CS 580: Algorithm Design and Analysis

Jeremiah Blocki
Purdue University
Spring 2019

Announcement: Homework 2 due on Tuesday, February 5^{th} at 11:59PM (Gradescope)

Recap: Divide and Conquer

Key Paradigm in Algorithm Design:
- Break up problem into several parts.
- Solve each part recursively.
- Combine solutions to sub-problems into overall solution.

Merge-Sort: Sort a list in time $O(n \log n)$
- Split list in half and sort each half
- Merge the sorted lists

Recurrence Relationships
- Solving: Recursion Trees, Telescoping, Induction
- Master Theorem: Generic solution for $T(n) = a\,T(n/b)+n^c$
- Other Recurrence Relationships

Counting Inversions: (in time $O(n \log n)$)
- Count number of pairs $i < j$ s.t. $A[i] > A[j]$
- Merge **and** Sort

Counting Inversions: Divide-and-Conquer

Divide-and-conquer.

- Divide: separate list into two pieces.
- Conquer: recursively count inversions in each half.
- Combine: count inversions where a_i and a_j are in different halves, and return sum of three quantities.

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

Divide: O(1).

1	5	4	8	10	2

6	9	12	11	3	7

5 blue-blue inversions

8 green-green inversions

Conquer: 2T(n / 2)

9 blue-green inversions
5-3, 4-3, 8-6, 8-3, 8-7, 10-6, 10-9, 10-3, 10-7

Combine: ???

Total = 5 + 8 + 9 = 22.

Counting Inversions: Combine

play

Combine: count blue-green inversions

- Assume each half is sorted.
- Count inversions where a_i and a_j are in different halves.
- Merge two sorted halves into sorted whole.

to maintain sorted invariant

3	7	10	14	18	19

2	11	16	17	23	25
6	3	2	2	0	0

13 blue-green inversions: 6 + 3 + 2 + 2 + 0 + 0 Count: O(n)

2	3	7	10	11	14	16	17	18	19	23	25

Merge: O(n)

$$T(n) \le T\left(\lfloor n/2 \rfloor\right) + T\left(\lceil n/2 \rceil\right) + O(n) \Rightarrow \mathrm{T}(n) = O(n \log n)$$

Counting Inversions: Implementation

Pre-condition. [Merge-and-Count] A and B are sorted.
Post-condition. [Sort-and-Count] L is sorted.

```
Sort-and-Count(L) {
   if list L has one element
      return 0 and the list L

   Divide the list into two halves A and B
   (rA, A) ← Sort-and-Count(A)
   (rB, B) ← Sort-and-Count(B)
   (r , L) ← Merge-and-Count(A, B)

   return r = rA + rB + r and the sorted list L
}
```

5.4 Closest Pair of Points

Closest Pair of Points

Closest pair. Given n points in the plane, find a pair with 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

Brute force. Check all pairs of points p and q with $\Theta(n^2)$ comparisons.

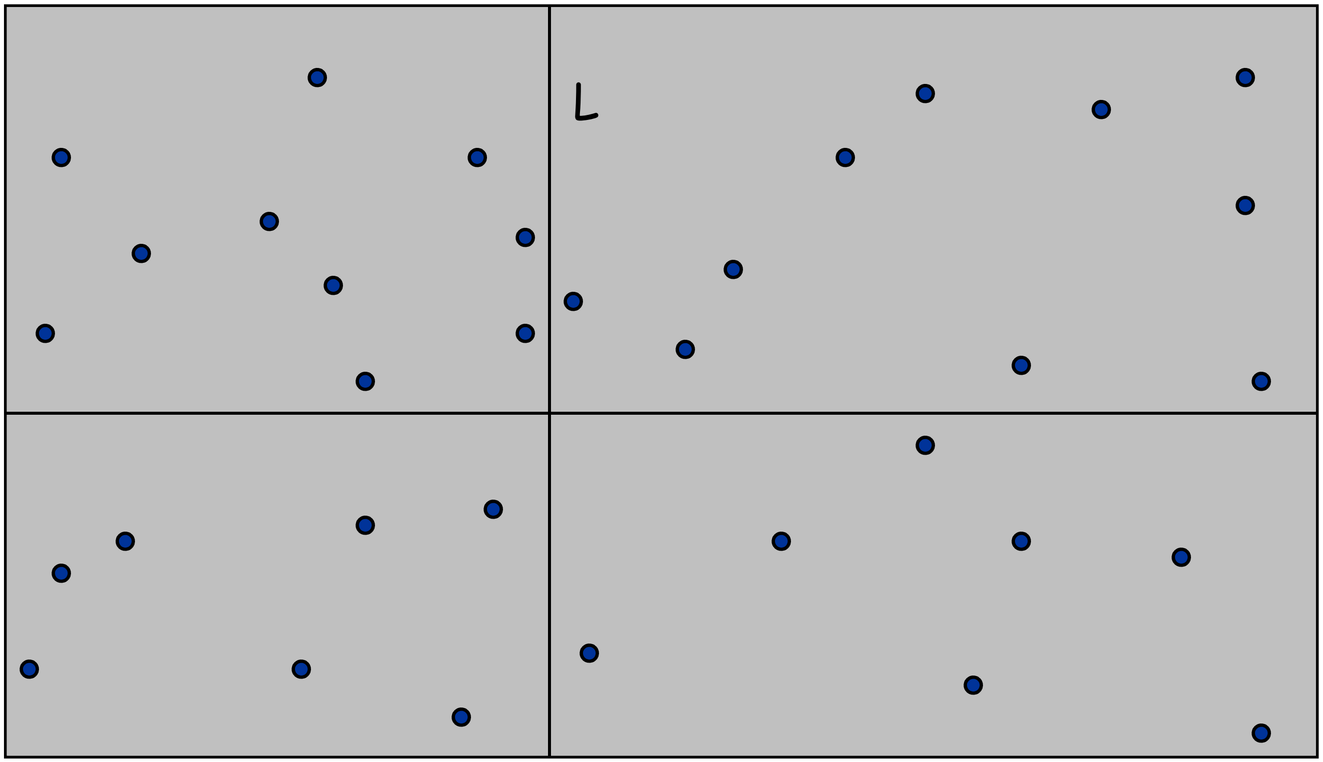
1-D version. O(n log n) easy if points are on a line.

Assumption. No two points have same x coordinate.

