CS340 - Analysis of Algorithms

Divide and Conquer III

Announcements:

Hw5 Due next Monday November 3rd

Divide and Conquer Quiz - November 6th in Lab

Project Checkpoint 2 due Thursday:

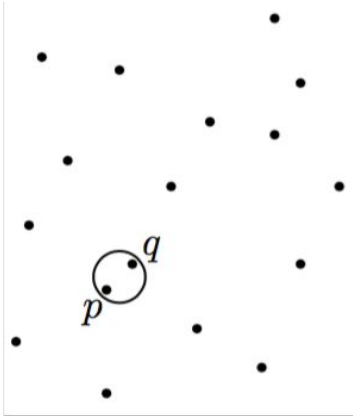
- Actually run your program
- Plot runtime with increasing sizes of problems (use script to generate different inputs increasing your dominant variable)
- If the curve is not obvious, run regression

Closest Pair

Closest Pair

Given n points in a plane, find the pair of points that is closest together

Use cases: traffic control, collusion detection, astrophysics, data analysis etc.



Closest Pair

Let's consider the problem of points in a **2D** plane

$P = \{p_1, \ldots, p_n\}$ where each point p_i has coordinates (x_i, y_i)

Distance between two points:

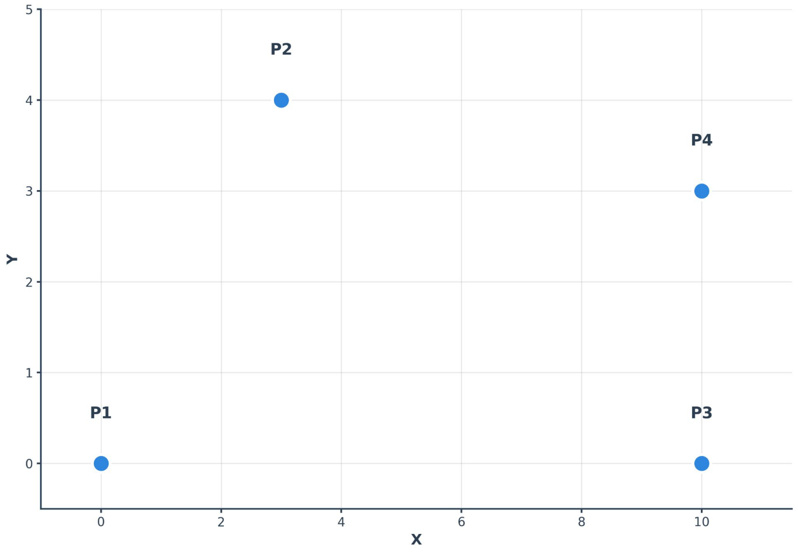
$$||pq|| = \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}$$

Closest Pair

$$\|pq\| = \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}$$

$\quad\quad\quad\quad P_1 \quad\; P_2 \quad\;\; P_3 \quad\quad P_4$

Input: P = { (0,0), (3, 4), (10, 0), (10, 3) }



Brute Force: Calculate distance between all pairs and return the pair with smallest dist

$\|P_1P_2\|$ = ...

$\|P_1P_3\|$ = ...

....

Runtime?

$O(n^2)$

Closest Pair in 1D

Let's consider a simpler version: 1 dimension

P_1 P_2 P_3 P_4

Input: P = { 7, 15, 2, 18 }

$$\|pq\| = \sqrt{(p_x - q_x)^2}$$

Algorithm:

- Sort inputs
- Walk through the list and compute distance from each P_i to its neighbor P_{i+1}
- One of the computed distances must be the smallest between pairs

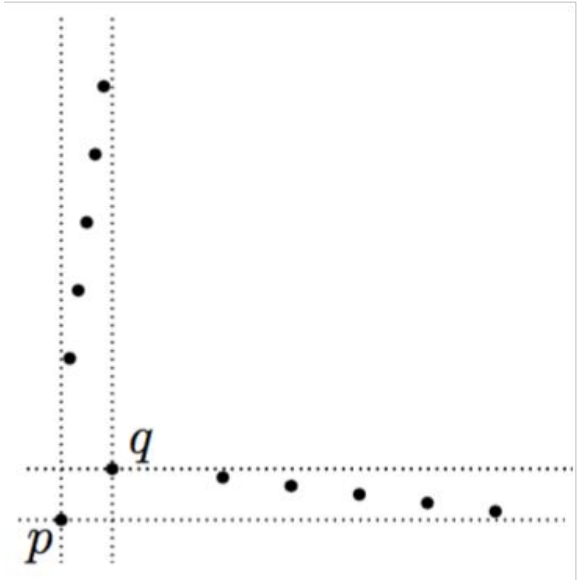
Key: two closest points are adjacent in sorted order

Closest Pair

Does sorting help us in 2D?

