# CS 3100
# Data Structures and Algorithms 2
## Lecture 11: Matrix Multiplication, Quickselect

**Co-instructors: Robbie Hott and Tom Horton**

**Fall 2023**

Readings in CLRS 4th edition:

- Section 4.5

# Announcements

- Upcoming dates
  - PS2 due September 29 (Friday) at 11:59pm
  - PA2 due October 8 (Sunday) at 11:59pm
  - Quizzes 1 and 2 Thursday October 5 in class
- Course email (comes to both professors and head TAs):

**cs3100@cshelpdesk.atlassian.net**

# Divide and Conquer

[CLRS Chapter 4]

**Divide**:

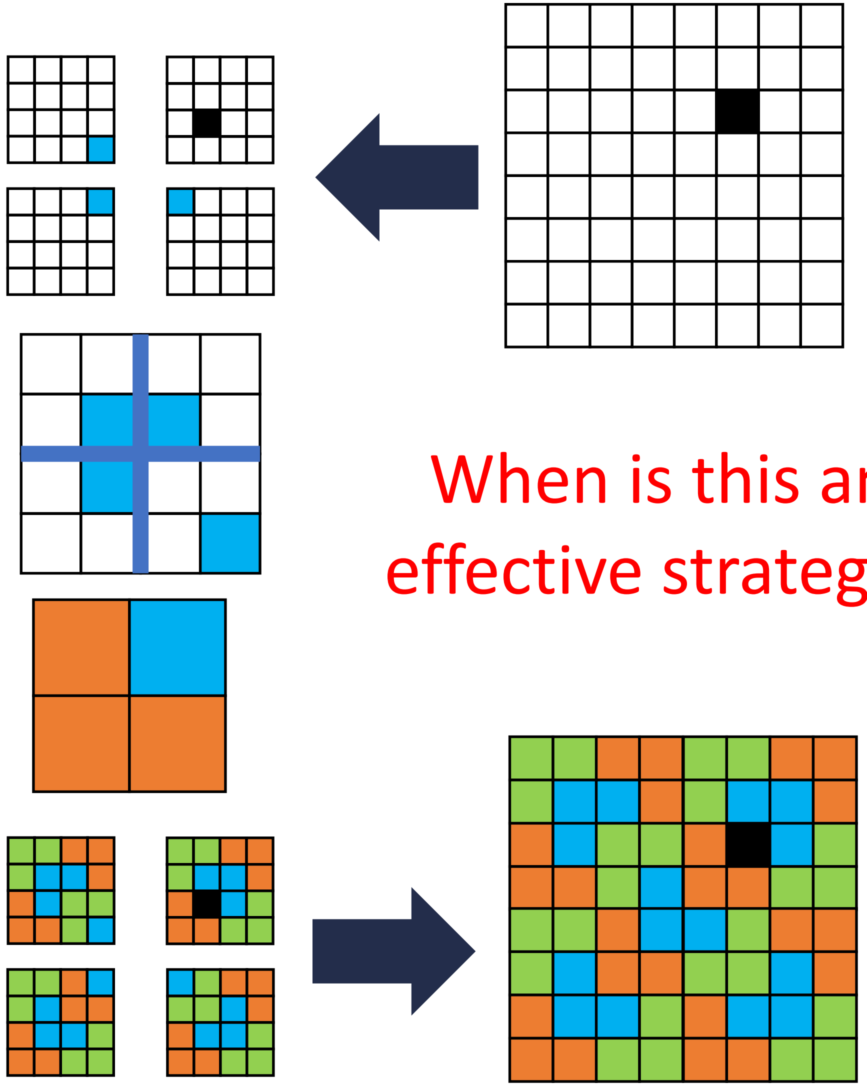
- Break the problem into multiple subproblems, each smaller instances of the original

**Conquer**:

- If the suproblems are “large”:
  - Solve each subproblem recursively
- If the subproblems are “small”:
  - Solve them directly (base case)

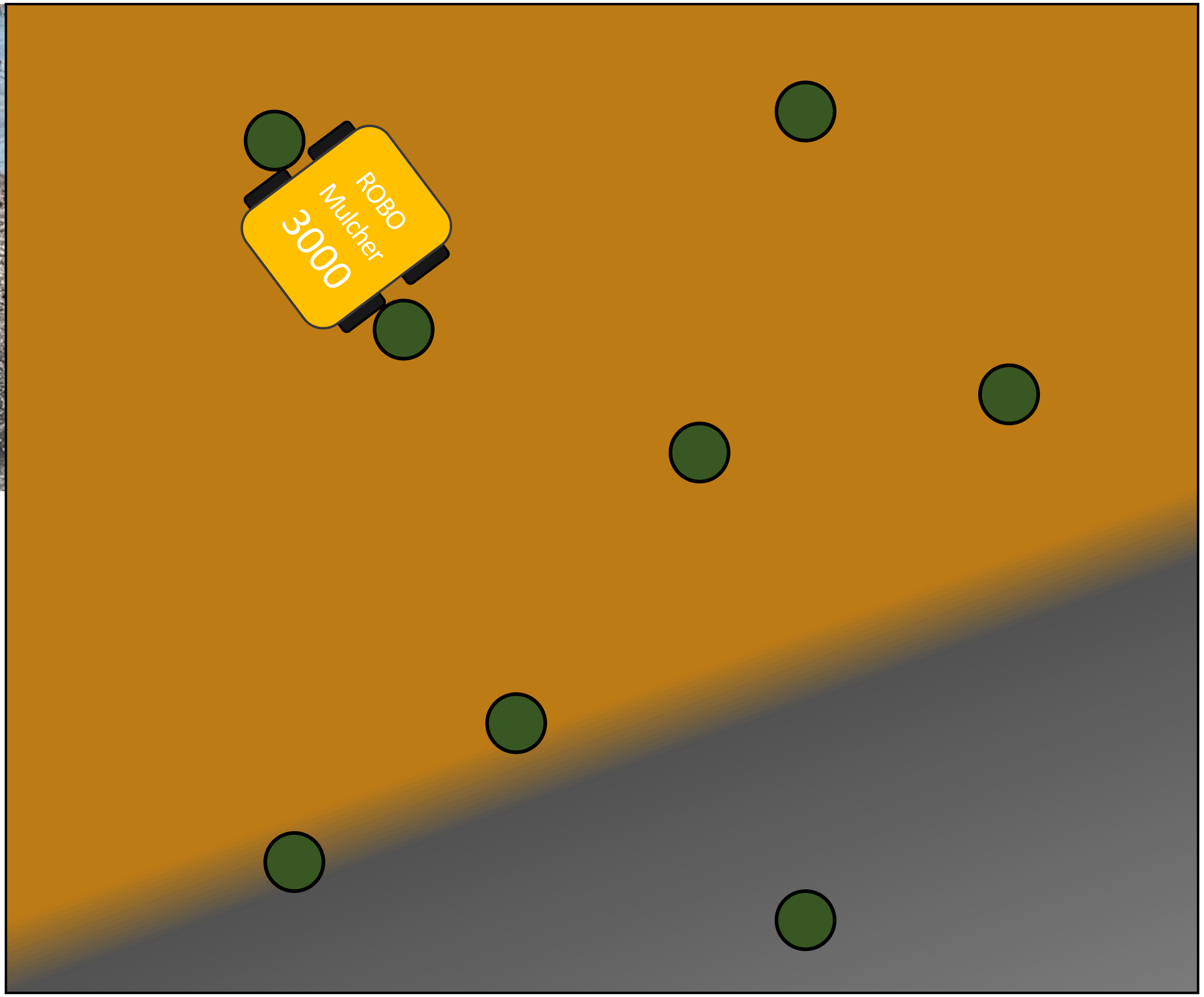
**Combine**:

- Merge solutions to subproblems to obtain solution for original problem



When is this an effective strategy?

# Constraints: Trees and Plants



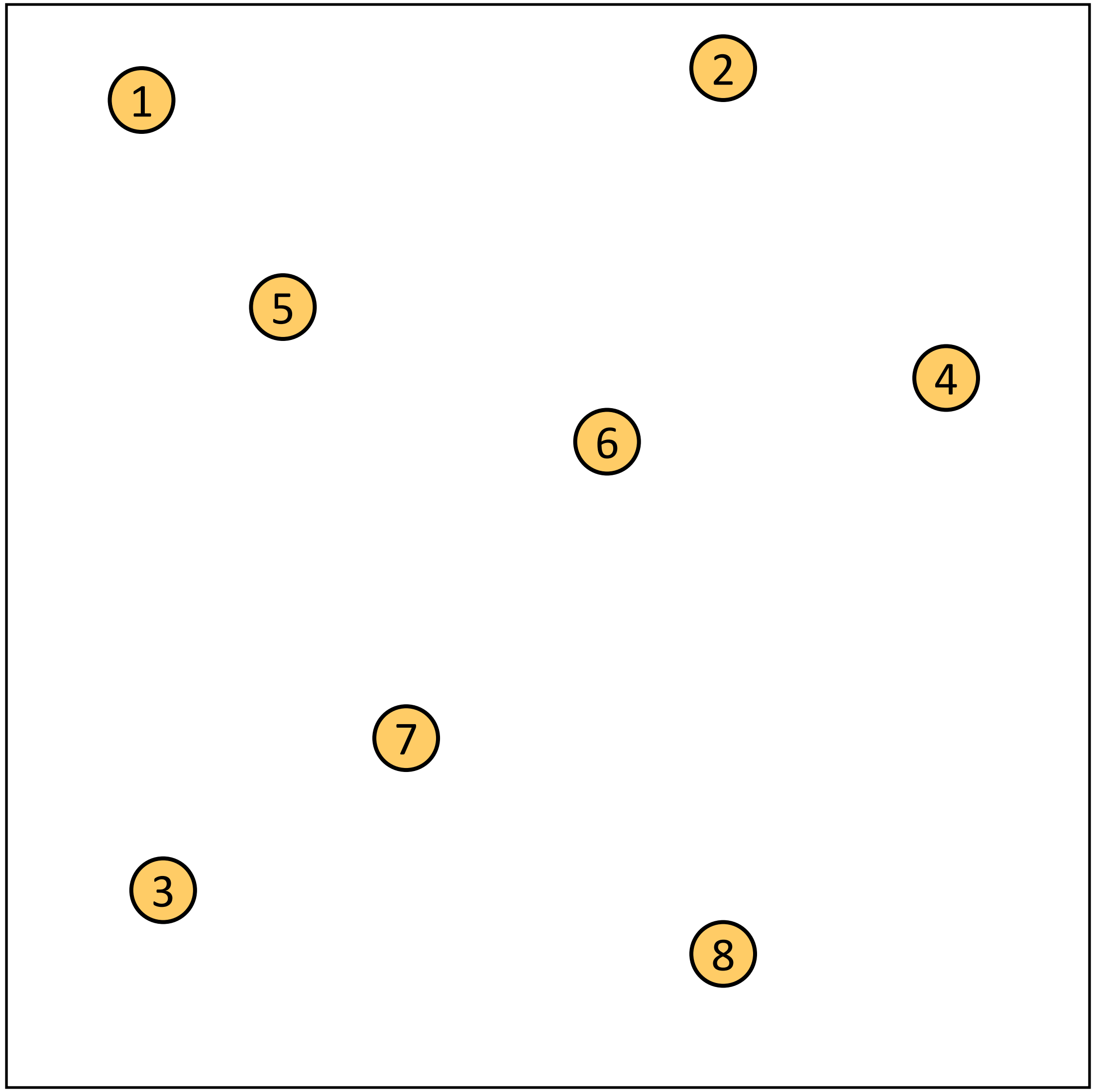

How wide can the robot be?

