CS4102 Algorithms

Fall 2019

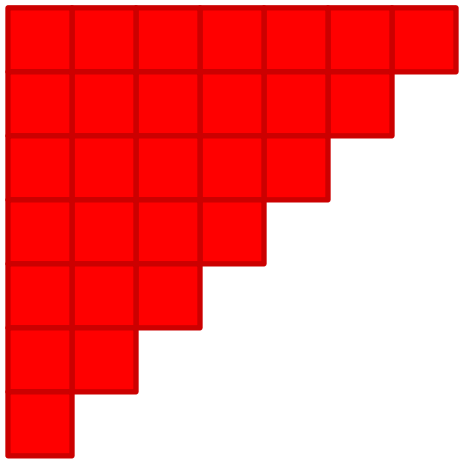
$$\begin{array}{r} 1+2+3+\cdots+98+99+100 \\ +\,100+99+98+\cdots+3+2+1 \\ \hline 101+101+101+\cdots+101+101+101 \end{array} = \frac{100\,(101)}{2}$$

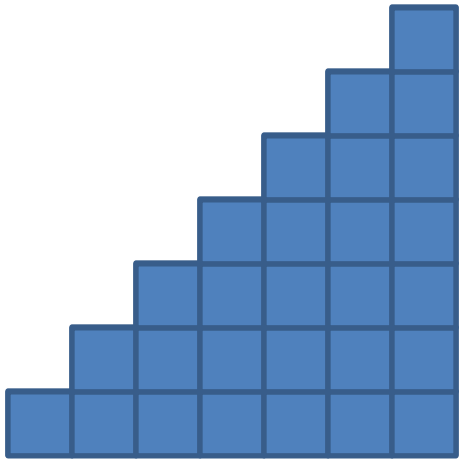
Warm up

Simplify:

$$1 + 2 + 3 + \cdots + (n-1) + n =$$

$$1 + 2 + 3 + \cdots + (n-1) + n = \frac{n(n+1)}{2}$$





Today's Keywords

- Divide and Conquer
- Strassen's Algorithm
- Sorting
- Quicksort

CLRS Readings

- Chapter 4
- Chapter 7

Homeworks

- Hw2 due 11pm Thursday!
  - Programming (use Python or Java!)
  - Divide and conquer
  - Closest pair of points
  - Note: you will need to write a recursive function in:
    - closest_pair.py or
    - ClosestPair.java

Matrix Multiplication

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

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

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

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!

