Strassen's Algorithm for Matrix Multiplication, QuickSelect, and Median of Medians

Readings: CLRS Ch. 4.2, Ch. 9

CS 4102: Algorithms

Spring 2021

Mark Floryan and Tom Horton

Readings

- CLRS Section 4.2 on Strassen's algorithm

- CLRS Chapter 9
- Wikipedia articles on Quickselect and Median of Medians

Matrix Multiplication

Matrix Multiplication

$$n\overbrace{\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}}^{n} \times \begin{bmatrix} 2 & 4 & 6 \\ 8 & 10 & 12 \\ 14 & 16 & 18 \end{bmatrix}$$

$$= \begin{bmatrix} 2 + 16 + 42 & 4 + 20 + 48 & 6 + 24 + 54 \\ \vdots & \vdots & \vdots \end{bmatrix}$$

$$= \begin{bmatrix} 60 & 72 & 84 \\ 132 & 162 & 192 \\ 204 & 252 & 300 \end{bmatrix}$$

Run time? $O(n^3)$ Lower Bound? $O(n^2)$

Matrix Multiplication D&C

Multiply $n \times n$ matrices (A and B)

Divide:

$$A = \left[\begin{array}{cc|cc} a_1 & a_2 & a_3 & a_4 \\ a_5 & a_6 & a_7 & a_8 \\ \hline a_9 & a_{10} & a_{11} & a_{12} \\ a_{13} & a_{14} & a_{15} & a_{16} \end{array}\right] \qquad B = \left[\begin{array}{cc|cc} b_1 & b_2 & b_3 & b_4 \\ b_5 & b_6 & b_7 & b_8 \\ \hline b_9 & b_{10} & b_{11} & b_{12} \\ b_{13} & b_{14} & b_{15} & b_{16} \end{array}\right]$$

Matrix Multiplication D&C

Multiply $n \times n$ matrices (A and B)

$$A = \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix} \qquad B = \begin{bmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{bmatrix}$$

Combine:

$$AB = \begin{bmatrix} A_{1,1}B_{1,1} + A_{1,2}B_{2,1} & A_{1,1}B_{1,2} + A_{1,2}B_{2,2} \\ A_{2,1}B_{1,1} + A_{2,2}B_{2,1} & A_{2,1}B_{1,2} + A_{2,2}B_{2,2} \end{bmatrix}$$

Run time? $T(n) = 8T\left(\frac{n}{2}\right) + 4\left(\frac{n}{2}\right)^2$

Case 1!

$T(n) = \Theta(n^3)$

Find an Algorithm with Better Recurrence?

$$T(n) = 8T\left(\frac{n}{2}\right) + 4\left(\frac{n}{2}\right)^2$$

- We've got a recurrence and want to improve things.
  You know how the Master Theorem works.
  What can we change to make it better?
  - Reduce the number of subproblems.
  - Reduce the order class of the non-recursive work.
    (OK to do more non-recursive work if new f(n) is same Θ)

Strassen's Algorithm

Multiply $n \times n$ matrices (A and B)

$$A = \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix} \qquad B = \begin{bmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{bmatrix}$$

Calculate:

$$Q_1 = (A_{1,1} + A_{2,2})(B_{1,1} + B_{2,2})$$
$$Q_2 = (A_{2,1} + A_{2,2})B_{1,1}$$
$$Q_3 = A_{1,1}(B_{1,2} - B_{2,2})$$
$$Q_4 = A_{2,2}(B_{2,1} - B_{1,1})$$
$$Q_5 = (A_{1,1} + A_{1,2})B_{2,2}$$
$$Q_6 = (A_{2,1} - A_{1,1})(B_{1,1} + B_{1,2})$$
$$Q_7 = (A_{1,2} - A_{2,2})(B_{2,1} + B_{2,2})$$

Find AB:

$$\begin{bmatrix} A_{1,1}B_{1,1} + A_{1,2}B_{2,1} & A_{1,1}B_{1,2} + A_{1,2}B_{2,2} \\ A_{2,1}B_{1,1} + A_{2,2}B_{2,1} & A_{2,1}B_{1,2} + A_{2,2}B_{2,2} \end{bmatrix} = \begin{bmatrix} Q_1 + Q_4 - Q_5 + Q_7 & Q_3 + Q_5 \\ Q_2 + Q_4 & Q_1 - Q_2 + Q_3 + Q_6 \end{bmatrix}$$

