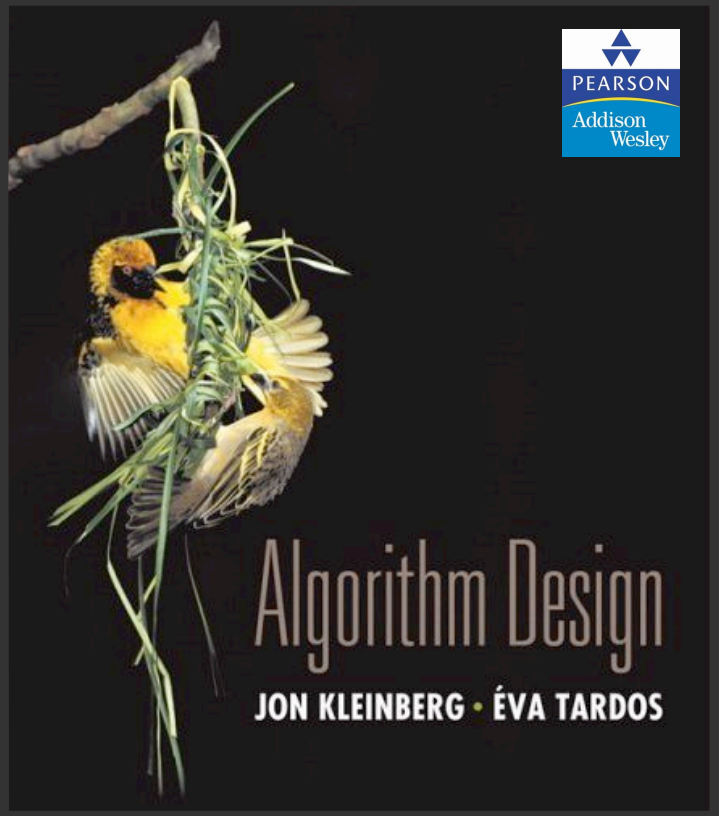



# 5. Divide And Conquer I

- *mergesort*
- *counting inversions*
- *closest pair of points*
- *randomized quicksort*
- *median and selection*

Lecture slides by Kevin Wayne

http://www.cs.princeton.edu/~wayne/kleinberg-tardos

Last updated on Oct 2, 2013 9:51 AM

# Divide-and-conquer paradigm

Divide-and-conquer.

- Divide up problem into several subproblems.
- Solve each subproblem recursively.
- Combine solutions to subproblems into overall solution.

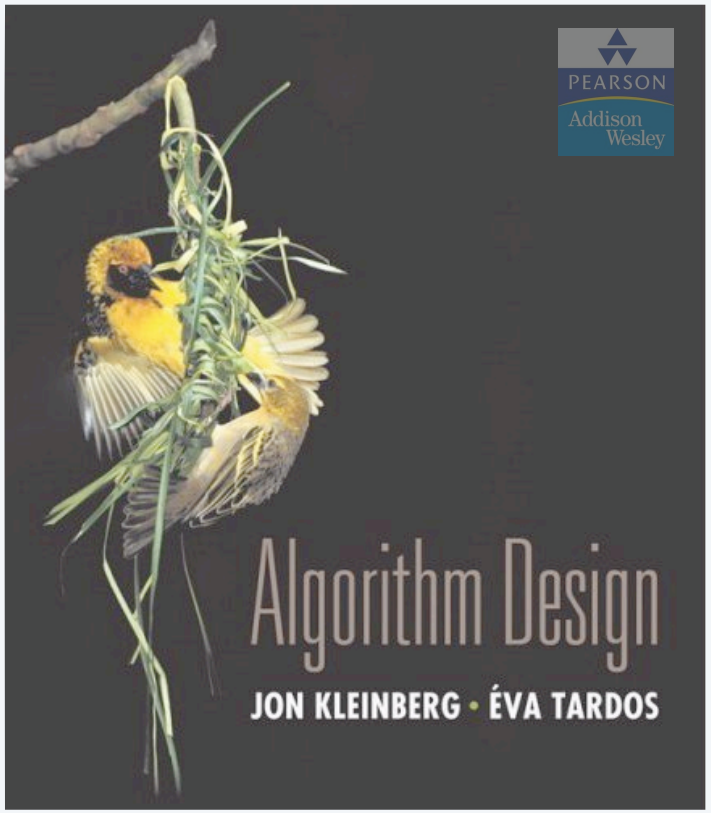
Most common usage.

- Divide problem of size $n$ into two subproblems of size $n/2$ in linear time.
- Solve two subproblems recursively.
- Combine two solutions into overall solution in linear time.

Consequence.

- Brute force: $\Theta(n^2)$.
- Divide-and-conquer: $\Theta(n \log n)$.


**attributed to Julius Caesar**



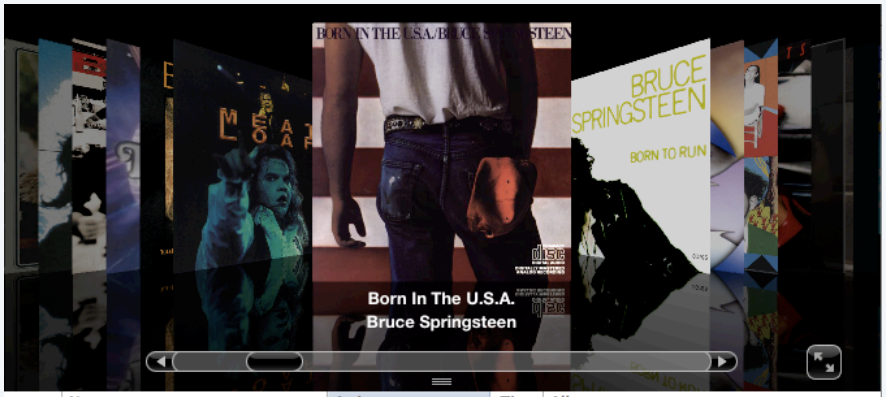

SECTION 5.1

# 5. DIVIDE AND CONQUER

- *mergesort*
- *counting inversions*
- *closest pair of points*
- *randomized quicksort*
- *median and selection*

# Sorting problem

Problem. Given a list of $n$ elements from a totally-ordered universe, rearrange them in ascending order.




| | Name | Artist | Time | Album |
|---|---|---|---|---|
| 12 | Let It Be | The Beatles | 4:03 | Let It Be |
| 13 | Take My Breath Away | BERLIN | 4:13 | Top Gun – Soundtrack |
| 14 | Circle Of Friends | Better Than Ezra | 3:27 | Empire Records |
| 15 | Dancing With Myself | Billy Idol | 4:43 | Don't Stop |
| 16 | Rebel Yell | Billy Idol | 4:49 | Rebel Yell |
| 17 | Piano Man | Billy Joel | 5:36 | Greatest Hits Vol. 1 |
| 18 | Pressure | Billy Joel | 3:16 | Greatest Hits, Vol. II (1978 – 1985) (Disc 2) |
| 19 | The Longest Time | Billy Joel | 3:36 | Greatest Hits, Vol. II (1978 – 1985) (Disc 2) |
| 20 | Atomic | Blondie | 3:50 | Atomic: The Very Best Of Blondie |
| 21 | Sunday Girl | Blondie | 3:15 | Atomic: The Very Best Of Blondie |
| 22 | Call Me | Blondie | 3:33 | Atomic: The Very Best Of Blondie |
| 23 | Dreaming | Blondie | 3:06 | Atomic: The Very Best Of Blondie |
| 24 | Hurricane | Bob Dylan | 8:32 | Desire |
| 25 | The Times They Are A-Changin' | Bob Dylan | 3:17 | Greatest Hits |
| 26 | Livin' On A Prayer | Bon Jovi | 4:11 | Cross Road |
| 27 | Beds Of Roses | Bon Jovi | 6:35 | Cross Road |
| 28 | Runaway | Bon Jovi | 3:53 | Cross Road |
| 29 | Rasputin (Extended Mix) | Boney M | 5:50 | Greatest Hits |
| 30 | Have You Ever Seen The Rain | Bonnie Tyler | 4:10 | Faster Than The Speed Of Night |
| 31 | Total Eclipse Of The Heart | Bonnie Tyler | 7:02 | Faster Than The Speed Of Night |
| 32 | Straight From The Heart | Bonnie Tyler | 3:41 | Faster Than The Speed Of Night |
| 33 | Holding Out For A Hero | Bonny Tyler | 5:49 | Meat Loaf And Friends |
| 34 | Dancing In The Dark | Bruce Springsteen | 4:05 | Born In The U.S.A. |
| 35 | Thunder Road | Bruce Springsteen | 4:51 | Born To Run |
| 36 | Born To Run | Bruce Springsteen | 4:30 | Born To Run |
| 37 | Jungleland | Bruce Springsteen | 9:34 | Born To Run |

# Sorting applications

Obvious applications.

- Organize an MP3 library.
- Display Google PageRank results.
- List RSS news items in reverse chronological order.

Some problems become easier once elements are sorted.

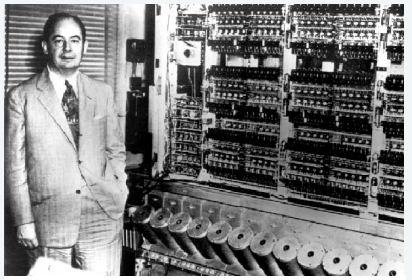
- Identify statistical outliers.
- Binary search in a database.
- Remove duplicates in a mailing list.

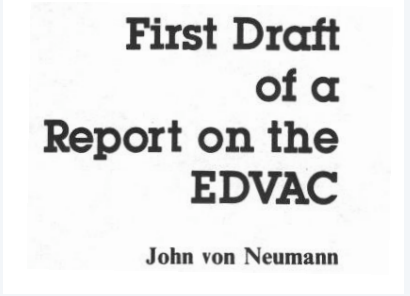
Non-obvious applications.

- Convex hull.
- Closest pair of points.
- Interval scheduling / interval partitioning.
- Minimum spanning trees (Kruskal's algorithm).
- Scheduling to minimize maximum lateness or average completion time.
- ...

# Mergesort

- Recursively sort left half.
- Recursively sort right half.
- Merge two halves to make sorted whole.






**input**

| A | L | G | O | R | I | T | H | M | S |
|---|---|---|---|---|---|---|---|---|---|

**sort left half**

| A | G | L | O | R | I | T | H | M | S |
|---|---|---|---|---|---|---|---|---|---|

**sort right half**

| A | G | L | O | R | H | I | M | S | T |
|---|---|---|---|---|---|---|---|---|---|

**merge results**

| A | G | H | I | L | M | O | R | S | T |
|---|---|---|---|---|---|---|---|---|---|

# Merging

Goal. Combine two sorted lists $A$ and $B$ into a sorted whole $C$.

- Scan $A$ and $B$ from left to right.
- Compare $a_i$ and $b_j$.
- If $a_i \leq b_j$, append $a_i$ to $C$ (no larger than any remaining element in $B$).
- If $a_i > b_j$, append $b_j$ to $C$ (smaller than every remaining element in $A$).

# A useful recurrence relation

Def. $T(n)$ = max number of compares to mergesort a list of size $\leq n$.

Note. $T(n)$ is monotone nondecreasing.

Mergesort recurrence.

$$T(n) \leq \begin{cases} 0 & \text{if } n = 1 \\ T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + n & \text{otherwise} \end{cases}$$

Solution. $T(n)$ is $O(n \log_2 n)$.

Assorted proofs. We describe several ways to prove this recurrence. Initially we assume $n$ is a power of $2$ and replace $\leq$ with $=$.

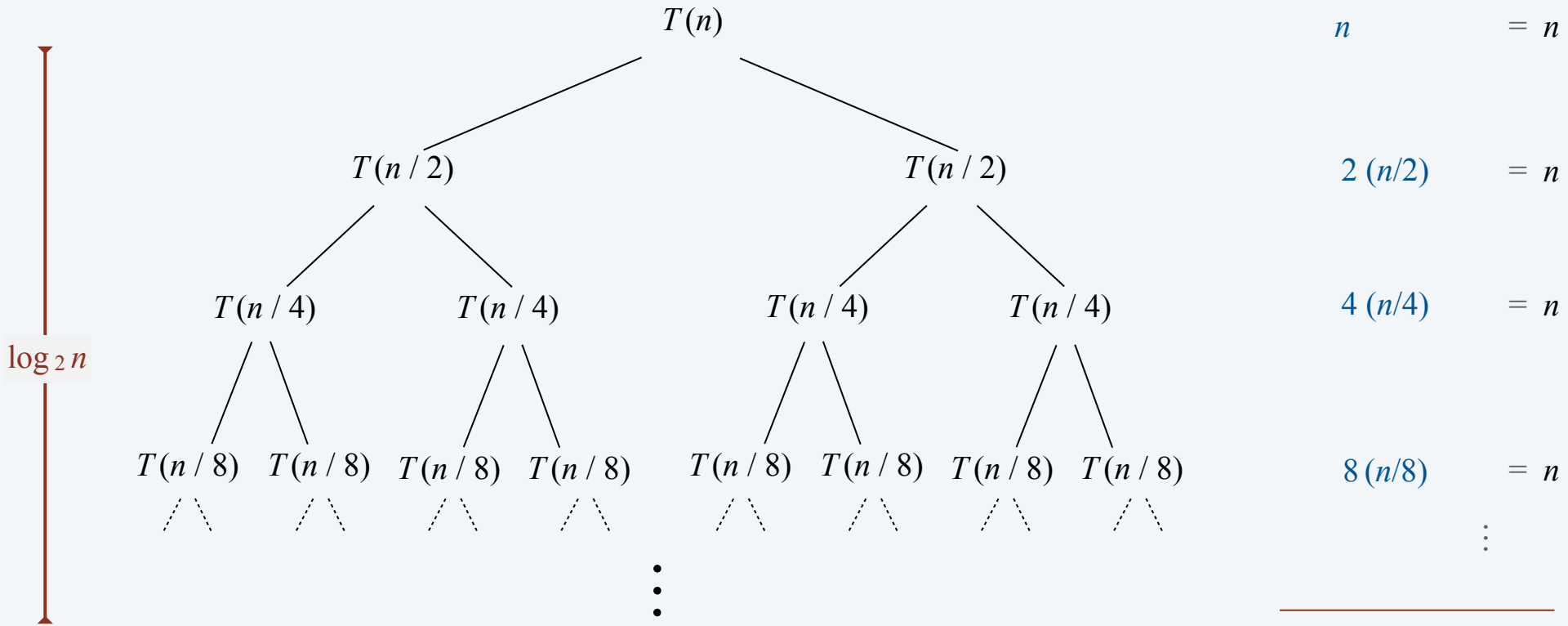
# Divide-and-conquer recurrence: proof by recursion tree

**Proposition.** If $T(n)$ satisfies the following recurrence, then $T(n) = n \log_2 n$.

assuming n is a power of 2

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ 2\,T(n/2) + n & \text{otherwise} \end{cases}$$

Pf 1.



