
A dark blue vertical bar is positioned on the left side of the slide, spanning the height of the main content area.

Divide-and-Conquer

A light blue vertical bar is positioned on the left side of the slide, spanning the height of the footer area.

Matrix Multiplication

Brute force

$$\begin{bmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{bmatrix} \times \begin{bmatrix} B_{00} & B_{01} \\ B_{10} & B_{11} \end{bmatrix} = \begin{bmatrix} C_{00} & C_{01} \\ C_{10} & C_{11} \end{bmatrix}$$

► $T(n) \in \Theta(n^3)$

Strassen's Matrix Multiplication

Strassen observed [1969] the product of two matrices as follows:

$$\begin{aligned} \triangleright & \begin{bmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{bmatrix} \times \begin{bmatrix} B_{00} & B_{01} \\ B_{10} & B_{11} \end{bmatrix} = \begin{bmatrix} C_{00} & C_{01} \\ C_{10} & C_{11} \end{bmatrix} \\ \triangleright & \begin{bmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{bmatrix} \times \begin{bmatrix} B_{00} & B_{01} \\ B_{10} & B_{11} \end{bmatrix} = \begin{bmatrix} M_1 + M_4 - M_5 + M_7 & M_3 + M_5 \\ M_2 + M_4 & M_1 + M_3 - M_2 + M_6 \end{bmatrix} \end{aligned}$$

- $\triangleright M_1 = (A_{00} + A_{11}) \times (B_{00} + B_{11})$
- $\triangleright M_2 = (A_{10} + A_{11}) \times B_{00}$
- $\triangleright M_3 = A_{00} \times (B_{01} - B_{11})$
- $\triangleright M_4 = A_{11} \times (B_{10} - B_{00})$
- $\triangleright M_5 = (A_{00} + A_{01}) \times B_{11}$
- $\triangleright M_6 = (A_{10} - A_{00}) \times (B_{00} + B_{01})$
- $\triangleright M_7 = (A_{01} - A_{11}) \times (B_{10} + B_{11})$

Analysis of Strassen's Algorithm

If n is not a power of 2, matrices can be padded with zeros.

Number of multiplications:

$$M(1) = 1$$

$$M(n) = 7M\left(\frac{n}{2}\right)$$

$$M(n) = 7^{\log_2 n} = n^{\log_2 7} \approx n^{2.807}$$

vs. n^3 of brute-force algorithm.

