CS4102 Algorithms

Spring 2022

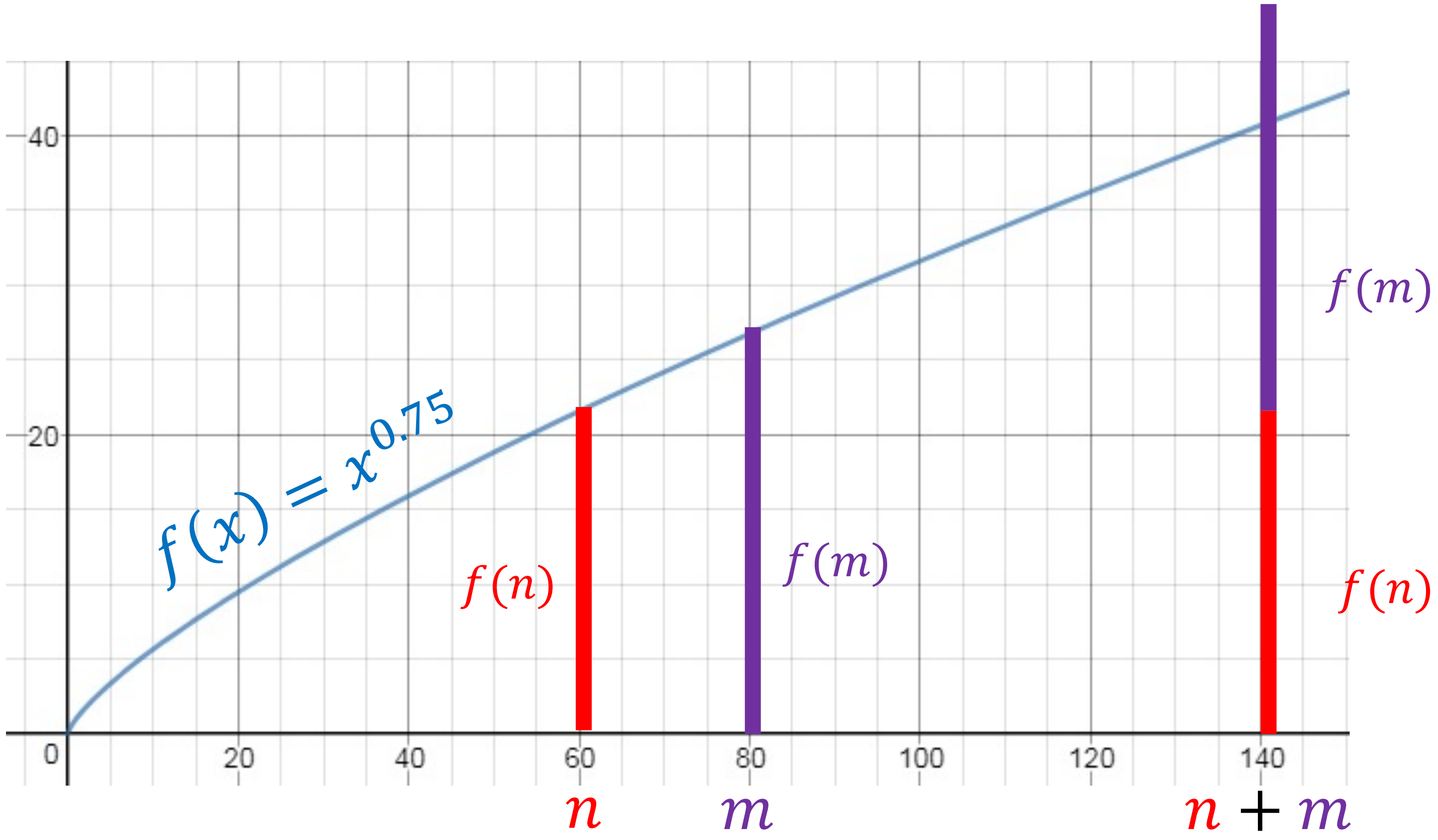
Warm up

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

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

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

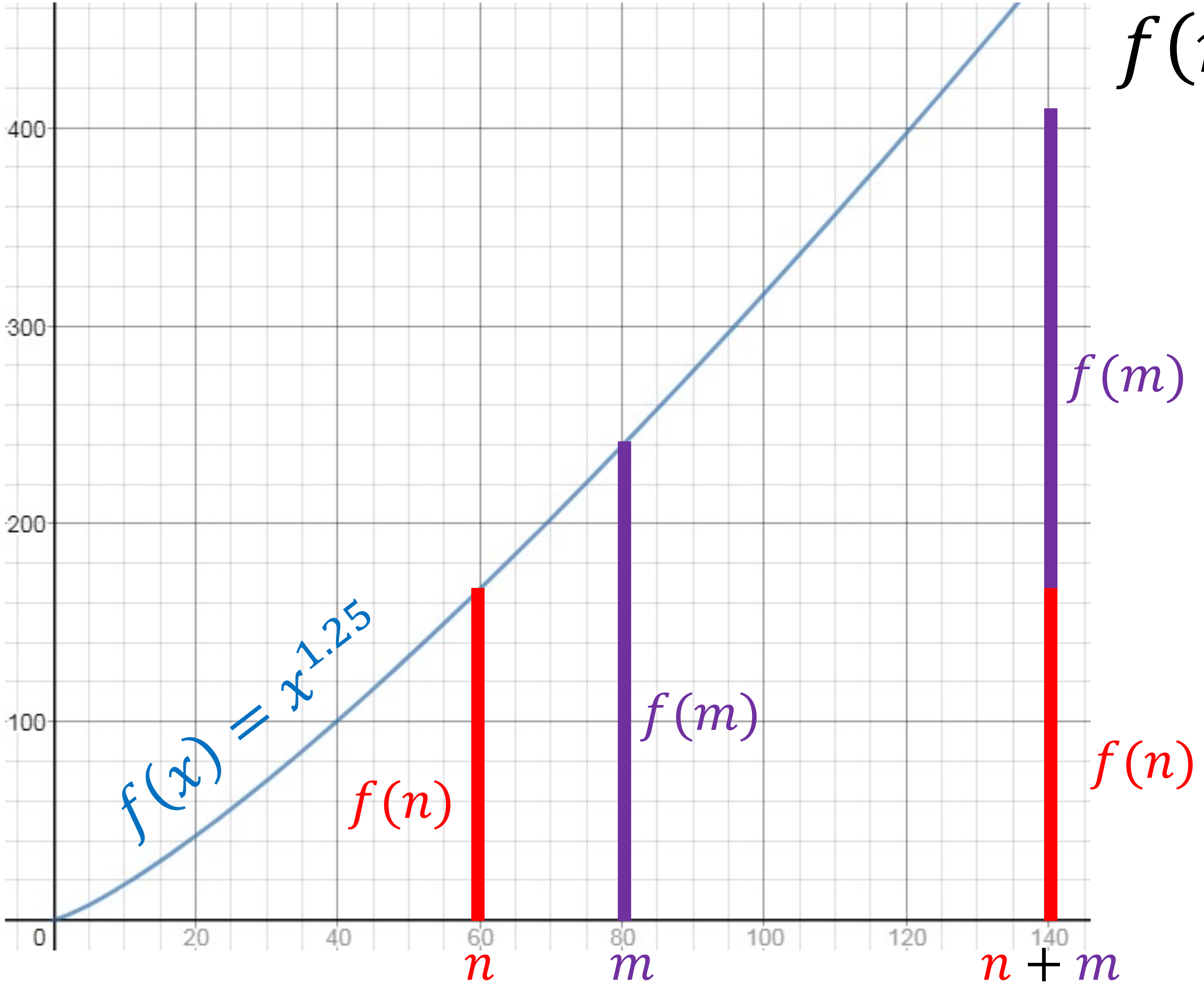
$f(n) \in O(n)$



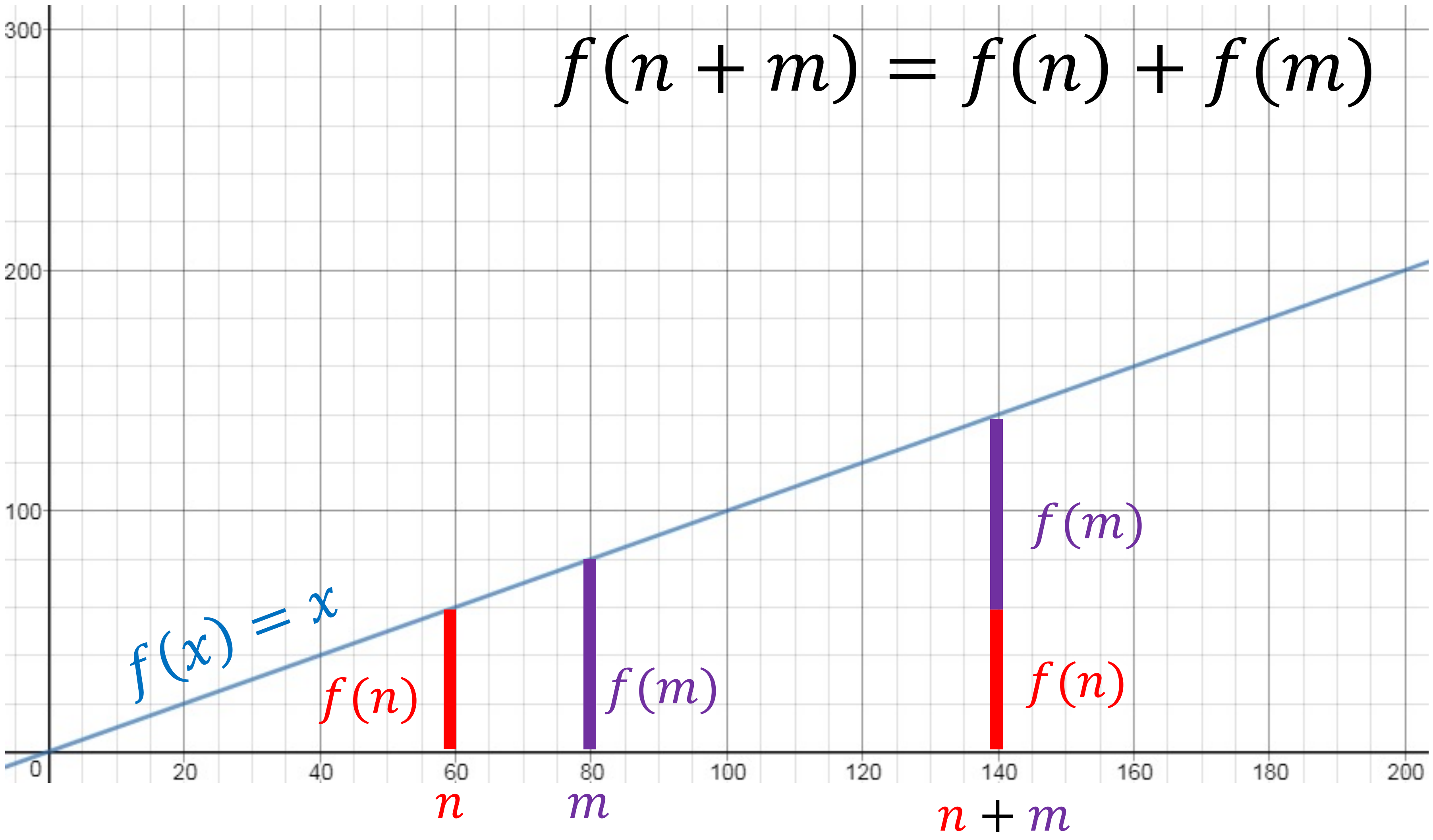
$$f(n+m) \leq f(n) + f(m)$$

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

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



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


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

Announcements

- Homework schedule posted on course website
  - Unit A Basic HW2 now available
  - Unit A Advanced and Programming HW coming soon
- TA Office Hours
  - 7-10pm Sun-Thurs in Ols 011
  - Online hours also available
- Unit A Exam: Tuesday, February 22, in class

Today’s Keywords

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

CLRS Ch 4, 7

Divide and Conquer, so far

What do they have in common?

