CS4102 Algorithms

Spring 2020 – Horton's Slides

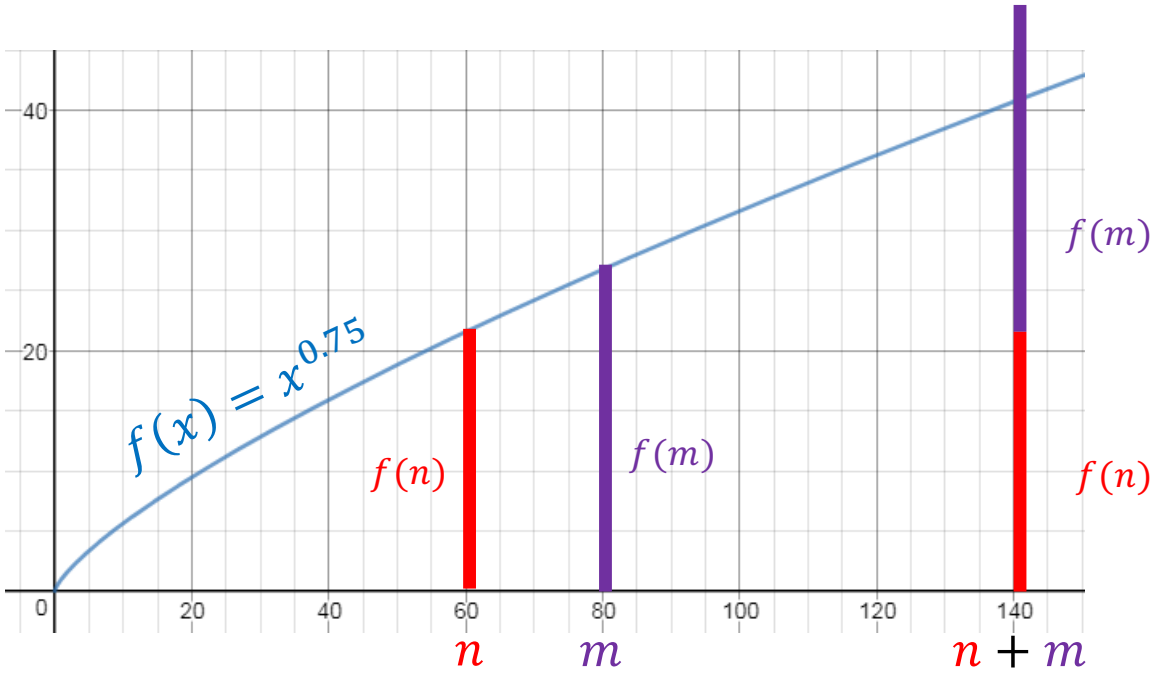
Reminder Warm-Up

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

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

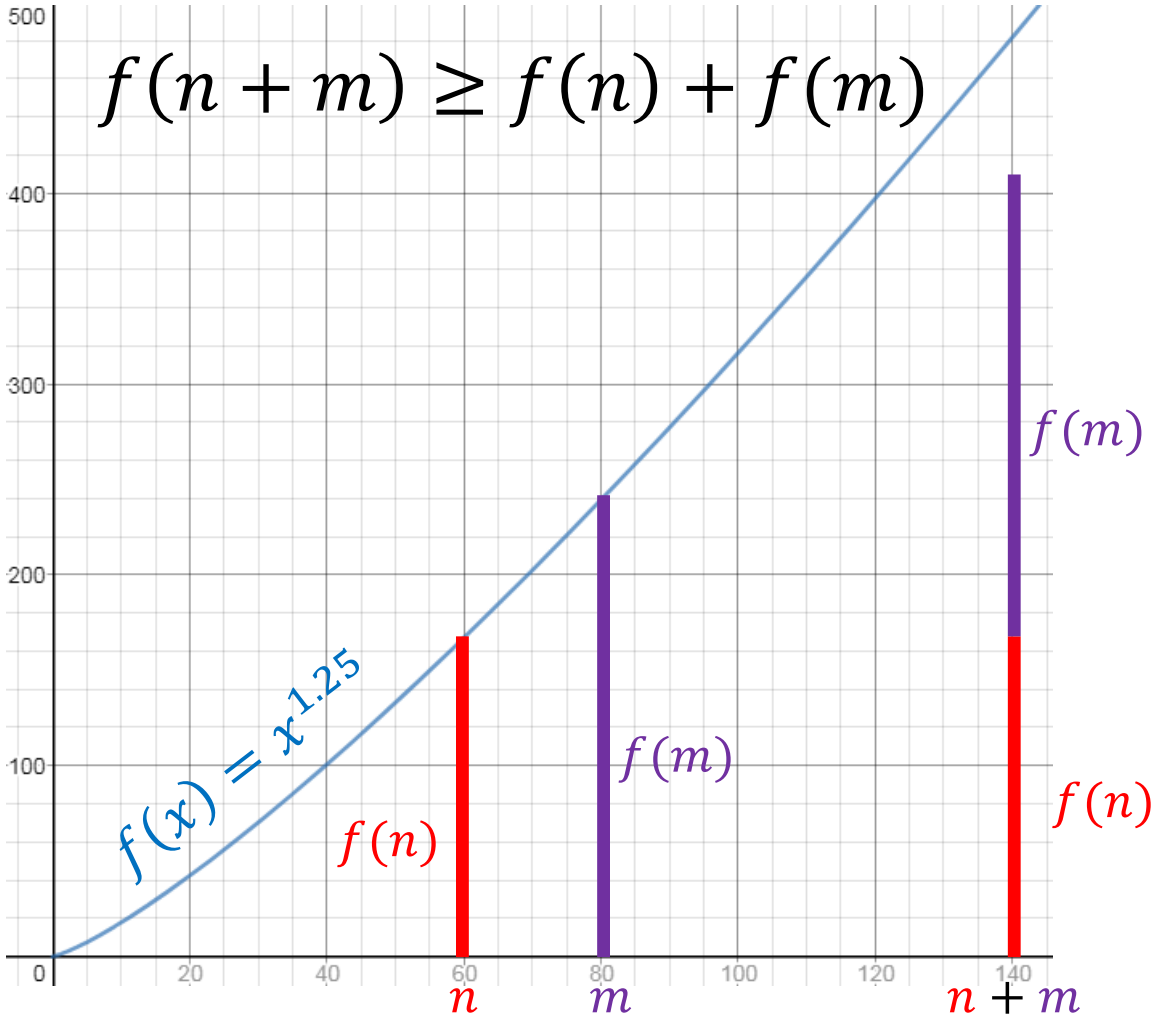
When $f(n) = \Omega(n)$

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

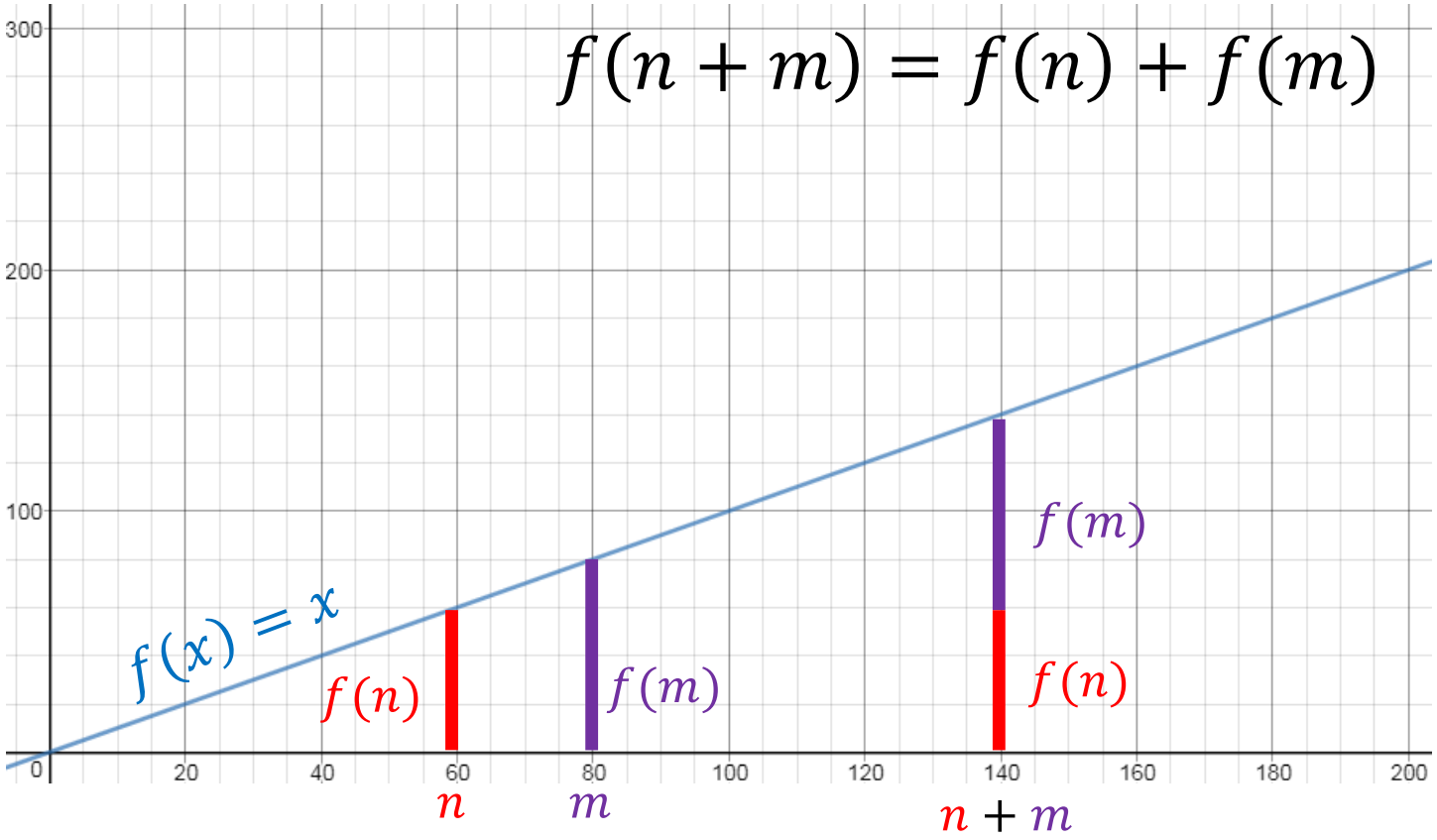


$$f(n+m) \leq f(n) + f(m)$$

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


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

$f(n) \in \Theta(n)$


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

Warm Up

Guess the solution to this recurrence:

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

where $c \geq 1$ is a constant

Warm Up

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

$$\frac{n}{5} + \frac{7n}{10} = \frac{9n}{10} < n$$

If this was $T\left(\frac{9n}{10}\right)$, then can use Master's Theorem to conclude $\Theta(n)$

Guess: $\Theta(n)$

Suffices to show $O(n)$ since non-recursive cost is already $\Omega(n)$

Warm Up

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

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

Warm Up

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

Inductive hypothesis: $\forall n \leq x_0 : T(n) \leq 10cn$

Inductive step:

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

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

$$= 9c(x_0 + 1) + c(x_0 + 1) = 10c(x_0 + 1)$$