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

Matrix Multiplication D&C

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

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

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

Idea: Use a Karatsuba-like technique on this

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

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

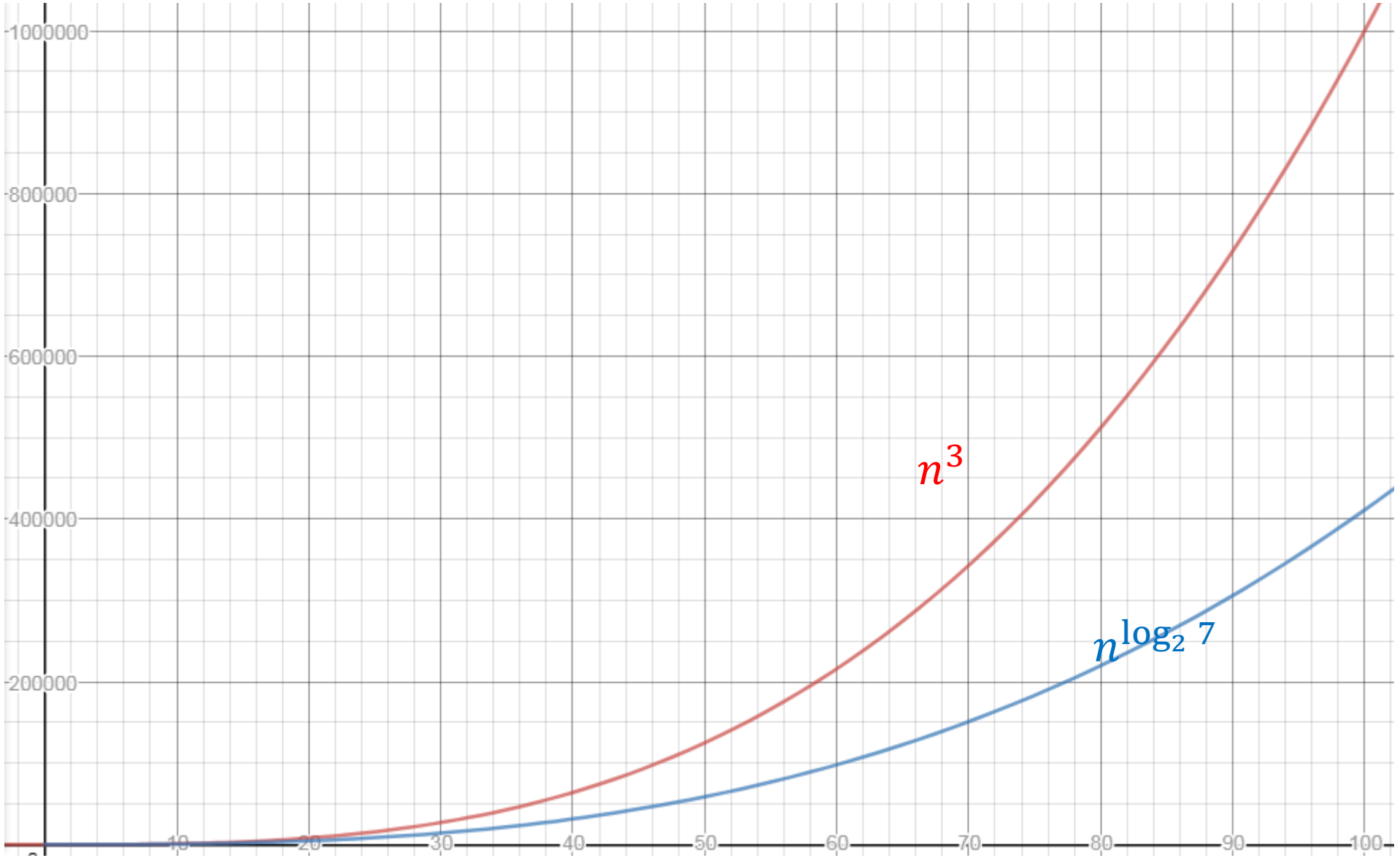
Strassen's Algorithm

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

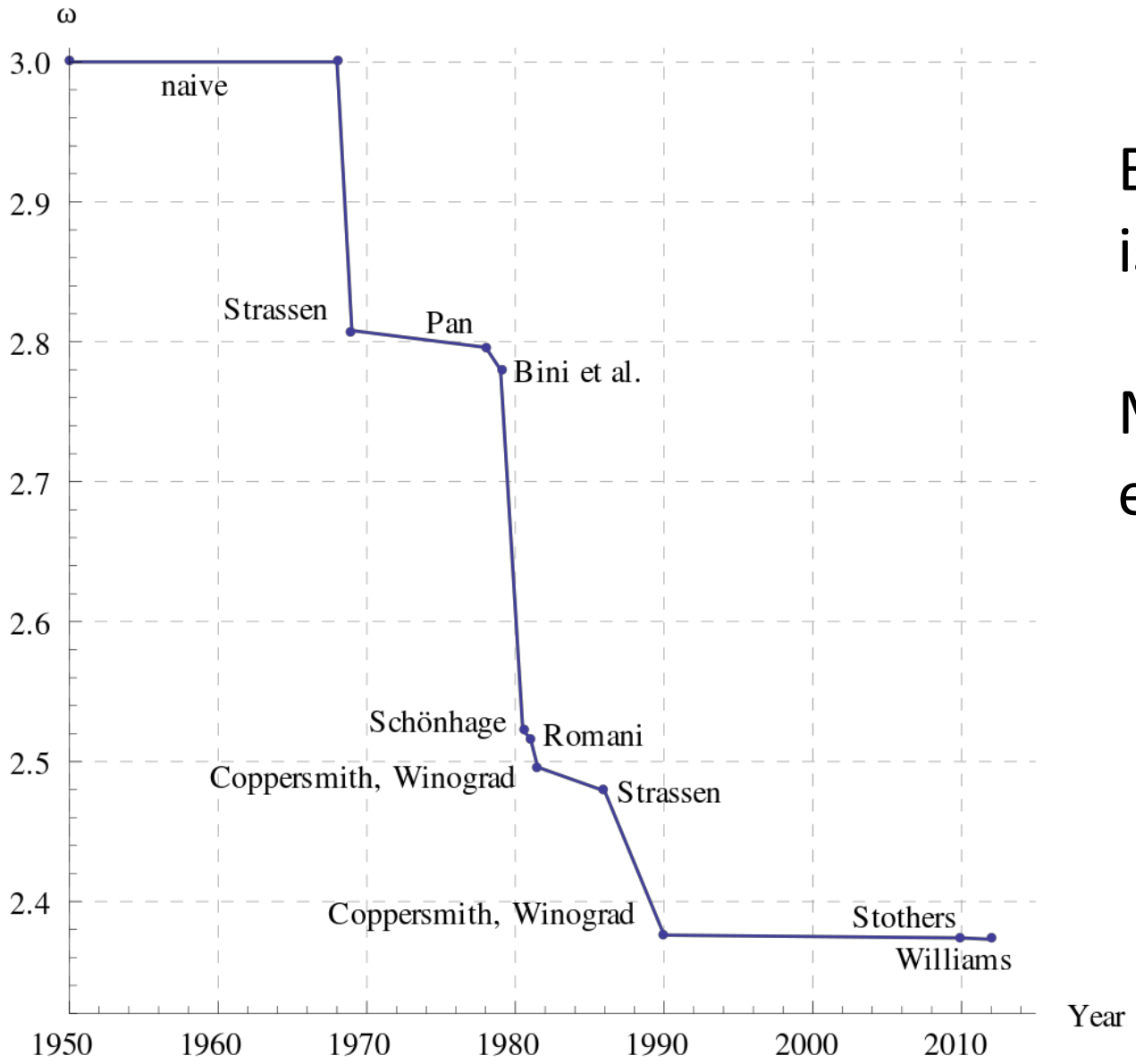
$$a = 7, b = 2, f(n) = \frac{9}{2}n^2$$

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

Divide and Conquer, so far

- Mergesort
- Naïve Multiplication
- Karatsuba
- Closest Pair of Points
- Naïve Matrix-Matrix Multiplication
- Strassen's

What do they have in common?

Divide: Very easy (i.e. $O(1)$)

Combine: Hard work ($\Omega(n)$)

Quicksort

- Like Mergesort:
  - Divide and conquer
  - $O(n \log n)$ run time (kind of…)
- Unlike Mergesort:
  - Divide step is the hard part
  - *Typically* faster than Mergesort

Quicksort

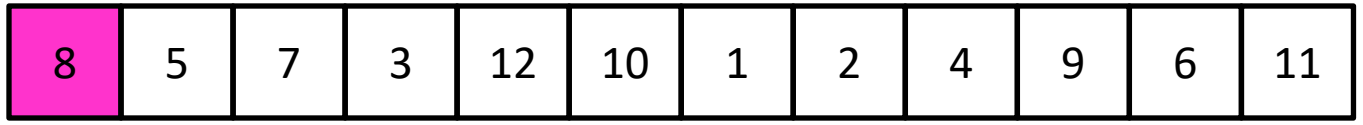
Idea: pick a pivot element, recursively sort two sublists around that element

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

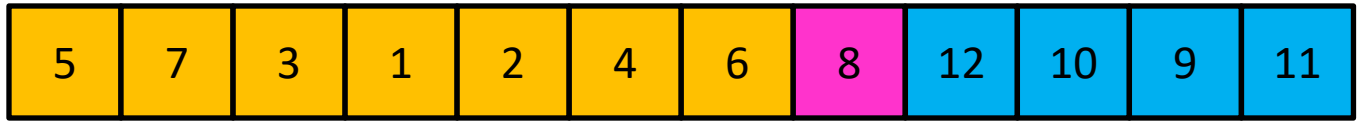
Partition (Divide step)

