Recap:

Quick-Sort

- $\Theta(n^2)$ worst case (worse than MergeSort!)
  $\Theta(n \log n)$ best/average case
- Unstable
- In place
- The constant in $\Theta(\cdot)$ is quite low, so works very well in practice!
- Based on Divide & Conquer
- Sort of "MergeSort in Reverse"

1. Integer Multiplication (Karatsuba)

Another example of Divide & Conquer

Input: two n-bit integers $a = a_1 a_2 \dots a_n$ and $b = b_1 b_2 \cdots b_n$,
$a, b = O(2^n)$

Goal: Compute (the binary representation of) $a \times b$

Addition:
$$\begin{array}{r} 1100 \\ 1011 \\ \hline 10111 \end{array}$$
$\Theta(n)$ runtime

Naive Alg.: $a \cdot b = \underbrace{a + a + a + \dots + a}_{b \text{ of them}}$

$\Theta(2^n \cdot n)$ runtime

Elem. School Alg.:
$$\begin{array}{r} 1100 \\ \times\ 1011 \\ \hline 1100 \\ 1100 \\ 0000 \\ 1100 \\ \hline \underbrace{10000100}_{\text{up to } 2n} \end{array}$$
n

$\Theta(n^2)$ runtime

Let's use divide and conquer!

e.g. 73×52

$= (70+3) \times (50+2)$

$= (7\times5)\times10^2 + (3\times5)\times10 + (7\times2)\times10 + 3\times2$

$= 3500+150+140+6$

$= 3796$

$$a = (a_1 \ldots a_{n/2})\cdot 2^{n/2} + (a_{n/2+1} \ldots a_n) = a_h \cdot 2^{n/2} + a_\ell$$

$$b = (b_1 \ldots b_{n/2})\cdot 2^{n/2} + (b_{n/2+1} \ldots b_n) = b_h \cdot 2^{n/2} + b_\ell$$

e.g. $\underbrace{1011}_{a_h}\underbrace{0100}_{a_\ell} = 1011\cdot 2^4 + 0100$

Now we have

$$a\cdot b = (a_h\cdot 2^{n/2} + a_\ell)(b_h \cdot 2^{n/2} + b_\ell)$$

$$= a_h\cdot b_h \cdot 2^n + (a_h\cdot b_\ell + a_\ell \cdot b_h)\cdot 2^{n/2} + a_\ell\cdot b_\ell \qquad (*)$$

Mult(a,b):

1. Compute Mult(a_h, b_h), Mult(a_h, b_ℓ), Mult(a_ℓ, b_h), Mult(a_ℓ, b_ℓ)
2. Return $(*)$

Correctness: follows (*)

Runtime: $T(n) = 4 \cdot T(\frac{n}{2}) + \Theta(n)$

$Q = n^{\log_b a} = n^2$

$\frac{\Theta(n)}{n^2} = O(n^{-1})$ Case 1!

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

Idea: $\boxed{a_h \cdot b_h} \, 2^n + (\boxed{a_h \cdot b_\ell + a_\ell \cdot b_h}) \cdot 2^{n/2} + \boxed{a_\ell \cdot b_\ell}$

Compute:

(1) $\boxed{a_h \cdot b_h}$

(2) $\boxed{a_\ell \cdot b_\ell}$

(3) $(a_h + a_\ell)(b_h + b_\ell) = a_h b_h + \boxed{a_h \cdot b_\ell + a_\ell b_h} + a_\ell b_\ell$

Then: $a_h \cdot b_\ell + a_\ell \cdot b_h = (3) - (1) - (2)$

4 Multiplications $\to$ 3 Multiplications

$T(n) = 3 \cdot T(\frac{n}{2}) + \Theta(n)$

$Q = n^{\log_2 3}$

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

Case 1!