Today's Keywords

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

CLRS Readings

- Chapter 7
- Chapter 9

Homeworks

- HW3 due 11pm tomorrow
  - Programming (use Python or Java!)
  - Divide and conquer
  - Closest pair of points
- HW4 coming soon
  - Written, using LaTeX

Review: Quicksort

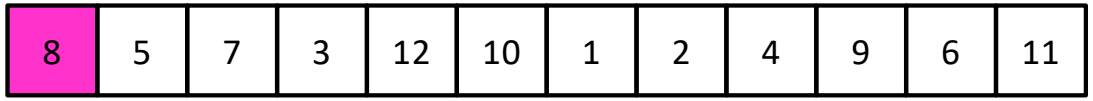
Idea: pick a pivot element, recursively sort two sublists around that element

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

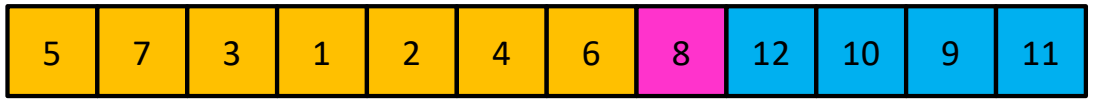
Partition (Divide step)

Given: a list, a pivot p

Start: unordered list



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

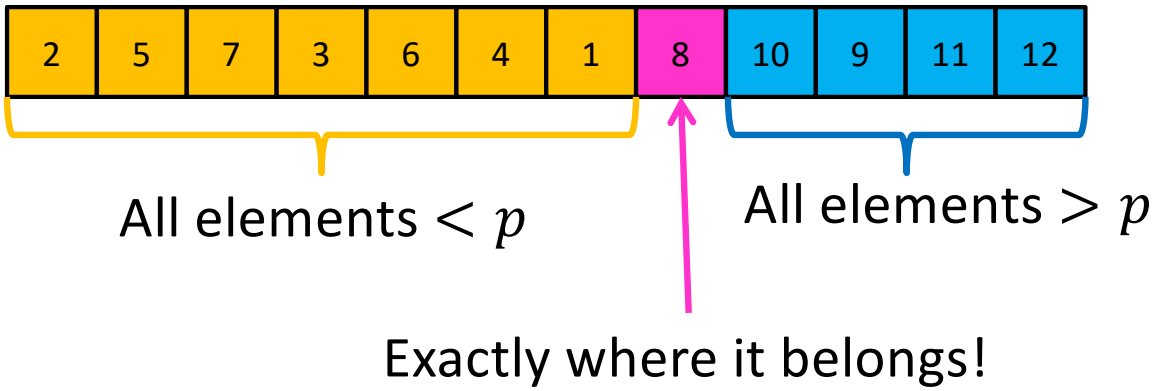


Partition Summary

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? $O(n)$

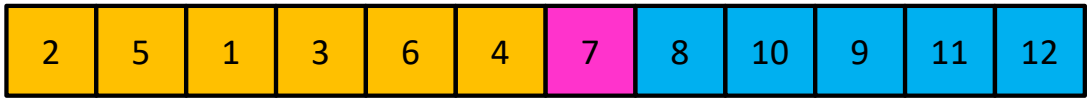
Conquer

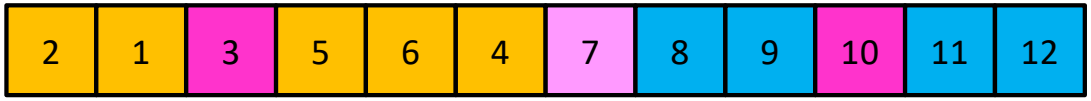


Recursively sort Left and Right sublists

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

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

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

How to pick the pivot?

CLRS, Chapter 9

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**: Give list of distinct numbers and value *i*, find value *x* in list that is larger than exactly *i-1* list elements

Quickselect

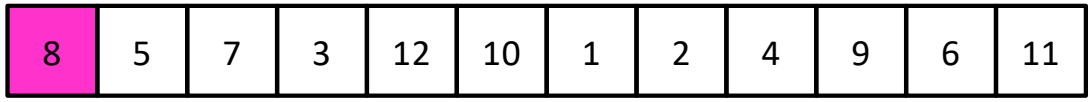
Idea: pick a pivot element, partition, then recurse on sublist containing index i

- 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!

