The Closest-Pair Problem

Algorithmic Thinking

Luay Nakhleh

Department of Computer Science

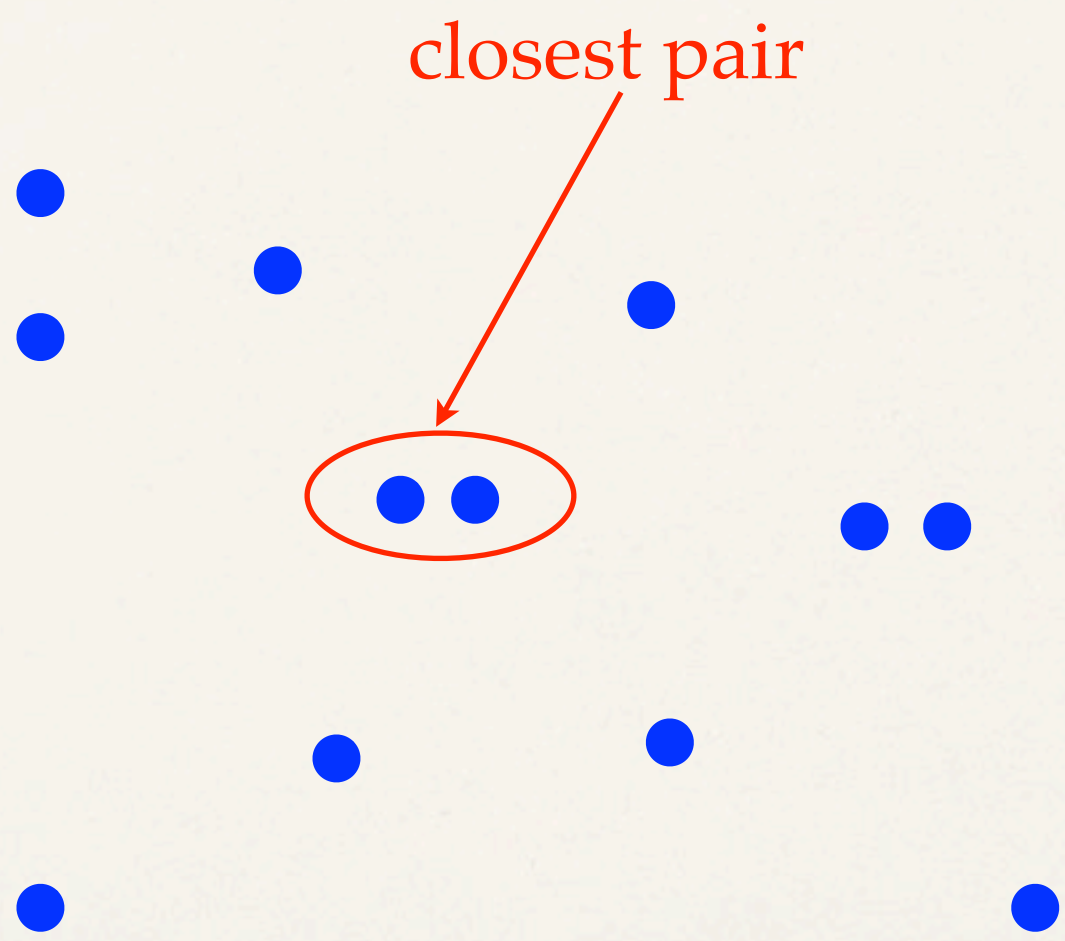
Rice University

The Problem

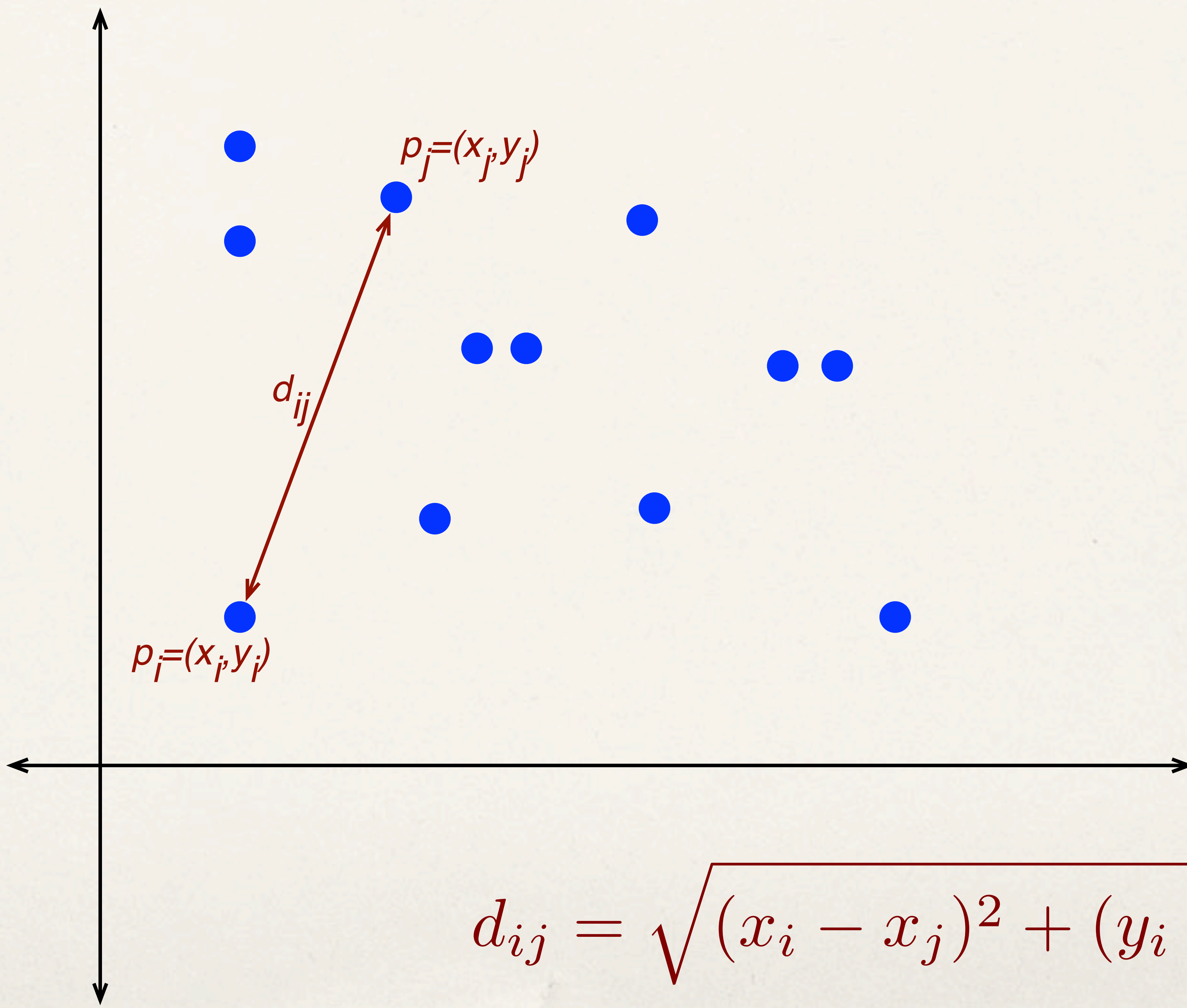
- Input: A set P of (distinct) points in the 2D space, each given by its horizontal (x) and vertical (y) coordinates.

- Output: A pair of points $p_i, p_j \in P$ $(p_i \neq p_j)$ that are closest to each other (under the Euclidian distance).

The Problem



Distance Between Points: The Euclidian Distance



$$d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

First Attempt: Brute-force

- A simple brute-force algorithm for the problem:
  - Compute the distance between every two points
  - Return a pair of points with the smallest distance

Algorithm 3: BFClosestPair.

Input: A set P of (≥ 2) points whose ith point, p_i, is a pair (x_i, y_i).
Output: A tuple (d, i, j) where d is the smallest pairwise distance of points in P, and i, j are the indices of two points whose distance is d.