to make presentation cleaner

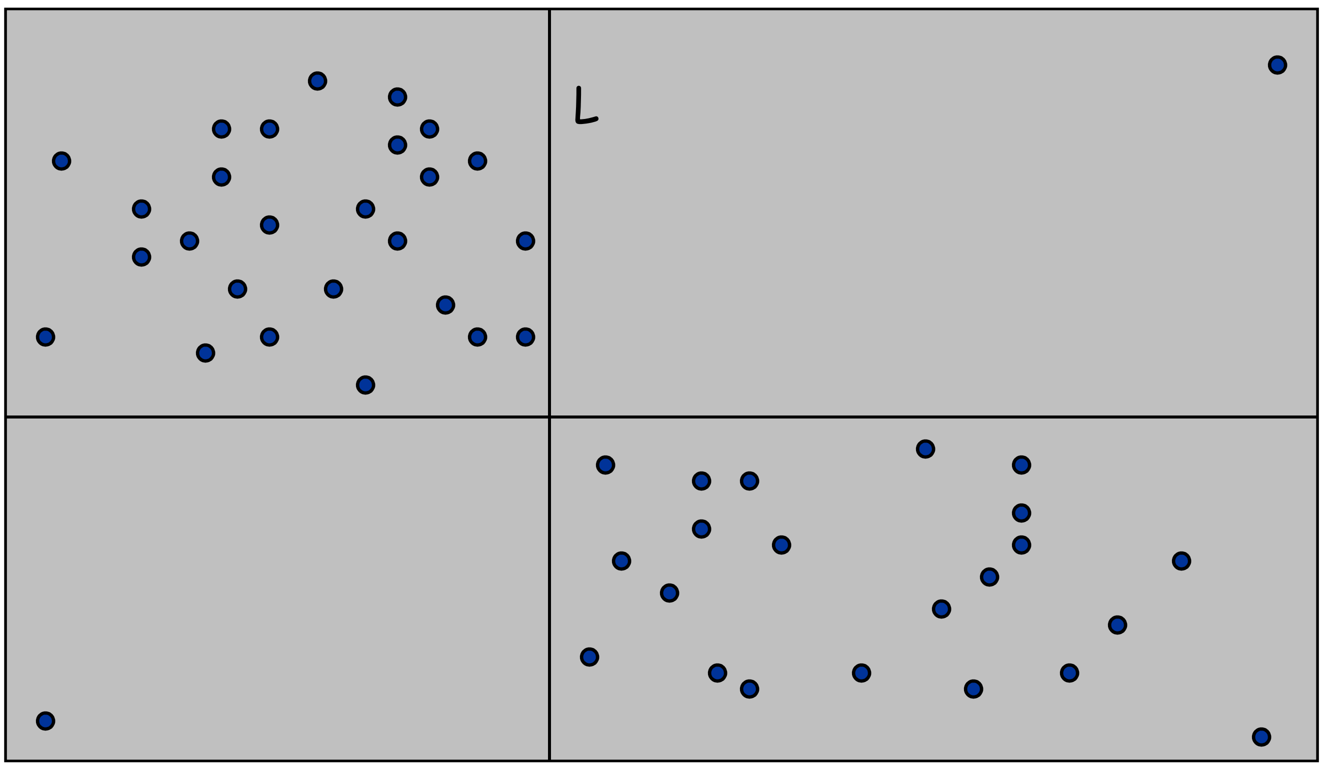
Closest Pair of Points: First Attempt

Divide. Sub-divide region into 4 quadrants.



Closest Pair of Points: First Attempt

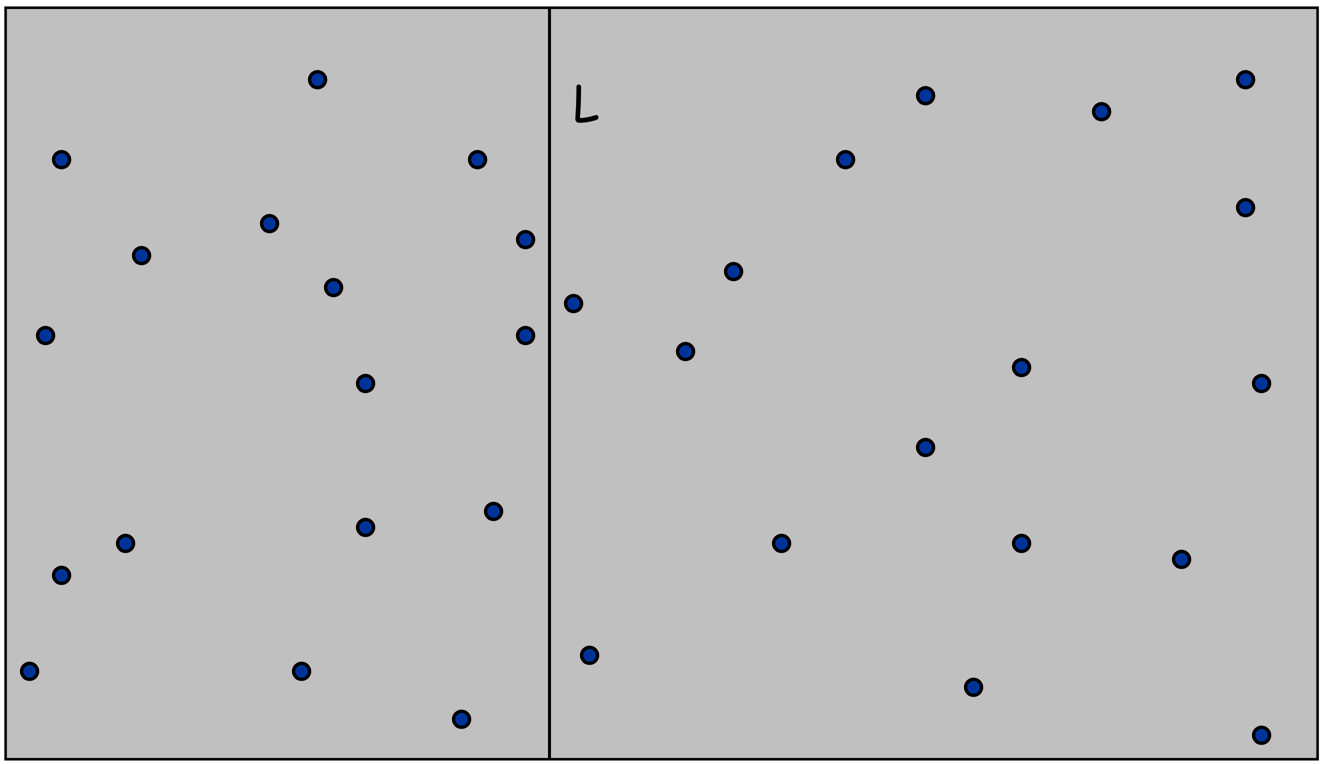
Divide. Sub-divide region into 4 quadrants.
Obstacle. Impossible to ensure n/4 points in each piece.



Closest Pair of Points

Algorithm.

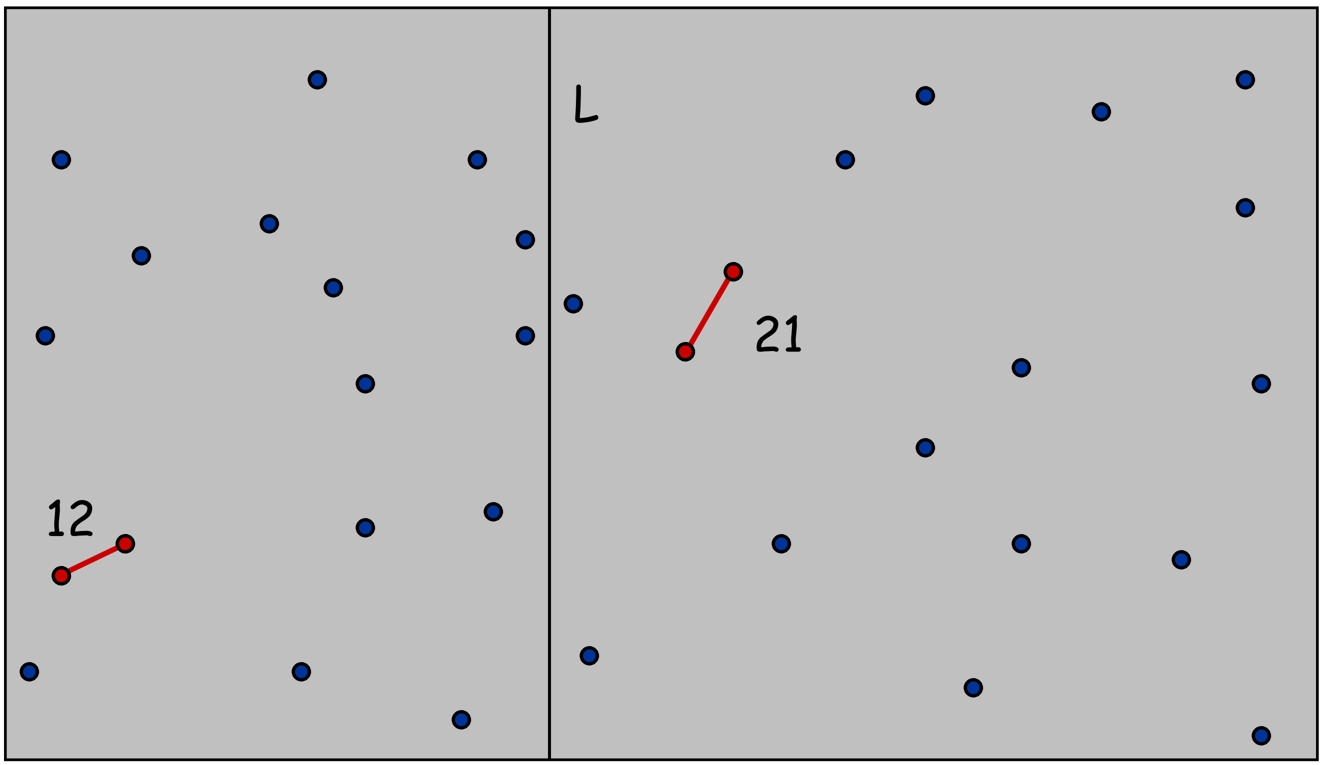
- Divide: draw vertical line L so that roughly $\frac{1}{2}$n points on each side.



Closest Pair of Points

Algorithm.

