# Closest pair

Abhiram Ranade

7 January 2016

# Closest pair in the plane

Input: $n$ points in the plane.
Output: Pair of points that are closest among those given.

Closest = smallest Euclidean distance.

Useful in graphics, geographical information systems, air traffic control..

Brute force: Check all pairs. Report closest. Time $= \theta(n^2)$.

One dimensional version: Points on a line.
Easy $O(n \log n)$ time algorithm = Sort, take minimum distance between consecutive points.

Two dimensions: Nice divide and conquer algorithm.

# How should we apply divide and conquer?

Idea 1:
Subproblem 1 = first $n/2$ points in given order.
Subproblem 2 = last $n/2$.

Idea 2:
Subproblem 1 = Points falling in "left half" of region.
Subproblem 2 = Points falling in "right half" of region.
Width of left half = width of right half

Idea 3:
Subproblem 1 = Leftmost $n/2$ points
Subproblem 2 = Rightmost $n/2$ points
Assume all $x$ coordinates are distinct.

How to decide: Equal sized problems? Overhead of splitting? Which will be easier to combine? Try all?

# The algorithm

We only discuss how to find the distance between the closest pair. The closest pair can be found with minor additional bookkeeping.

1. Divide: Draw vertical line *Sep* with $n/2$ points on each side.
2. Conquer: Recursively find
   $\delta_L$ : closest distance between points to left of *Sep*.
   $\delta_R$ : closest distance between points to right of *Sep*.
3. Combine:
   Check if there can be points closer than $\delta = \min(\delta_L, \delta_R)$.
   Return appropriately.

Details soon.

$$T(n) = T_{divide} + 2T(n/2) + T_{combine}$$

Divide: sorting. $T_{divide} = O(n \log n)$.

# The combine step

We must somehow find the minimum over all pairs of points.

The conquer steps considered pairs such that

- both points are to the left of *Sep*
- both points are to the right of *Sep*

What remains: pairs in which one point is on either side of *Sep*.

We could find $\delta_B =$ the closest distance between $p_i, p_j$ s.t. $p_i$ is on left, and $p_j$ is on right.
Do we really need to?

If $\delta_B \geq \delta = \min(\delta_L, \delta_R)$, we dont need to know its value.

Observation: We only need to consider points which are within distance $\delta$ of *Sep*.

Observation: This vertical strip problem is almost one dimensional..

# The combine step (contd.)

1. Sort the points in the vertical strip by their $y$ coordinate.
2. For $i = 1$ to number of points
3. $\quad \delta_i =$ minimum distance between $p_i$ and $p_{i+1}, \ldots, p_{i+11}$.
4. Report $\delta_c = \min_i \delta_i$

Thm: If distance$(p_i, p_j) < \delta$ then $|i - j| \leq 11$.
Proof: Divide the strip into squares of width $\delta/2$.
Every such square can contain at most 1 point.
If distance $(p_i, p_j) < \delta$, then they can be separated by at most one rows of squares.
$p_i, p_j$ together belong to 3 consecutive rows.
There must be $< 10$ points between them.

$\square$

The number 11 can be reduced with better analysis..

$T_{combine} = O(n \log n)$

# Summary

$$T(n) = O(n \log n) + 2T(n/2)$$

Solves to: $T(n) = O(n \log^2 n)$

Improvements Possible: Can sort points once for all into array $X$ by x coordinate and array $Y$ by y coordinate.
Pass these arrays to recursive calls and eliminate sorting.

$$T(n) = T_{presort} + T_{rec}(n)$$
$$T_{rec}(n) = O(n) + 2T_{rec}(n/2)$$
$$T_{rec}(n) = O(n \log n)$$

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