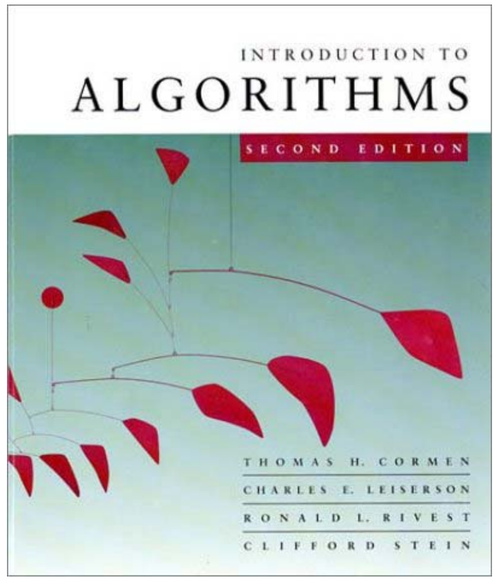
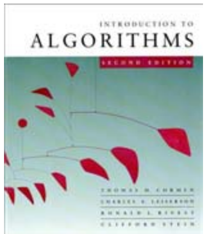
Introduction to Algorithms

6.046J/18.401

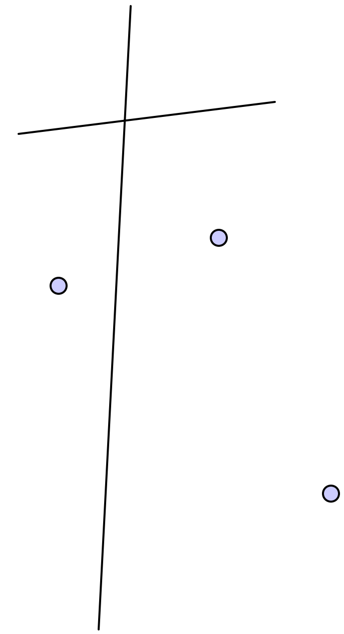


Lecture 18

Prof. Piotr Indyk



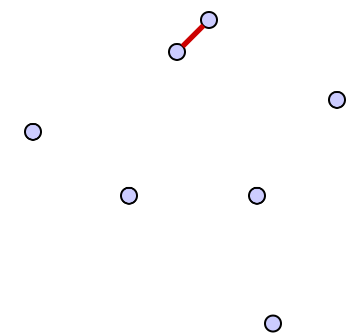
Today



- We have seen algorithms for:
  - “numerical” data (sorting, median)
  - graphs (shortest path, MST)
- Today and the next lecture: algorithms for geometric data

Computational Geometry

- Algorithms for geometric problems
- Applications: CAD, GIS, computer vision,…….
- E.g., the *closest pair* problem:
  - Given: a set of points $P=\{p_1 \ldots p_n\}$ in the plane, such that $p_i=(x_i,y_i)$
  - Goal: find a pair $p_i \neq p_j$ that minimizes $||p_i - p_j||$



$||p-q|| = [(p_x-q_x)^2+(p_y-q_y)^2]^{1/2}$

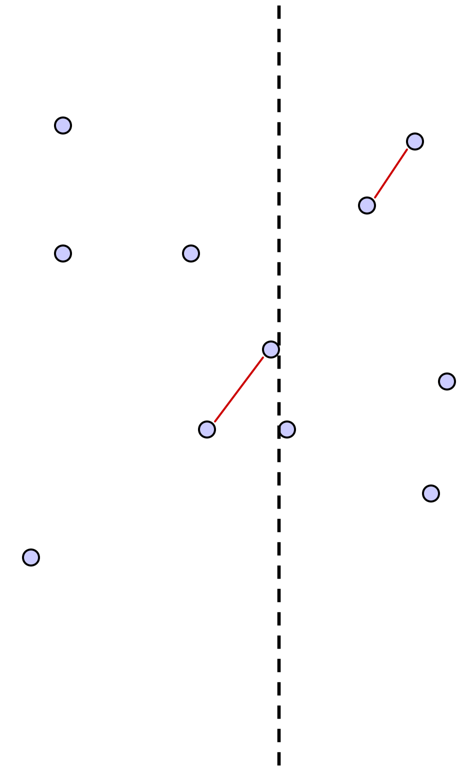
- We will see more examples in the next lecture