Sort by x-coordinate (or y-coordinate)

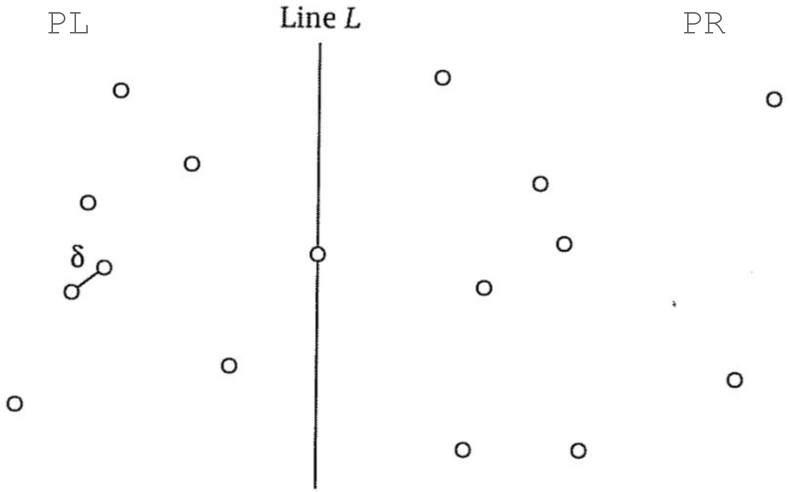
Focusing on one coordinate doesn’t help



Closest Pair - Divide and Conquer

Divide and Conquer:

- Find closest pair in left half of P
- Find closest pair in right half of P



Divide step:

Split P evenly into PL and PR via a line L

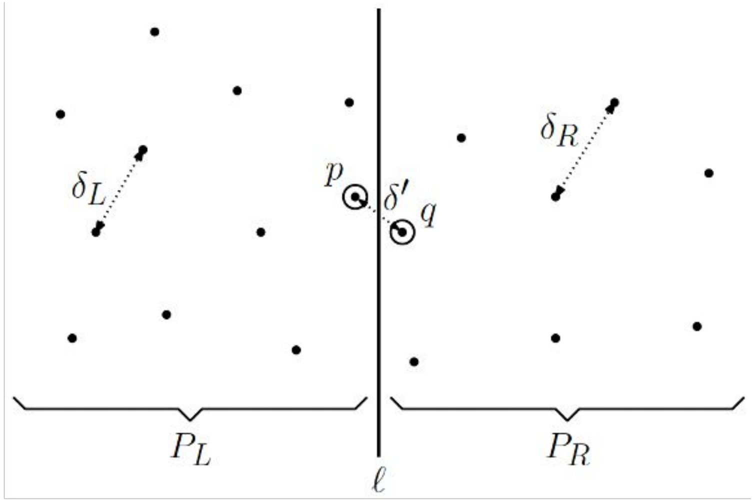
Base case?

P = 3

Closest Pair - Divide and Conquer

- Find closest pair in left half of P
- Find closest pair in right half of P
- Combine: use these to find overall solution

$$\delta = \min(\delta_L, \delta_R)$$? No!

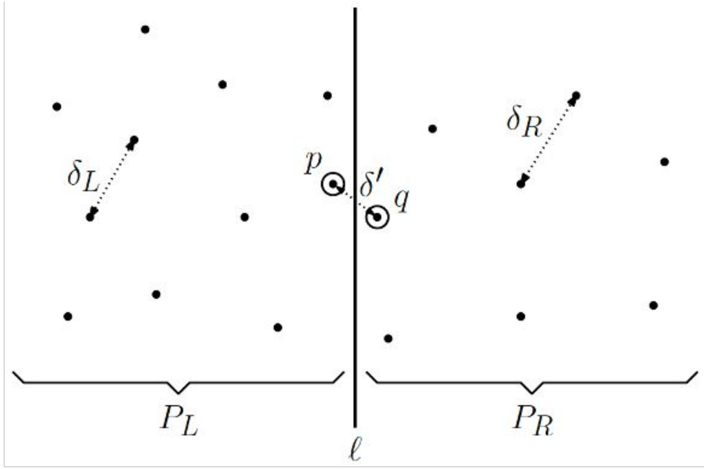


Combine is the tricky part. Pairs that occur across the split

(Remember our inversion counting?)

