CSE331 Introduction to Algorithm
Lecture 6: Closest Pair of Points

Antoine Vigneron
antoine@unist.ac.kr

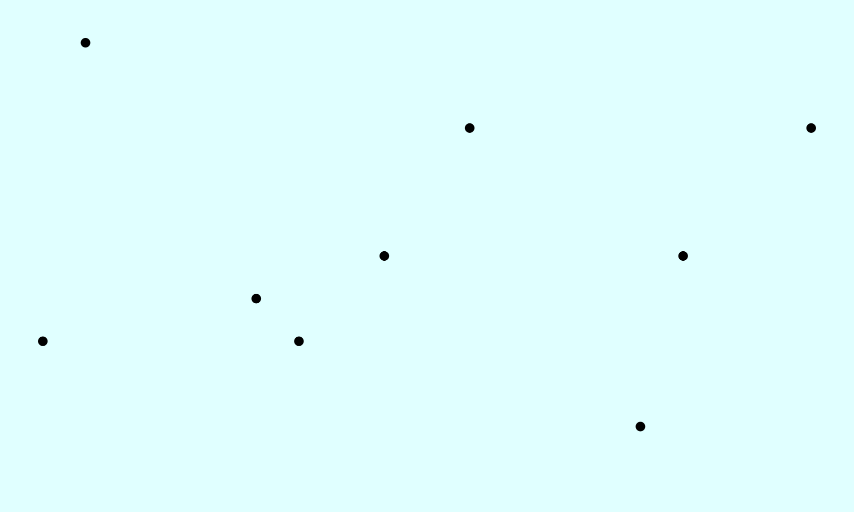
Ulsan National Institute of Science and Technology

July 11, 2017

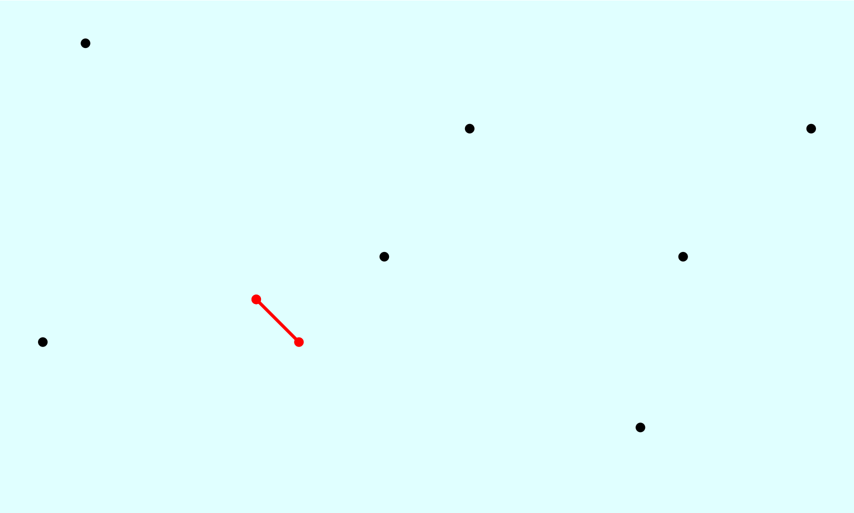
Introduction

- Reference: Section 33.4 of the textbook Introduction to Algorithms by Cormen, Leiserson, Rivest and Stein.
  - I modified the algorithm slightly.

Problem Statement



Problem Statement



Problem Statement

Problem (Closest Pair)

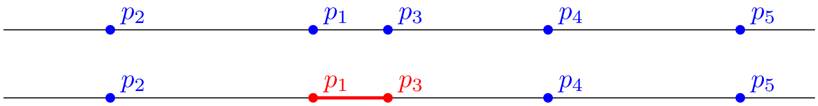
Given a set P of n points in the plane, the closest pair problem is to find two points $p^*, q^* \in P$ such that their distance $\delta^* = d(p^*, q^*)$ is minimum.