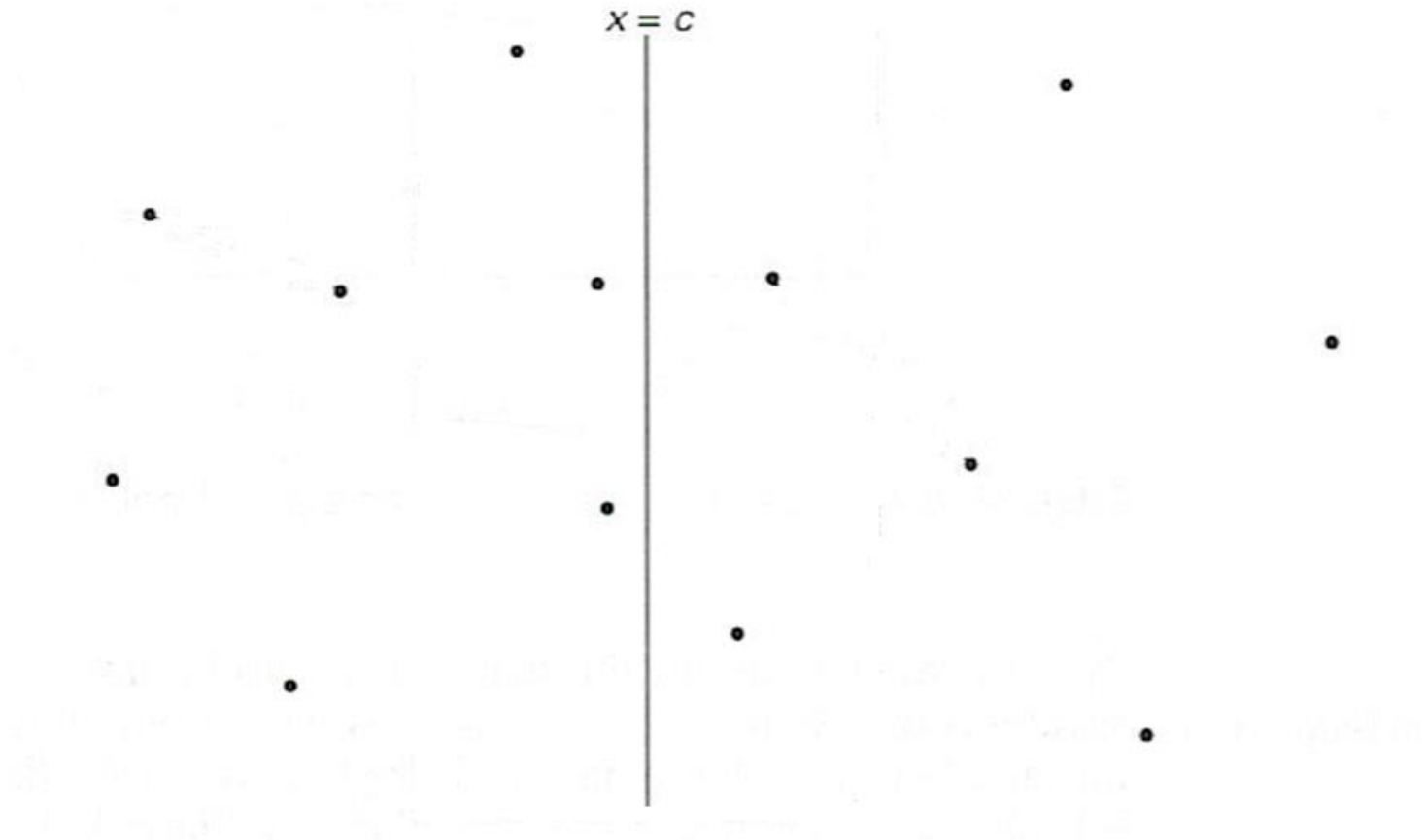
Closest-Pair Problem

- ▶ Find the two closest points in a set of n points (in the two-dimensional Cartesian plane).
- ▶ Brute-force algorithm:
 - ▶ Compute the distance between every pair of distinct points
 - ▶ Return the indexes of the points for which the distance is the smallest.

$$T(n) \in O(n^2)$$

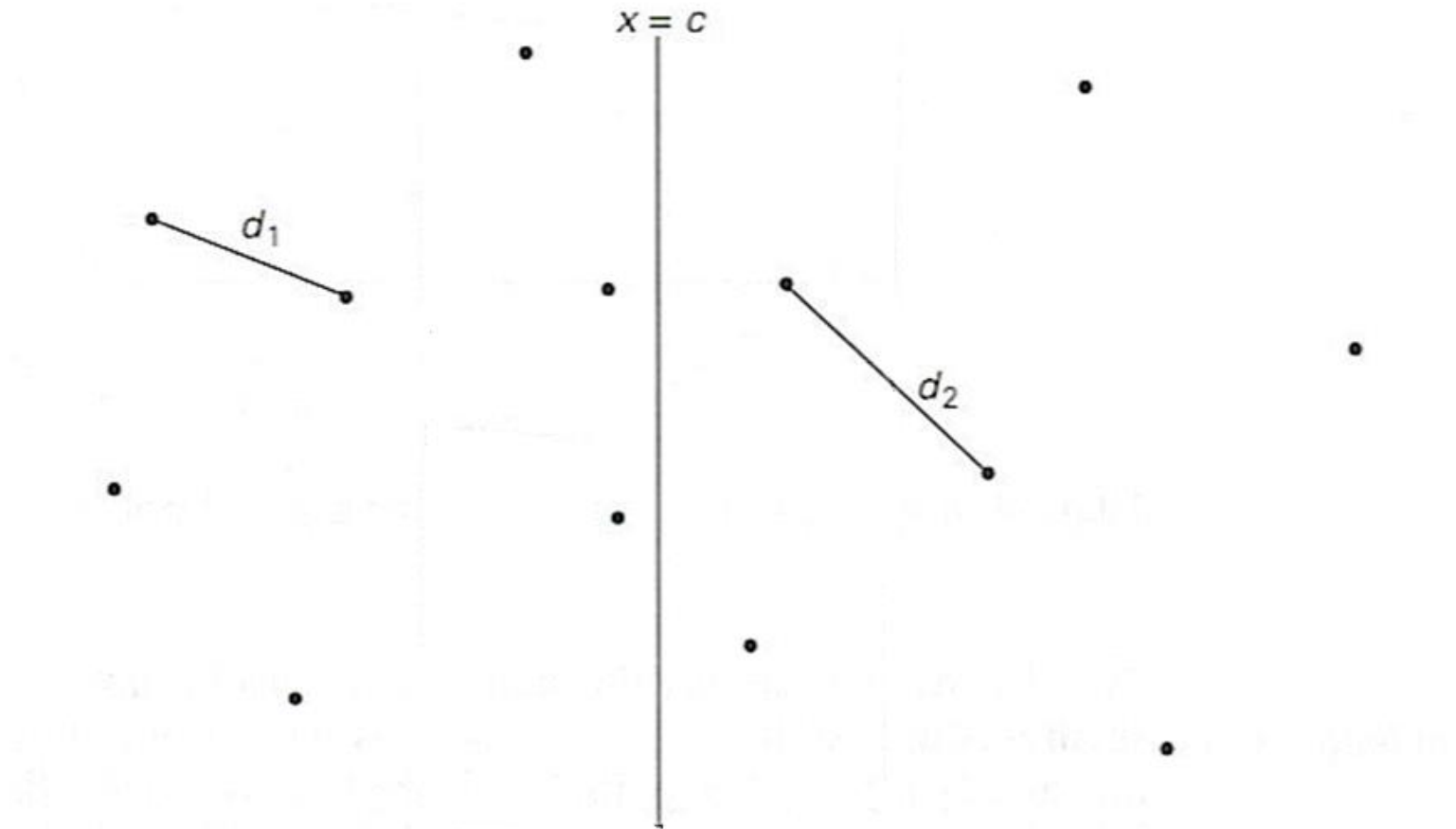
Closest-Pair Problem – Divide-and-Conquer