Number Mults.: 7 Number Adds: 18

$$T(n) = 7T\left(\frac{n}{2}\right) + 18\left(\frac{n}{2}\right)^2$$

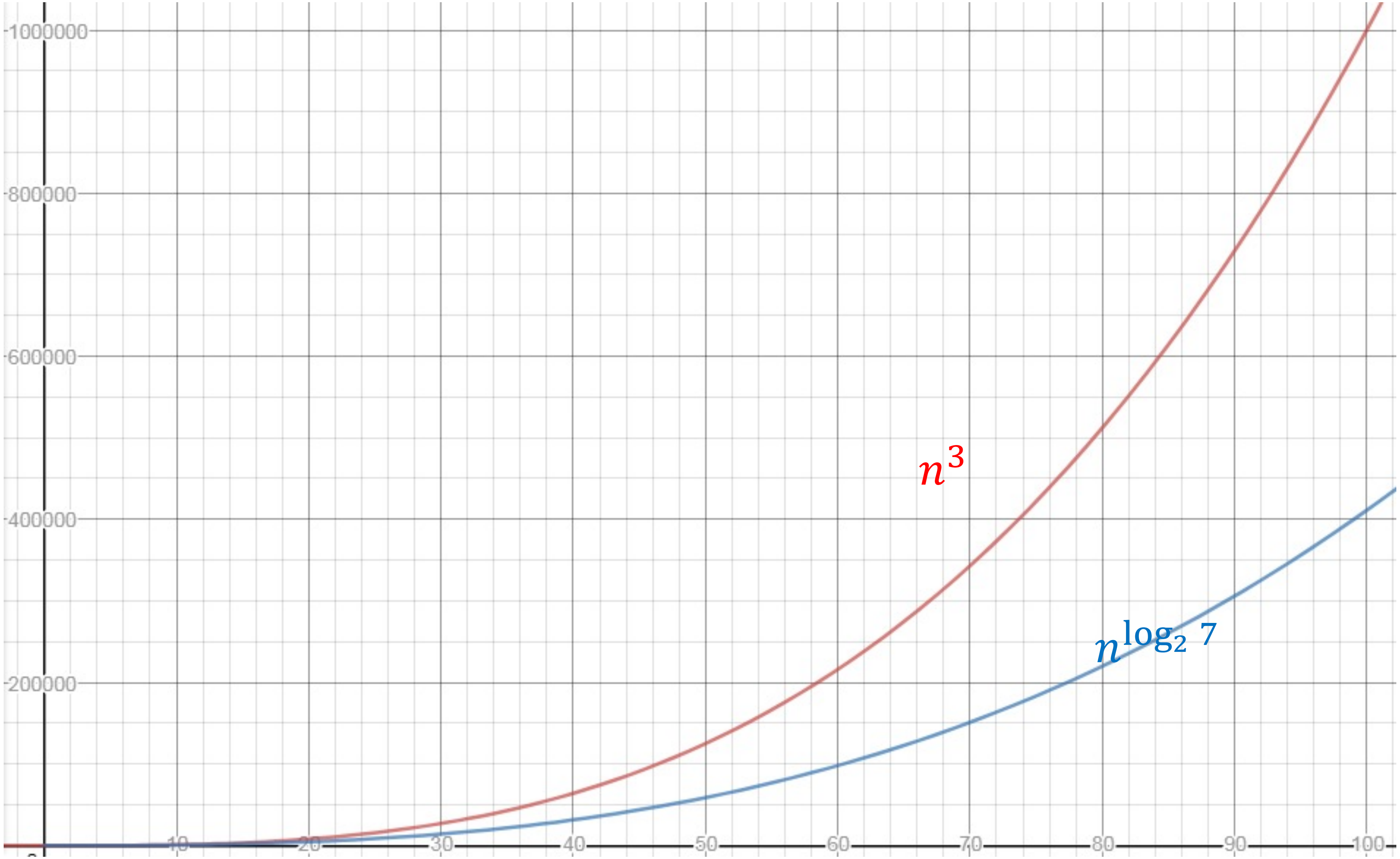
Strassen's Algorithm

$$T(n) = 7T\left(\frac{n}{2}\right) + \frac{9}{2}n^2$$

$$a = 7, b = 2, f(n) = \frac{9}{2}n^2$$

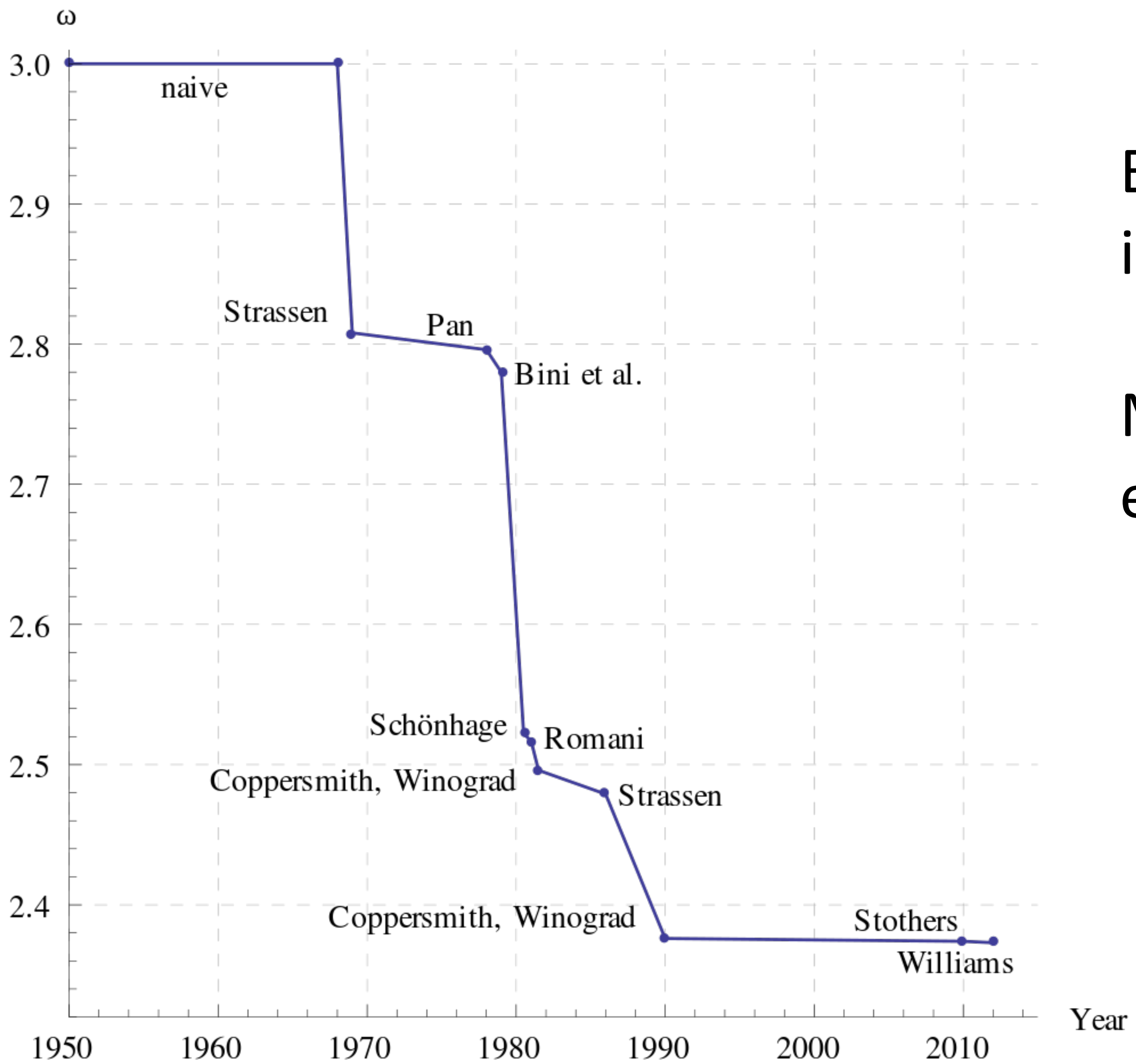
$$n^{\log_b a} = n^{\log_2 7} \approx n^{2.807}$$ Case 1!

$$T(n) = \Theta\left(n^{\log_2 7}\right) \approx \Theta(n^{2.807})$$



1000000
800000
600000
400000
200000
10
20
30
40
50
60
70
80
90
100
n^3
$n^{\log_2 7}$

Is this the fastest?



Best possible is unknown

May not even exist!

Quickselect

Review: Quicksort

Idea: pick a pivot element, recursively sort two sublists around that element

- Divide: select pivot element p, Partition(p)
- Conquer: recursively sort left and right sublists
- Combine: Nothing!

Partition (Divide step)

Given: a list, a pivot p

Start: unordered list

8	5	7	3	12	10	1	2	4	9	6	11

Goal: All elements $< p$ on left, all $> p$ on right

5	7	3	1	2	4	6	8	12	10	9	11

Quicksort Run Time (Best)

If the pivot is always the median:

2	5	1	3	6	4	7	8	10	9	11	12

2	1	3	5	6	4	7	8	9	10	11	12

Then we divide in half each time

$$T(n) = 2T\left(\frac{n}{2}\right) + n$$

$$T(n) = O(n \log n)$$

Quicksort Run Time (Worst)

If the pivot is always at the extreme:

1	5	2	3	6	4	7	8	10	9	11	12

1	2	3	5	6	4	7	8	10	9	11	12

Then we shorten by 1 each time

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

$$T(n) = O(n^2)$$

Can we Pick a Good Pivot for Quicksort?

