COMP 251

Algorithms & Data Structures (Winter 2021)

Algorithm Paradigms – Divide and Conquer 2

School of Computer Science
McGill University

Slides of (Comp321 ,2021), Langer (2014), slides by K. Wayne Snoeyink, Kleinberg & Tardos, 2005 & Cormen et al., 2009

Divide and Conquer – Arithmetic Operations

- Given 2 (binary) numbers, we want efficient algorithms to:
  - Add 2 numbers
  - **Multiply 2 numbers** (using divide-and-conquer!)

Divide and Conquer – Arithmetic Operations

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.

$$\begin{array}{r} 352 \\ +\ 964 \\ \hline 1316 \end{array}$$

$$\begin{array}{r} x[n] \\ y[n] \\ \hline sum[n+1] \end{array}$$

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

352 x[n]
× 964 y[n]
1408
2112 tmp[n][2n]
3168
339328 r[2n]

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

Theorem. [Karatsuba 1960] Conjecture is wrong.

Divide and Conquer – Arithmetic Operations

$\frac{n}{2}$ digits $\qquad$ $\frac{n}{2}$ digits

x_1	x_0
y_1	y_0

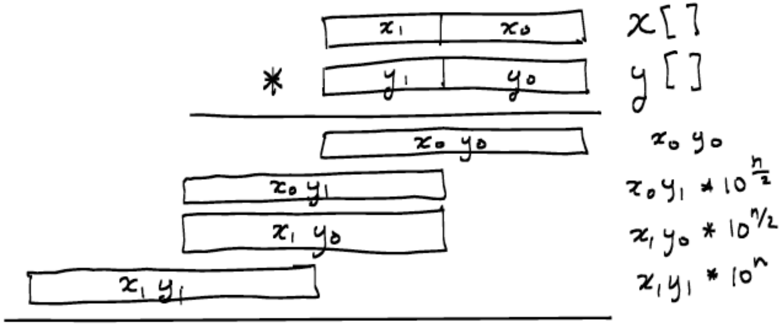
n digits

$$x = x_1 * 10^{n/2} + x_0$$
$$y = y_1 * 10^{n/2} + y_0$$

e.g.

$$3527 = 3500 + 27$$
$$= 35 \times 10^2 + 27$$

$$x * y$$
$$= (x_1 * 10^{\frac{n}{2}} + x_0) * (y_1 * 10^{\frac{n}{2}} + y_0)$$
$$= \underbrace{x_1 y_1}_{t(\frac{n}{2})} * 10^n + (\underbrace{x_0 y_1}_{t(\frac{n}{2})} + \underbrace{x_1 y_0}_{t(\frac{n}{2})}) * 10^{n/2} + \underbrace{x_0 y_0}_{t(\frac{n}{2})}$$



Note:

$* 10^{n/2}$ shifts left by $\frac{n}{2}$ positions

$* 10^n$ " " " n positions.

Divide and Conquer – Arithmetic Operations

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 ½n-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

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

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

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 – Arithmetic Operations

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:

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

$k = \log_2 4 = 2$

$n = O(n^{2-1}) \Rightarrow$ case 1

Multiplication. Given two n-bit integers a and b, compute $a \times b$.
Grade-school algorithm. $\Theta(n^2)$ bit operations.

Divide and Conquer – Karatsuba trick

$$x = \begin{array}{|c|c|}\hline x_1 & x_0 \\ \hline \end{array}$$
$$y = \begin{array}{|c|c|}\hline y_1 & y_0 \\ \hline \end{array}$$

$$x * y$$
$$= x_1 y_1 * 10^n + \underbrace{(x_0 y_1 + x_1 y_0)}_{(x_1 + x_0)(y_1 + y_0) - x_1 y_1 - x_0 y_0} * 10^{n/2} + x_0 y_0$$

1, 2, 3

Thus, $\boxed{t(n) = 3\,t\left(\frac{n}{2}\right) + c\,n.}$

Avengers Assemble In Final Battle Scene - AVENGERS: ENDGAME (2019). Taken from youtube