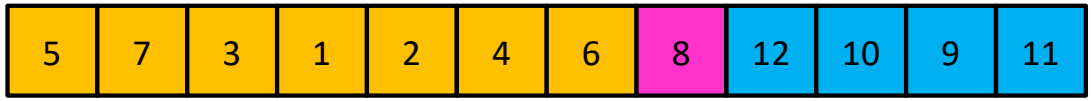
Partition (Divide step)

Given: a list, a pivot value p

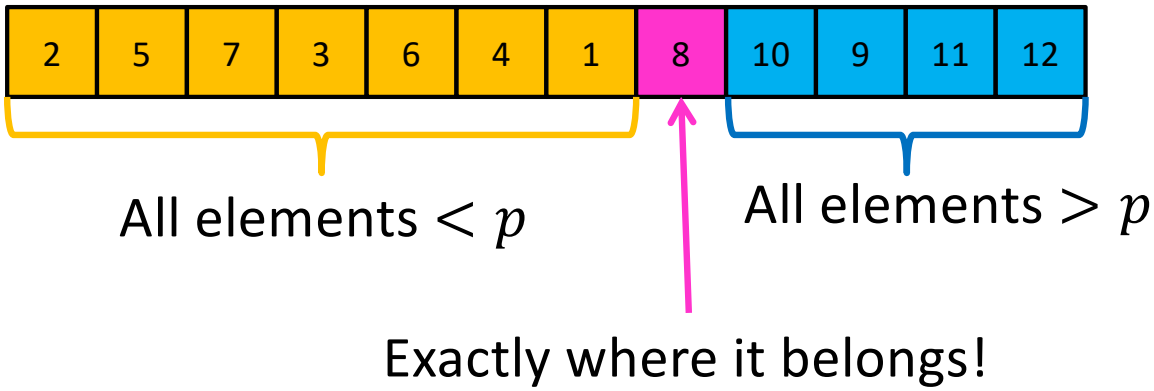
Start: unordered list



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



Conquer



Recurse on sublist that contains index i

(adjust i accordingly if recursing right)

CLRS Pseudocode

```
RANDOMIZED-SELECT(A, p, r, i)
1  if p == r
2       return A[p]
3  q = RANDOMIZED-PARTITION(A, p, r)
4  k = q − p + 1      // number of elements on left-side of pivot
5  if i == k          // the pivot value is the answer
6       return A[q]
7  elseif i < k
8       return RANDOMIZED-SELECT(A, p, q − 1, i)
9  else return RANDOMIZED-SELECT(A, q + 1, r, i − k)
                                  // note adjustment to next call's i
```

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.

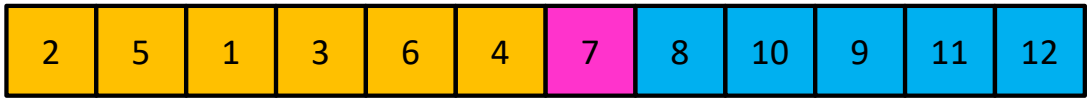
Work These Examples!

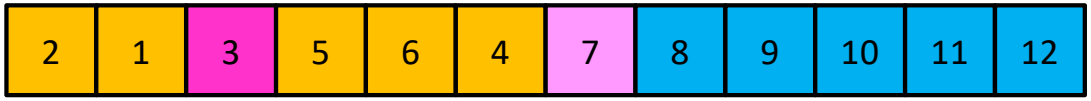
- For each of the following calls, show
  - The value of *q* after each partition,
  - Which recursive calls made

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:





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

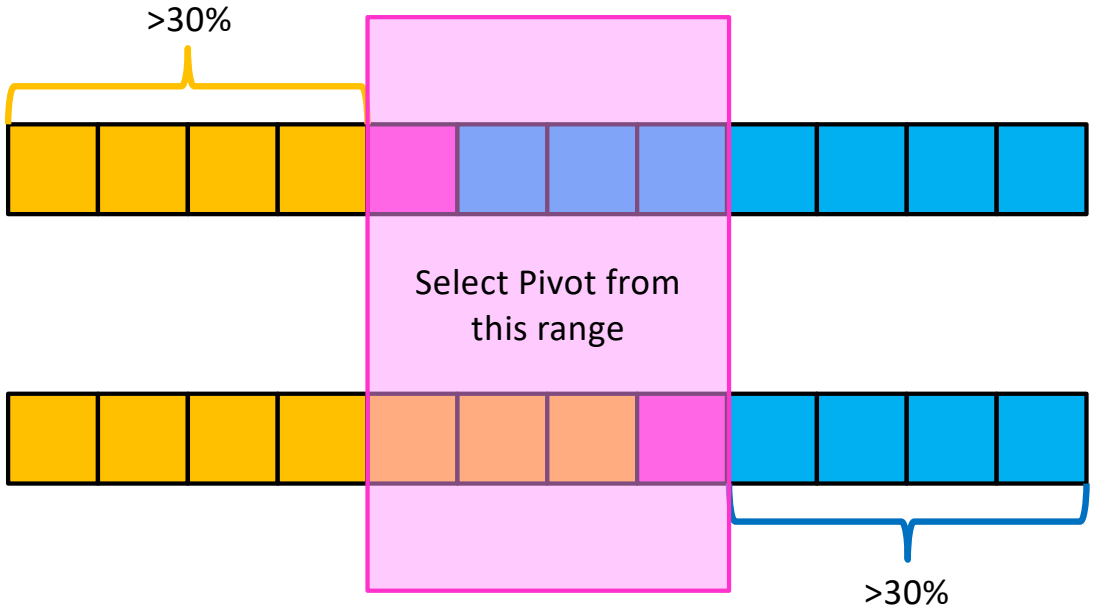
Déjà vu?

- Here's what's next:
  - An algorithm that can find the median in linear time
    1. It starts by finding a pivot that is a "rough" split (Median of Medians)
    2. Uses that pivot with Quickselect shown earlier to recursively find median
       - (Recall that Quickselect shown earlier used first element to do Partition. Now use the pivot value found in step 1.)

Good Pivot

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

Or



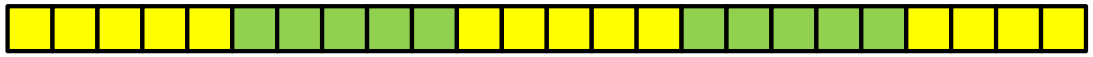
Median of Medians

- 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

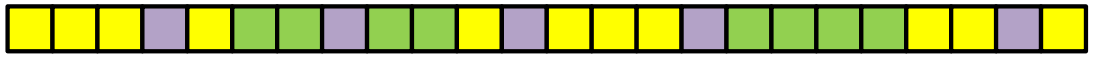
- CLRS, pp. 220-221

Median of Medians

1. Break list into chunks of size 5

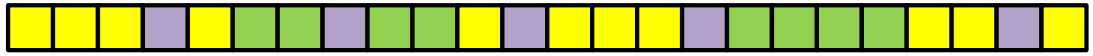


2. Find the median of each chunk
   (using insertion sort: n=5, 20 comparisons)



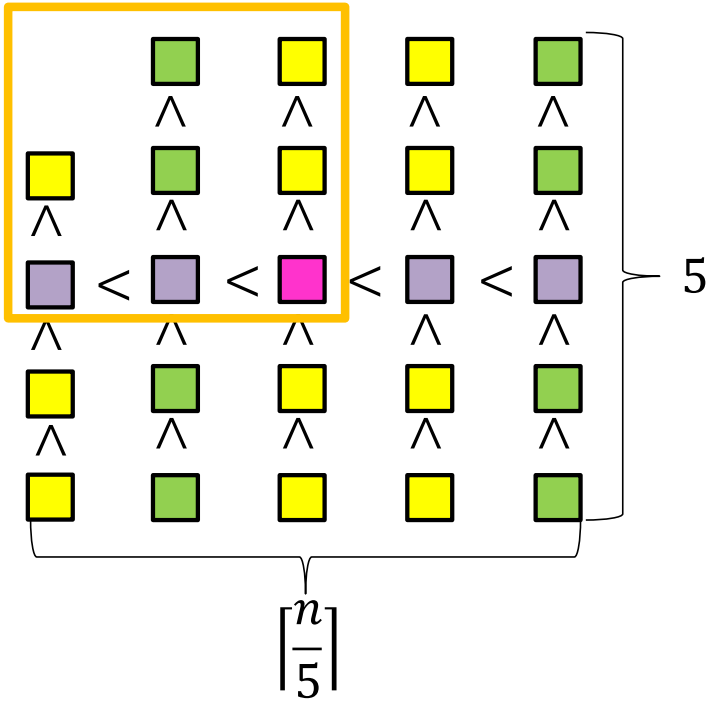
3. Return median of medians (using Quickselect, this algorithm, called recursively, on list of medians)

Why is this good?



