Dividir y Conquistar

Dividir y Conquistar

- Conocida como
  - “Divide & Conquer”
  - “Dividir y Conquistar
  - “Divide y Reinarás
  - D&C
  - etc.
- La metodología consiste en
  - dividir un problema en problemas similares....pero más chicos
  - resolver los problemas menores
  - Combinar las soluciones de los problemas menores para obtener la solución del problema original.
- El método tiene sentido siempre y cuando la división y la combinación no sean excesivamente caras
- ¿Entonces?

Esquema general de D&C

- Algoritmo DC(X)
  - Si X es suficientemente chico (o simple)
    - ADHOC(X)
  - En caso contrario,
    - Descomponer X en subinstancias X_1, X_2,.... X_k
    - Para i desde 1 hasta k hacer
      - $Y_i \leftarrow DC(X_i)$
    - Combinar las soluciones Y_i para construir una solución Y para X

Ejemplo: Algoritmo de Karatsuba

- Queremos multiplicar dos enteros de *n* cifras en base *b*, *X* e *Y*
- Algoritmo tradicional requiere....$O(n^2)$
- Pero...
  - Sea $X = X_1\, b^{(n/2)} + X_0$, e $Y = Y_1\, b^{(n/2)} + Y_0$
- Entonces,
  - $XY = X_1 Y_1\, b^n + (X_0 Y_1 + X_1 Y_0)\, b^{n/2} + X_0 Y_0$
- Si calculamos la complejidad de este método, nos da $O(n^2)$ (ver después)

Ejemplo: Algoritmo de Karatsuba(II)

- Pero si definimos
  - $m_1 = X_0 Y_0$, $m_2 = X_1 Y_1$ y $m_3 = (X_0 - X_1)(Y_1 - Y_0)$
- Resulta que
  - $XY = m_2 b^n + (m_1 + m_2 + m_3) b^{n/2} + m_1$
- O sea tenemos 3 subproblemas de tamaño n/2, lo que mejora la complejidad total del algoritmo llevándola a $T(n)=O(n^{\log_2 3}) \sim O(n^{1.59})$ (ver después).
- Año: 1962

Algoritmo de Strassen

- Strassen aplicó esta misma idea a la multiplicación de matrices. Dividiendo cada matriz en 4, logró pasar del algoritmo clásico de $O(n^3)$ a $O(n^{\log_2 7 + o(1)}) \sim O(n^{2.8074})$
- Es de 1969
- (¡pensarla!)
- Cota superior actual $O(n^{2.3728639})$ (¡2014!)

Recurrencias típicas de D&C

- Esquema D&C: dividir las instancias de tamaño mayor que cierto n_0 en sub-instancias, resolverlas y luego combinar las soluciones para el problema inicial.
- Situación típica: dividir en a subproblemas, de tamaño máximo n/c, el costo de determinar subproblemas (divide) y unirlos (conquer) es bn^d. Suponemos la base ($n=1$) cuesta b.

Solución de la recurrencia típica

- Costo (suponiendo $n=c^k$)
  - $T(n)=aT(n/c)+bn^d = aT(c^{k-1})+bn^d=$
  - $= a(aT(c^{k-2})+(bc^{d(k-1)}))+ bc^{kd}= a^2T(c^{k-2})+abc^{d(k-1)}+ bc^{kd}=$
  - $= a^2(a(T(c^{k-3})+b(c^{k-2})^d)+abc^{d(k-1)}+ bc^{kd} =$
  - $= a^3T(c^{k-3})+a^2b(c^{k-2})^d+abc^{d(k-1)}+ bc^{kd} =$
  - $= a^3T(c^{k-3})+a^2bc^{d(k-2)}+abc^{d(k-1)}+ bc^{kd} =$
  - =.................................=
  - $a^jT(c^{k-j})+ \sum_{i=0}^{j-1} a^ibc^{d(k-i)}=$
  - =..............................= ¿hasta cuándo? Hasta $c^{(k-j)}=1$ o sea $j=\log_c n$
  - $= a^kb+ b\sum_{i=0}^{k-1} a^ic^{d(k-i)}= b\sum_{i=0}^{k} a^ic^{d(k-i)}=$
  - $bc^{dk}\sum_{i=0}^{k} a^ic^{-di}= bn^d\sum_{i=0}^{\log n} (a/c^d)^i = ?$

Análisis por casos $bn^d \sum_{i=0}^{\log_c n} \left(\frac{a}{c^d}\right)^i$

- a=1, d=0 (o sea, un solo subproblema, la combinación tiene costo constante)

$$b \sum^{\log_c n} 1 = O(\log_c n)$$

- d=1 (o sea "conquer lineal")
  - Caso a<c (o sea, "pocos subproblemas")
  - Como a/c<1, la serie converge →

$$bn \sum_{i=0}^{\log_c n} (a/c)^i = bn.const = O(n)$$

Análisis por casos $bn^d \sum_{i=0}^{\log_c n} \left(\frac{a}{c^d}\right)^i$

- d=1 (o sea "conquer lineal")
  - Caso a=c

$$bn\sum^{\log_c n} 1 = O(n \log_c n)$$

  - Caso a>c (o sea, muchos subproblemas)

$$T(n) = bn \sum_{i=0}^{\log_c n} (a/c)^i = bn \frac{\left(\frac{a}{c}\right)^{\log_c n + 1} - 1}{\frac{a}{c} - 1} = O\left(n \left(\frac{a}{c}\right)^{\log_c n}\right) =$$

$$O\left(n \frac{a^{\log_c n}}{c^{\log_c n}}\right) = O\left(n \frac{a^{\log_c n}}{n}\right) = O\left(a^{\log_a n \times \log_c a}\right) = O\left(n^{\log_c a}\right)$$

Teorema Maestro (Master theorem)

- Permite resolver relaciones de recurrencia de la forma:

$$T(n) = \begin{cases} aT(n/c) + f(n) & si \quad n > 1 \\ 1 & si \quad n = 1 \end{cases}$$

- La solución es

$$T(n) = \Theta(n^{\log_c a}) \text{ si } \quad f(n) = O(n^{\log_c a - \varepsilon}) \text{ para } \varepsilon > 0$$