**Objective:** find closest pair of trees

# Closest Pair of Points

**Given:** A list of points

**Return:** Pair of points with smallest distance apart



# Closest Pair of Points: Divide and Conquer

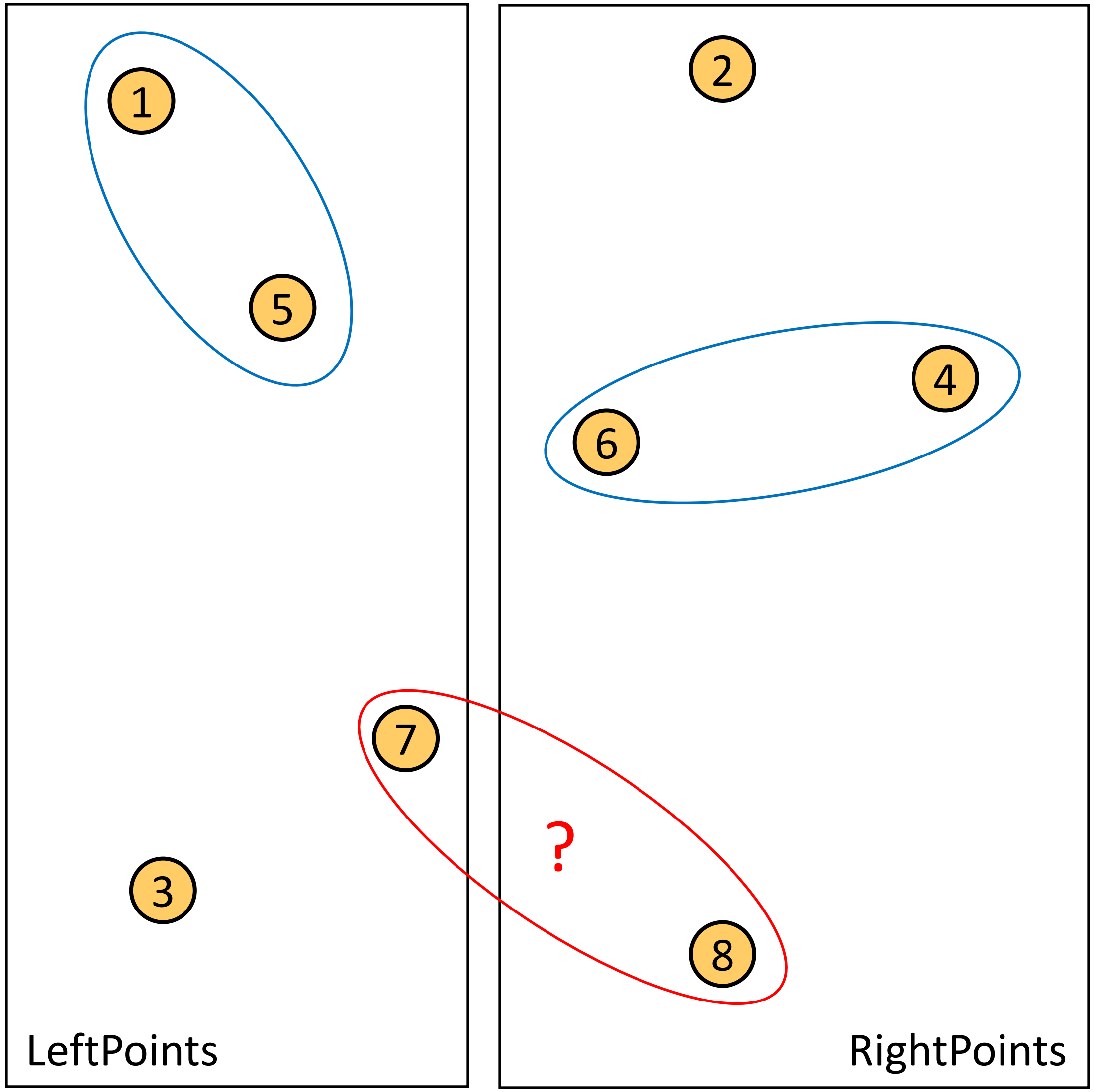
**Initialization:** Sort points by $x$-coordinate

**Divide:** Partition points into two lists of points based on $x$-coordinate

**Conquer:** Recursively compute the closest pair of points in each list

**Combine:**

- Construct list of points in the boundary
- Sort boundary points by $y$-coordinate
- Compare each point in boundary to 15 points above it and save the closest pair
- Output closest pair among left, right, and boundary points



# Closest Pair of Points: Divide and Conquer

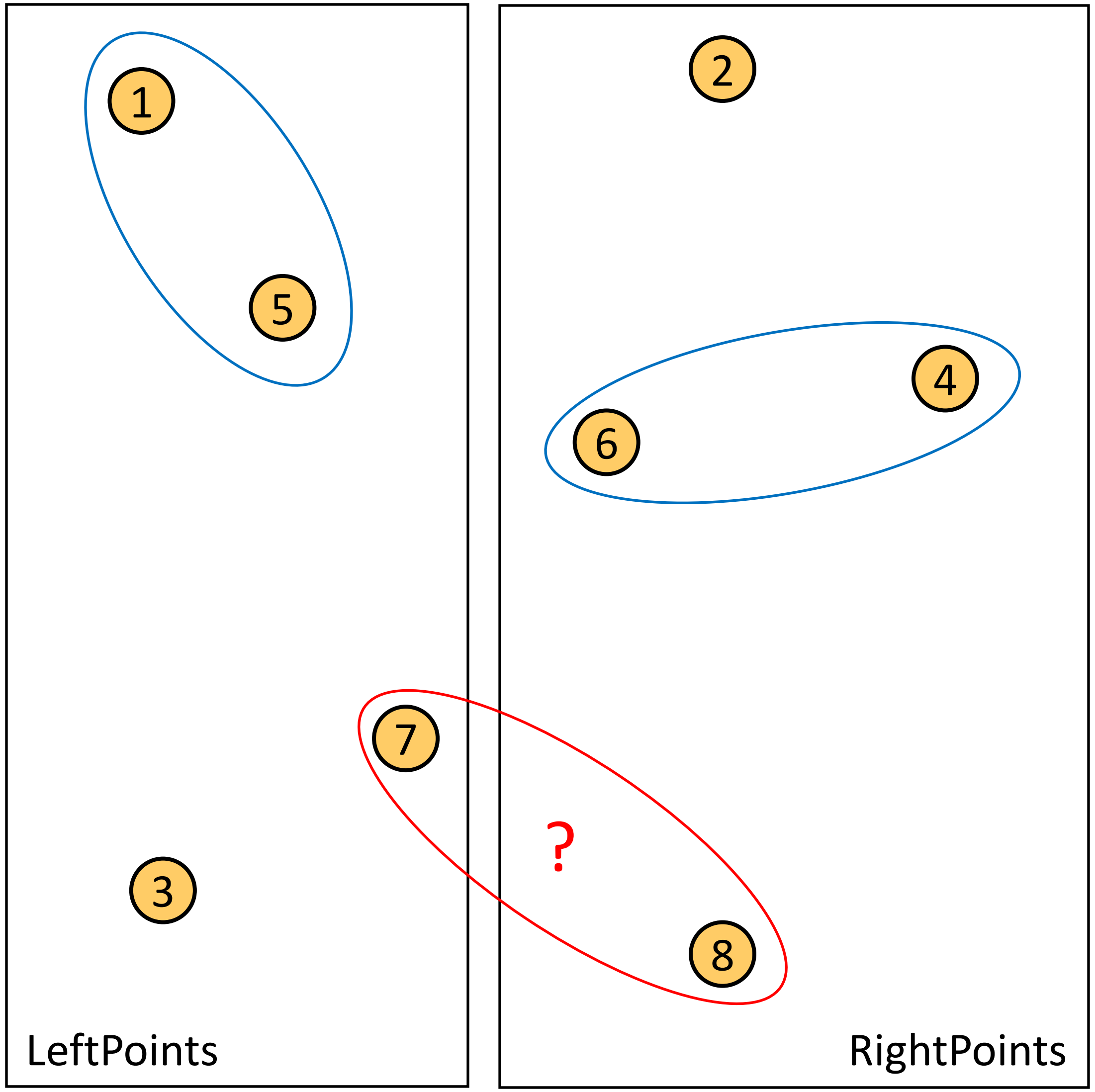
**Initialization:** Sort points by $x$-coordinate

**Divide:** Partition points into two lists of points

Looks like another $O(n \log n)$ algorithm – combine step is still too expensive

**Combine:**

- Construct list of points in the boundary
- Sort boundary points by $y$-coordinate
- Compare each point in boundary to 15 points above it and save the closest pair
- Output closest pair among left, right, and boundary points



# Closest Pair of Points: Divide and Conquer

**Initialization:** Sort points by $x$-coordinate

**Divide:** Partition points into two lists of points based on $x$-coordinate

**Conquer:** Recursively compute the closest pair of points in each list

**Combine:**
- Construct list of points in the boundary
- Sort boundary points by $y$-coordinate
- Compare each point in boundary to 15 points above it and save the closest pair
- Output closest pair among left, right, and boundary points

**Solution:** Maintain additional information in the recursion
- Minimum distance among pairs of points in the list
- List of points sorted according to $y$-coordinate