# Proof by induction

**Proposition.** If $T(n)$ satisfies the following recurrence, then $T(n) = n \log_2 n$.

assuming n is a power of 2

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ 2\,T(n/2) + n & \text{otherwise} \end{cases}$$

**Pf 2.** [by induction on $n$]

- Base case: when $n = 1$, $T(1) = 0$.
- Inductive hypothesis: assume $T(n) = n \log_2 n$.
- Goal: show that $T(2n) = 2n \log_2 (2n)$.

$$\begin{aligned} T(2n) &= 2\,T(n) + 2n \\ &= 2\,n \log_2 n + 2n \\ &= 2\,n\,(\log_2 (2n) - 1) + 2n \\ &= 2\,n \log_2 (2n). \quad \blacksquare \end{aligned}$$

# Analysis of mergesort recurrence

Claim. If $T(n)$ satisfies the following recurrence, then $T(n) \leq n \lceil \log_2 n \rceil$.

$$T(n) \leq \begin{cases} 0 & \text{if } n = 1 \\ T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + n & \text{otherwise} \end{cases}$$

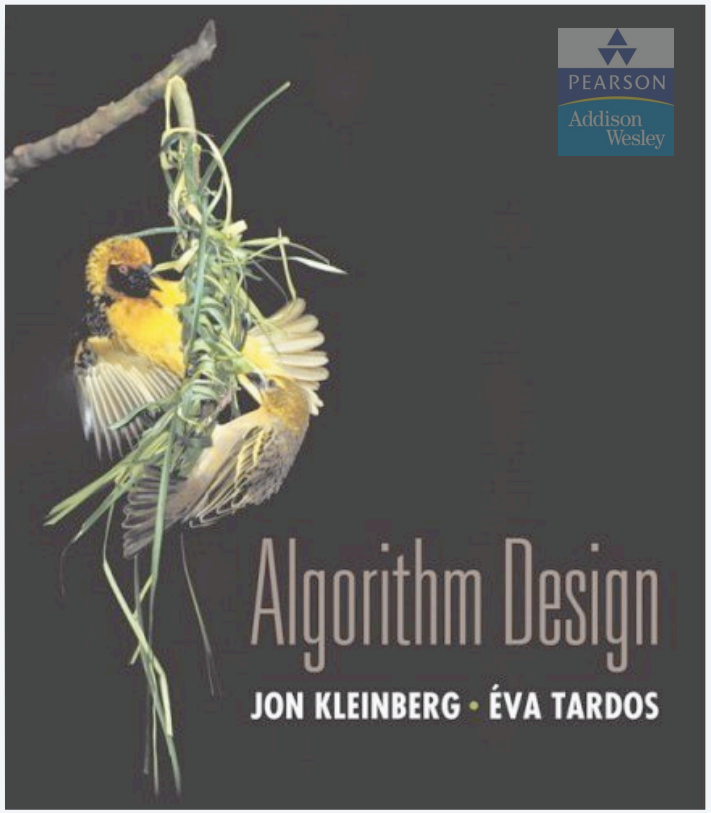
Pf. [by strong induction on $n$]

- Base case: $n = 1$.
- Define $n_1 = \lfloor n/2 \rfloor$ and $n_2 = \lceil n/2 \rceil$.
- Induction step: assume true for $1, 2, \ldots, n-1$.

$$\begin{aligned} T(n) &\leq T(n_1) + T(n_2) + n \\ &\leq n_1 \lceil \log_2 n_1 \rceil + n_2 \lceil \log_2 n_2 \rceil + n \\ &\leq n_1 \lceil \log_2 n_2 \rceil + n_2 \lceil \log_2 n_2 \rceil + n \\ &= n \lceil \log_2 n_2 \rceil + n \\ &\leq n (\lceil \log_2 n \rceil - 1) + n \\ &= n \lceil \log_2 n \rceil. \quad \blacksquare \end{aligned}$$

$\log_2 n_2 \leq \lceil \log_2 n \rceil - 1$

$$\begin{aligned} n_2 &= \lceil n/2 \rceil \\ &\leq \left\lceil 2^{\lceil \log_2 n \rceil} / 2 \right\rceil \\ &= 2^{\lceil \log_2 n \rceil} / 2 \end{aligned}$$




SECTION 5.4

# 5. DIVIDE AND CONQUER

- *mergesort*
- *counting inversions*
- *closest pair of points*
- *randomized quicksort*
- *median and selection*

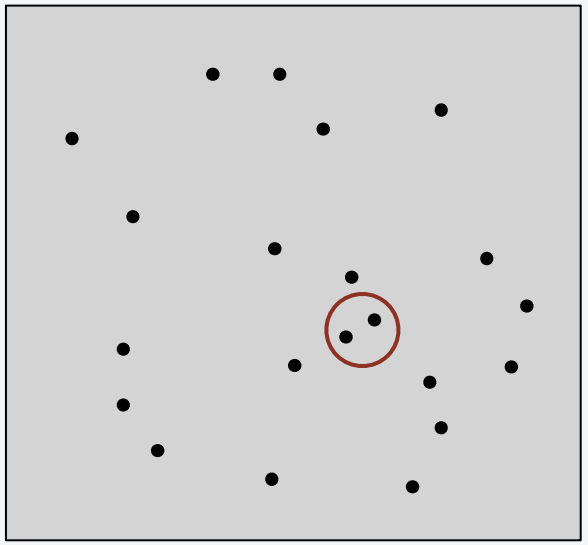
# Closest pair of points

Closest pair problem. Given $n$ points in the plane, find a pair of points with the smallest Euclidean distance between them.

Fundamental geometric primitive.

- Graphics, computer vision, geographic information systems, molecular modeling, air traffic control.
- Special case of nearest neighbor, Euclidean MST, Voronoi.

fast closest pair inspired fast algorithms for these problems



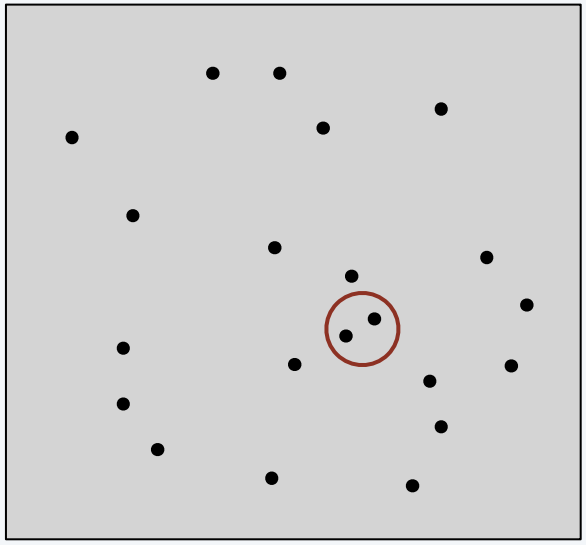
# Closest pair of points

Closest pair problem. Given $n$ points in the plane, find a pair of points with the smallest Euclidean distance between them.

Brute force. Check all pairs with $\Theta(n^2)$ distance calculations.

1d version. Easy $O(n \log n)$ algorithm if points are on a line.

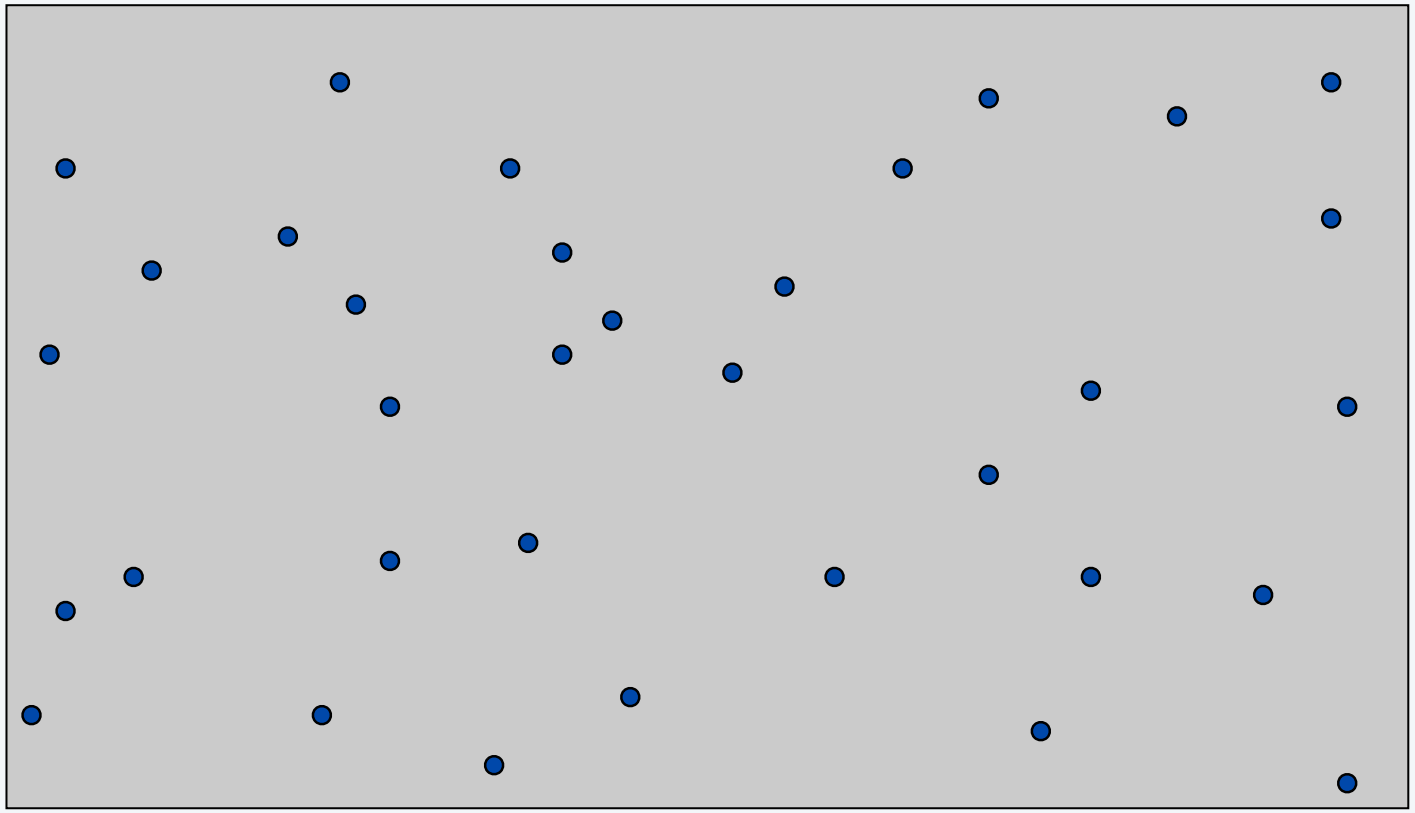
Nondegeneracy assumption. No two points have the same $x$-coordinate.



# Closest pair of points: first attempt

Sorting solution.

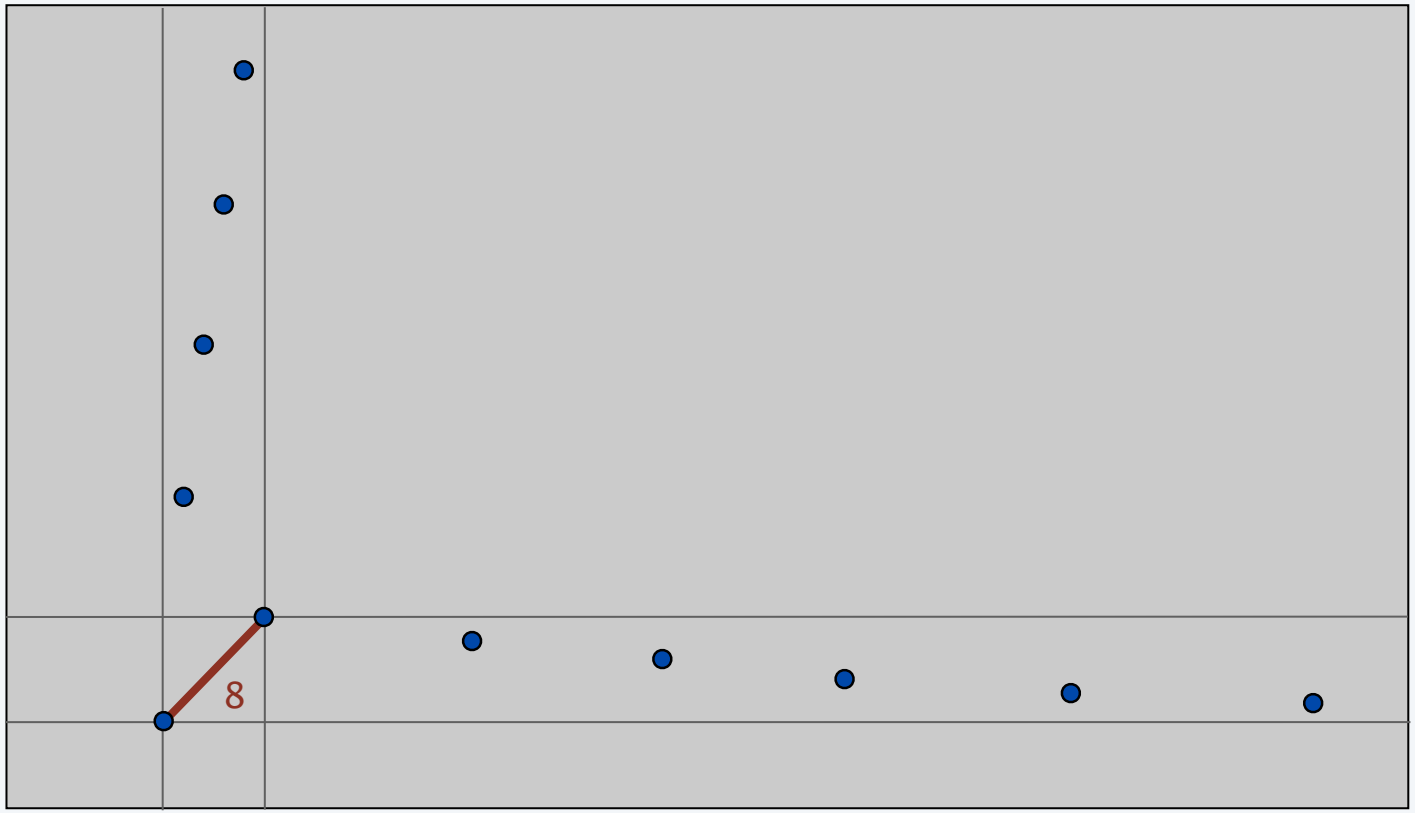
- Sort by $x$-coordinate and consider nearby points.
- Sort by $y$-coordinate and consider nearby points.