Closest Pair - Combine

- $\delta = \min(\delta_L, \delta_R)$
- δ is not necessarily right
- We need the closest pair between the sets
- Think of δ as an upper bound
- Restriction:
  - Find (p, q) where $p \in P_L$ and $q \in P_R$ such that $\|pq\| < \delta$, if one exists



Brute Force: check distance of all pairs across the split

Runtime complexity?

$O(n^2)$

Quadratic Combine...

An $O(n^2)$ combine step would give us an overall runtime complexity of:

$T(n) = 2T(n/2) + O(n^2)$

$a = 2$, $b = 2$, $d = 2$

If $T(n) = aT(n/b) + O(n^d)$ for constants $a > 0$, $b > 1$, $d \geq 0$, then

$$T(n) = \begin{cases} O(n^d) & \text{if } d > \log_b a \\ O(n^d \log n) & \text{if } d = \log_b a \\ O(n^{\log_b a}) & \text{if } d < \log_b a \end{cases}$$

$O(n^2)$

No better than brute force!

Linear Combine

An O(n) combine step would give us an overall runtime complexity of:

If $T(n) = aT(n/b) + O(n^d)$ for constants $a > 0$, $b > 1$, $d \geq 0$, then

$$T(n) = \begin{cases} O(n^d) & \text{if } d > \log_b a \\ O(n^d \log n) & \text{if } d = \log_b a \\ O(n^{\log_b a}) & \text{if } d < \log_b a \end{cases}$$

T(n) = 2T(n/2) + O(n)

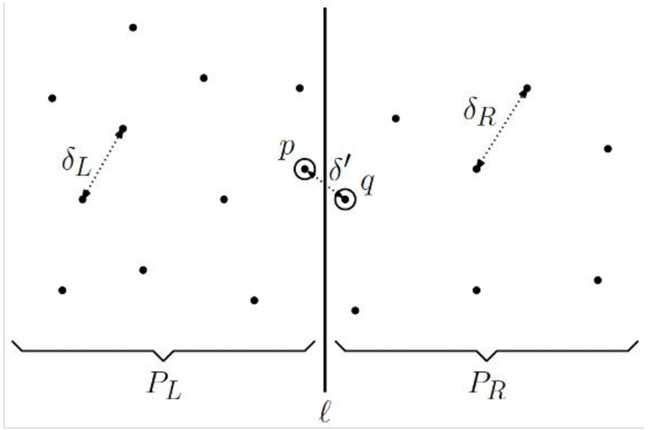
a = 2, b = 2, d = 1

O(nlogn)

How to combine

$$\delta = \min(\delta_L, \delta_R)$$

- δ is not necessarily right
- We need the closest pair between the sets
- Think of δ as an upper bound
- Restriction:
  - Find (p, q) where $p \in P_L$ and $q \in P_R$ such that $\|pq\| < \delta$, if one exists
  - We can afford $O(n)$



We can't check every pair of points across the split. It's too expensive.
Only check points within δ of L

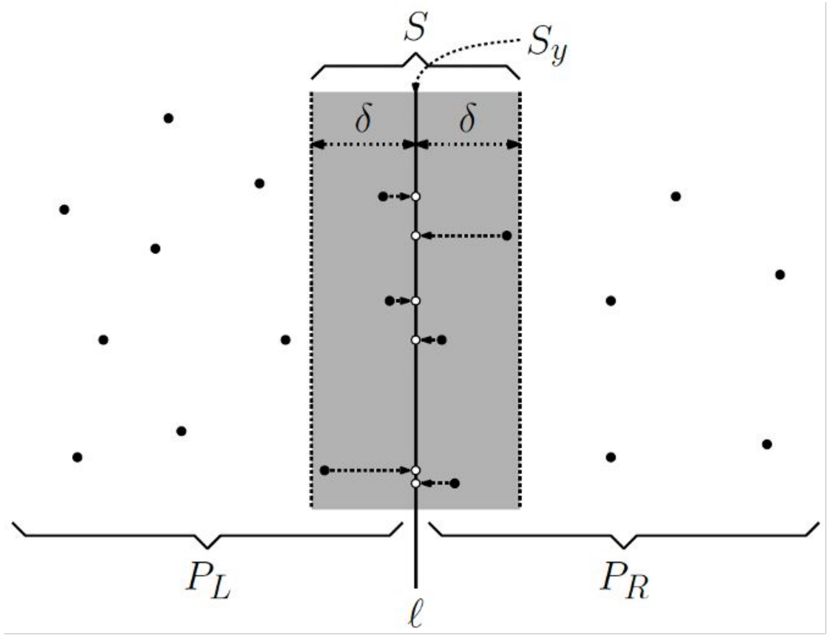
Combine - Dividing Line

$\delta = \min(\delta_L, \delta_R)$

If a pair (`p,q`) exists such that $||pq|| < \delta$,

p and q must be

- across the dividing line
- within distance δ of the line



Let S denote the subset of such points in P (within δ of the line)

