Divide and Conquer: Closest Pair of Points

Madhavan Mukund

https://www.cmi.ac.in/~madhavan

Programming, Data Structures and Algorithms using Python

Week 8

Recall: Video Game

- Several objects on screen
- Basic step: find closest pair of objects

Recall: Video Game

- Several objects on screen
- Basic step: find closest pair of objects
- n objects — naive algorithm is n^2
  - For each pair of objects, compute their distance
  - Report minimum distance across all pairs

Recall: Video Game

- Several objects on screen
- Basic step: find closest pair of objects
- n objects — naive algorithm is n^2
  - For each pair of objects, compute their distance
  - Report minimum distance across all pairs
- There is a clever algorithm that takes time $n \log n$

Recall: Video Game

- Several objects on screen
- Basic step: find closest pair of objects
- n objects — naive algorithm is n^2
  - For each pair of objects, compute their distance
  - Report minimum distance across all pairs
- There is a clever algorithm that takes time $n \log n$
- Use divide and conquer

The problem statement

- Points p in 2D — $p = (x, y)$

The problem statement

- Points p in 2D — $p = (x, y)$
- Usual Euclidean distance between $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$
  - $d(p_1, p_2) = \sqrt{(y_2 - y_1)^2 + (x_2 - x_1)^2}$

The problem statement

- Points p in 2D — $p = (x, y)$
- Usual Euclidean distance between $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$
  - $d(p_1, p_2) = \sqrt{(y_2 - y_1)^2 + (x_2 - x_1)^2}$
- Given n points $p_1, p_2, \ldots, p_n$, find the closest pair
  - Assume no two points have same x or y coordinate
  - Can always rotate points slightly to ensure this

The problem statement

- Points p in 2D — $p = (x, y)$
- Usual Euclidean distance between $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$
  - $d(p_1, p_2) = \sqrt{(y_2 - y_1)^2 + (x_2 - x_1)^2}$
- Given n points $p_1, p_2, \ldots, p_n$, find the closest pair
  - Assume no two points have same x or y coordinate
  - Can always rotate points slightly to ensure this
  - . . . or modify the algorithm slightly!

The problem statement

- Points p in 2D — $p = (x, y)$
- Usual Euclidean distance between $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$
  - $d(p_1, p_2) = \sqrt{(y_2 - y_1)^2 + (x_2 - x_1)^2}$
- Given n points $p_1, p_2, \ldots, p_n$, find the closest pair
  - Assume no two points have same x or y coordinate
  - Can always rotate points slightly to ensure this
  - ...or modify the algorithm slightly!
- Brute force
  - Compute $d(p_i, p_j)$ for every pair of points
  - $O(n^2)$

Finding the closest pair of points

In 1 dimension

Finding the closest pair of points

In 1 dimension

- Given n 1D points $x_1, x_2, \ldots, x_n$, find the closest pair
  - $d(p_i, p_j) = |x_j - x_i|$

Finding the closest pair of points

In 1 dimension

- Given n 1D points $x_1, x_2, \ldots, x_n$, find the closest pair
  - $d(p_i, p_j) = |x_j - x_i|$
- Sort the points — $O(n \log n)$

Finding the closest pair of points

In 1 dimension

- Given n 1D points $x_1, x_2, \ldots, x_n$, find the closest pair
  - $d(p_i, p_j) = |x_j - x_i|$
- Sort the points — $O(n \log n)$
- In sorted order, nearest points to p are its neighbours
  - $O(n)$ scan to find minimum separation between adjacent points

Finding the closest pair of points

In 1 dimension

- Given n 1D points $x_1, x_2, \ldots, x_n$, find the closest pair
  - $d(p_i, p_j) = |x_j - x_i|$
- Sort the points — $O(n \log n)$
- In sorted order, nearest points to p are its neighbours
  - $O(n)$ scan to find minimum separation between adjacent points

