Scientific Computing

Feb 17, 2025

Announcements

→ HW 2 due tonight

Last time:

Merge sort

Coding demo

Compared timings

Today

→ Divide and Conquer

Office Hours:

Mon + Fri

9:30am - 10:30am

Cudahy 307

A few overall notes:

* We are splitting the <u>input</u> in half, not the search space.

* These algorithms are not obvious! Many times there isn't one.

* If there is, it's usually faster than brute force - the recombining function is always the hard part!

Ex #2 (easy) – The simplest divide-and-conquer algo. is "binary search".

* Guess the number

50 ↓
25 ↓
13 ↑
19 ↓
17 ✓

Ex #2 – The simplest divide-and-conquer algo. is "binary search".

* Guess the number

In binary search, you just throw away half of your input each time.

n = # of elements

List containment: $O(n)$

Set containment: $O(\log(n))$

Recurrence: $T(n) = T(n/2) + 1$

Solution: $T(n) = O(\log(n))$

Ex #3 - Counting Inversions (medium)

Consider a list of distinct #s.

$L = 3 \quad 19 \quad -7 \quad 2 \quad 1 \quad 6 \quad 0 \quad -10$

$(19, 2)$

An $\boxed{\text{inversion}}$ is a pair (L_i, L_j) where $i < j$ but $L_i > L_j$ (an out-of-order pair).

The list L has: $5 + \overset{6}{\cancel{0}} + 1 + 3 + 2 + 2 + 1 = \cancel{14}\ 20$

Goal: compute the # of inversions in a list of n elements

Obvious algorithm: $\underbrace{\text{Check all pairs,}}_{}$ $\underline{O(n^2)}$.

$\binom{n}{2}$ all ways of picking 2 things out of n

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

Divide-and-conquer: $\frac{n}{2} \cdot \frac{n}{2} = \frac{n^2}{4}$ $O(n^2)$

$$L = [\underbrace{\overbrace{3 \quad 19 \quad -7 \quad 2}^{n/2}} \mid \underbrace{\overbrace{1 \quad 6 \quad 0 \quad -10}^{n/2}}]$$

recursively count inversions
4

recursively count inversions
5

So, 9 inversions <u>within</u> a half. How many <u>between</u> the lists? That would be a blue element that is larger than a red one.

Right now, to do that, we'd have to go through all (blue, red) pairs, which takes $n^2/4$ time (still $O(n^2)$, not good!)

Here's the trick: While we're counting inversions, we'll <u>also</u> sort the lists, which we know takes $O(n\log(n))$ time.

$$L = \underbrace{3 \quad 19 \quad -7 \quad 2}_{4} \;\Big|\; \underbrace{1 \quad 6 \quad 0 \quad -10}_{5}$$

$$-7 \quad 2 \quad 3 \quad 19 \;\Big|\; -10 \quad 0 \quad 1 \quad 6$$

Now we recombine the lists just like the mergesort, and when do we detect an inversion? Anytime we take from the red list, there is an inversion for everything left in the blue list.

$$\overset{}{-7} \quad \overset{4}{2} \quad 3 \quad 19 \;\Big|\; -10 \quad \overset{5}{0} \quad 1 \quad 6$$

* *

$4+5=9$

$\parallel$

20

$$-10 \quad -7 \quad 0 \quad 1 \quad 2 \quad 3 \quad 6 \quad 19$$

$$4 + 3+3 + 1$$

Time: $T(n) = 2T(\frac{n}{2}) + 2n$

$\rightsquigarrow T(n) = O(n \log(n))$

Ex #4: Closest Pair of Points (hard) (70s)

Input: n points $P = \{p_1, p_2, \dots, p_n\}$

Goal: Find the pair (p_i, p_j) such that $d(p_i, p_j)$ = Euclidean Distance is minimized.

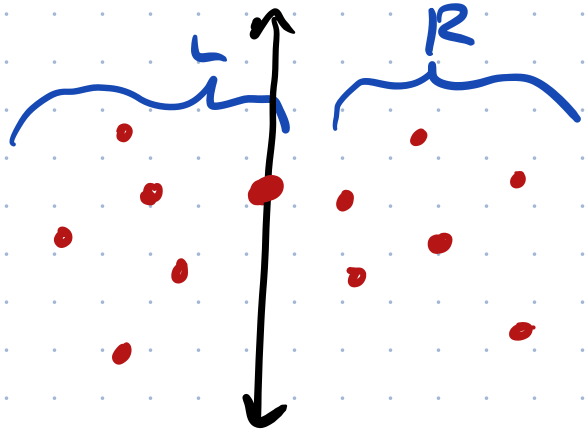
(Assume distinct x and y values for simplicity.)

