Scientific Computing

Fri, Feb 13

Announcements

* Homework 2 due **tonight**, 11:59pm
  pdf + zip file on D2L

Don't forget to keep track of and **cite** any external resources you use - friends, websites, AI, etc.

Two kinds of things to cite:

(1) A resource helped me learn about a topic

(2) A resource wrote this line of code.
Be specific.

* Also, written explanations should be your own words.

Office Hours:

Mon, 9:30-10:30

Fri, 2:00-3:00

Cudahy 307

* Homework 3 assigned, due in 2 weeks
  covers search spaces, brute force,
  and divide-and-conquer

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

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

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

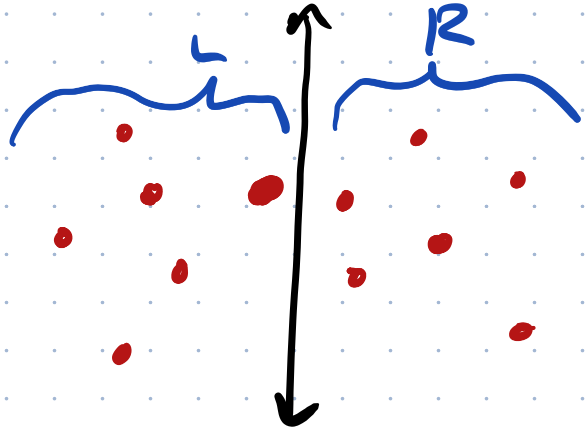
$\sqrt{(x_2-x_1)^2+(y_2-y_1)^2}$

(Assume distinct x and y values for simplicity.)

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. $O(n)$

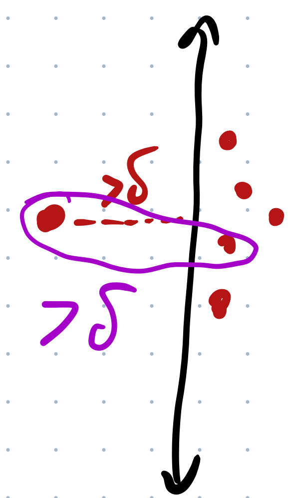
- Find closest pair in L: (ℓ_1, ℓ_2) and closest pair in R: (r_1, r_2) } recursion.

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

- Now the hard part: how do we combine?

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

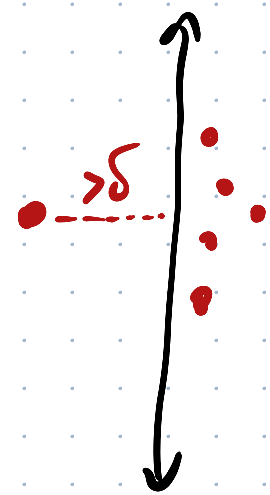
<u>Fact 1:</u> If the closest pair is split across the middle line, then each point has to be <u>within</u> δ of the line



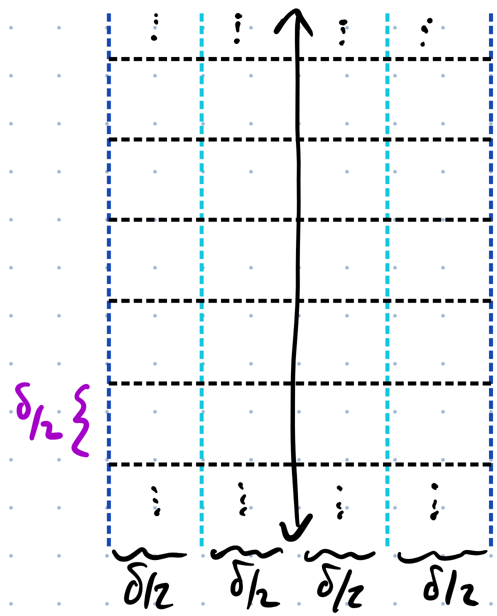
Define S to be just the points within δ of the line. $O(n)$