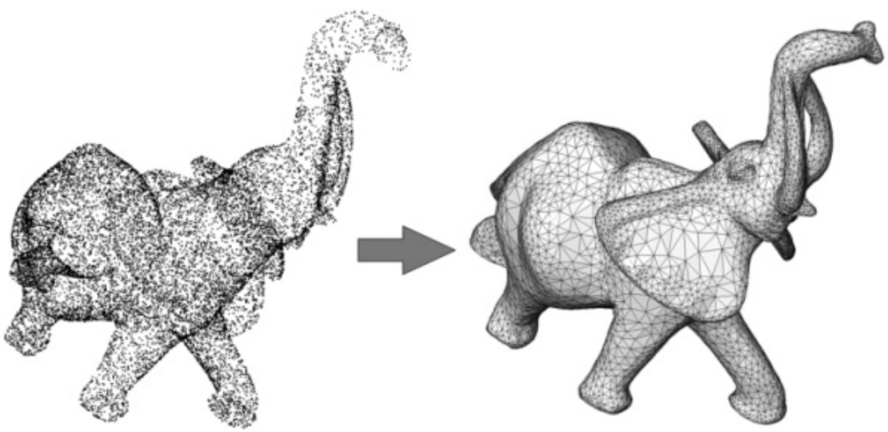
$$T(n) = \Theta(n^{\log_c a} \log n) \text{ si } \quad f(n) = \Theta(n^{\log_c a})$$

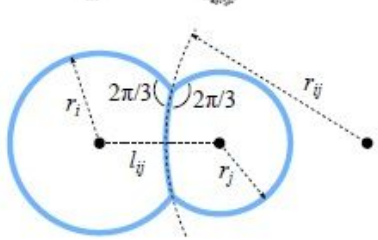
$$T(n) = \Theta(f(n)) \text{ si } \quad \begin{array}{l} f(n) = \Omega(n^{\log_c a + \varepsilon}) \text{ para } \varepsilon > 0 \text{ y} \\ af(n/c) < kf(n) \text{ para } k < 1 \text{ y } n \text{ suficientemente grande} \end{array}$$

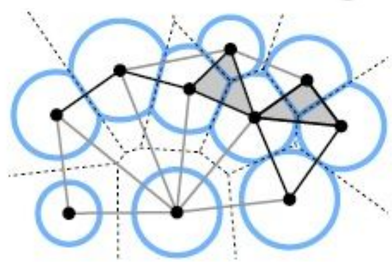
- En los casos 1 y 3 es $T(n) \in \Theta(f(n) + n^{log_c a})$
- Hay otras formulaciones con pequeñas variantes