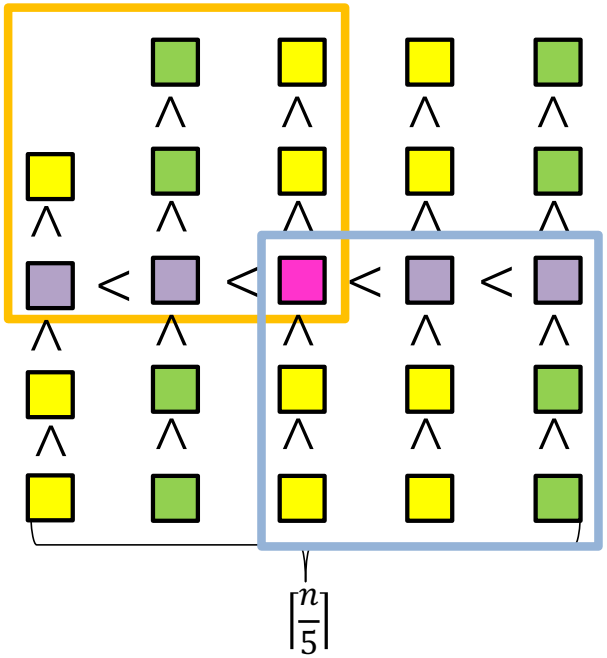
Imagine each chunk sorted, chunks ordered by their medians

MedianofMedians is Greater than all of these



Why is this good?

MedianofMedians is larger than all of these



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 $>$ ■

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

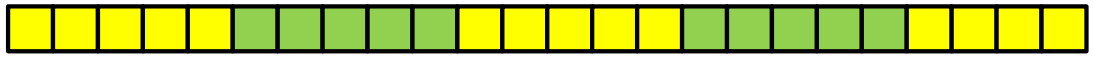
Quickselect

- 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!

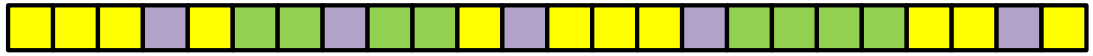
$$S(n) \leq S\left(\frac{7}{10}n\right) + M(n) + \Theta(n)$$

Median of Medians, Run Time

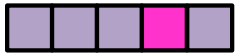
1. Break list into chunks of 5 $\Theta(n)$



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

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

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

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

... Guess and Check ...

Warm Up!

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

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

Linear work done at top level (even if no recursion costs)

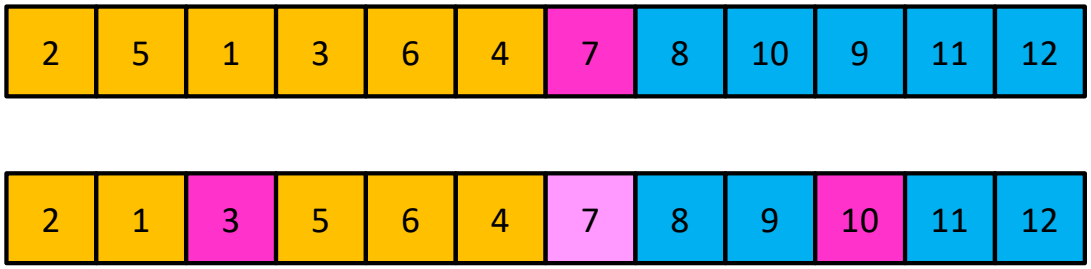
$$S(n) = \Theta(n)$$

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, we're guaranteed to use true median, so:



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?

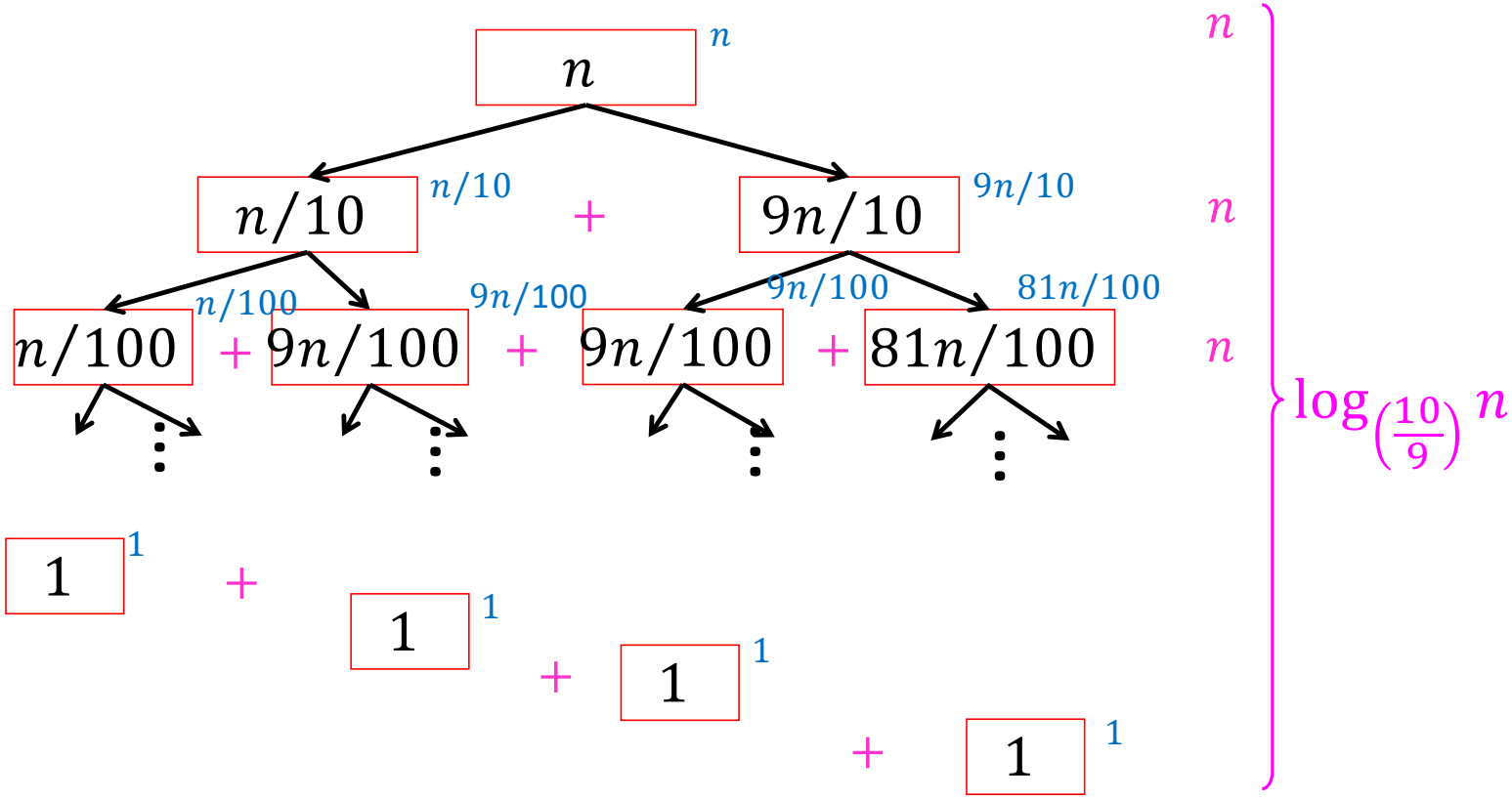
- Using Quickselect to pick median guarantees $\Theta(n \log n)$ run time
- Approach has very large constants
  - If you really want $\Theta(n \log n)$, better off using MergeSort
- Better approach: Random pivot
  - Very small constant (very fast algorithm)
  - Expected to run in $\Theta(n \log n)$ time
    - Why? Unbalanced partitions are very unlikely
- But let's explore "very uneven partitions" in the next slides…