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

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

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

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

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

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

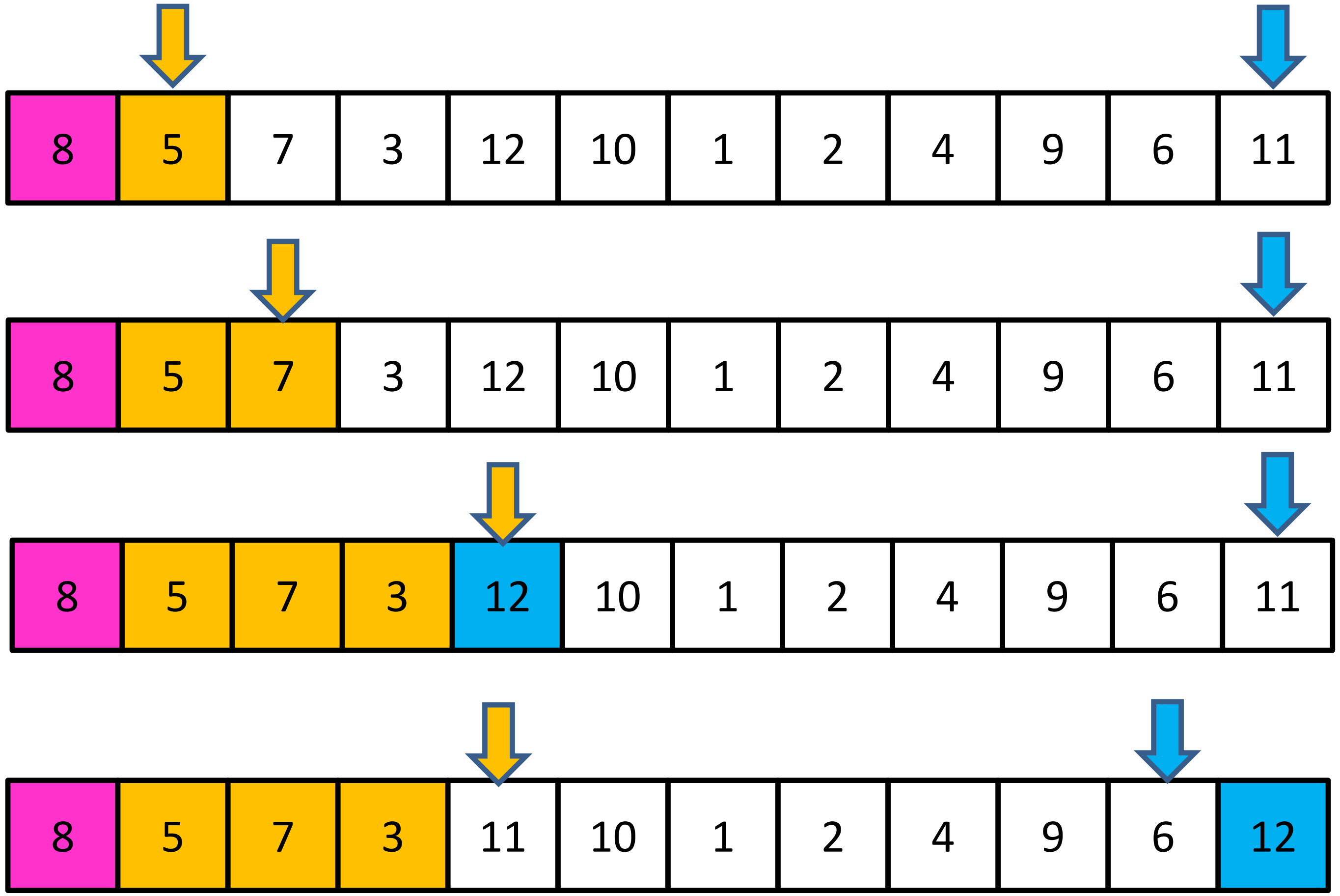
5	7	3	1	2	4	6	8	12	10	9	11

Partition, Procedure

If Begin value $< \ p$, move Begin right

Else swap Begin value with End value, move End Left

Done when Begin = End

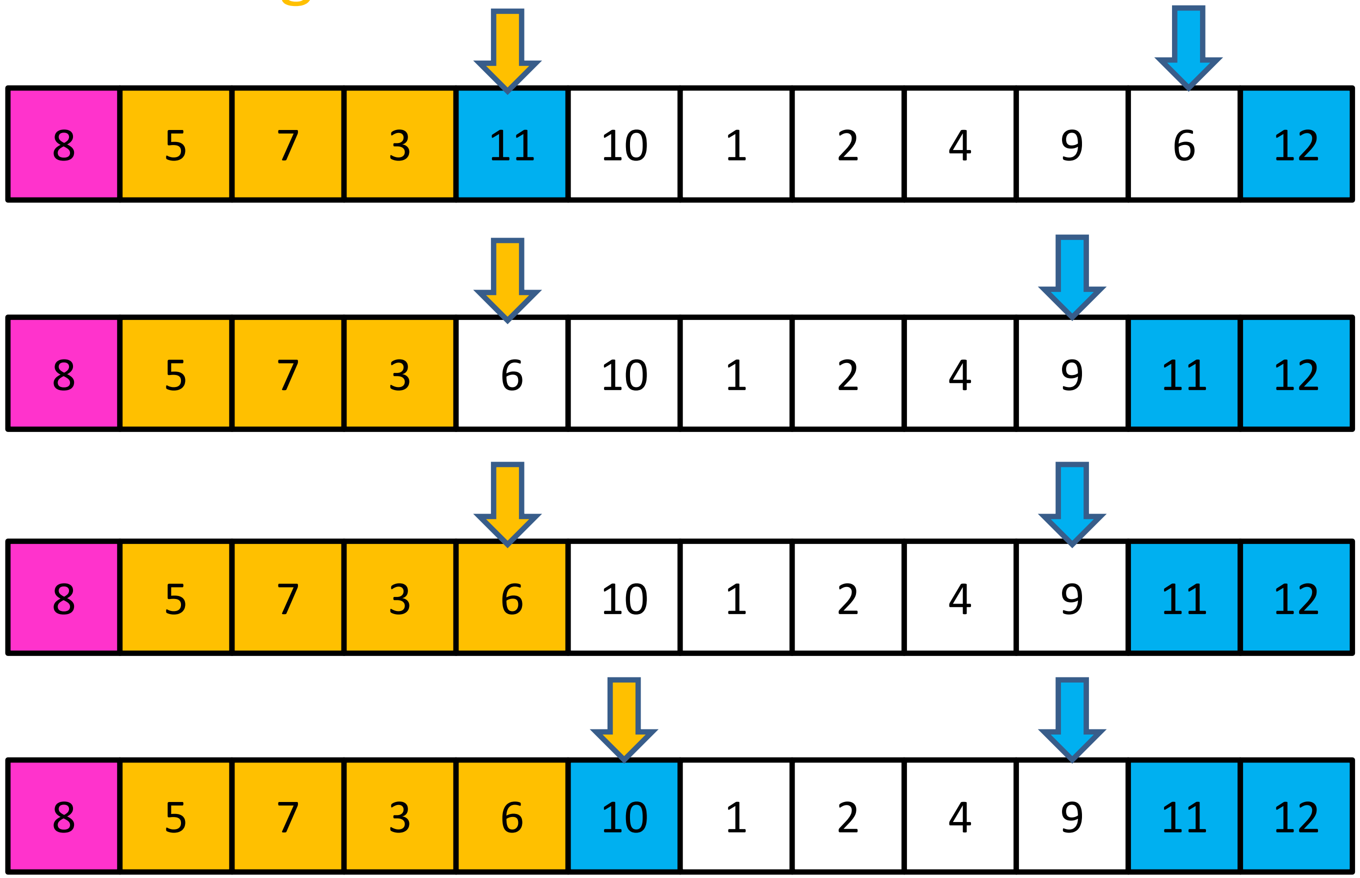


Partition, Procedure

If Begin value $<\ p$, move Begin right

Else swap Begin value with End value, move End Left

Done when Begin = End

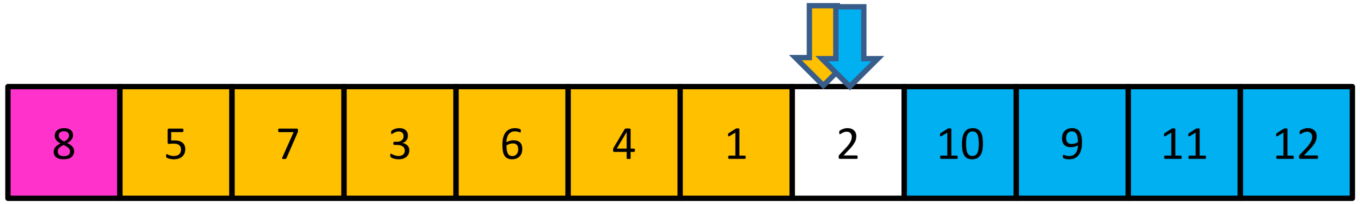


Partition, Procedure

If Begin value $<$ p, move Begin right

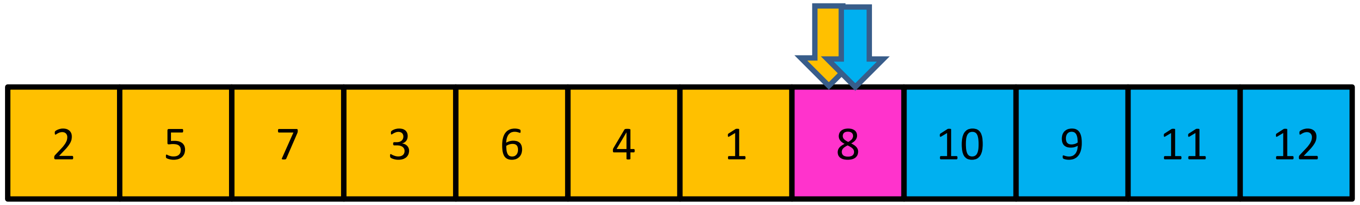
Else swap Begin value with End value, move End Left

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

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

Case 2: meet at element $> p$

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