It can also be stated as follows:

- INPUT: A set of points $\{p_1, \dots, p_n\}$ in the plane
- OUTPUT: A pair (p_i, p_j) such that $i < j$ and $d(p_i, p_j) \leqslant d(p_k, p_\ell)$ for every $k \neq \ell$

- Applications: Air traffic control (in order to detect potential collisions), ...

One-Dimensional Version



Property

The two closest points are adjacent.

- So we can just sort P, and scan from left to right.

One-Dimensional Version

Pseudocode

procedure $\textsc{1DClosestPair}(P = \{p_1, \ldots, p_n\})$
    $Q[1 \ldots n] \leftarrow P$ in sorted order
    $q \leftarrow Q[1]$, $r \leftarrow Q[2]$
    for $i \leftarrow 2, n-1$ **do**
        if $d(Q[i], Q[i+1]) < d(q, r)$ **then**
            $q \leftarrow Q[i]$, $r \leftarrow Q[i+1]$
    return (q, r)

- **Analysis**: Using MERGE SORT, it takes $O(n) + \Theta(n \log n) = \Theta(n \log n)$ time.

Brute Force Approach

Pseudocode

```
procedure SlowClosestPair(P = {p_1, ..., p_n})
    a ← 1, b ← 2
    for i ← 1, n − 1 do
        for j ← i + 1, n do
            if d(p_i, p_j) < d(p_a, p_b) then
                a ← i, b ← j
    return (p_a, p_b)
```

- Running time: $\Theta(n^2)$

Brute Force Approach

- Line 5 implementation: We can use the formula

$$d(p_i, p_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

  where (x_i, y_i) denote the coordinates of p_i.
- Or compare the *squares* of the distances

$$(x_i - x_j)^2 + (y_i - y_j)^2 < (x_a - x_b)^2 + (y_a - y_b)^2$$

  which does not require the square root function.
- We can also store the distance $d(p_a, p_b)$ (or its square) and update it each time a, b is updated, so that we don't need to recompute it each time the test from Line 6 is executed.
- In any case the running time remains $\Theta(n^2)$.

Divide and Conquer Approach

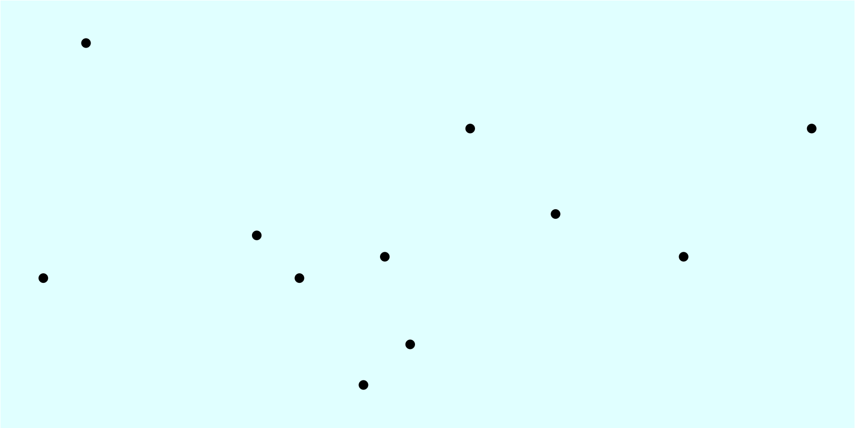


Figure: Input point set P.

Divide and Conquer Approach

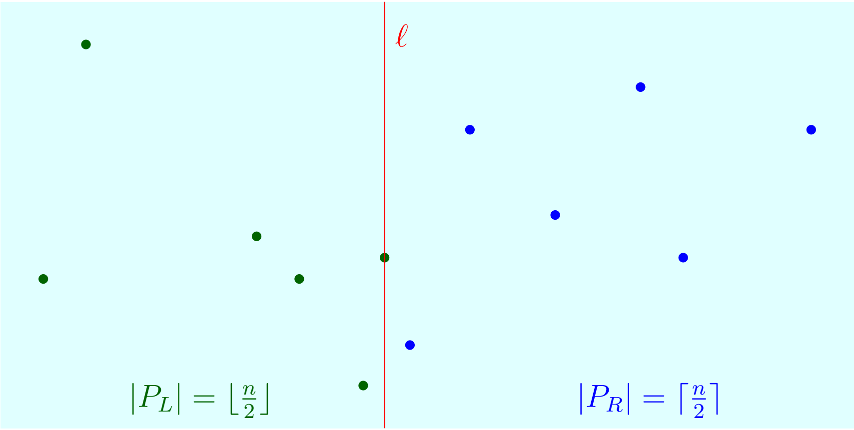


Figure: Split P evenly using a vertical line ℓ.

