Divide and Conquer Example

Finding the closest pair of points — an example where the "conquer" step is more complicated.

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

$d(p,q)$ = distance between p and q

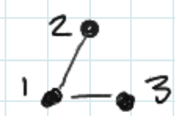
$= \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}$

Note: we can compare $d(p,q)$ to $d(r,s)$ without $\sqrt{\ }$.

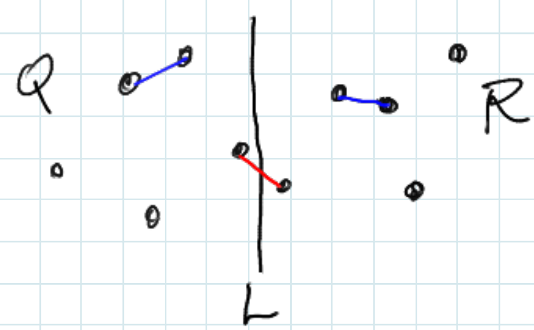
Brute force: try all pairs $O(n^2)$

In 1D (points on a line): sort and compare consecutive pairs. $O(n \log n)$

Note that this does not work in 2D.



Divide and Conquer (after sorting by x coord.)



- divide points into left half, Q, right half, R, dividing line L
- recursively find closest pair in Q, closest pair in R
- combine

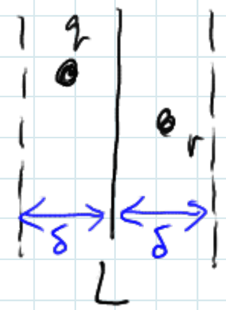
To combine, we need to find close pairs crossing L

This is the tricky part.

Let δ = min distance of $\begin{cases} \text{closest pair in } Q \\ \text{closest pair in } R \end{cases}$

Must check pairs $q \in Q$, $r \in R$ with $d(q,r) < \delta$

<u>Claim</u> Such points satisfy $d(q, L) < \delta$, $d(r, L) < \delta$



Pf. Otherwise horizontal distance $\geq \delta$ so distance $\geq \delta$.

Let S = points in this vertical strip of width 2δ

We can restrict our search to S (might be all n points!)

This problem seems more 1-dimensional

Sort S by y-coordinate

Note: <u>do not</u> do this in each recursive call. Do it once for all points $O(n \log n)$ and extract the sorted sublist for any S in linear time.

needed for Prog. Assign

This is like "unmerging"

points sorted by y

p_1 p_2 p_3 $\cdots$ p_j p_i $\cdots$ p_n

points of S circled

p_2 p_3 $\cdots$ p_i found by scanning list

Note: each recursive call needs to know its points sorted by y-coordinate.

Overall structure of the algorithm:

$X \leftarrow$ sort points by x-coord.

$Y \leftarrow$ sort points by y-coord

Closest(X, Y) – returns distance between closest pair of points

Closest(X, Y)

$L \leftarrow$ dividing line (middle of X)

"unmerge" to get X_Q, X_R, Y_Q, Y_R

sorted lists for left side (Q), right side (R)

$\delta_Q \leftarrow$ Closest (X_Q, Y_Q)

$\delta_R \leftarrow$ Closest (X_R, Y_R)

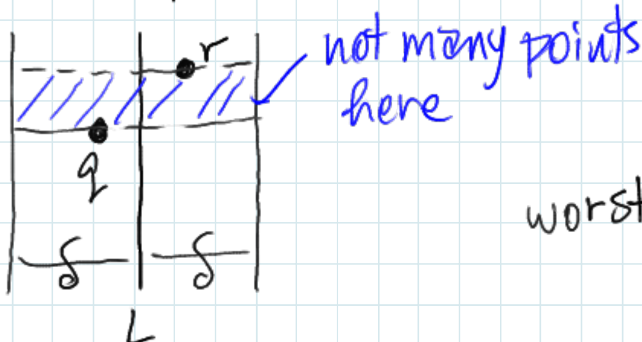
$\delta \leftarrow \min\{\delta_Q, \delta_R\}$

find set S as above

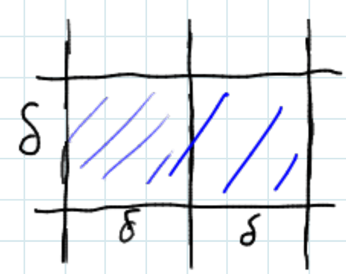
$Y_S \leftarrow S$ sorted by y-coord (extract from Y)

Finally, what do we do with S, Y_S ?

Our hope: if $q, r \in S$, $q \in Q$, $r \in R$ and $d(q,r) < \delta$ then q and r are near each other in the sorted S

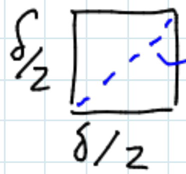


worst case



Claim At most 8 points here.

why? We have 8 squares $\frac{\delta}{2} \times \frac{\delta}{2}$ and each square has ≤ 1 point.