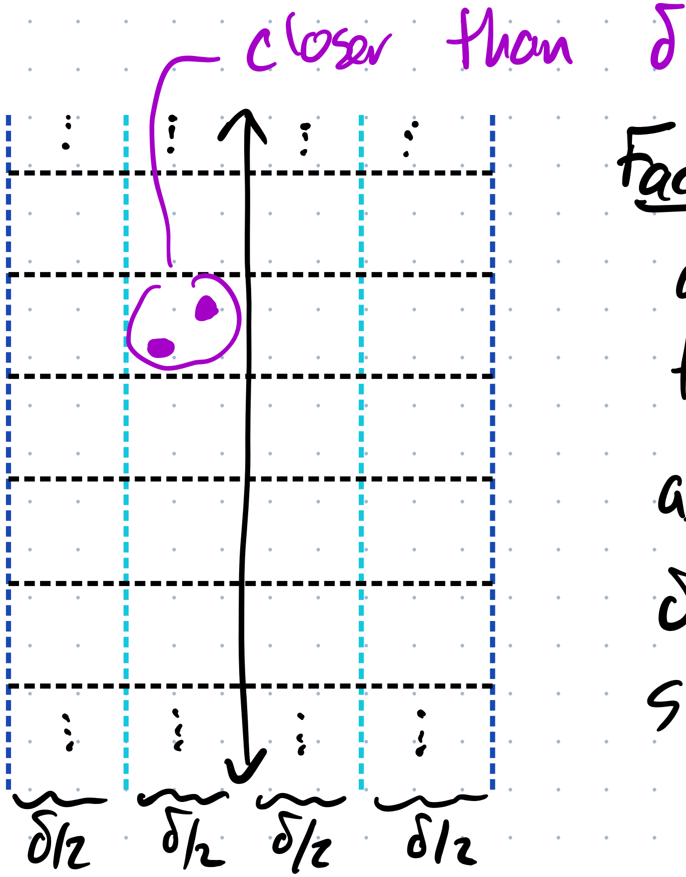
Note that $S = P$ is possible!

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.





<u>Fact 2</u>: Each box contains <u>at most</u> 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.)

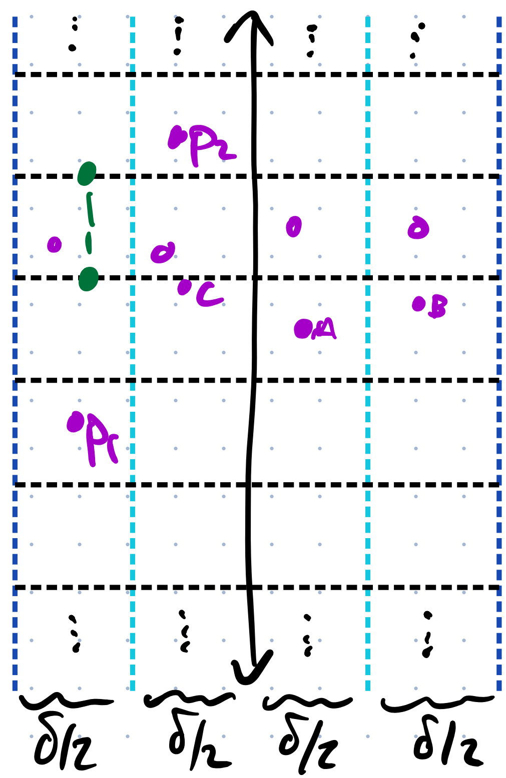
Let's think about S_y, the points in S ordered by y-value.

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.

$S_y = [\ldots\ldots p_1, A, B, C, p_2, \ldots\ldots]$

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

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



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

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

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

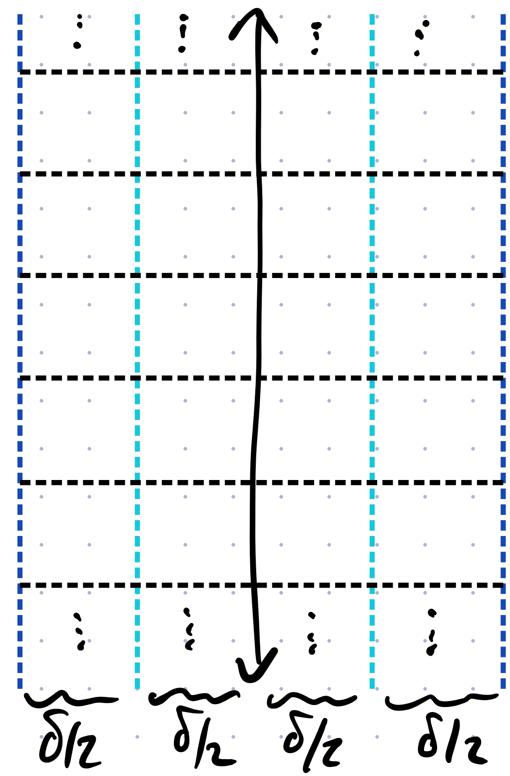
$(s_2, s_3), (s_2, s_4), \dots (s_2, s_{13})$ $\quad +11$

$\vdots$ $\quad +11$

$\vdots$

$11 \cdot n$ things to check

$= O(n)$



Summary:

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

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

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

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

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

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: $T(n) \leq 3T(n/2) + O(n)$

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

Kind of crazy!

Summary:

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

Topic 8 – Backtracking

Like Divide + Conquer, Backtracking is a framework for finding the optimal solution in a search space without checking every candidate one-by-one.

Very simple idea: Build solutions one part at a time, and give up when a partial solution violates the constraints.