$(d, i, j) \leftarrow (\infty, -1, -1)$;
foreach $p_u \in P$ **do**
    foreach $p_v \in P$ $(u \neq v)$ **do**
        $(d, i, j) \leftarrow \min\{(d, i, j), (d_{p_u, p_v}, u, v)\}$
return (d, i, j);

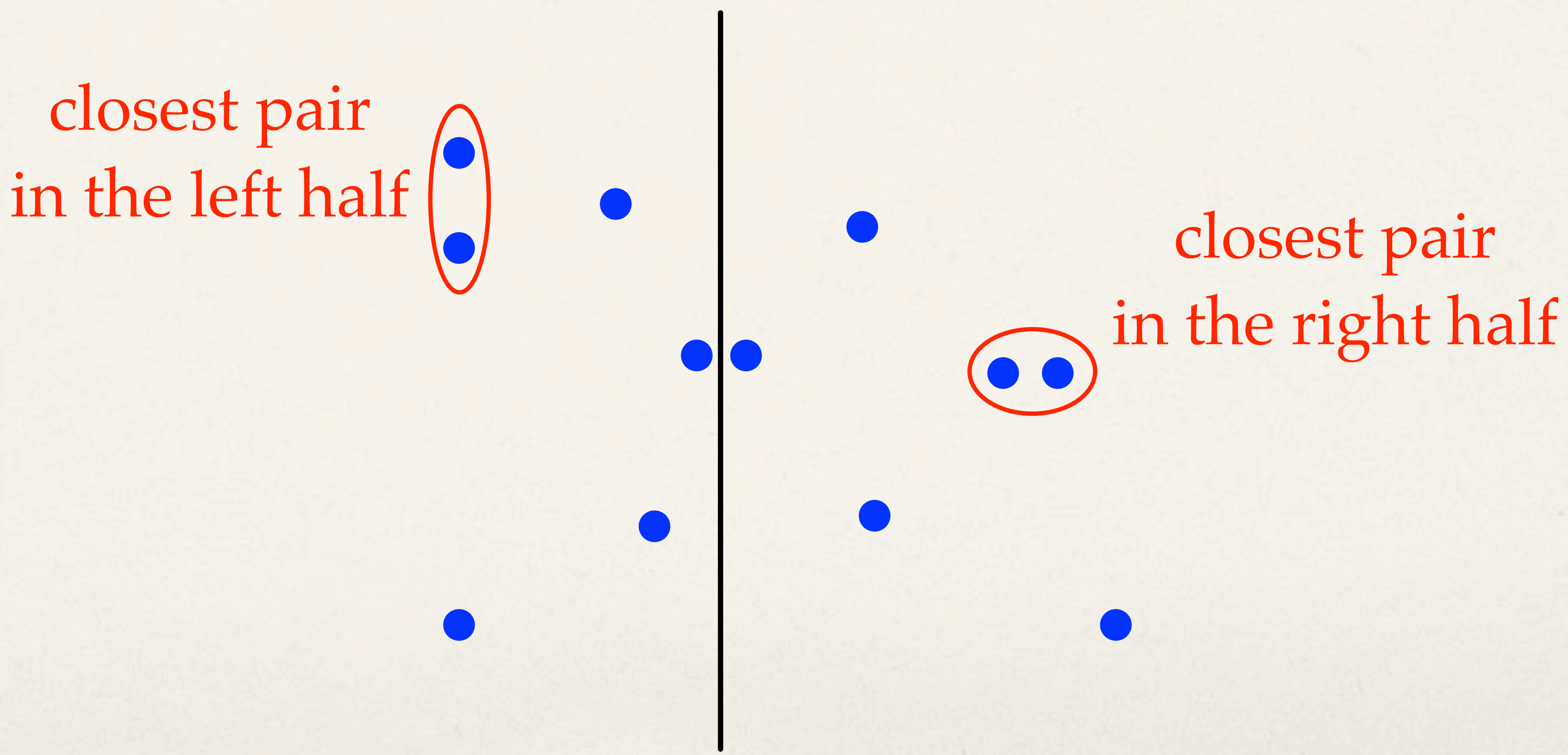
- What is the running time of the brute-force algorithm?
- Can we do better?