Given: a list, a pivot p

Start: unordered list



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

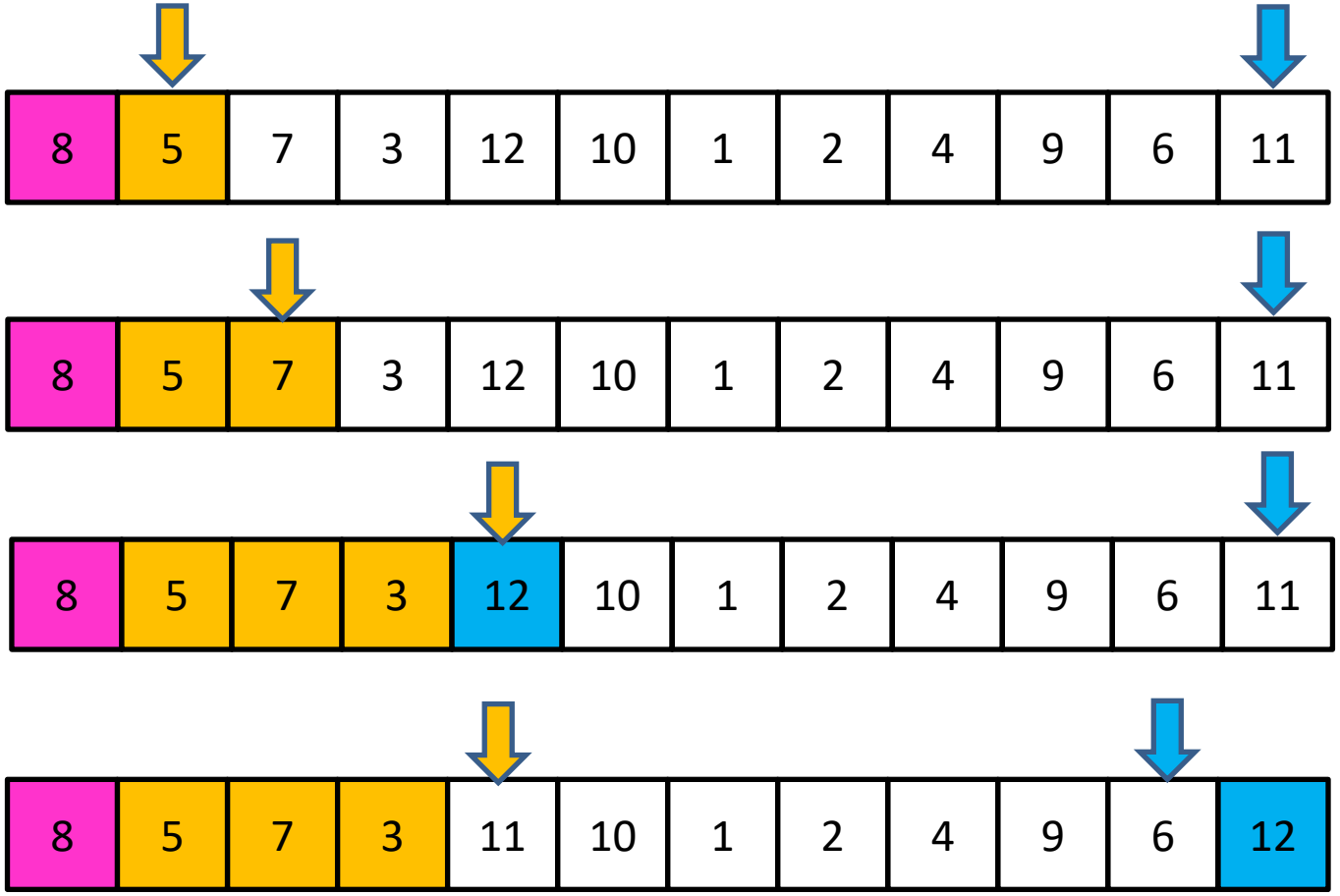


Partition, Procedure

If Begin value $<$ p, move Begin right

Else swap Begin value with End value, move End Left

Done when Begin = End

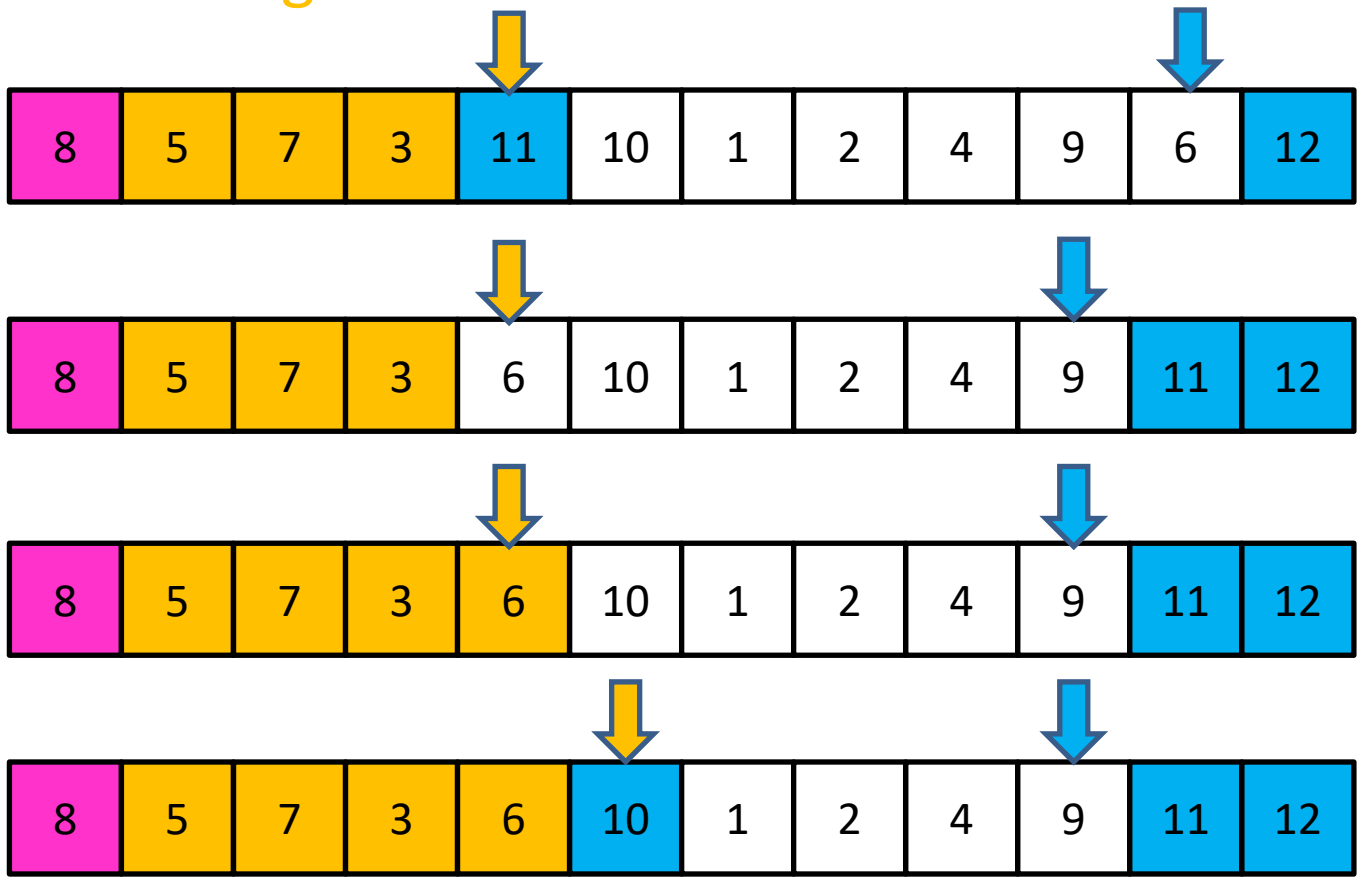


Partition, Procedure

If Begin value $<$ p, move Begin right

Else swap Begin value with End value, move End Left

Done when Begin = End

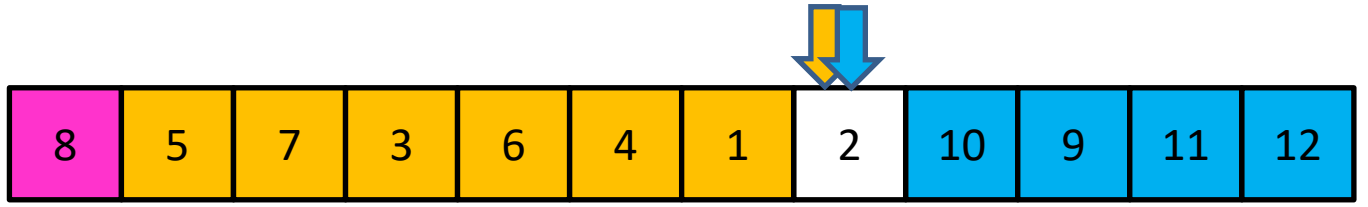


Partition, Procedure

If Begin value $<$ p, move Begin right

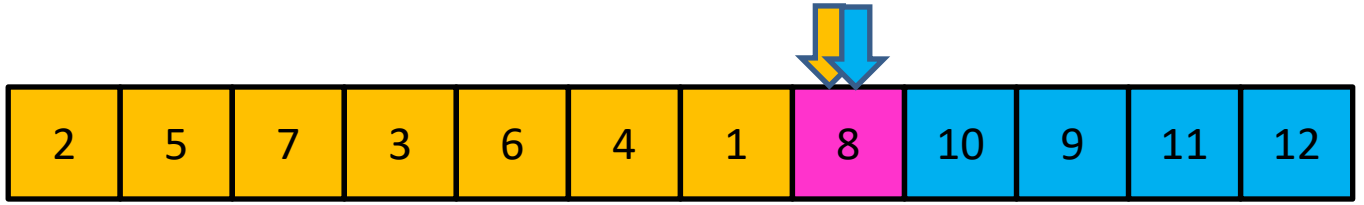
Else swap Begin value with End value, move End Left

Done when Begin = End



Case 1: meet at element $< p$

Swap p with pointer position (2 in this case)