Divide and Conquer Approach

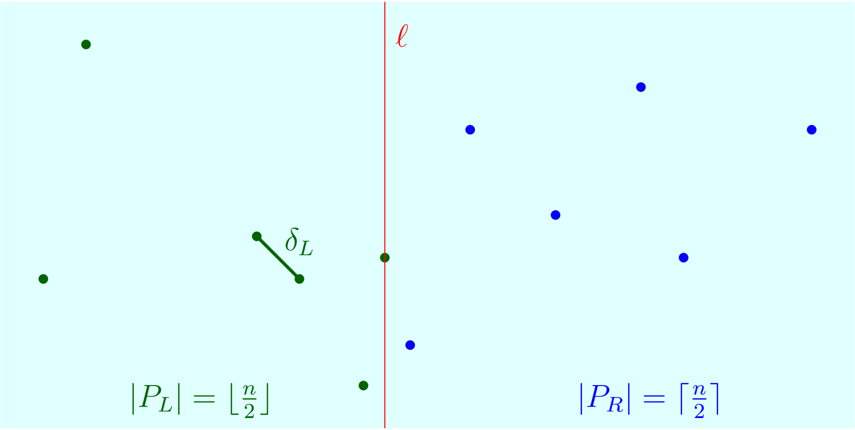


Figure: Compute recursively the closest pair in P_L.

Divide and Conquer Approach

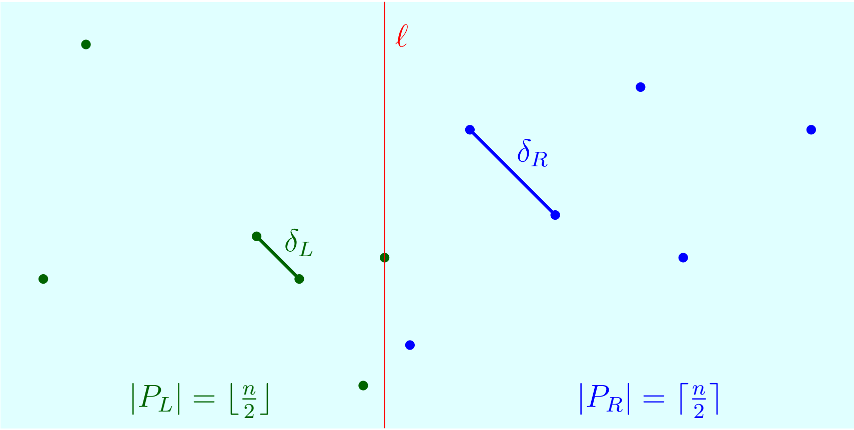


Figure: Compute recursively the closest pair in P_R.

Divide and Conquer Approach

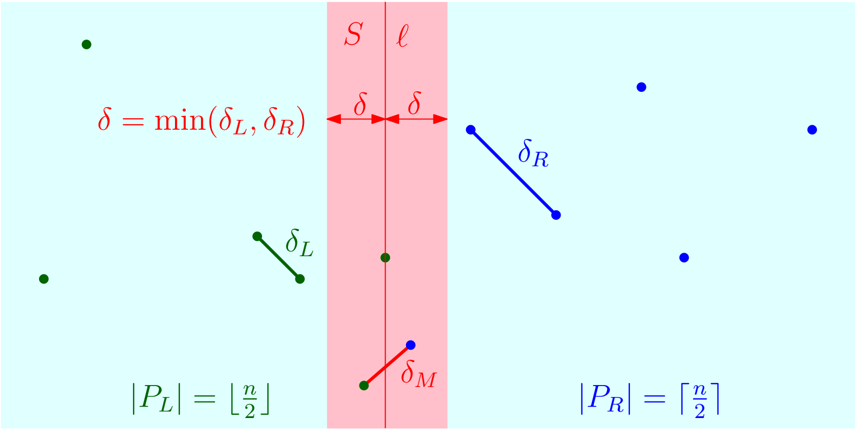


Figure: Compute the closest pair in the vertical strip S around ℓ.