In 2 dimensions

- Divide and conquer

Finding the closest pair of points

In 1 dimension

- Given n 1D points $x_1, x_2, \ldots, x_n$, find the closest pair
  - $d(p_i, p_j) = |x_j - x_i|$
- Sort the points — $O(n \log n)$
- In sorted order, nearest points to p are its neighbours
  - $O(n)$ scan to find minimum separation between adjacent points

In 2 dimensions

- Divide and conquer
- Split the points into two halves by vertical line

Finding the closest pair of points

In 1 dimension

- Given n 1D points $x_1, x_2, \ldots, x_n$, find the closest pair
  - $d(p_i, p_j) = |x_j - x_i|$
- Sort the points — $O(n \log n)$
- In sorted order, nearest points to p are its neighbours
  - $O(n)$ scan to find minimum separation between adjacent points

In 2 dimensions

- Divide and conquer
- Split the points into two halves by vertical line
- Recursively compute closest pair in each half

Finding the closest pair of points

In 1 dimension

- Given n 1D points $x_1, x_2, \ldots, x_n$, find the closest pair
  - $d(p_i, p_j) = |x_j - x_i|$
- Sort the points — $O(n \log n)$
- In sorted order, nearest points to p are its neighbours
  - $O(n)$ scan to find minimum separation between adjacent points

In 2 dimensions

- Divide and conquer
- Split the points into two halves by vertical line
- Recursively compute closest pair in each half
- Compare shortest distance in each half to shortest distance across the dividing line

Finding the closest pair of points

In 1 dimension

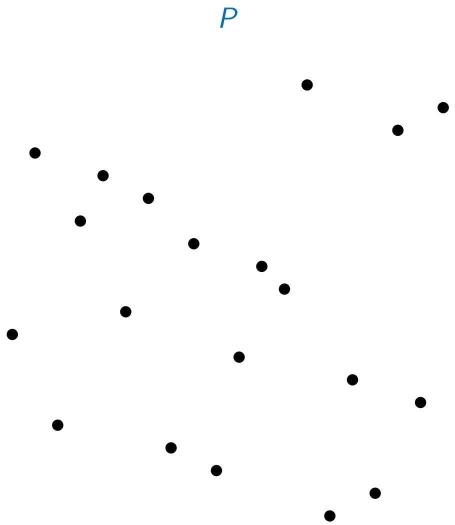
- Given n 1D points $x_1, x_2, \ldots, x_n$, find the closest pair
  - $d(p_i, p_j) = |x_j - x_i|$
- Sort the points — $O(n \log n)$
- In sorted order, nearest points to p are its neighbours
  - $O(n)$ scan to find minimum separation between adjacent points

In 2 dimensions

- Divide and conquer
- Split the points into two halves by vertical line
- Recursively compute closest pair in each half
- Compare shortest distance in each half to shortest distance across the dividing line
- How to do this efficiently?

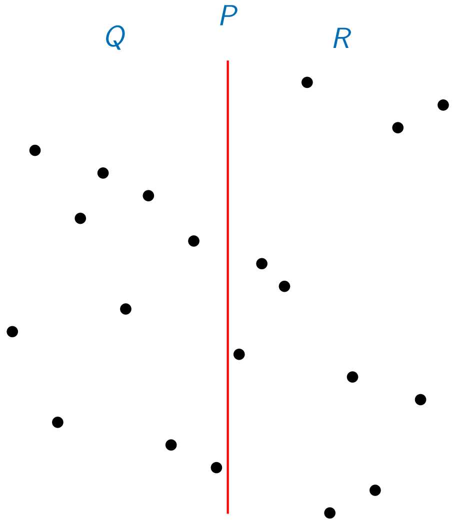
Dividing points

- Given n points $P = \{p_1, p_2, \ldots, p_n\}$, compute
  - P_x, P sorted by x-coordinate
  - P_y, P sorted by y-coordinate