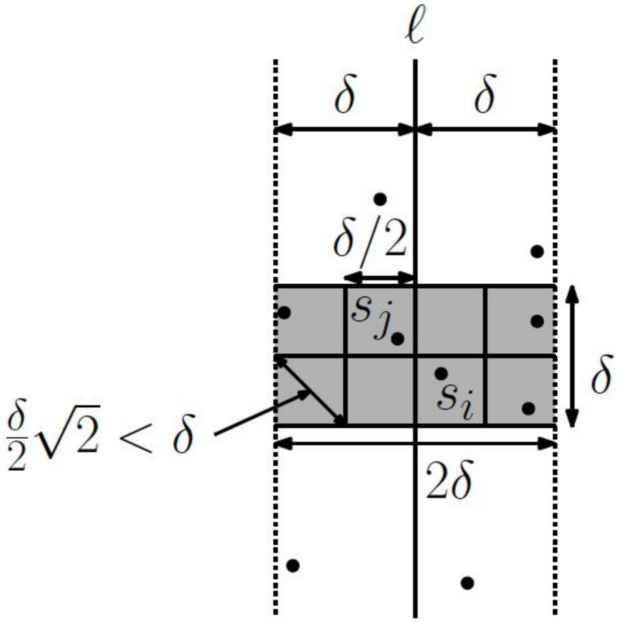
Given a point S_i, how many points do we need to check it against?

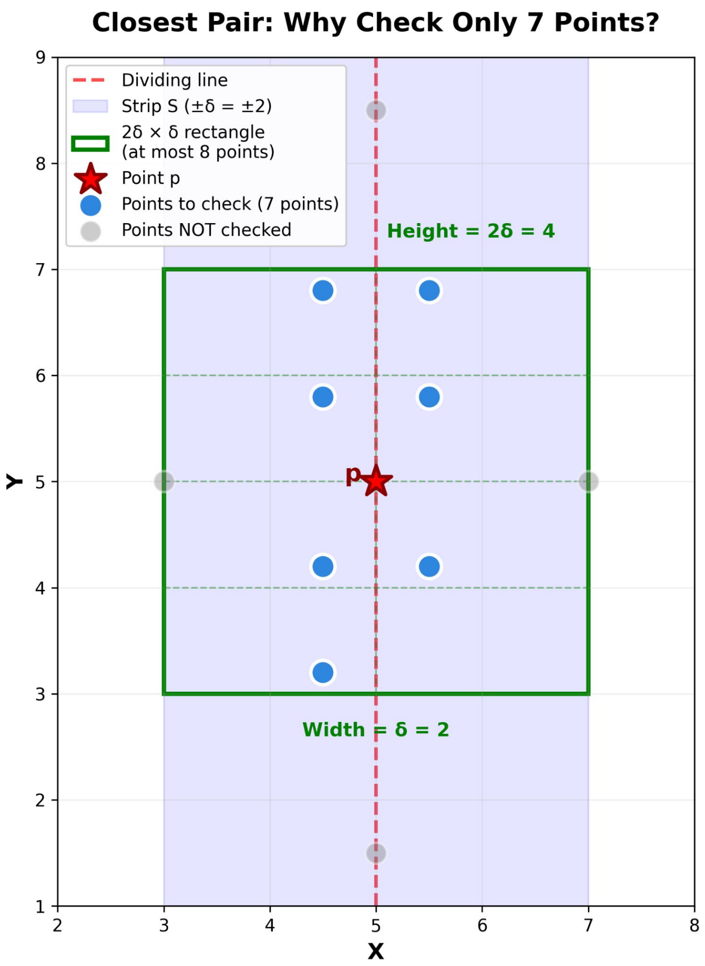
Combine - Dividing Line

Let S_y be all points within distance δ of the line, sorted by y coordinate

Lemma 1:
Given any two points: s_i, $s_j \in S_y$,

if $||s_i s_j|| < \delta$, then $|j - i| <= 7$




Closest Pair: Why Check Only 7 Points?
Dividing line
Strip S (±δ = ±2)
2δ × δ rectangle
(at most 8 points)
Point p
Points to check (7 points)
Points NOT checked
Height = 2δ = 4
Width = δ = 2
p
X
Y
9
8
7
6
5
4
3
2
1
2
3
4
5
6
7
8

Lemma 2

By lemma #1, a pair (p,q) where $p \in P_L$ and $q \in P_R$ such that $||pq|| < \delta$, if one exists, can be found by **linear search** in S_y in 8-point blocks.

