Lecture 21: April 30, 2002

Computational Geometry

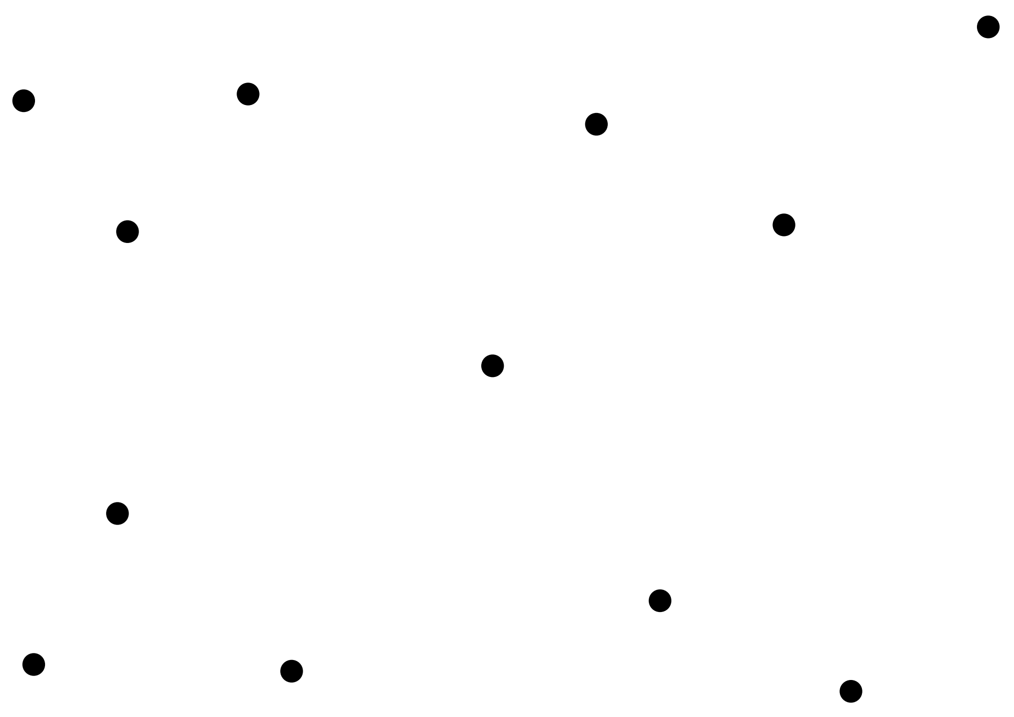
- Closest pair

Closest Pair

- Given: $p_1 \ldots p_n \in \Re^2$ (i.e., in the plane)
- Goal: find a pair p_i, p_j, $i \neq j$, such that

$$||p_i - p_j|| = \min_{i \neq j} ||p_i - p_j||$$

I.e., find the closest pair of points according to Euclidean distance.



Can do it easily in $O(n^2)$ time.
We will aim for $O(n \log n)$.

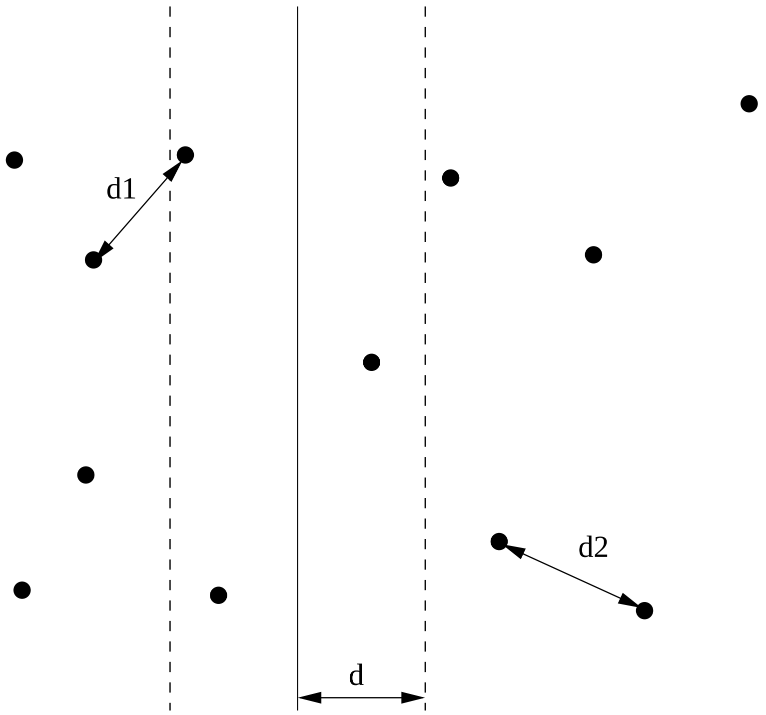
Divide and conquer

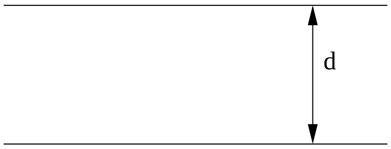
- Divide the points using a vertical line, such that the left set and the right set contain $n/2$ points each (i.e., the line is defined by the median of x coordinates)
- Compute the closest pair for the left and right sets recursively
- Combine the results (the hard part)

Merge

Assume the length of the closest pair in the left set is d_1 (and d_2 for the right set).
Let $d = \min(d_1, d_2)$.