Second Attempt: Divide-and-Conquer

The Divide Phase

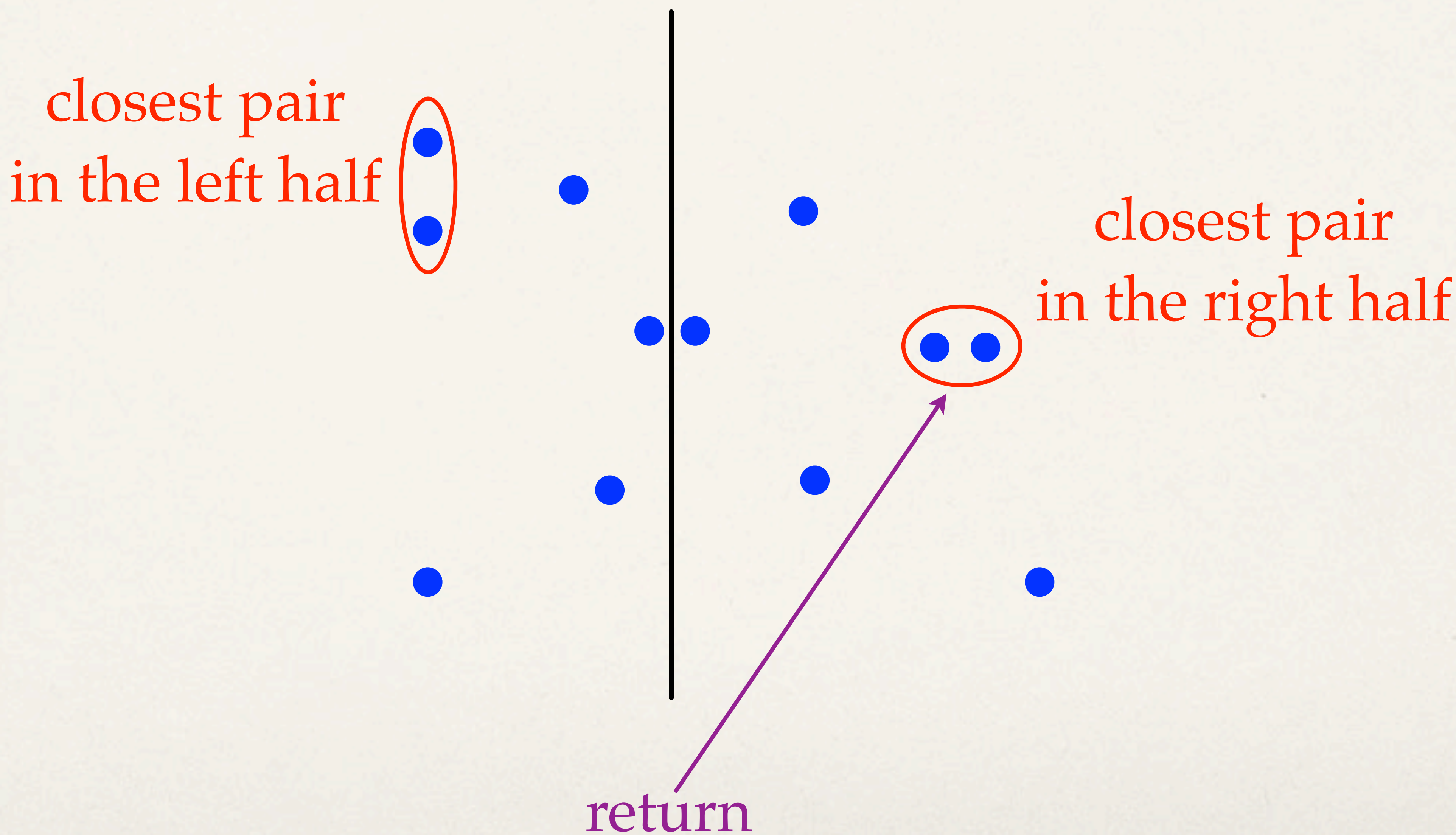
Draw a vertical line so that half of the points are to the left of the line and the other half are to the right of the line