Pseudo Code

```
closestPair(P=(Px, Py)) {
  if (|P|<=3) solve by brute force
  else {
    Split P into PL and PR (via L)
    dL = closestPair(PL) dR = closestPair(PR)
    d = min(dL, dR)
    for (i = 1 to n) {
      if (Py[i] is within distance d of L) {
        append Py[i] to Sy
      }
    }
    d’= findClosest(Sy)
  }
  return min(d, d’);
}
```

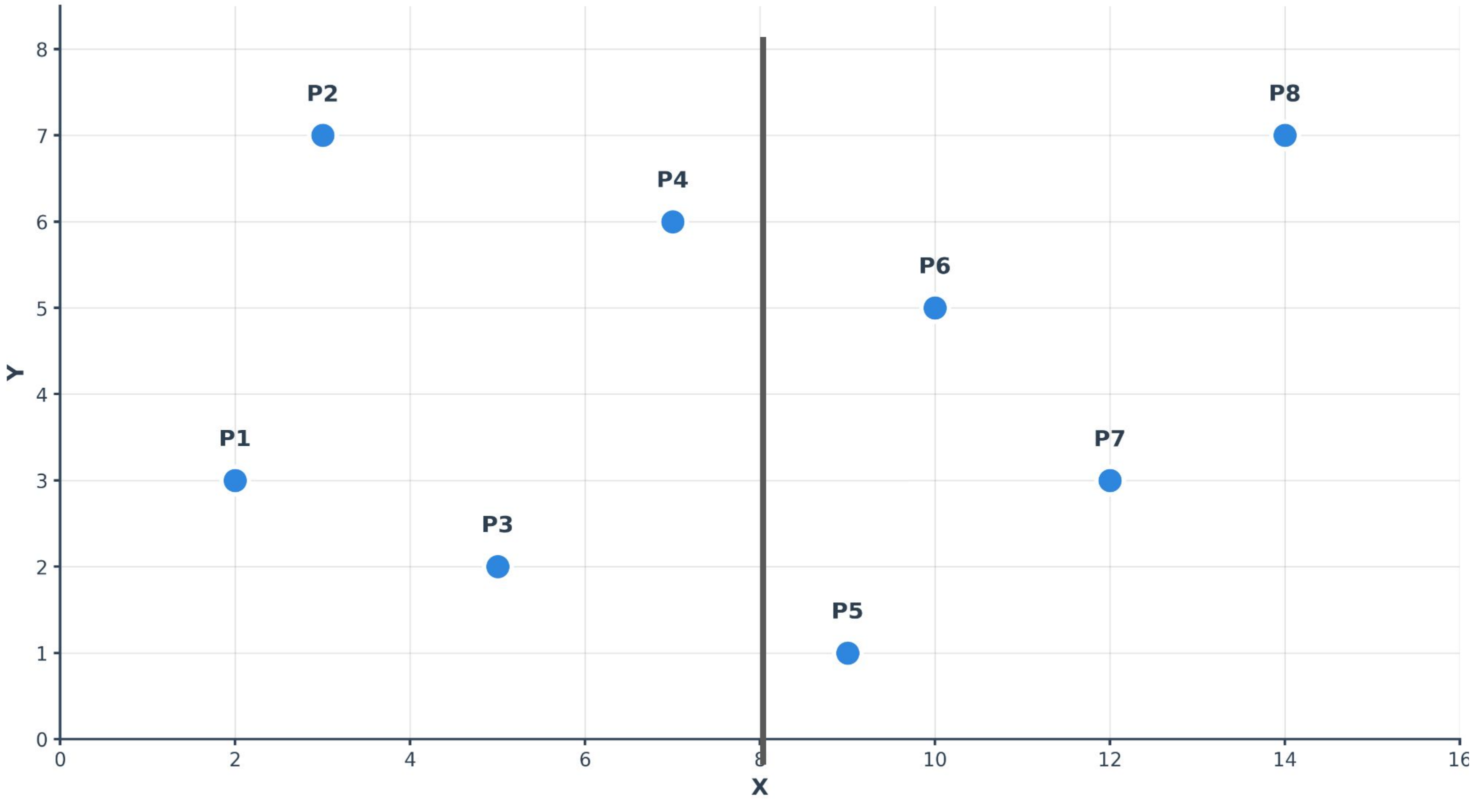
```
findClosest(Sy) {
  m = |Sy|
  d’ = infinity
  for (i = 1 to m) {
    for (j = i+1 to min(m, i+7)) {
      if (dist(Sy[i], Sy[j]) < d’) {
        d’ = dist(Sy[i], Sy[j])
      }
    }
  }
  return d’
}
```