Sorting boundary points by $y$-coordinate now becomes a **merge**

# Listing Points in the Boundary

LeftPoints:

Closest Pair: $(1, 5)$, $d_{1,5}$

Sorted Points: $[3,7,5,1]$

RightPoints:

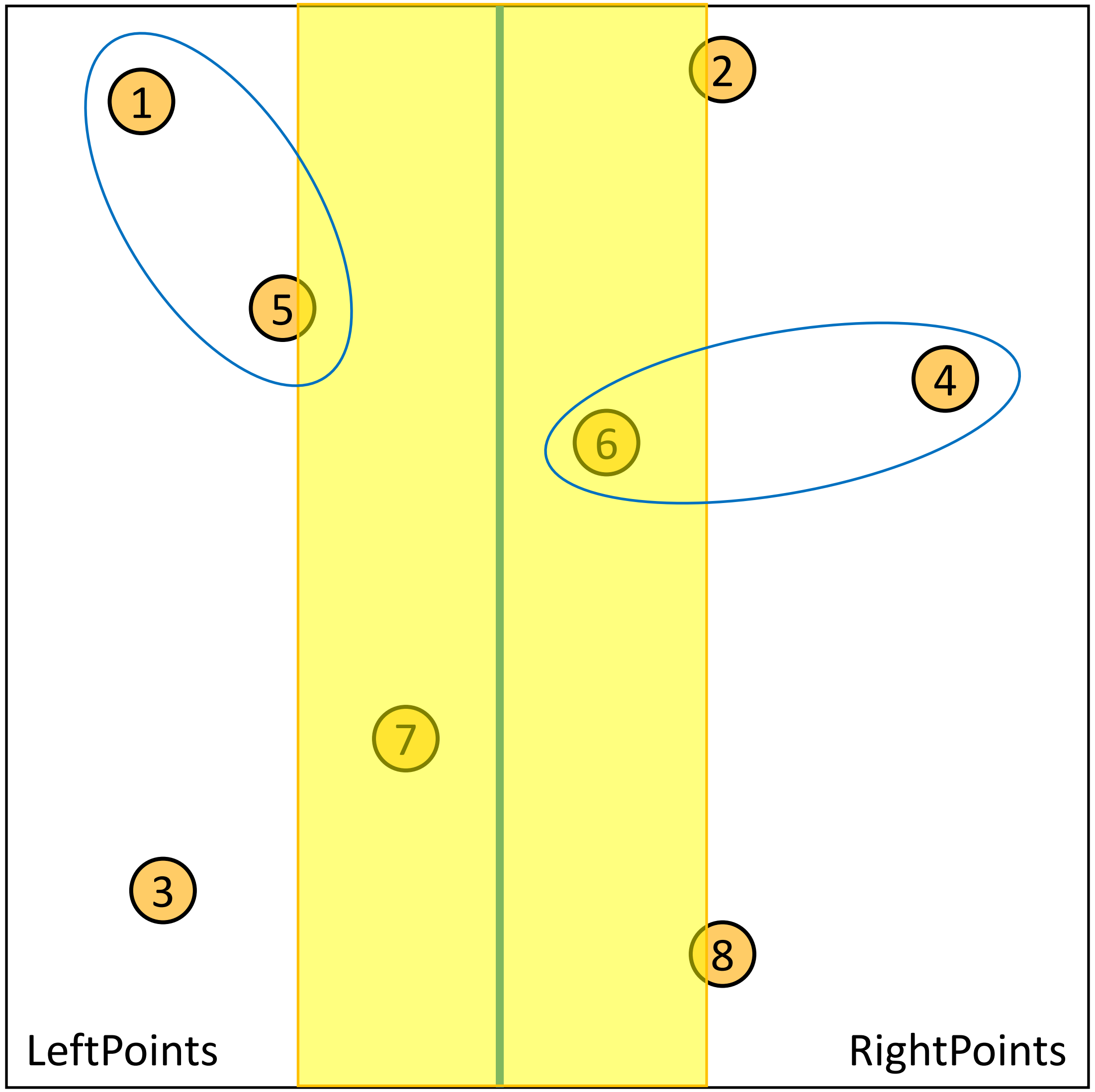
Closest Pair: $(4,6)$, $d_{4,6}$

Sorted Points: $[8,6,4,2]$

Merged Points: $[8,3,7,6,4,5,1,2]$

Boundary Points: $[8,7,6,5,2]$

Both of these lists can be computed by a *single* pass over the lists



# Closest Pair of Points: Divide and Conquer

**Initialization:** Sort points by $x$-coordinate

**Divide:** Partition points into two lists of points based on $x$-coordinate (split at the median $x$)

**Conquer:** Recursively compute the closest pair of points in each list

Base case?

**Combine:**

- Construct list of points in the runway ($x$-coordinate within distance $\delta$ of median)
- Sort runway points by $y$-coordinate
- Compare each point in runway to 7 or 15 points above it and save the closest pair
- Output closest pair among left, right, and runway points

**Possible Solution #1 to this?** Maintain additional information in the recursion

- Minimum distance among pairs of points in the list
- List of points sorted according to $y$-coordinate

Instead of sorting runway points by $y$-coordinate, use this index by y coordinate?

# Closest Pair of Points: Divide and Conquer

**Initialization:** Sort points by $x$-coordinate

**Divide:** Partition points into two lists of points based on $x$-coordinate (split at the median $x$)

**Conquer:** Recursively compute the closest pair of points in each list

Base case?

**Combine:**

- Construct list of points in the runway ($x$-coordinate within distance $\delta$ of median)
- Sort runway points by $y$-coordinate
- Compare each point in runway to 7 or 15 points above it and save the closest pair
- Output closest pair among left, right, and runway points

**Possible Solution #2 to this?**

- Merge sorted list of points by $y$-coordinate and construct list of points in the runway (sorted by $y$-coordinate)
- Compare each point in runway to 7 or 15 points above it and save the closest pair
- Output closest pair among left, right, and runway points

# Closest Pair of Points: Divide and Conquer

What is the running time?

$$\Theta(n \log n)$$

$$T(n)$$

$$T(n) = 2T(n/2) + \Theta(n)$$

**Case 2 of Master's Theorem**

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

$\Theta(n \log n)$ **Initialization:** Sort points by $x$-coordinate

$\Theta(1)$ **Divide:** Partition points into two lists of points based on $x$-coordinate (split at the median $x$)

$2T(n/2)$ **Conquer:** Recursively compute the closest pair of points in each list

**Combine:**

- $\Theta(n)$ Somehow access runway points in increasing y-coordinate order
- $\Theta(n)$ Compare each point in runway to 7 or 15 points above it and save the closest pair
- $\Theta(1)$ Output closest pair among left, right, and runway points

# Multipling Two Matrices

# Matrix Multiplication

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

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

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

Run time? $O(n^3)$

Lower Bound? $\Omega(n^2)$

# Matrix Multiplication Divide and Conquer

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 Divide and Conquer

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? $T(n) = 8T\left(\frac{n}{2}\right) + \boxed{4\left(\frac{n}{2}\right)^2}$ Cost of additions

# Matrix Multiplication Divide and Conquer

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

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

$$a = 8, b = 2, f(n) = n^2$$

Case 1!

$$n^{\log_b a} = n^{\log_2 8} = n^3$$

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

Can we do better?

# Matrix Multiplication Divide and Conquer

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

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

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

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 $AB$:

$$AB = \begin{bmatrix} Q_1 + Q_4 - Q_5 + Q_7 & Q_3 + Q_5 \\ Q_2 + Q_4 & Q_1 - Q_2 + Q_3 + Q_6 \end{bmatrix}$$

7 Multiplications

18 Additions

$$T(n) = 7T\left(\frac{n}{2}\right) + 18\frac{n^2}{4}$$

# Strassen's Algorithm

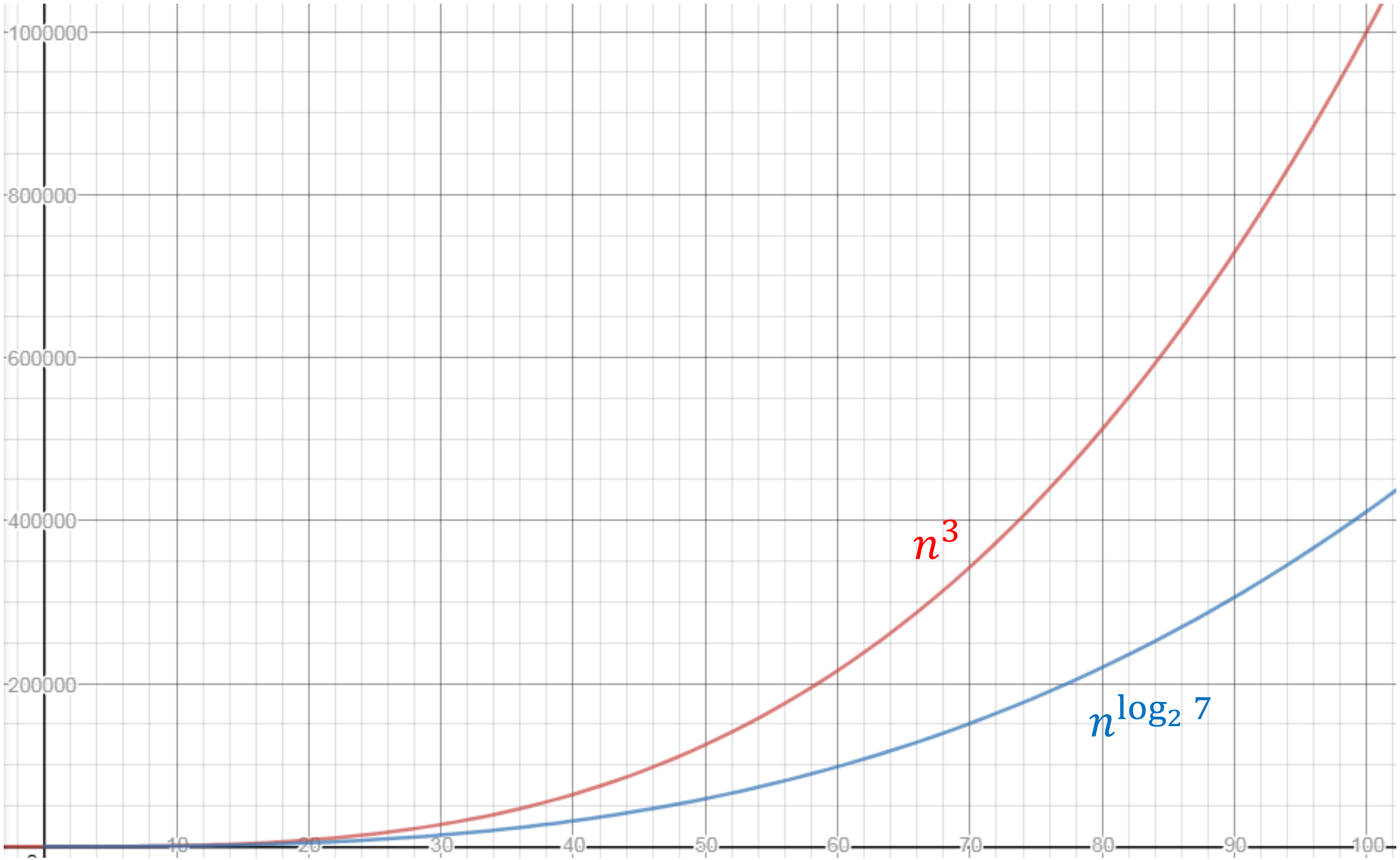
$$T(n) = 7T\left(\frac{n}{2}\right) + \frac{9}{2}n^2$$