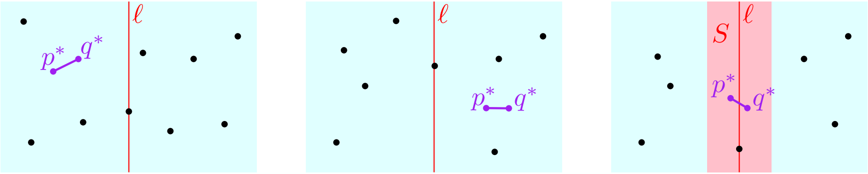
Divide and Conquer Approach

Finding a closest pair of points

- If $n \leqslant 4$, solve the problem by brute force. (Base case)
- Otherwise:
  1. Find a vertical line ℓ such that splits P evenly into two sets P_L and P_R of size at most $\lceil n/2 \rceil$ each.
  2. Compute recursively the closest pair distance δ_L in P_L.
  3. Compute recursively the closest pair distance δ_R in P_R.
  4. Let $\delta = \min(\delta_L, \delta_R)$, and let δ_M be the closest pair distance in the strip S of width 2δ centered at ℓ. If $\delta_M < \delta$, compute δ_M and return the corresponding pair.
  5. Otherwise, the closest pair distance in P is $\delta^* = \delta$. Return the corresponding pair.

- Idea: Step 4 deals with a narrow vertical strip, so it is almost like the 1D case, and thus we may be able to solve it quickly.

Proof of Correctness



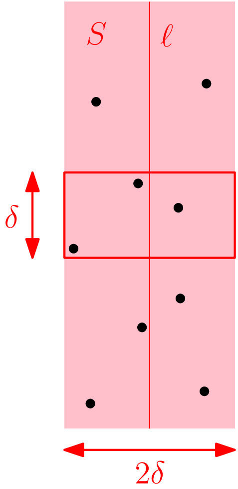
Case 1 Case 2 Case 3

- Let p^*, q^* be a closest pair and $\delta^* = d(p^*, q^*)$.
- Then we are in one of the three cases below:
  1. $p^* \in P_L$ and $q^* \in P_L$. Then $\delta^* = \delta_L$ and $\delta^* \leqslant \delta_R$. Then our algorithm returns $\delta = \min(\delta_L, \delta_R) = \delta^*$.
  2. $p^* \in P_R$ and $q^* \in P_R$. Similar to previous case.
  3. $p^* \in P_L$ and $q^* \in P_R$, or $p^* \in P_R$ and $q^* \in P_L$. Then the segment p^*q^* intersects ℓ. We know that $\delta^* \leqslant \delta_L$ and $\delta^* \leqslant \delta_R$, so $\delta^* \leqslant \delta$. As this segment has length $\delta^* \leqslant \delta$, it follows that p^* and q^* lie in S, and thus $\delta^* = \delta_M$. In this case our algorithm either returns δ_M, or it returns δ when $\delta_M = \delta$, which is the correct answer.

Handling the Strip S

Lemma

Within any $2\delta \times \delta$ box in the strip S, there are at most 8 points.



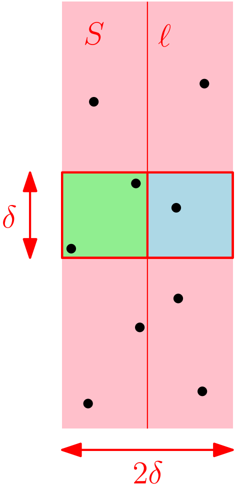
Proof.