- Need to check only pairs which cross the median line
- We are interested only in pairs within distance $< d$
- $\Rightarrow$ Sufficient to look at points in the stripe around the median line




d

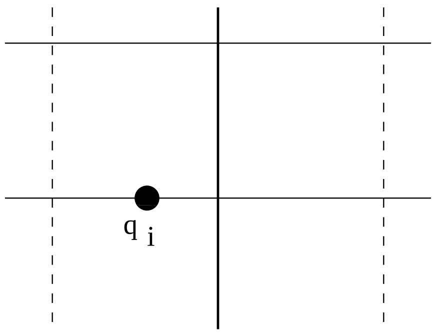
How to search the stripe ?

- Sort all points in the stripe by their y coordinates
- Let the resulting sequence be $q_1 \dots q_k$
- For $i = 1$ to k
  - Let y_i be the y-th coordinate of q_i
  - For all points q_j, $j < i$, such that $y_i - y_j < d$, check the pair $q_i - q_j$

Correctness is immediate.

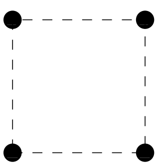
What about the running time ?

How many q_j's close to q_i are there ?



At most a constant number of them !

- C points in the $2d \times d$ rectangle
  $\Rightarrow \geq C/2$ points in one of the two $d \times d$ sub-squares
- But each square lies on only one side of the median
  $\Rightarrow$ pairwise distances between the points are $\geq d$
- We cannot pack more than 4 such points in a $d \times d$ square

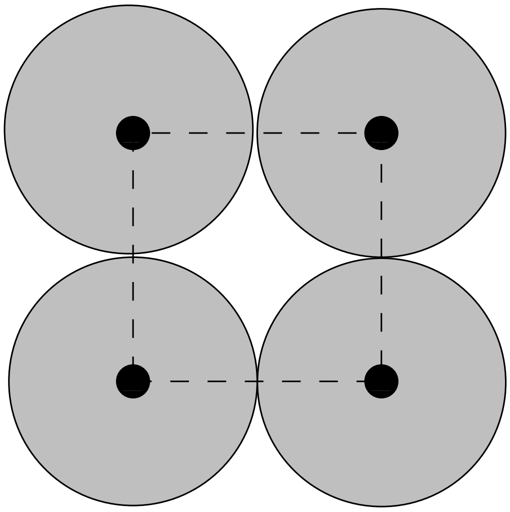


- Therefore, $C/2 \leq 4 \Rightarrow C \leq 8$

Packing argument made precise

We will show one cannot pack 6 points, with pairwise distance 1, into a unit square:

- Draw a disk of radius $1/2$ around each point
- Disks are disjoint
- The intersection of any disk with the square has area $\geq 1/4\pi(1/2)^2 = \pi/16 \approx 0.196$
- The unit square has area $= 1$
- $\Rightarrow$ cannot pack more than $16/\pi \approx 5.1$ points



Running time

- Divide step takes $O(n)$ time
- Merge step takes $O(n \log n)$ due to sorting. However, we can sort only once at the beginning, so we get $O(n)$ time
- Then we get recursion $T(n) = 2T(n/2) + O(n)$, so $T(n) = n \log n$

Close pair

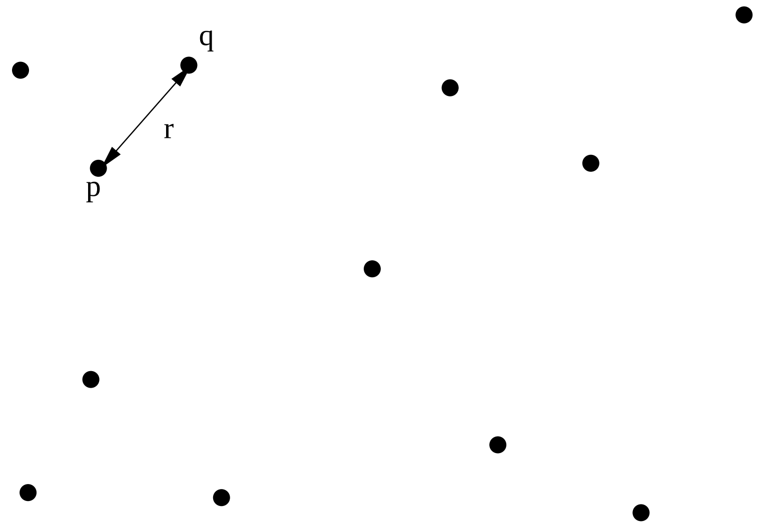
- Given: $p_1 \ldots p_n \in \Re^2$, $r > 0$
- Goal: check if there is any pair of distinct points p_i, p_j such that the distance between them is $\leq r$

Clearly, can be solved via *closest* pair in $O(n \log n)$ time.