Recursively find a closest pair in each half



The Merge Phase: A First Attempt

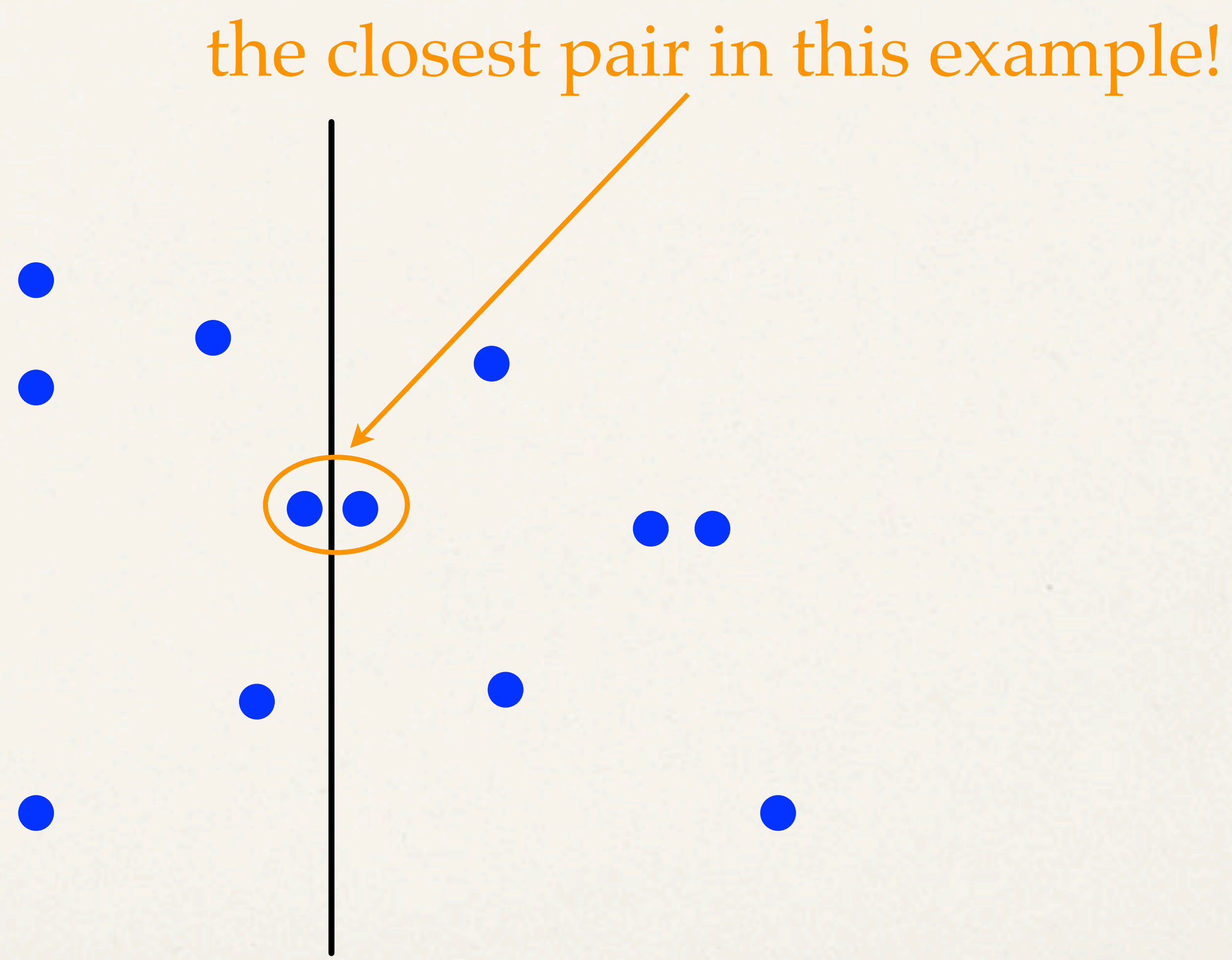
If (p_i,p_j) is a closest pair on the left, and (p_k,p_l) is a closest pair on the right, then return (p_i,p_j) if $d_{ij}<d_{kl}$, and return (p_k,p_l) otherwise.



The Merge Phase: A First Attempt

- This attempt at solving the problem doesn't work if at any iteration of the divide phase, two closest points reside on either side of the vertical line!

The Merge Phase: A First Attempt



The Merge Phase: A Second Attempt

- Suppose we've found a closest pair (p_i,p_j) on the left, and a closest pair (p_k,p_l) on the right
- Now, compute the distance between every point on the left and every point on the right, and find a closest such pair of points, say (p_r,p_s)
- Finally, return a pair out of the three that has the smallest distance, and we're done!

The Merge Phase: A Second Attempt

- The problem with this second attempt is that it is basically the brute-force algorithm!
- Can we do the Merge phase more efficiently?
- The answer is yes!