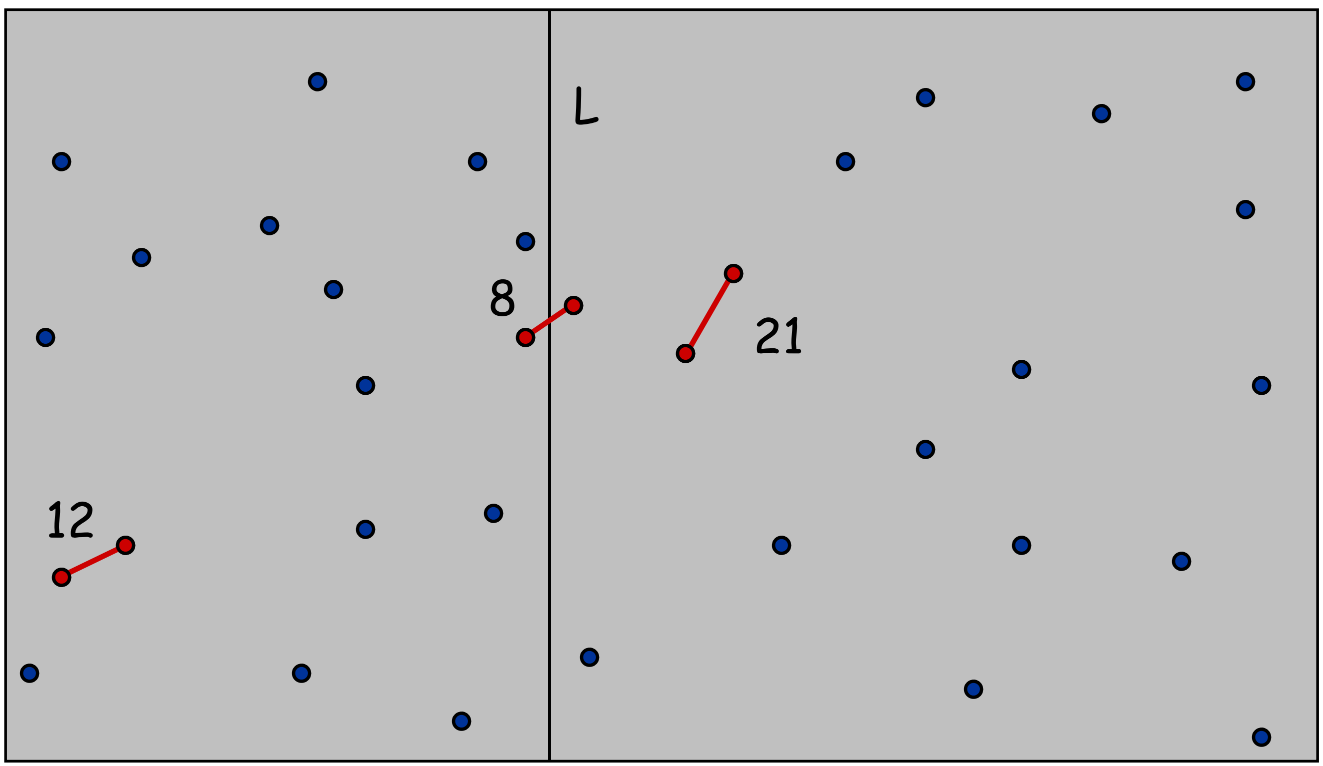
- Divide: draw vertical line L so that roughly $\frac{1}{2}n$ points on each side.
- Conquer: find closest pair in each side recursively.



Closest Pair of Points

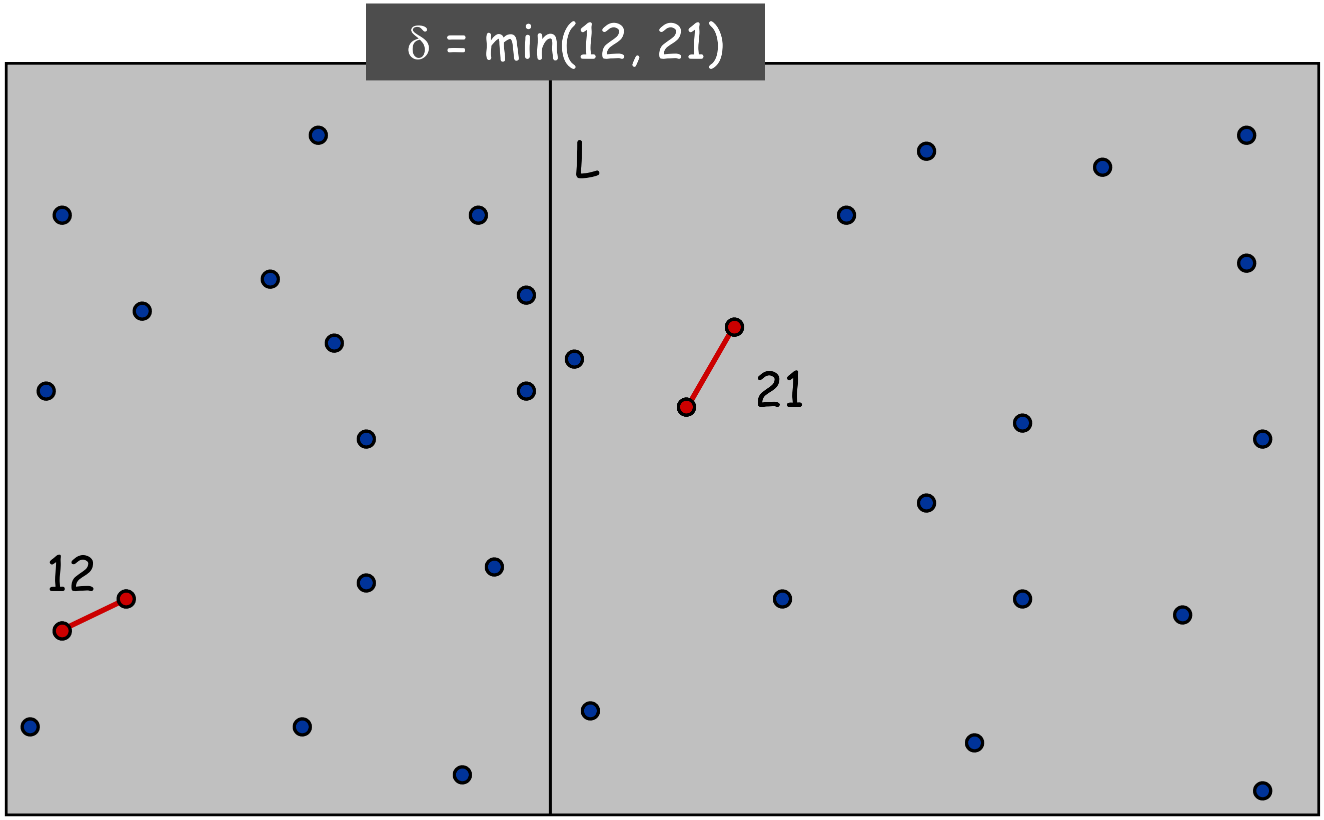
Algorithm.

- Divide: draw vertical line L so that roughly $\frac{1}{2}n$ points on each side.
- Conquer: find closest pair in each side recursively.
- Combine: find closest pair with one point in each side. ← seems like $\Theta(n^2)$
- Return best of 3 solutions.