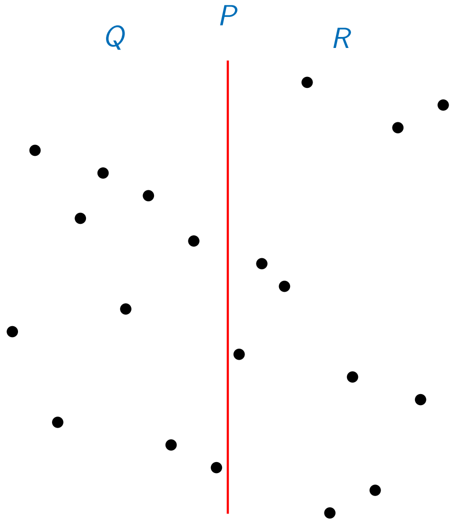
Dividing points

- Given n points $P = \{p_1, p_2, \ldots, p_n\}$, compute
  - P_x, P sorted by x-coordinate
  - P_y, P sorted by y-coordinate
- Divide P by vertical line into equal size Q, R



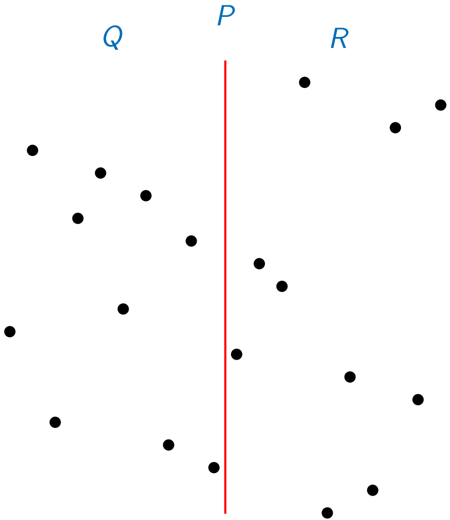
Dividing points

- Given n points $P = \{p_1, p_2, \ldots, p_n\}$, compute
  - P_x, P sorted by x-coordinate
  - P_y, P sorted by y-coordinate
- Divide P by vertical line into equal size Q, R
- How to compute Q_x, Q_y, R_x, R_y efficiently?



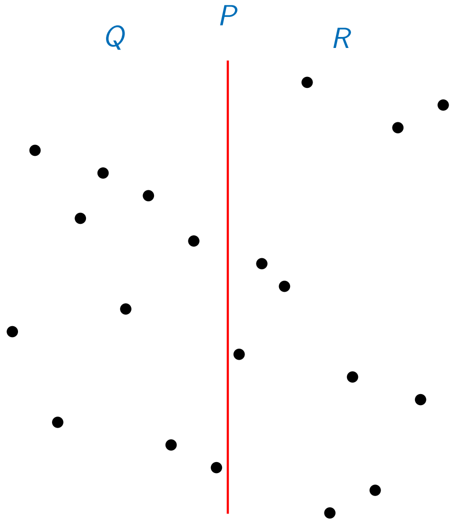
Dividing points

- Given n points $P = \{p_1, p_2, \ldots, p_n\}$, compute
  - P_x, P sorted by x-coordinate
  - P_y, P sorted by y-coordinate
- Divide P by vertical line into equal size Q, R
- How to compute Q_x, Q_y, R_x, R_y efficiently?
- Q_x is first half of P_x, R_x is second half of P_x



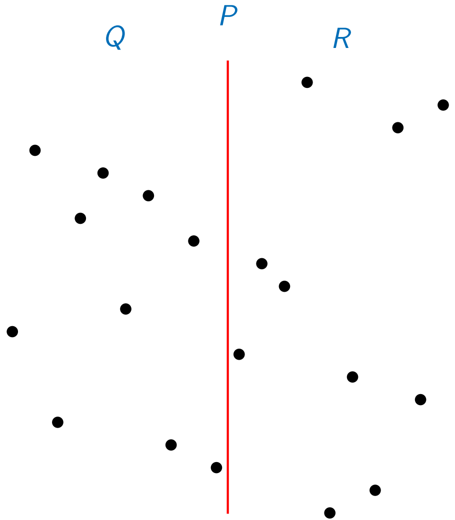
Dividing points

- Given n points $P = \{p_1, p_2, \ldots, p_n\}$, compute
  - P_x, P sorted by x-coordinate
  - P_y, P sorted by y-coordinate
- Divide P by vertical line into equal size Q, R
- How to compute Q_x, Q_y, R_x, R_y efficiently?
- Q_x is first half of P_x, R_x is second half of P_x
- Let x_R be smallest x coordinate in R