- What makes a good Pivot for Quicksort?
  - Roughly even split between left and right
  - Ideally: the median
- Can we find a list's median in linear time?
  - Quickselect (https://en.wikipedia.org/wiki/Quickselect)
    - Finds the median
    - Works a lot like Quicksort: needs to do a Partition
    - We need a good pivot for Quickselect for it to have good time-complexity
  - Median of Medians (https://en.wikipedia.org/wiki/Median_of_medians)
    - Can be used to find "pretty good" pivot for QS, or with Quickselect

Quickselect

- Finds i^{th} order statistic
  - i^{th} smallest element in the list
  - 1^{st} order statistic: minimum
  - n^{th} order statistic: maximum
  - $\frac{n}{2}^{\text{th}}$ order statistic: median
- CLRS, Section 9.1
  - **Selection problem**: Give list of distinct numbers and value *i*, find value *x* in list that is larger than exactly *i-1* list elements

Quickselect

Idea: pick a pivot element, partition, then recurse on the sublist containing index i

- Divide: select an element p, Partition(p)
- Conquer: if $i =$ index of p, done!
  - if $i <$ index of p recurse left. Else recurse right
- Combine: Nothing!

(Note: just one recursive call, unlike Quicksort.)

Partition (Divide step)

Given: a list, a pivot value x

Note: now using "x" to refer to pivot value. We called it "p" in previous slides.

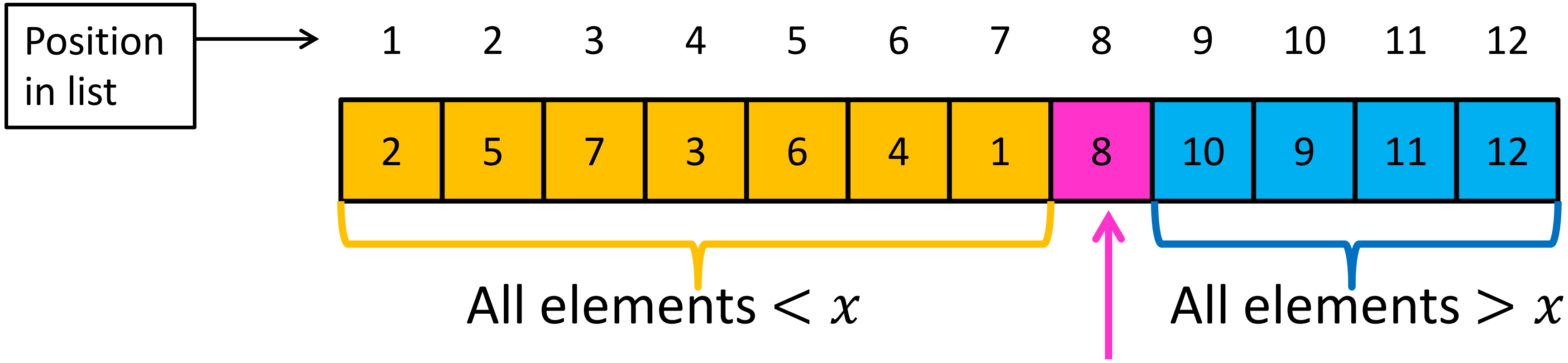
Start: unordered list

11	5	7	3	12	10	1	2	4	9	6	8

Goal: All elements $< x$ on left, all $> x$ on right

5	7	3	1	2	4	6	8	12	10	9	11

Conquer



$x = 8$ put exactly where it belongs at <u>position 8</u> (the split-point)

Remember: we're looking for the i^{th} order statistic

- If the split-point (8) is i we're done! The value stored at the split-point is the result.
- If $i <$ split-point, look in left sub-list (using same value i)
- If $i >$ split-point, look in right sub-list (using an adjusted value of i)
  - For example, if we wanted the 10^{th} order statistic in the entire list, here that would be the 2^{nd} order statistic in the right sub-list

CLRS Pseudocode for Quickselect

A – the list
p – index of first item
r – index of last item
i – find *i*th smallest item
q – pivot location
k – number on left + 1

```
RANDOMIZED-SELECT(A, p, r, i)
if p == r
    return A[p]
q = RANDOMIZED-PARTITION(A, p, r)
k = q − p + 1          // number of elements in left sub-list + 1
if i == k              // the pivot value is the answer
    return A[q]
elseif i < k
    return RANDOMIZED-SELECT(A, p, q − 1, i)
else return RANDOMIZED-SELECT(A, q + 1, r, i − k)
```

// note adjustment to *i* when recursing on right side

Note: In CLRS, they're using a partition that randomly chooses the pivot element.
That's why you see "Randomized" in the names here. Ignore that for the moment.

Work These Examples!

