CS4102 Algorithms

Spring 2019

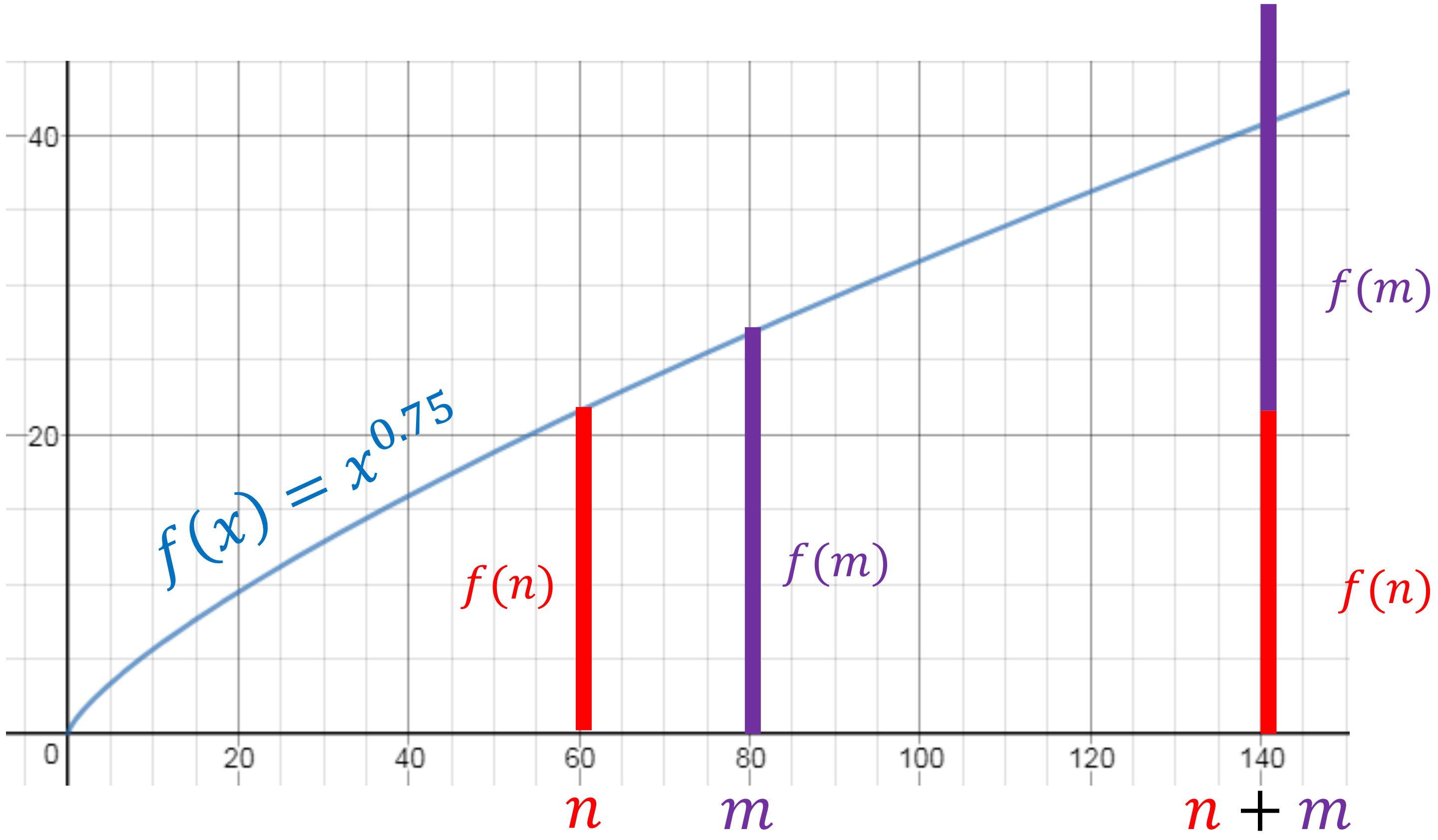
Warm up

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

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

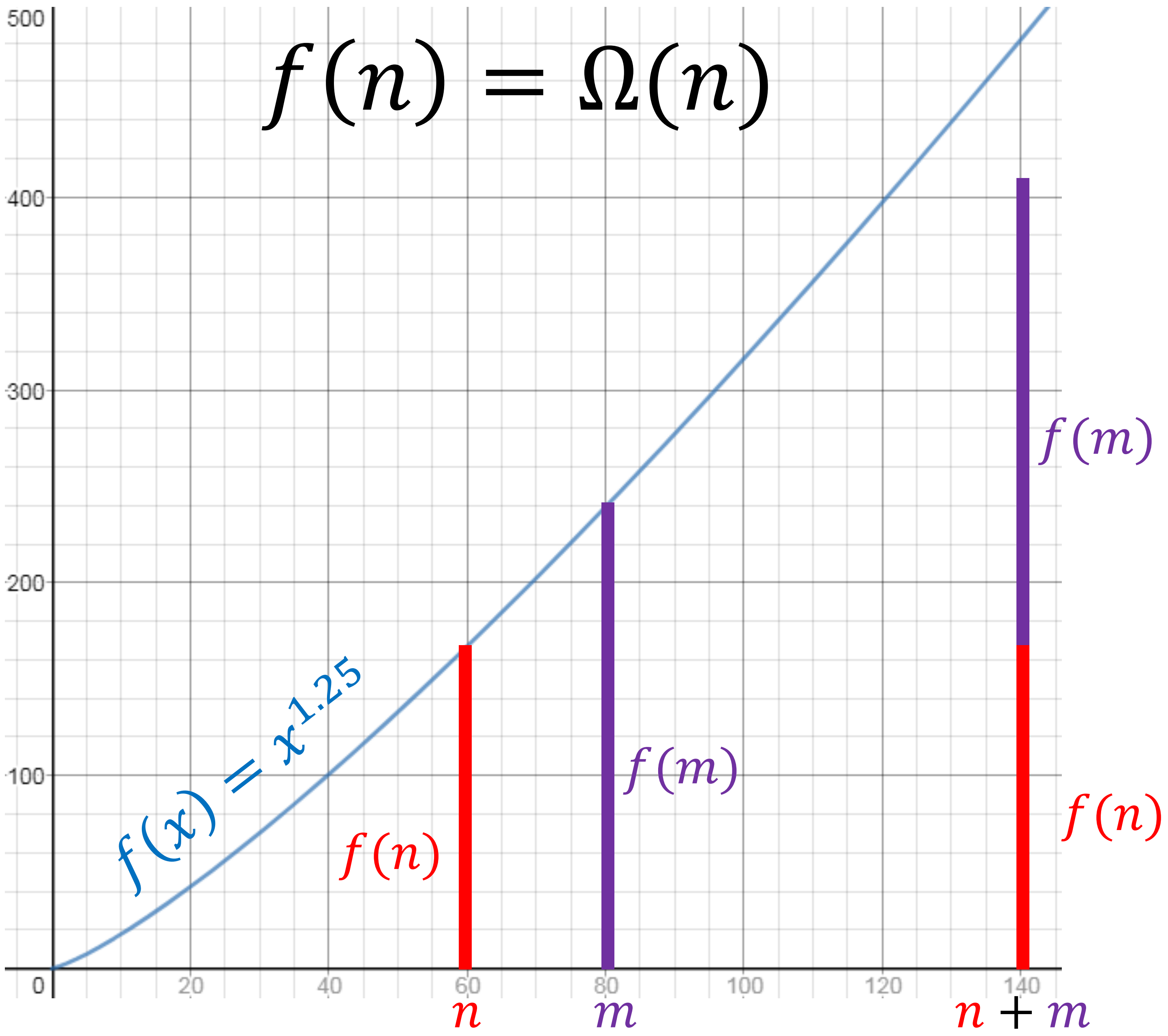
When $f(n) = \Omega(n)$

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