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

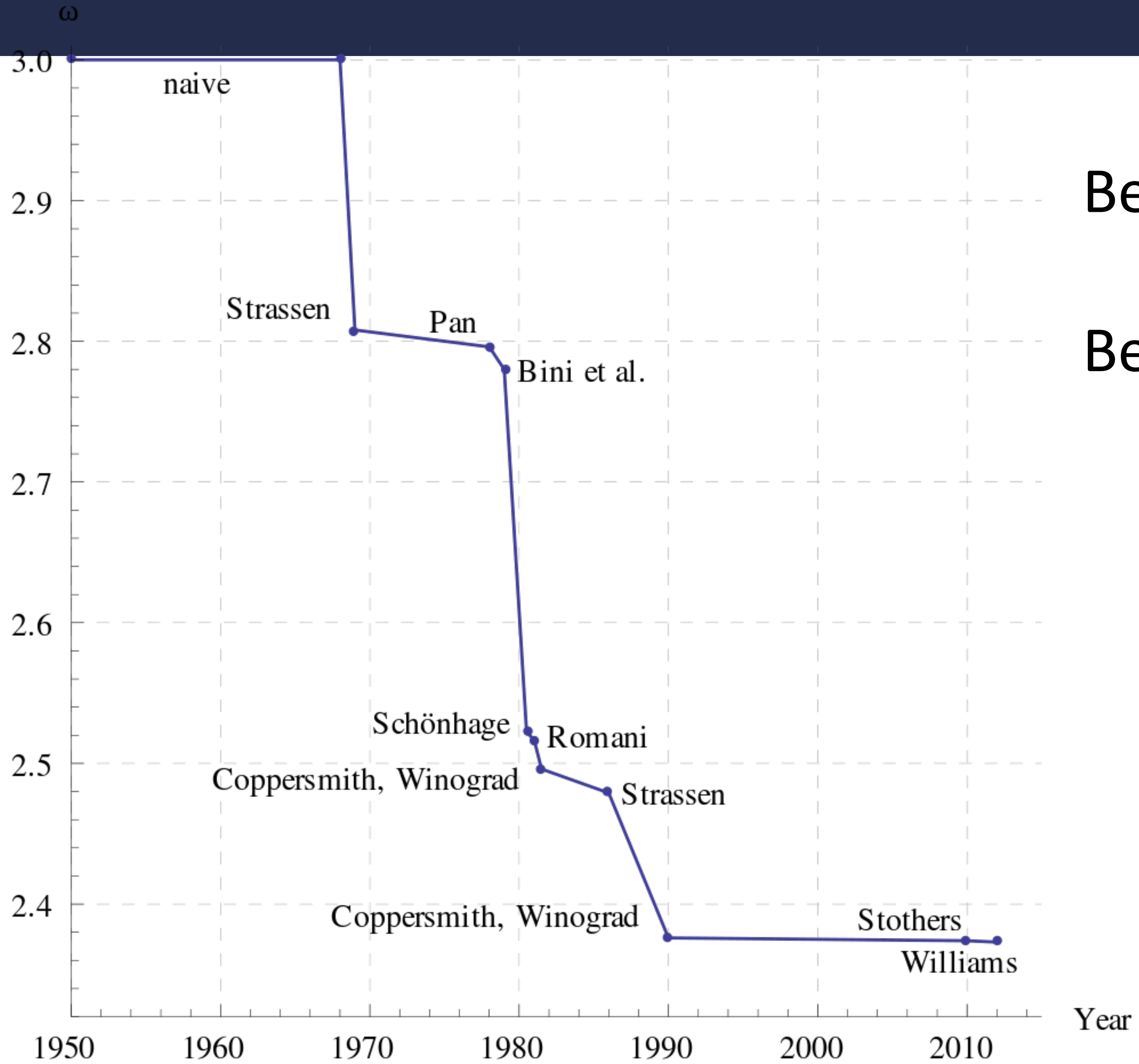
Case 1!

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

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


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 still unknown

Best lower bound: $\Omega(n^2)$

# Divide and Conquer Algorithms (Thus Far)

Mergesort

Naïve Multiplication

Karatsuba Multiplication

Closest Pair of Points

Strassen's Algorithm

What they have in common:

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

**Combine:** More complex ($\Omega(n)$)

# Quicksort

Like Mergesort:

- Divide and conquer algorithm
- $O(n \log n)$ run time (on expectation)

Unlike Mergesort:

- **Divide** step is the hard part
- Typically faster than Mergesort (often is the basis of sorting algorithms in standard library implementations)

# Quicksort

**General idea:** choose 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 Procedure (Divide Step)

**Input:** an <u>unordered</u> list, a pivot $p$

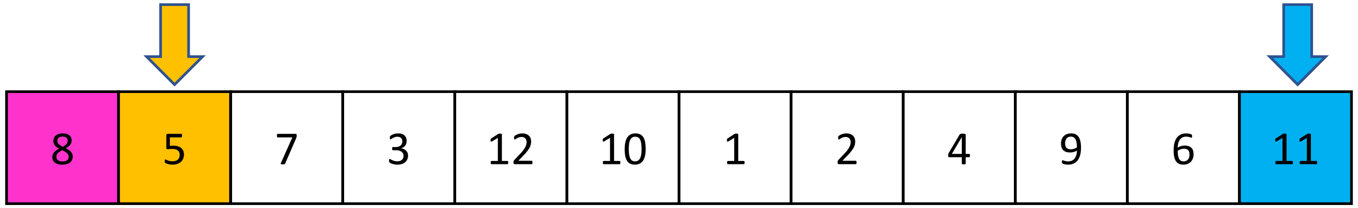
| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 8 | 5 | 7 | 3 | 12 | 10 | 1 | 2 | 4 | 9 | 6 | 11 |