- For each of the following calls, show
  - The value of *q* after each partition,
  - Which recursive calls made

1. Select([3, 2, 9, 0, 7, 5, 6, 1], p=0, r=7, i=2)
2. Select([3, 2, 9, 0, 7, 5, 6, 1], p=0, r=7, i=5)
3. Select([3, 2, 9, 0, 7, 5, 6, 1], p=0, r=7, i=7)

Quickselect Run Time

If the pivot is always the median:

2	5	1	3	6	4	7	8	10	9	11	12

2	1	3	5	6	4	7	8	9	10	11	12

Then we divide in half each time

$$S(n) = S\left(\frac{n}{2}\right) + n$$

$$S(n) = O(n)$$

Quickselect Run Time

If the partition is always unbalanced:

1	5	2	3	6	4	7	8	10	9	11	12

1	2	3	5	6	4	7	8	10	9	11	12

Then we shorten by 1 each time

$$S(n) = S(n-1) + n$$

$$S(n) = O(n^2)$$

Good Pivot for Quickselect

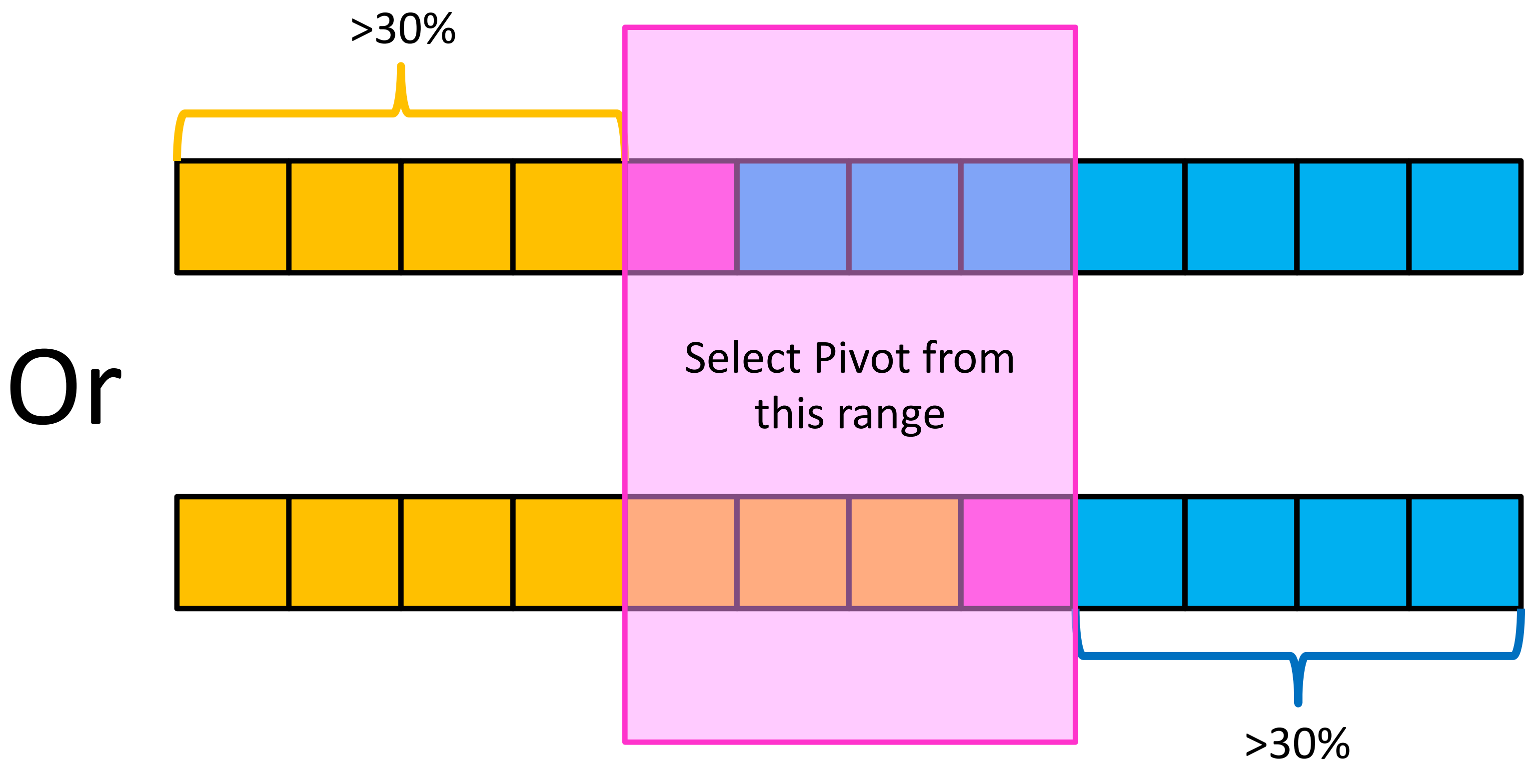
- What makes a good Pivot for Quickselect?
  - Roughly even split between left and right
  - Ideally: median

Déjà vu?