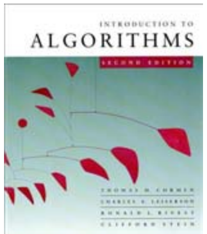
Closest Pair

- Find a closest pair among $p_1 \ldots p_n$
- Easy to do in $O(n^2)$ time
  - For all $p_i \neq p_j$, compute $||p_i - p_j||$ and choose the minimum
- We will aim for $O(n \log n)$ time

Divide and conquer

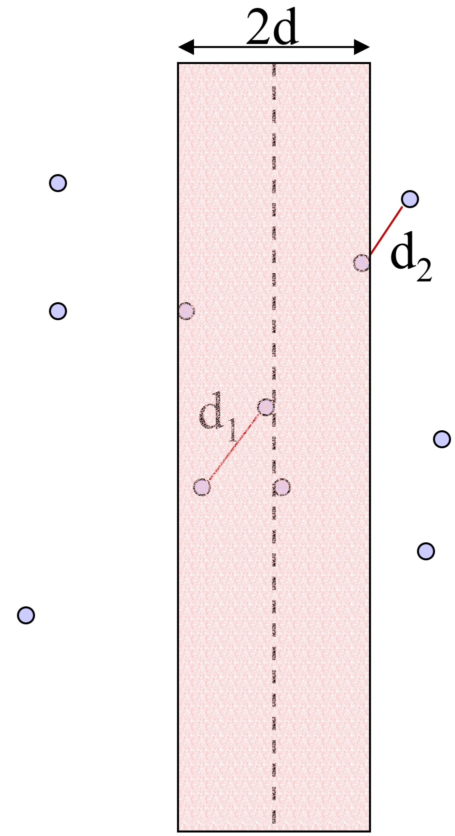
- Divide:
  – Compute the median of x-coordinates
  – Split the points into P_L and P_R, each of size $n/2$
- Conquer: compute the closest pairs for P_L and P_R
- Combine the results (the hard part)





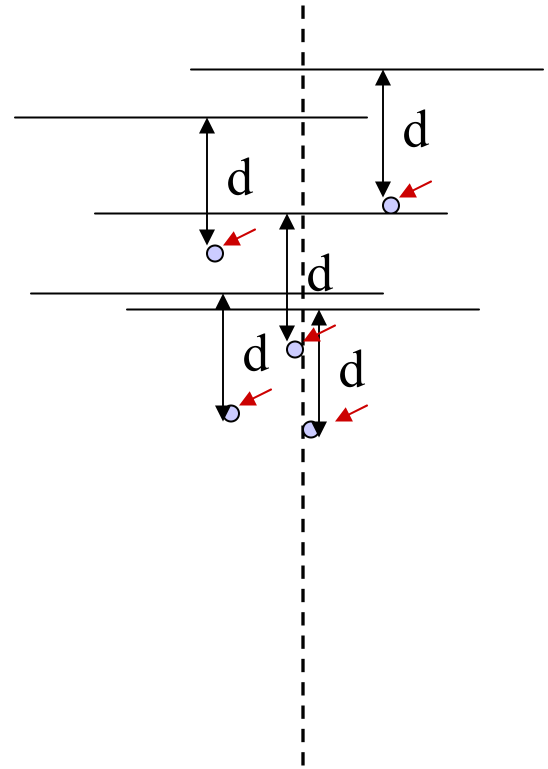
Combine

- Let $d=\min(d_1,d_2)$
- Observe:
  - Need to check only pairs which cross the dividing line
  - Only interested in pairs within distance $< d$
- Suffices to look at points in the $2d$-width strip around the median line