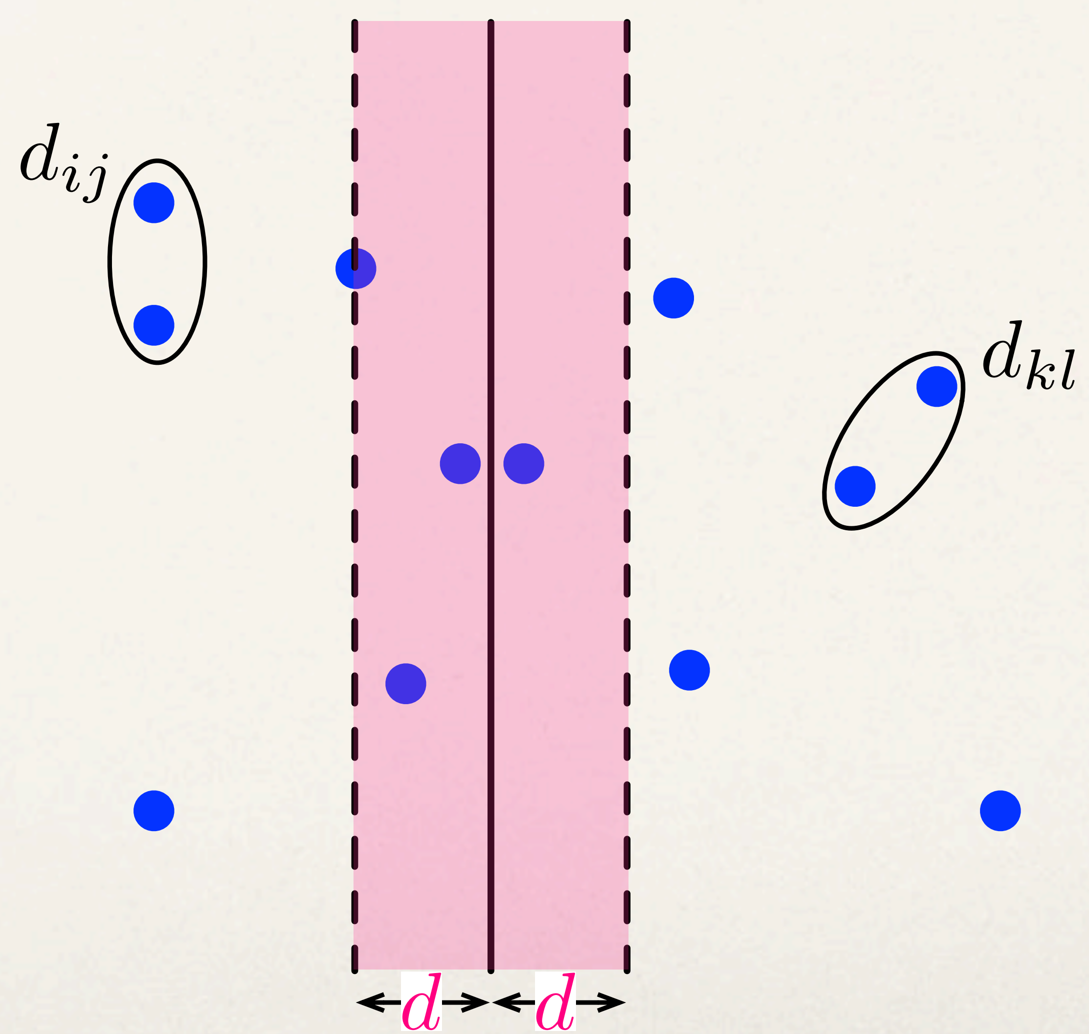
The Merge Phase: A Third Attempt

- Key observation 1:
  - If d_{ij} is the smallest pairwise distance on the left and d_{kl} is the smallest pairwise distance on the right, then we need to consider points on either side of the line that are at most $d=\min\{d_{ij},d_{kl}\}$ distance apart.

The Merge Phase: A Third Attempt

- Key observation 1:

$$d = \min\{d_{ij}, d_{kl}\}$$

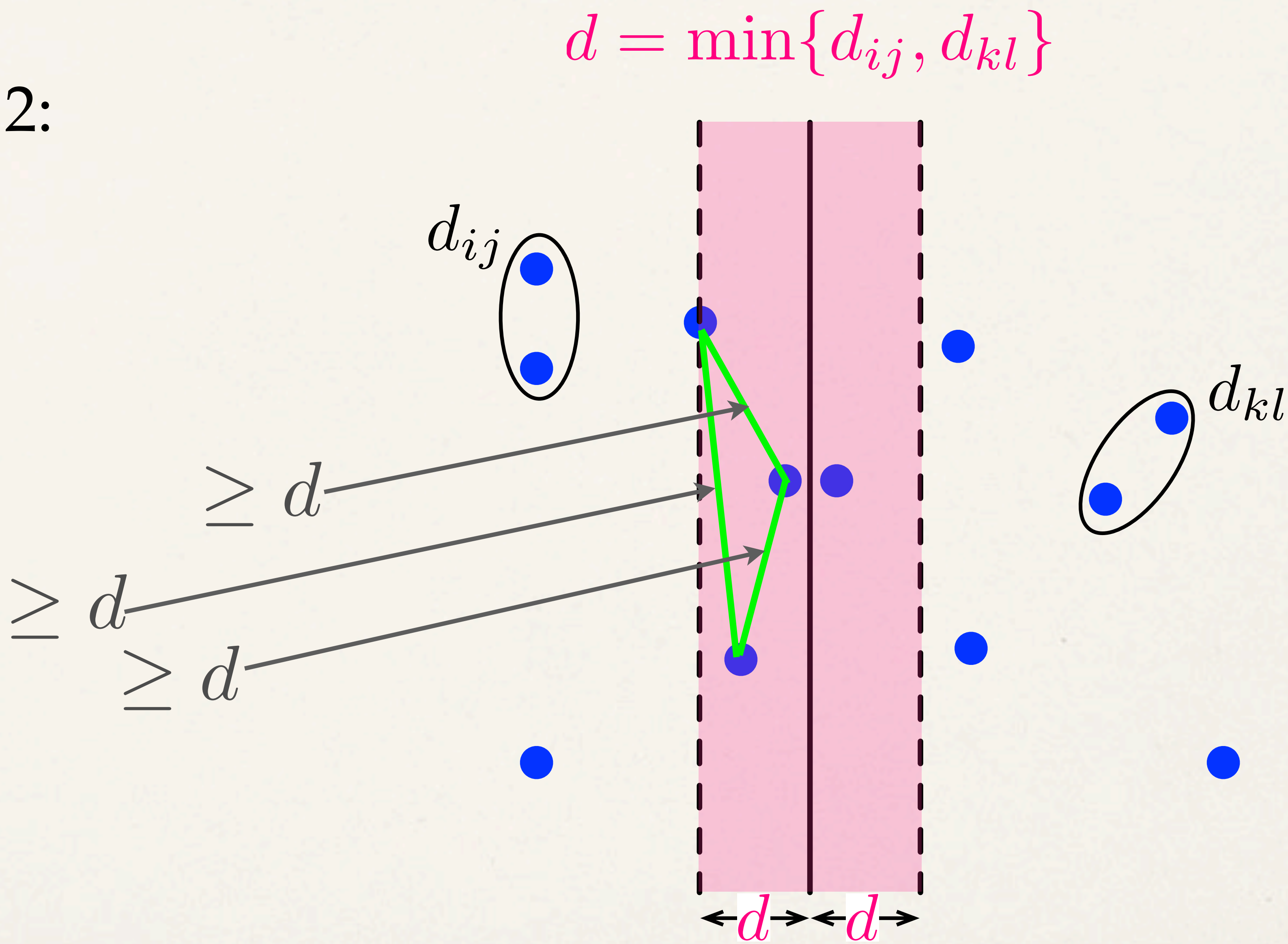


The Merge Phase: A Third Attempt

- Key observation 2:
  - In each half (left and right) of the (shaded) rectangle, every pair of points is at least distance d apart (why?).

The Merge Phase: A Third Attempt

- Key observation 2:



Due to this observation, non-consecutive points in each half of the rectangle must be separated by some minimum vertical distance.

The Merge Phase: A Third Attempt

- Consequence of observation 2:
  - Let $S[0..m\text{-}1]$ be an array (or a list) of all the points inside the rectangle sorted by nondecreasing order of their y-coordinates. Then, if the distance between $S[i]$ and $S[j]$ is smaller than d, then $|i\text{-}j| < 4$.
  - In other words, given a point p_i in either half (left of right) of the rectangle, if there is a point p_j in the same half whose distance to p_i is smaller than d, then p_j is either one of the first three points "above" p_i or the first three points "below" p_i.

The Merge Phase: A Third Attempt

- Putting all three observations together gives the idea for efficiently finding a closest pair across the vertical line:
  - Identify all points with x-coordinate with distance d from the vertical line.
  - Let $S[0..m\text{-}1]$ be an array (or a list) of all these points sorted by nondecreasing order of their y-coordinates.
  - Going through the element of S in order, for each element $S[i]$ inspect the next three ones to find the closest to $S[i]$, and record the pair of indices i and j with correspond to the closest pair thus found.