- Here's what's next:
  - First, **median of medians** algorithm
    - Finds something close to the median in $\Theta(n)$ time
  - Second, we can prove that when its result used with Quickselect's partition, then Quickselect is guaranteed $\Theta(n)$
    - Because we now have a $\Theta(n)$ way to find the median, this guarantees Quicksort will be $\Theta(n \lg n)$
  - Notes:
    - We have to do all this for every call to Partition in Quicksort
    - We could just use the value returned by median of medians for Quicksort's Partition
      - See CLRS section "Balanced Partitioning" starting on p. 175

Pretty Good Pivot

- What makes a "pretty good" Pivot?
  - Both sides of Pivot >30%



Median of Medians

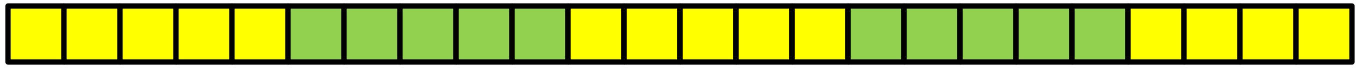
- Fast way to select a "pretty good" pivot
- Guarantees pivot is greater than 30% of elements and less than 30% of the elements
  - I.e. it's in the middle 40% (±20% of the true median)
- Idea: break list into chunks, find the median of each chunk, use the median of those medians

- CLRS, pp. 220-221
- https://en.wikipedia.org/wiki/Median_of_medians

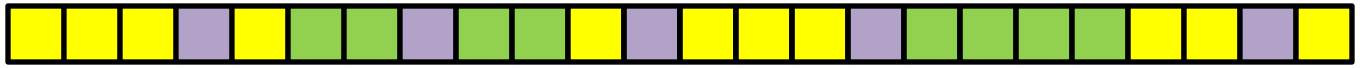
Median of Medians

1. Break list into chunks of size 5

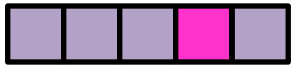
List could be long, many more than 5 chunks!