**Goal:** All elements $< p$ on left, all $\geq p$ on right

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

# Partition Procedure

Initialize two pointers Begin and End

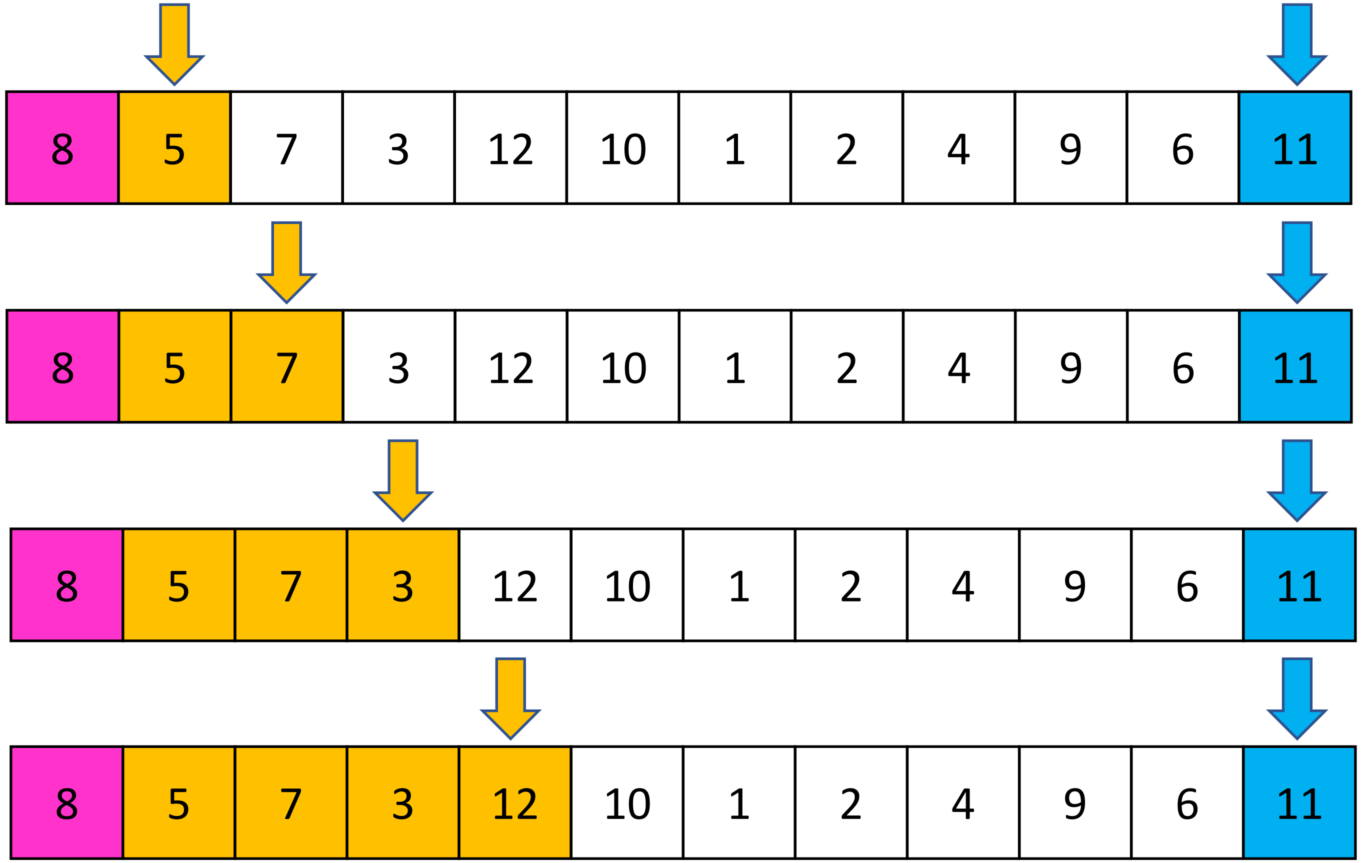


# Partition Procedure

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

Else swap Begin value with End value, move End Left

Stop when Begin = End



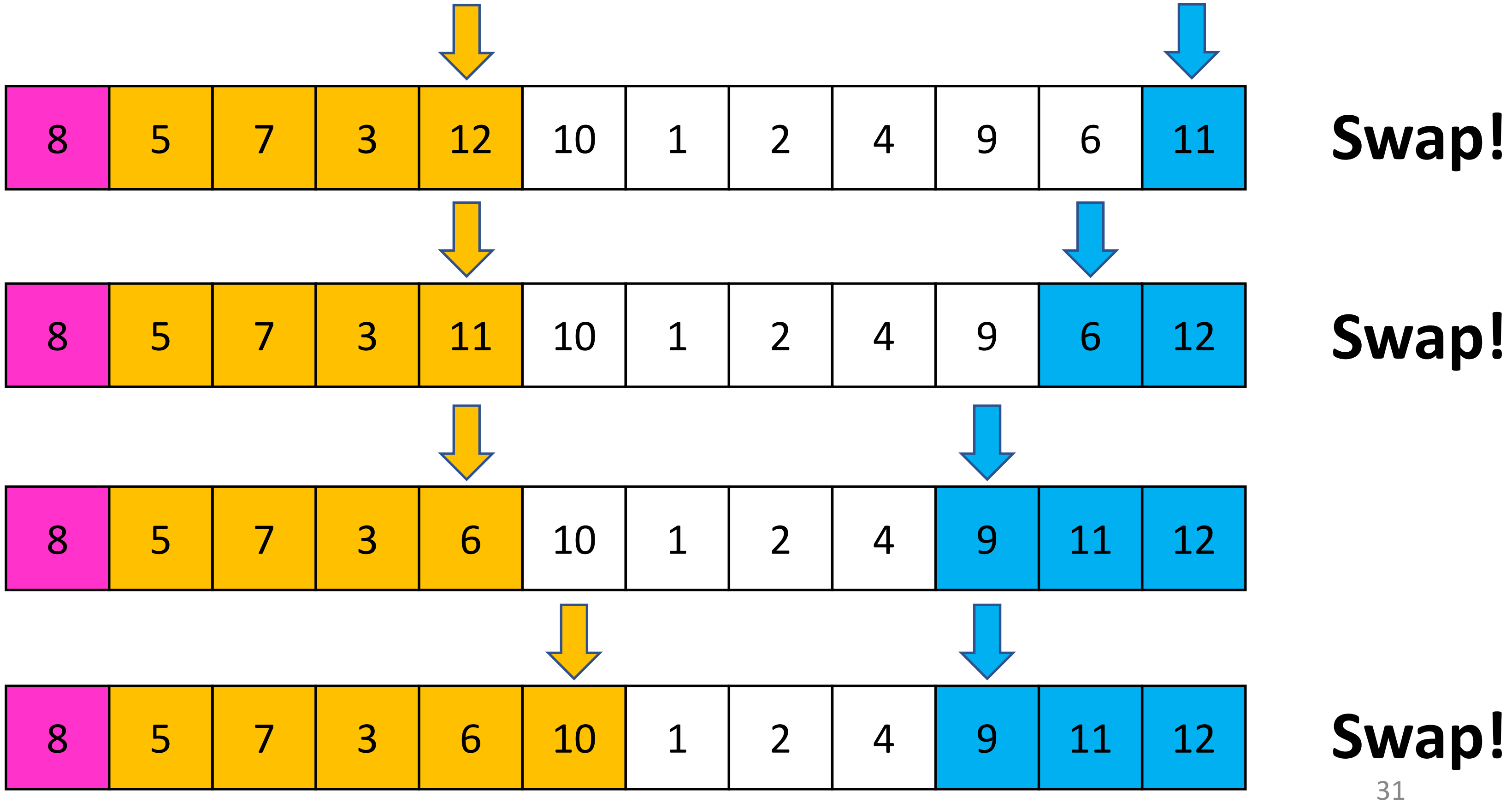

**Swap!**

# Partition Procedure

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

Else swap Begin value with End value, move End Left

Stop when Begin = End

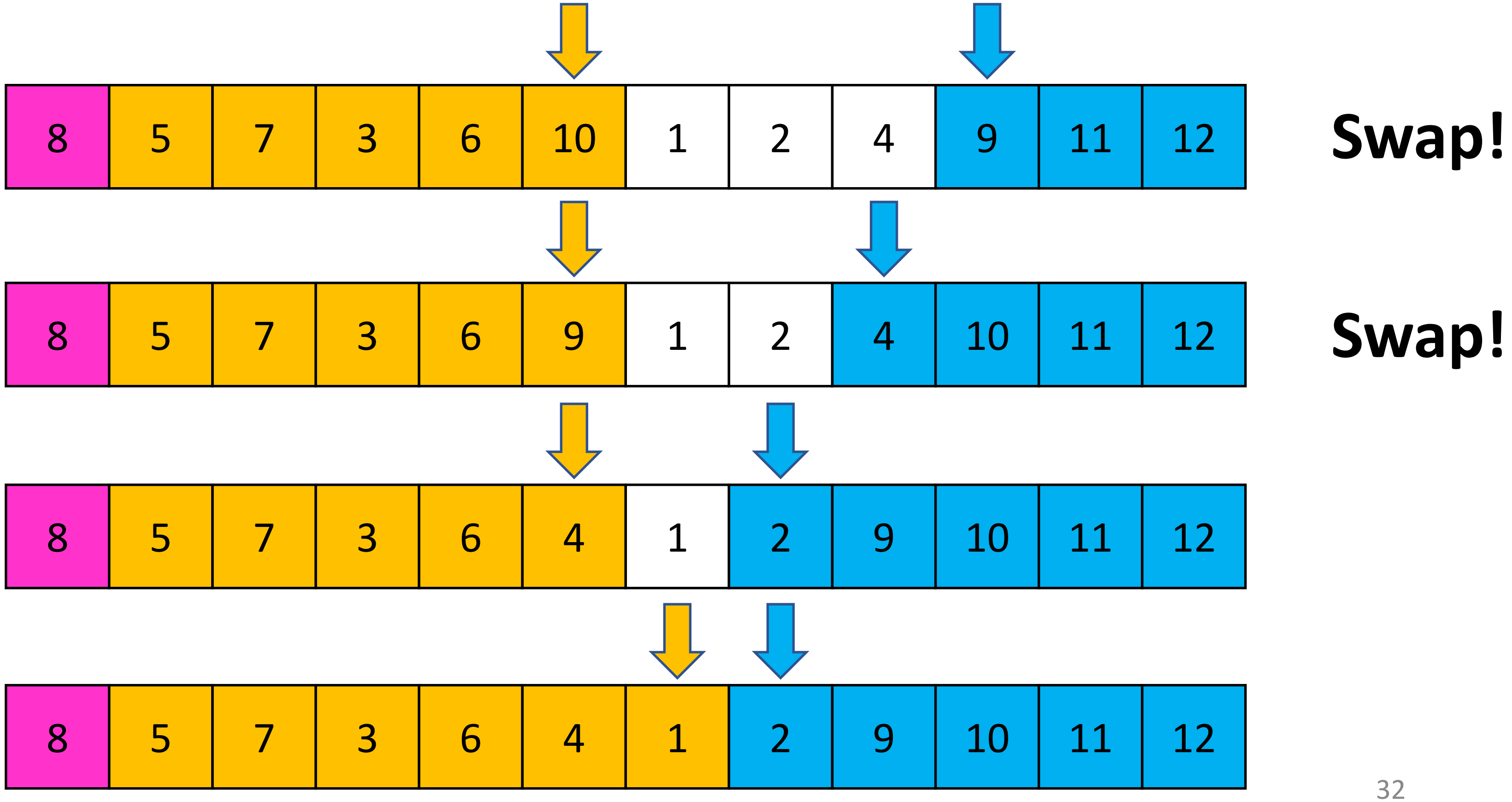


# Partition Procedure

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

Else swap Begin value with End value, move End Left

Stop when Begin = End

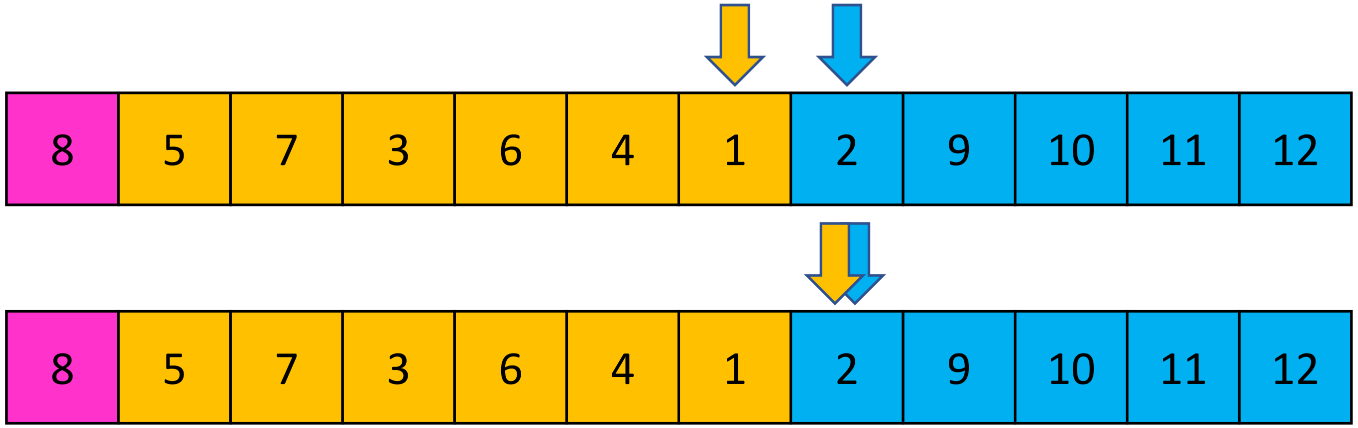


# Partition Procedure

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

Else swap Begin value with End value, move End Left

Stop when Begin = End




**Remaining item:** where do we place the pivot?

# Partition Procedure

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

Else swap Begin value with End value, move End Left

Stop when Begin = End

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

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

**Case 1:** meet at element $< p$

Swap $p$ with pointer position

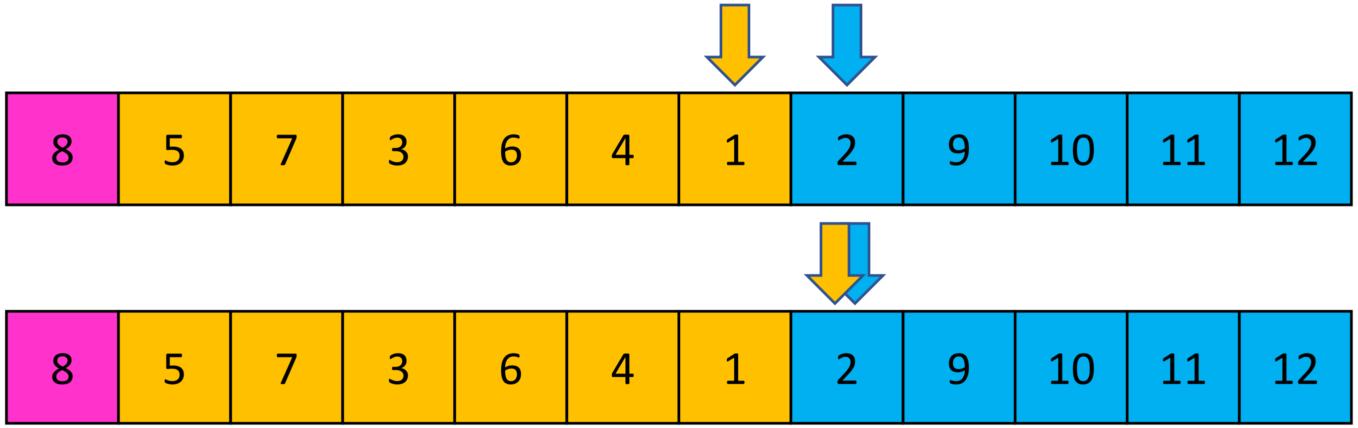
| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 5 | 7 | 3 | 6 | 4 | 1 | 8 | 9 | 10 | 11 | 12 |

# Partition Procedure

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

Else swap Begin value with End value, move End Left

Stop when Begin = End




**Case 2:** meet at element $> p$

Swap $p$ with value to the left

# Partition Procedure Summary

1. Choose the pivot $p$ to be the first element of the list
2. Initialize two pointers Begin (just after $p$), and End (at end of list)
3. While Begin < End:
   - If value of Begin $< \ p$, advance Begin to the right
   - Otherwise, swap value of Begin value with value of End value, and advance End to the left
4. If pointers meet at element $< p$: swap $p$ with pointer position
5. Otherwise, if pointers meet at element $> p$: swap $p$ with value to the left

Run time? $\Theta(n)$

# Conquer Step

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

All elements $< p$

All elements $> p$

Exactly where it belongs!

Recursively sort Left and Right sublists

# Quicksort Run Time (Optimistic)

If the pivot is 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