Step 1: Divide the points given into two subsets S_1 and S_2 by a vertical line $x = c$ so that half the points lie to the left or on the line and half the points lie to the right.



Closest-Pair Problem

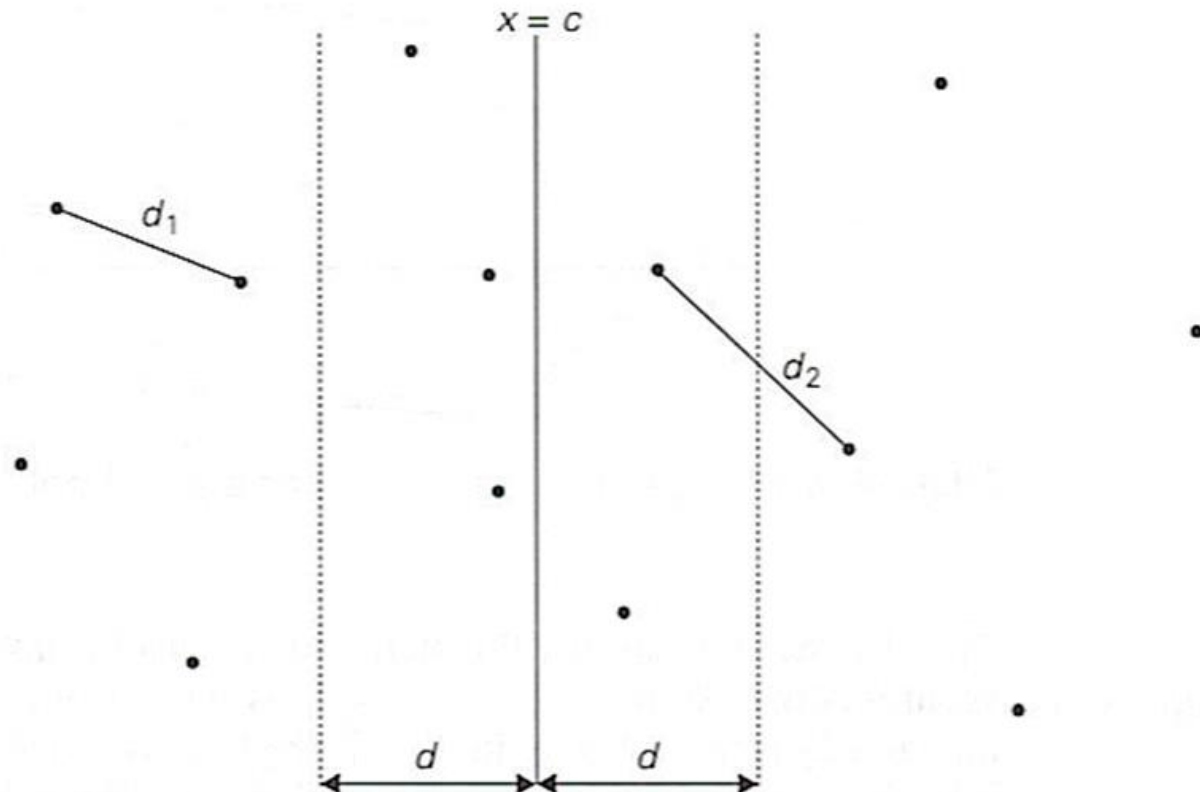
Step 2: Find recursively the closest pairs for the left and right subsets.