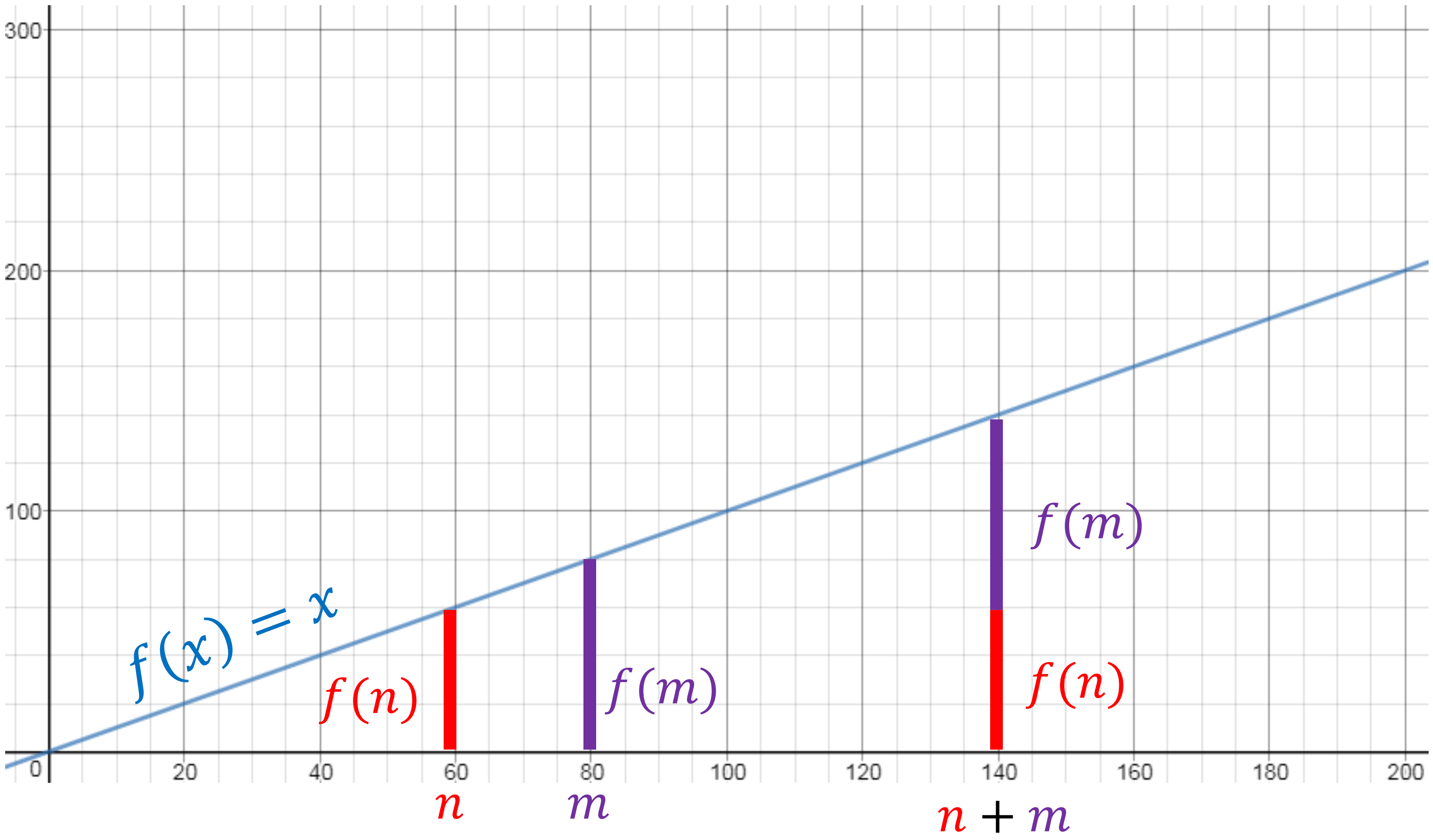
$$f(n+m) \leq f(n) + f(m)$$

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



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

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



$$f(n + m) = f(n) + f(m)$$

Today's Keywords

- Divide and Conquer
- Sorting
- Quicksort
- Median
- Order statistic
- Quickselect
- Median of Medians

CLRS Readings

- Chapter 7

Homeworks

- Hw2 due 11pm Wednesday!
  - Programming (use Python or Java!)
  - Divide and conquer
  - Closest pair of points
- Hw3 released tonight!
  - Divide and conquer
  - Written (use LaTeX!)