Closest Pair of Points

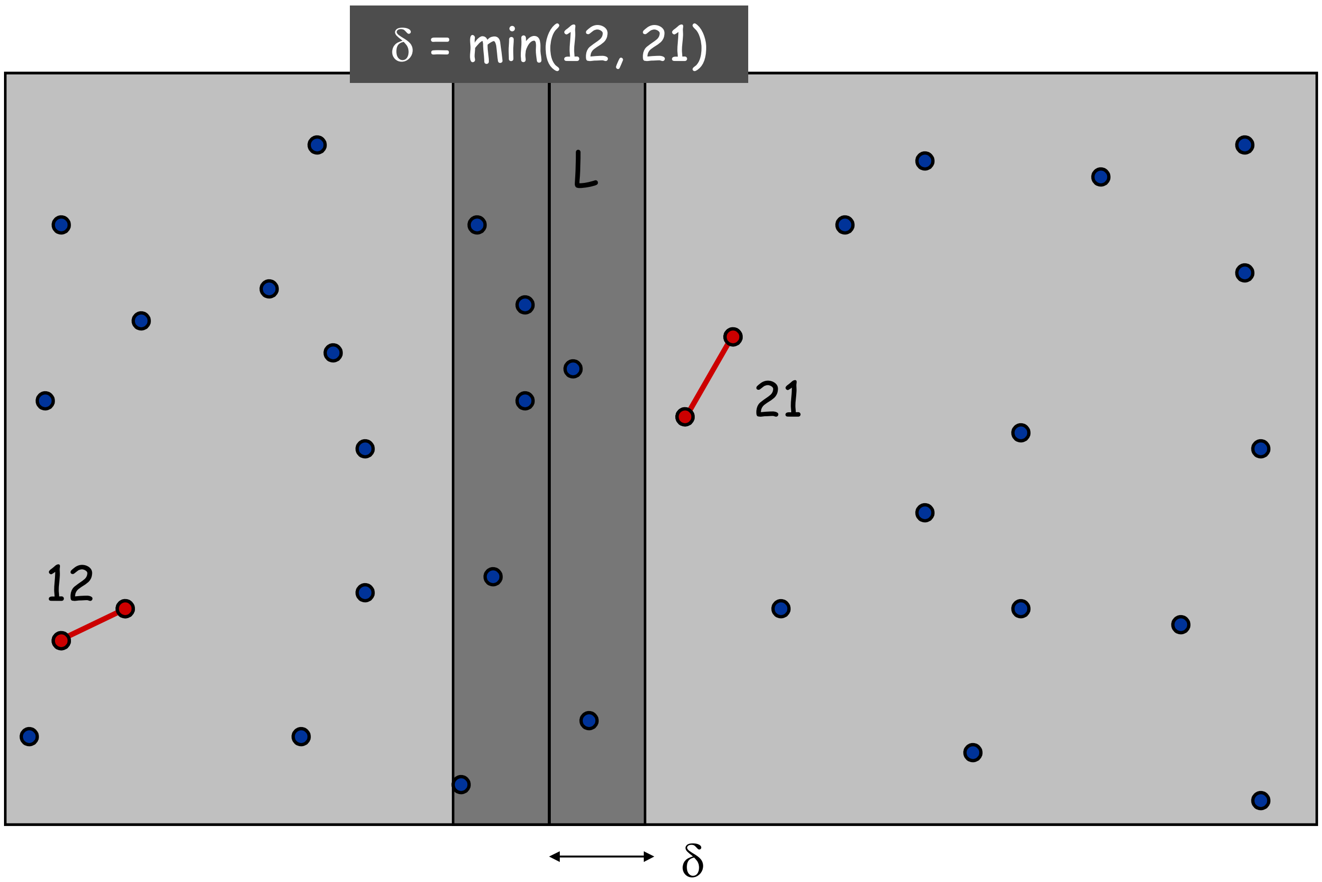
Find closest pair with one point in each side, assuming that distance < δ.



Closest Pair of Points

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

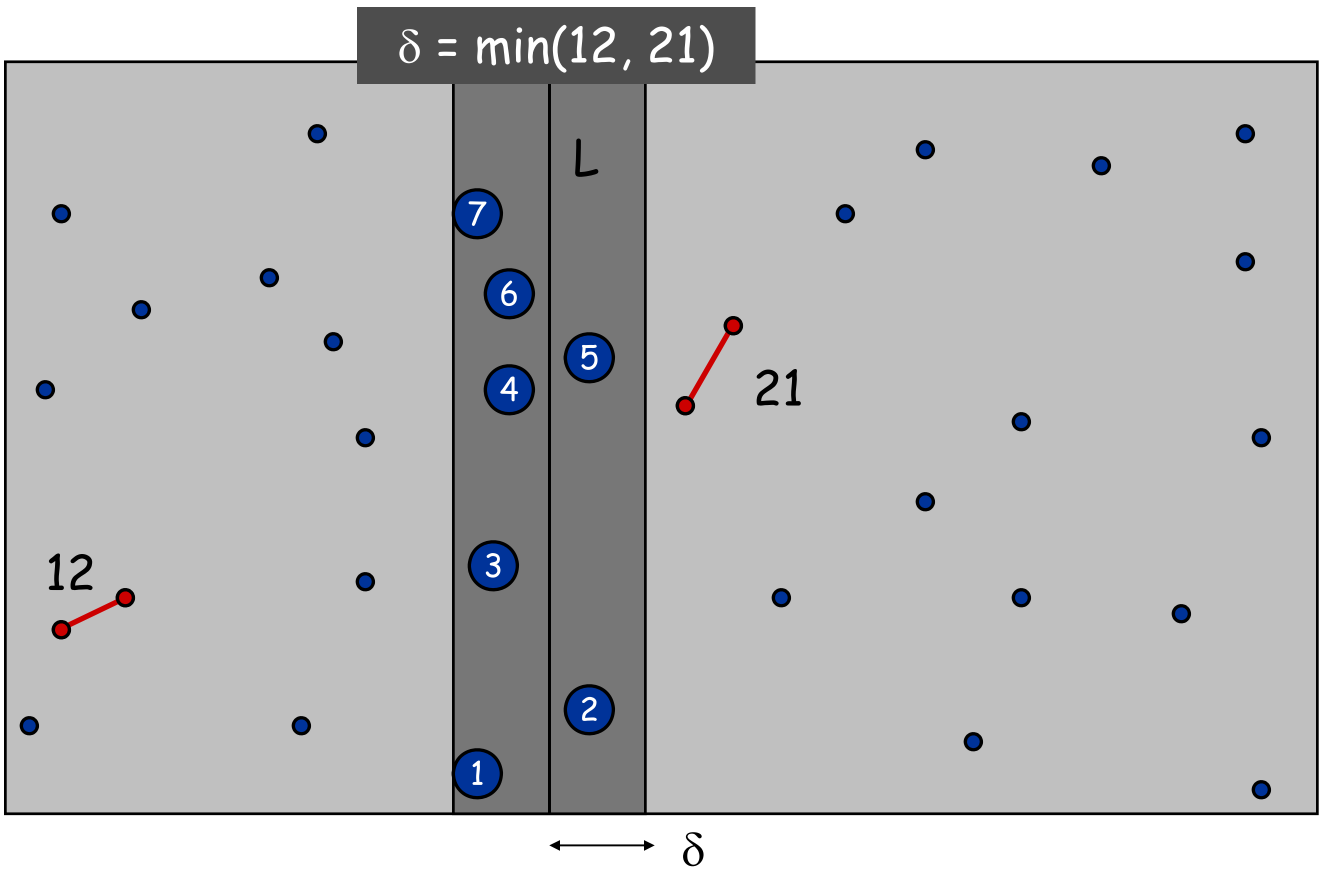
- Observation: only need to consider points within δ of line L.