$O(2 \cdot n \cdot \log(n)) = O(n \cdot \log(n))$

Step 1: - Create a version of P that is sorted by x-value, call it P_x.
- Create a version of P that is sorted by y-value, call it P_y. $O(n \log(n))$

Step 2: Begin divide-and-conquer.

- Split P into left half L and right half R using P_x. $O(1)$



- Form L_x, L_y, R_x, R_y using P_x and P_y. $O(n)$

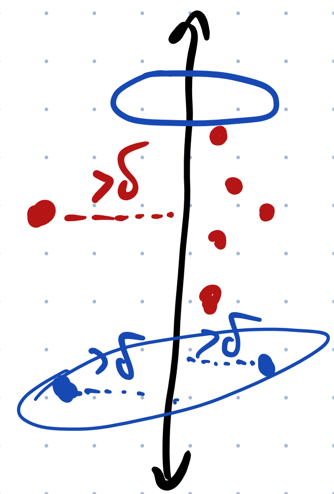
- Find closest pair in L: (ℓ_1, ℓ_2) and closest pair in R: (r_1, r_2) recursion.

- Set $\delta = \min(d(\ell_1, \ell_2), d(r_1, r_2))$. $O(1)$

- Now the hard part: how do we combine?

Closest pair could be in L, in R, or have one point in each.

Fact 1: If the closest pair is split across the middle line, then each point has to be within δ of the line

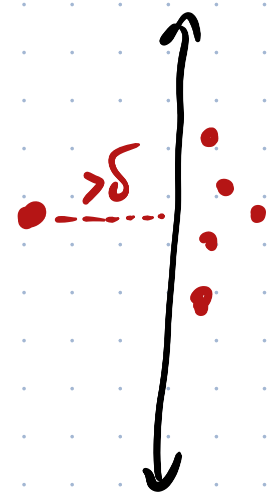


$\delta = \min(d(l_1, l_2), d(r_1, r_2))$

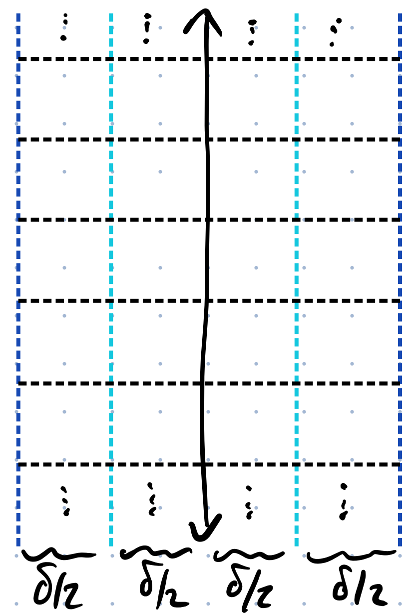
Define S to be just the points of P within δ of the line. $O(n)$

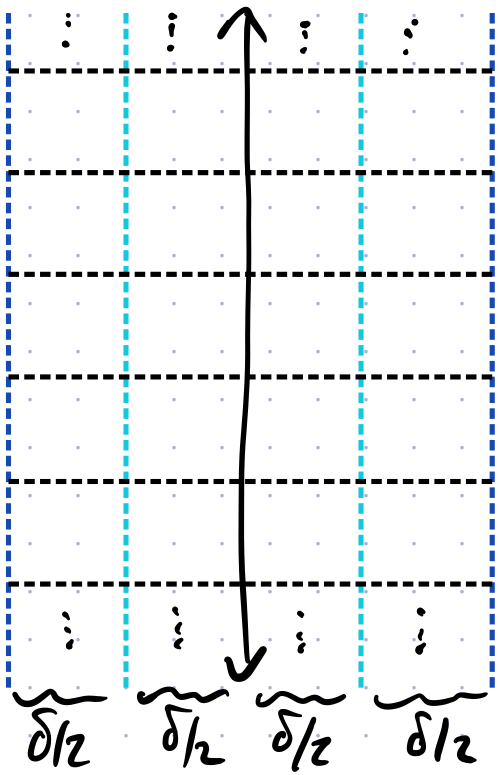
Note that $S = P$ is possible!

Form S_x and S_y using P_x and P_y. $O(n)$



Here's where it gets really weird! Split up the 2δ-wide vertical strip centered on the middle line into $\delta/2 \times \delta/2$ boxes.