Quicksort Run Time

If the pivot is always $\frac{n}{10}^{\text{th}}$ 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 $\frac{n}{10}^{\text{th}}$ 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:

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:

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

Worst-case, d=1, and the probability is very, very small!

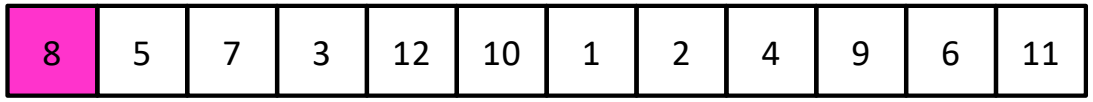
Formal Argument for $n \log n$ Average

- Remember, run time counts comparisons!
- Quicksort only compares against a pivot
  - Element i only compared to element j if one of them was the pivot

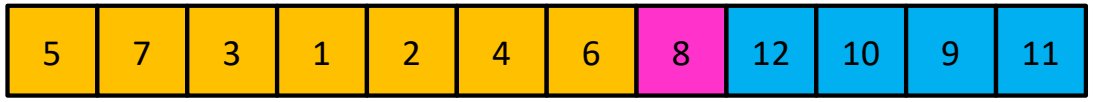
Partition (Divide step)

Given: a list, a pivot value p

Start: unordered list

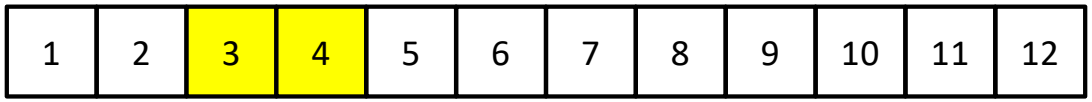


Goal: All elements $< p$ on left, all $> p$ on right



Formal Argument for $n \log n$ Average

What is the probability of comparing two given elements?



Consider the sorted version of the list

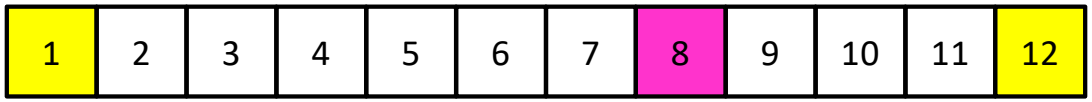
Observation: Adjacent elements must be compared

- **Why?** Otherwise I would not know which came first
- **Every** sorting algorithm **must** compare adjacent elements

In quicksort: adjacent elements always end up in same sublist, unless one is the pivot

Formal Argument for $n \log n$ Average

What is the probability of comparing two given elements?



Consider the sorted version of the list

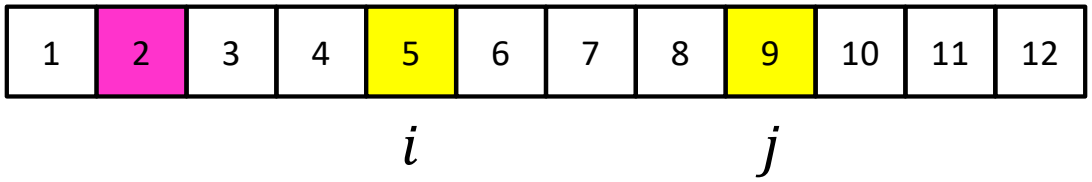
$$\Pr[\text{we compare 1 and 12}] = \frac{2}{12}$$

Assuming pivot is chosen uniformly at random

Only compared if 1 or 12 was chosen as the first pivot since otherwise they are in different sublists

Formal Argument for $n \log n$ Average

What is the probability of comparing two given elements?



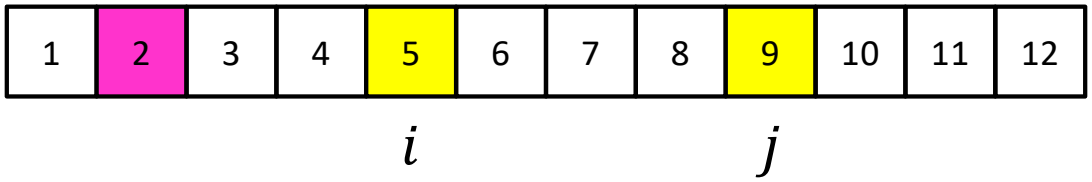
Case 1: Pivot less than i

Then sublist $[i, i+1, \ldots, j]$ will be in right sublist and will be processed in future recursive invocation of Quicksort

$$\Pr[\text{we compare } i \text{ and } j] = \Pr[\text{we compare } i \text{ and } j \text{ in Quicksort}([p+1, \ldots, n])]$$

Formal Argument for $n \log n$ Average

What is the probability of comparing two given elements?



Case 1: Pivot less than i

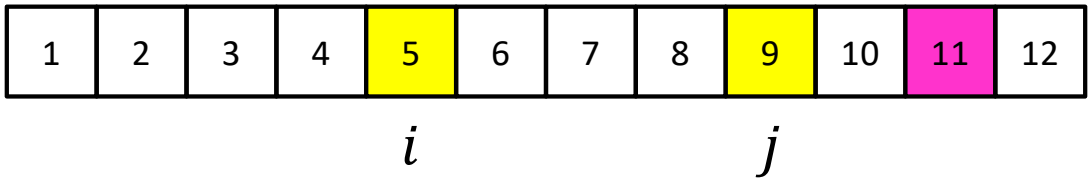
Then sublist $[i, i+1, \ldots, j]$ will be processed in future recursive invo

$[p+1, \ldots, n]$ denotes the right sublist (in some order) that we are recursively sorting

$$\Pr[\text{we compare } i \text{ and } j] = \Pr[\text{we compare } i \text{ and } j \text{ in Quicksort}([p+1, \ldots, n])]$$

Formal Argument for $n \log n$ Average

What is the probability of comparing two given elements?



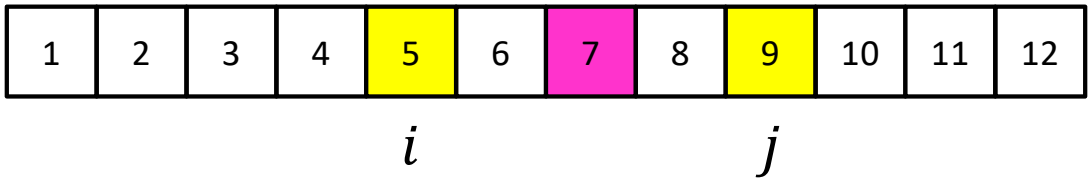
Case 2: Pivot greater than j

Then sublist $[i, i+1, \ldots, j]$ will be in left sublist and will be processed in future recursive invocation of Quicksort

$$\Pr[\text{we compare } i \text{ and } j] = \Pr[\text{we compare } i \text{ and } j \text{ in Quicksort}([1, \ldots, p])]$$

Formal Argument for $n \log n$ Average

What is the probability of comparing two given elements?