$T(n) = \Theta(n^{\log_2 3}) = \Theta(n^{1.585})$

Integer Multiplications:

- Karatsuba (1962): $\Theta(n^{\log_2 3}) = \Theta(n^{1.585})$
- Toom-Cook (1963): $\Theta(n^{\log_3 5}) = \Theta(n^{1.465})$
- Schönhage-Strassen (1971): $\Theta(n \log n \log\log n)$
- Fürer (2007): $\Theta(n \log n \cdot 2^{\Theta(\log^* n)})$

$\log^* n$: Iterated logarithm

$\underbrace{\log \log \log \cdots \log}_{\text{the \# of these}} n < 1$

- Harvey-Hoeven (2021): $\Theta(n \log n)$

Takeaway: To optimize $T(n) = a \cdot T(\frac{n}{b}) + f(n)$

1. $a \downarrow$
2. $b \uparrow$
3. $f(n) \downarrow$

2. Closest pair of points

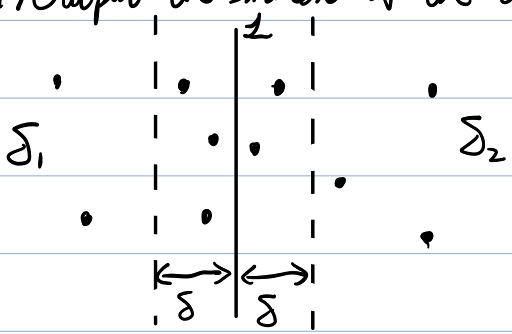
Input: n points on a plane $(x_1, y_1)(x_2, y_2) \dots (x_n, y_n)$

Goal: find two points that are closest to each other

- Brute force: check all pair of points. $\#\text{pairs} = \binom{n}{2} = \frac{n(n-1)}{2} = \Theta(n^2)$
- If all points are on a line: easy in $\Theta(n \log n)$ (sort & look at adjacent points)
- For simplicity, assume all x_i's, y_i's are distinct

Divide and Conquer!

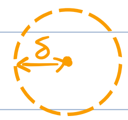
1. Divide points into two halves by their x coordinate (draw a vertical line)
2. Conquer: Recurse on both halves
3. Combine: Find closest pairs across the line (Check the points "near the border")
4. Output the smallest of the three



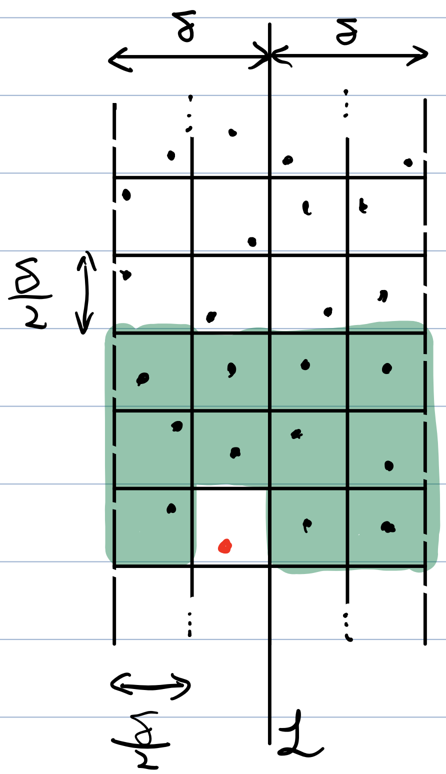
$\delta = \min(\delta_1, \delta_2)$

But Step 3 can be $\Theta(n^2)$!

Observation: only need to consider points that are closer than $\delta = \min(\delta_1, \delta_2)$



So for each point, only need to consider the ones in its δ-neighborhood.



1. Sort the points by their y coordinate
2. Overlay a $\frac{\delta}{2} \times \frac{\delta}{2}$ grid
3. Process the points according to their y coordinate

Claims:

1. At most 1 point per cell
2. Only need to look at points at most two rows away $\Rightarrow$ at most 11 cells! (with some careful argument, this can be 6)

Compute "cross-border" closest pair:

1. Take points within δ of L
2. Sort these points by y coordinate $p_1, p_2, \ldots, p_k$
3. Check the distance between each point p_i with the 11 points after it in the list

   For $i = 1$ to k:

   For $j = i+1$ to $i+11$:

   Check $dist(p_i, p_j)$, and update min if needed.

4. Output the closest pair found

Correctness: ✓

Runtime:

$$T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n \log n) \Rightarrow T(n) = \Theta(n \log^2 n)$$

Optimization:

Notice we repeated sort every time!

Sort once at the very beginning:

$$T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n) \Rightarrow T(n) = \Theta(n \log n)$$

$$(f(n) \downarrow)$$