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

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

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

All elements $< p$

All elements $> p$

Exactly where it belongs!

Recursively sort Left and Right sublists

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

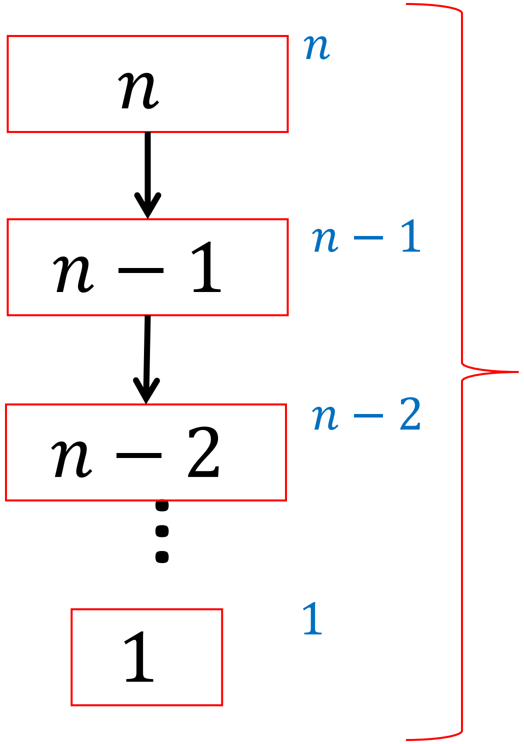
Then we shorten by 1 each time

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

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

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

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

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

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
- CLRS, Section 9.1
  - **Selection problem**: Given a list of distinct numbers and value *i*, find value *x* in list that is larger than exactly *i-1* list elements

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

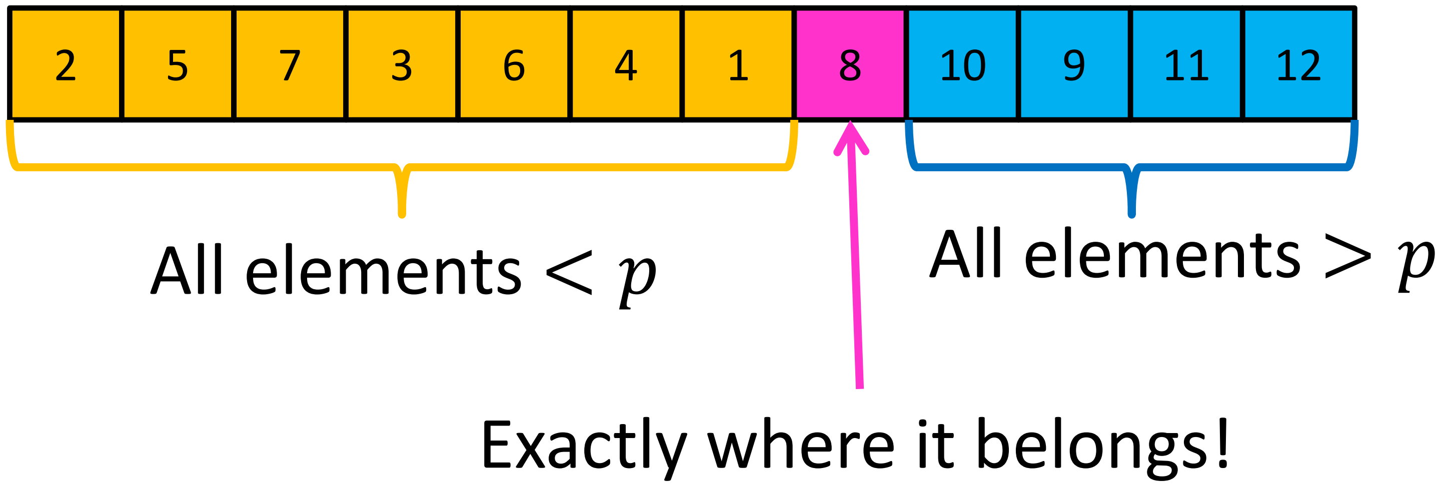
Start: unordered list

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

(adjust i accordingly if recursing right)