# Closest pair of points: first attempt

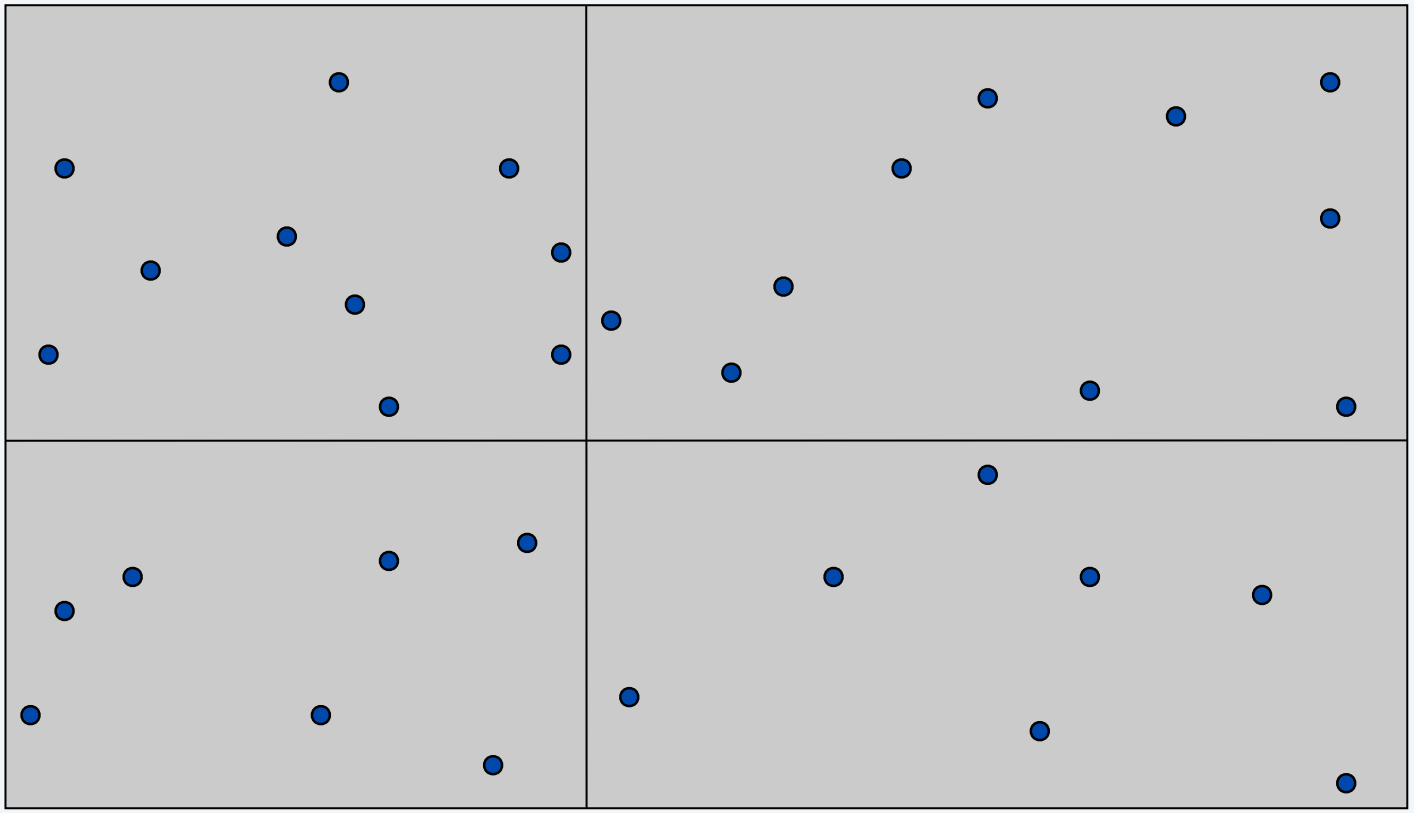
Sorting solution.

- Sort by $x$-coordinate and consider nearby points.
- Sort by $y$-coordinate and consider nearby points.



# Closest pair of points: second attempt

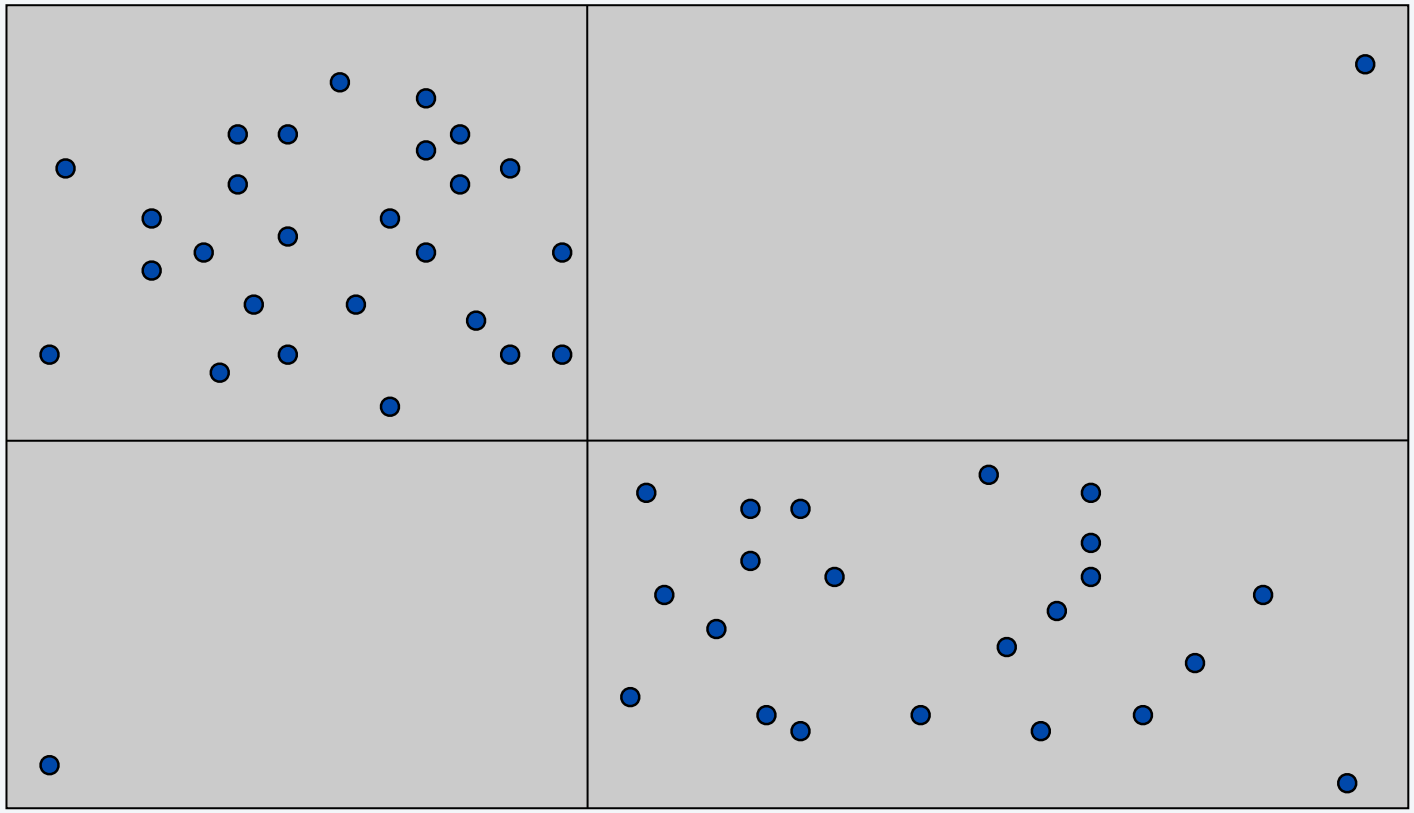
Divide. Subdivide region into 4 quadrants.



# Closest pair of points: second attempt

Divide. Subdivide region into 4 quadrants.

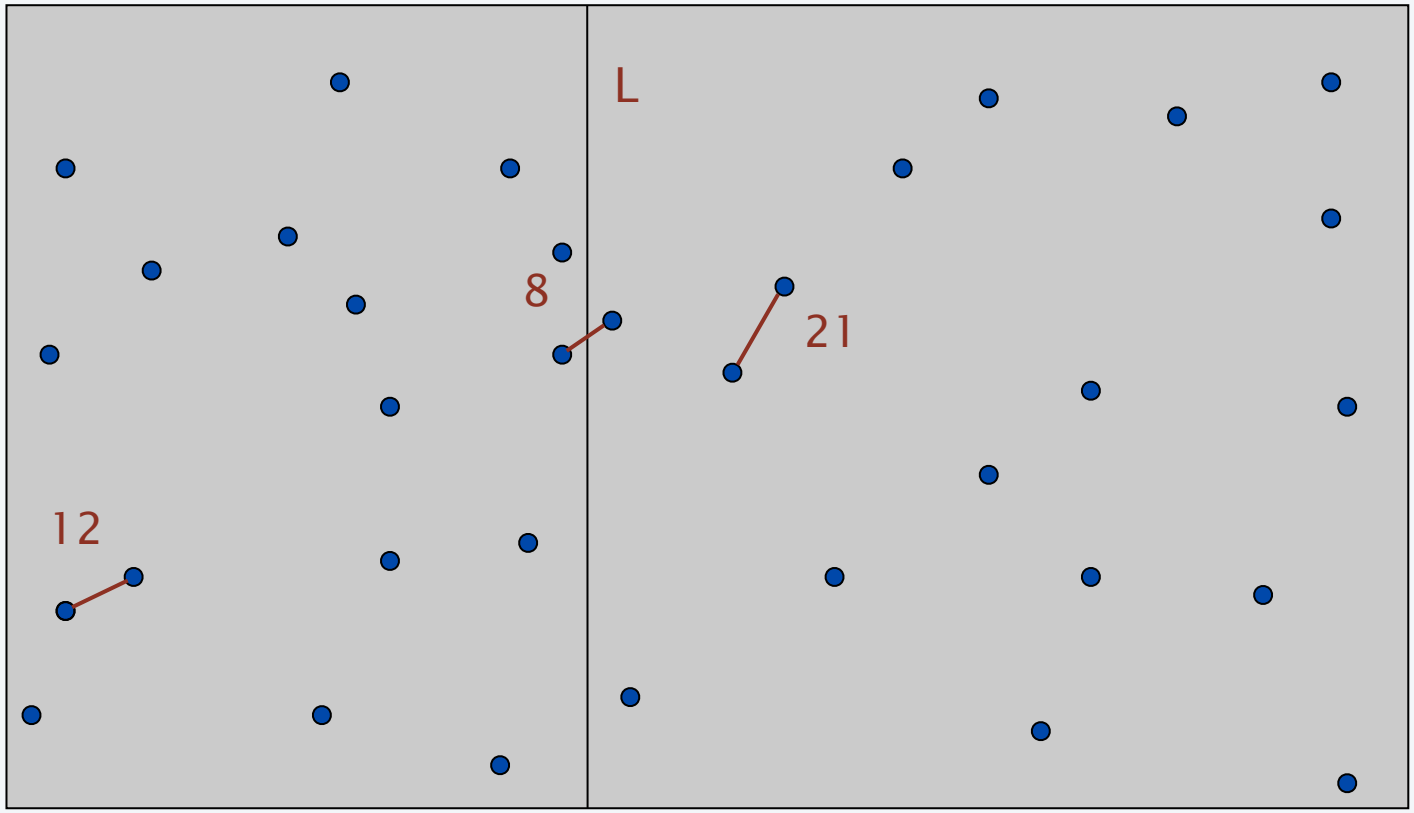
Obstacle. Impossible to ensure $n/4$ points in each piece.



# Closest pair of points: divide-and-conquer algorithm

- Divide: draw vertical line $L$ so that $n/2$ points on each side.
- Conquer: find closest pair in each side recursively.
- Combine: find closest pair with one point in each side.
- Return best of 3 solutions.

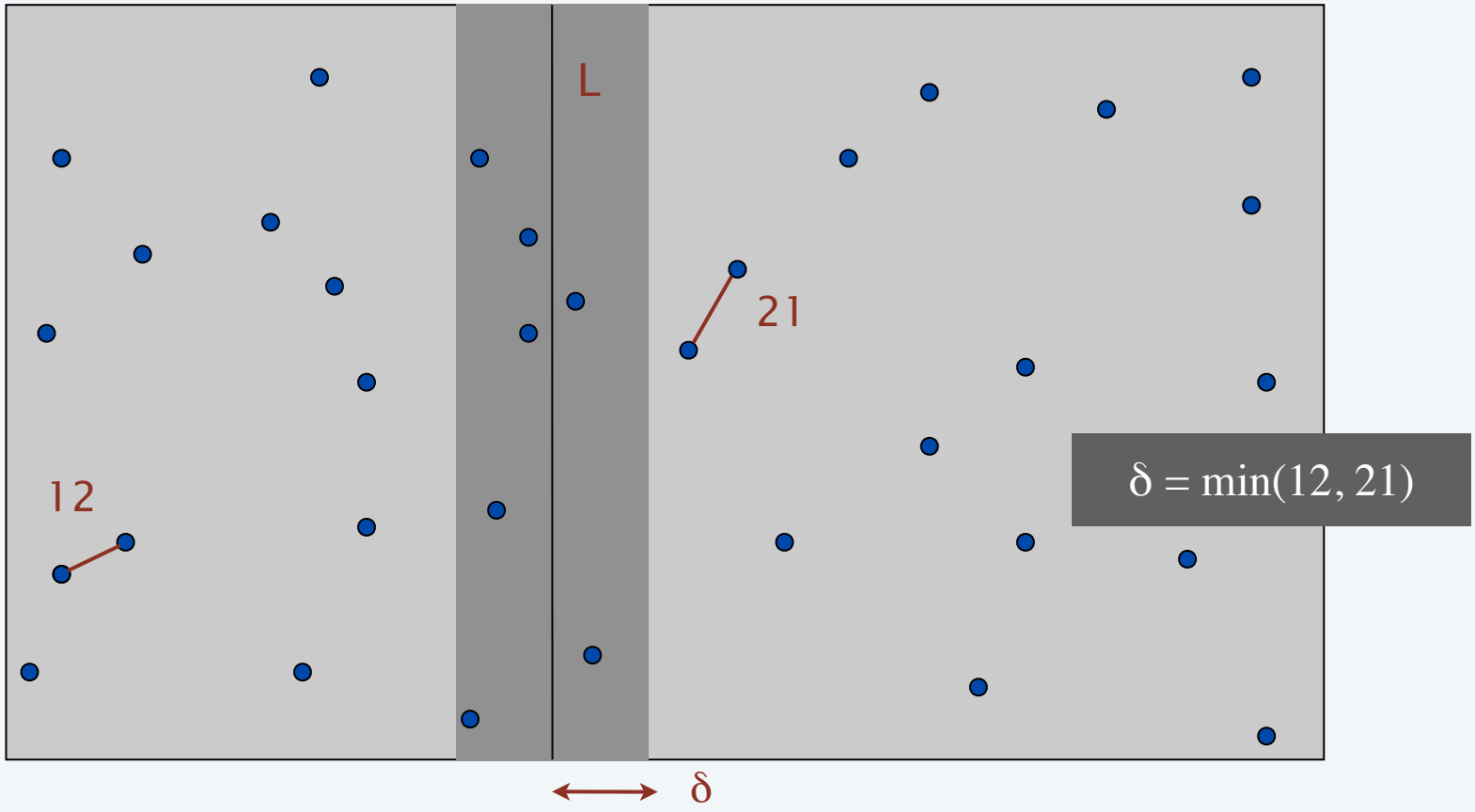
seems like $\Theta(N^2)$



# How to find closest pair with one point in each side?

Find closest pair with one point in each side, assuming that distance < δ.

- Observation: only need to consider points within δ of line $L$.