Divide and Conquer – 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

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

$$= 2^{2m}\, \underbrace{ac}_{1} + 2^m (\underbrace{ac}_{1} + \underbrace{bd}_{3} - \underbrace{(a-b)(c-d)}_{2}) + \underbrace{bd}_{3}$$

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

Divide and Conquer – Karatsuba trick

```
KARATSUBA-MULTIPLY(x, y, n)
_______________________________________
IF (n = 1)
   RETURN x × y.
ELSE
   m ← ⌈ n / 2 ⌉.
   a ← ⌊ x / 2^m ⌋;   b ← x mod 2^m.
   c ← ⌊ y / 2^m ⌋;   d ← y mod 2^m.
   e ← KARATSUBA-MULTIPLY(a, c, m).
   f ← KARATSUBA-MULTIPLY(b, d, m).
   g ← KARATSUBA-MULTIPLY(a – b, c – d, m).
   RETURN 2^(2m) e + 2^m (e + f – g) + f.
_______________________________________
```

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:

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

$k = \log_2 3$

$n = O(n^{\log_2 3 - (\log_2 3 - 1)})$

$\Rightarrow$ case 1

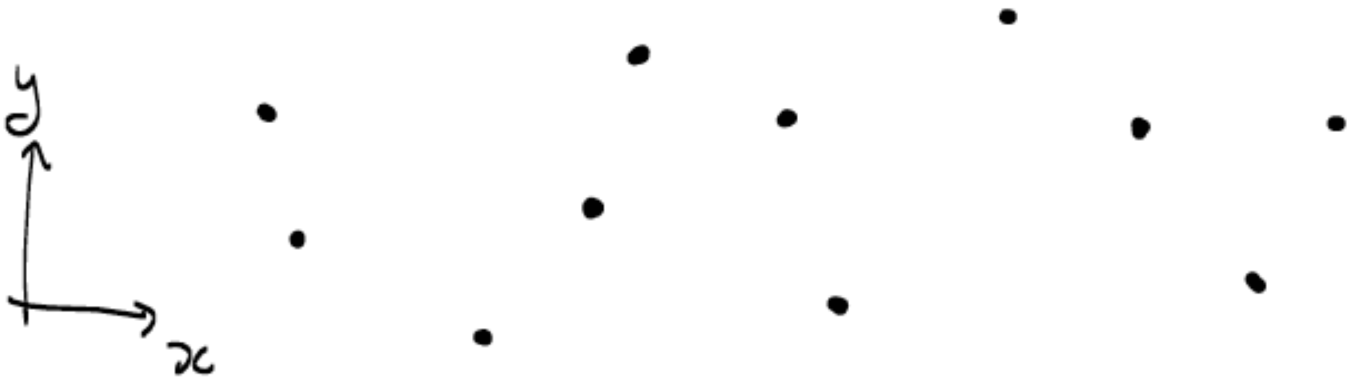
Divide and Conquer – Integer Multiplication

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

Divide and Conquer – Closest points

- Given n points in the plane, find the pair that is closest together.



- Applications in:
  - Computational Geometry.
    - Graphics, computer vision, geographic information systems, molecular modeling.

Divide and Conquer – Closest points

- Given n points in the plane, find the pair that is closest together.

Solution ("brute force"):

```
closest pair = null
δ = ∞
for each i = 1 to n
    for each j = i+1 to n
        if d(i,j) < δ {
            closestpair = (i,j)
            δ = d(i,j)
        }
return closestpair
```

$O(n^2)$

too slow!