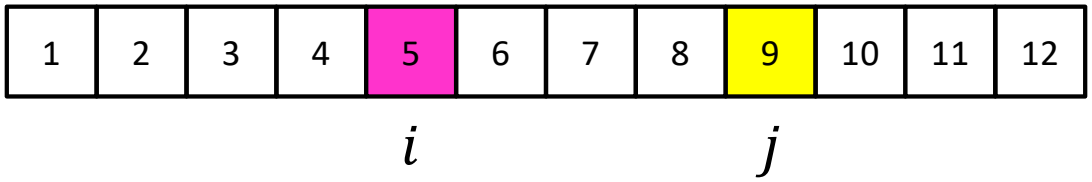
Case 3.1: Pivot contained in $[i+1, \ldots, j-1]$

Then i and j are in different sublists and will never be compared

$$\Pr[\text{we compare } i \text{ and } j] = 0$$

Formal Argument for $n \log n$ Average

What is the probability of comparing two given elements?



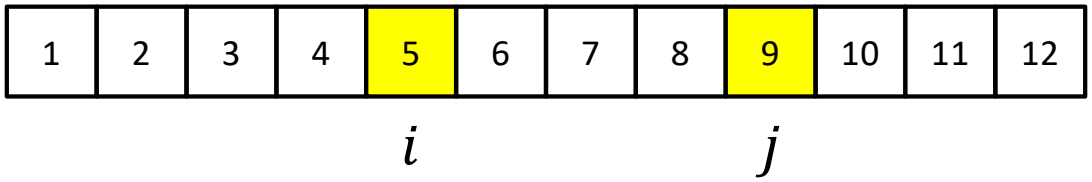
Case 3.2: Pivot is either i or j

Then we will always compare i and j

$$\Pr[\text{we compare } i \text{ and } j] = 1$$

Formal Argument for $n \log n$ Average

What is the probability of comparing two given elements?



Case 1: Pivot less than i

$$\Pr[\text{we compare } i \text{ and } j] = \Pr[\text{we compare } i \text{ and } j \text{ in Quicksort}([p+1, \dots, n])]$$

Case 2: Pivot greater than j

$$\Pr[\text{we compare } i \text{ and } j] = \Pr[\text{we compare } i \text{ and } j \text{ in Quicksort}([1, \dots, p])]$$

Case 3: Pivot in $[i, i+1, \dots, j]$

$$\Pr[\text{we compare } i \text{ and } j] = \Pr[i \text{ or } j \text{ is selected as pivot}] = \frac{2}{j-i+1}$$

Formal Argument for $n \log n$ Average

Probability of comparing i with j ($j > i$):

- dependent on the number of elements between (and including) i and j

$$\frac{2}{j - i + 1}$$

Expected number of comparisons for Quicksort:

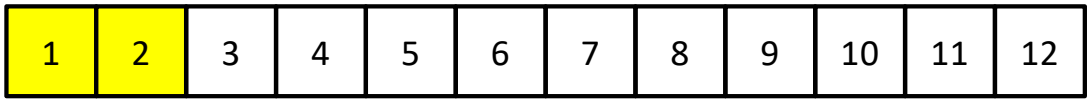
$$\sum_{i<j} \frac{2}{j - i + 1}$$

$$\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j - i + 1}$$

Expected number of Comparisons

$$\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1}$$

Consider when $i = 1$



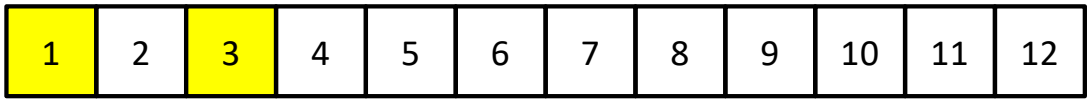
Compared if 1 or 2 are chosen as pivot
(these will always be compared)

Sum so far: $\frac{2}{2}$

Expected number of Comparisons

$$\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1}$$

Consider when $i = 1$



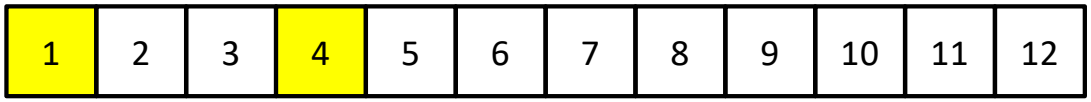
Compared if 1 or 3 are chosen as pivot
(but never if 2 is ever chosen)

Sum so far: $\frac{2}{2} + \frac{2}{3}$

Expected number of Comparisons

$$\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1}$$

Consider when $i = 1$



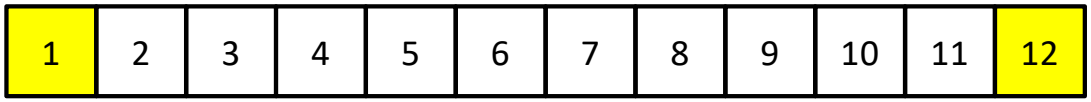
Compared if 1 or 4 are chosen as pivot
(but never if 2 or 3 are chosen)

Sum so far: $\frac{2}{2} + \frac{2}{3} + \frac{2}{4}$

Expected number of Comparisons

$$\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1}$$

Consider when $i = 1$



Compared if 1 or 12 are chosen as pivot
(but never if 2 -> 11 are chosen)

Overall sum: $\frac{2}{2} + \frac{2}{3} + \frac{2}{4} + \frac{2}{5} + \cdots + \frac{2}{n}$

Expected number of Comparisons

$$\sum_{i<j} \frac{2}{j-i+1}$$

When $i = 1$:

$$2\left(\frac{1}{2}+\frac{1}{3}+\frac{1}{4}+\cdots+\frac{1}{n}\right) < 2\boxed{\sum_{x=1}^{n}\frac{1}{x}} \qquad \Theta(\log n)$$

Formal Argument for $n \log n$ Average

- Probability of comparing element i with element j:
- $\Pr[\text{we compare } i \text{ and } j] = \frac{2}{j-i+1}$

- Expected number of comparisons:

$$\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1} = \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k+1} < 2 \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{1}{k} < 2 \sum_{i=1}^{n-1} \sum_{k=1}^{n} \frac{1}{k}$$

Substitution: $k = j - i$

$\frac{1}{k+1} < \frac{1}{k}$

Formal Argument for $n \log n$ Average

$$\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1} = \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k+1} < 2 \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{1}{k} < 2 \sum_{i=1}^{n-1} \sum_{k=1}^{n} \frac{1}{k}$$

Substitution: $k = j - i$

$$\frac{1}{k+1} < \frac{1}{k}$$

Useful fact: $\sum_{i=1}^{n} \frac{1}{i} = \Theta(\log n)$

Intuition (not proof!):

$$\sum_{i=1}^{n} \frac{1}{i} \approx \int_{1}^{n} \frac{1}{x} dx = \ln n$$

Formal Argument for $\boldsymbol{n \log n}$ Average

$$\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1} = \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k+1} < 2 \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{1}{k} < 2 \sum_{i=1}^{n-1} \sum_{k=1}^{n} \frac{1}{k}$$

$$= 2 \sum_{i=1}^{n-1} \Theta(\log n) = \Theta(n \log n)$$

Quicksort overall: expected $\Theta(n \log n)$

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 $O(n \log n)$?