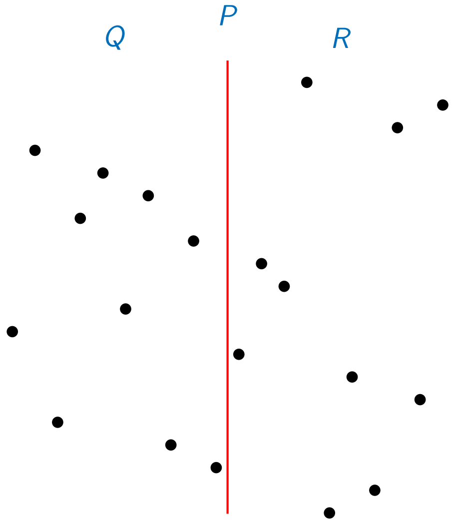
Dividing points

- Given n points $P = \{p_1, p_2, \ldots, p_n\}$, compute
  - P_x, P sorted by x-coordinate
  - P_y, P sorted by y-coordinate
- Divide P by vertical line into equal size Q, R
- How to compute Q_x, Q_y, R_x, R_y efficiently?
- Q_x is first half of P_x, R_x is second half of P_x
- Let x_R be smallest x coordinate in R
- For $p \in P_y$, if x coordinate of p less than x_R, move p to Q_y, else R_y



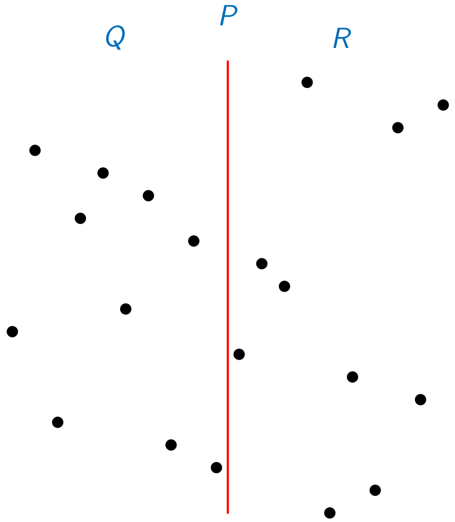
Dividing points

- Given n points $P = \{p_1, p_2, \ldots, p_n\}$, compute
  - P_x, P sorted by x-coordinate
  - P_y, P sorted by y-coordinate
- Divide P by vertical line into equal size Q, R
- How to compute Q_x, Q_y, R_x, R_y efficiently?
- Q_x is first half of P_x, R_x is second half of P_x
- Let x_R be smallest x coordinate in R
- For $p \in P_y$, if x coordinate of p less than x_R, move p to Q_y, else R_y
- All of this can be done in $O(n)$



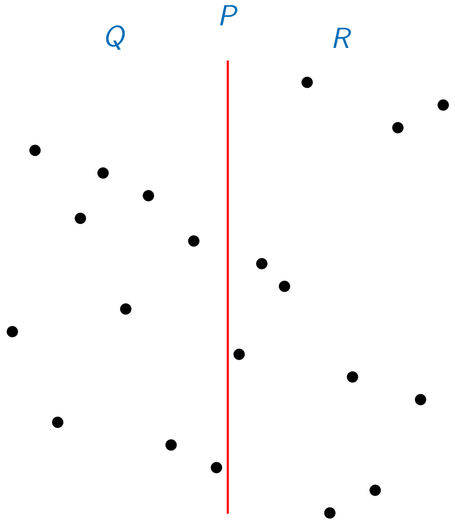
Divide and conquer

- Want to compute `ClosestPair`(P_x, P_y)