$$d(i,j) \equiv \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

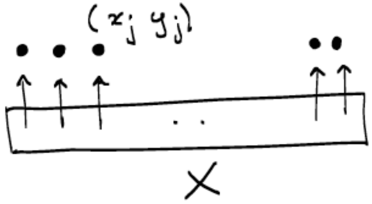
Divide and Conquer – Closest points

- 1-D Solution.
  - We first sort the points (merge sort) => O(n log n).
  - We'd walk through the sorted list, computing the distance from each point to the one that comes after it => O(n).
    - One of these distances must be the minimum one.
- 2-D Solution.
  - we could try sorting the points by their y-coordinate (or x-coordinate) and hoping that the two closest points were near one another in the order of this sorted list.
    - it is easy to construct examples in which they are very far apart
  - Mimic Merge sort.
    - Find the closest pair among the points in the "left half"
    - Find the closest pair among the points in the "right half"
      - Be careful with the distances that have not been considered.
        - One point is the left and one point in the right half.

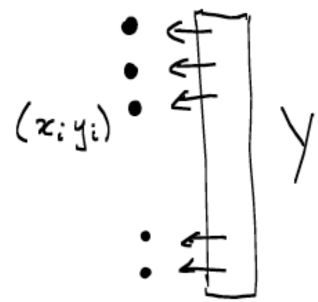
Divide and Conquer – Closest points

- 2-D Solution.

Solution for 2D (Shamos & Hoey 1970's)

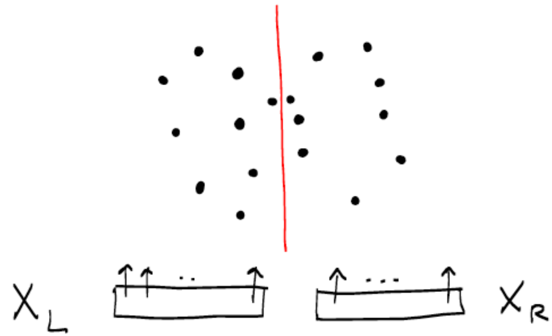


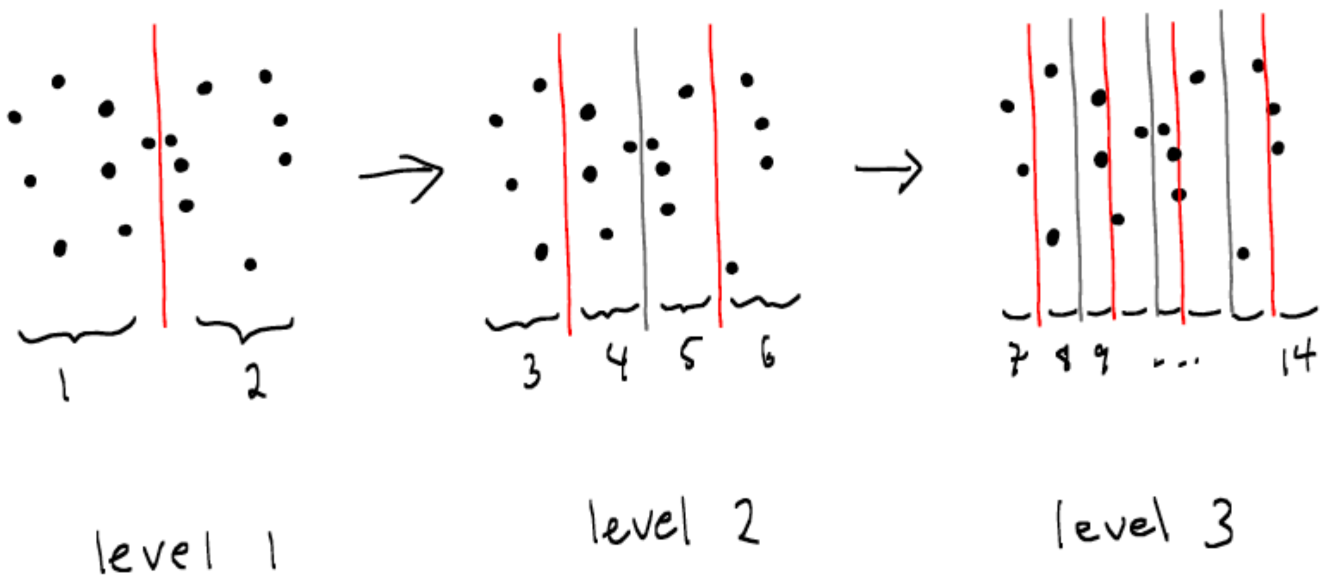
Begin by sorting points by x value, and sorting points by y value, giving two sorted arrays X and Y.