P = {(2,3), (3,7), (5,2), (7,6), (9,1), (10,5), (12,3), (14,7)}

```
closestPair(P=(Px, Py)) {
  if (|P|<=3) solve by brute force
  else {
    Split P into PL and PR (via L)
    dL = closestPair(PL) dR = closestPair(PR)
    d = min(dL, dR)
    for (i = 1 to n) {
      if (Py[i] is within distance d of L) {
        append Py[i] to Sy
      }
    }
    d’= findClosest(Sy)
  }
  return min(d, d’);
}
```

```
findClosest(Sy) {
  m = |Sy|
  d’ = infinity
  for (i = 1 to m) {
    for (j = i+1 to min(m, i+7)) {
      if (dist(Sy[i], Sy[j]) < d’) {
        d’ = dist(Sy[i], Sy[j])
      }
    }
  }
  return d’
}
```

$P = \{(2,3), (3,7), (5,2), (7,6), (9,1), (10,5), (12,3), (14,7)\}$

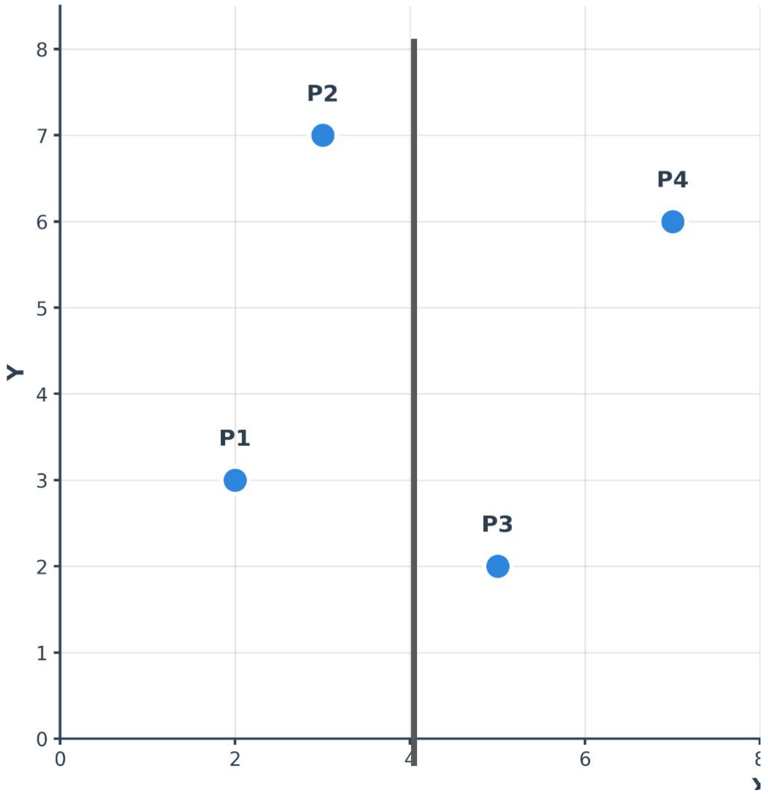


P = {(2,3), (3,7), (5,2), (7,6)}

```
closestPair(P=(Px, Py)) {
  if (|P|<=3) solve by brute force
  else {
    Split P into PL and PR (via L)
    dL = closestPair(PL) dR = closestPair(PR)
    d = min(dL, dR)
    for (i = 1 to n) {
      if (Py[i] is within distance d of L) {
        append Py[i] to Sy
      }
    }
    d’= findClosest(Sy)
  }
  return min(d, d’);
}
```

```
findClosest(Sy) {
  m = |Sy|
  d’ = infinity
  for (i = 1 to m) {
    for (j = i+1 to min(m, i+7)) {
      if (dist(Sy[i], Sy[j]) < d’) {
        d’ = dist(Sy[i], Sy[j])
      }
    }
  }
  return d’
}
```