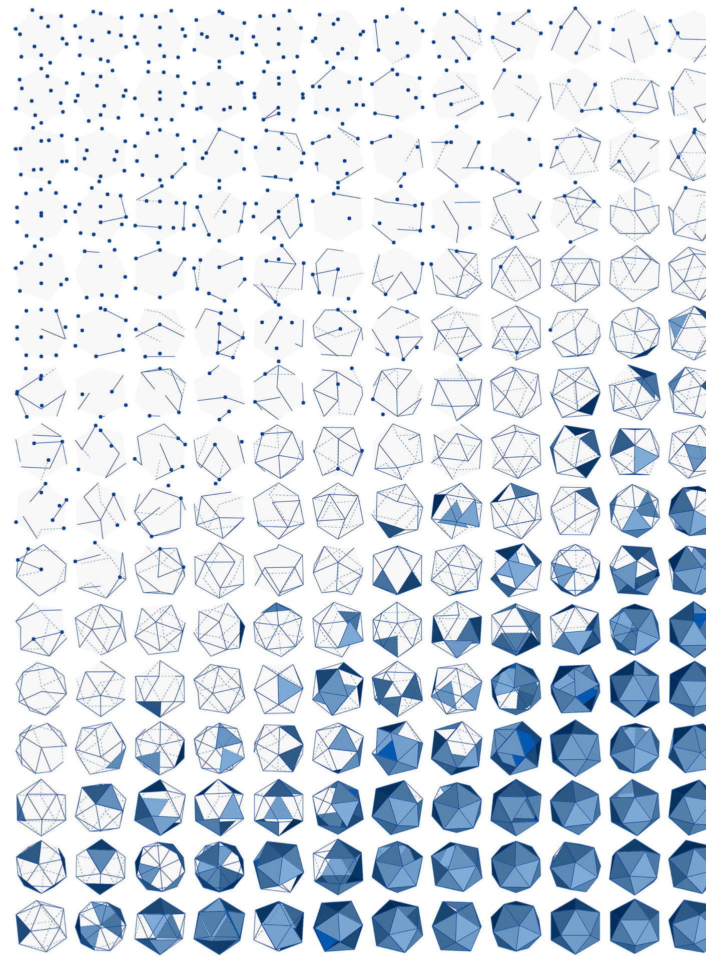
Ejemplo: par de puntos más cercano

- Algoritmo de Geometría Computacional



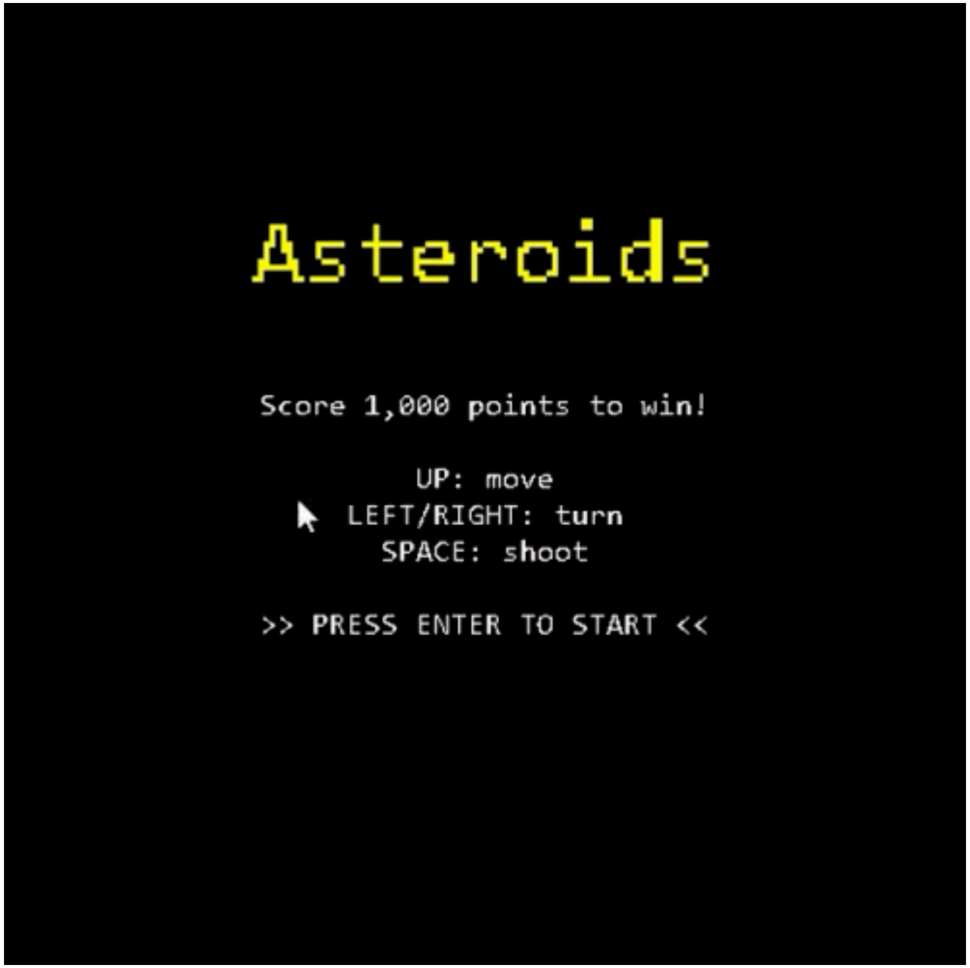






Motivación


ASTEROIDS

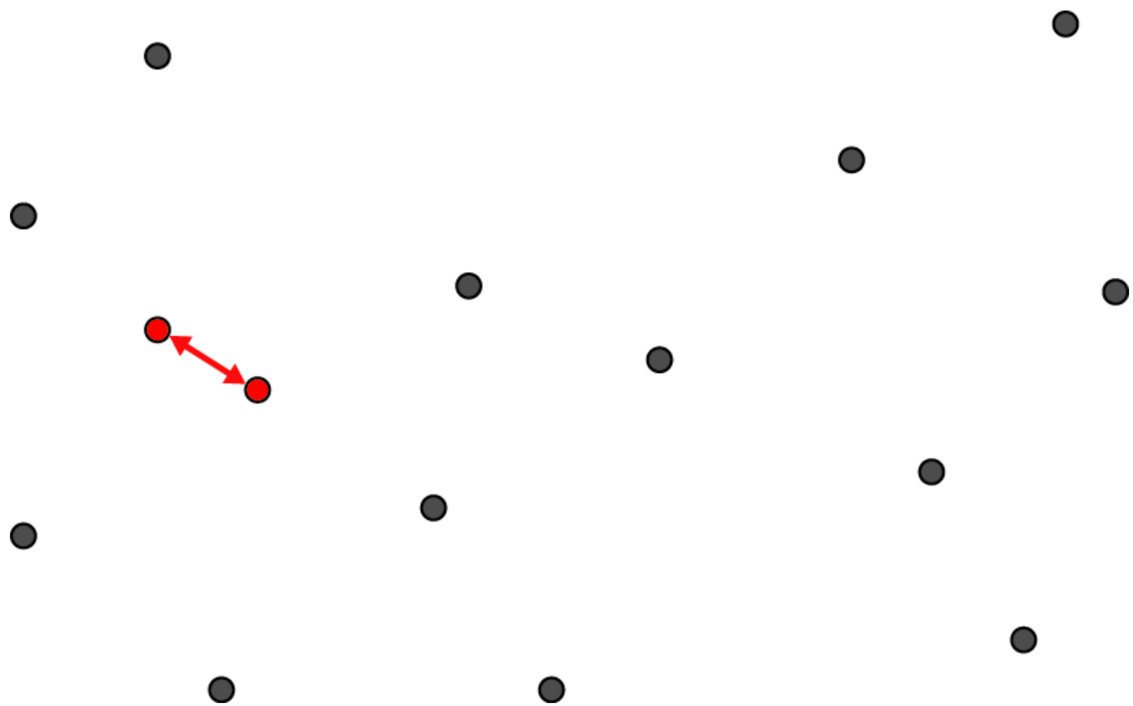

Asteroids
Score 1,000 points to win!
UP: move
LEFT/RIGHT: turn
SPACE: shoot
>> PRESS ENTER TO START <<

Ejemplo: par de puntos más cercano

- **Objetivo**: dado un conjunto de puntos en el plano, encontrar el par de puntos más cercano.