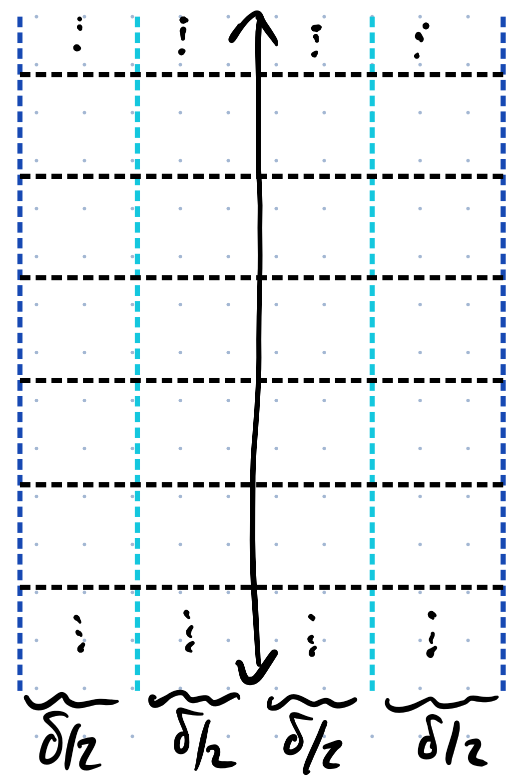
Fact 2: Each box contains at most a single point of S. (Otherwise, those points would be $< \frac{\delta}{2}\sqrt{2} < \delta$ apart, contradicting the fact that δ is min. distance on either side of the line.)

Let's think about S_y, the points in S ordered by y-value.

If you have two points in S_y that are 4 positions apart (e.g., the 10^{th} and 14^{th}), they have to be on different rows.

8 apart $\rightsquigarrow$ empty row between them $\rightsquigarrow \geq \delta/2$ apart

12 apart $\rightsquigarrow$ 2 empty rows between them $\rightsquigarrow \geq \delta$ apart



<u>Fact 3</u>: If two points in S are $\leq \delta$ apart, their positions in S_y differ by <u>at most 11</u>.

So, to find the closest pair in S, we don't have to check every pair ($O(|S|^2)$), only the pairs at most 11 apart in the list

$\cancel{(s_1, s_{13})}$

$(s_1, s_2), (s_1, s_3), \ldots (s_1, s_{12})$ 11

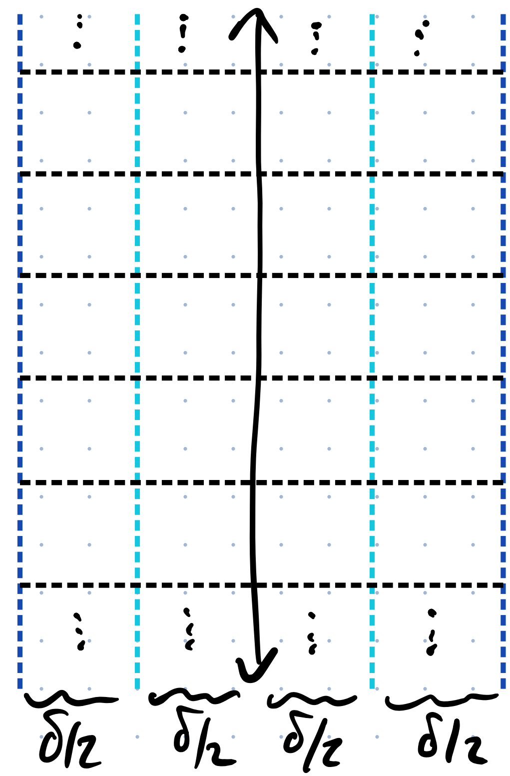
$(s_2, s_3), (s_2, s_4), \ldots (s_2, s_{13})$ + 11

+ 11

$\vdots$

$11 \cdot n$ things to check

$= O(n)$



Summary:

- Presort to get P_x, P_y $O(n \log(n))$
- Split in half and form L_x, L_y, R_x, R_y $O(n)$
- Recursively solve on L and R
- Find δ, S_x, S_y $O(n)$
- Check pairs in S at most 11 apart $O(n)$

$$T(n) = O(n \cdot \log(n)) + S(n)$$

$$S(n) = O(n) + 2 \cdot S(n/2) + O(n) + O(n)$$

$$\Rightarrow S(n) = O(n \cdot \log(n))$$

$$\Rightarrow T(n) = O(n \cdot \log(n)).$$

Other famous divide-and-conquer examples.

Integer Multiplication

Input: Two n-digit numbers x and y

Output: $x \cdot y$

Simple algorithm:

$$
\begin{array}{r}
172 \\
424 \\
\hline
688 \\
3440 \\
68800 \\
\hline
72928
\end{array}
$$

$O(n^2)$

D+C: $T(n) \le 3T(n/2) + O(n)$

$\Rightarrow T(n) = O(n^{\log_2(3)}) = O(n^{1.59\ldots})$

Kind of crazy!

Summary:

- Split in two
- Solve each half recursively
- Combine into a big solution **faster** than brute force.

$O(n^3)$

n