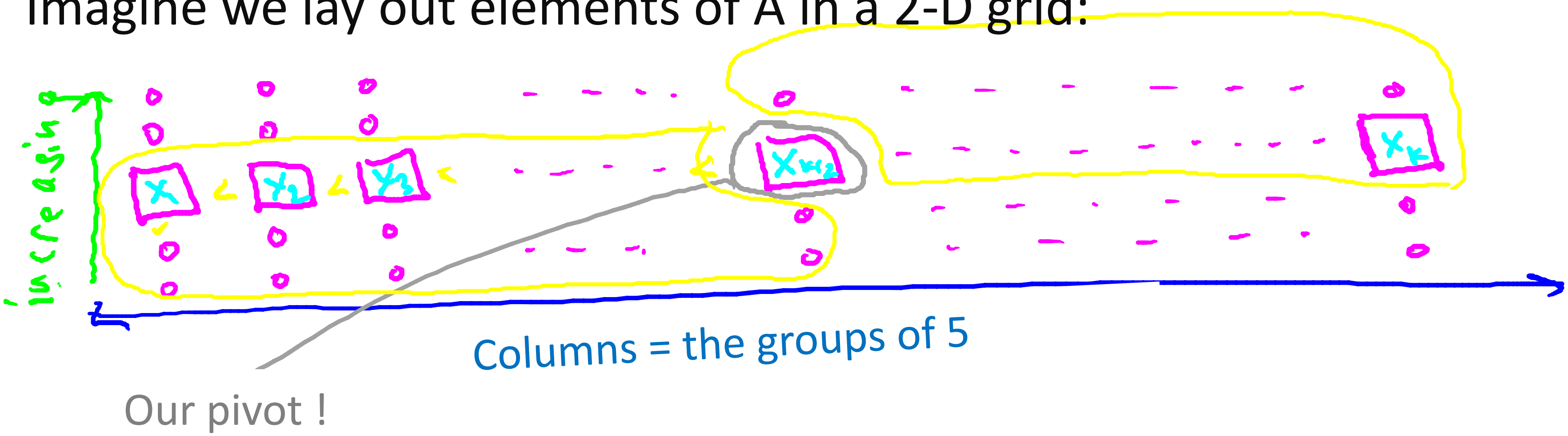
Goal : >= 30% of input array smaller than $x_{k/2}$,
>= 30% is bigger

Tim Roughgarden

Rough Proof of Key Lemma

Thought Experiment :

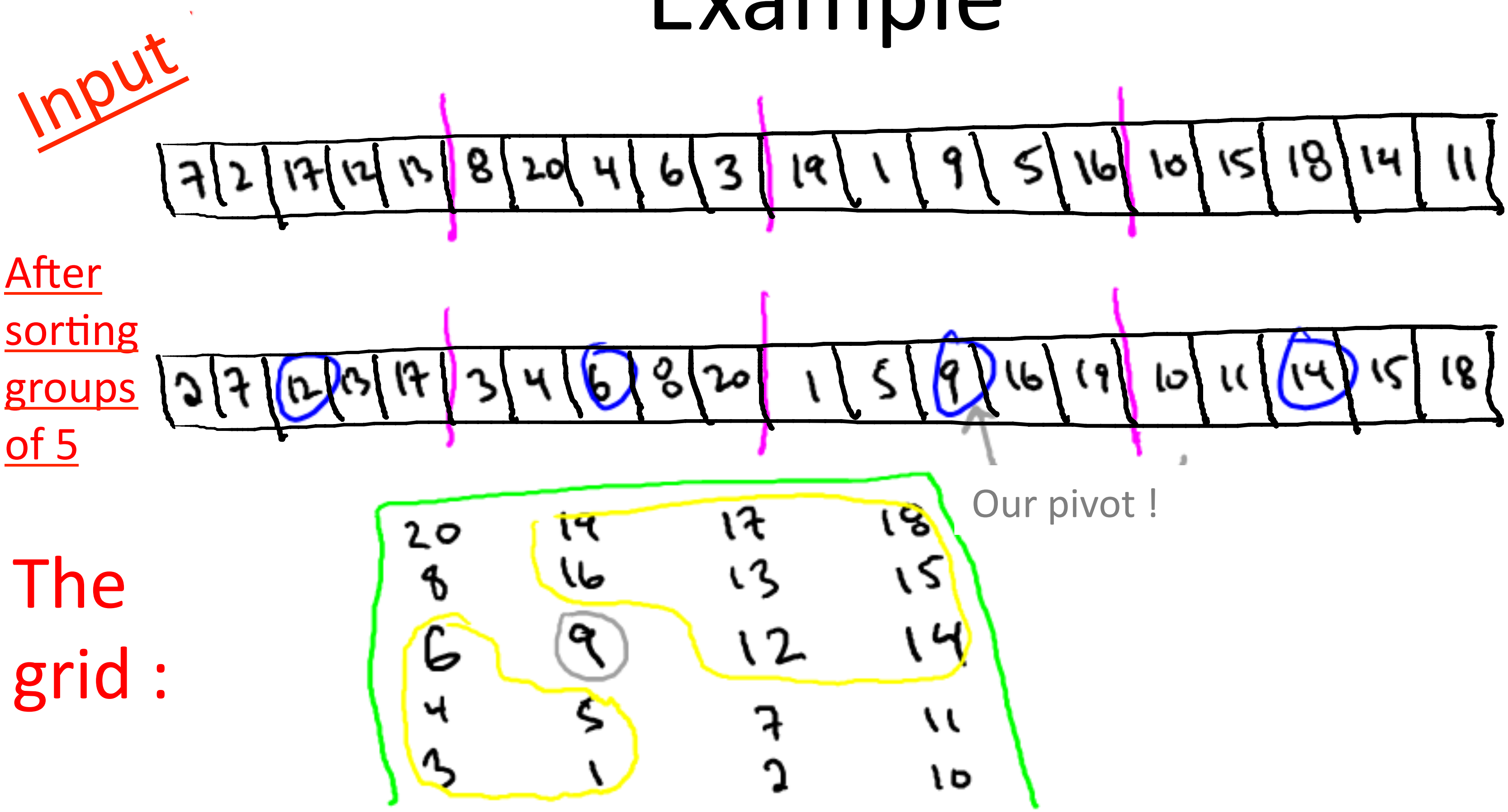
Imagine we lay out elements of A in a 2-D grid:



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



Tim Roughgarden