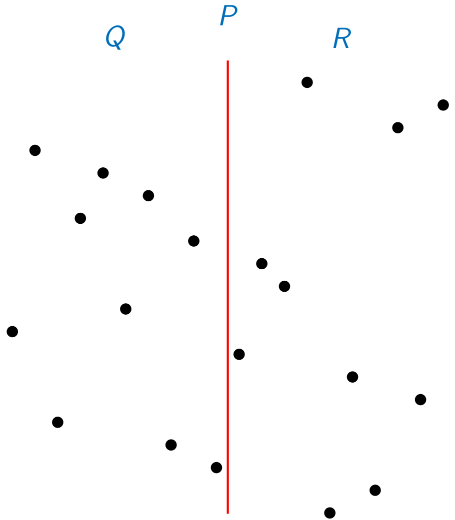
Divide and conquer

- Want to compute `ClosestPair`(P_x, P_y)
- Split (P_x, P_y) as (Q_x, Q_y), (R_x, R_y)



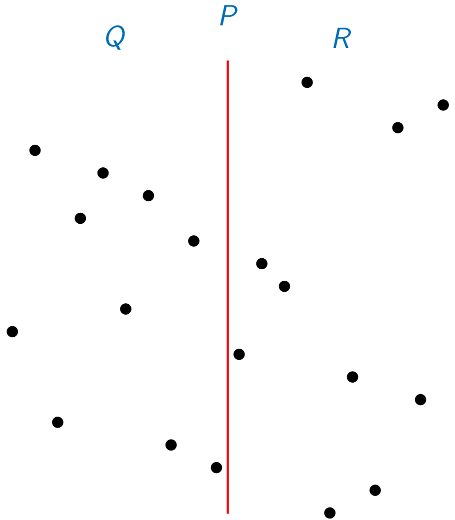
Divide and conquer

- Want to compute `ClosestPair`(P_x, P_y)
- Split (P_x, P_y) as (Q_x, Q_y), (R_x, R_y)
- Recursively compute `ClosestPair`(Q_x, Q_y) and `ClosestPair`(R_x, R_y)



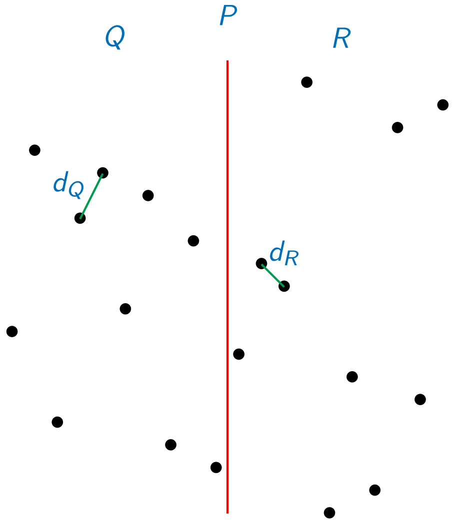
Divide and conquer

- Want to compute `ClosestPair`(P_x, P_y)
- Split (P_x, P_y) as (Q_x, Q_y), (R_x, R_y)
- Recursively compute `ClosestPair`(Q_x, Q_y) and `ClosestPair`(R_x, R_y)
- How to combine these recursive solutions?