Divide and Conquer – Closest points

- Partition X into two sets:
  - X_L has the n/2 smallest x values ('left')
  - X_R has the n/2 largest x values ('right')





Divide and Conquer – Closest points

Find closest pair (X) { → $t(n)$

  if $|X| \leq 3$ then compute closest pair by brute force and return it → c_1

  else {

    Compute X_L X_R → c_2

    Find closest pair (X_L) → $t(n/2)$

    Find closest pair (X_R) → $t(n/2)$

    Find the closest pair such that one point is in X_L and the other point is in X_R. → ?

    Return the closest of the three pairs. → c_3

  }

}

$$t(n) = 2t\left(\frac{n}{2}\right) + ? + c$$

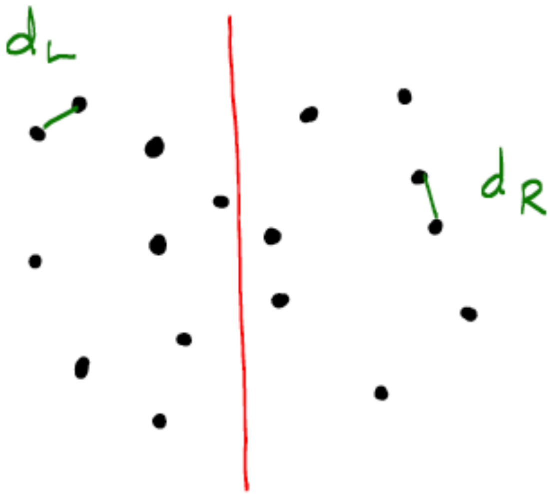
Divide and Conquer – Closest points

$$t(n) = 2t\left(\frac{n}{2}\right) + ? + c$$

- X_L and X_R each have n/2 points. Thus there are n/2 * n/2 pairs of points such that one is in X_L and the other in X_R.
  - Finding the pair with minimum distance using "brute force" would take $O(n^2)$, which is too slow.
  - Can we solve this problem in time $O(n)$, instead on $O(n^2)$?

Divide and Conquer – Closest points

- Let the closest pair in X_L have distance d_L.
- Let the closest pair in X_R have distance d_R.



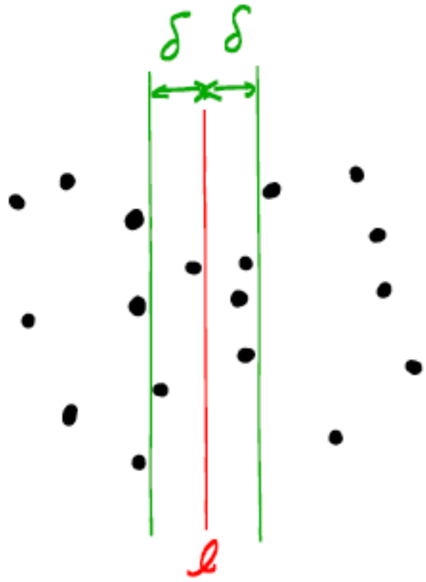
These are the pairs returned by the two recursive calls

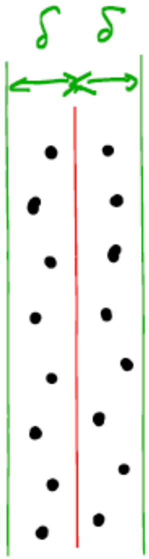
findClosestPair(X_L)
findClosestPair(X_R)

Let $\delta = \min(d_L, d_R)$

Divide and Conquer – Closest points

- Observe that to find the closest pair with one point in X_L and the other point in X_R, we only need to consider points that are a distance δ from the line ℓ that separates L and R.