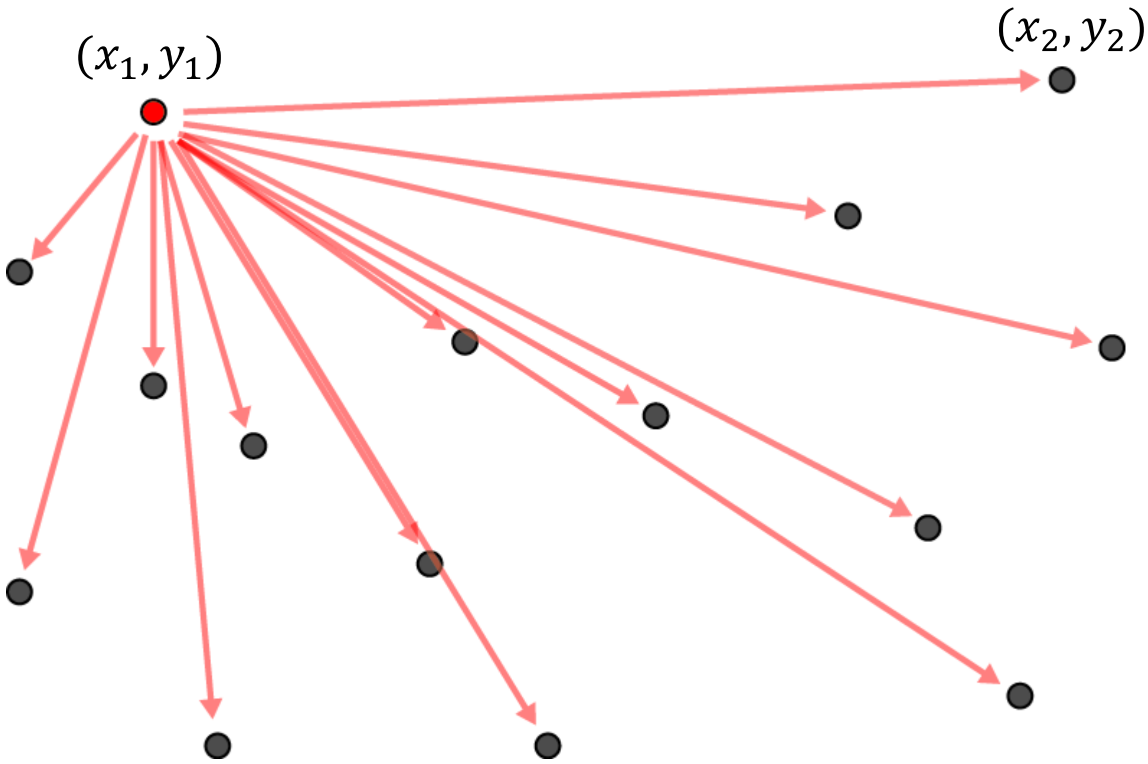
Ejemplo: par de puntos más cercano

- **Objetivo**: dado un conjunto de puntos en el plano, encontrar el par de puntos más cercano.