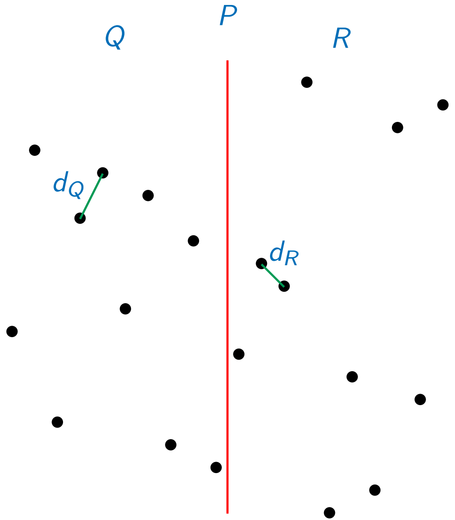
Combining solutions

- Let d_Q, d_R be closest distances in Q, R, respectively



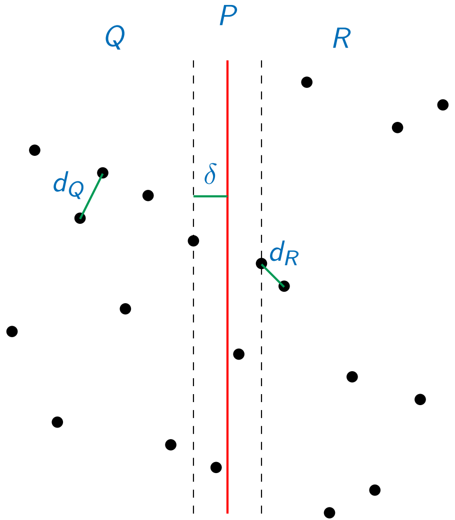
Combining solutions

- Let d_Q, d_R be closest distances in Q, R, respectively
- Set $\delta = \min(d_Q, d_R)$



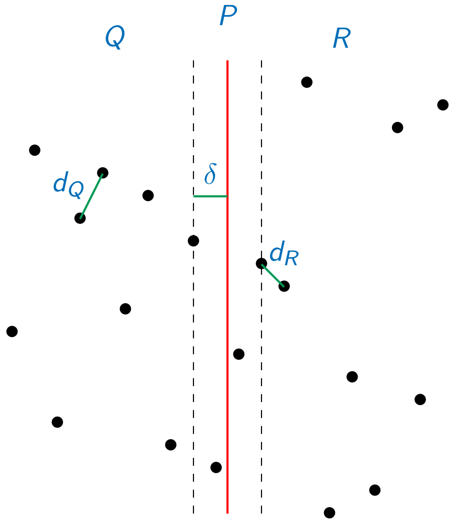
Combining solutions

- Let d_Q, d_R be closest distances in Q, R, respectively
- Set $\delta = \min(d_Q, d_R)$
- Only need to consider points within distance δ on either side of the separator



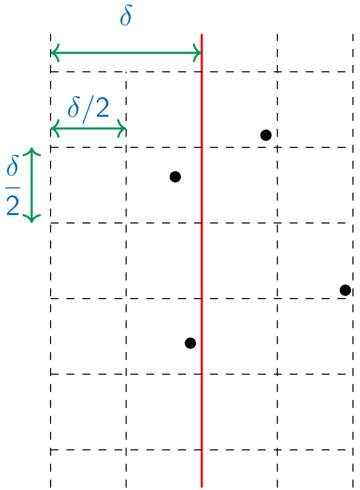
Combining solutions

- Let d_Q, d_R be closest distances in Q, R, respectively
- Set $\delta = \min(d_Q, d_R)$
- Only need to consider points within distance δ on either side of the separator
- No pair outside this band can be closer than δ



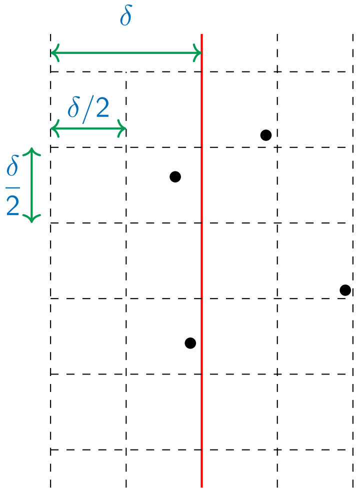
Combining solutions

- Divide the distance δ band into boxes of side $\delta/2$



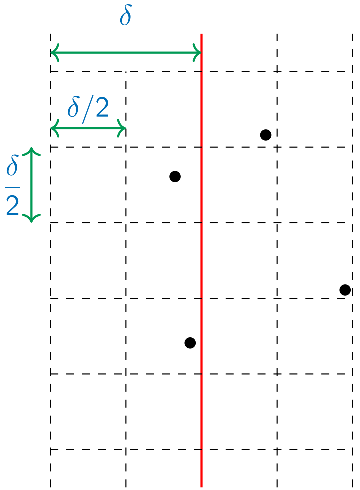
Combining solutions

- Divide the distance δ band into boxes of side $\delta/2$
- Cannot have two points inside the same box
  - Box diagonal is $\delta/\sqrt{2}$