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* parameter 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

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

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

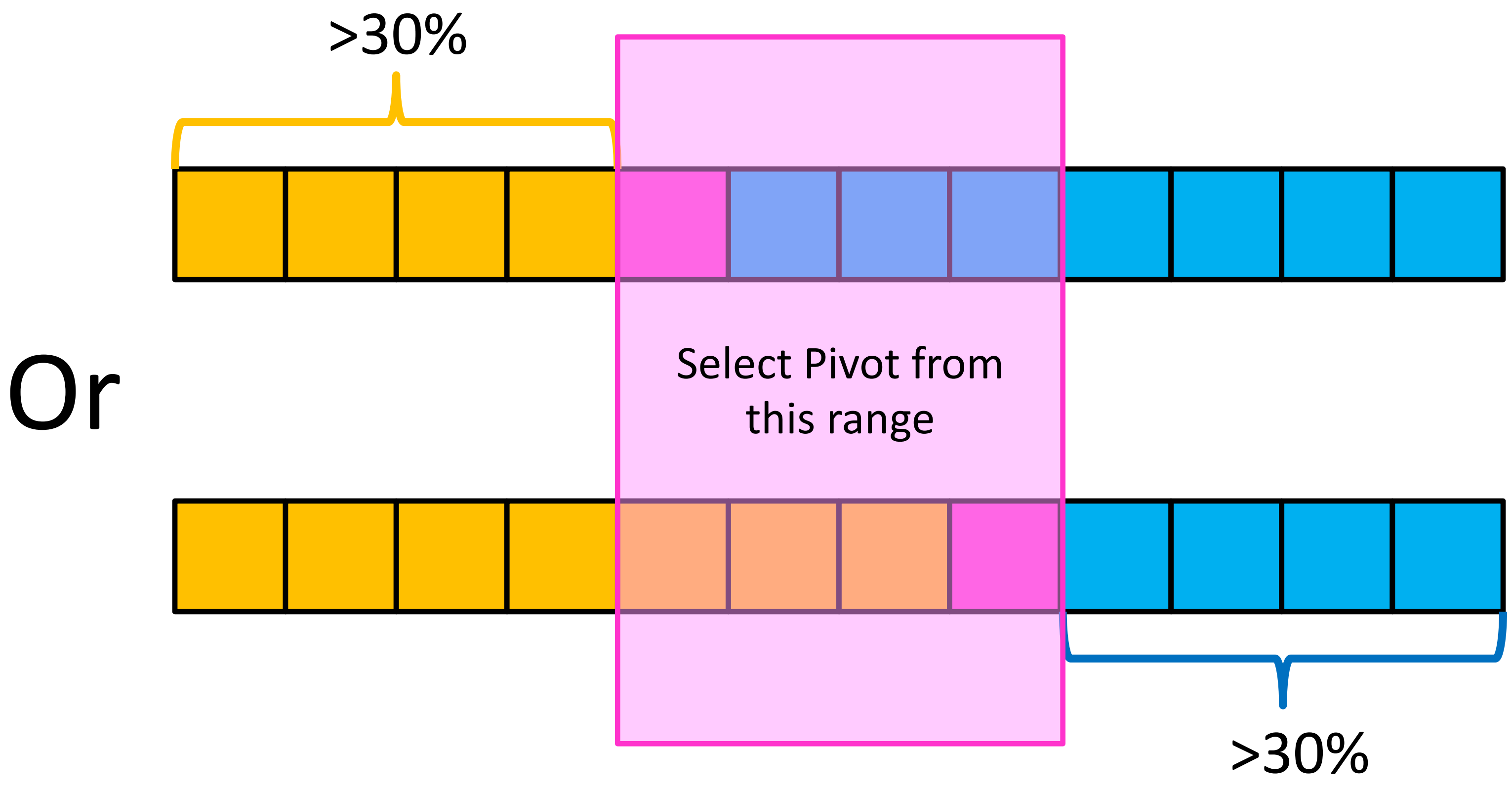
Good Pivot for Quickselect

Déjà vu?

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

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

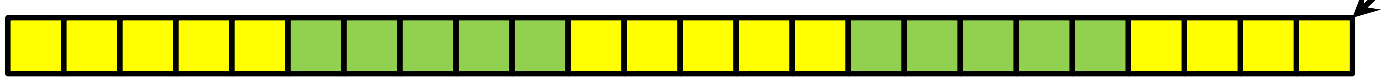
Median of Medians

- 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

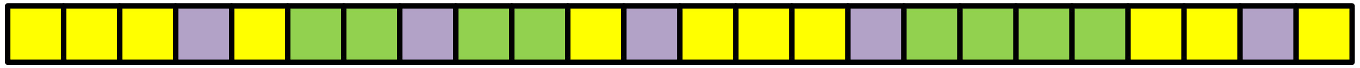
Median of Medians

1. Break list into chunks of size 5

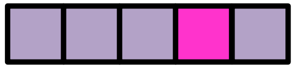
List could be long, many more than 5 chunks!



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

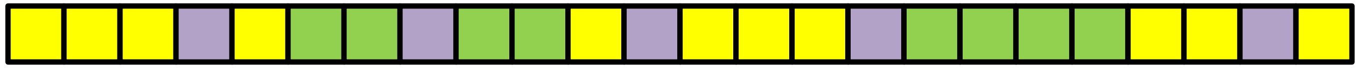


3. Return median of medians (using Quickselect, this algorithm, called recursively, on list of medians)

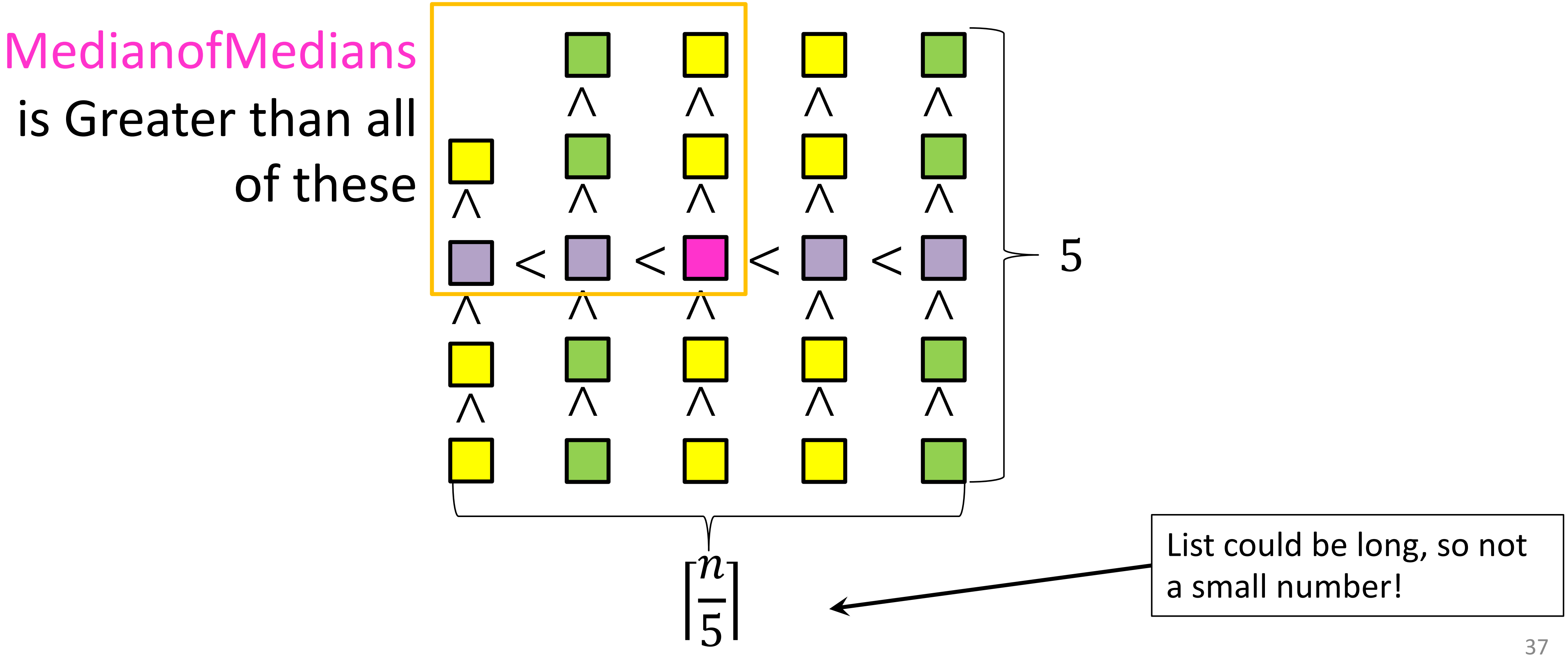


List could be long, many more than 5 medians!