Office Hours Wednesday

- Slight shift in my office hours Wednesday
  - 10-11am, 12-12:30pm
  - Scheduling conflict at 11am

Quicksort

- Idea: pick a pivot element, recursively sort two sublists around that element
- Divide: select an 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

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

Quicksort 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

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

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

Quicksort 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

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

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

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

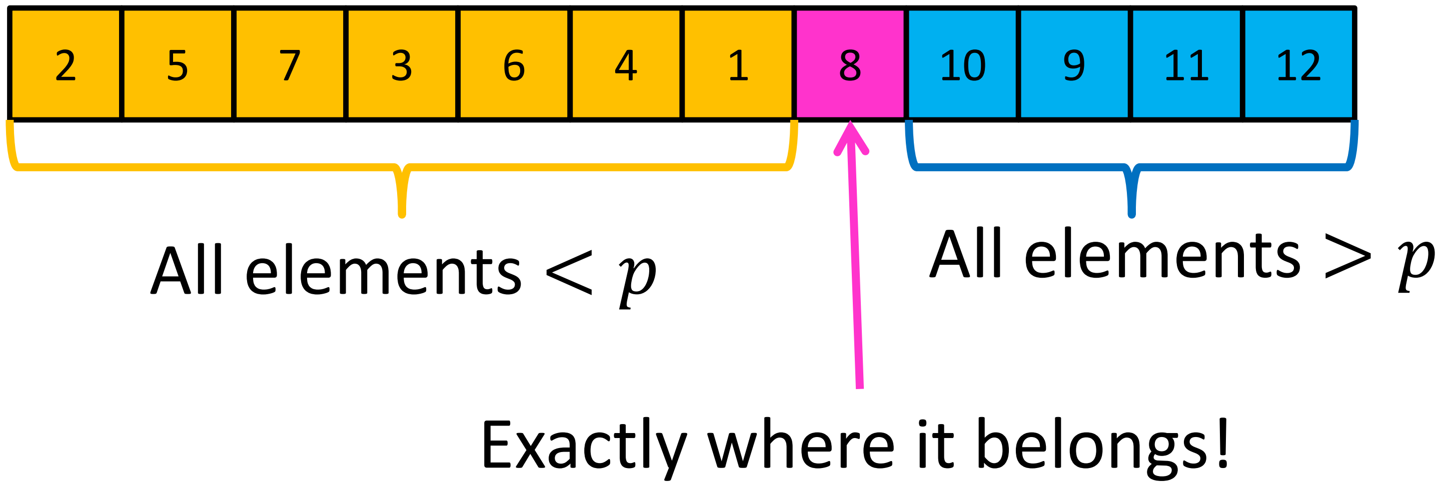
Start: unordered list

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

Conquer



Recurse on sublist that contains index i

(add index of the pivot to i if recursing right)