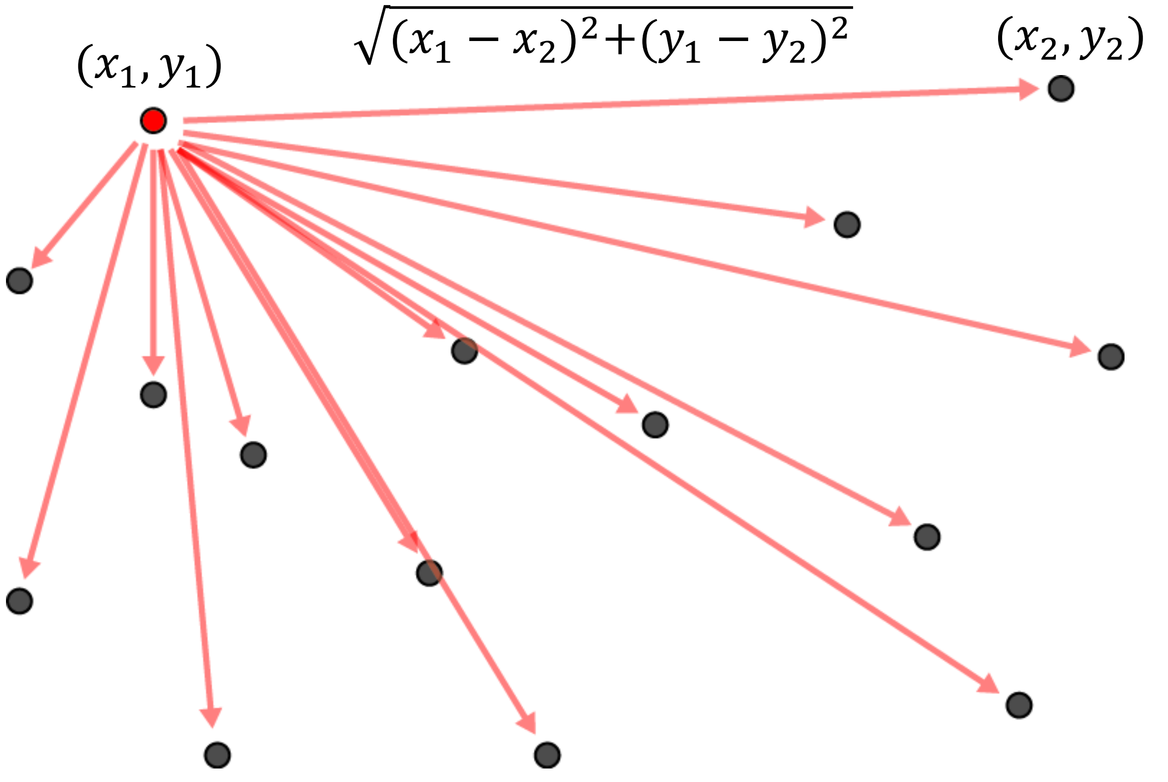
Ejemplo: par de puntos más cercano

- **Solución fuerza bruta**: comparar todos con todos



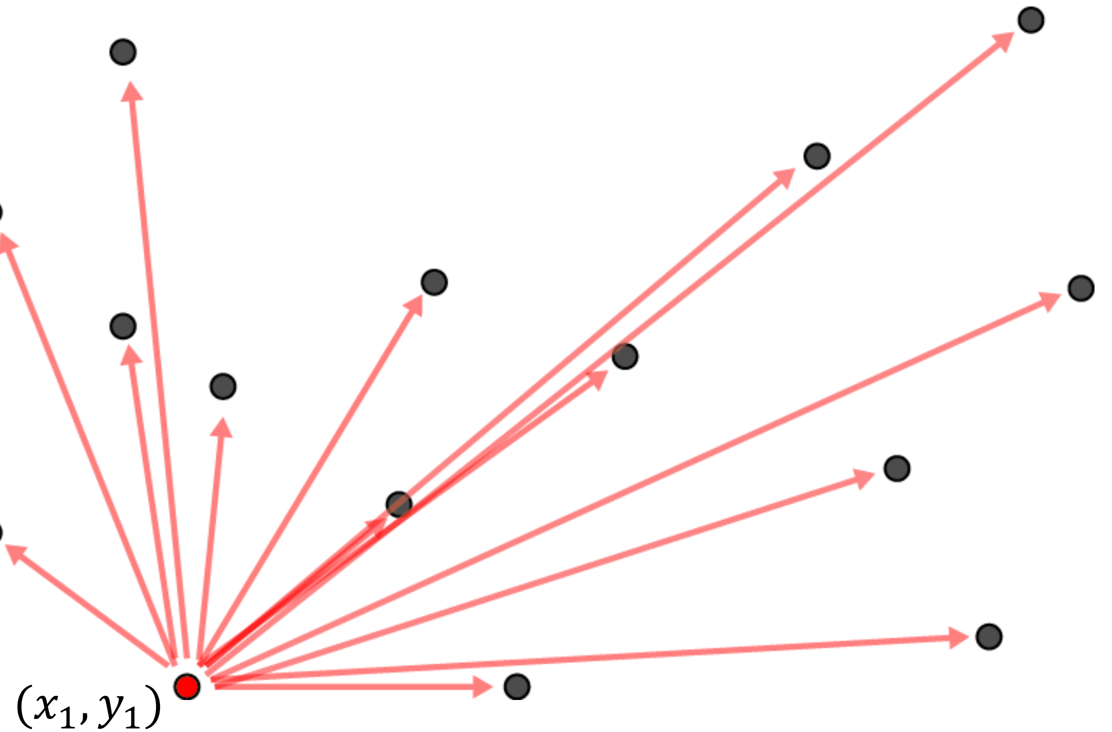
Ejemplo: par de puntos más cercano

- **Solución fuerza bruta**: comparar todos con todos



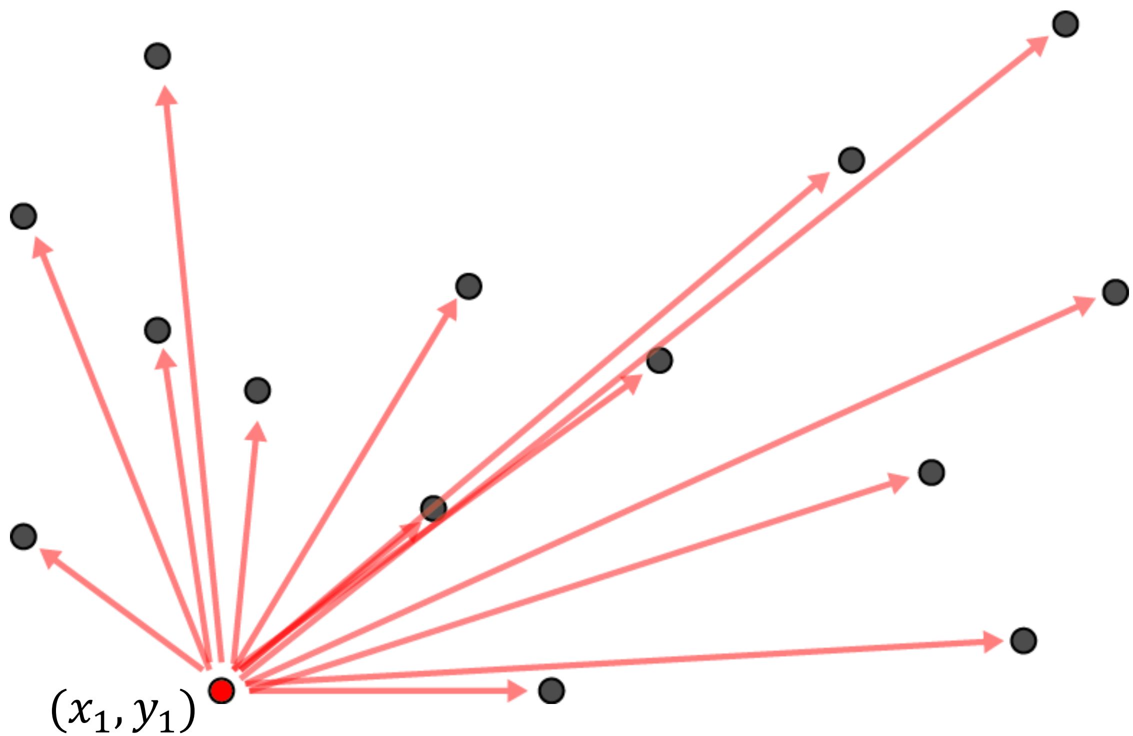
Ejemplo: par de puntos más cercano

- **Solución fuerza bruta**: comparar todos con todos



Ejemplo: par de puntos más cercano

- **Solución fuerza bruta**: comparar todos con todos $\binom{n}{2} = \theta(n^2)$ operaciones