- And, now to the full algorithm...

Algorithm 4: SlowDCClosestPair.

Input: A set P of (≥ 2) points whose ith point, p_i, is a pair (x_i, y_i).
Output: A tuple (d, i, j) where d is the smallest pairwise distance of the points in P, and i, j are the indices of two points whose distance is d.

base case

$n \leftarrow |P|$;
if $n \leq 3$ **then**
  return BFClosestPair(P);
else
  Let H be a sorted list of the points in P in nondecreasing order of their horizontal (x) coordinates;
  $m \leftarrow \lceil n/2 \rceil$; `// the number of points in each half`
  $mid \leftarrow \frac{1}{2}(x_{H[m-1]} + x_{H[m]})$; `// the horizontal coordinate of the vertical dividing line`
  $P_\ell \leftarrow \{H[i] : 0 \leq i \leq m-1\}$; $P_r \leftarrow \{H[i] : m \leq i \leq n-1\}$;
  $(d_\ell, i_\ell, j_\ell) \leftarrow \mathbf{SlowDCClosestPair}(P_\ell)$;
  $(d_r, i_r, j_r) \leftarrow \mathbf{SlowDCClosestPair}(P_r)$;
  $(d, i, j) \leftarrow \min\{(d_\ell, i_\ell, j_\ell), (d_r, i_r, j_r)\}$; `// min is based on the first element of the tuple`
  Let S be a list of the set $\{p_i : |x_i - mid| < d\}$ **sorted** in nondecreasing order of their vertical (y) coordinates;
  Let k be the number of elements in S;
  for $u \leftarrow 0$ **to** $k-2$ **do**
    for $v \leftarrow u+1$ **to** $\min\{u+3, k-1\}$ **do**
      $(d, i, j) \leftarrow \min\{(d, i, j), (d_{S[u],S[v]}, S[u], S[v])\}$;
return (d, i, j);

divide

merge

- Algorithm **SlowDCClosestPair** sorts lists in every recursive call.
- Q: Can we avoid this?
- A: Yes, by passing indices of points to the recursive calls.

Algorithm 5: FastDCClosestPair.

Input: A set P of (≥ 2) points whose ith point, p_i, is a pair (x_i, y_i); two lists H and V such that H contains the indices of the points sorted in nondecreasing order of their horizontal (x) coordinates, and V contains the indices of the points sorted in nondecreasing order of their vertical (y) coordinates.

Output: A tuple (d, i, j) where d is the smallest pairwise distance of the points in P, and i, j are the indices of two points whose distance is d.

Let n be the number of elements in H;
if $n \leq 3$ **then**
    $Q \leftarrow \{p_{H[i]} : 0 \leq i \leq n-1\}$;
    return BFClosestPair(Q);
else
    $m \leftarrow \lceil n/2 \rceil$; `// the number of points in each half`
    $mid \leftarrow \frac{1}{2}(x_{H[m-1]} + x_{H[m]})$; `// the horizontal coordinate of the vertical dividing line`
    $H_\ell \leftarrow H[0..m-1]$; $H_r \leftarrow H[m..n-1]$;
    Copy to V_ℓ, in order, the elements $V[i]$ that are elements of H_ℓ;
    Copy to V_r, in order, the elements $V[i]$ that are elements of H_r;
    $(d_\ell, i_\ell, j_\ell) \leftarrow$ **FastDCClosestPair**(P, H_ℓ, V_ℓ); `// Use the original P`
    $(d_r, i_r, j_r) \leftarrow$ **FastDCClosestPair**(P, H_r, V_r); `// Use the original P`
    $(d, i, j) \leftarrow \min\{(d_\ell, i_\ell, j_\ell), (d_r, i_r, j_r)\}$; `// min is based on the first element of the tuple`
    Copy to S, in order, every $V[i]$ for which $|x_{V[i]} - mid| < d$;
    Let k be the number of elements in S;
    for $u \leftarrow 0$ **to** $k-2$ **do**
        for $v \leftarrow u+1$ **to** $\min\{u+3, k-1\}$ **do**
            $(d, i, j) \leftarrow \min\{(d, i, j), (d_{S[u],S[v]}, S[u], S[v])\}$;
return (d, i, j);

base case

divide

merge