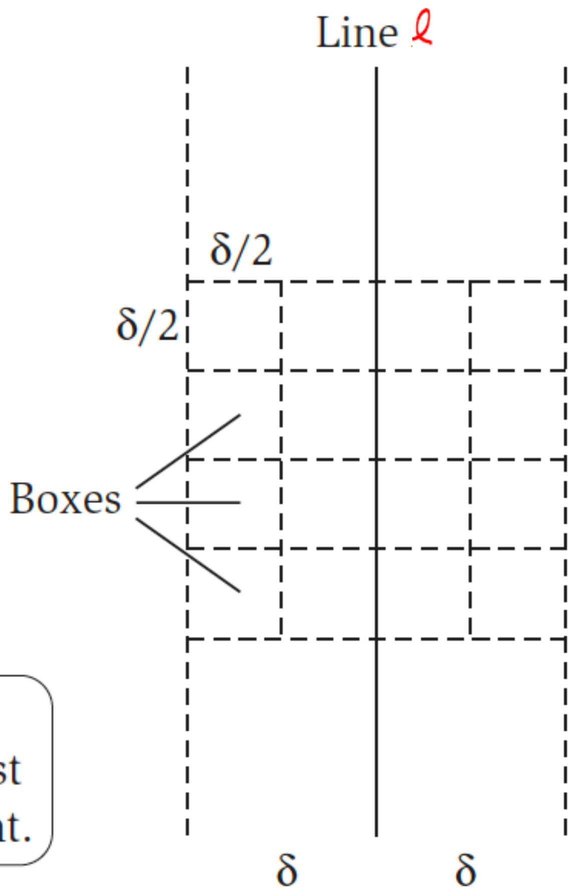




The observation does not necessarily reduce the number of points we Need to consider

Divide and Conquer – Closest points

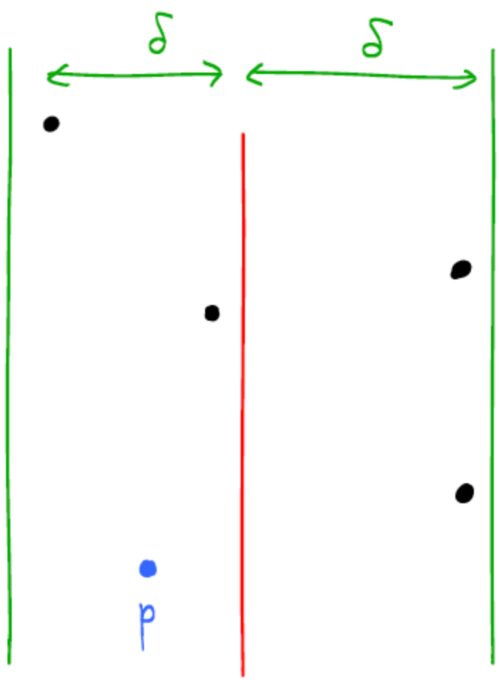
- Consider the subset of the plane consisting of all points within distance δ of ℓ. We can partition this subset into boxes (squares with horizontal and vertical sides of length $\delta/2$). One row of this subset will consist of four boxes whose horizontal sides have the same *y*-coordinates.



Each box can contain at most one input point.