Why is this good?

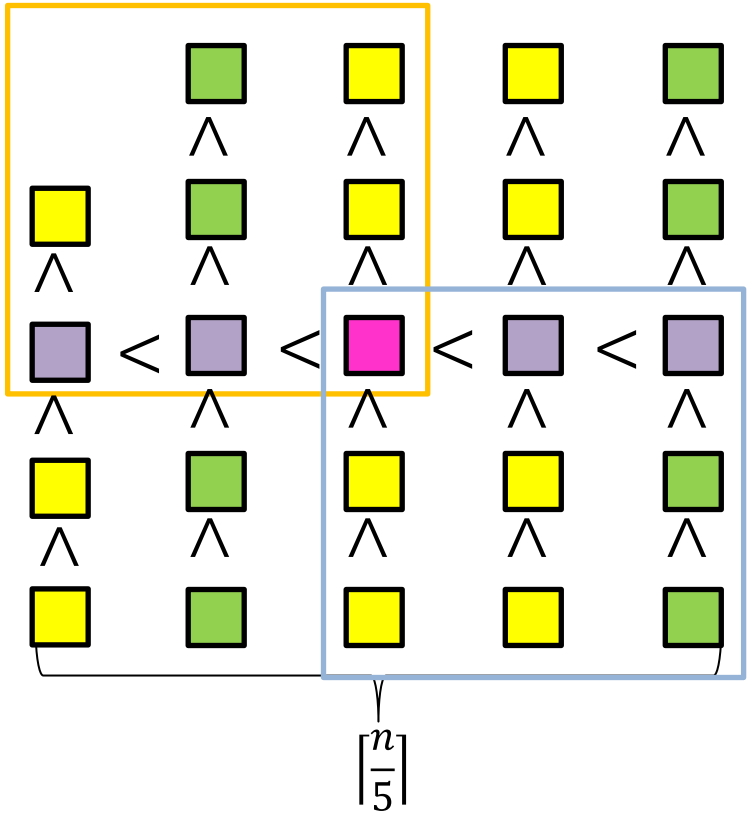


Each chunk sorted, chunks ordered by their medians



Why is this good?

MedianofMedians is larger than all of these



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

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

Similarly:

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

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

Run-time of 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

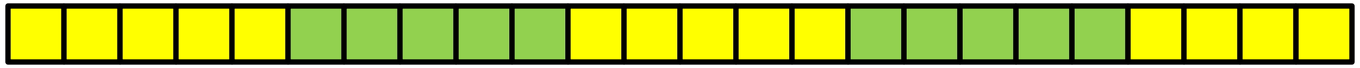
  $\leq S\left(\frac{7}{10}n\right)$

- Combine: Nothing!

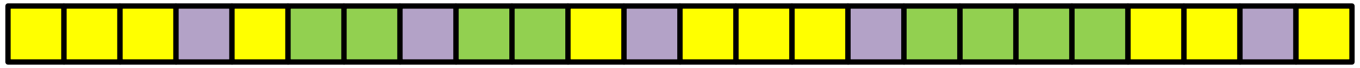
$$S(n) \leq S\left(\frac{7}{10}n\right) + M(n) + \Theta(n)$$

Median of Medians, Run Time

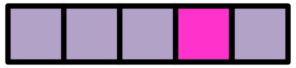
1. Break list into chunks of 5 $\Theta(n)$



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

CLRS gives a more rigorous proof!
See p. 222 for more details

Master theorem Case 3!

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

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

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 $\Omega(n \log n)$
  - Later we'll 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:

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?

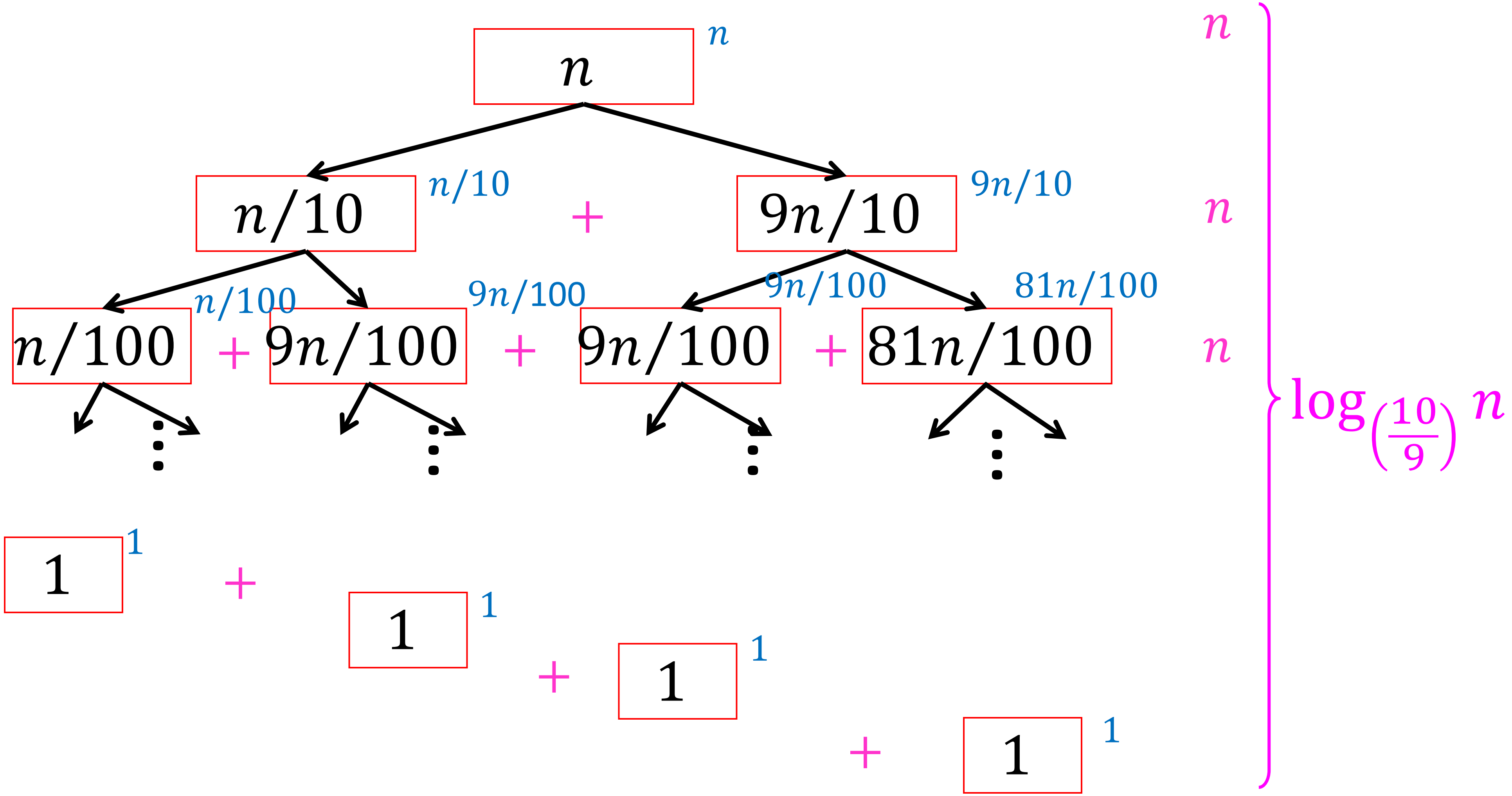
- Using Quickselect to pick median guarantees $\Theta(n \log n)$ run time
- But, this approach has very large constants
  - If you really want $\Theta(n \log n)$, better off using MergeSort
- Better approach: Choose random pivot for Quicksort
  - Very small constant (random() is a fast algorithm)
  - Can prove the *expected runtime* is $\Theta(n \log n)$
    - Why? Unbalanced partitions are very unlikely

Quicksort Run Time

If the pivot is always $\frac{n}{10}$th order statistic:

$$T(n) = T\left(\frac{n}{10}\right) + T\left(\frac{9n}{10}\right) + n$$

$$T(n) = T\left(\frac{n}{10}\right) + T\left(\frac{9n}{10}\right) + n$$



Quicksort Run Time

If the pivot is always $\frac{n}{10}$th order statistic:

$$T(n) = T\left(\frac{n}{10}\right) + T\left(\frac{9n}{10}\right) + n$$

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

Quicksort Run Time

If the pivot is always d^{th} order statistic:

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 d each time

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

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

What's the probability of this occurring?