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

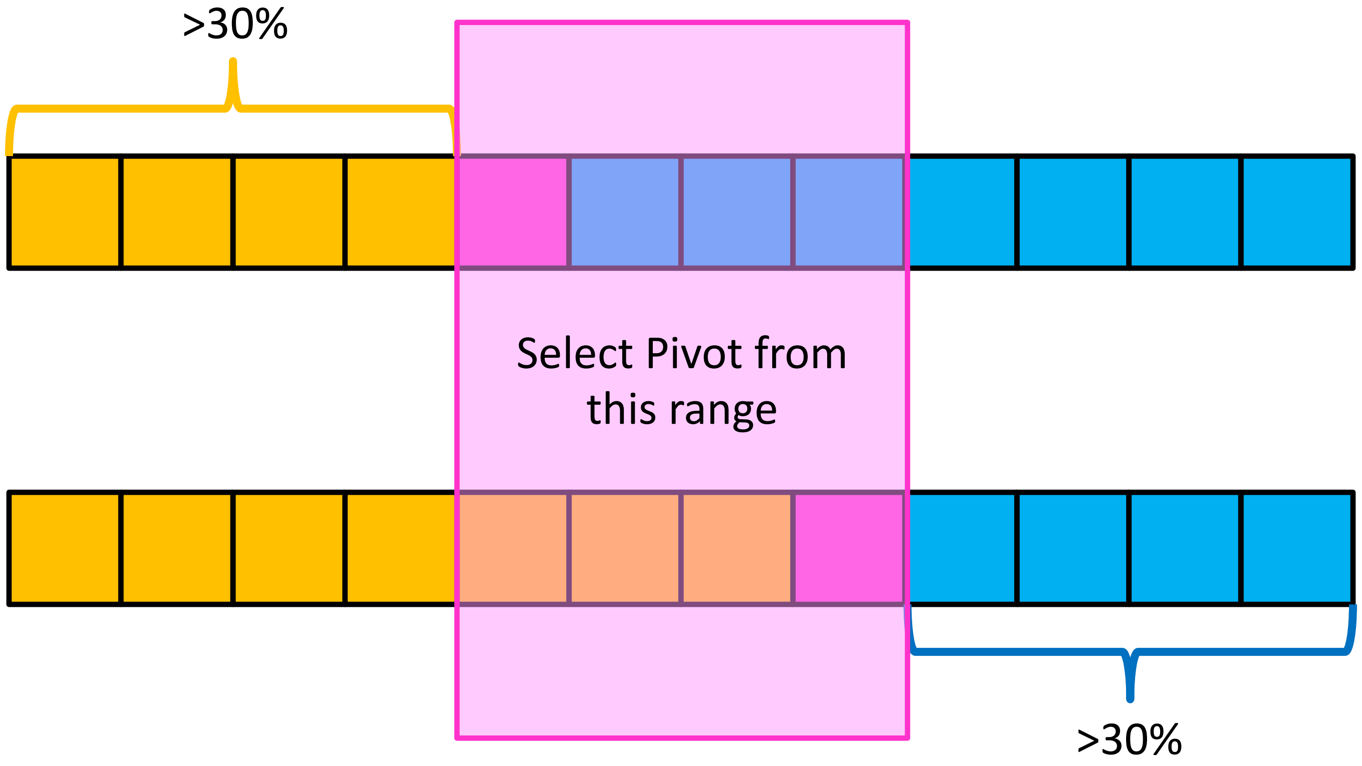
- What makes a good Pivot?
  - Roughly even split between left and right
  - Ideally: median

Déjà vu?

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

Good Pivot

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



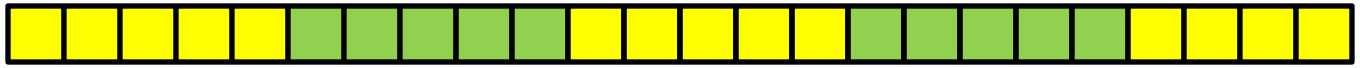
Or

Median of Medians

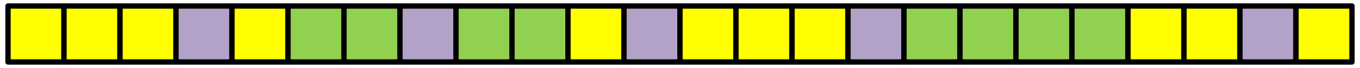
- Fast way to select a “good” pivot
- Guarantees pivot is greater than 30% of elements and less than 30% of the elements
- Idea: break list into chunks, find the median of each chunk, use the median of those medians