$P = \{(2,3), (3,7), (5,2), (7,6)\}$

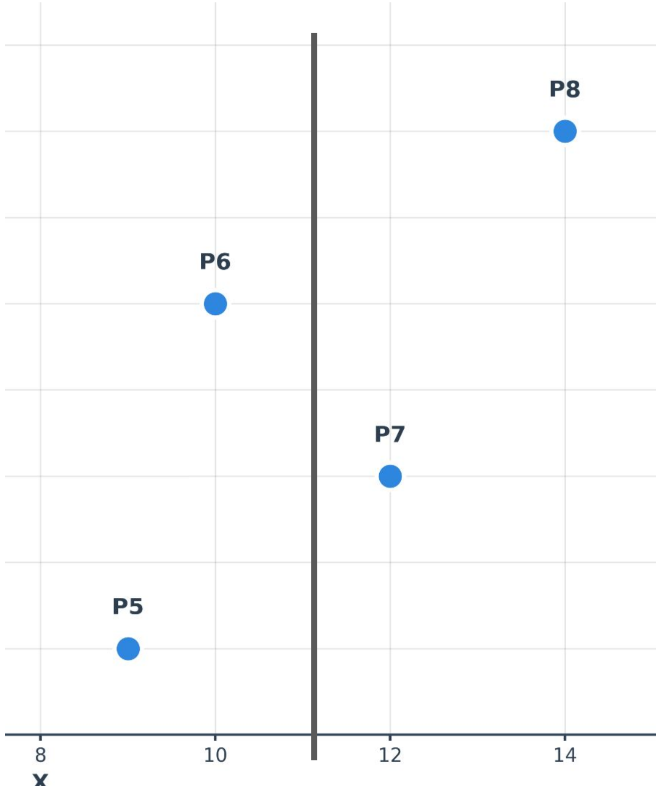


P = {(9,1), (10,5), (12,3), (14,7)}

```
closestPair(P=(Px, Py)) {
  if (|P|<=3) solve by brute force
  else {
    Split P into PL and PR (via L)
    dL = closestPair(PL) dR = closestPair(PR)
    d = min(dL, dR)
    for (i = 1 to n) {
      if (Py[i] is within distance d of L) {
        append Py[i] to Sy
      }
    }
    d’= findClosest(Sy)
  }
  return min(d, d’);
}
```

```
findClosest(Sy) {
  m = |Sy|
  d’ = infinity
  for (i = 1 to m) {
    for (j = i+1 to min(m, i+7)) {
      if (dist(Sy[i], Sy[j]) < d’) {
        d’ = dist(Sy[i], Sy[j])
      }
    }
  }
  return d’
}
```