Probability of n^2 run time

We must consistently select pivot from within the first d terms

Probability first pivot is among d smallest: $\frac{d}{n}$

Probability second pivot is among d smallest: $\frac{d}{n-d}$

Probability all pivots are among d smallest:

$$\frac{d}{n} \cdot \frac{d}{n-d} \cdot \frac{d}{n-2d} \cdot \ldots \cdot \frac{d}{2d} \cdot 1 = \frac{1}{\left(\frac{n}{d}\right)!}$$

Sorting, so far

- Sorting algorithms we have discussed:
  - Mergesort $O(n \log n)$
  - Quicksort $O(n \log n)$
- Other sorting algorithms (will discuss):
  - Bubblesort $O(n^2)$
  - Insertionsort $O(n^2)$
  - Heapsort $O(n \log n)$

Can we do better than $O(n \log n)$?

<u>Mental Stretch</u>

Show $\log(n!) = \Theta(n \log n)$

Hint: show $n! \leq n^n$

Hint 2: show $n! \geq \left(\frac{n}{2}\right)^{\frac{n}{2}}$

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

$$\begin{array}{ccccccccccc} n! = & n & \cdot & (n-1) & \cdot & (n-2) & \cdot \ldots \cdot & 2 & \cdot & 1 \\ & \| & & \wedge & & \wedge & & \wedge & & \wedge \\ n^n = & n & \cdot & n & \cdot & n & \cdot \ldots \cdot & n & \cdot & n \end{array}$$

$$n! \le n^n$$
$$\Rightarrow \log(n!) \le \log(n^n)$$
$$\Rightarrow \log(n!) \le n \log n$$
$$\Rightarrow \log(n!) = O(n \log n)$$

$\log n! = \Omega(n \log n)$

$$
\begin{array}{ccccccccccccccc}
n! = & n & \cdot & (n-1) & \cdot & (n-2) & \cdot \ldots \cdot & \frac{n}{2} & \cdot & \left(\frac{n}{2}-1\right) & \cdot \ldots \cdot & 2 & \cdot & 1 \\
 & \vee & & \vee & & \vee & & \parallel & & \vee & & \vee & & \parallel \\
\left(\frac{n}{2}\right)^{\frac{n}{2}} = & \frac{n}{2} & \cdot & \frac{n}{2} & \cdot & \frac{n}{2} & \cdot \ldots \cdot & \frac{n}{2} & \cdot & 1 & \cdot \ldots \cdot & 1 & \cdot & 1
\end{array}
$$

$$n! \geq \left(\frac{n}{2}\right)^{\frac{n}{2}}$$

$$\Rightarrow \log(n!) \geq \log\left(\left(\frac{n}{2}\right)^{\frac{n}{2}}\right)$$

$$\Rightarrow \log(n!) \geq \frac{n}{2} \log \frac{n}{2}$$

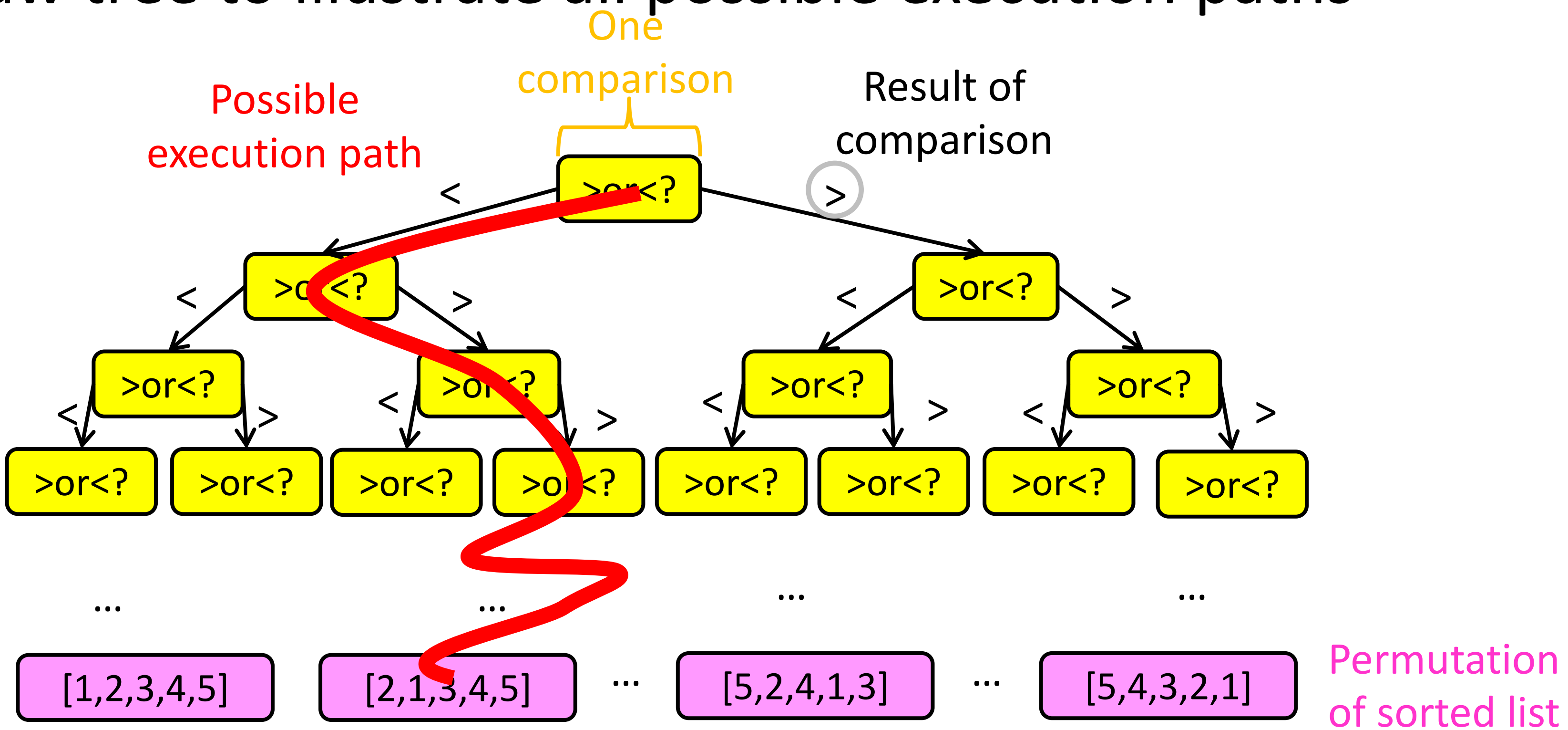
$$\Rightarrow \log(n!) = \Omega(n \log n)$$

Worst Case Lower Bounds

- Prove that there is no algorithm which can sort faster than $O(n \log n)$
- Non-existence proof!
  - Very hard to do

