Design and Analysis of Algorithms I

Linear-Time Selection

Deterministic Selection (Algorithm)

The Problem

for simplicity

Input: array A with n distinct numbers and a number $i \in \{1, 2, \ldots, n\}$.

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

Example: median.

($i = \frac{n+1}{2}$ for n odd,

$i = n/2$ for n even)

10	8	2	4

3^{rd} order statistic

Randomized Selection

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

⓪ if n=1 return A[1]

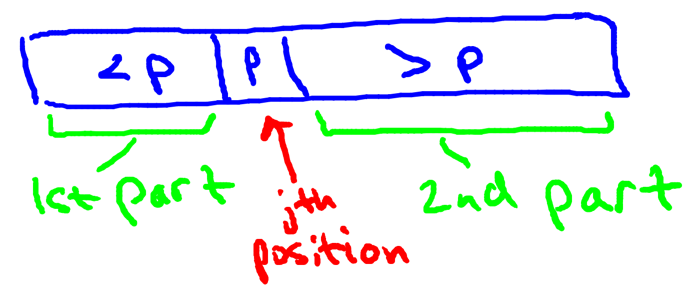
① choose pivot p from A uniformly at random

② partition A around p
let j = new index of p

③ if j=i return p

④ if j>i return RSelect (1st part of A, j−1, i)

⑤ [if j<i] return RSelect (2nd part of A, n−j, i−j)



Tim Roughgarden

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"!

Tim Roughgarden

A Deterministic ChoosePivot

ChoosePivot(A,n)

- logically break A into $n/5$ groups of size 5 each
- sort each group (e.g., using MergeSort)
- 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 = 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(1st part of A, j-1, i)
7. [else if j > i] return DSelect(2nd 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 constants 2) not-in-place

History: from 1973.
Blum - Floyd - Pratt - Rivest - Tarjan
'95 '78 '02 '86
'70

Tim Roughgarden

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(1st 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 in RSelect

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 MergeSort]

$$\underbrace{6m(\log_2 m+1)}_{} \to 6 \cdot 5 \cdot (\underbrace{\log_2 5}_{\leq 3}+1) \leq 120$$

So: $\leq \underbrace{n/5}_{\text{\# groups}} \cdot \underbrace{120}_{\text{ops per group}} = 24n = O(n)$ for all groups

The DSelect Algorithm

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

1. Break A into groups of 5, sort each group $\rightarrow \Theta(n)$
2. C = the n/5 "middle elements" $\rightarrow \Theta(n)$
3. p = DSelect(C,n/5,n/10) [recursively computes median of C] $\rightarrow T(\frac{n}{5})$
4. Partition A around p $\rightarrow \Theta(n)$
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)

$T(?)$

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 $T(n) \leq \underbrace{cn}_{\text{sorting the groups, partition}} + \underbrace{T\left(\frac{n}{5}\right)}_{\text{recursive call in line 3}} + \underbrace{T(?)}_{\text{recursive call in line 6 or 7}}$

The Key Lemma

Key Lemma: 2nd recursive call (in line 6 or 7) guaranteed to be on an array of size $\leq \frac{7}{10}n$ (roughly).

Upshot: can replace "?" by "$\frac{7}{10}n$".

Rough Proof: Let $k = \frac{n}{5}$ = # of groups.

Let x_i = i^{th} smallest of the k "middle elements".

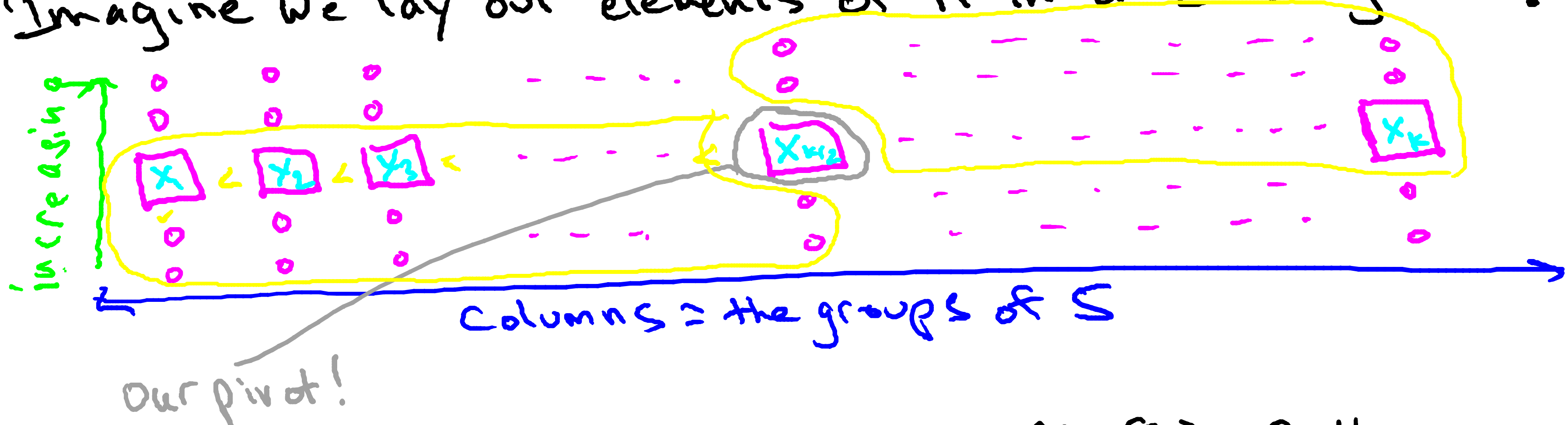
[so pivot = $x_{k/2}$]

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

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 5 (60%) of the elements in $\approx$ 50% of the groups

$\Rightarrow$ bigger than 30% of A (Similarly, smaller than 30% of A)

Example

Input

7	2	17	12	13	8	20	4	6	3	19	1	9	5	16	10	15	18	14	11

After sorting groups of 5

2	7	12	13	17	3	4	6	8	20	1	5	9	16	19	10	11	14	15	18

our pivot!

The grid:

20	14	17	18
8	16	13	15
6	9	12	14
4	5	7	11
3	1	2	10

Rough Recurrence (Revisited)

Tim Roughgarden

Analysis of Rough Recurrence

Tim Roughgarden