$\frac{\sqrt{2}\,\delta}{2} = \frac{\delta}{\sqrt{2}} < \delta$

(in fact can prove 6 instead of 8)

Thus If $q, r \in S$ $q \in Q, r \in R$ $d(q,r) < \delta$
then q and r are at most 8 positions apart in sorted S

$s_1 \cdots\ s_i\ \underbrace{s_{i+1} \cdots\ s_{i+7}}_{\text{just need to look here for } r} \cdots$

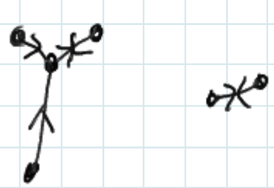
$q = s_i$

so $7n$ comparisons

Wrapping up: preliminary sort by x and by y
then $T(n) = 2\,T(\frac{n}{2}) + c \cdot n$
so $T(n) \in \Theta(n \log n)$

This alg. was due to Preparata & Shamos in the early days of computational geometry (70's and 80's)

In fact, one can do much more in $O(n \log n)$ — find closest neighbours of all points



EX. This graph has no cycles, no crossing edges, max. degree 6

Divide and Conquer – Multiplying Large Numbers

School method

$$
\begin{array}{r}
981 \\
\times\ 1234 \\
\hline
3924 \\
2943 \\
1962 \\
981 \\
\hline
1210554
\end{array}
$$

This takes $O(n^2)$ time to multiply two n digit numbers

Divide and conquer

easiest when both numbers have same no. digits so pad 981 to 0981

09|81 × 12|34

	shift	
09 × 12	4	108_ _ _ _
09 × 34	2	306_ _
81 × 12	2	972_ _
81 × 34	0	2754
		1210554

Compute each of these 4 products recursively

e.g. 0|9 × 1|2

	shift	
0 × 1	2	0
0 × 2	1	0
9 × 1	1	9
9 × 2	0	18
		208

$$T(n) = 4T\left(\frac{n}{2}\right) + \underbrace{O(n)}_{\text{time for shifts, addition}}$$

Apply Master Method

$T(n) = a\,T(\frac{n}{b}) + c \cdot n^k$

$a=4 \quad b=2 \quad k=1$ compare a to b^k

$4 \qquad 2$

thus $a > b^k$

$T(n) \in \Theta(n^{\log_b a}) = \Theta(n^2)$

No gain so far!

We can avoid one of the four multiplications:

$\underset{w}{9} | \underset{x}{81} \times \underset{y}{12} | \underset{z}{34}$

$(10^2 w + x) \times (10^2 y + z)$

$= 10^4 wy + 10^2(wz + xy) + xz$

We don't need wz and xy. We just need $wz + xy$

Consider $\underbrace{(w+x)\times(y+z)}_{\text{Let } r =} = wy + (wz+xy) + xz$

we know these

$p = wy \qquad 09 \times 12 = 108$

$q = xz \qquad 81 \times 34 = 2754$

$r = (w+x)\times(y+z) \qquad 90 \times 46 = 4140$

Our answer: $10^4 p + 10^2 (r - p - q) + q$

$$\begin{array}{r} 1080000 \\ 127800 \\ 2754 \\ \hline 1210554 \end{array}$$ ✓

Now we get

$$T(n) = 3T(\tfrac{n}{2}) + O(n)$$

$a=3 \quad b=2 \quad k=1 \qquad a=3 > b^k=2$

$$T(n) = \Theta(n^{\log_b a}) = \Theta(n^{\log_2 3})$$

$\log_2 3 \sim 1.585$

This alg. is due to Karatsuba and Ofman 1962

Practical issues

1. What about numbers of odd length? } pad?
   numbers of different length?
2. What base should we use?
   Above we used base 10. Use largest base s.t. two "digits" can be multiplied directly.
   Actually, can be better to stop recursion before this and revert to school method (4 mult., fewer additions)
3. When is this alg. useful?
   - for small n the overhead is too high
   - this alg. beats the basic $O(n^2)$ one for numbers of 1000 digits and up [Brassard & Bratley]
   - for large n there are asymptotically better methods (even worse overhead) $O(n \log n \log \log n)$
     Schönhage & Strassen

Multiplying matrices (similar idea)
multiply two $n \times n$ matrices
standard method is $O(n^3)$
Divide and conquer
divide into submatrices of size $\frac{n}{2}$

$$\begin{bmatrix} C_{11} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{bmatrix} = \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix} \times \begin{bmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{bmatrix}$$

$C_{1,1} = A_{1,1} \times B_{1,1} + A_{1,2} \times B_{2,1}$ – two mult. of $\frac{n}{2}$ size submatrices
etc. – one addition $O(n^2)$

$T(n) = 8T(\frac{n}{2}) + O(n^2)$
$a=8$ $b=2$ $k=2$ $a=8 > b^k = 4$
$T(n) \in \Theta(n^{\log_b a}) = \Theta(n^3)$
So far, no progress.

Strassen's alg.
- like idea for integer multiplication
- get by with 7 subproblems instead of 8 (tricky!)

$T(n) = 7T(\frac{n}{2}) + O(n^2)$
$a=7$ $b=2$ $k=1$ $a=7 > b^k = 4$
$T(n) \in \Theta(n^{\log_b a}) = \Theta(n^{\log_2 7})$ $\log 7 \sim 2.808$
Again, there are more sophisticated methods that do better, but with greater overhead.

Here is Strassen's tricky way of using 7 subproblems.

Suppose $C = A \times B$, all $n \times n$ matrices

$$\begin{bmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{bmatrix} = \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix} \begin{bmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{bmatrix}$$

each block is $n/2 \times n/2$

Define:

$M_1 = (A_{1,1} + A_{2,2})(B_{1,1} + B_{2,2})$

$M_2 = (A_{2,1} + A_{2,2}) B_{1,1}$

$M_3 = A_{1,1} (B_{1,2} - B_{2,2})$

$M_4 = A_{2,2} (B_{2,1} - B_{1,1})$

$M_5 = (A_{1,1} + A_{1,2}) B_{2,2}$

$M_6 = (A_{2,1} - A_{1,1})(B_{1,1} + B_{1,2})$

$M_7 = (A_{1,2} - A_{2,2})(B_{2,1} + B_{2,2})$

Then check that

$C_{1,1} = M_1 + M_4 - M_5 + M_7$

$C_{1,2} = M_3 + M_5$

$C_{2,1} = M_2 + M_4$

$C_{2,2} = M_1 - M_2 + M_3 + M_6$