Closest-Pair Problem

Step 3: $d = \min(d_1, d_2)$

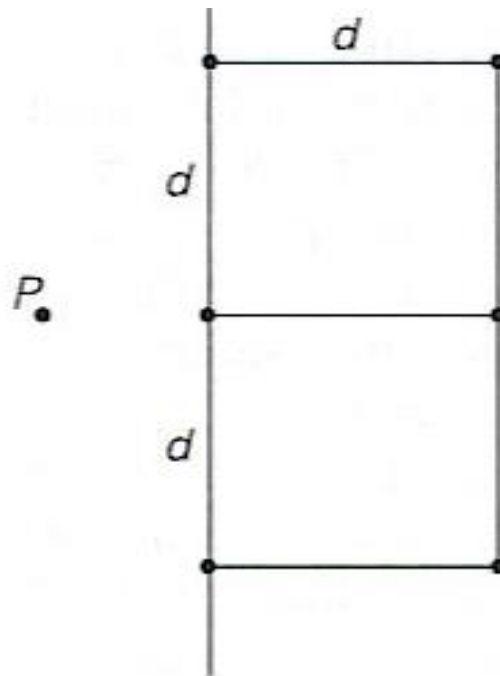
- Unfortunately, d is not necessarily the smallest distance between all pairs of points in S_1 and S_2 because a closer pair of points can lie on the opposite sides separating the line.



Closest-Pair Problem

Step 4: Checking boundary points:

P needs to be compared to at most six points across the boundary



Efficiency of the Closest-Pair Algorithm

Running time of the algorithm is described by

$$T(n) = 2T\left(\frac{n}{2}\right) + f(n)$$

Where $f(n) \in O(n)$

Efficiency of the Closest-Pair Algorithm

Running time of the algorithm is described by

$$T(n) = 2T\left(\frac{n}{2}\right) + f(n)$$

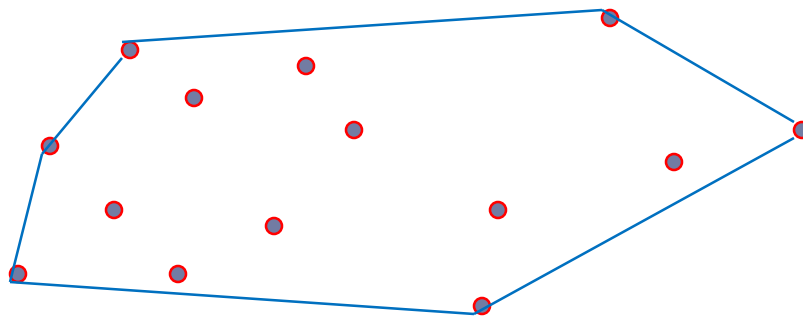
Where $f(n) \in O(n)$

By the Master Theorem (with $a = 2, b = 2, d = 1$)