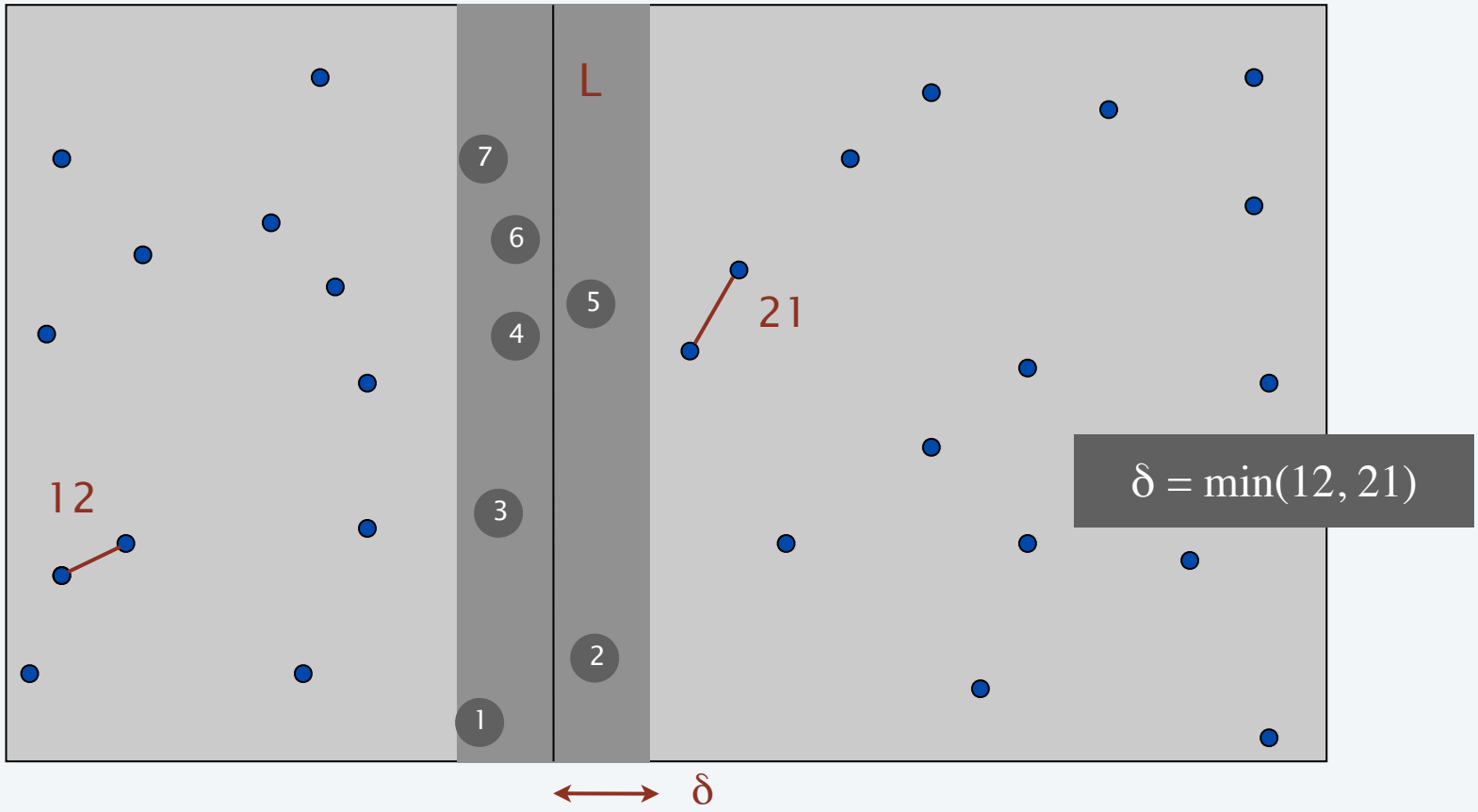


# How to find closest pair with one point in each side?

Find closest pair with one point in each side, assuming that distance < $\delta$.

- Observation: only need to consider points within $\delta$ of line $L$.
- Sort points in $2\delta$-strip by their $y$-coordinate.
- Only check distances of those within $11$ positions in sorted list!

why 11?



# How to find closest pair with one point in each side?

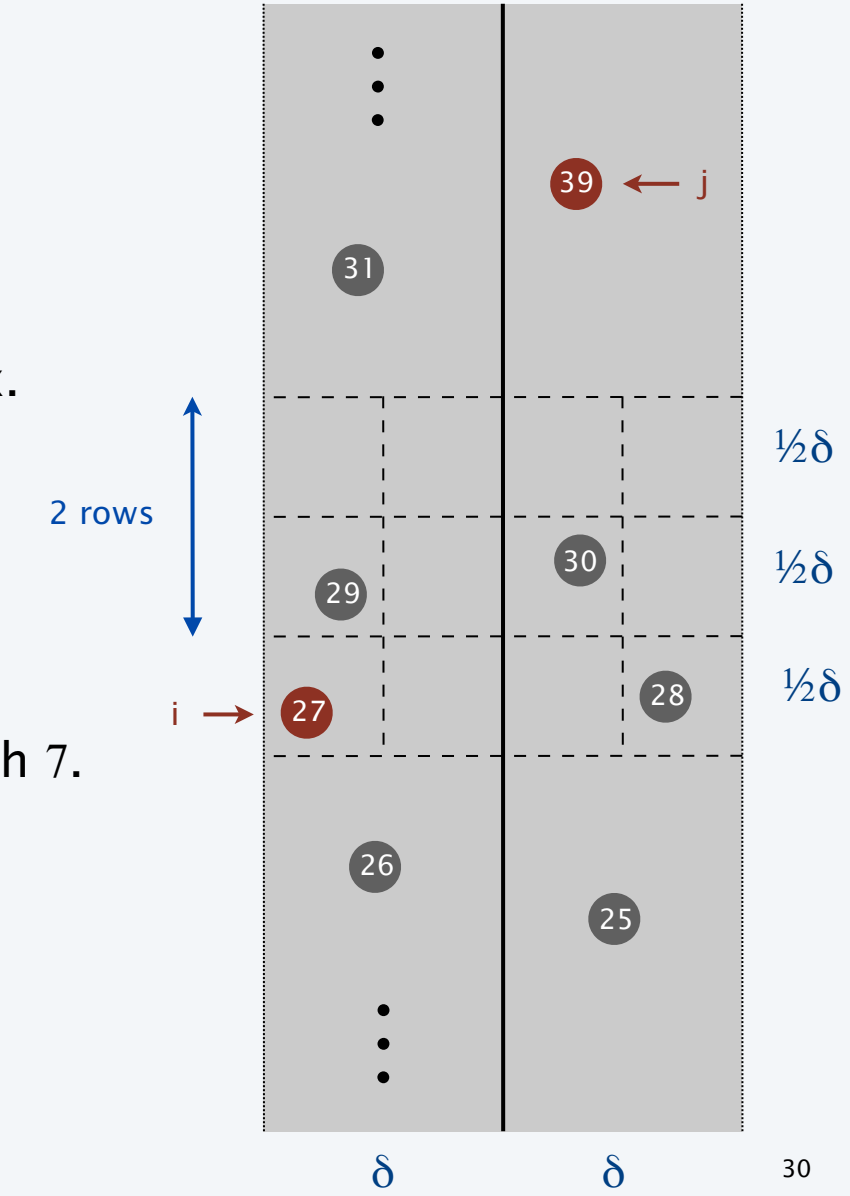
Def. Let $s_i$ be the point in the $2\delta$-strip, with the $i^{th}$ smallest $y$-coordinate.

Claim. If $|i - j| \geq 12$, then the distance between $s_i$ and $s_j$ is at least $\delta$.

Pf.

- No two points lie in same $\frac{1}{2}\delta$-by-$\frac{1}{2}\delta$ box.
- Two points at least 2 rows apart have distance $\geq 2(\frac{1}{2}\delta)$. ▪

Fact. Claim remains true if we replace 12 with 7.



# Closest pair of points: divide-and-conquer algorithm

CLOSEST-PAIR $(p_1, p_2, \ldots, p_n)$

Compute separation line $L$ such that half the points are on each side of the line. ← $O(n \log n)$

$\delta_1 \leftarrow$ CLOSEST-PAIR (points in left half).

$\delta_2 \leftarrow$ CLOSEST-PAIR (points in right half). ← $2\,T(n / 2)$

$\delta \leftarrow \min \{ \delta_1 , \delta_2 \}$.

Delete all points further than $\delta$ from line $L$. ← $O(n)$

Sort remaining points by $y$-coordinate. ← $O(n \log n)$

Scan points in $y$-order and compare distance between each point and next 11 neighbors. If any of these distances is less than $\delta$, update $\delta$. ← $O(n)$

RETURN $\delta$.

# Closest pair of points: analysis

Theorem. The divide-and-conquer algorithm for finding the closest pair of points in the plane can be implemented in $O(n \log^2 n)$ time.

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + O(n \log n) & \text{otherwise} \end{cases}$$

$(x_1 - x_2)^2 + (y_1 - y_2)^2$

Lower bound. In quadratic decision tree model, any algorithm for closest pair (even in 1D) requires $\Omega(n \log n)$ quadratic tests.

# Improved closest pair algorithm

Q. How to improve to $O(n \log n)$ ?

A. Yes. Don't sort points in strip from scratch each time.

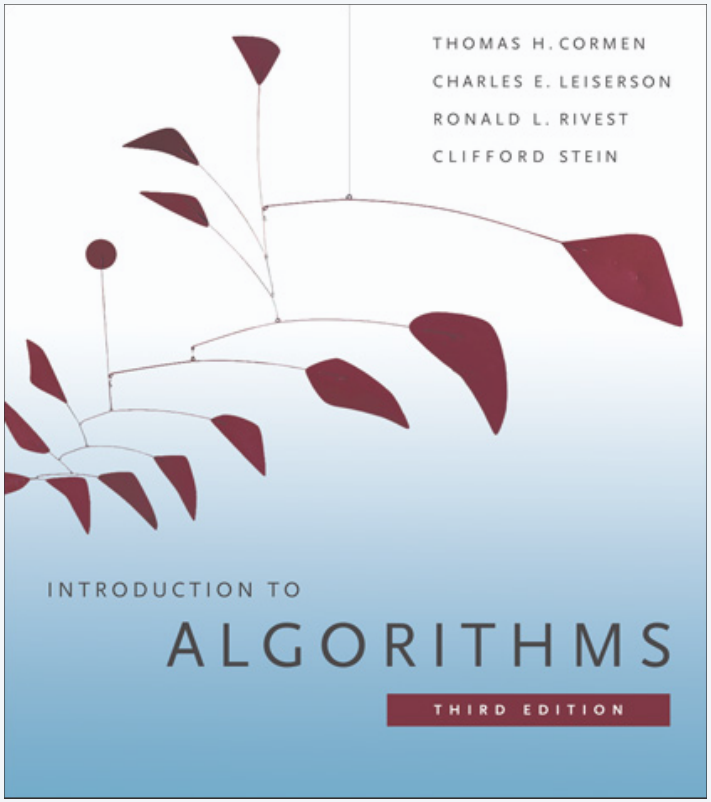
- Each recursive returns two lists: all points sorted by $x$-coordinate, and all points sorted by $y$-coordinate.
- Sort by merging two pre-sorted lists.

Theorem. [Shamos 1975] The divide-and-conquer algorithm for finding the closest pair of points in the plane can be implemented in $O(n \log n)$ time.

Pf.
$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \Theta(n) & \text{otherwise} \end{cases}$$

Note. See SECTION 13.7 for a randomized $O(n)$ time algorithm.

not subject to lower bound
since it uses the floor function




CHAPTER 7

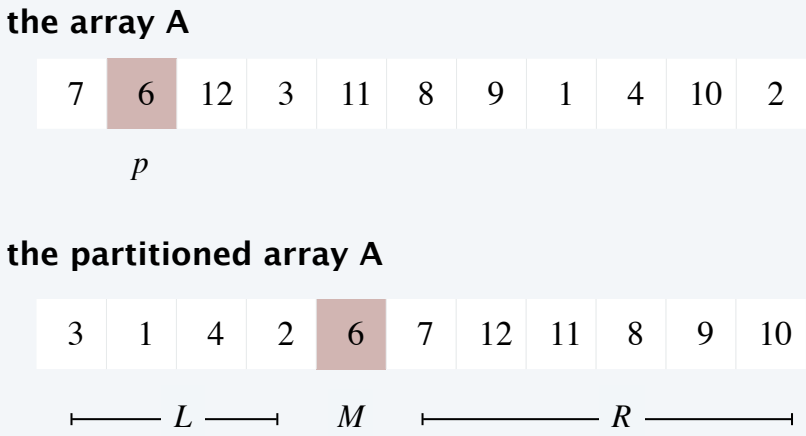
# 5. DIVIDE AND CONQUER

- *mergesort*
- *counting inversions*
- *closest pair of points*
- *randomized quicksort*
- *median and selection*

# Randomized quicksort

3-way partition array so that:

- Pivot element $p$ is in place.
- Smaller elements in left subarray $L$.
- Equal elements in middle subarray $M$.
- Larger elements in right subarray $R$.




Recur in both left and right subarrays.

RANDOMIZED-QUICKSORT $(A)$

IF list $A$ has zero or one element

  RETURN.

Pick pivot $p \in A$ uniformly at random.

$(L, M, R) \leftarrow$ PARTITION-3-WAY $(A, a_i)$.

RANDOMIZED-QUICKSORT$(L)$.

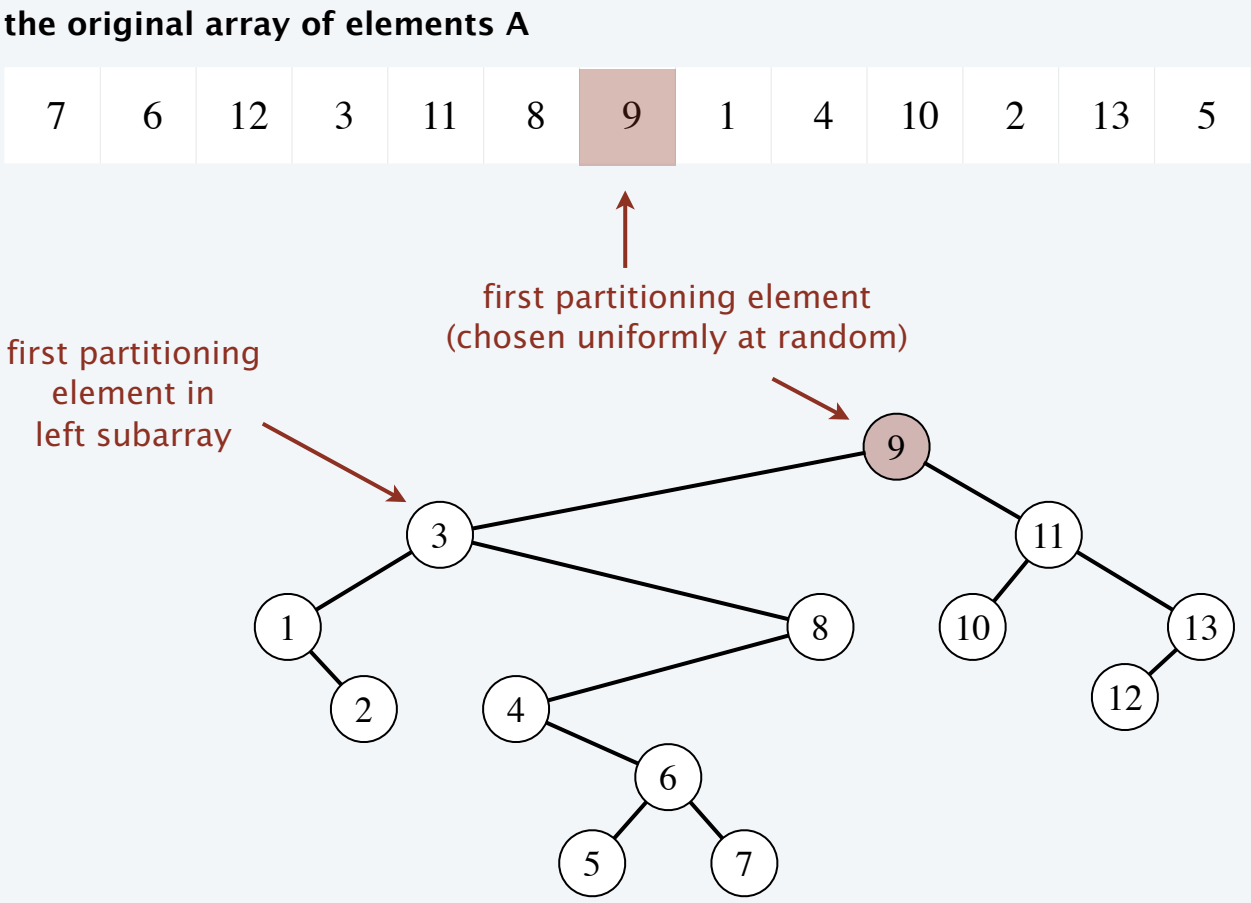
RANDOMIZED-QUICKSORT$(R)$.