Median of Medians

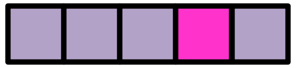
1. Break list into chunks of size 5



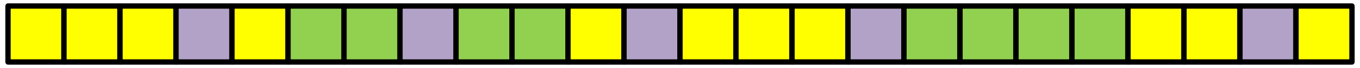
2. Find the median of each chunk



3. Return median of medians (using Quickselect)

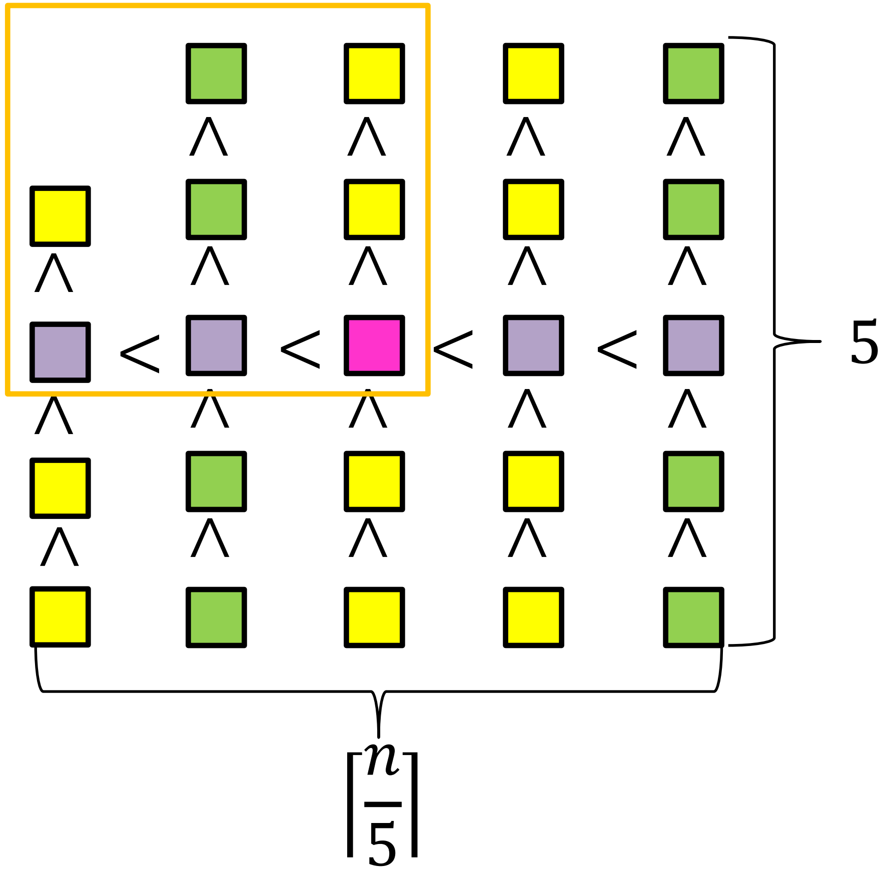


Why is this good?