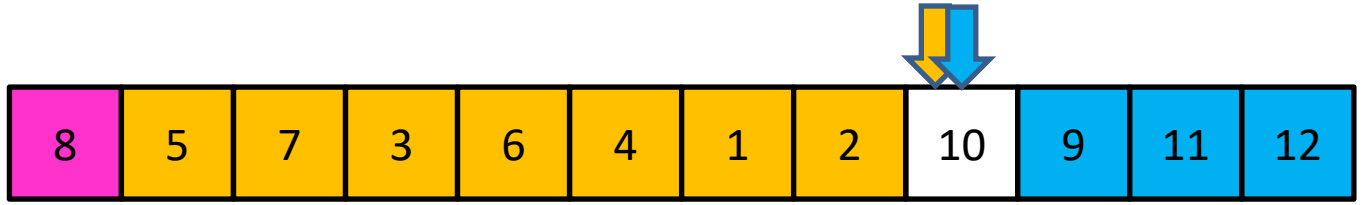


Partition, Procedure

If Begin value $<$ p, move Begin right

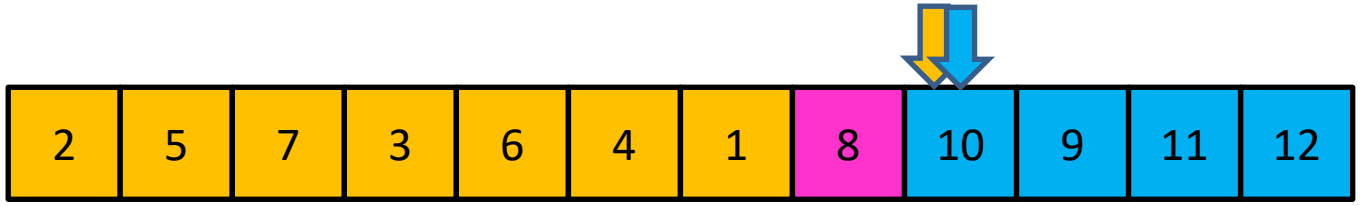
Else swap Begin value with End value, move End Left

Done when Begin = End



Case 2: meet at element $>$ p

Swap p with value to the left (2 in this case)

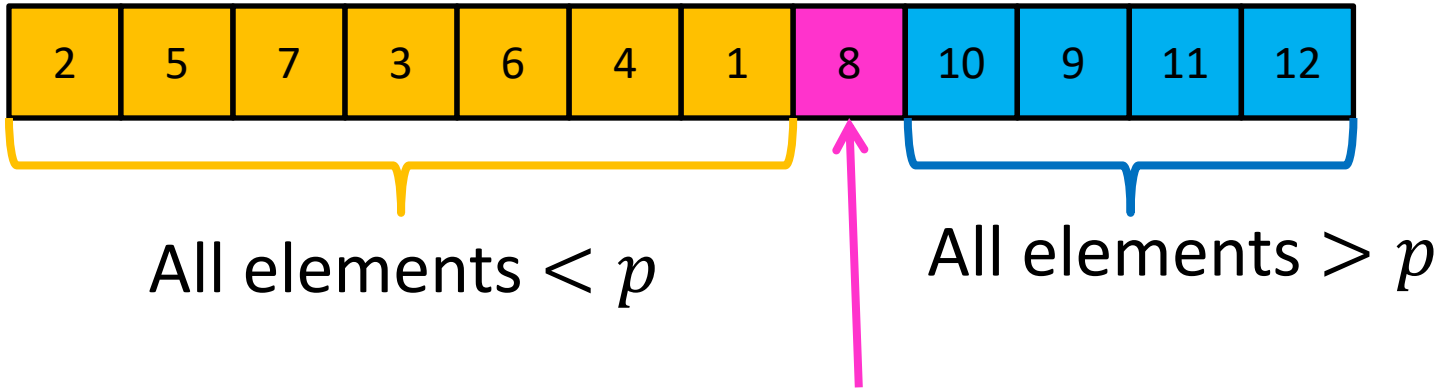


Partition Summary

1. Put p at beginning of list
2. Put a pointer (Begin) just after p, and a pointer (End) at the end of the list
3. While Begin < End:
   1. If Begin value $<$ p, move Begin right
   2. Else swap Begin value with End value, move End Left
4. If pointers meet at element $< p$: Swap p with pointer position
5. Else If pointers meet at element $> p$: Swap p with value to the left

Run time? $O(n)$

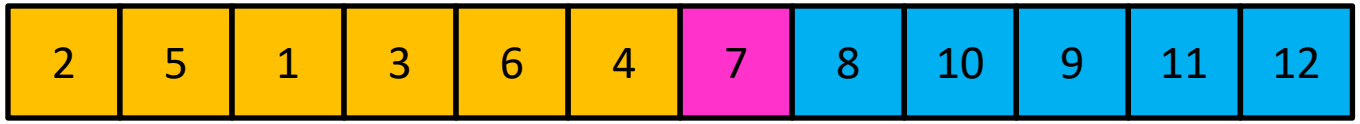
Conquer

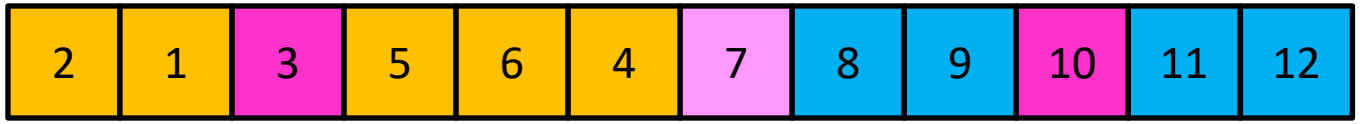


Recursively sort Left and Right sublists

Quicksort Run Time (Best)

If the pivot is always the median:





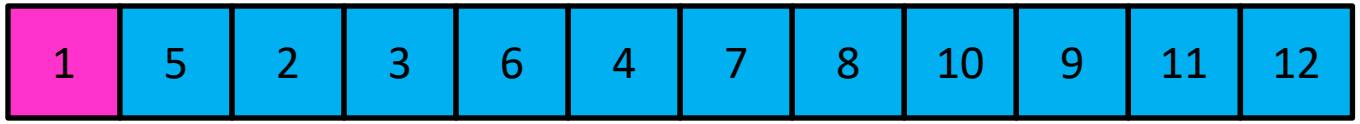
Then we divide in half each time

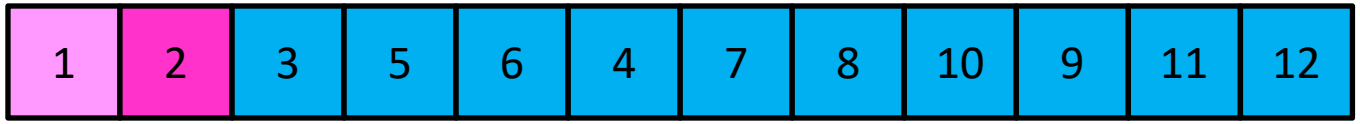
$$T(n) = 2T\left(\frac{n}{2}\right) + n$$

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

Quicksort Run Time (Worst)

If the pivot is always at the extreme:





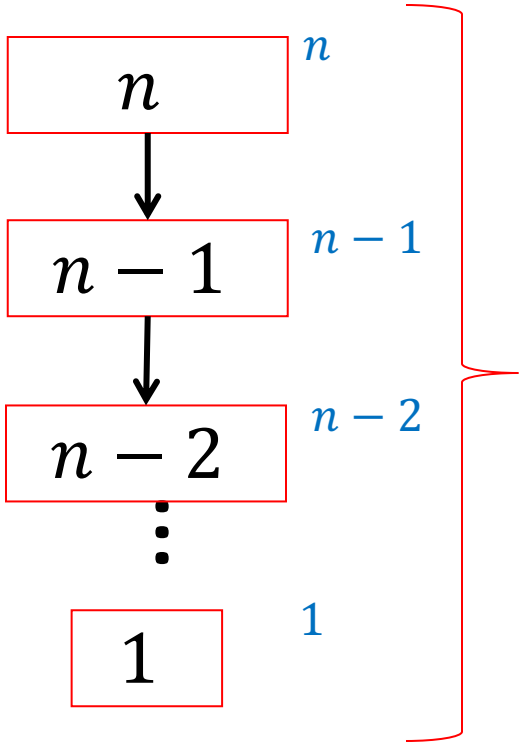
Then we shorten by 1 each time

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

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

Quicksort Run Time (Worst)