Closest Pair of Points

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

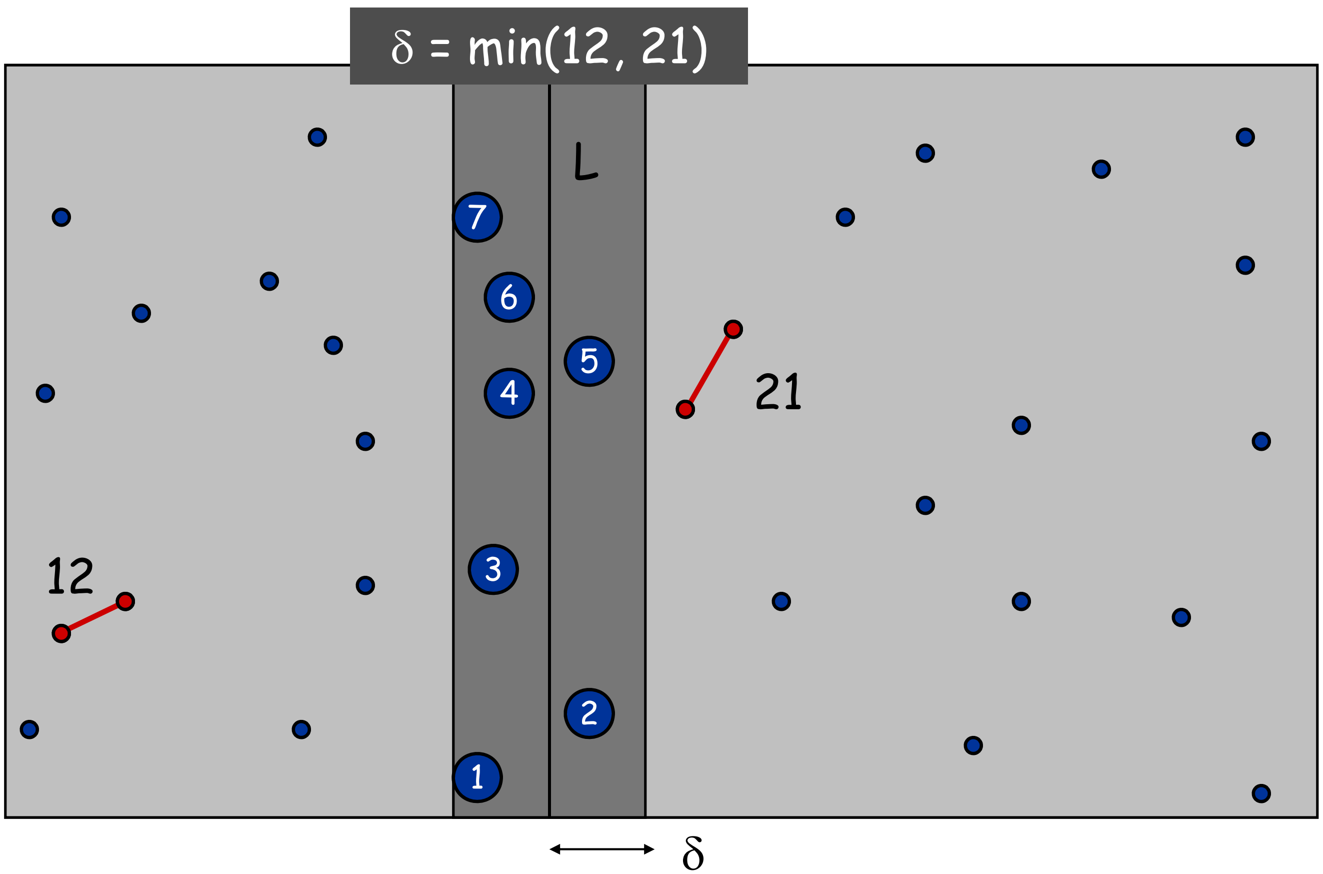
- Observation: only need to consider points within δ of line L.
- Sort points in 2δ-strip by their y coordinate.



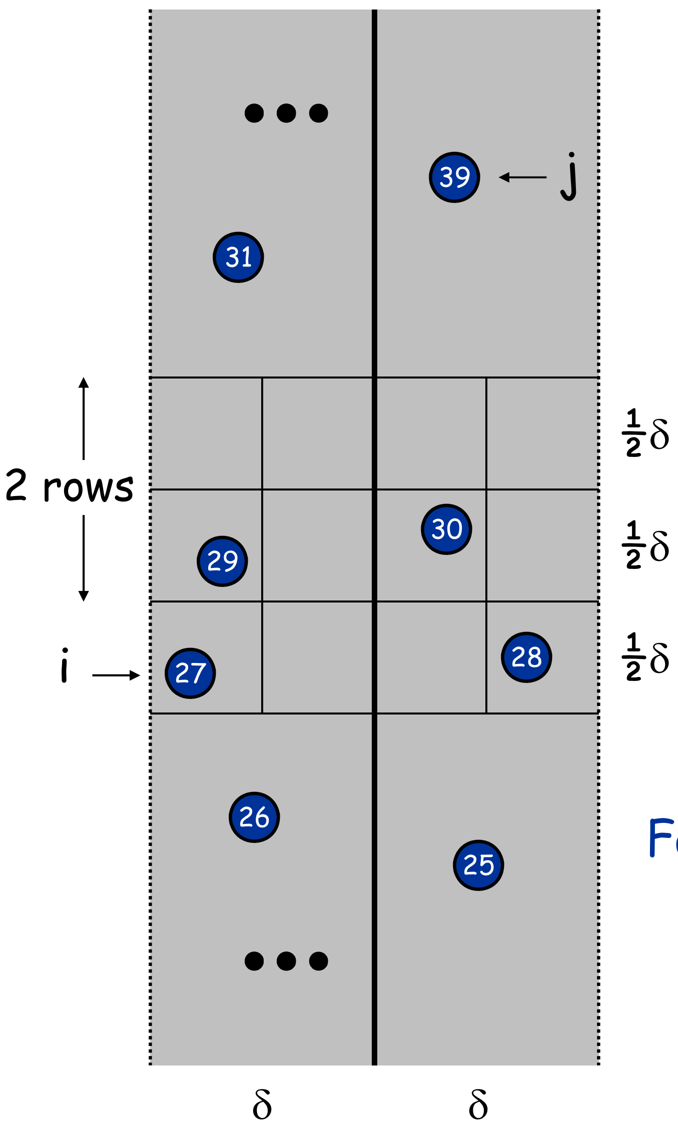
Closest Pair of Points

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

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



Closest Pair of Points



Def. Let s_i be the point in the 2δ-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 δ.

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. Still true if we replace 12 with 7.

Closest Pair Algorithm

```
Closest-Pair(p₁, …, pₙ) {
   Compute separation line L such that half the points
   are on one side and half on the other side.

   δ₁ = Closest-Pair(left half)
   δ₂ = Closest-Pair(right half)
   δ  = min(δ₁, δ₂)

   Delete all points further than δ from separation line L

   Sort remaining points by y-coordinate.

   Scan points in y-order and compare distance between
   each point and next 11 neighbors. If any of these
   distances is less than δ, update δ.

   return δ.
}
```

$O(n \log n)$

$2T(n / 2)$

$O(n)$

$O(n \log n)$

$O(n)$

Closest Pair of Points: Analysis

Running time.

$$T(n) \le 2T\big(n/2\big) + O(n \log n) \;\Rightarrow\; T(n) = O(n \log^2 n)$$

Q. Can we achieve 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 y coordinate, and all points sorted by x coordinate.
- Sort by merging two pre-sorted lists.

$$T(n) \le 2T\big(n/2\big) + O(n) \;\Rightarrow\; T(n) = O(n \log n)$$

5.5 Integer Multiplication

Motivation: Complex Multiplication

Complex multiplication. $(a + bi)\,(c + di) = x + yi.$

Grade-school. $x = ac - bd,\ \ y = bc + ad.$

4 multiplications, 2 additions

Q. Is it possible to do with fewer multiplications?

Our Prices Are Fantastic!

Multiplication: \$100 (reals only $\mathbb{R}$)
Addition: \$1 (reals only $\mathbb{R}$)

\$402 for Grade-School Approach: 4 multiplications, 2 additions

Complex Multiplication

Complex multiplication. $(a + bi)\ (c + di) = x + yi.$

Grade-school. $x = ac - bd,\ \ y = bc + ad.$

4 multiplications, 2 additions

Q. Is it possible to do with fewer multiplications?

A. Yes. [Gauss] $x = ac - bd,\ \ y = (a + b)\ (c + d) - ac - bd.$

$(= \mathrm{ac} + \mathrm{ad} + \mathrm{bc} + \mathrm{bd} - ac - bd = bc + ad)$

3 multiplications, 5 additions ($305)

Remark. Improvement if no hardware multiply.

Integer Addition

Addition. Given two n-bit integers x and y, compute $x + y$.

Grade-school. $\Theta(n)$ 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 algorithm is optimal.