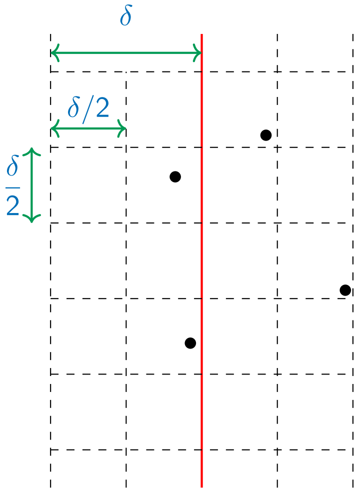
Combining solutions

- Divide the distance δ band into boxes of side $\delta/2$
- Cannot have two points inside the same box
  - Box diagonal is $\delta/\sqrt{2}$
- Any point within distance δ must lie in a 4×4 neighbourhood of boxes
  - Check each point against 15 others



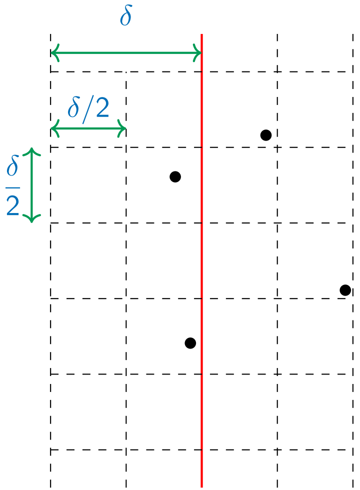
Combining solutions

- Divide the distance δ band into boxes of side $\delta/2$
- Cannot have two points inside the same box
  - Box diagonal is $\delta/\sqrt{2}$
- Any point within distance δ must lie in a 4×4 neighbourhood of boxes
  - Check each point against 15 others
- From Q_y, R_y, extract S_y, points in δ band sorted by y



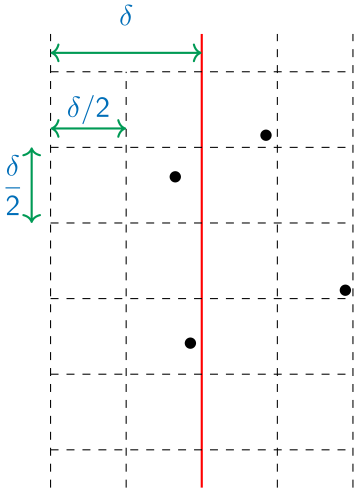
Combining solutions

- Divide the distance δ band into boxes of side $\delta/2$
- Cannot have two points inside the same box
  - Box diagonal is $\delta/\sqrt{2}$
- Any point within distance δ must lie in a 4×4 neighbourhood of boxes
  - Check each point against 15 others
- From Q_y, R_y, extract S_y, points in δ band sorted by y
- Scan S_y from bottom to top, comparing each p with next 15 points in S_y



Combining solutions

- Divide the distance δ band into boxes of side $\delta/2$
- Cannot have two points inside the same box
  - Box diagonal is $\delta/\sqrt{2}$
- Any point within distance δ must lie in a 4×4 neighbourhood of boxes
  - Check each point against 15 others
- From Q_y, R_y, extract S_y, points in δ band sorted by y
- Scan S_y from bottom to top, comparing each p with next 15 points in S_y
- Linear scan



Algorithm and analysis

Pseudocode

```
def ClosestPair(Px,Py):

  if len(Px) <= 3:
    compute pairwise distances
    return closest pair and distance

  Construct (Qx,Qy), (Rx,Ry)

  (q1,q2,dQ) = ClosestPair(Qx,Qy)
  (r1,r2,dR) = ClosestPair(Rx,Ry)

  Construct Sy from Qy,Ry
  Scan Sy, find (s1,s2,dS)

  return (q1,q2,dQ), (r1,r2,QR), (s1,s2,dS)
  depending on which of dQ, dR, dS is minimum
```