3-way partitioning can be done in-place (using n–1 compares)

# Analysis of randomized quicksort

Proposition. The expected number of compares to quicksort an array of $n$ distinct elements is $O(n \log n)$.

Pf. Consider BST representation of partitioning elements.

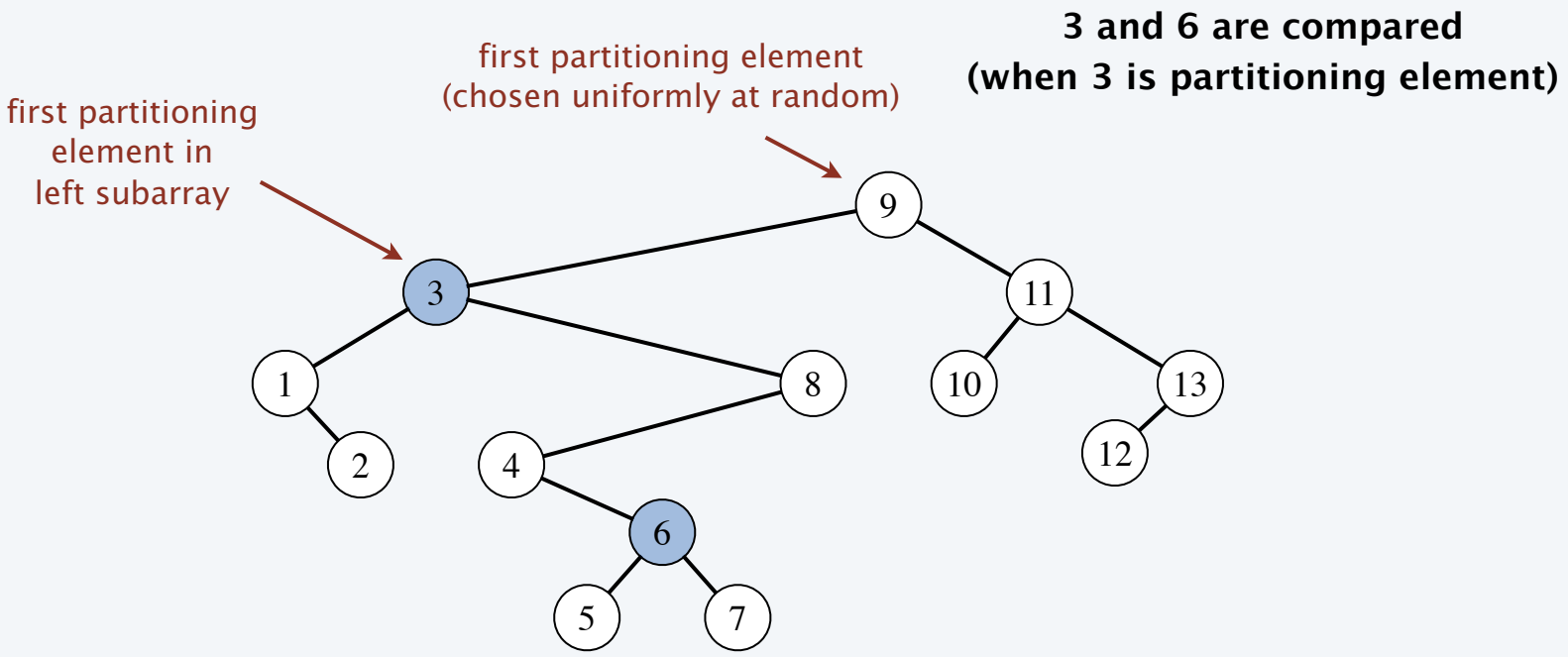


# Analysis of randomized quicksort

Proposition. The expected number of compares to quicksort an array of $n$ distinct elements is $O(n \log n)$.

Pf. Consider BST representation of partitioning elements.

- An element is compared with only its ancestors and descendants.

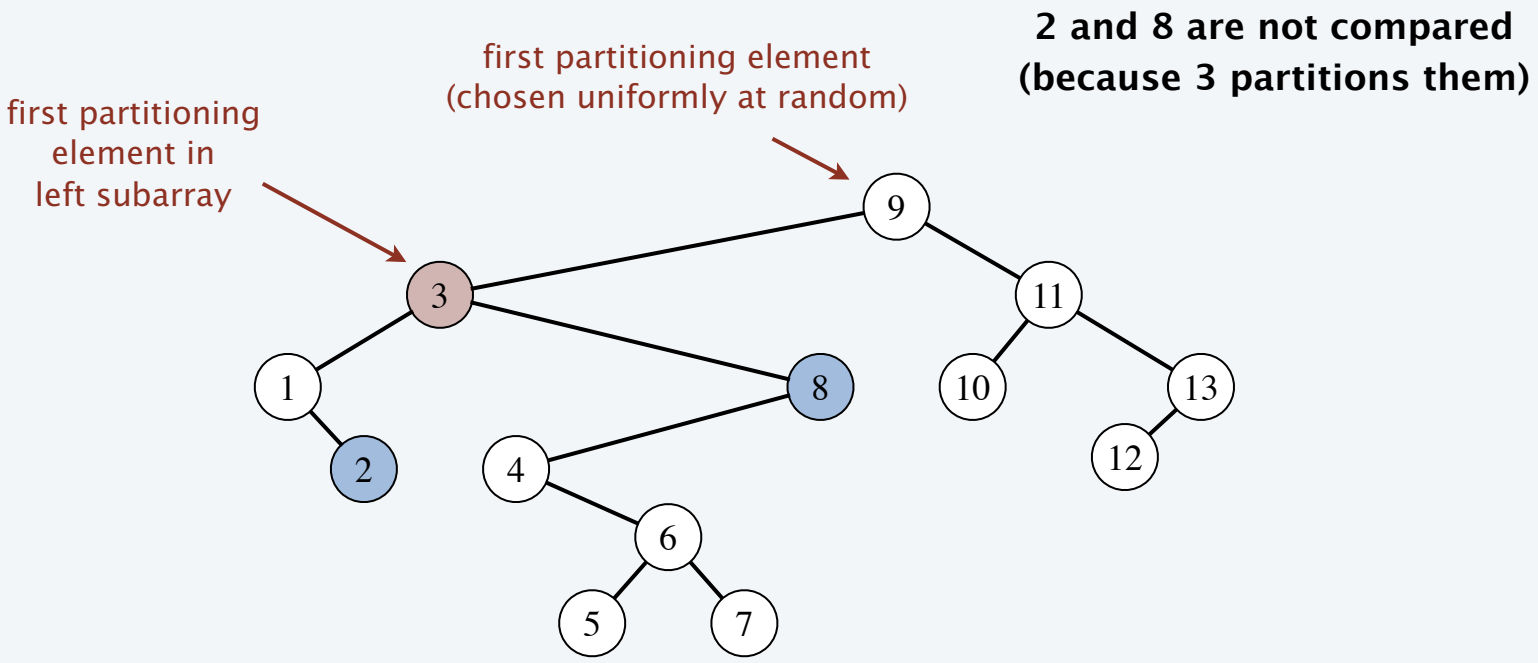


# Analysis of randomized quicksort

Proposition. The expected number of compares to quicksort an array of $n$ distinct elements is $O(n \log n)$.

Pf. Consider BST representation of partitioning elements.

- An element is compared with only its ancestors and descendants.

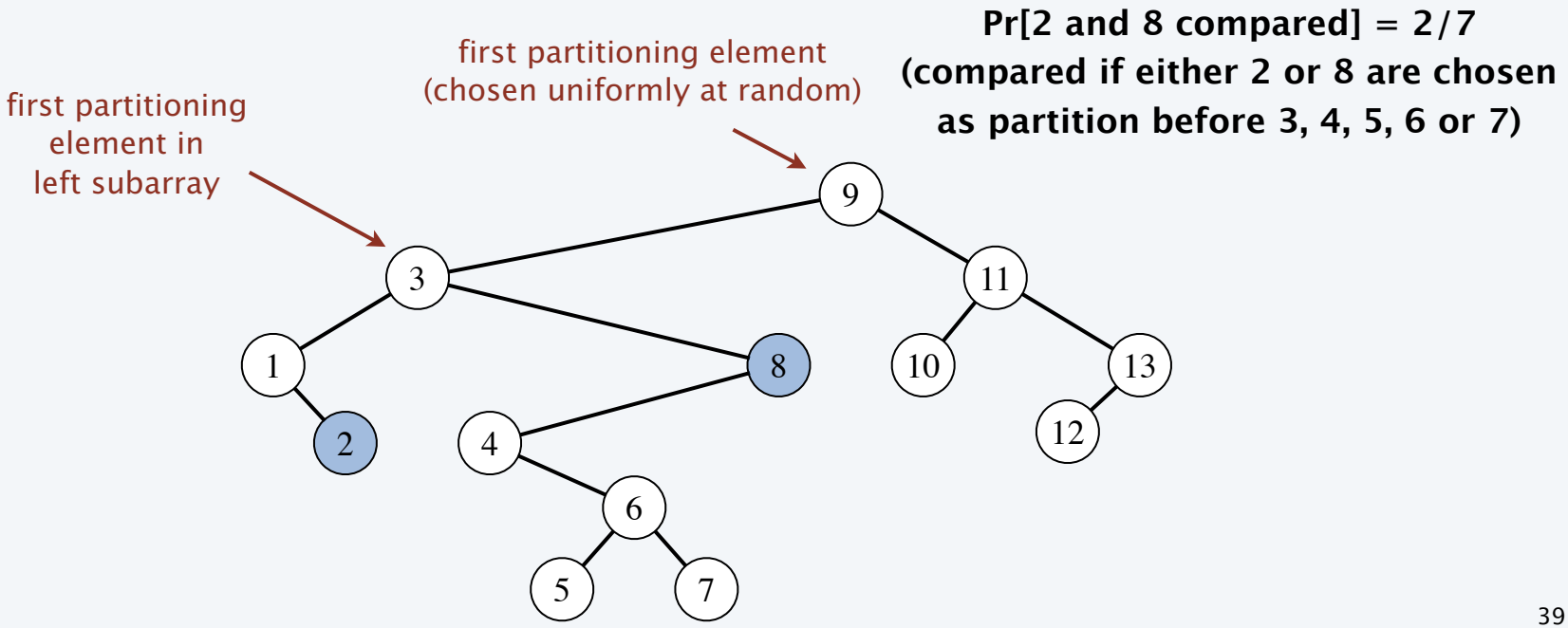


# Analysis of randomized quicksort

Proposition. The expected number of compares to quicksort an array of $n$ distinct elements is $O(n \log n)$.

Pf. Consider BST representation of partitioning elements.

- An element is compared with only its ancestors and descendants.
- **Pr** [ $a_i$ and $a_j$ are compared ] $= 2 \ / \ |j - i + 1|$.



# Analysis of randomized quicksort

Proposition. The expected number of compares to quicksort an array of $n$ distinct elements is $O(n \log n)$.

Pf. Consider BST representation of partitioning elements.

- An element is compared with only its ancestors and descendants.
- **Pr** [ $a_i$ and $a_j$ are compared ] $= 2 \ / \ |j - i + 1|$.
- Expected number of compares $=$

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

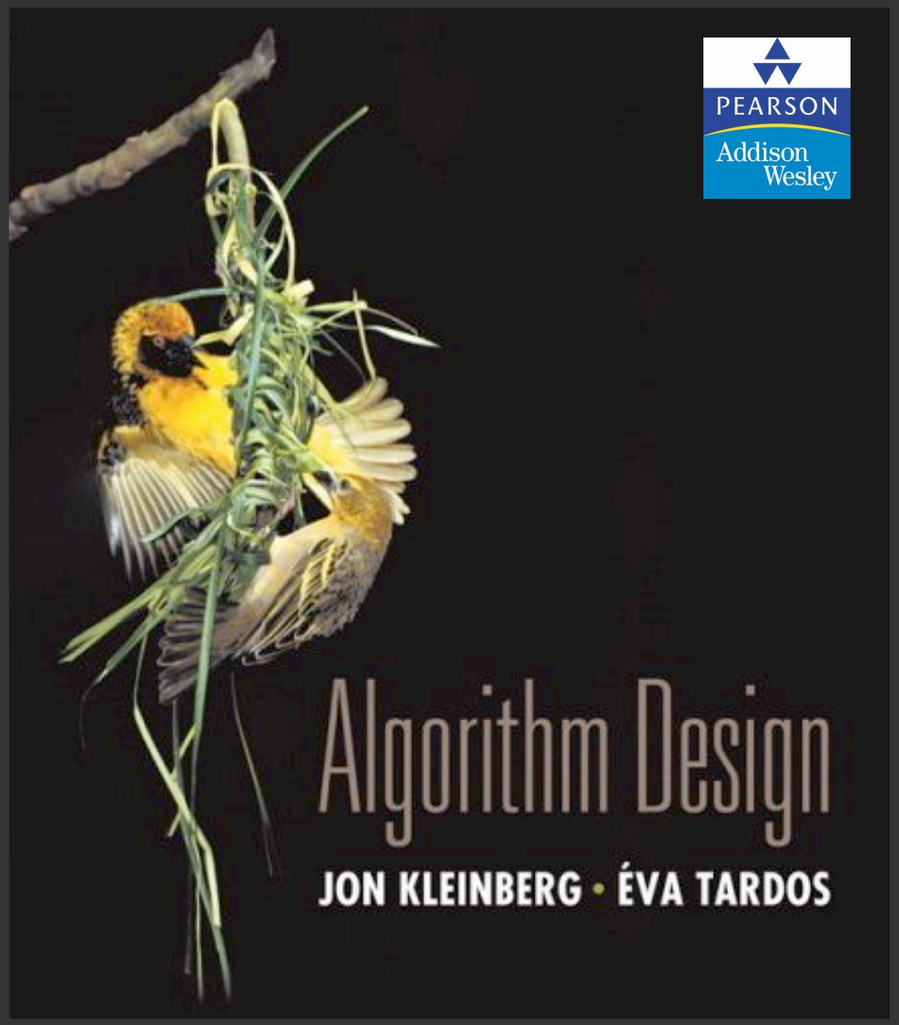
all pairs i and j

$$\leq 2N \sum_{j=1}^{N} \frac{1}{j}$$

$$\sim 2N \int_{x=1}^{N} \frac{1}{x} \, dx$$

$$= 2N \ln N$$

Remark. Number of compares only decreases if equal elements.