Ex #1: Knapsack

Capacity: 10

item	weight	value
1	8	13
2	3	7
3	5	10
4	5	10
5	2	1
6	2	1
7	2	1

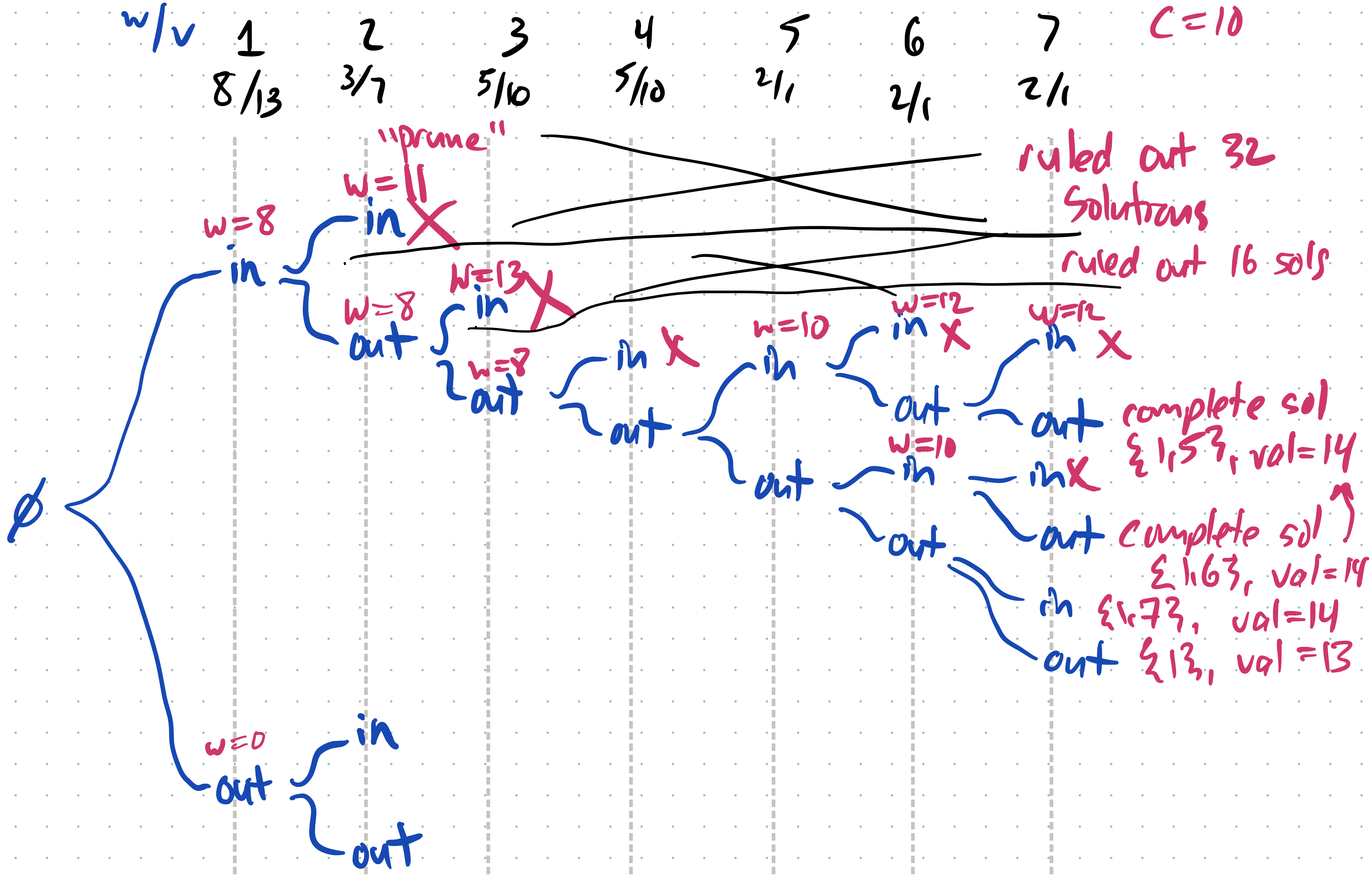
With brute force:

Possibilities: $\emptyset$, $\{1\}$, $\{2\}$, ...

$\{1,3,4,5,7\}$, ...

not just too heavy, but still too heavy if you remove any single item, so this is silly to even try!

128 possibilities



w/v
1 8/13
2 3/7
3 5/10
4 5/10
5 2/1
6 2/1
7 2/1
C=10
"prune"
ruled out 32 Solutions
ruled out 16 sols
w=8 in
w=11 in
w=8 out
w=13 in
w=8 out
w=10 in
w=12 in
w=12 in
w=10 in
w=0 out
complete sol {1,5}, val=14
complete sol {1,6}, val=14
in {1,7}, val=14
out {1}, val=13
∅