Algorithm and analysis

Pseudocode

```
def ClosestPair(Px,Py):

  if len(Px) <= 3:
    compute pairwise distances
    return closest pair and distance

  Construct (Qx,Qy), (Rx,Ry)

  (q1,q2,dQ) = ClosestPair(Qx,Qy)
  (r1,r2,dR) = ClosestPair(Rx,Ry)

  Construct Sy from Qy,Ry
  Scan Sy, find (s1,s2,dS)

  return (q1,q2,dQ), (r1,r2,QR), (s1,s2,dS)
  depending on which of dQ, dR, dS is minimum
```

Analysis

Algorithm and analysis

Pseudocode

```
def ClosestPair(Px,Py):

  if len(Px) <= 3:
    compute pairwise distances
    return closest pair and distance

  Construct (Qx,Qy), (Rx,Ry)

  (q1,q2,dQ) = ClosestPair(Qx,Qy)
  (r1,r2,dR) = ClosestPair(Rx,Ry)

  Construct Sy from Qy,Ry
  Scan Sy, find (s1,s2,dS)

  return (q1,q2,dQ), (r1,r2,QR), (s1,s2,dS)
  depending on which of dQ, dR, dS is minimum
```

Analysis

- Sort P to get P_x, P_y — $O(n \log n)$

Algorithm and analysis

Pseudocode

```
def ClosestPair(Px,Py):

  if len(Px) <= 3:
    compute pairwise distances
    return closest pair and distance

  Construct (Qx,Qy), (Rx,Ry)

  (q1,q2,dQ) = ClosestPair(Qx,Qy)
  (r1,r2,dR) = ClosestPair(Rx,Ry)

  Construct Sy from Qy,Ry
  Scan Sy, find (s1,s2,dS)

  return (q1,q2,dQ), (r1,r2,QR), (s1,s2,dS)
  depending on which of dQ, dR, dS is minimum
```

Analysis

- Sort P to get P_x, P_y — $O(n \log n)$
- Recursive algorithm
  - Construct (Q_x, Q_y), (R_x, R_y) — $O(n)$
  - Construct S_y from Q_y, R_y — $O(n)$
  - Scan S_y — $O(n)$

Algorithm and analysis

Pseudocode

```
def ClosestPair(Px,Py):

  if len(Px) <= 3:
    compute pairwise distances
    return closest pair and distance

  Construct (Qx,Qy), (Rx,Ry)

  (q1,q2,dQ) = ClosestPair(Qx,Qy)
  (r1,r2,dR) = ClosestPair(Rx,Ry)

  Construct Sy from Qy,Ry
  Scan Sy, find (s1,s2,dS)

  return (q1,q2,dQ), (r1,r2,QR), (s1,s2,dS)
  depending on which of dQ, dR, dS is minimum
```

Analysis

- Sort P to get P_x, P_y — $O(n \log n)$
- Recursive algorithm
  - Construct (Q_x, Q_y), (R_x, R_y) — $O(n)$
  - Construct S_y from Q_y, R_y — $O(n)$
  - Scan S_y — $O(n)$
- Recurrence: $T(n) = 2T(n/2) + O(n)$, like merge sort

Algorithm and analysis

Pseudocode

```
def ClosestPair(Px,Py):

  if len(Px) <= 3:
    compute pairwise distances
    return closest pair and distance

  Construct (Qx,Qy), (Rx,Ry)

  (q1,q2,dQ) = ClosestPair(Qx,Qy)
  (r1,r2,dR) = ClosestPair(Rx,Ry)

  Construct Sy from Qy,Ry
  Scan Sy, find (s1,s2,dS)

  return (q1,q2,dQ), (r1,r2,QR), (s1,s2,dS)
  depending on which of dQ, dR, dS is minimum
```

Analysis

- Sort P to get P_x, P_y — $O(n \log n)$
- Recursive algorithm
  - Construct (Q_x, Q_y), (R_x, R_y) — $O(n)$
  - Construct S_y from Q_y, R_y — $O(n)$
  - Scan S_y — $O(n)$
- Recurrence: $T(n) = 2T(n/2) + O(n)$, like merge sort
- Overall, $O(n \log n)$