$$T(n) = 2T(n/2) + n = \Theta(n \log n)$$

# Quicksort Run Time (Worst-Case)

If the pivot is the extreme (min/max):

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 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

$$
\begin{aligned}
T(n) &= T(n-1) + n \\
&= n + (n-1) + \cdots + 2 + 1 \\
&= \frac{n(n+1)}{2} = \Theta(n^2)
\end{aligned}
$$

# 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 |

Then we shorten by 1 each time

$$T(n) = \Theta(n^2)$$

# How to Choose the Pivot?

Good choice: $\Theta(n \log n)$

Bad choice: $\Theta(n^2)$

# Good Pivot

What makes a good pivot?

- Roughly even split between left and right
- Ideally: median

Can we find median in linear time?

- Yes! <u>Quickselect algorithm</u>

# Quickselect Algorithm

Algorithm to compute the $i^{\text{th}}$ order statistic

- $i^{\text{th}}$ smallest element in the list
- $1^{\text{st}}$ order statistic: minimum
- $n^{\text{th}}$ order statistic: maximum
- $(n/2)^{\text{th}}$ order statistic: median

# Quickselect Algorithm

Finds $i^{\text{th}}$ order statistic

**General idea:** choose a pivot element, partition around the pivot, and recurse on sublist containing index $i$

**Divide:** select pivot element $p$, Partition($p$)

**Conquer:**

- if $i =$ index of $p$, then we are done and return $p$
- if $i <$ index of $p$ recurse left. Otherwise, recurse right

**Combine:** Nothing!

# 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.

# Partition Procedure (Divide Step)

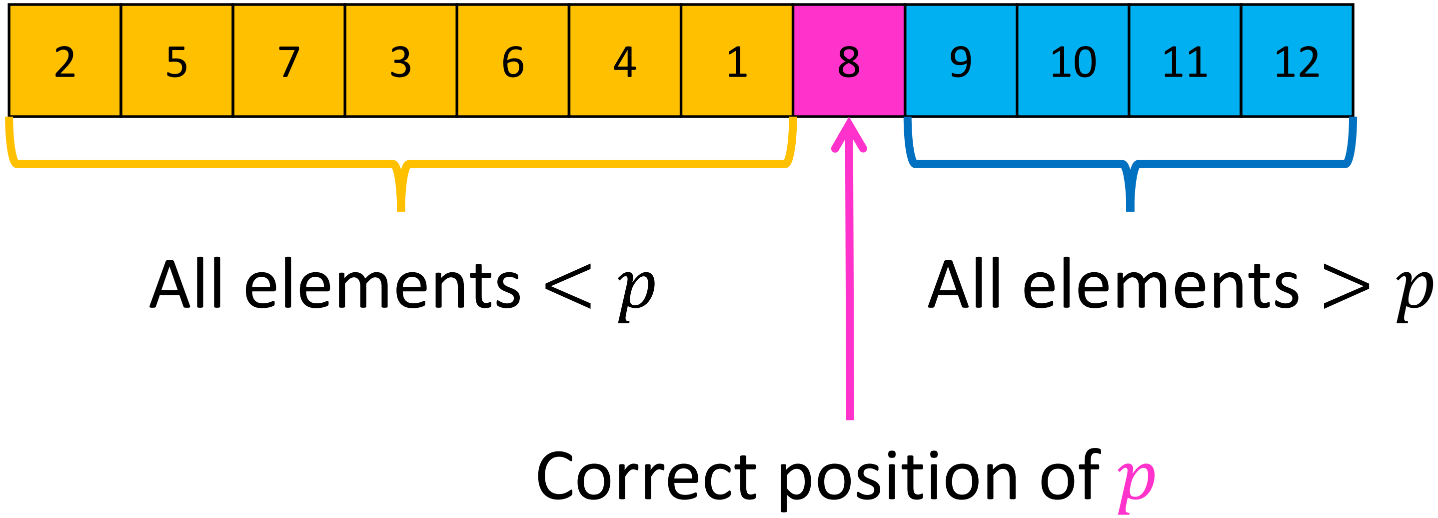
**Input:** an <u>unordered</u> list, a pivot $p$

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

**Goal:** All elements $< p$ on left, all $\geq p$ on right

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

# Conquer Step




Recurse on sublist that contains index $i$

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

# Quickselect Run Time (Optimistic)

If the pivot is 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

$$T(n) = T(n/2) + n = \Theta(n)$$

# Quickselect Run Time (Worst-Case)

If the pivot is the extreme (min/max):

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 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

$$T(n) = T(n-1) + n \; = \Theta(n^2)$$

# How to Choose the Pivot?

Good choice: $\Theta(n)$

Bad choice: $\Theta(n^2)$

# Good Pivot

What makes a good pivot?

- Roughly even split between left and right
- Ideally: median

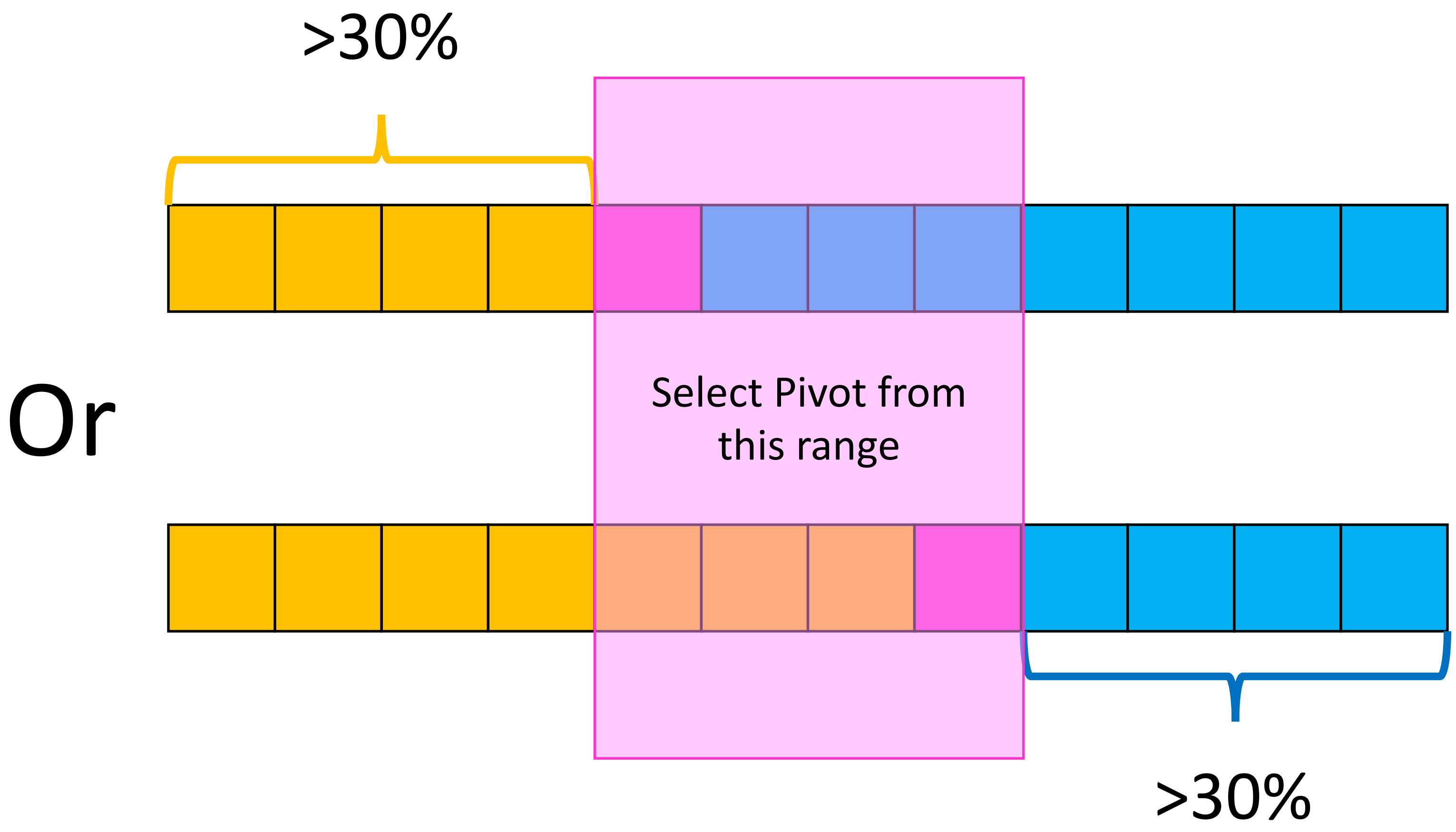
But this is the problem that Quickselect is supposed to solve!