Integer Multiplication

Multiplication. Given two n-bit integers x and y, compute $x \times y$.
Grade-school. $\Theta(n^2)$ 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 0
            1 1 0 1 0 1 0 1 0
          1 1 0 1 0 1 0 1 0
        1 1 0 1 0 1 0 1 0
      1 1 0 1 0 1 0 1 0
    1 1 0 1 0 1 0 1 0
  0 0 0 0 0 0 0 0 0
-------------------------------
0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 1
```

Q. Is grade-school multiplication algorithm optimal?

Divide-and-Conquer Multiplication: Warmup

To multiply two n-bit integers x and y:

- Multiply four $\frac{1}{2}n$-bit integers, recursively.
- Add and shift to obtain result.

$$x = 2^{n/2} \cdot x_1 + x_0$$
$$y = 2^{n/2} \cdot y_1 + y_0$$
$$xy = \left(2^{n/2} \cdot x_1 + x_0\right)\left(2^{n/2} \cdot y_1 + y_0\right)$$
$$= 2^n \cdot \underbrace{x_1 y_1}_{1} + 2^{\frac{n}{2}} \cdot (\underbrace{x_0 y_1}_{2} + \underbrace{x_1 y_0}_{3}) + \underbrace{x_0 y_0}_{4}$$

Ex. $x = \underbrace{1000}_{x_1}\underbrace{1101}_{x_0}$ $\quad y = \underbrace{1110}_{y_1}\underbrace{0001}_{y_0}$

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

Divide-and-Conquer Multiplication: Warmup

To multiply two n-bit integers x and y:

- Multiply four $\frac{1}{2}n$-bit integers, recursively.
- Add and shift to obtain result.

$$x = 2^{n/2} \cdot x_1 + x_0$$
$$y = 2^{n/2} \cdot y_1 + y_0$$
$$xy = \left(2^{n/2} \cdot x_1 + x_0\right)\left(2^{n/2} \cdot y_1 + y_0\right)$$
$$= 2^n \cdot \underbrace{x_1 y_1}_{1} + 2^{\frac{n}{2}} \cdot (\underbrace{x_0 y_1}_{2} + \underbrace{x_1 y_0}_{3}) + \underbrace{x_0 y_0}_{4}$$

Bit Shifts: O(n) cheap

Ex. $x = \underbrace{1000}_{x_1}\underbrace{1101}_{x_0}$ $y = \underbrace{1110}_{y_1}\underbrace{0001}_{y_0}$

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

Master's Theorem: a = 4, b=2, c=1 $\left(\frac{a}{b^c}\right) > 1,\ O\left(n^{\log_b a}\right) = O(n^2)$

Divide-and-Conquer Multiplication: Warmup

To multiply two n-bit integers x and y:

- Multiply four $\frac{1}{2}n$-bit integers, recursively.
- Add and shift to obtain result.

$$x = 2^{n/2} \cdot x_1 + x_0$$
$$y = 2^{n/2} \cdot y_1 + y_0$$
$$xy = \left(2^{n/2} \cdot x_1 + x_0\right)\left(2^{n/2} \cdot y_1 + y_0\right)$$
$$= 2^n \cdot \underset{1}{x_1 y_1} + 2^{\frac{n}{2}} \cdot (\underset{2}{x_0 y_1} + \underset{3}{x_1 y_0}) + \underset{4}{x_0 y_0}$$

Bit Shifts: O(n) cheap

Ex. $x = \underbrace{1000}_{x_1}\underbrace{1101}_{x_0}$ $\quad y = \underbrace{1110}_{y_1}\underbrace{0001}_{y_0}$

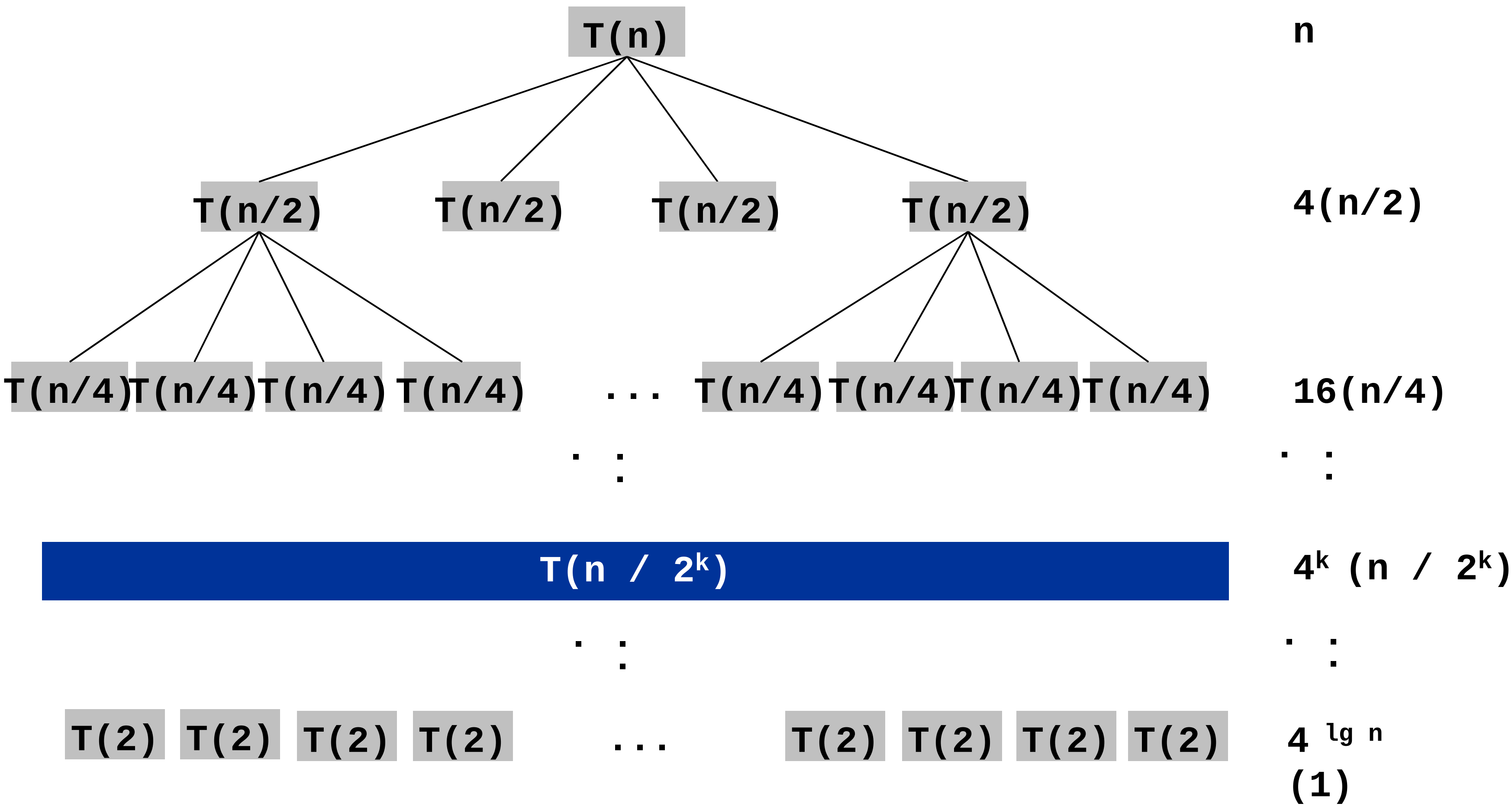
$$T(n) = \underbrace{4T(n/2)}_{\text{recursive calls}} + \underbrace{\Theta(n)}_{\text{add, shift}} \Rightarrow T(n) = \Theta(n^2)$$

Master's Theorem: a = 4, b=2, c=1 $\left(\frac{a}{b^c}\right) > 1$, $O\left(n^{\log_b a}\right) = O(n^2)$

Recursion Tree

$$T(n) = \begin{cases} 0 & \text{if } n = 0 \\ 4T(n/2) + n & \text{otherwise} \end{cases}$$

$$T(n) = \sum_{k=0}^{\lg n} n\, 2^k \;=\; n\left(\frac{2^{1+\lg n} - 1}{2-1}\right) \;=\; 2n^2 - n$$



Karatsuba Multiplication

To multiply two n-bit integers x and y:

- Add two $\frac{1}{2}n$ bit integers.
- Multiply three $\frac{1}{2}n$-bit integers, recursively.
- Add, subtract, and shift to obtain result.

$$x = 2^{n/2} \cdot x_1 + x_0$$
$$y = 2^{n/2} \cdot y_1 + y_0$$

$$\begin{aligned} xy &= 2^n \cdot x_1 y_1 + 2^{\frac{n}{2}} \cdot (x_0 y_1 + x_1 y_0) + x_0 y_0 \\ &= 2^n \cdot \underbrace{x_1 y_1}_{1} + 2^{\frac{n}{2}} \cdot \big(\underbrace{(x_0 + x_1)(y_0 + y_1)}_{2} - \underbrace{x_0 y_0}_{3} - \underbrace{x_1 y_1}_{1} \big) + \underbrace{x_0 y_0}_{3} \end{aligned}$$

Karatsuba Multiplication

To multiply two n-bit integers x and y:

- Add two $\frac{1}{2}n$ bit integers.
- Multiply three $\frac{1}{2}n$-bit integers, recursively.
- Add, subtract, and shift to obtain result.

$$x = 2^{n/2} \cdot x_1 + x_0$$
$$y = 2^{n/2} \cdot y_1 + y_0$$

$$xy = 2^n \cdot x_1y_1 + 2^{\frac{n}{2}} \cdot (x_0y_1 + x_1y_0) + x_0y_0$$
$$= 2^n \cdot \underbrace{x_1y_1}_{1} + 2^{\frac{n}{2}} \cdot \big(\underbrace{(x_0 + x_1)(y_0 + y_1)}_{2} - \underbrace{x_0y_0}_{3} - \underbrace{x_1y_1}_{1}\big) + \underbrace{x_0y_0}_{3}$$

Theorem. [Karatsuba-Ofman 1962] Can multiply two n-bit integers in $O(n^{1.585})$ bit operations.

$$T(n) \le \underbrace{T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + T(1 + \lceil n/2 \rceil)}_{\text{recursive calls}} + \underbrace{\Theta(n)}_{\text{add, subtract, shift}} \quad \Rightarrow T(n)$$

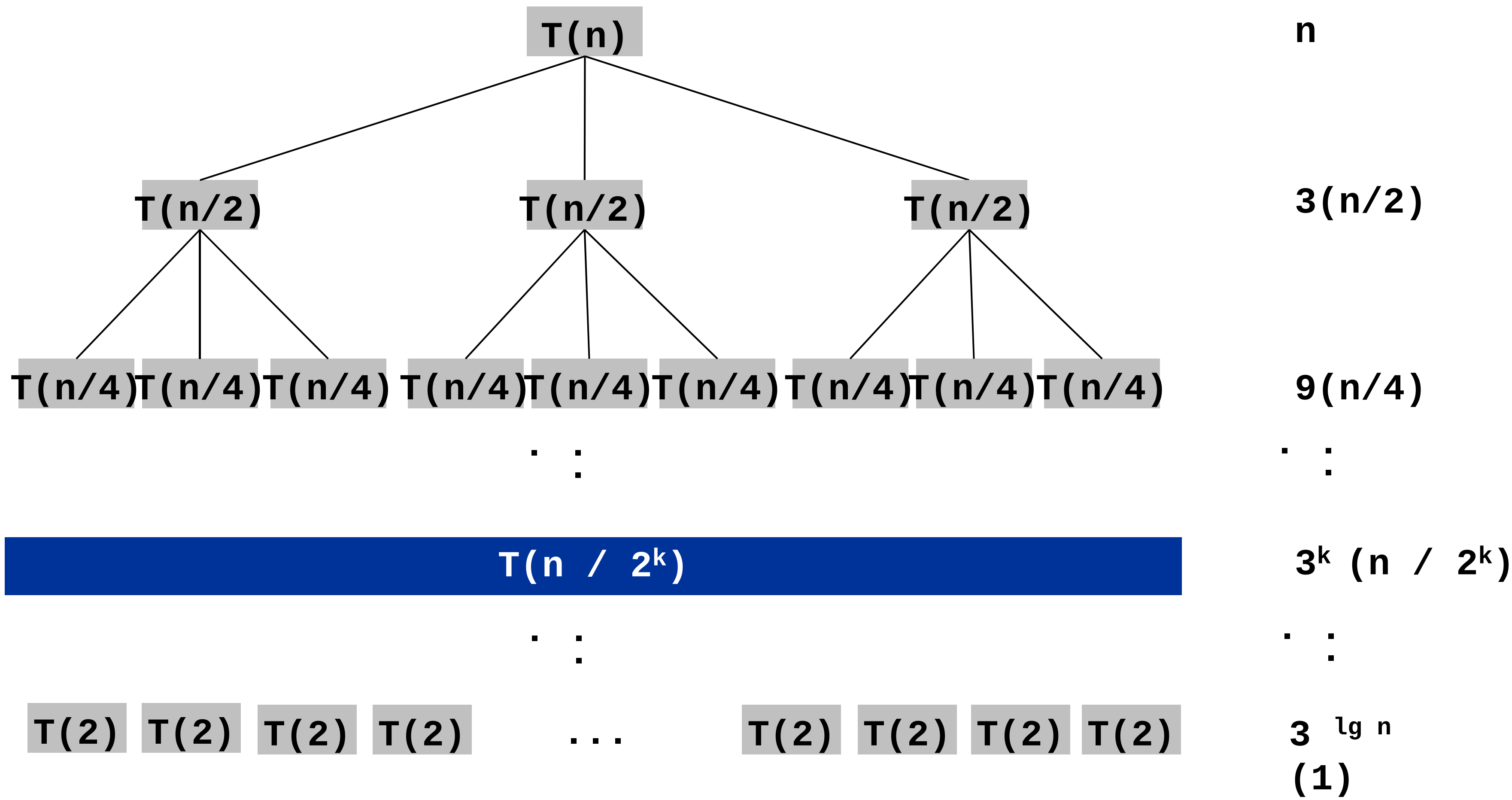
Master's Theorem: a = 3, b=2, c=1 $\left(\frac{a}{b^c}\right) > 1 \Rightarrow T(n) \in O\left(n^{\log_b a}\right)$

$[\log_2 3 < 1.585]$

Karatsuba: Recursion Tree

$$T(n) = \begin{cases} 0 & \text{if } n = 0 \\ 3T(n/2) + n & \text{otherwise} \end{cases}$$

$$T(n) = \sum_{k=0}^{\lg n} n \left(\tfrac{3}{2}\right)^k = n\left(\frac{\left(\tfrac{3}{2}\right)^{1+\lg n} - 1}{\tfrac{3}{2} - 1}\right) = 3n^{\lg 3} - 2n$$



Fast Integer Division Too (!)

Integer division. Given two n-bit (or less) integers s and t, compute quotient $q = \lfloor s / t \rfloor$ and remainder $r = s \bmod t$ (such that s=qt+r).

Fact. Complexity of integer division is (almost) same as integer multiplication.

To compute quotient q: $\quad x_{i+1} = 2x_i - t\,x_i^2$ using fast multiplication

- Approximate $x = 1/t$ using Newton's method:
- After i=$\log n$ iterations, either $q = \lfloor s\,x_i \rfloor$ or $q = \lceil s\,x_i \rceil$.
  - If $\lfloor s\,x \rfloor$ t > s then $q = \lceil s\,x \rceil$ (1 multiplication)
  - Otherwise $q = \lfloor s\,x \rfloor$
  - r=s-qt (1 multiplication)

- **Total**: $O(\log n)$ multiplications and subtractions

Toom-3 Generalization

Split into 3 parts

$$a = 2^{2n/3} \cdot a_2 + 2^{\frac{n}{3}} \cdot a_1 + a_0$$
$$b = 2^{2n/3} \cdot b_2 + 2^{\frac{n}{3}} \cdot b_1 + b_0$$

Requires: 5 multiplications of n/3 bit numbers and O(1) additions, shifts

$$T(n) = 5 \cdot T\left(\frac{n}{3}\right) + O(n) \Rightarrow T(n) \in O\left(n^{\log_3 5}\right)$$

≈ 1.465

Toom-Cook Generalization (split into k parts):

$$a = 2^{\frac{n(k-1)}{k}} \cdot a_{k-1} + \cdots + 2^{\frac{n}{k}} \cdot a_1 + a_0$$

$$b = 2^{\frac{n(k-1)}{k}} \cdot a_k + \cdots + 2^{\frac{n}{k}} \cdot a_1 + a_0$$

$$T_k(n) = (2k-1) \cdot T_k\left(\frac{n}{k}\right) + O(n) \Rightarrow T_k(n) \in O\left(n^{\log_k(2k-1)}\right)$$

$\forall \varepsilon > 0 \exists k$ s.t $T_k(n) \in O(n^{1+\varepsilon})$

$\lim_{k\to\infty} (\log_k(2k-1)) = 1$

Toom-3 Generalization

Split into 3 parts

$$a = 2^{2n/3} \cdot a_2 + 2^{\frac{n}{3}} \cdot a_1 + a_0$$
$$b = 2^{2n/3} \cdot b_2 + 2^{\frac{n}{3}} \cdot b_1 + b_0$$

Requires: 5 multiplications of n/3 bit numbers and O(1) additions, shifts

$$T(n) = 5 \cdot T\left(\frac{n}{3}\right) + O(n) \Rightarrow T(n) \in O\left(n^{\log_3 5}\right)$$

≈ 1.465

Schönhage–Strassen algorithm

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

Only used for really big numbers: $a > 2^{2^{15}}$

State of the Art: $O(n \ \log n \ g(n))$ for increasing small

$g(n) \ll \log\log n$

Matrix Multiplication

Dot Product

Dot product. Given two length n vectors a and b, compute $c = a \cdot b$.
Grade-school. $\Theta(n)$ arithmetic operations.

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

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