2. Find the median of each chunk
   (using insertion sort: n=5, 20 comparisons)

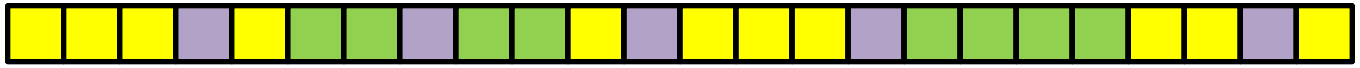


3. Return median of medians (using Quickselect, this algorithm, called recursively, on list of medians)



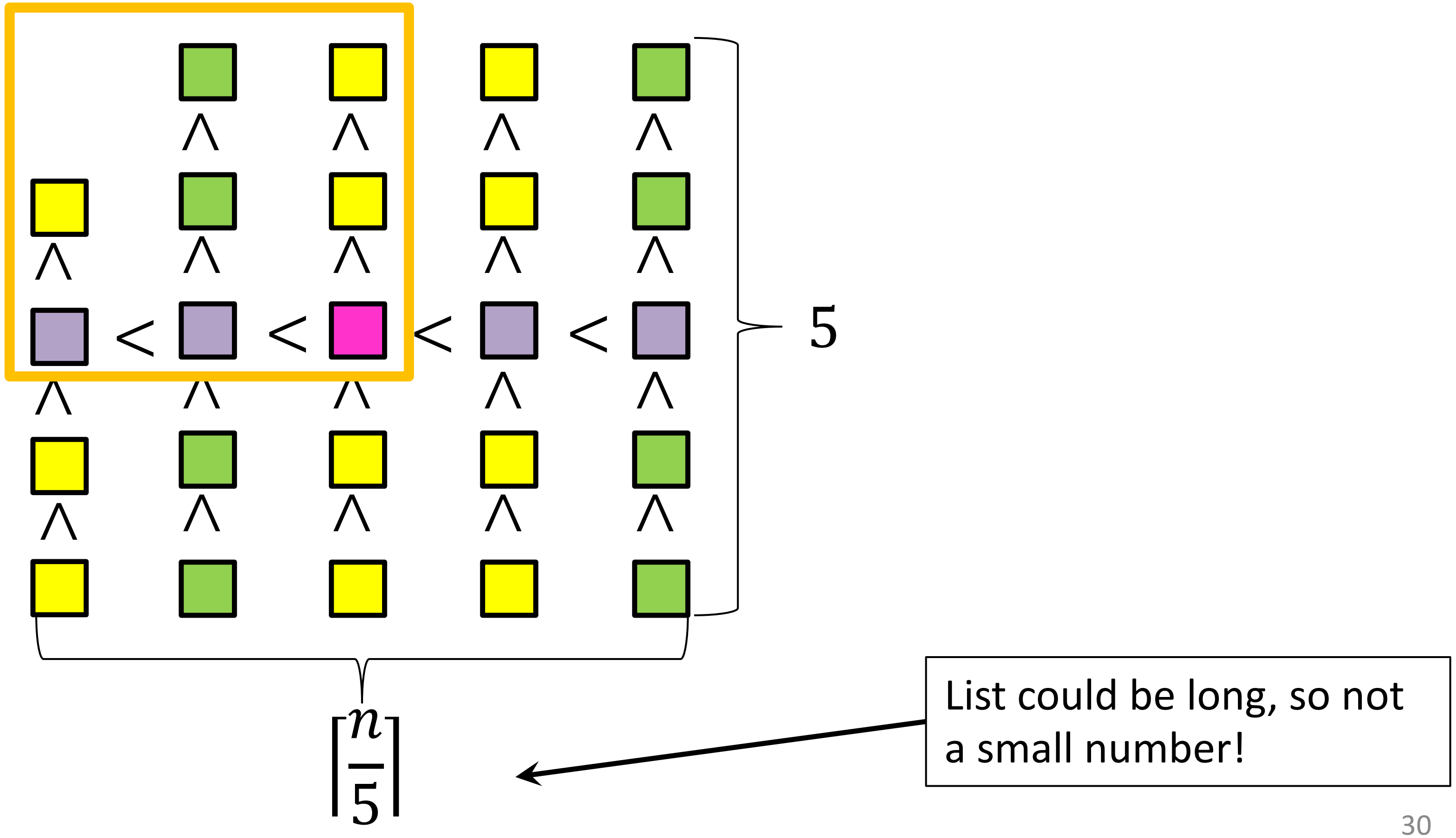
List could be long, many more than 5 medians!

Why is this good?



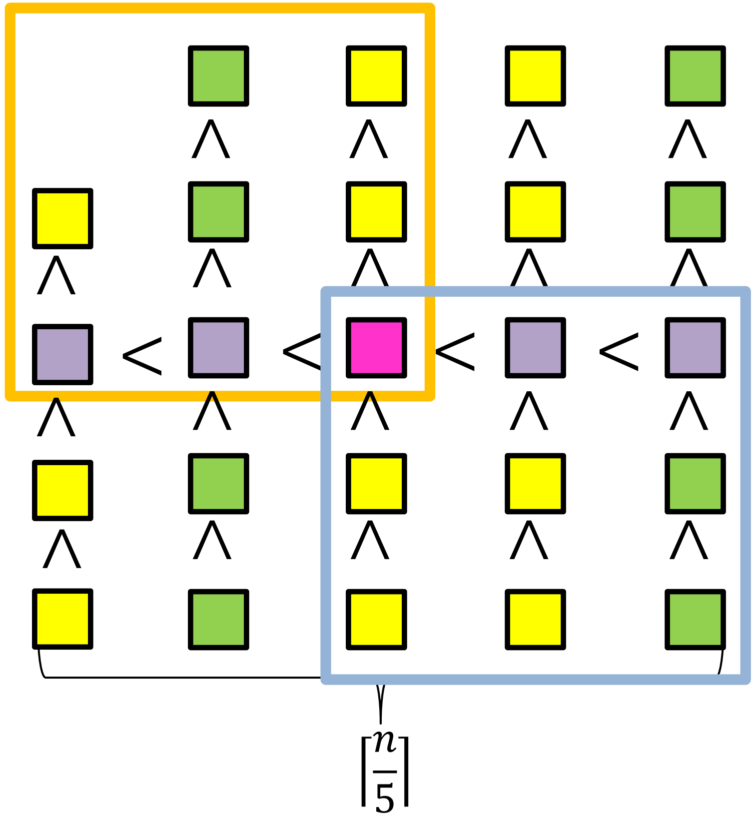
Imagine each chunk sorted, chunks ordered by their medians

MedianofMedians is Greater than all of these



Why is this good?