Mental Stretch

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

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

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

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

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

$n! \leq n^n$

$\Rightarrow \log(n!) \leq \log(n^n)$

$\Rightarrow \log(n!) \leq n \log n$

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

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

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

$$\left(\frac{n}{2}\right)^{\frac{n}{2}} = \frac{n}{2} \cdot \frac{n}{2} \cdot \frac{n}{2} \cdot \ldots \cdot \frac{n}{2} \cdot 1 \cdot \ldots \cdot 1 \cdot 1$$

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

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

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

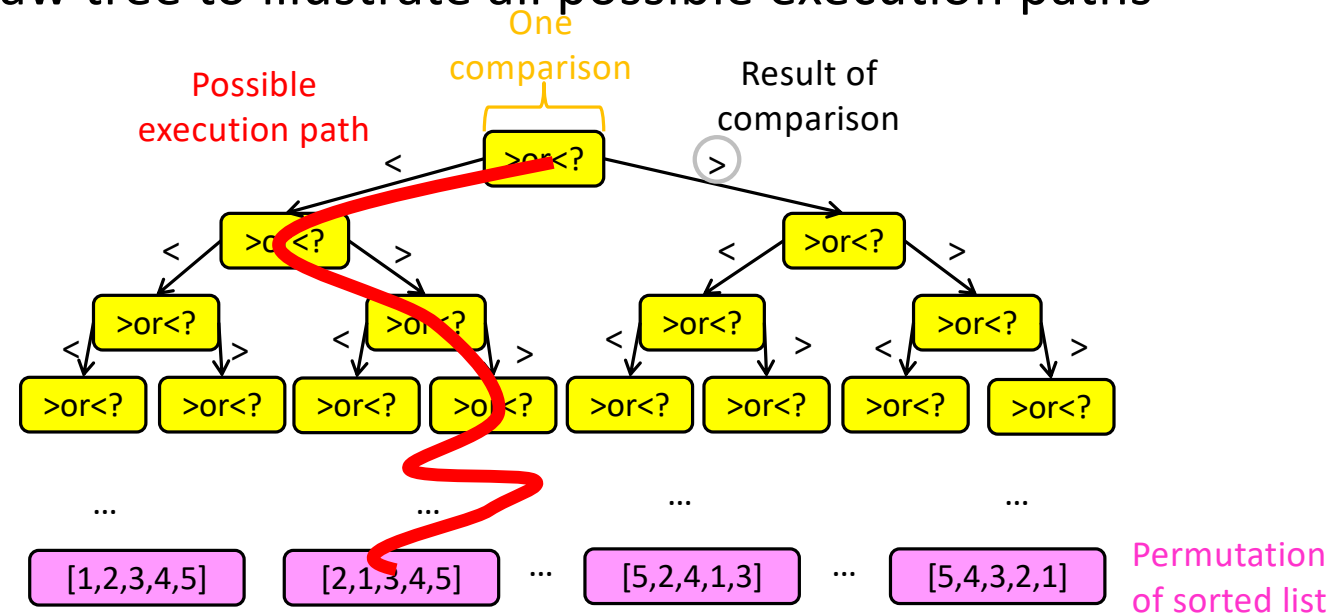
$$\Rightarrow \log(n!) = \Omega(n \log n)$$