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



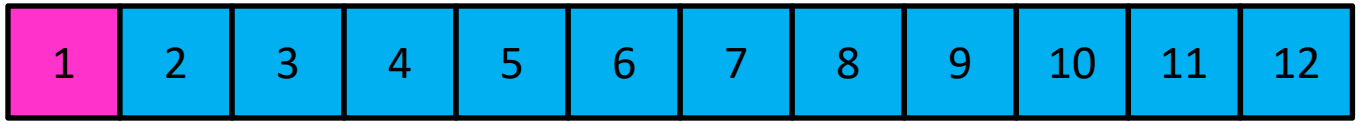
$$T(n) = 1 + 2 + 3 + \cdots + n$$

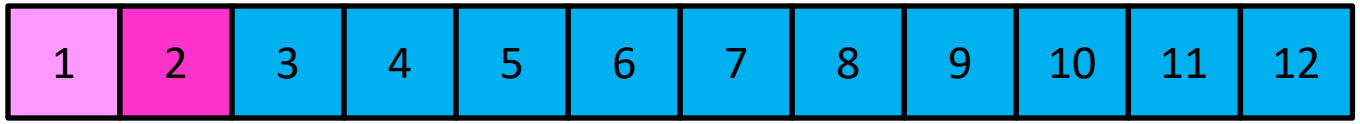
$$T(n) = \frac{n(n+1)}{2}$$

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

Quicksort on a (nearly) Sorted List

First element always yields unbalanced pivot





So we shorten by 1 each time

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

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

How to pick the pivot?

Good Pivot

- What makes a good Pivot?
  - Roughly even split between left and right
  - Ideally: median
- Can we find median in linear time?
  - Yes!
  - 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

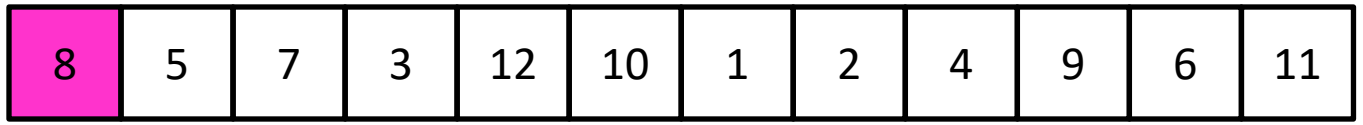
Quickselect

- Finds i^{th} order statistic
- Idea: pick a pivot element, partition, then recurse on 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!

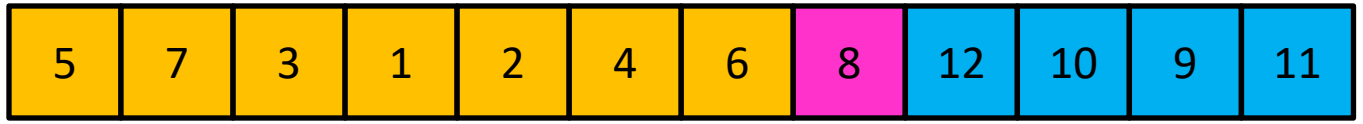
Partition (Divide step)

Given: a list, a pivot value p

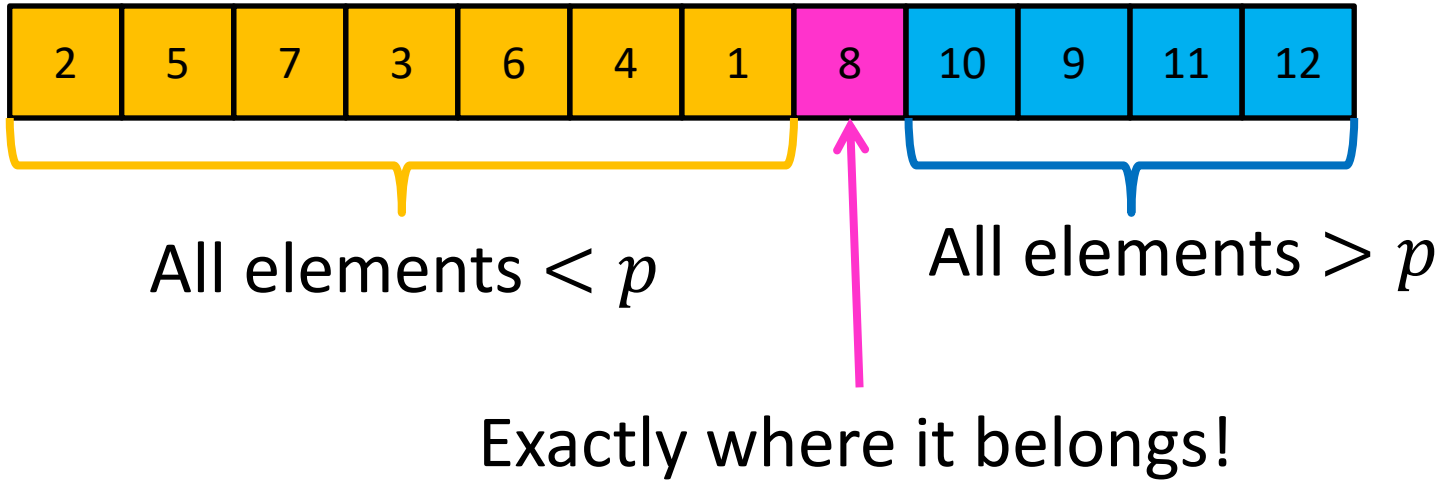
Start: unordered list



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



Conquer

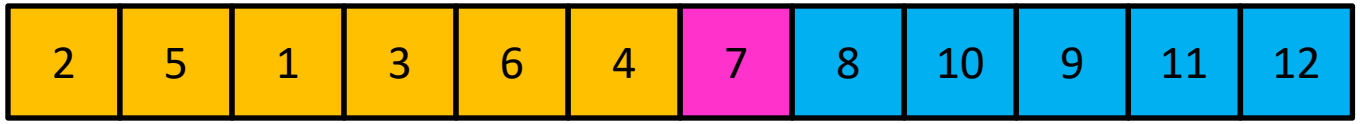


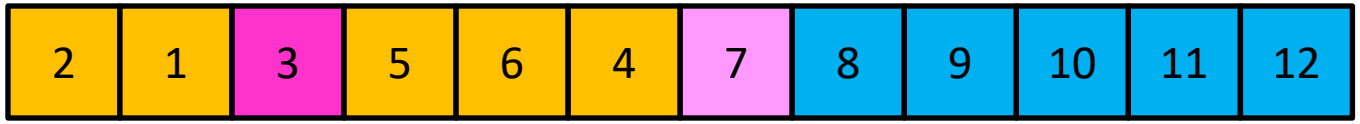
Recurse on sublist that contains index i

(adjust i accordingly if recursing right)

Quickselect Run Time

If the pivot is always the median:





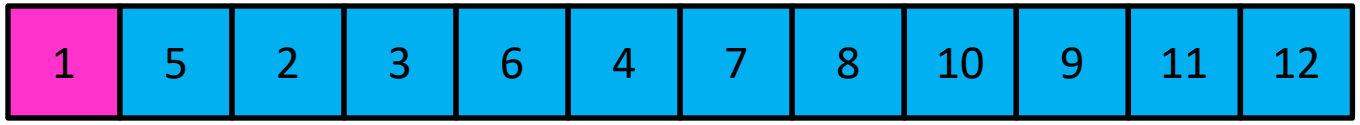
Then we divide in half each time

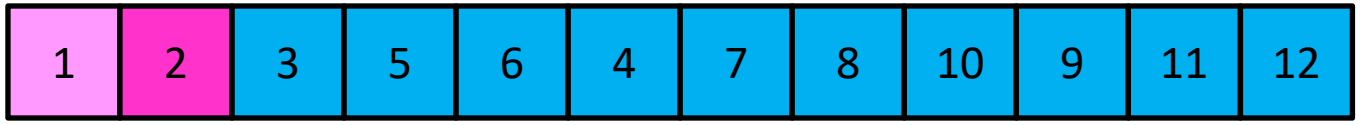
$$S(n) = S\left(\frac{n}{2}\right) + n$$

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

Quickselect Run Time

If the partition is always unbalanced:





Then we shorten by 1 each time

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

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

Good Pivot

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

Déjà vu?

- Here's what's next:
  - An algorithm for finding a "rough" split (Median of Medians)
  - This algorithm uses Quickselect as a subroutine