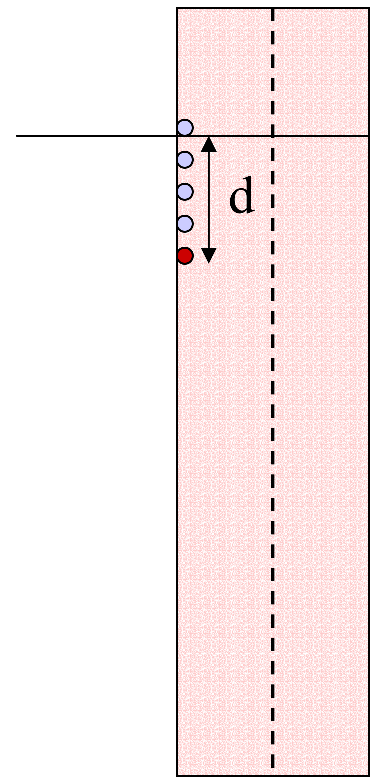
Scanning the strip

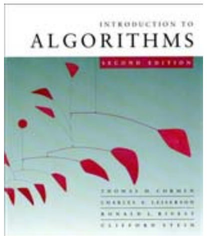
- Sort all points in the strip by their y-coordinates, forming $q_1 \ldots q_k$, $k \le n$.
- Let y_i be the y-coordinate of q_i
- $d_{min} = d$
- For $i=1$ to k
  - $j=i-1$
  - While $y_i - y_j < d$
    - If $\|q_i - q_j\| < d$ then $d_{min} = \|q_i - q_j\|$
    - $j := j-1$
- Report d_{min} (and the corresponding pair)



Analysis

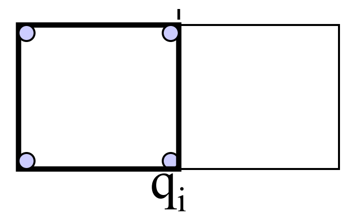
- Correctness: easy
- Running time is more involved
- Can we have many q_j's that are within distance d from q_i ?
- No
- Proof by *packing* argument





Analysis, ctd.

Theorem: there are at most 7 q_j's such that $y_i - y_j \le d$.

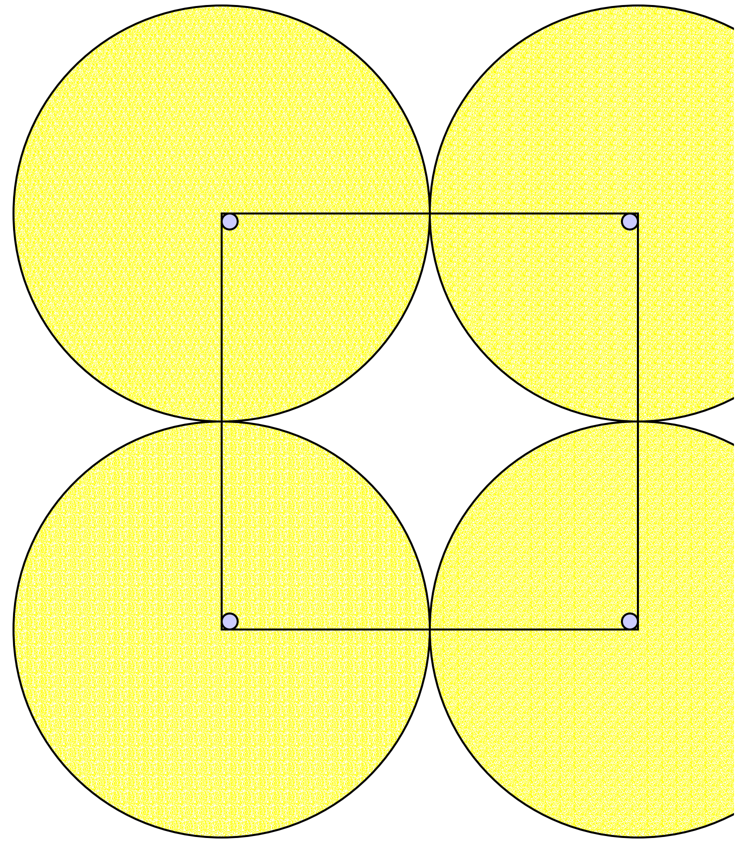


Proof:

- Each such q_j must lie either in the left or in the right $d \times d$ square
- Within each square, all points have distance distance $\ge d$ from others
- We can pack at most 4 such points into one square, so we have 8 points total (incl. q_i)