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

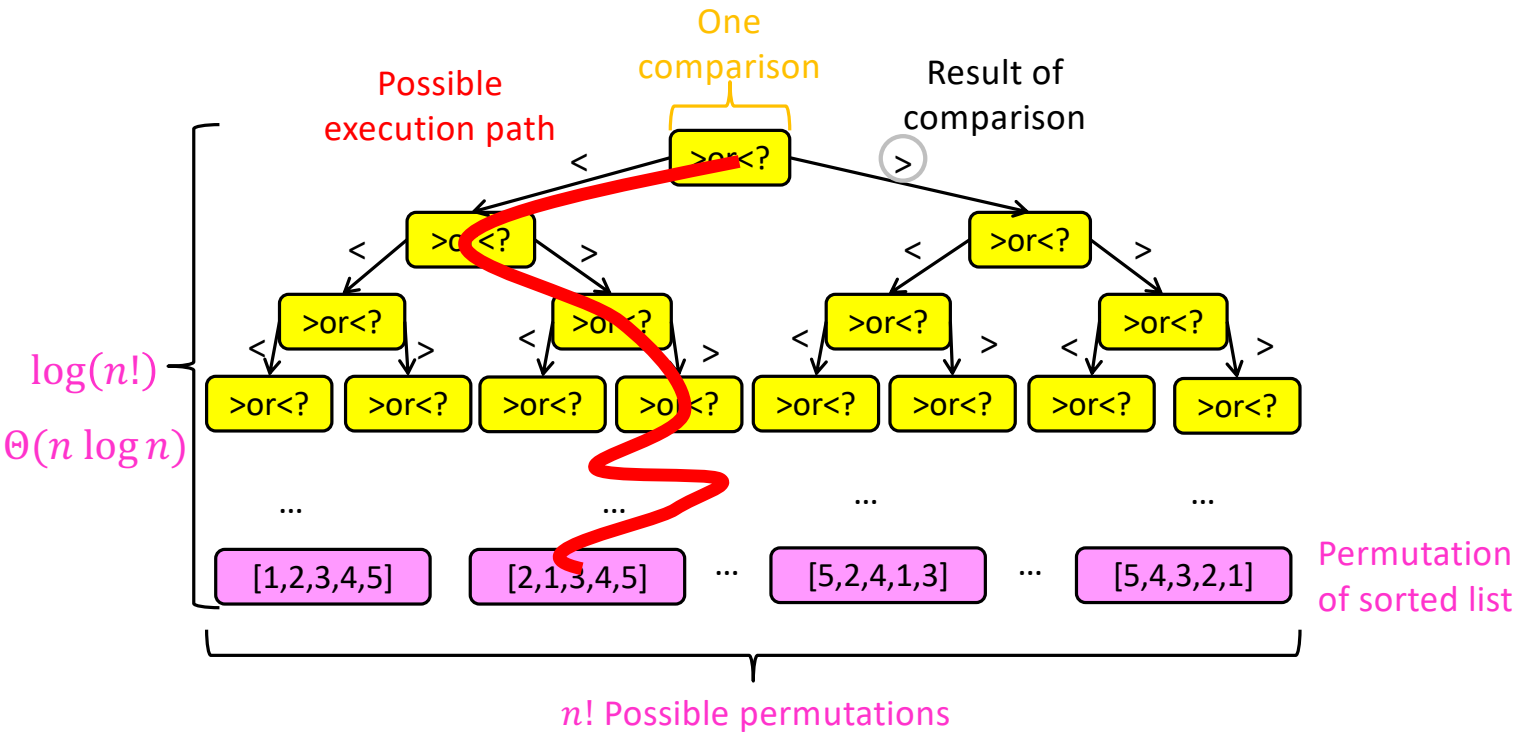
Strategy: Decision Tree

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



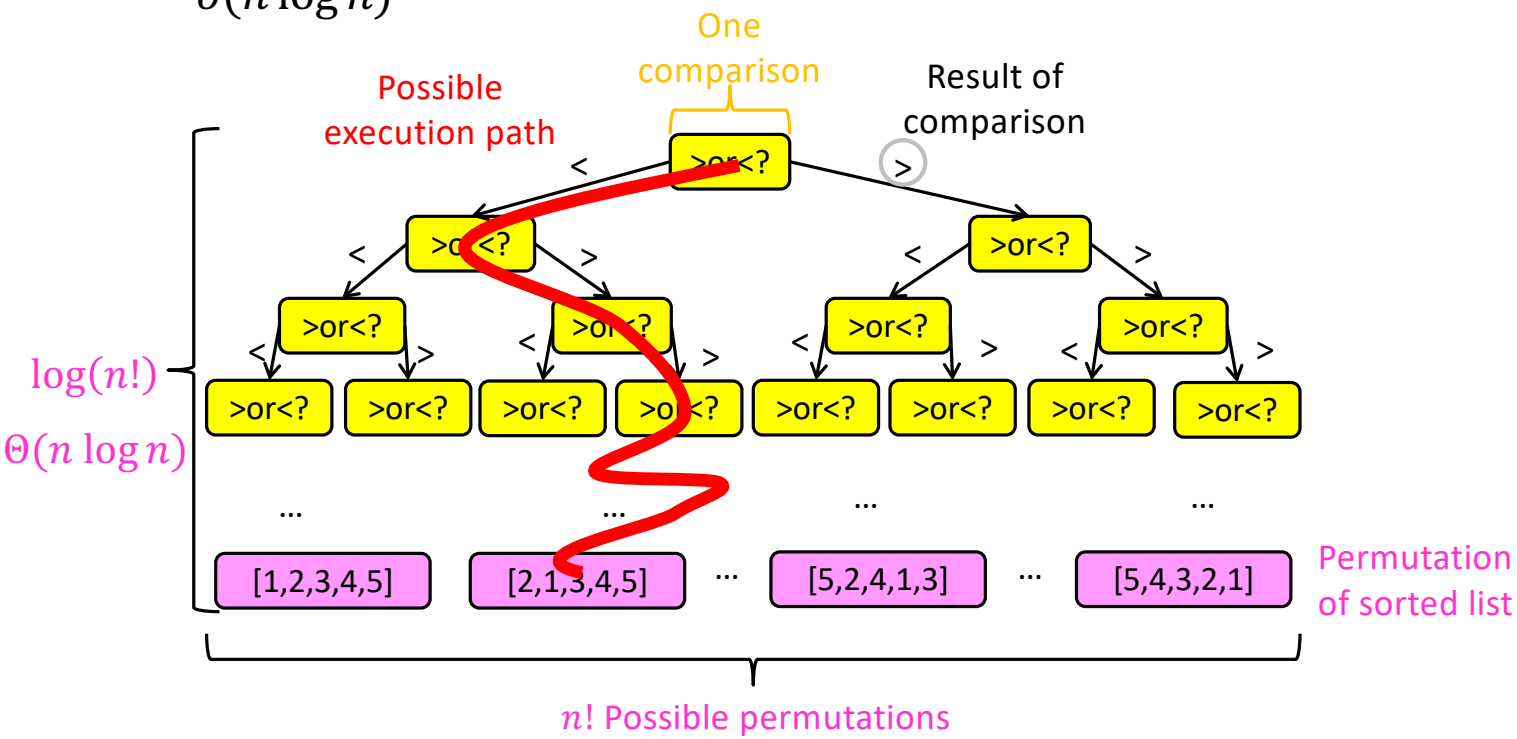
Strategy: Decision Tree

- 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)

Adaptive:
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!

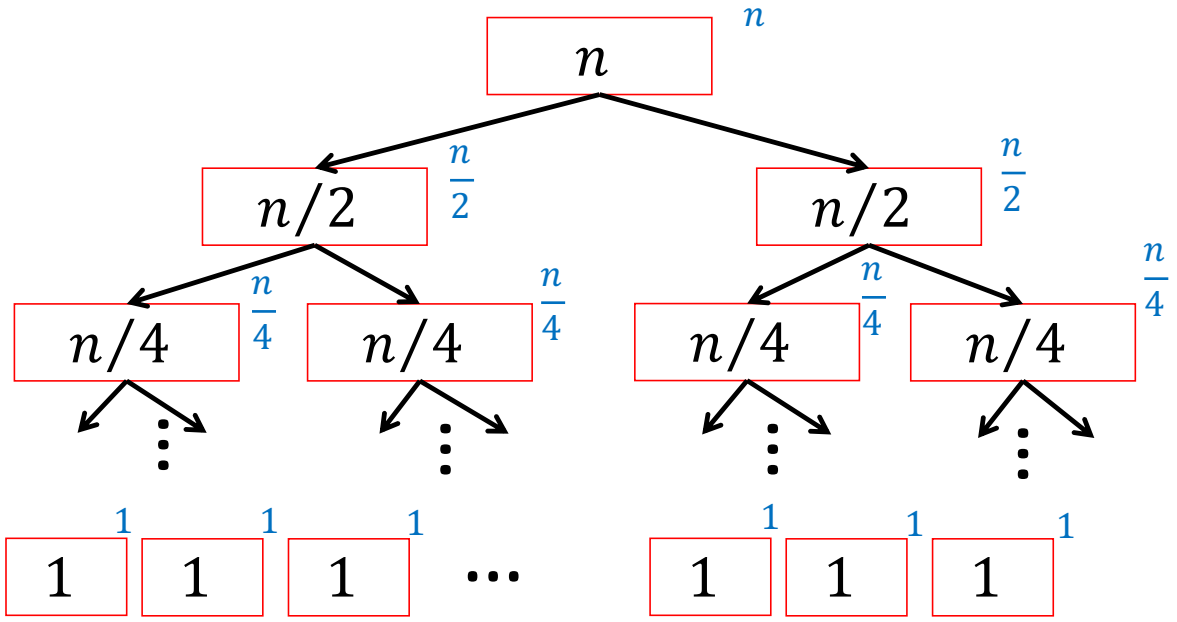
Mergesort

- **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)

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



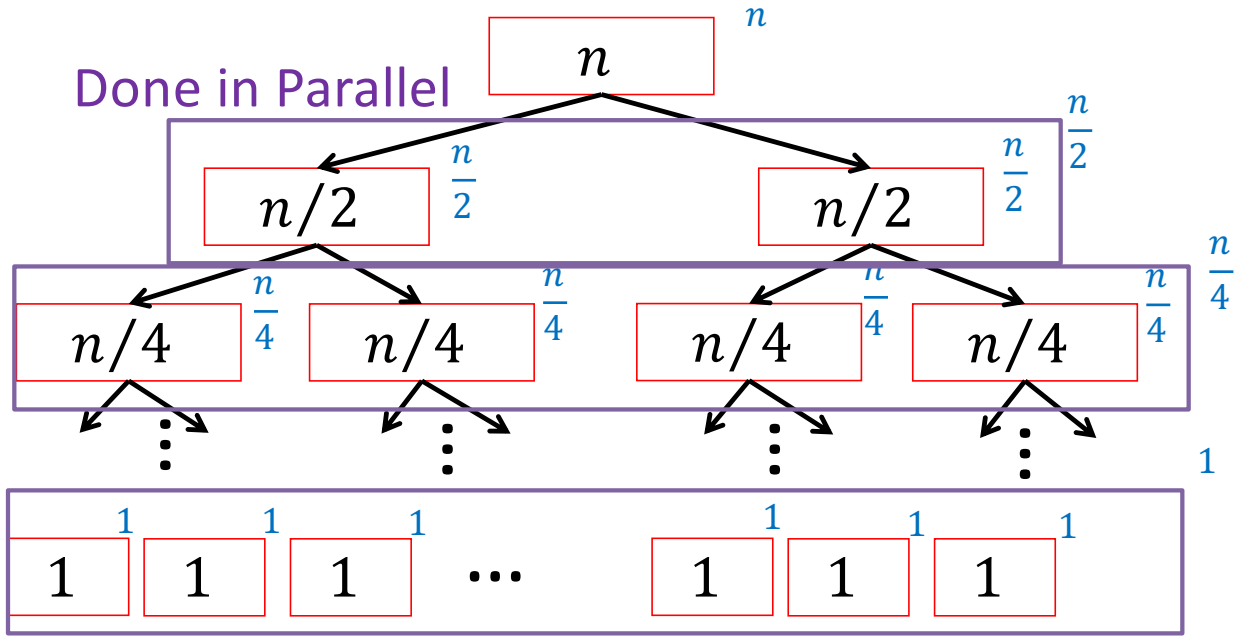
n total / level

$\log_2 n$ 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!