Déjà vu?

**What's next:** an algorithm for choosing a "decent" pivot (median of medians)

# Good Pivot

Decent pivot: both sides of Pivot >30%



# Median of Medians

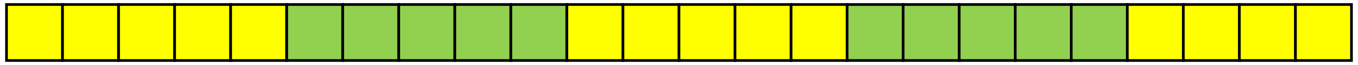
Fast way to select a “good” pivot

Guarantees pivot is greater than ≈30% of elements and less than ≈30% of the elements

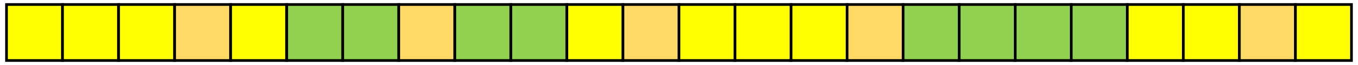
**Main idea:** break list into blocks, find the median of each blocks, use the median of those medians

# Median of Medians

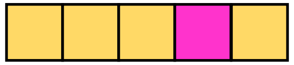
1. Break list into blocks of size 5



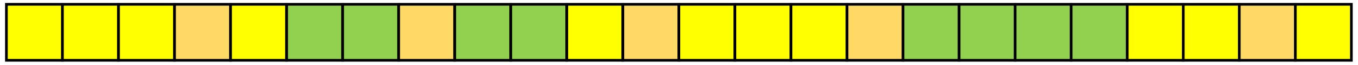
2. Find the median of each chunk



3. Return median of medians (using Quickselect)

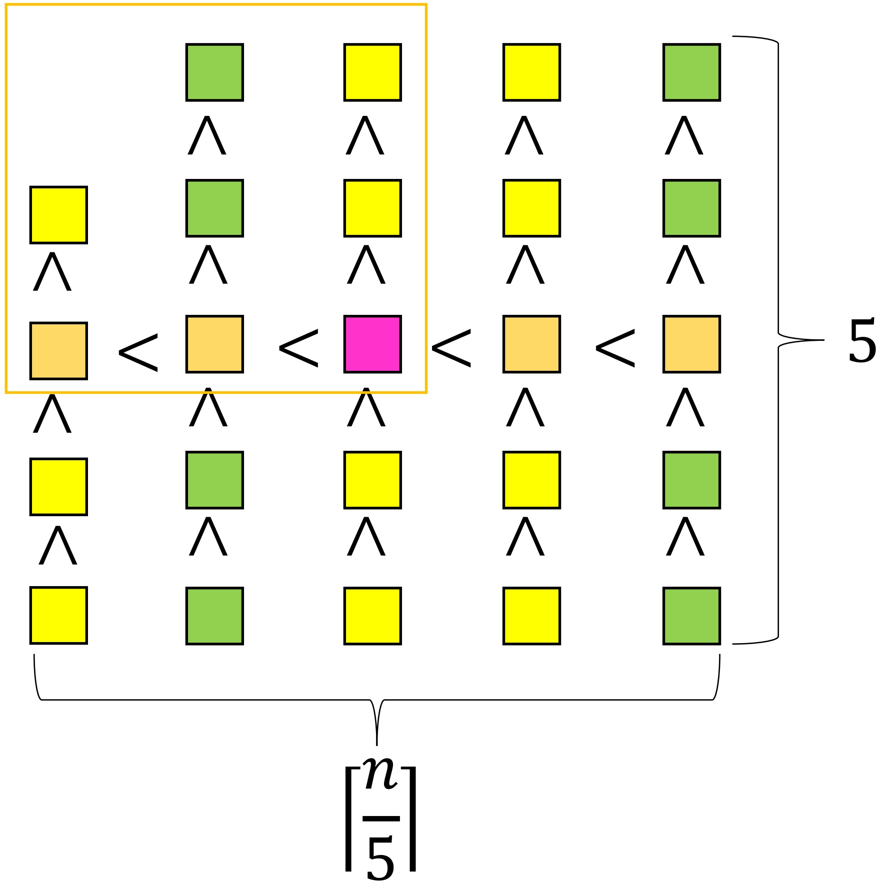


# Median of Medians



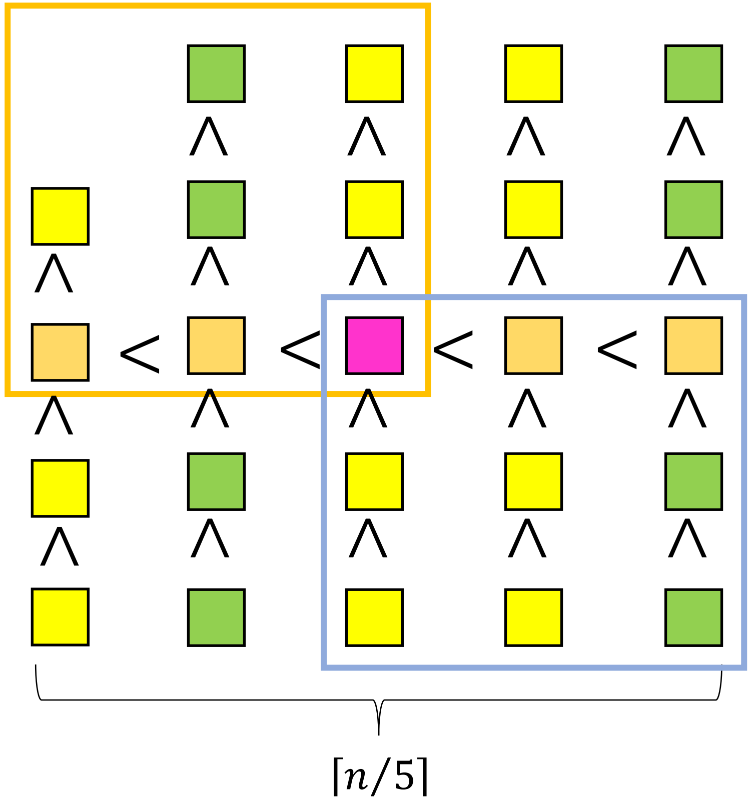
Each chunk sorted, chunks ordered by their medians

MedianofMedians
is larger than all
of these



# Median of Medians

MedianofMedians is larger than all of these




Elements smaller than MedianofMedians:

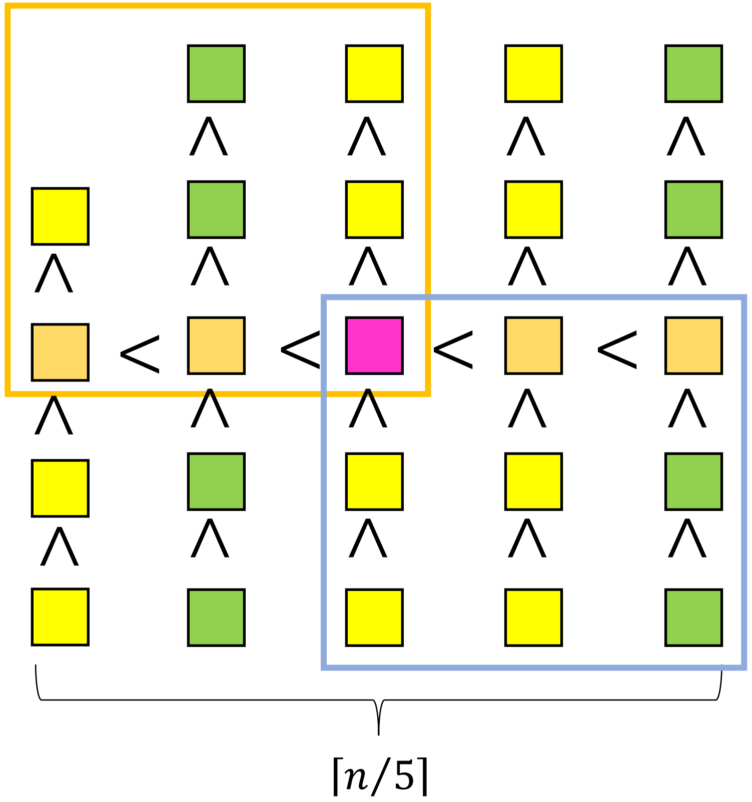
$$3\left(\left\lceil \frac{1}{2} \cdot \left\lceil \frac{n}{5} \right\rceil \right\rceil - 2\right) \geq \frac{3n}{10} - 6 \text{ elements}$$

Number of lists to the “left”

Exclude list on the endpoint, and “middle” list

# Median of Medians

MedianofMedians is larger than all of these




Elements smaller than MedianofMedians: $3\left(\left\lceil \frac{1}{2}\cdot\left\lceil\frac{n}{5}\right\rceil\right\rceil - 2\right) \geq \frac{3n}{10} - 6$ elements

Elements greater than MedianofMedians: $3\left(\left\lceil \frac{1}{2}\cdot\left\lceil\frac{n}{5}\right\rceil\right\rceil - 2\right) \geq \frac{3n}{10} - 6$ elements

# Quickselect

Divide: select an element $p$ using Median of Medians, Partition($p$)

$$M(n) + \Theta(n)$$

median of medians algorithm

partition algorithm

# 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 (with index $i - p$)

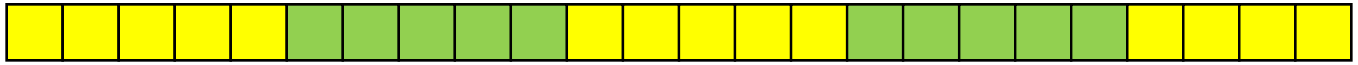
$$\leq S\left(\frac{7n}{10}\right)$$

Combine: Nothing!