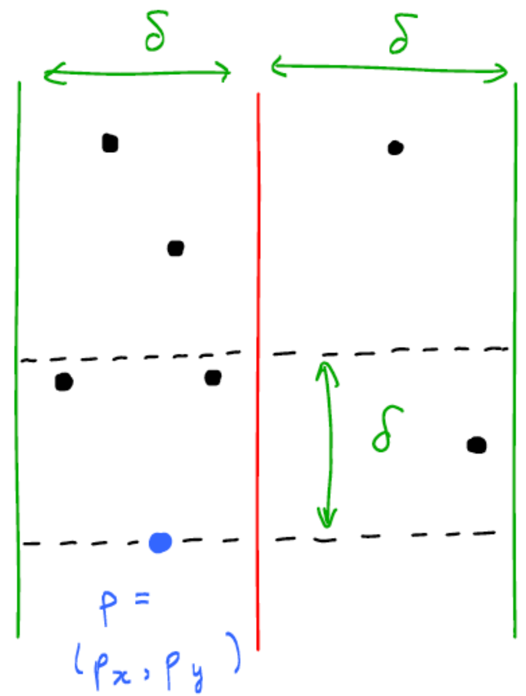
Divide and Conquer – Closest points

- Consider a point p that lies between the two green lines.
  - Is there another point between the green lines that has a *y* value greater than that of p **and** is at a distance less than δ from p?



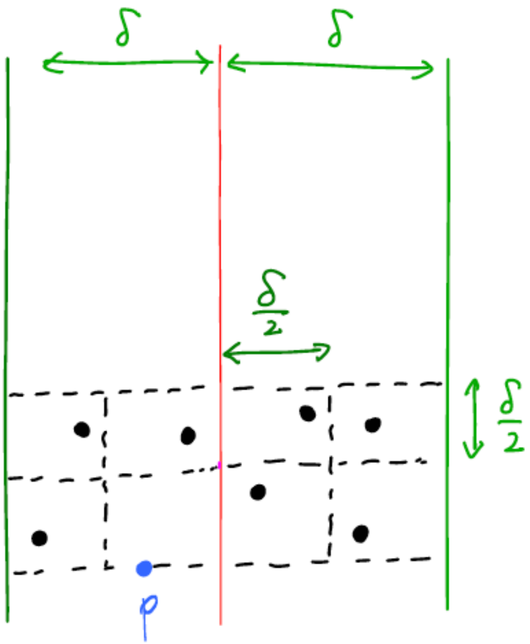
It is sufficient to check those points whose y values are between p_y and $p_y + \delta$

Q: How many points do we need to check in the worst case?



Divide and Conquer – Closest points

- **Q: How many points do we need to check in the worst case?**
- **A: At most 7.**
  - Remember that square cells of width $\frac{\delta}{2}$ can contain at most 1 point.
  - Remember that we also sorted the points by their *y* coordinate



Find the closest pair such that one point is in X_L and the other point is in X_R. {

Find all points that lie between the green lines.

Starting from point with min y value, examine the distance to next 7 points (sorted by y). If we find a pair with distance $< \delta$, make it the new closest pair & update δ.

}

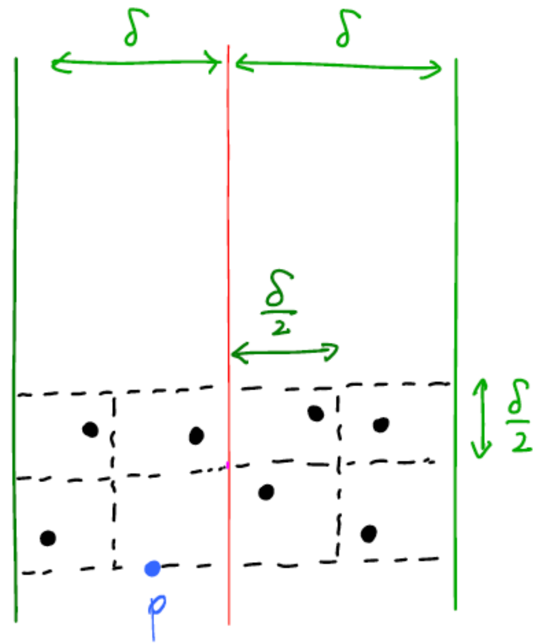
Divide and Conquer – Closest points

- **Q: How many points do we need to check in the worst case?**
- **A: At most 7.**

```
middle = empty list

for i = 1 to n                                 // find points between
    if Y[i] is between green lines             // green lines: O(n)
        middle.add( Y[i] )

for i = 1 to middle.size {                     // O(n)
    for j = 1 to 7 {   // ignore out of bounds error
        tmp = d( middle[i], middle[i+j] )
        if tmp < δ {
            closest pair = ( middle[i], middle[i+j] )
            δ = tmp
        }
    }
}
```