MedianofMedians is larger than all of these



Worried about the details of this math? See CLRS p. 221

Larger than 3 things in each (but one) list to the left

$3\left(\frac{1}{2}\cdot\left\lceil\frac{n}{5}\right\rceil - 2\right) \approx \frac{3n}{10} - 6$ elements $<$

Similarly:

$3\left(\frac{1}{2}\cdot\left\lceil\frac{n}{5}\right\rceil - 2\right) \approx \frac{3n}{10} - 6$ elements $>$

Run-time of Quickselect with Median of Medians

- What's the cost $S(n)$ for Quickselect with Median of Medians?
- Divide: select an element p using Median of Medians, Partition(p)

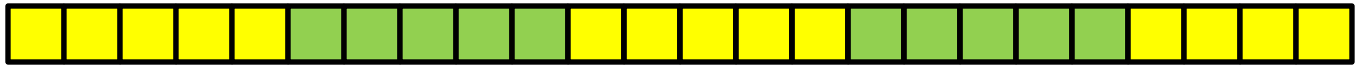
  $M(n) + \Theta(n)$
- Conquer: if i = index of p, done, if $i <$ index of p recurse left. Else recurse right

  $\leq S\left(\frac{7}{10}n\right)$
- Combine: Nothing!

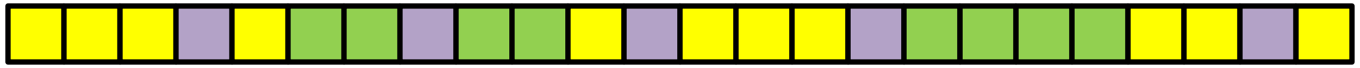
$$S(n) \leq S\left(\frac{7}{10}n\right) + M(n) + \Theta(n)$$

Run-time M(n) for Median of Medians

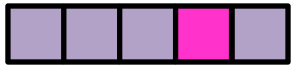
1. Break list into chunks of 5 $\Theta(n)$



2. Find the median of each chunk $\Theta(n)$



3. Return median of medians (using Quickselect)



$S\left(\frac{n}{5}\right)$

$$M(n) = S\left(\frac{n}{5}\right) + \Theta(n)$$

Quickselect

$$S(n) \le S\left(\frac{7n}{10}\right) + M(n) + \Theta(n) \qquad M(n) = S\left(\frac{n}{5}\right) + \Theta(n)$$

$$= S\left(\frac{7n}{10}\right) + S\left(\frac{n}{5}\right) + \Theta(n)$$

We can show by proof by induction that:

$S(n) = O(n)$ (next two slides)

$S(n) = \Omega(n)$

$$\therefore S(n) = \Theta(n)$$

Proof by Induction

$$T(n) = T(n/5) + T(7n/10) + c \cdot n$$

Prove $T(n) = O(n)$

Claim: $T(n) \leq 10cn$

Base Case: $T(0) = 0$

$T(1) = c \leq 10c$ which is true since $c \geq 1$

Strictly speaking, we can handle any $c > 0$, but assuming $c \geq 1$ to simplify the analysis here

Proof by Induction

$$T(n) = T(n/5) + T(7n/10) + c \cdot n$$

Inductive hypothesis: $\forall n \le x_0 : \underline{T(n) \le 10cn}$

Inductive step:

$$T(x_0 + 1) = T\left(\frac{1}{5}(x_0 + 1)\right) + T\left(\frac{7}{10}(x_0 + 1)\right) + c(x_0 + 1)$$

Use inductive hypothesis

$$\le 10c\left(\frac{1}{5}(x_0 + 1)\right) + 10c\left(\frac{7}{10}(x_0 + 1)\right) + c(x_0 + 1)$$

Simplify terms w/ algebra

$$= 10c\left(\frac{1}{5} + \frac{7}{10}\right)(x_0 + 1) + c(x_0 + 1)$$

$$= 9c(x_0 + 1) + c(x_0 + 1) = \underline{10c(x_0 + 1)}$$

We've proved inductive hypothesis for $x_0 + 1$

Compare to 'Obvious' Approach

- An "obvious" approach to Selection Problem:
  - Given list and value *i*: Sort list, then choose *i*-th item
  - We've only seen sorting algorithms that are $\Omega(n \log n)$
  - We can show this really is a lower-bound
  - So this approach is $\Theta(n \log n)$
- Therefore Quickselect is asymptotically better than this sorting-based solution for Selection Problem!

Phew! Back to Quicksort

Using Quickselect, with a median-of-medians partition, we're guaranteed to use true median, so:

2	5	1	3	6	4	7	8	10	9	11	12

2	1	3	5	6	4	7	8	9	10	11	12

Then we divide in half each time

$$T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n)$$

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

Is it worth it?

- Using Quickselect to pick median guarantees $\Theta(n \log n)$ run time
- But, this approach has very large constants
  - If you absolutely must know it will be $\Theta(n \log n)$, choose MergeSort
- Better approach: Choose random pivot for Quicksort
  - Very small constant (random() is a fast algorithm)
  - Can prove the *expected runtime* is $\Theta(n \log n)$
    - Why? Getting unbalanced partitions every time is extremely unlikely