Remark. Grade-school dot product algorithm is optimal.

Matrix Multiplication

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

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 optimal?

Block Matrix Multiplication

C_{11} $\quad A_{11}$ $\quad A_{12}$ $\quad B_{11}$

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

B_{21}

$$C_{11} = A_{11} \times B_{11} + A_{12} \times B_{21}$$

$$= \begin{bmatrix} 0 & 1 \\ 4 & 5 \end{bmatrix} \times \begin{bmatrix} 16 & 17 \\ 20 & 21 \end{bmatrix} + \begin{bmatrix} 2 & 3 \\ 6 & 7 \end{bmatrix} \times \begin{bmatrix} 24 & 25 \\ 28 & 29 \end{bmatrix}$$

$$= \begin{bmatrix} 152 & 158 \\ 504 & 526 \end{bmatrix}$$

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.

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

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

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

Fast Matrix Multiplication

Key idea. multiply 2-by-2 blocks with only 7 multiplications.

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

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

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

- 7 multiplications.
- $18 = 8 + 10$ additions and subtractions.

Fast Matrix Multiplication

To multiply two n-by-n matrices A and B: [Strassen 1969]

- Divide: partition A and B into $\frac{1}{2}n$-by-$\frac{1}{2}n$ blocks.
- Compute: 14 $\frac{1}{2}n$-by-$\frac{1}{2}n$ matrices via 10 matrix additions.
- Conquer: multiply 7 pairs of $\frac{1}{2}n$-by-$\frac{1}{2}n$ matrices, recursively.
- Combine: 7 products into 4 terms using 8 matrix additions.

Analysis.

- $T(n)$ = # arithmetic operations.

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

- Apply Master Theorem (a=7,b=2,c=2)
  - $\left(\frac{a}{b^c}\right) = \frac{7}{4} > 1 \quad \Longrightarrow T(n) = \Theta\left(n^{\log_b a}\right) = \Theta\left(n^{\log_2 7}\right) = \Theta(n^{2.81})$

Fast Matrix Multiplication: Practice

Implementation issues.

- Sparsity.
- Caching effects.
- Numerical stability.
- Odd matrix dimensions.
- Crossover to classical algorithm around $n = 128$.

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

- Apple reports 8x speedup on G4 Velocity Engine when $n \approx 2{,}500$.
- Range of instances where it's useful is a subject of controversy.

Remark. Can "Strassenize" $Ax = b$, determinant, eigenvalues, SVD, ….

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. Two 3-by-3 matrices with 21 scalar multiplications?
A. Also impossible. $\Theta(n^{\log_3 21}) = O(n^{2.77})$

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. $O(n^{2.521801})$

Fast Matrix Multiplication: Theory

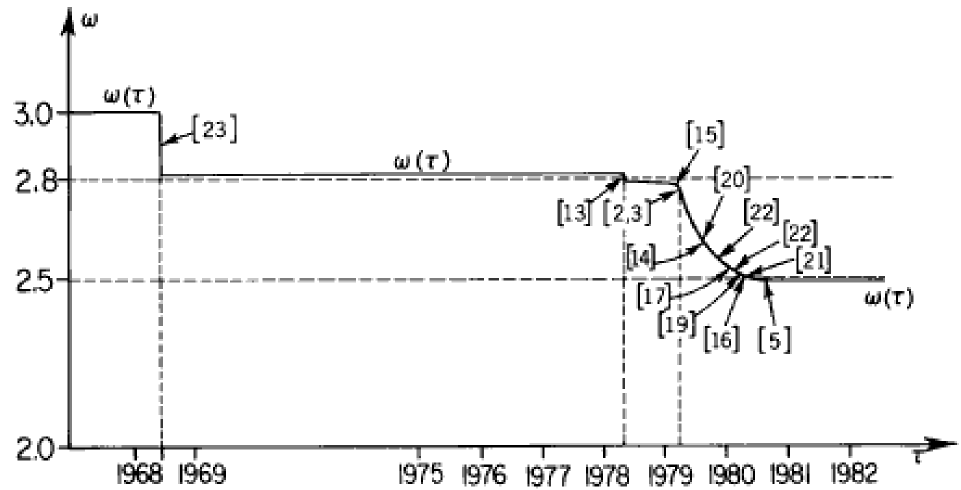


FIG. 1. *$\omega(t)$ is the best exponent announced by time τ.*

Best known. $O(n^{2.376})$ [Coppersmith-Winograd, 1987]

Conjecture. $O(n^{2+\varepsilon})$ for any $\varepsilon > 0$.

Caveat. Theoretical improvements to Strassen are progressively less practical.

Fast Matrix Multiplication: Theory

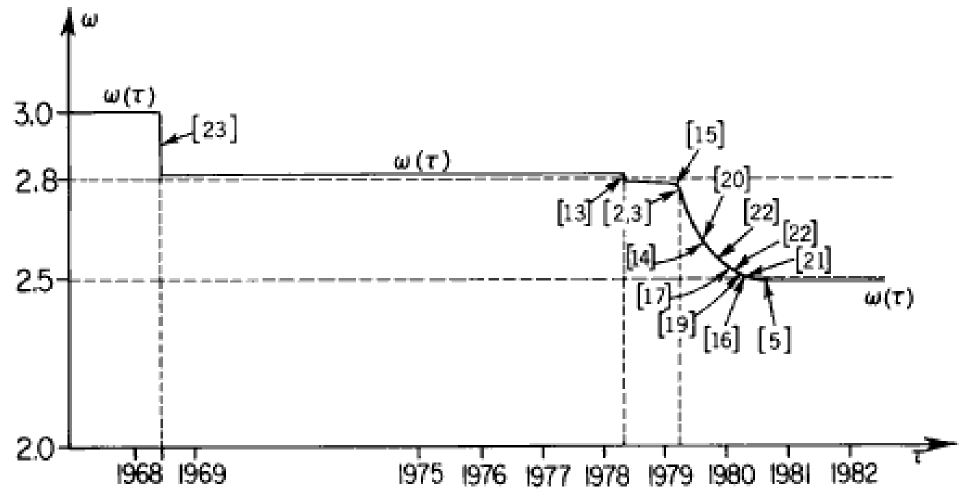


FIG. 1. $\omega(t)$ *is the best exponent announced by time* τ.

Best known. $O(n^{2.373})$ [Williams, 2014]

Conjecture. $O(n^{2+\varepsilon})$ for any $\varepsilon > 0$.

Caveat. Theoretical improvements to Strassen are progressively less practical.

Fast Matrix Multiplication: Theory

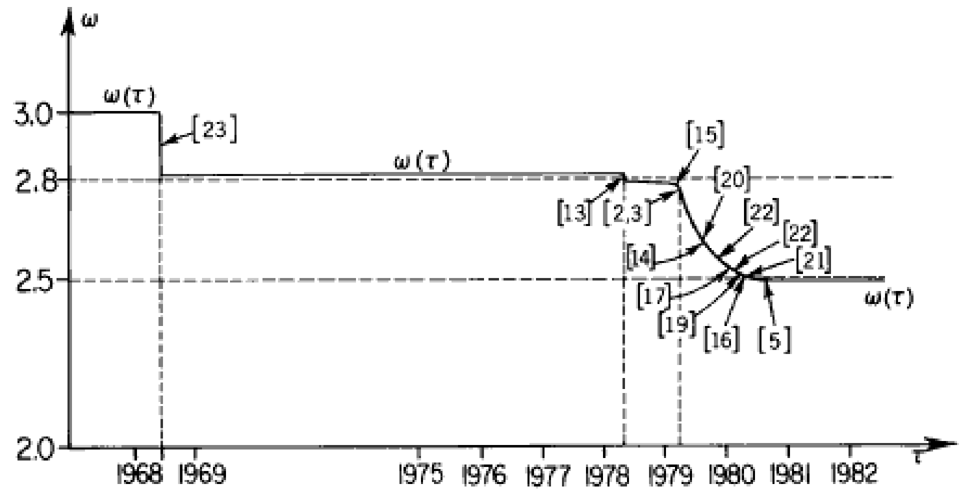


FIG. 1. $\omega(t)$ *is the best exponent announced by time* τ.

Best known. $O(n^{2.3729})$ [Le Gall, 2014]

Conjecture. $O(n^{2+\varepsilon})$ for any $\varepsilon > 0$.

Caveat. Theoretical improvements to Strassen are progressively less practical.

Extra Slides