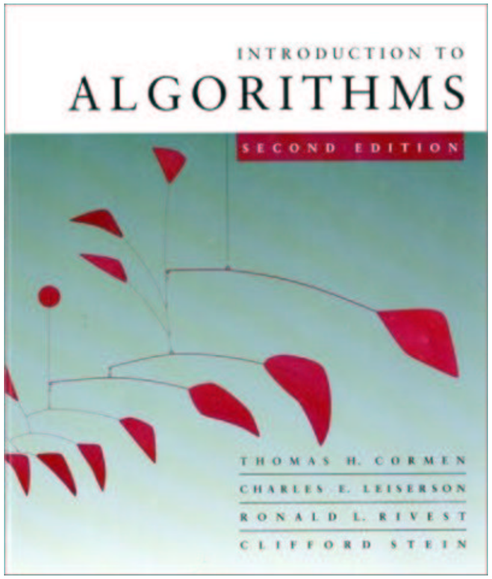
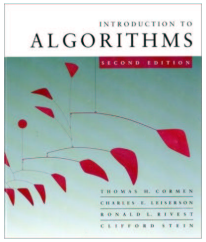
Introduction to Algorithms

6.046J/18.401



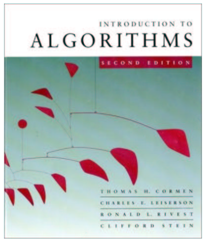
Lecture 19

Prof. Piotr Indyk



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 $||p_i - p_j||$
- We will see more examples in the next lecture



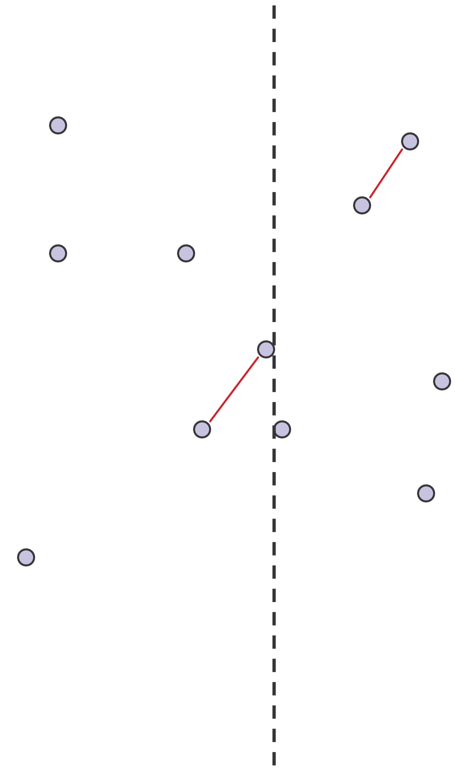
Computational Model

- In the next two lectures, we will 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

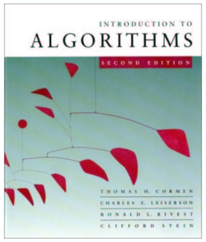
Closest Pair

- 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

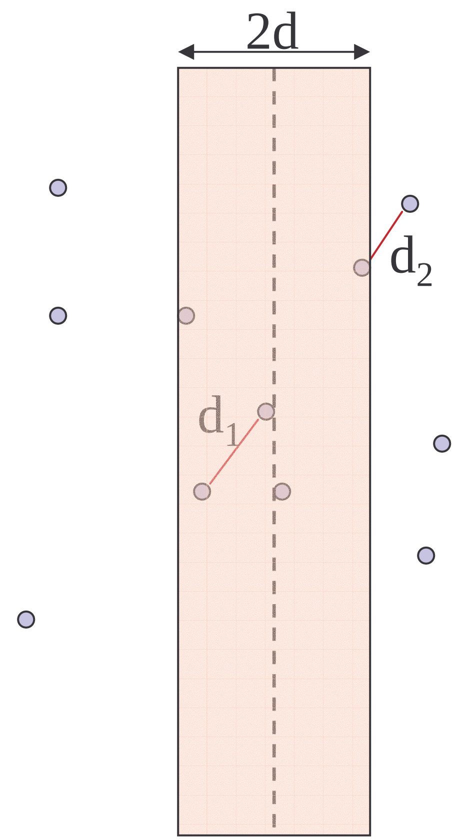
Divide and conquer



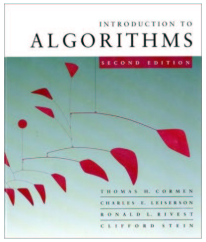
- Divide:
  - Compute the median of x-coordinates
  - Split the points into P_L and P_R, each of size $n/2$
- Conquer: compute the closest pairs for P_L and P_R
- Combine the results (the hard part)