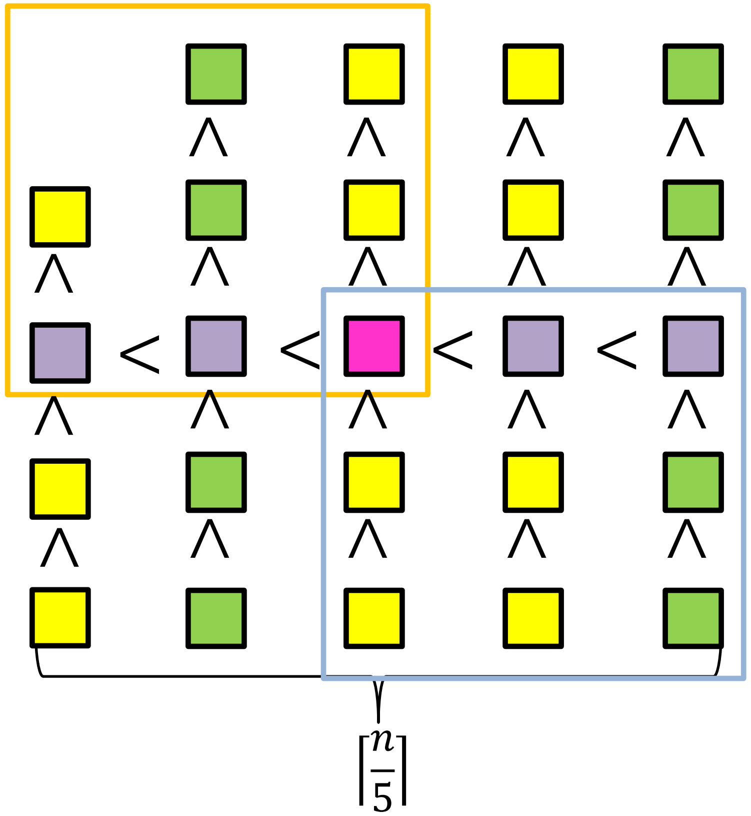
Each chunk sorted, chunks ordered by their medians

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

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

Similarly:

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

Quickselect

- 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

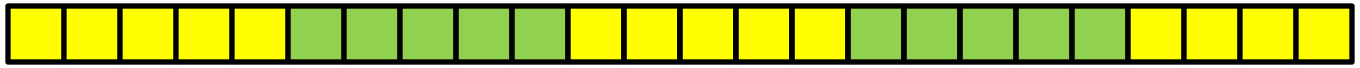
$$\leq S\left(\frac{7}{10}n\right)$$

- Combine: Nothing!

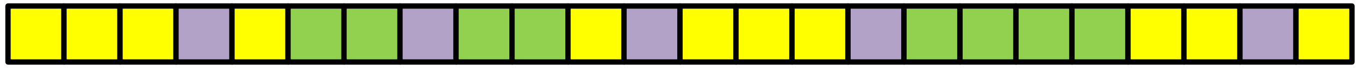
$$S(n) \leq S\left(\frac{7}{10}n\right) + M(n) + \Theta(n)$$

Median of Medians, Run Time

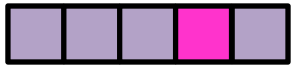
1. Break list into chunks of 5 $\Theta(n)$



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



3. Return median of medians (using Quickselect)



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

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

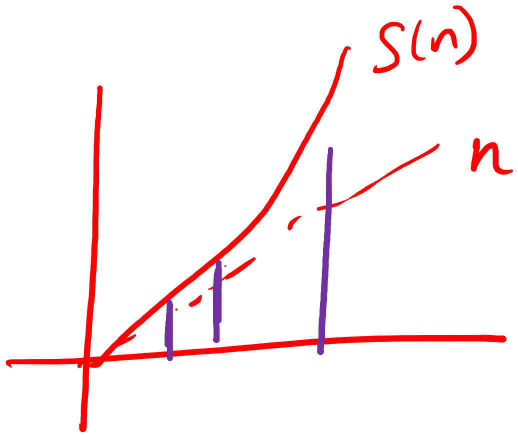
Quickselect

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

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

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

$$= S\left(\frac{7n}{10}\right) + S\left(\frac{2n}{10}\right) + \Theta(n)$$



$$\leq S\left(\frac{9n}{10}\right) + \Theta(n)$$ Because $S(n) = \Omega(n)$

Master theorem Case 3!

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

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

Phew! Back to Quicksort

- Using Quickselect, with a median-of-medians partition:

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
- Approach has very large constants
  – If you really want $\Theta(n \log n)$, better off using MergeSort
- Better approach: Random pivot
  – Very small constant (very fast algorithm)
  – Expected to run in $\Theta(n \log n)$ time
    - Why? Unbalanced partitions are very unlikely