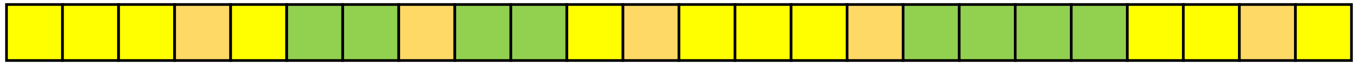
$$S(n) \leq S\left(\frac{7n}{10}\right) + M(n) + \Theta(n)$$

# Median of Medians

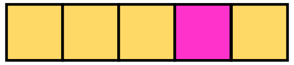
1. Break list into blocks of size 5 $\Theta(n)$



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

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

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

Combine: Nothing!

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

# 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

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

Combine: Nothing!

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

# Phew! Back to Quicksort

**Divide:** Select a pivot element, and partition about the pivot

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

Using Quickselect, always pivot about the median

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

**Conquer:** Recursively sort left and right sublists

If pivot is the median, list is split in half each iteration

# Phew! Back to Quicksort

**Divide:** Select a pivot element, and partition about the pivot

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

Using Quickselect, always pivot about the median

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

$$T(n) = 2T(n/2) + \Theta(n)$$

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

# A Worthwhile Choice?

Using Quickselect to pick median guarantees $\Theta(n \log n)$ worst-case run-time

Approach has very large constants

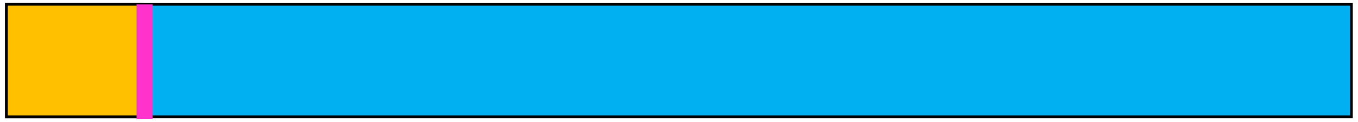
- If you really want $\Theta(n \log n)$, better off using MergeSort

More efficient approach: Random pivot

- Very small constant (very fast algorithm)
- Expected to run in $\Theta(n \log n)$ time
  - Why? Unbalanced partitions are very unlikely

# Quicksort Running Time

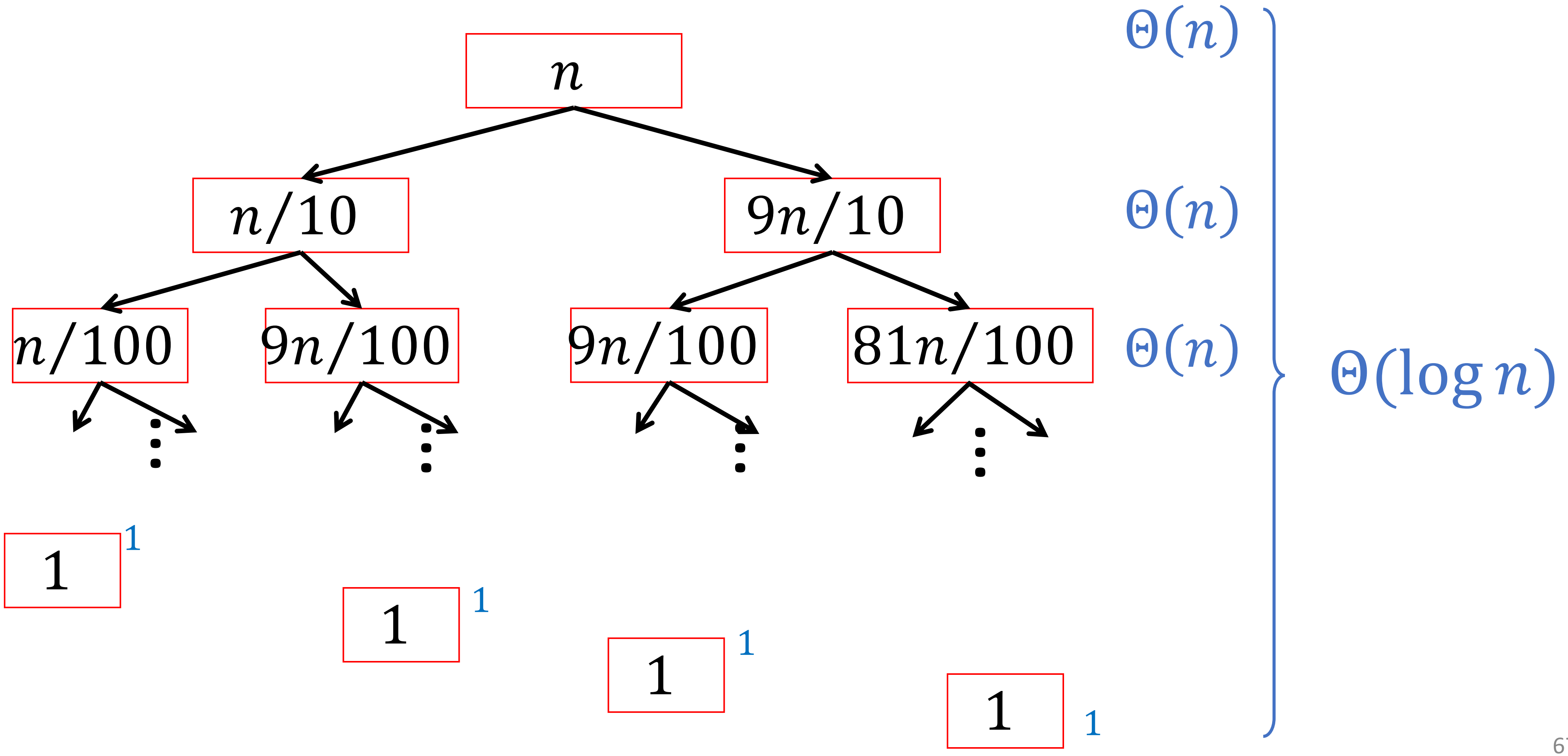
If the pivot is always $(n/10)^{\text{th}}$ order statistic:



$$T(n) = T(n/10) + T(9n/10) + \Theta(n)$$

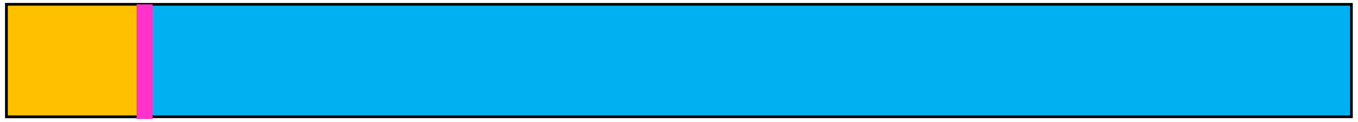
# Quicksort Running Time

$$T(n) = T(n/10) + T(9n/10) + \Theta(n)$$



# Quicksort Running Time

If the pivot is always $(n/10)^{\text{th}}$ order statistic:



$$T(n) = T(n/10) + T(9n/10) + \Theta(n)$$
$$= \Theta(n \log n)$$

This is true if the pivot is any $(n/k)^{\text{th}}$ order statistic for any constant $k > 1$ (as long as the size of the smaller list is a constant fraction of the full list, we get $\Theta(n \log n)$ running time)

# Quicksort Running Time

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

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

Then we shorten by $d$ each time

$$T(n) = T(n-d) + n$$
$$= \Theta(n^2)$$

What's the probability of this occurring (for a <u>random</u> pivot)?

# Probability of Always Choosing $d^{\text{th}}$ Order Statistic

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:

Very small probability!

$$\frac{d}{n} \times \frac{d}{n-d} \times \frac{d}{n-2d} \times \cdots \times \frac{d}{2d} \times 1 = \left(\frac{n}{d} \times \left(\frac{n}{d} - 1\right) \times \cdots \times 1\right)^{-1} = \frac{1}{\left(\frac{n}{d}\right)!}$$

# Formal Argument for $n \log n$ Average

We will focus on counting the number of comparisons

**For simplicity:** suppose all elements are distinct

Quicksort only compares against a pivot

- Element $i$ only compared to element $j$ if one of them was the pivot

# Formal Argument for $n \log n$ Average

What is the probability of comparing two given elements?

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

Consider the sorted version of the list

**Observation:** Adjacent elements must be compared

- **Why?** Otherwise I would not know their order
- **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?

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

Consider the sorted version of the list

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

Assuming pivot is chosen uniformly at random

Elements only compared if 1 or 12 was chosen as the first pivot since otherwise they are in <u>different</u> sublists

# Formal Argument for $n \log n$ Average

What is the probability of comparing two given elements?

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| | | | | $i$ | | | | $j$ | | | |

**Case 1:** Pivot less than $i$

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

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

# Formal Argument for $n \log n$ Average

What is the probability of comparing two given elements?

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| | | | | $i$ | | | | $j$ | | | |

**Case 1:** Pivot less than $i$

Then sublist $[i, i+1, \dots, j]$ will be

processed in future invocation of

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

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

# Formal Argument for $n \log n$ Average

What is the probability of comparing two given elements?

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| | | | | $i$ | | | | $j$ | | | |

**Case 2:** Pivot greater than $j$

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

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

# Formal Argument for $n \log n$ Average

What is the probability of comparing two given elements?

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| | | | | $i$ | | | | $j$ | | | |

**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?

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| | | | | $i$ | | | | $j$ | | | |

**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?

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| | | | | $i$ | | | | $j$ | | | |

**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 element $i$ with element $j$:

$$\Pr[\text{we compare } i \text{ and } j] = \frac{2}{j - i + 1}$$

# Formal Argument for $n \log n$ Average

Probability of comparing element $i$ with element $j$:

$$\Pr[\text{we compare } i \text{ and } j] = \frac{2}{j - i + 1}$$

<u>Expected</u> 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:** $\displaystyle\sum_{k=1}^{n} \frac{1}{k} = \Theta(\log n)$

**Intuition (not proof!):**

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

# Formal Argument for $n \log n$ Average

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

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

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