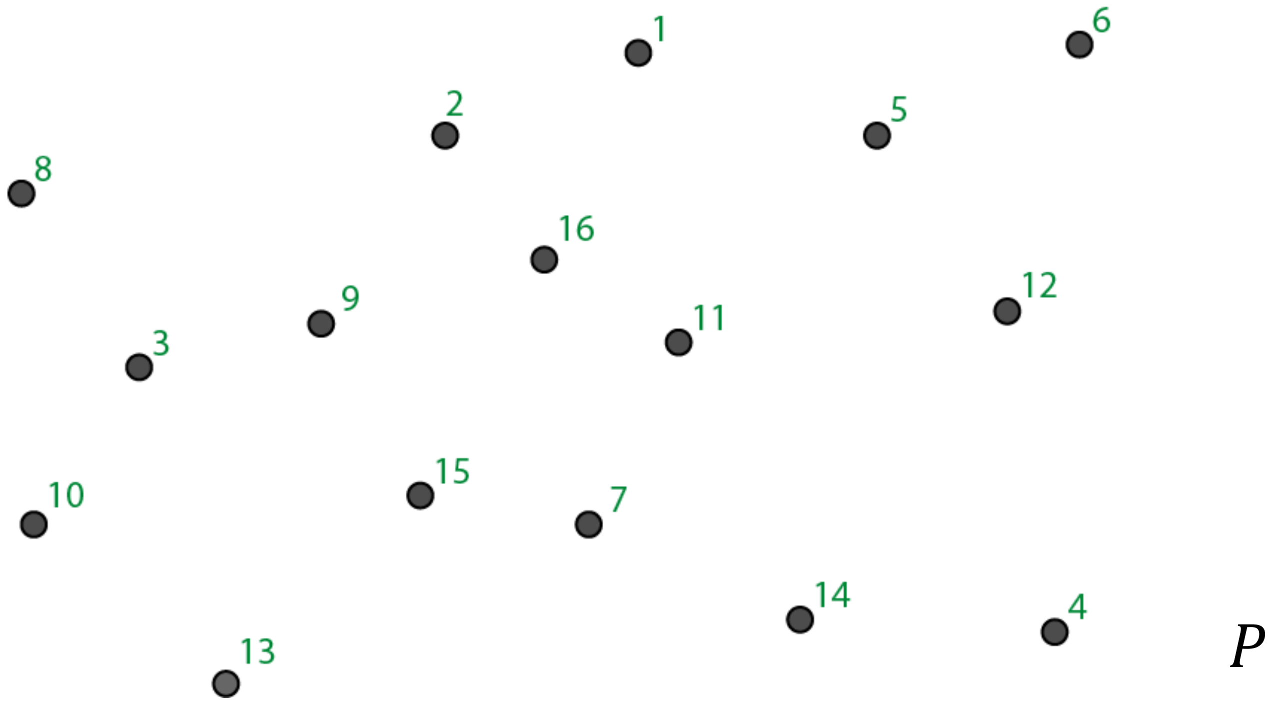


Ejemplo: par de puntos más cercano

- **Solución D&C**

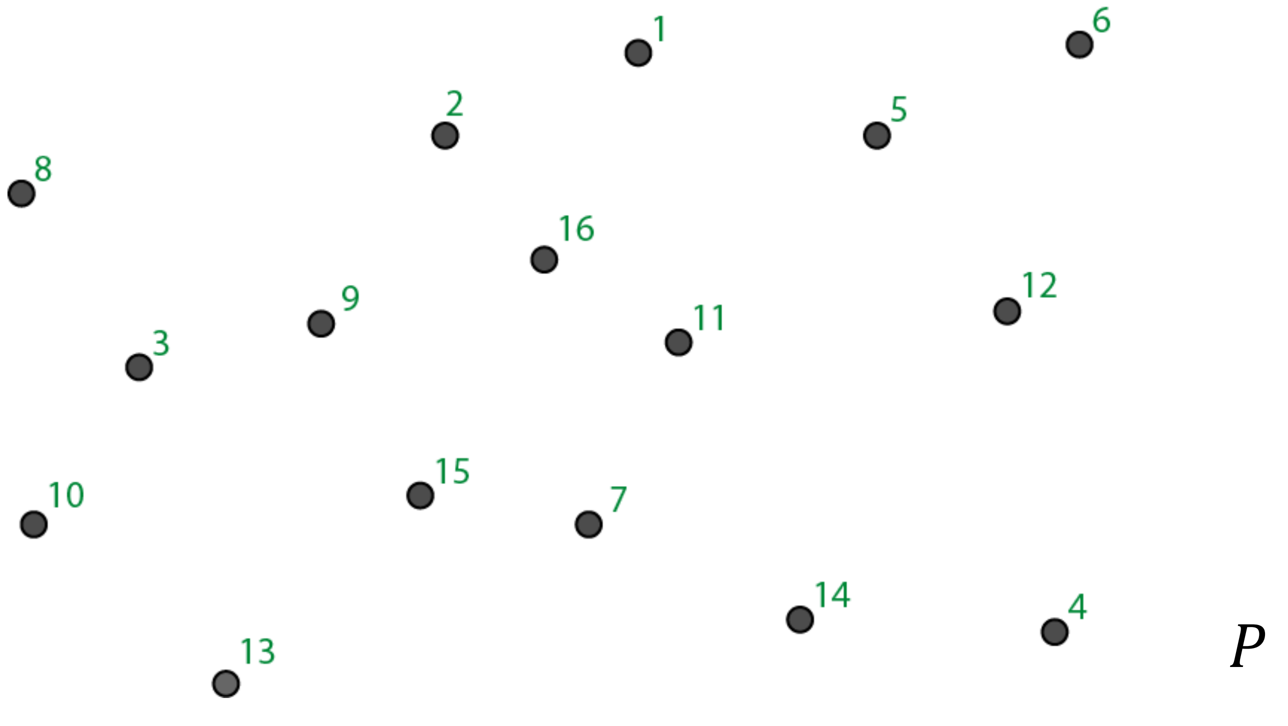
Ejemplo: par de puntos más cercano

- **Solución D&C**
  - **Entrada:** Conjunto p con n puntos



Ejemplo: par de puntos más cercano

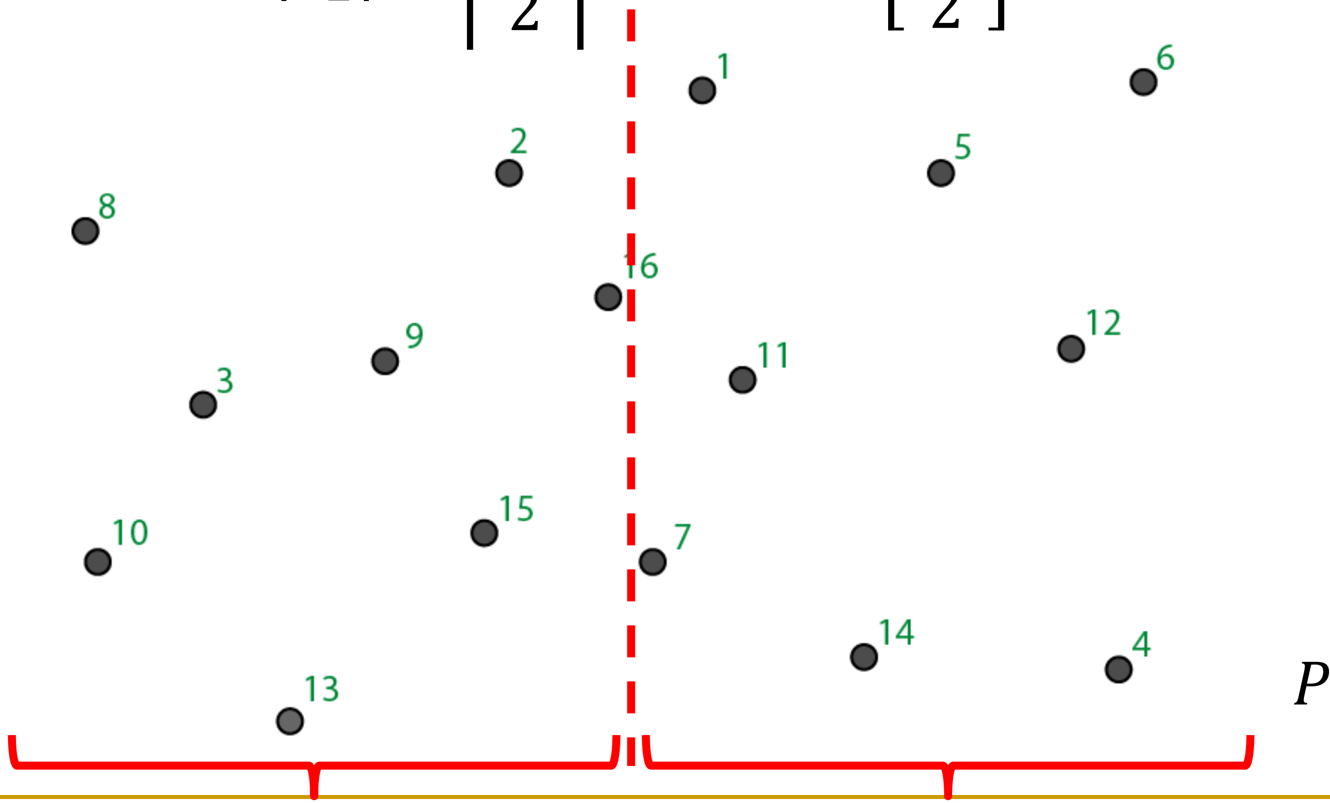