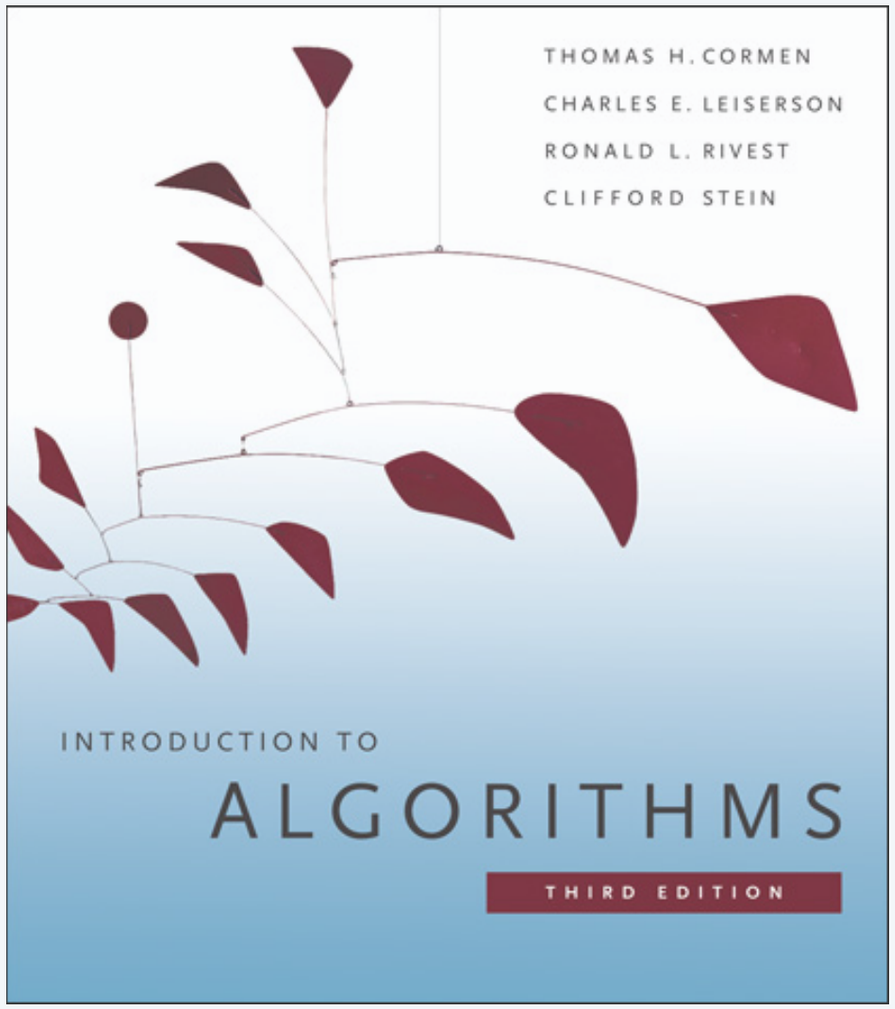



# Divide and Conquer II

- *master theorem*
- *integer multiplication*
- *matrix multiplication*
- *convolution and FFT*



Last updated on Jul 20, 2015, 3:31 PM




# Divide and Conquer II

- *master theorem*
- *integer multiplication*
- *matrix multiplication*
- *convolution and FFT*

Sections 4.3–4.6

# Master method

Goal. Recipe for solving common divide-and-conquer recurrences:

$$T(n) = a\,T\left(\frac{n}{b}\right) \;+\; f(n)$$

Terms.

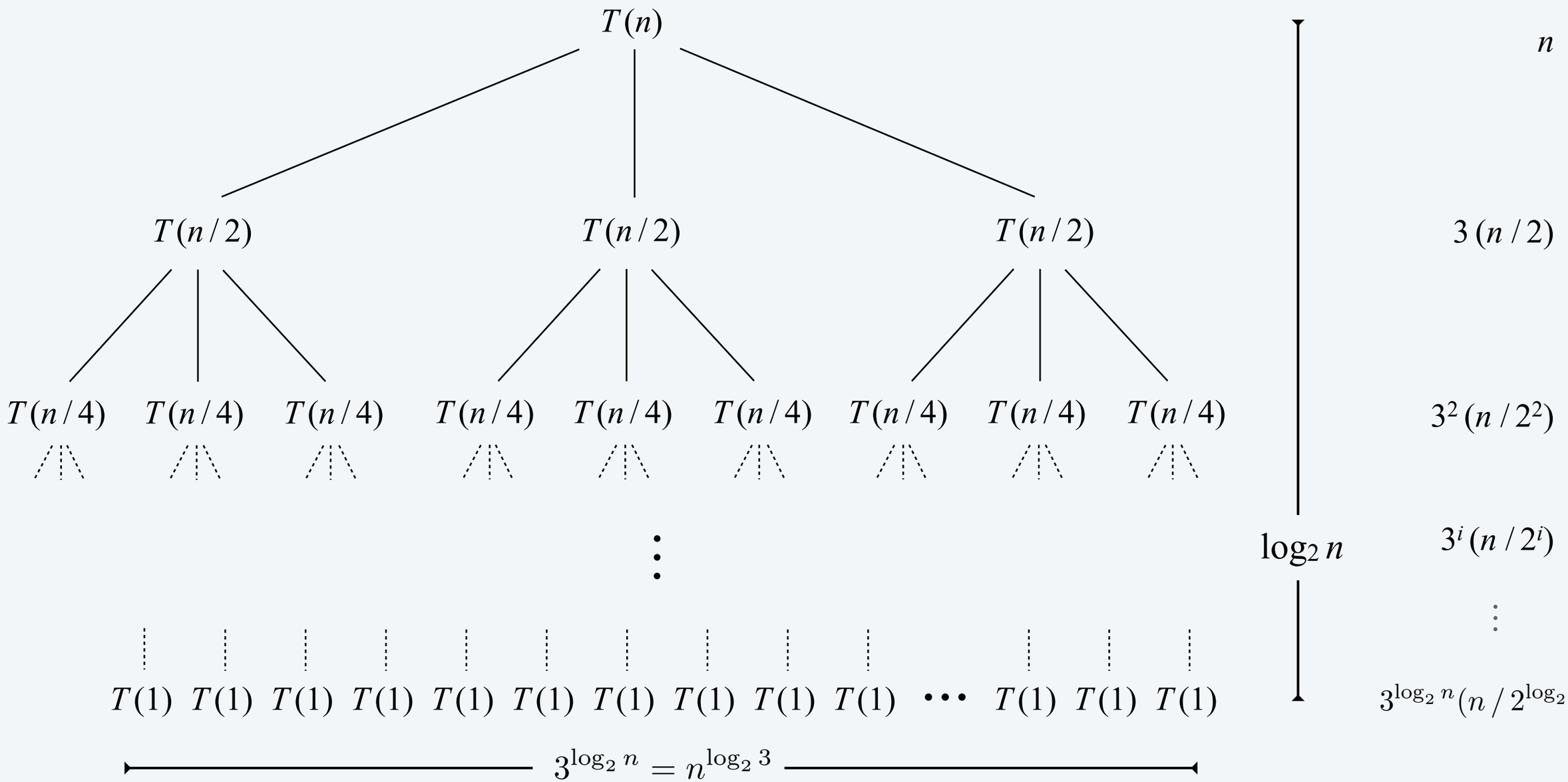
- $a \geq 1$ is the number of subproblems.
- $b > 0$ is the factor by which the subproblem size decreases.
- $f(n)$ = work to divide/merge subproblems.

Recursion tree.

- $k = \log_b n$ levels.
- $a^i$ = number of subproblems at level $i$.
- $n / b^i$ = size of subproblem at level $i$.

# Case 1: total cost dominated by cost of leaves

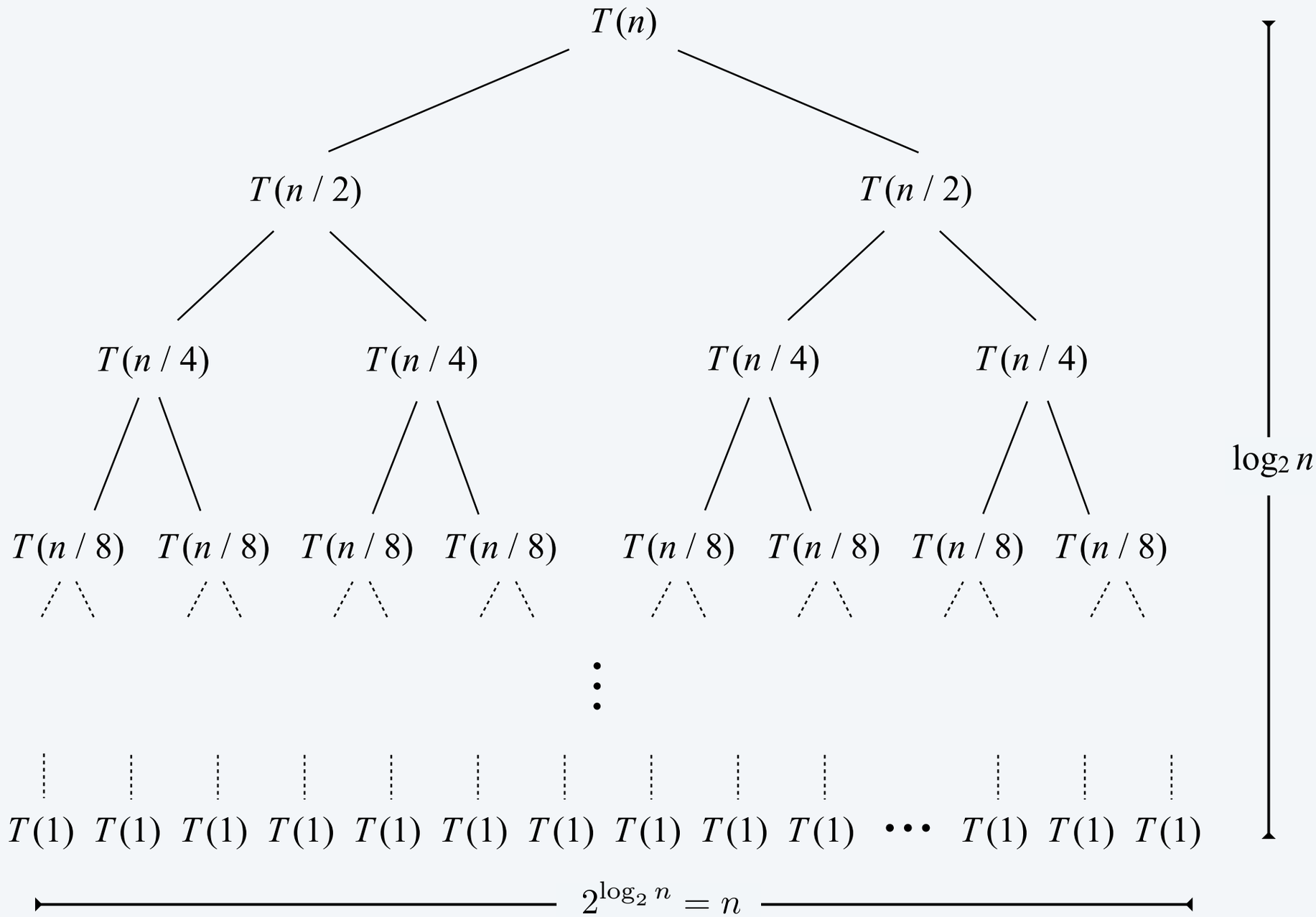
Ex 1. If $T(n)$ satisfies $T(n) = 3\ T(n/2) + n$, with $T(1) = 1$, then $T(n) = \Theta(n^{\lg 3})$.




$$r = 3/2 > 1 \qquad T(n) = (1 + r + r^2 + r^3 + \ldots + r^{\log_2 n})\ n = \frac{r^{1+\log_2 n} - 1}{r - 1}\ n = 3n^{\log_2 3} - 2n$$

# Case 2: total cost evenly distributed among levels

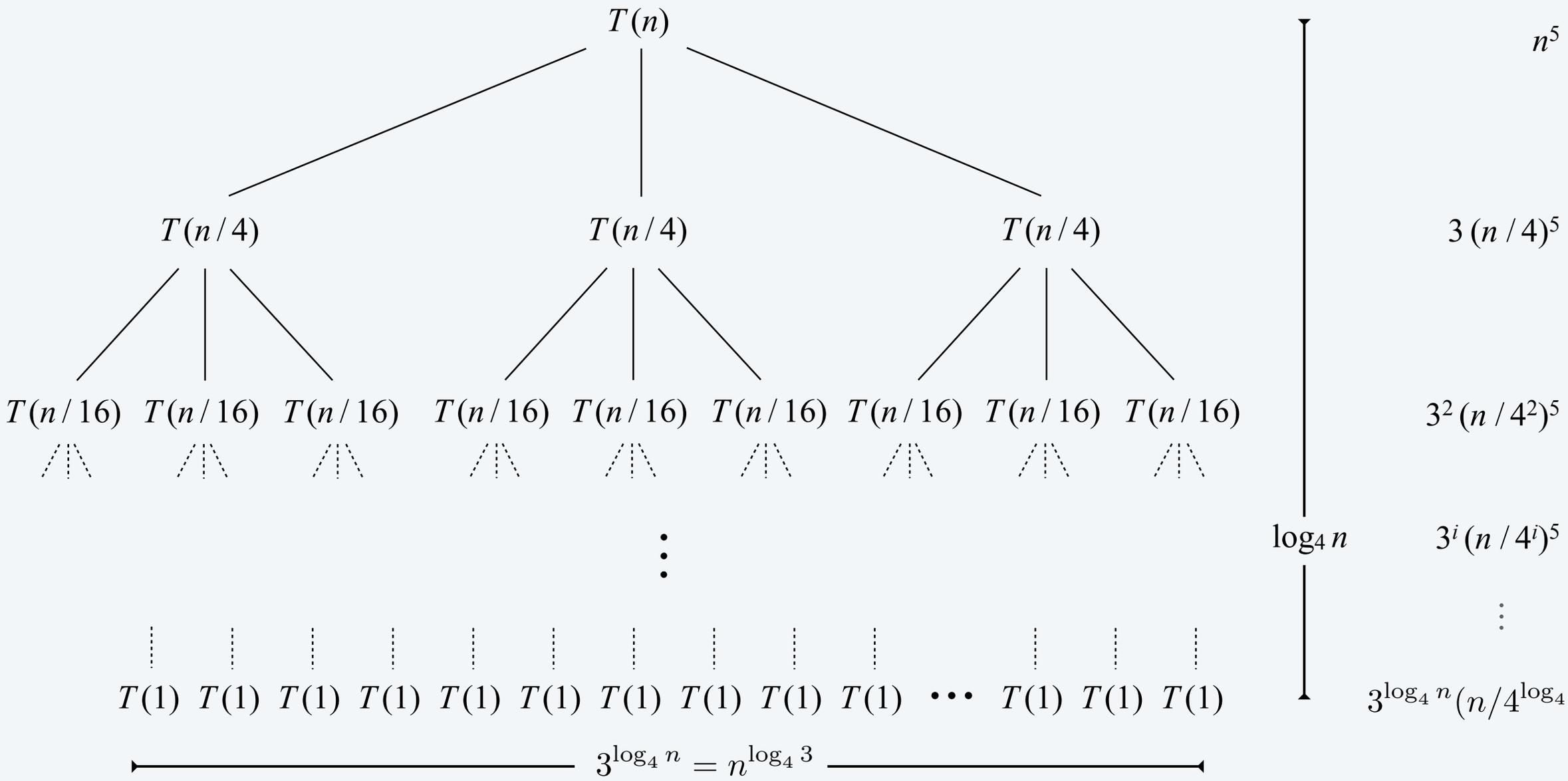
Ex 2. If $T(n)$ satisfies $T(n) = 2\,T(n/2) + n$, with $T(1) = 1$, then $T(n) = \Theta(n \log n)$.