Packing bound

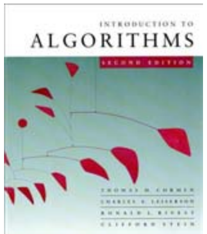
- Proving "4" is not easy
- Will prove "5"
  - Draw a disk of radius $d/2$ around each point
  - Disks are disjoint
  - The disk-square intersection has area $\geq \pi (d/2)^2/4 = \pi/16\ d^2$
  - The square has area d^2
  - Can pack at most $16/\pi \approx 5.1$ points



Running time

- Divide: $O(n)$
- Combine: $O(n \log n)$ because we sort by y
- However, we can:
  - Sort all points by y at the beginning
  - Divide preserves the y-order of points

  Then combine takes only $O(n)$
- We get $T(n)=2T(n/2)+O(n)$, so $T(n)=O(n \log n)$



Close pair

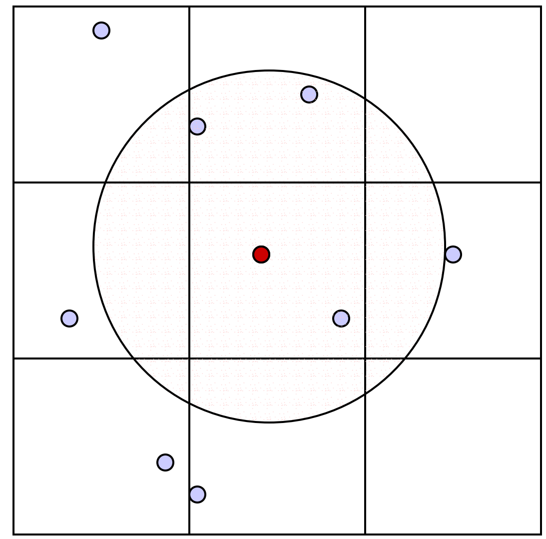
- Given: $P=\{p_1 \ldots p_n\}$
- Goal: check if there is any pair $p_i \neq p_j$ within distance R from each other
- Will give an $O(n)$ time algorithm, using…

…radix sort !

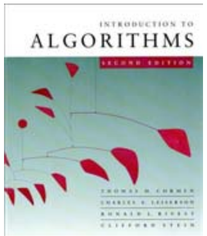
(assuming coordinates are small integers)

Algorithm

- Impose a square grid onto the plane, where each cell is an $R \times R$ square
- Put each point into a bucket corresponding to the cell it belongs to. That is:
  - For each point $p=(x,y)$, create computes its bucket ID $b(p)=(\lfloor x/R \rfloor, \lfloor y/R \rfloor)$
  - Radix sort all $b(p)$'s
  - Each sequence of the same $b(p)$ forms a bucket
- If there is a bucket with > 4 points in it, answer YES and exit
- Otherwise, for each $p \in P$:
  - Let $c = b(p)$
  - Let C be the set of bucket IDs of the 8 cells adjacent to c
  - For all points q from buckets in $C \cup \{c\}$
    - If $\|p-q\| \le R$, then answer YES and exit
- Answer NO

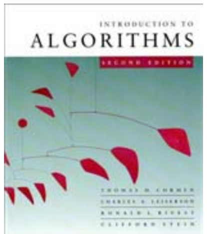


(1,1), (1,2), (1,2), (2,1), (2,2), (2,2), (2,3), (3,1), (3,2)



Bucket access

- Given a bucket ID c, how can we quickly retrieve all points p such that b(p)=c ?
- This is exactly the dictionary problem (Lecture 7)
- E.g., we can use hashing.



Analysis

- Running time:
  - Putting points into the buckets: $O(n)$ time
  - Checking if there is a heavy bucket: $O(n)$
  - Checking the cells: $9 \times 4 \times n = O(n)$
- Overall: linear time

Computational Model

- In the two lectures, we assume that
  - The input (e.g., point coordinates) are *real* numbers
  - We can perform (natural) operations on them in *constant* time, with perfect precision
- Advantage: simplicity
- Drawbacks: highly non-trivial issues:
  - Theoretical: if we allow arbitrary operations on reals, we can compress n numbers into a one number
  - Practical: algorithm designed for infinite precision sometimes fail on real computers