Combine

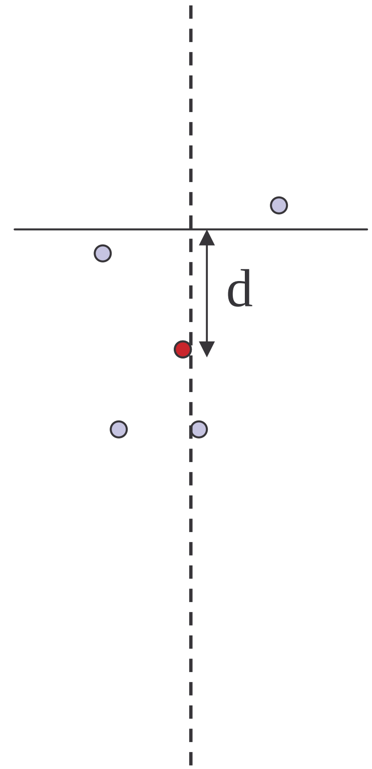


- Let $d=\min(d_1,d_2)$
- Observe:
  - Need to check only pairs which cross the dividing line
  - Only interested in pairs within distance $< d$
- Suffices to look at points in the 2d-width strip around the median line



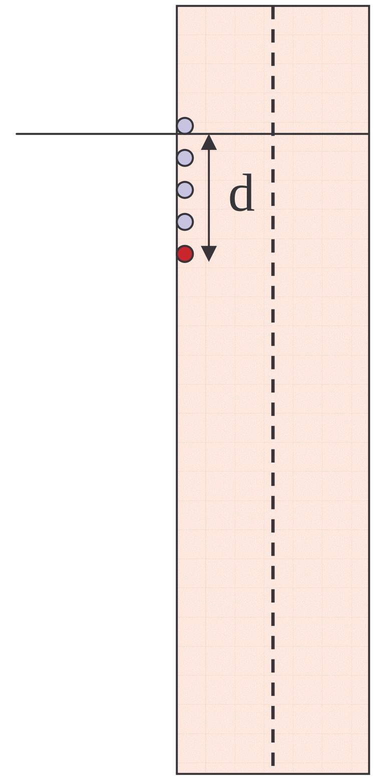
Scanning the strip

- 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
- For $i=1$ to k
  - $j=i-1$
  - While $y_i-y_j < d$
    - Check the pair q_i,q_j
    - j:=j-1



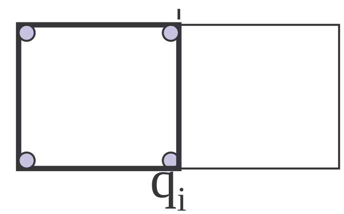
Analysis

- 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 $y_i - y_j \leq d$.