$r = 1$ $\qquad T(n) = (1 + r + r^2 + r^3 + \ldots + r^{\log_2 n})\, n \;=\; n\,(\log_2 n + 1)$

# Case 3: total cost dominated by cost of root

Ex 3. If $T(n)$ satisfies $T(n) = 3\ T(n/4) + n^5$, with $T(1) = 1$, then $T(n) = \Theta(n^5)$.




$$r = 3/4^5 < 1 \qquad n^5 \le T(n) \le (1 + r + r^2 + r^3 + \dots)\, n^5 \le \frac{1}{1-r}\, n^5$$

# Master theorem

Master theorem. Suppose that $T(n)$ is a function on the nonnegative integers that satisfies the recurrence

$$T(n) = a\,T\left(\frac{n}{b}\right) \;+\; f(n)$$

where $n/b$ means either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Let $k = \log_b a$. Then,

Case 1. If $f(n) = O(n^{k-\varepsilon})$ for some constant $\varepsilon > 0$, then $T(n) = \Theta(n^k)$.

Ex. $T(n) = 3\,T(n/2) + n$.

- $a = 3,\ b = 2,\ f(n) = n,\ \ k = \log_2 3$.
- $T(n) = \Theta(n^{\lg 3})$.

# Master theorem

Master theorem. Suppose that $T(n)$ is a function on the nonnegative integers that satisfies the recurrence

$$T(n) = a\,T\left(\frac{n}{b}\right) \;+\; f(n)$$

where $n/b$ means either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Let $k = \log_b a$. Then,

Case 2. If $f(n) = \Theta(n^k \log^p n)$, then $T(n) = \Theta(n^k \log^{p+1} n)$.

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

- $a = 2,\ b = 2,\ f(n) = 17\,n,\ \ k = \log_2 2 = 1,\ p = 1.$
- $T(n) = \Theta(n \log^2 n)$.

# Master theorem

Master theorem. Suppose that $T(n)$ is a function on the nonnegative integers that satisfies the recurrence

$$T(n) = a\,T\left(\frac{n}{b}\right) + f(n)$$

where $n/b$ means either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Let $k = \log_b a$. Then,

regularity condition holds if f(n) = $\Theta(n^{k+\varepsilon})$

Case 3. If $f(n) = \Omega(n^{k+\varepsilon})$ for some constant $\varepsilon > 0$ and if $a\,f(n/b) \le c\,f(n)$ for some constant $c < 1$ and all sufficiently large $n$, then $T(n) = \Theta(f(n))$.

Ex. $T(n) = 3\,T(n/4) + n^5$.

- $a = 3,\ b = 4,\ f(n) = n^5,\ k = \log_4 3$.
- $T(n) = \Theta(n^5)$.

# Master theorem

Master theorem. Suppose that $T(n)$ is a function on the nonnegative integers that satisfies the recurrence

$$T(n) = a\,T\left(\frac{n}{b}\right) \;+\; f(n)$$

where $n/b$ means either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Let $k = \log_b a$. Then,

Case 1. If $f(n) = O(n^{k-\varepsilon})$ for some constant $\varepsilon > 0$, then $T(n) = \Theta(n^k)$.

Case 2. If $f(n) = \Theta(n^k \log^p n)$, then $T(n) = \Theta(n^k \log^{p+1} n)$.

Case 3. If $f(n) = \Omega(n^{k+\varepsilon})$ for some constant $\varepsilon > 0$ and if $a\,f(n/b) \le c\,f(n)$ for some constant $c < 1$ and all sufficiently large $n$, then $T(n) = \Theta(f(n))$.

Pf sketch.

- Use recursion tree to sum up terms (assuming $n$ is an exact power of $b$).
- Three cases for geometric series.
- Deal with floors and ceilings.

# Akra-Bazzi theorem

Desiderata. Generalizes master theorem to divide-and-conquer algorithms where subproblems have substantially different sizes.

Theorem. [Akra-Bazzi] Given constants $a_i > 0$ and $0 < b_i \le 1$, functions $h_i(n) = O(n / \log^2 n)$ and $g(n) = O(n^c)$, if the function $T(n)$ satisfies the recurrence:

$$T(n) = \sum_{i=1}^{k} a_i T(b_i n + h_i(n)) + g(n)$$

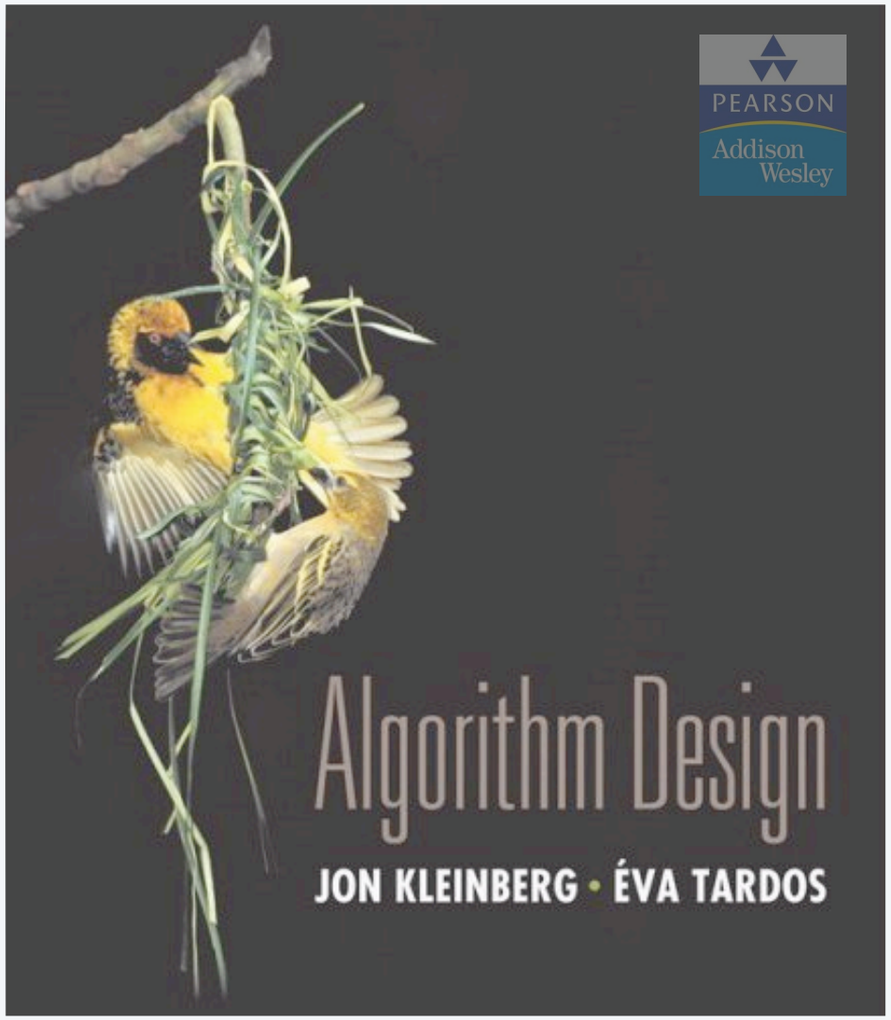
$a_i$ subproblems of size $b_i$ n

small perturbation to handle floors and ceilings

Then $T(n) = \Theta\left(n^p \left(1 + \int_1^n \frac{g(u)}{u^{p+1}} du\right)\right)$ where $p$ satisfies $\sum_{i=1}^{k} a_i b_i^p = 1$ .

Ex. $T(n) = 7/4\ T(\lfloor n/2 \rfloor) + T(\lceil 3/4\ n \rceil) + n^2$.

- $a_1 = 7/4,\ b_1 = 1/2,\ a_2 = 1,\ b_2 = 3/4 \Rightarrow p = 2$.
- $h_1(n) = \lfloor 1/2\ n \rfloor - 1/2\ n,\ h_2(n) = \lceil 3/4\ n \rceil - 3/4\ n$.
- $g(n) = n^2 \Rightarrow T(n) = \Theta(n^2 \log n)$.




# DIVIDE AND CONQUER II

- *master theorem*
- *integer multiplication*
- *matrix multiplication*
- *convolution and FFT*

SECTION 5.5

# Integer addition

Addition. Given two $n$-bit integers $a$ and $b$, compute $a + b$.

Subtraction. Given two $n$-bit integers $a$ and $b$, compute $a - b$.

Grade-school algorithm. $\Theta(n)$ bit operations.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 |
| | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| + | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 |

Remark. Grade-school addition and subtraction algorithms are asymptotically optimal.

# Integer multiplication

Multiplication. Given two $n$-bit integers $a$ and $b$, compute $a \times b$.

Grade-school algorithm. $\Theta(n^2)$ bit operations.

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | | | | |
| | | | × | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | | | | |
| | | | | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | | | | |
| | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | | | | |
| | | | | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | | | | |
| | | | | | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | | | |
| | | | | | | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | | |
| | | | | | | | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | |
| | | | | | | | | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| | | | | | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

Conjecture. [Kolmogorov 1952] Grade-school algorithm is optimal.

Theorem. [Karatsuba 1960] Conjecture is wrong.

# Divide-and-conquer multiplication

To multiply two $n$-bit integers $x$ and $y$:

- Divide $x$ and $y$ into low- and high-order bits.
- Multiply four $\frac{1}{2}n$-bit integers, recursively.
- Add and shift to obtain result.

$$m = \lceil n / 2 \rceil$$

$$a = \lfloor x / 2^m \rfloor \quad b = x \bmod 2^m$$

$$c = \lfloor y / 2^m \rfloor \quad d = y \bmod 2^m$$

← use bit shifting to compute 4 terms

$$(2^m a + b)(2^m c + d) = \underset{1}{2^{2m} ac} + \underset{2}{2^m (bc} + \underset{3}{ad)} + \underset{4}{bd}$$

Ex. $x = \underbrace{1\,0\,0\,0}_{a}\,\underbrace{1\,1\,0\,1}_{b} \qquad y = \underbrace{1\,1\,1\,0}_{c}\,\underbrace{0\,0\,0\,1}_{d}$

# Divide-and-conquer multiplication

MULTIPLY$(x, y, n)$

IF $(n = 1)$

  RETURN $x \times y$.

ELSE

  $m \leftarrow \lceil n / 2 \rceil$.

  $a \leftarrow \lfloor x / 2^m \rfloor$;  $b \leftarrow x \bmod 2^m$.

  $c \leftarrow \lfloor y / 2^m \rfloor$;  $d \leftarrow y \bmod 2^m$.

  $e \leftarrow$ MULTIPLY$(a, c, m)$.

  $f \leftarrow$ MULTIPLY$(b, d, m)$.

  $g \leftarrow$ MULTIPLY$(b, c, m)$.

  $h \leftarrow$ MULTIPLY$(a, d, m)$.

  RETURN $2^{2m} e + 2^m (g + h) + f$.

# Divide-and-conquer multiplication analysis

Proposition. The divide-and-conquer multiplication algorithm requires $\Theta(n^2)$ bit operations to multiply two $n$-bit integers.

Pf. Apply case 1 of the master theorem to the recurrence:

$$T(n) = \underbrace{4T(n/2)}_{\text{recursive calls}} + \underbrace{\Theta(n)}_{\text{add, shift}} \Rightarrow T(n) = \Theta(n^2)$$

# Karatsuba trick

To compute middle term $bc + ad$, use identity:

$$bc + ad = ac + bd - (a - b)(c - d)$$

$$m = \lceil n / 2 \rceil$$

$$a = \lfloor x / 2^m \rfloor \quad b = x \bmod 2^m$$

$$c = \lfloor y / 2^m \rfloor \quad d = y \bmod 2^m$$

middle term

$$(2^m a + b)(2^m c + d) = 2^{2m} ac + 2^m (bc + ad) + bd$$

$$= 2^{2m} ac + 2^m (ac + bd - (a - b)(c - d)) + bd$$

1 1 3 2 3

Bottom line. Only three multiplication of $n/2$-bit integers.

# Karatsuba multiplication

KARATSUBA-MULTIPLY$(x, y, n)$

IF $(n = 1)$

  RETURN $x \times y$.

ELSE

$m \leftarrow \lceil n / 2 \rceil$.

$a \leftarrow \lfloor x / 2^m \rfloor$; $b \leftarrow x \bmod 2^m$.

$c \leftarrow \lfloor y / 2^m \rfloor$; $d \leftarrow y \bmod 2^m$.

$e \leftarrow$ KARATSUBA-MULTIPLY$(a, c, m)$.

$f \leftarrow$ KARATSUBA-MULTIPLY$(b, d, m)$.

$g \leftarrow$ KARATSUBA-MULTIPLY$(a - b, c - d, m)$.

RETURN $2^{2m} e + 2^m (e + f - g) + f$.

## Karatsuba analysis

Proposition. Karatsuba's algorithm requires $O(n^{1.585})$ bit operations to multiply two $n$-bit integers.

Pf. Apply case 1 of the master theorem to the recurrence:

$$T(n) = 3\,T(n/2) + \Theta(n) \quad \Rightarrow \quad T(n) = \Theta(n^{\lg 3}) = O(n^{1.585}).$$

Practice. Faster than grade-school algorithm for about 320-640 bits.

# Integer arithmetic reductions

Integer multiplication. Given two $n$-bit integers, compute their product.

| problem | arithmetic | running time |
|---|---|---|
| integer multiplication | $a \times b$ | $\Theta(M(n))$ |
| integer division | $a / b,\ a \bmod b$ | $\Theta(M(n))$ |
| integer square | $a^2$ | $\Theta(M(n))$ |
| integer square root | $\lfloor \sqrt{a} \rfloor$ | $\Theta(M(n))$ |

**integer arithmetic problems with the same complexity as integer multiplication**

# History of asymptotic complexity of integer multiplication

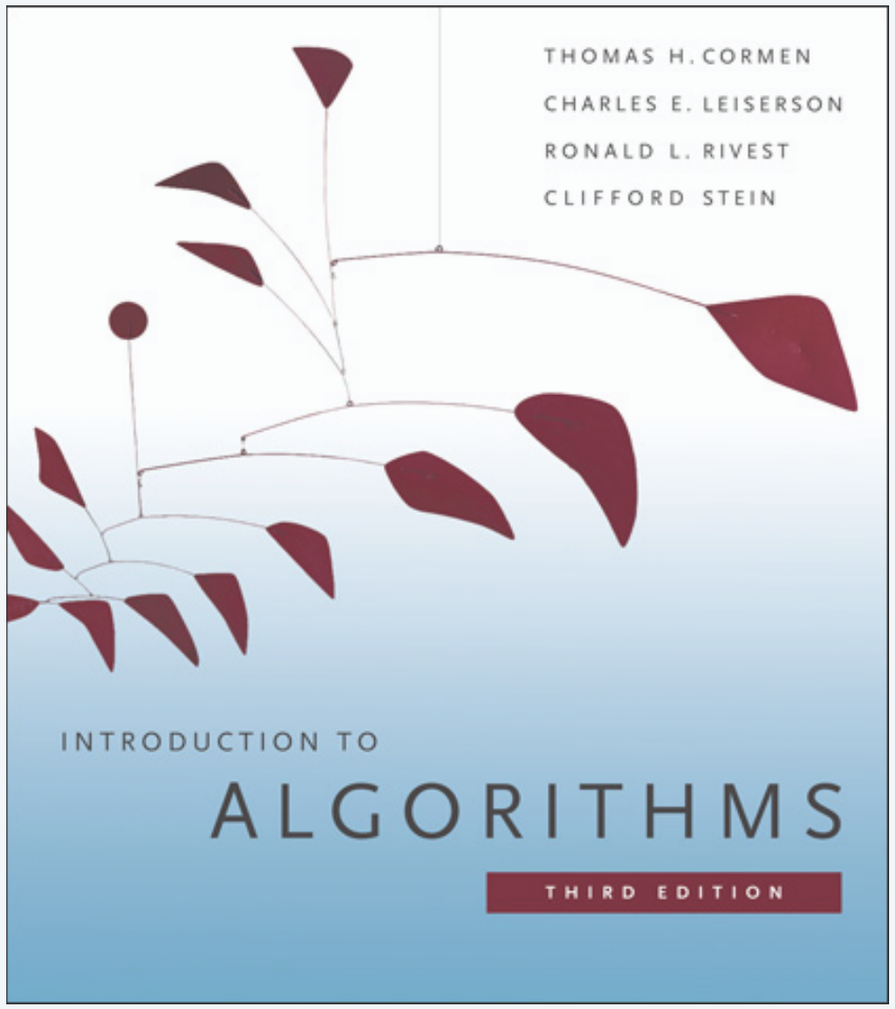
| year | algorithm | order of growth |
|---|---|---|
| ? | brute force | $\Theta(n^2)$ |
| 1962 | Karatsuba-Ofman | $\Theta(n^{1.585})$ |
| 1963 | Toom-3, Toom-4 | $\Theta(n^{1.465})$, $\Theta(n^{1.404})$ |
| 1966 | Toom-Cook | $\Theta(n^{1+\varepsilon})$ |
| 1971 | Schönhage–Strassen | $\Theta(n \log n \log \log n)$ |
| 2007 | Fürer | $n \log n \, 2^{O(\log^* n)}$ |
| ? | ? | $\Theta(n)$ |

**number of bit operations to multiply two n-bit integers**

used in Maple, Mathematica, gcc, cryptography, ...