Any two points in P_L are at distance at least $\delta_L \geqslant \delta$. □

Handling Strip S

Lemma

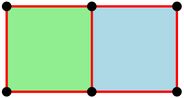
Within any $2\delta \times \delta$ box in the strip S, there are at most 8 points.



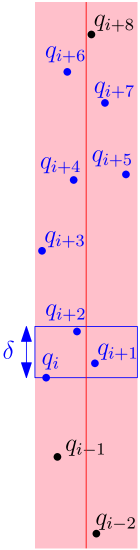
Proof.

Any two points in P_L are at distance at least $\delta_L \geqslant \delta$. So there are at most 4 points of P_L in the green square. Similarly, there are $\leqslant 4$ points of P_R in the blue square. □

Worst case: the two points in the middle appear twice, once in P_L and once in P_R.



Handling the strip S



- Let M denote $P \cap S$.
- We assume that $M = (q_1, \dots, q_m)$ is sorted by y coordinates.
- The lemma above suggests the following approach:
  - For each q_i, compute the 7 distances $d(q_i, q_{i+1}), \dots, d(q_i, q_{i+7})$.
  - Return the closest pair (q_a, q_b) among them.
- It runs in $\Theta(m)$ time, since we only consider $7m$ pairs.
- Proof of correctness: By the lemma, if $j > i + 7$, then m_i and m_j do not lie in the same box, and hence their distance is more than δ.

Handling the strip S

Pseudocode (assuming that M is sorted by y-coordinate)

```
procedure HandleStrip(M = (q_1, ..., q_m))
    if m ≤ 1 then
        return NotFound
    a ← 1, b ← 2
    for i ← 1, m − 1 do
        for j ← i + 1, i + 7 do
            if j ≤ m and d(q_i, q_j) < d(q_a, q_b) then
                a ← i, b ← j
    return (q_a, q_b)
```

- Remark: This is very similar to the 1D algorithm.
- Difference: We check 7 points ahead instead of just 1.

First Version of the Algorithm

- Step 1 can be done as follows:
  - Sort P by x-coordinate into an array $X[1 \dots n]$.
  - Let $r = \lfloor n/2 \rfloor$.
  - The arrays $X_L = X[1 \dots r]$ and $X_R = X[r+1 \dots n]$ record P_L and P_r.
- So it takes $\Theta(n \log n)$ time.
- Step 4 also takes $\Theta(n \log n)$ time if we include the time needed to sort the points by y-coordinates.

Analysis

- So the running time $T(n)$ satisfies the relation:

$$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + \Theta(n \log n).$$

- The master method fails here, neither of the three cases apply.
- It can be shown that $T(n) = \Theta(n \log^2 n)$. (See exercise set 3.)

Faster Implementation

- The $\Theta(n \log n)$ term in the previous slide comes from:
  - Sorting P by x-coordinate.
  - Sorting M by y-coordinate.
- We can replace it with $\Theta(n)$ if we *presort* P into two arrays $X[1 \dots n]$ and $Y[1 \dots n]$, sorted by x and y-coordinates respectively.
- Then at each recursive call, we can split these arrays into sorted arrays $X_L[]$, $X_R[]$, $Y_L[]$, $Y_R[]$ in $\Theta(n)$ time.
- Implementation details are left as an exercise (see Exercise set 3).
- So the recurrence relation becomes

$$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + \Theta(n).$$

- It solves to $T(n) = \Theta(n \log n)$.

Conclusion

Theorem

The closest pair problem can be solved in $O(n \log n)$ time.

- Under a fairly general model of computation, one can prove that this is optimal: Any algorithm takes $\Omega(n \log n)$ in the worst case, even in one dimension. (Not covered in CSE331.)

Conclusion

- This approach applies to several 2D geometric problems: Divide into two parts of size $n/2$ using a vertical line, and handle the objects that cross the line using a 1D algorithm.
- It also applies in dimension $d \geqslant 3$ or higher: use a vertical plane (hyperplane), and near the separating plane, use the $d-1$ dimensional algorithm.
- So this approach combines *divide and conquer*, and *recursion on the dimension* of the problem.
- The closest pair problem is a *Computational Geometry* problem. This is my research field.