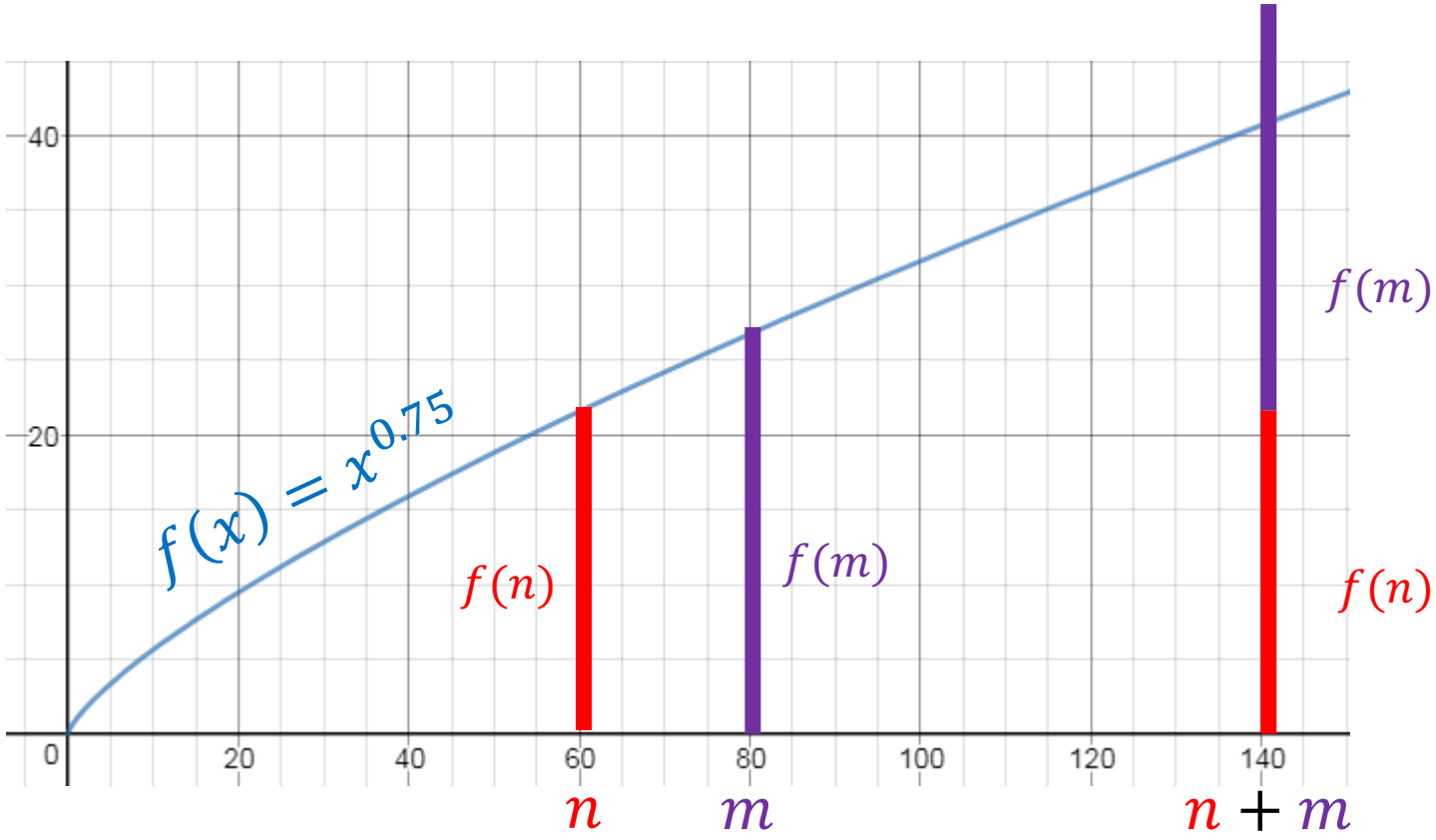
Mental Stretch

Compare $f(n+m)$ with $f(n)+f(m)$

When $f(n)=O(n)$

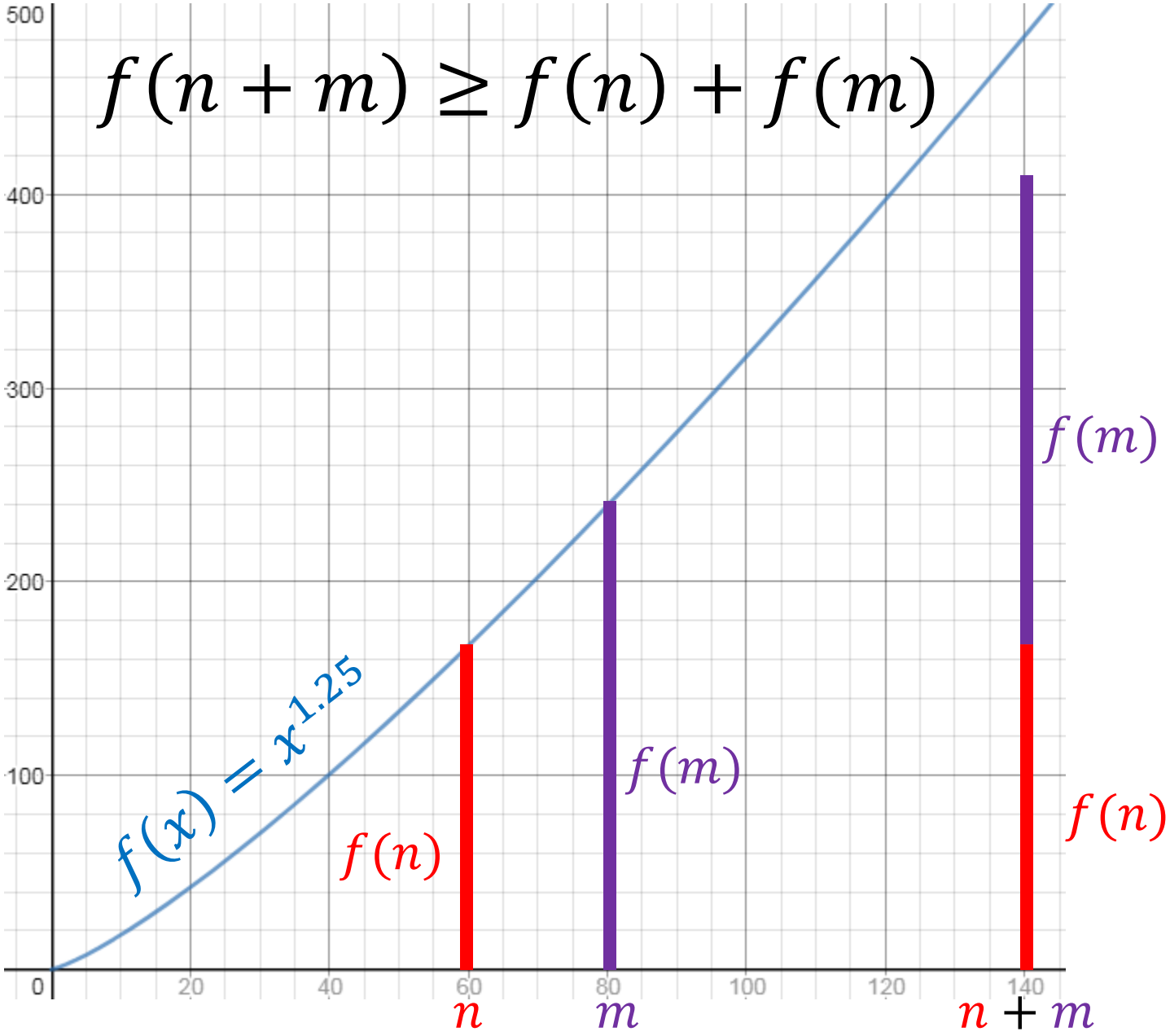
When $f(n)=\Omega(n)$

$f(n) \in O(n)$

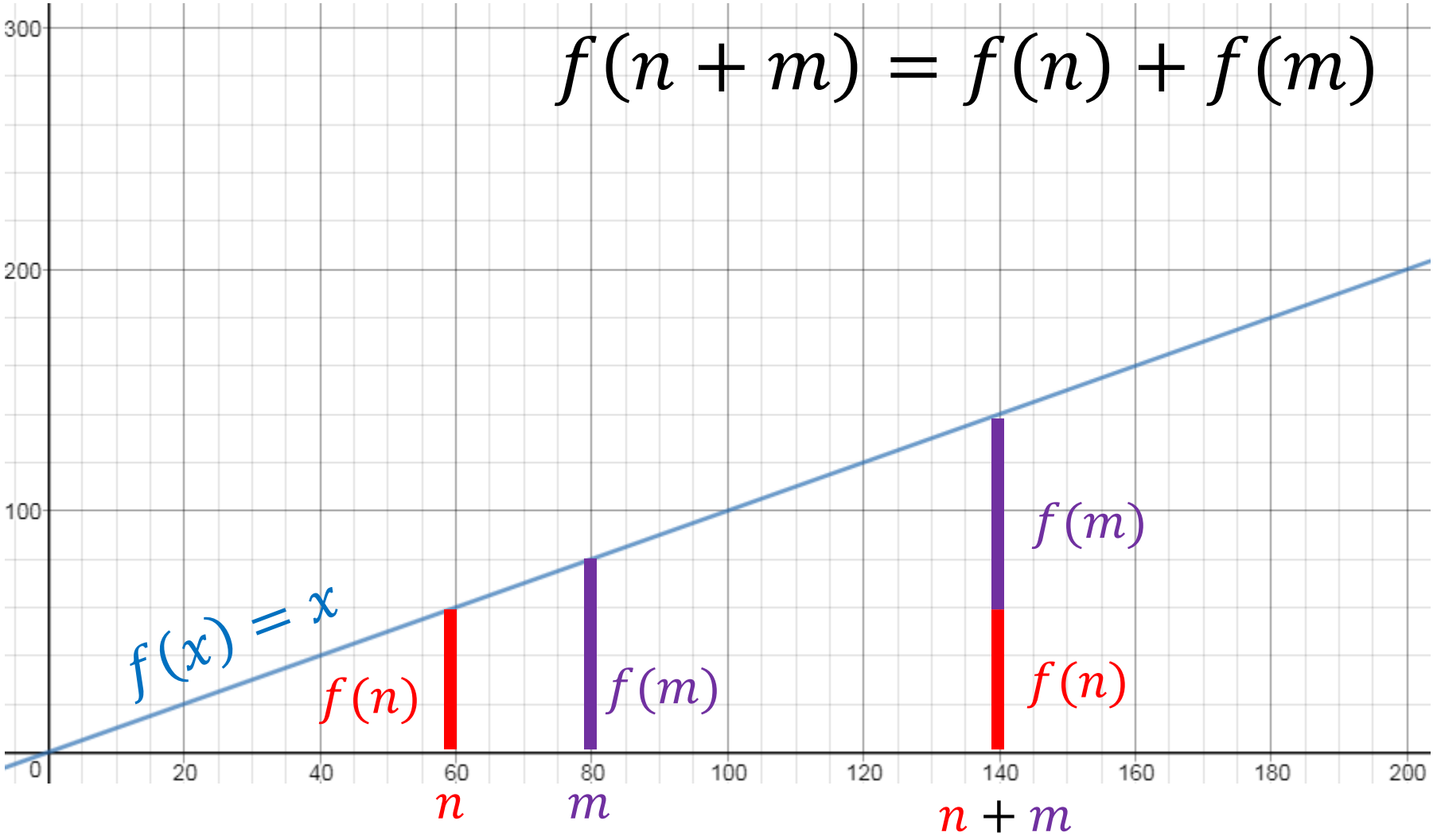


$$f(n+m) \leq f(n) + f(m)$$

$f(n) \in \Omega(n)$


$f(n+m) \geq f(n) + f(m)$
500
400
300
200
100
0
20
40
60
80
100
120
140
$f(x) = x^{1.25}$
$f(n)$
$f(m)$
$f(m)$
$f(n)$
n
m
$n+m$