Will give $O(n)$-time algorithm using hashing.

Randomization

Idea: Impose a regular square grid onto the plane, such that each cell has side length $4r$, and translate it at random.



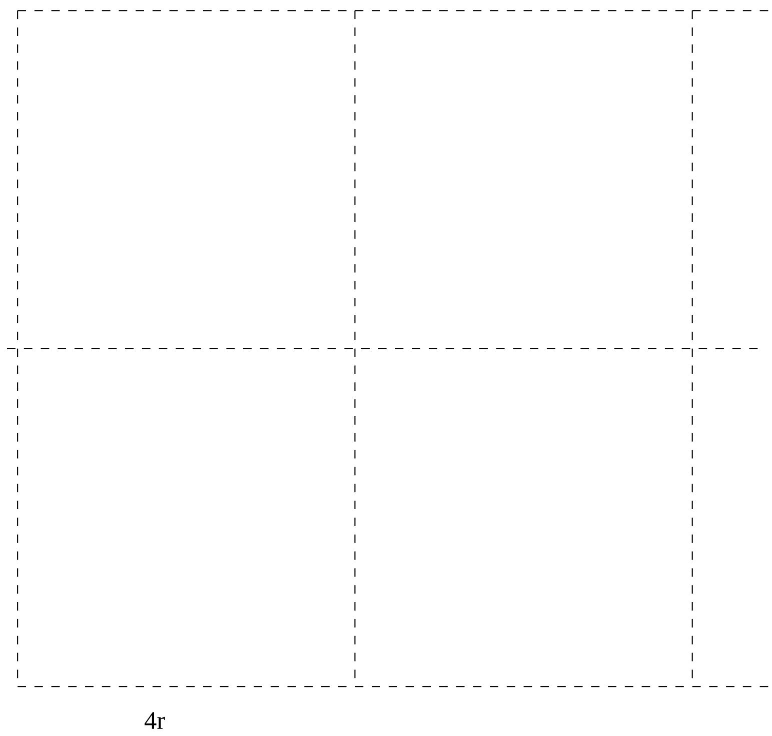
Fact: The prob. that p and q are in same cell is $\geq 1/2$.

Proof:

$\Pr[p, q$ separated by a horizontal line$] \leq 1/4$

$\Pr[p, q$ separated by a vertical line$] \leq 1/4$

$\Rightarrow \Pr[p, q$ not separated$] \geq 1 - 1/4 - 1/4 = 1/2$


4r

Algorithm

- For each cell c containing at least one point, compute the set B_c containing all points in that cell (this can be done in $O(n)$ using hashing)
- For all such sets B_c, search for a pair $p, q \in B_c$ such that p and q are within distance r from each other

How to implement the last step efficiently ? All points could go to the same bucket...

Fact: If $|B_c| \geq 256/\pi$, then B_c *must* contain a pair of points with distance $\leq r$.

Proof: Packing argument, same as before.

Algorithm ctd.

- If there is B_c with cardinality $> 256/\pi$, we know a close pair must exist
- Otherwise, enumeration of all pairs of points from any B_c takes constant time
- There are at most n non-empty sets B_c, so we are done in $O(n)$ time!

Final remark: do we really need to shift the grid at random ? And how to *find* the close pair ?

Computation over reals

- We assume the input numbers (e.g., coordinates) are reals, arbitrary precision
- Therefore, we need to perform computation over reals

Why ?

- Avoid the discretization issues, e.g.:
  - What is a line in the discrete setting ?
  - Do two discrete lines intersect at only one point ?

Drawbacks:

- We have finite precision in real life, so sometimes we need to adapt the algorithms to finite precision
- If we allow arbitrary operations on real numbers, we get strange implications

A paradox: number compression

- Given: two reals $x, y \in (0,1)$ such that $x = 0.x_1x_2\ldots$ and $y = 0.y_1y_2\ldots$
- Compressor: transform x, y into $z = 0.x_1y_1x_2y_2\ldots$
- Can compress many numbers into one !
- We are abusing the system ...

Solution

Allow only certain operations, e.g.

- Compute a (small degree) polynomial of the input coordinates, e.g., $x^2 + y^2$
- Compare result with 0, e.g., $x > 0$?
- Perhaps use floor $\lfloor x \rfloor$ (for hashing)
- Try to do only reasonable things.

References

- All the algorithms of this lecture were discovered in the 1970's.
- The close pair algorithm is also due to M. O. Rabin (he actually showed how to find the *closest* pair, though).