Remark. GNU Multiple Precision Library uses one of five different algorithm depending on size of operands.

GMP
«Arithmetic without limitations»




# DIVIDE AND CONQUER II

- *master theorem*
- *integer multiplication*
- *matrix multiplication*
- *convolution and FFT*

SECTION 4.2

# Dot product

Dot product. Given two length $n$ vectors $a$ and $b$, compute $c = a \cdot b$.

Grade-school. $\Theta(n)$ arithmetic operations.

$$a \cdot b = \sum_{i=1}^{n} a_i b_i$$

$$a = \begin{bmatrix} .70 & .20 & .10 \end{bmatrix}$$

$$b = \begin{bmatrix} .30 & .40 & .30 \end{bmatrix}$$

$$a \cdot b = (.70 \times .30) + (.20 \times .40) + (.10 \times .30) = .32$$

Remark. Grade-school dot product algorithm is asymptotically optimal.

# Matrix multiplication

Matrix multiplication. Given two $n$-by-$n$ matrices $A$ and $B$, compute $C = AB$.

Grade-school. $\Theta(n^3)$ arithmetic operations.

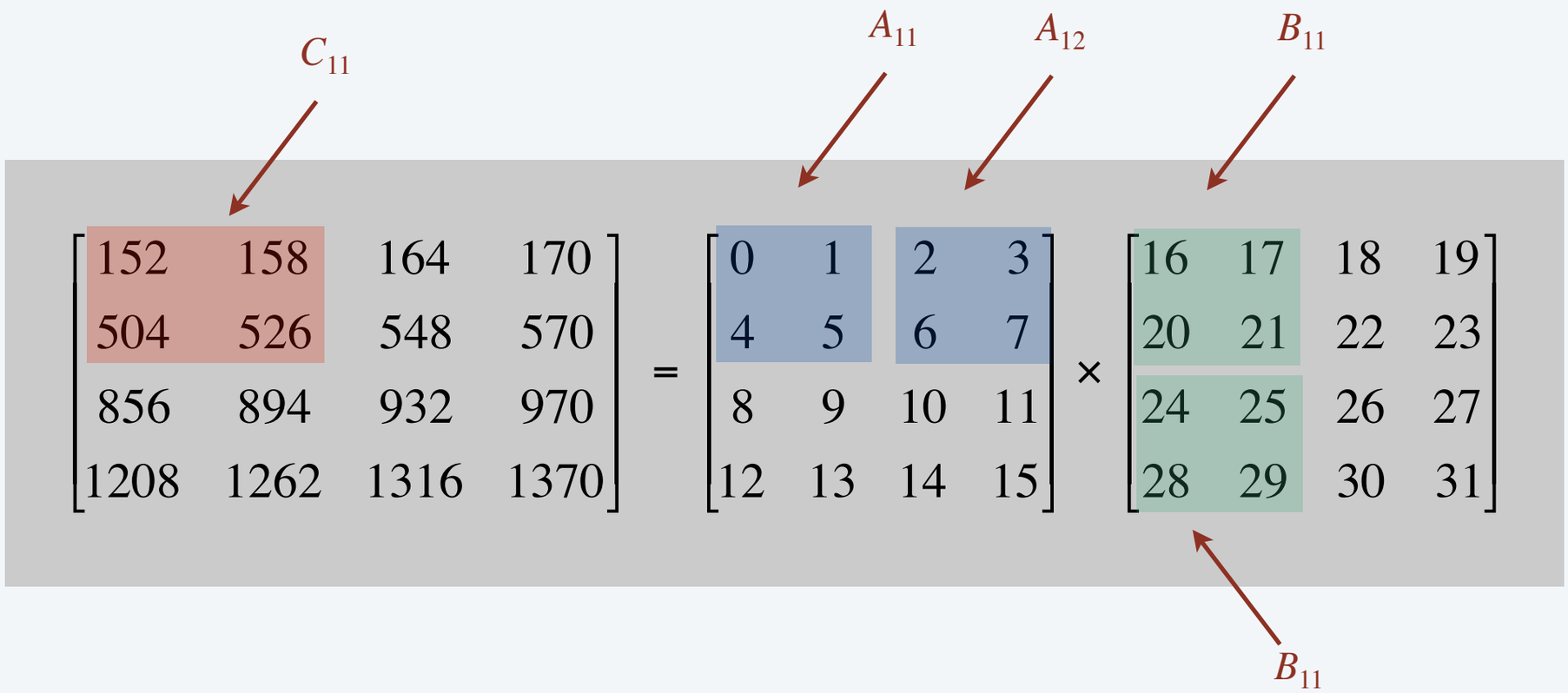
$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

$$\begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nn} \end{bmatrix}$$

$$\begin{bmatrix} .59 & .32 & .41 \\ .31 & .36 & .25 \\ .45 & .31 & .42 \end{bmatrix} = \begin{bmatrix} .70 & .20 & .10 \\ .30 & .60 & .10 \\ .50 & .10 & .40 \end{bmatrix} \times \begin{bmatrix} .80 & .30 & .50 \\ .10 & .40 & .10 \\ .10 & .30 & .40 \end{bmatrix}$$

Q. Is grade-school matrix multiplication algorithm asymptotically optimal?

# Block matrix multiplication




$$\begin{bmatrix} 152 & 158 & 164 & 170 \\ 504 & 526 & 548 & 570 \\ 856 & 894 & 932 & 970 \\ 1208 & 1262 & 1316 & 1370 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 \\ 8 & 9 & 10 & 11 \\ 12 & 13 & 14 & 15 \end{bmatrix} \times \begin{bmatrix} 16 & 17 & 18 & 19 \\ 20 & 21 & 22 & 23 \\ 24 & 25 & 26 & 27 \\ 28 & 29 & 30 & 31 \end{bmatrix}$$

$$C_{11} = A_{11} \times B_{11} + A_{12} \times B_{21} = \begin{bmatrix} 0 & 1 \\ 4 & 5 \end{bmatrix} \times \begin{bmatrix} 16 & 17 \\ 20 & 21 \end{bmatrix} + \begin{bmatrix} 2 & 3 \\ 6 & 7 \end{bmatrix} \times \begin{bmatrix} 24 & 25 \\ 28 & 29 \end{bmatrix} = \begin{bmatrix} 152 & 158 \\ 504 & 526 \end{bmatrix}$$

# Matrix multiplication: warmup

To multiply two $n$-by-$n$ matrices $A$ and $B$:

- Divide: partition $A$ and $B$ into $\frac{1}{2}n$-by-$\frac{1}{2}n$ blocks.
- Conquer: multiply $8$ pairs of $\frac{1}{2}n$-by-$\frac{1}{2}n$ matrices, recursively.
- Combine: add appropriate products using $4$ matrix additions.

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$$\begin{aligned} C_{11} &= (A_{11} \times B_{11}) + (A_{12} \times B_{21}) \\ C_{12} &= (A_{11} \times B_{12}) + (A_{12} \times B_{22}) \\ C_{21} &= (A_{21} \times B_{11}) + (A_{22} \times B_{21}) \\ C_{22} &= (A_{21} \times B_{12}) + (A_{22} \times B_{22}) \end{aligned}$$

Running time. Apply case 1 of Master Theorem.

$$T(n) = \underbrace{8T(n/2)}_{\text{recursive calls}} + \underbrace{\Theta(n^2)}_{\text{add, form submatrices}} \Rightarrow T(n) = \Theta(n^3)$$

# Strassen's trick

Key idea. multiply 2-by-2 blocks with only 7 multiplications.
(plus 11 additions and 7 subtractions)

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$$\begin{aligned} C_{11} &= P_5 + P_4 - P_2 + P_6 \\ C_{12} &= P_1 + P_2 \\ C_{21} &= P_3 + P_4 \\ C_{22} &= P_1 + P_5 - P_3 - P_7 \end{aligned}$$

$$\begin{aligned} P_1 &\leftarrow A_{11} \times (B_{12} - B_{22}) \\ P_2 &\leftarrow (A_{11} + A_{12}) \times B_{22} \\ P_3 &\leftarrow (A_{21} + A_{22}) \times B_{11} \\ P_4 &\leftarrow A_{22} \times (B_{21} - B_{11}) \\ P_5 &\leftarrow (A_{11} + A_{22}) \times (B_{11} + B_{22}) \\ P_6 &\leftarrow (A_{12} - A_{22}) \times (B_{21} + B_{22}) \\ P_7 &\leftarrow (A_{11} - A_{21}) \times (B_{11} + B_{12}) \end{aligned}$$

Pf. $C_{12} = P_1 + P_2$

$= A_{11} \times (B_{12} - B_{22}) + (A_{11} + A_{12}) \times B_{22}$

$= A_{11} \times B_{12} + A_{12} \times B_{22}$. ✔

# Strassen's algorithm

assume n is a power of 2

STRASSEN $(n, A, B)$

IF $(n = 1)$ RETURN $A \times B$.

Partition $A$ and $B$ into 2-by-2 block matrices.

$P_1 \leftarrow$ STRASSEN $(n / 2, A_{11}, (B_{12} - B_{22}))$.

$P_2 \leftarrow$ STRASSEN $(n / 2, (A_{11} + A_{12}), B_{22})$.

$P_3 \leftarrow$ STRASSEN $(n / 2, (A_{21} + A_{22}), B_{11})$.

$P_4 \leftarrow$ STRASSEN $(n / 2, A_{22}, (B_{21} - B_{11}))$.

$P_5 \leftarrow$ STRASSEN $(n / 2, (A_{11} + A_{22}) \times (B_{11} + B_{22}))$.

$P_6 \leftarrow$ STRASSEN $(n / 2, (A_{12} - A_{22}) \times (B_{21} + B_{22}))$.

$P_7 \leftarrow$ STRASSEN $(n / 2, (A_{11} - A_{21}) \times (B_{11} + B_{12}))$.

$C_{11} = P_5 + P_4 - P_2 + P_6$.

$C_{12} = P_1 + P_2$.

$C_{21} = P_3 + P_4$.

$C_{22} = P_1 + P_5 - P_3 - P_7$.

RETURN $C$.

keep track of indices of submatrices (don't copy matrix entries)

# Analysis of Strassen's algorithm

Theorem. Strassen's algorithm requires $O(n^{2.81})$ arithmetic operations to multiply two $n$-by-$n$ matrices.

Pf. Apply case 1 of the master theorem to the recurrence:

$$T(n) = \underbrace{7T(n/2)}_{\text{recursive calls}} + \underbrace{\Theta(n^2)}_{\text{add, subtract}} \Rightarrow T(n) = \Theta(n^{\log_2 7}) = O(n^{2.81})$$

Q. What if $n$ is not a power of $2$?

A. Could pad matrices with zeros.

$$\begin{bmatrix} 1 & 2 & 3 & 0 \\ 4 & 5 & 6 & 0 \\ 7 & 8 & 9 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 10 & 11 & 12 & 0 \\ 13 & 14 & 15 & 0 \\ 16 & 17 & 18 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 84 & 90 & 96 & 0 \\ 201 & 216 & 231 & 0 \\ 318 & 342 & 366 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

# Strassen's algorithm: practice

Implementation issues.

- Sparsity.
- Caching effects.
- Numerical stability.
- Odd matrix dimensions.
- Crossover to classical algorithm when $n$ is "small" .

Common misperception. *"Strassen is only a theoretical curiosity."*

- Apple reports $8$x speedup on G4 Velocity Engine when $n \approx 2{,}048$.
- Range of instances where it's useful is a subject of controversy.

# Linear algebra reductions

Matrix multiplication. Given two $n$-by-$n$ matrices, compute their product.

| problem | linear algebra | order of growth |
|---|---|---|
| matrix multiplication | $A \times B$ | $\Theta(MM(n))$ |
| matrix inversion | $A^{-1}$ | $\Theta(MM(n))$ |
| determinant | $\lvert A \rvert$ | $\Theta(MM(n))$ |
| system of linear equations | $Ax = b$ | $\Theta(MM(n))$ |
| LU decomposition | $A = L\,U$ | $\Theta(MM(n))$ |
| least squares | $\min \lVert Ax - b \rVert_2$ | $\Theta(MM(n))$ |

**numerical linear algebra problems with the same complexity as matrix multiplication**

# Fast matrix multiplication: theory

Q. Multiply two 2-by-2 matrices with 7 scalar multiplications?

A. Yes! [Strassen 1969] $\Theta(n^{\log_2 7}) = O(n^{2.807})$

Q. Multiply two 2-by-2 matrices with 6 scalar multiplications?

A. Impossible. [Hopcroft and Kerr 1971] $\Theta(n^{\log_2 6}) = O(n^{2.59})$

Q. Multiply two 3-by-3 matrices with 21 scalar multiplications?

A. Unknown. $\Theta(n^{\log_3 21}) = O(n^{2.77})$

Begun, the decimal wars have. [Pan, Bini et al, Schönhage, ...]

- Two 20-by-20 matrices with 4,460 scalar multiplications. $O(n^{2.805})$
- Two 48-by-48 matrices with 47,217 scalar multiplications. $O(n^{2.7801})$
- A year later. $O(n^{2.7799})$
- December 1979. $O(n^{2.521813})$
- January 1980. $O(n^{2.521801})$

# History of asymptotic complexity of matrix multiplication

| year | algorithm | order of growth |
|---|---|---|
| ? | brute force | $O(n^3)$ |
| 1969 | Strassen | $O(n^{2.808})$ |
| 1978 | Pan | $O(n^{2.796})$ |
| 1979 | Bini | $O(n^{2.780})$ |
| 1981 | Schönhage | $O(n^{2.522})$ |
| 1982 | Romani | $O(n^{2.517})$ |
| 1982 | Coppersmith-Winograd | $O(n^{2.496})$ |
| 1986 | Strassen | $O(n^{2.479})$ |
| 1989 | Coppersmith-Winograd | $O(n^{2.376})$ |
| 2010 | Strother | $O(n^{2.3737})$ |
| 2011 | Williams | $O(n^{2.3727})$ |
| ? | ? | $O(n^{2+\varepsilon})$ |

**number of floating-point operations to multiply two n-by-n matrices**