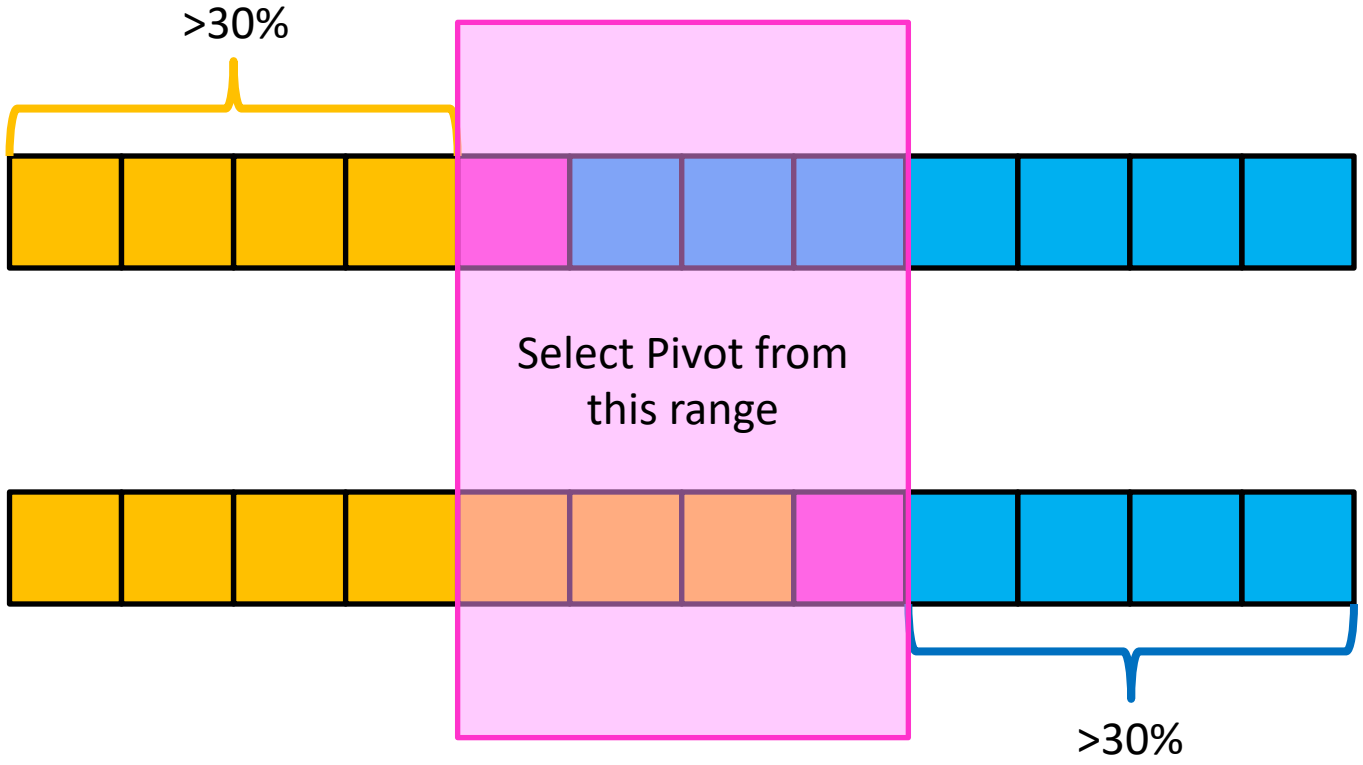
$f(n) = \Theta(n)$


$f(n+m) = f(n) + f(m)$
300
200
100
0
20
40
60
80
100
120
140
160
180
200
$f(x) = x$
$f(n)$
$f(m)$
$f(m)$
$f(n)$
n
m
$n + m$

Good Pivot

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

Or

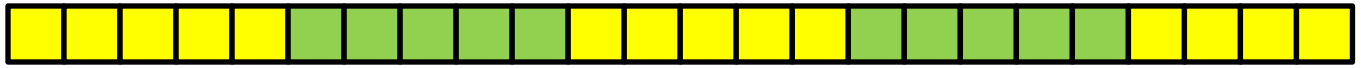


Median of Medians

- Fast way to select a “good” pivot
- Guarantees pivot is greater than 30% of elements and less than 30% of the elements
- Idea: break list into chunks, find the median of each chunk, use the median of those medians

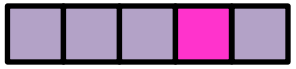
Median of Medians

1. Break list into chunks of size 5



2. Find the median of each chunk

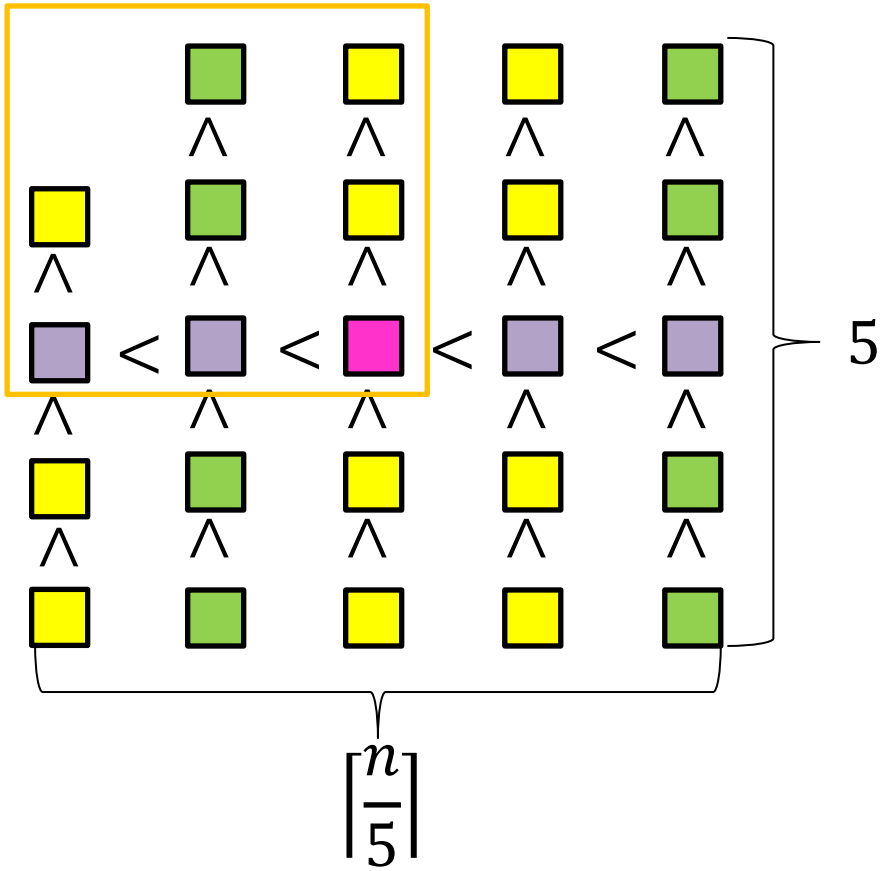
3. Return median of medians (using Quickselect)



Why is this good?

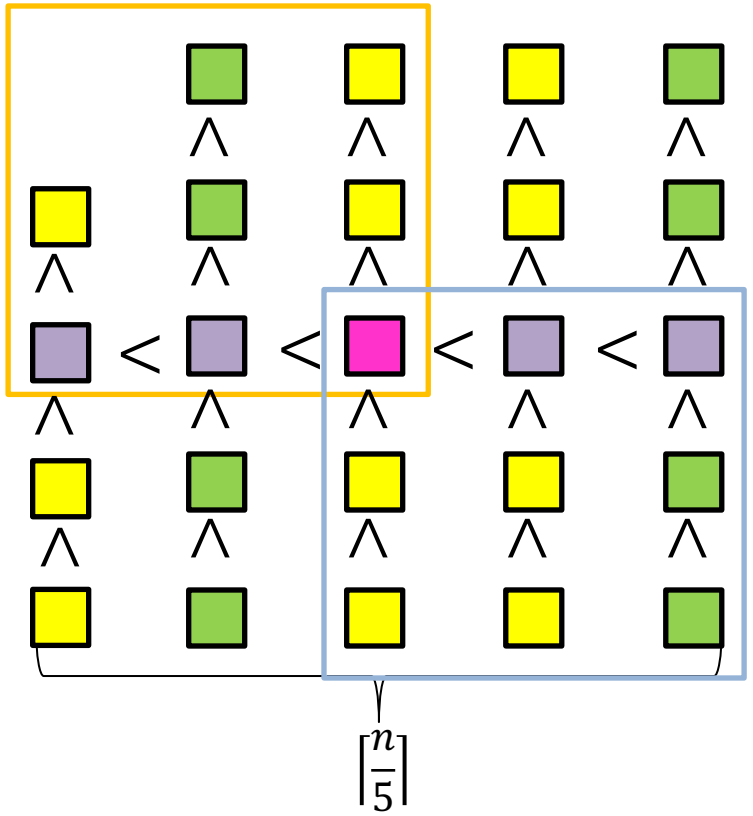
Each chunk sorted, chunks ordered by their medians

MedianofMedians is Greater than all of these



Why is this good?

MedianofMedians is larger than all of these



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 $>$ ■

Quickselect

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