$$T(n) \in O(n \log n)$$

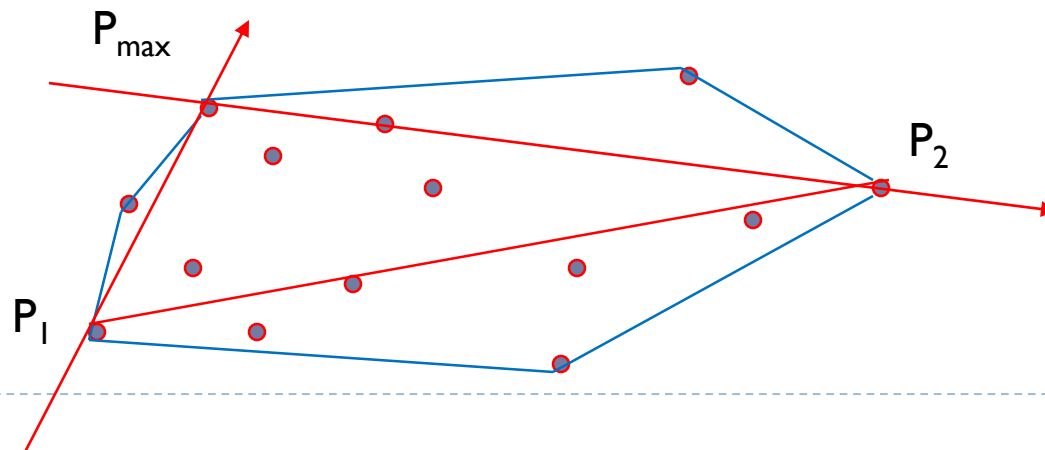
Convex hull Problem

Smallest convex polygon that includes given points



Quickhull Algorithm

- ▶ Assume points are sorted by x -coordinate values
- ▶ Identify *extreme points* P_1 and P_2 (leftmost and rightmost)
- ▶ Compute *upper hull* recursively:
 - ▶ find point P_{\max} that is farthest away from line P_1P_2
 - ▶ compute the upper hull of the points to the left of line P_1P_{\max}
 - ▶ compute the upper hull of the points to the left of line $P_{\max}P_2$
- ▶ Compute *lower hull* in a similar manner



Efficiency of Quickhull Algorithm

- ▶ Finding point farthest away from line P_1P_2 can be done in linear time
- ▶ Time efficiency:
 - ▶ worst case: $\Theta(n^2)$
 - ▶ average case: $\Theta(n)$
 - ▶ under reasonable assumptions about distribution of points given
- ▶ Several $O(n \log n)$ algorithms for convex hull are known

Discussion

Explain how one can find point P_{\max} in the quickhull algorithm analytically.

Discussion

Explain how one can find point P_{\max} in the quickhull algorithm analytically.

Since all the points in question serve as the third vertex for triangles with the same base P_1P_n , the farthest point is the one that maximizes the area of such a triangle. The area of a triangle, in turn, can be computed as one half of the magnitude of the determinant

$$\begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} = x_1y_2 + x_3y_1 + x_2y_3 - x_3y_2 - x_2y_1 - x_1y_3.$$

In other words, P_{\max} is a point whose coordinates (x_3, y_3) maximize the absolute value of the above expression in which (x_1, y_1) and (x_2, y_2) are the coordinates of P_1 and P_n , respectively.

Discussion

Shortest path around There is a fenced area in the two-dimensional Euclidean plane in the shape of a convex polygon with vertices at points $P_1(x_1, y_1)$, $P_2(x_2, y_2)$, ..., $P_n(x_n, y_n)$ (not necessarily in this order). There are two more points, $A(x_A, y_A)$ and $B(x_B, y_B)$, such that $x_A < \min\{x_1, x_2, \dots, x_n\}$ and $x_B > \max\{x_1, x_2, \dots, x_n\}$. Design a reasonably efficient algorithm for computing the length of the shortest path between A and B . [ORo98], p.68

Discussion

Shortest path around There is a fenced area in the two-dimensional Euclidean plane in the shape of a convex polygon with vertices at points $P_1(x_1, y_1)$, $P_2(x_2, y_2), \dots, P_n(x_n, y_n)$ (not necessarily in this order). There are two more points, $A(x_A, y_A)$ and $B(x_B, y_B)$, such that $x_A < \min\{x_1, x_2, \dots, x_n\}$ and $x_B > \max\{x_1, x_2, \dots, x_n\}$. Design a reasonably efficient algorithm for computing the length of the shortest path between A and B . [ORo98], p.68

Find the upper and lower hulls of the set $\{A, B, P_1, \dots, P_n\}$ (e.g., by quickhull), compute their lengths (by summing up the lengths of the line segments making up the polygonal chains) and return the smaller of the two.