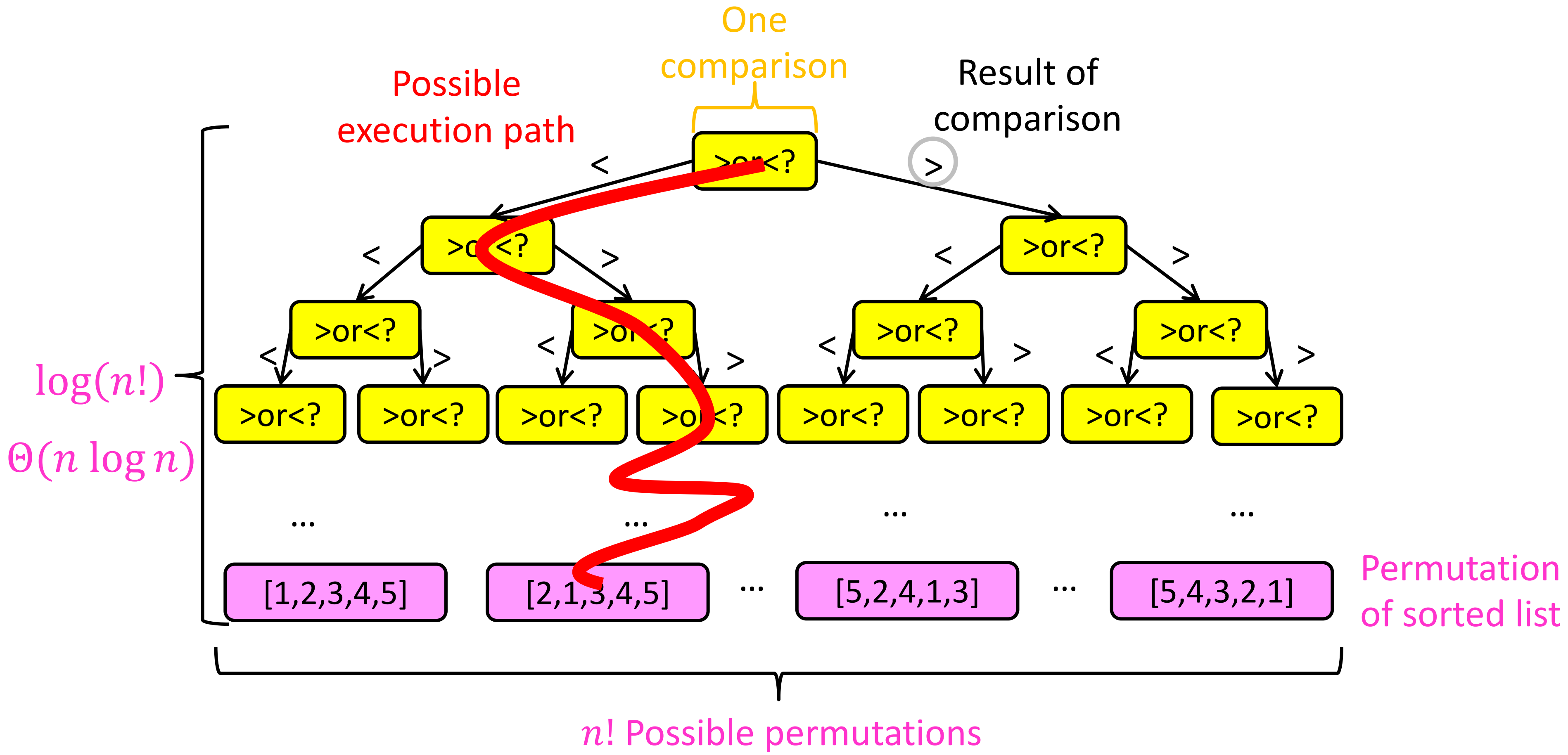
Strategy: Decision Tree

- Sorting algorithms use comparisons to figure out the order of input elements
- Draw tree to illustrate all possible execution paths



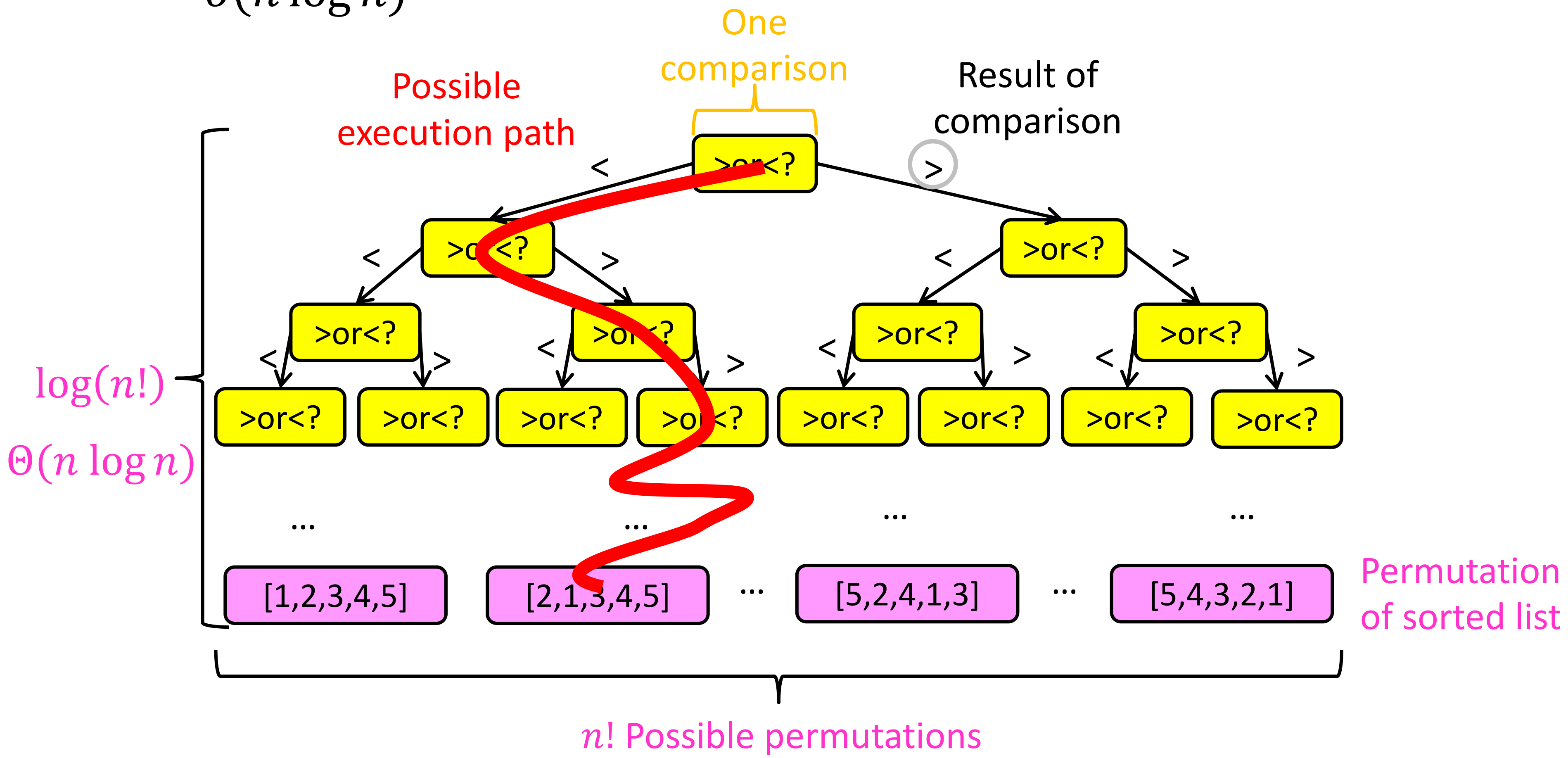
Strategy: Decision Tree

- Worst case run time is the longest execution path
- i.e., “height” of the decision tree



Strategy: Decision Tree

- Conclusion: Worst Case Optimal run time of sorting is $\Theta(n \log n)$
  - There is no (comparison-based) sorting algorithm with run time $o(n \log n)$



Sorting, so far

- Sorting algorithms we have discussed:
  - Mergesort $O(n \log n)$ Optimal!
  - Quicksort $O(n \log n)$ Optimal!
- Other sorting algorithms (will discuss):
  - Bubblesort $O(n^2)$
  - Insertionsort $O(n^2)$
  - Heapsort $O(n \log n)$ Optimal!

Speed Isn't Everything

- Important properties of sorting algorithms:
- Run Time
  - Asymptotic Complexity
  - Constants
- In Place (or In-Situ)
  - Done with only constant additional space
- Adaptive
  - Faster if list is nearly sorted
- Stable
  - Equal elements remain in original order
- Parallelizable
  - Runs faster with many computers

Mergesort

- **Divide**:
  - Break n-element list into two lists of $^n/_2$ elements
- **Conquer**:
  - If $n > 1$: Sort each sublist recursively
  - If $n = 1$: List is already sorted (base case)
- **Combine**:
  - Merge together sorted sublists into one sorted list

Run Time?

$\Theta(n \log n)$

Optimal!

In Place?	Adaptive?	Stable?
No	No	Yes! (usually)

Merge

- **Combine:** Merge sorted sublists into one sorted list
- We have:
  - 2 sorted lists (L_1, L_2)
  - 1 output list (L_{out})

While (L_1 and L_2 not empty):

    If $L_1[0] \leq L_2[0]$:

        L_{out}.append(L_1.pop())

    Else:

        L_{out}.append(L_2.pop())

L_{out}.append(L_1)

L_{out}.append(L_2)

<u>Adaptive</u>:

If elements are equal, leftmost comes first

Mergesort

- **Divide**:
  - Break n-element list into two lists of $^{n}/_{2}$ elements
- **Conquer**:
  - If $n > 1$: Sort each sublist recursively
  - If $n = 1$: List is already sorted (base case)
- **Combine**:
  - Merge together sorted sublists into one sorted list

Run Time?

$\Theta(n \log n)$

Optimal!