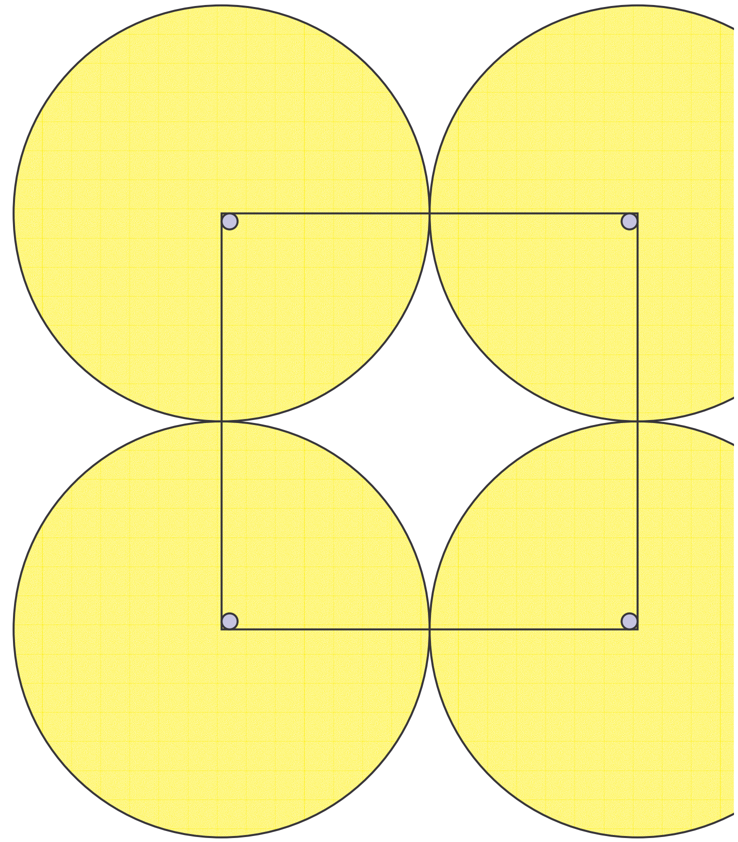


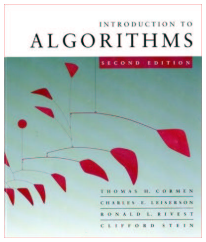
Proof:

- 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 $\geq d$ 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 trivial
- 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

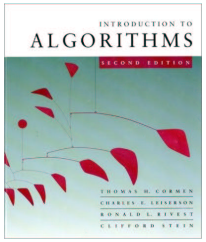
- Divide: $O(n)$
- Combine: $O(n \log n)$ because we sort by y
- However, we can:
  - Sort all points by y at the beginning
  - Divide preserves the y-order of points

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

Close pair

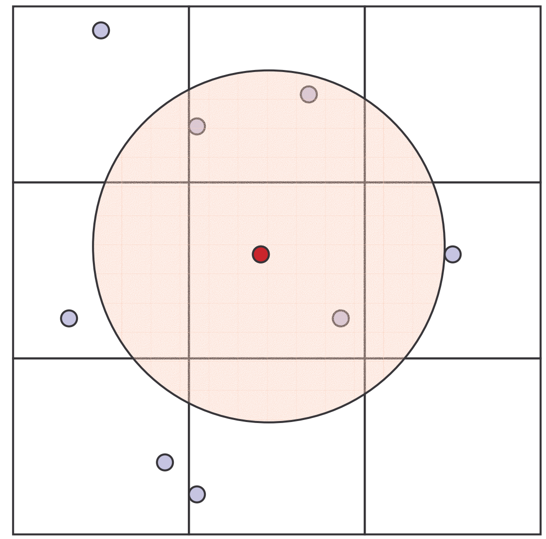
- Given: $P=\{p_1 \ldots p_n\}$
- Goal: check if there is any pair $p_i \neq p_j$ within distance 1 from each other
- Will give an $O(n)$ time randomized algorithm, using…

… hashing!



Algorithm

1. Impose a square grid onto the plane, where each cell is a 1×1 square
2. Put each point into a bucket corresponding to the cell it belongs to (see last slide)
3. If there is a bucket with > 4 points in it, answer YES (by the packing theorem)
4. Otherwise, for each $p \in P$, check all points in the cell containing p, as well as the cells adjacent to it



Analysis

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

To hash or not to hash

- In step 2 of the algorithm, we need to partition the points into "buckets", i.e., sets $B_1 \ldots B_k$, $k \leq n$. Each bucket contains all points that belong to some non-empty cell.
- This can be solved using any data structure for the "symbol table" problem, as in Lecture 7. The key of a point $p=(x,y)$ is the identifier of the cell that p belongs to. Note that now the keys are not unique, i.e., many points can have the same key.
- We could solve the symbol table problem using direct access table. However, the space used by the algorithm would be proportional to the total number of cells in the grid, which could be much larger than n. In particular, we would not be able to initialize that much space in $O(n)$ time.
- Hashing allows us to reduce the space (and initialization time) to $O(n)$, since the space depends only on the number of nonempty cells. Since hashing uses randomness, the resulting algorithm is randomized.