Divide and Conquer – Closest points

Find closest pair (X) { → $t(n)$

if $|X| \le 3$ then compute closest pair by brute force and return it → c_1

else {

Compute X_L X_R → c_2

Find closest pair (X_L) → $t(n/2)$

Find closest pair (X_R) → $t(n/2)$

Find the closest pair such that one point is in X_L and the other point is in X_R. → $c_4 n$

Return the closest of the three pairs. → c_3

}

}

$$t(n) = 2t\left(\frac{n}{2}\right) + c_4 n + c$$

Divide and Conquer – Closest points

Find closest pair (X) { // $\longrightarrow$ $t(n)$

if $|X| \le 3$ then compute closest pair by brute force and return it $\longrightarrow$ c_1

else {

Compute X_L X_R $\longrightarrow$ c_2

Find closest pair (X_L) $\longrightarrow$ $t(n/2)$

Find closest pair (X_R) $\longrightarrow$ $t(n/2)$

Find the closest pair such that one point is in X_L and the other point is in X_R. $\longrightarrow$ c_4n

Return the closest of the three pairs. $\longrightarrow$ c_3

}

}

$$t(n) = 2t\left(\frac{n}{2}\right) + c_4 n + c$$ $\longrightarrow$ Same than merge sort O(nlogn)

Matrix multiplication – If time allows

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

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

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

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

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

SQUARE-MATRIX-MULTIPLY(A, B)

```
n = A.rows
let C be a new n × n matrix
for i = 1 to n
    for j = 1 to n
        c_ij = 0
        for k = 1 to n
            c_ij = c_ij + a_ik · b_kj
return C
```

Matrix multiplication – divide and conquer

Suppose that we partition each of A, B, and C into four $n/2 \times n/2$ matrices

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \quad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}, \quad C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix},$$

so that we rewrite the equation $C = A \cdot B$ as

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \cdot \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}.$$

Equation (4.10) corresponds to the four equations

$$\begin{aligned}
C_{11} &= A_{11} \cdot B_{11} + A_{12} \cdot B_{21}, \\
C_{12} &= A_{11} \cdot B_{12} + A_{12} \cdot B_{22}, \\
C_{21} &= A_{21} \cdot B_{11} + A_{22} \cdot B_{21}, \\
C_{22} &= A_{21} \cdot B_{12} + A_{22} \cdot B_{22}.
\end{aligned}$$

C_{11} A_{11} A_{12} 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_{11}

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

Matrix multiplication – divide and conquer

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

- Divide: partition A and B into ½n-by-½n blocks.
- Conquer: multiply 8 pairs of ½n-by-½n 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.

$k = \log_2 8 = 3$

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

Matrix multiplication – Strassen's trick

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

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

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

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

Pf.
$$\begin{aligned}
C_{12} &= P_1 + P_2 \\
&= A_{11} \times (B_{12} - B_{22}) + (A_{11} + A_{12}) \times B_{22} \\
&= A_{11} \times B_{12} + A_{12} \times B_{22}. \quad \checkmark
\end{aligned}$$

Matrix multiplication – Strassen's trick

STRASSEN (n, A, B)

---

assume n is a power of 2

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

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

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

$P_1 \leftarrow$ 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.

Matrix multiplication – Strassen's trick

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:

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

Matrix multiplication – Strassen's trick

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

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	$O(n^{2.3727})$
?	?	$O(n^{2+\varepsilon})$

Outline

- Complete Search
- Divide and Conquer.
  - Introduction.
  - Examples.
- Dynamic Programming.
- Greedy.