In Place?	Adaptive?	Stable?	Parallelizable?
No	No	Yes! (usually)	Yes!

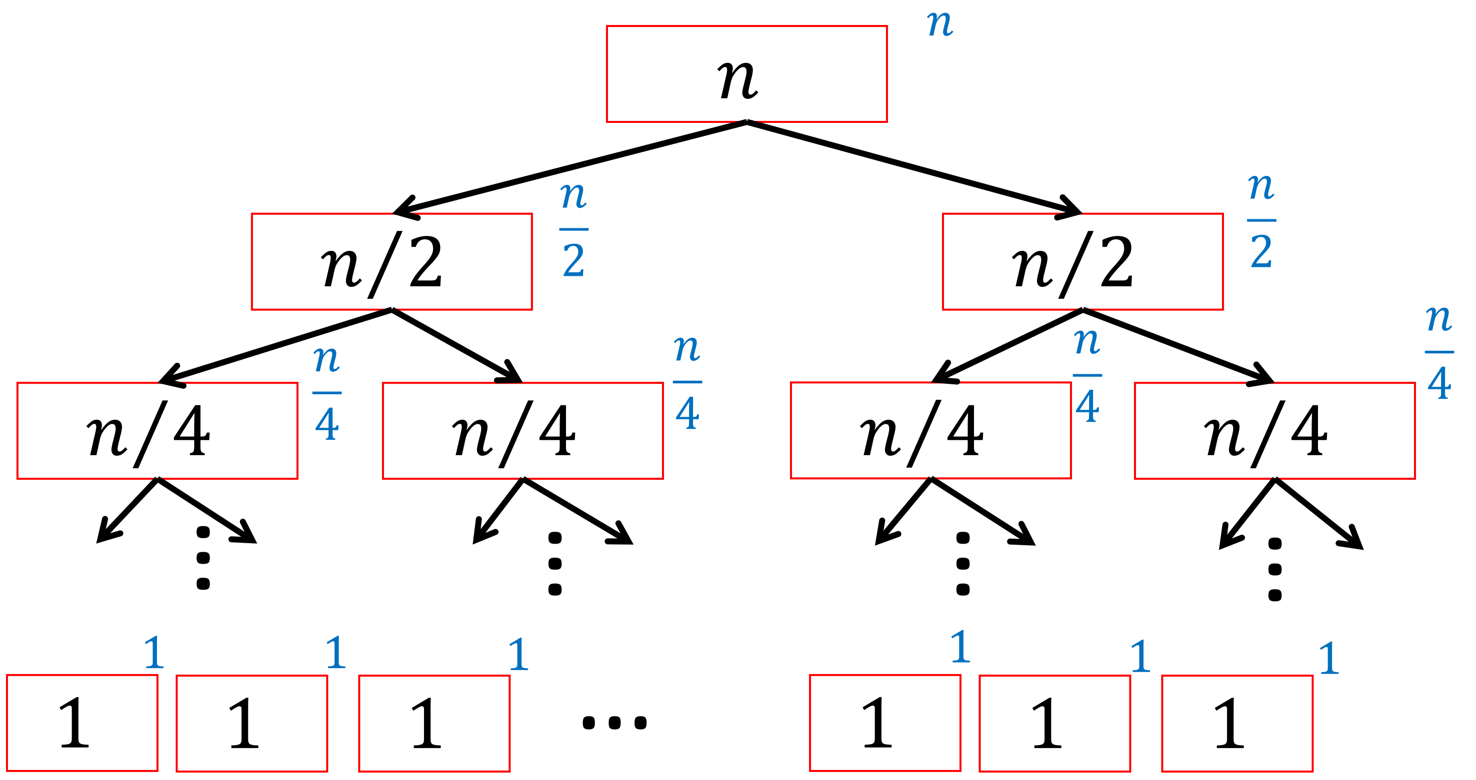
Mergesort

- **Divide**:
  - Break n-element list into two lists of $^n/_2$ elements
- **Conquer**:
  - If $n > 1$:
    - Sort each sublist recursively
  - If $n = 1$:
    - List is already sorted (base case)
- **Combine**:
  - Merge together sorted sublists into one sorted list

Parallelizable: Allow different machines to work on each sublist

Mergesort (Sequential)

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



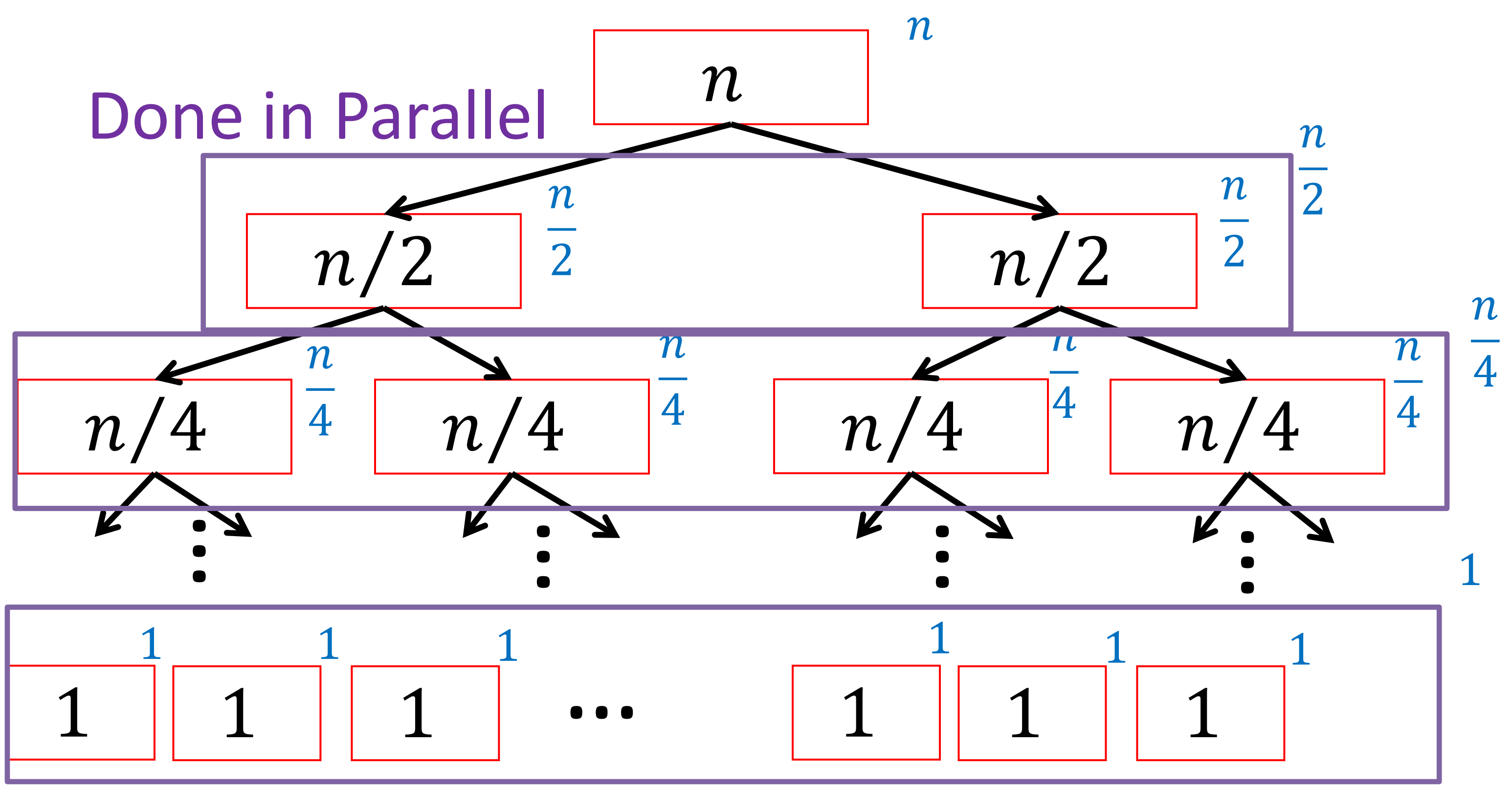
n total / level

$\log_2 n$ levels of recursion

Run Time: $\Theta(n \log n)$

Mergesort (Parallel)

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



Run Time: $\Theta(\log n)$

Quicksort

- Idea: pick a partition element, recursively sort two sublists around that element
- Divide: select an element p, Partition(p)
- Conquer: recursively sort left and right sublists
- Combine: Nothing!

Run Time?

$\Theta(n \log n)$

Optimal!

(almost always)

In Place?	Adaptive?	Stable?	Parallelizable?
No…	No!	No	Yes!