$P = \{(9,1), (10,5), (12,3), (14,7)\}$

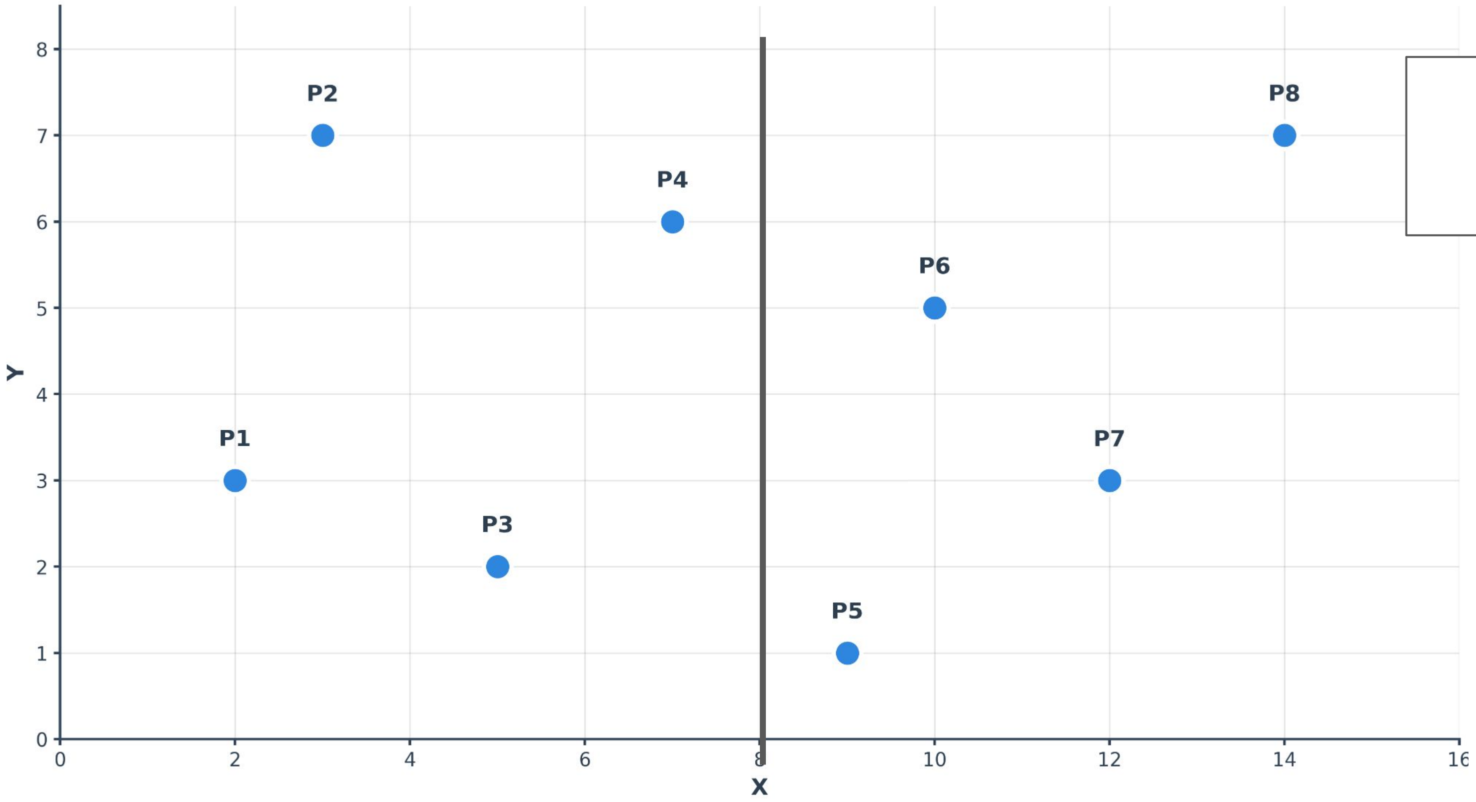


P = {(9,1), (10,5), (12,3), (14,7)}

```
closestPair(P=(Px, Py)) {
  if (|P|<=3) solve by brute force
  else {
    Split P into PL and PR (via L)
    dL = closestPair(PL) dR = closestPair(PR)
    d = min(dL, dR)
    for (i = 1 to n) {
      if (Py[i] is within distance d of L) {
        append Py[i] to Sy
      }
    }
    d’= findClosest(Sy)
  }
  return min(d, d’);
}
```

```
findClosest(Sy) {
  m = |Sy|
  d’ = infinity
  for (i = 1 to m) {
    for (j = i+1 to min(m, i+7)) {
      if (dist(Sy[i], Sy[j]) < d’) {
        d’ = dist(Sy[i], Sy[j])
      }
    }
  }
  return d’
}
```

$P = \{(2,3), (3,7), (5,2), (7,6), (9,1), (10,5), (12,3), (14,7)\}$



$\delta = \sqrt{8}$

Runtime analysis?

```
closestPair(P=(Px, Py)) {
  if (|P|<=3) solve by brute force
  else {
    Split P into PL and PR (via L)
    dL = closestPair(PL) dR = closestPair(PR)
    d = min(dL, dR)
    for (i = 1 to n) {
      if (Py[i] is within distance d of L) {
        append Py[i] to Sy
      }
    }
    d’= findClosest(Sy)
  }
  return min(d, d’);
}
```

```
findClosest(Sy) {
  m = |Sy|
  d’ = infinity
  for (i = 1 to m) {
    for (j = i+1 to min(m, i+7)) {
      if (dist(Sy[i], Sy[j]) < d’) {
        d’ = dist(Sy[i], Sy[j])
      }
    }
  }
  return d’
}
```

Master Theorem

If $T(n) = aT(n/b) + O(n^d)$ for constants $a > 0$, $b > 1$, $d \geq 0$, then

$$T(n) = \begin{cases} O(n^d) & \text{if } d > \log_b a \\ O(n^d \log n) & \text{if } d = \log_b a \\ O(n^{\log_b a}) & \text{if } d < \log_b a \end{cases}$$

T(n) = 2T(n/2) + O(n)

O(nlogn)

Proof of Correctness

By induction: The algorithm correctly outputs a closest pair of points in P

Base Case: brute force works

Induction Hypothesis: Assume that `closestPair(n/2)` returns the correct δ

We will use this to show that `closestPair(n)` returns the correct δ

The closest pair either has

(a) Both points in P_L
(b) Both points in P_R
(c) Both points in S_y

Proof of Correctness

<u>Induction Hypothesis:</u> Assume that `closestPair(n/2)` returns the correct δ

The closest pair either has

(a) Both points in P_L - correct by IH
(b) Both points in P_R - correct by IH
(c) Both points in S_y - prove by correctness of `findClosest(`S_y`)`

By Lemma 1 and Lemma 2, we only need to check 7 points away, so the for loop for `findClosest` is correct.

Finding the smallest of δ_L, δ_R, and δ' is the closest pair distance.

Summary

- HW5 due next Monday

- Quiz on divide and conquer Nov 6

- Project Checkpoint 2 due thursday night