- **Divide?**



Ejemplo: par de puntos más cercano

- **Divide?**

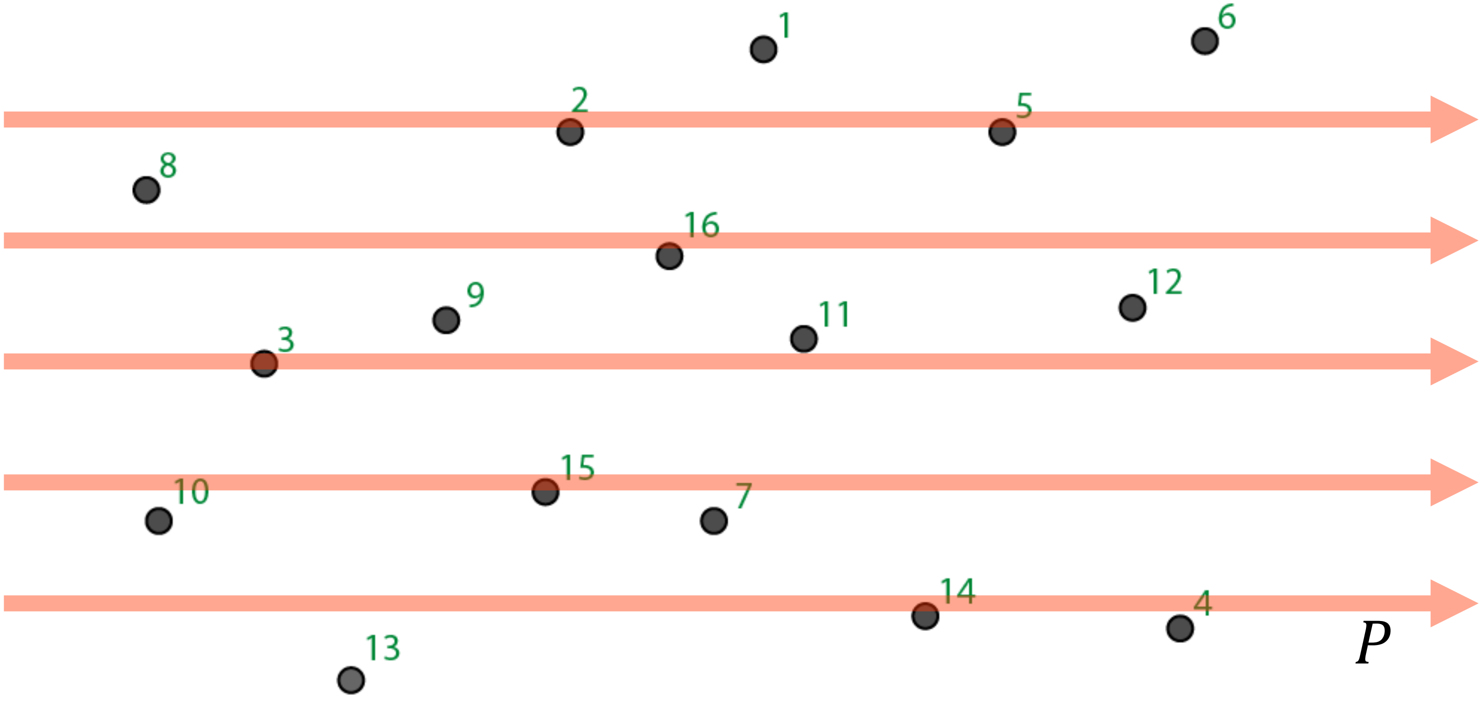
$$|P_L| = \left\lceil \frac{|P|}{2} \right\rceil \qquad |P_R| = \left\lfloor \frac{|P|}{2} \right\rfloor$$



Ejemplo: par de puntos más cercano

- Encontrar el punto medio en el arreglo ordenado según x:

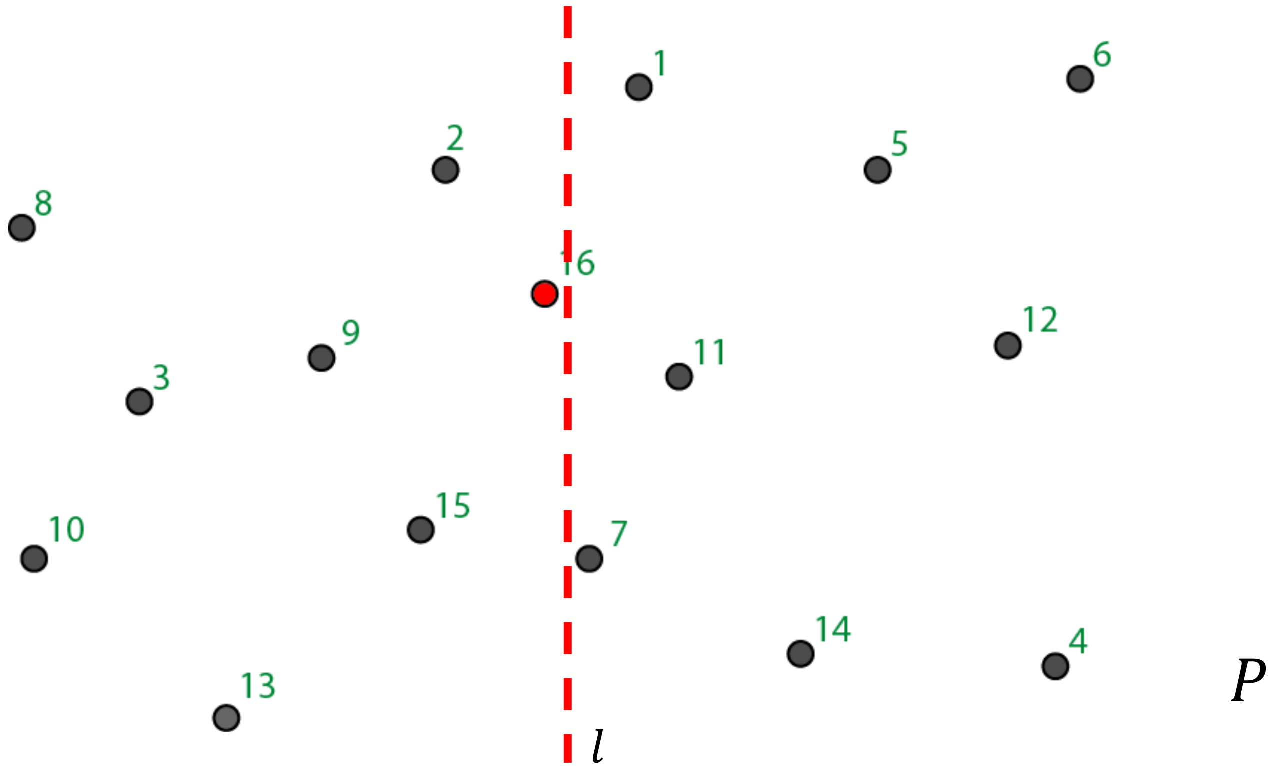
$$P = [8,10,3,13,9,15,2,16,7,1,11,14,5,12,4,6]$$



Ejemplo: par de puntos más cercano

- Encontrar el punto medio en el arreglo ordenado según x:

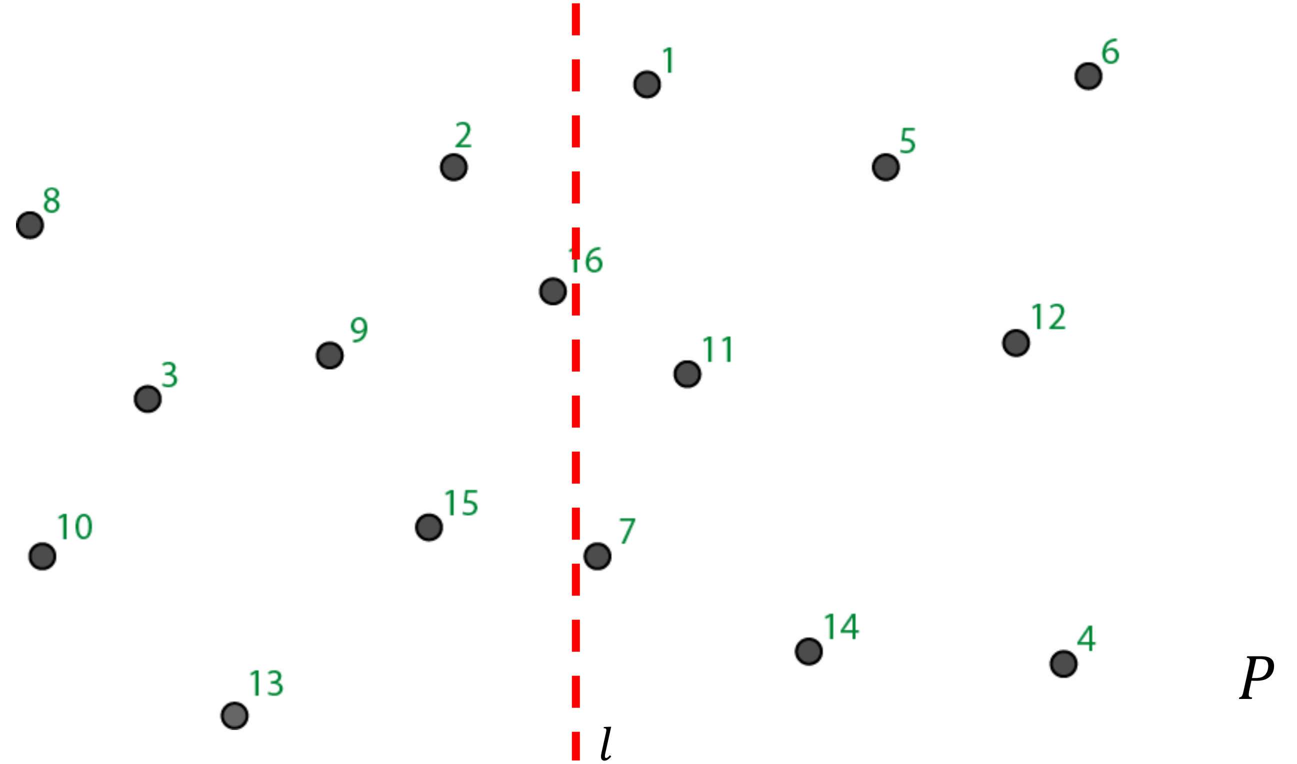
$$P = [8,10,3,13,9,15,2,16,7,1,11,14,5,12,4,6]$$



Ejemplo: par de puntos más cercano

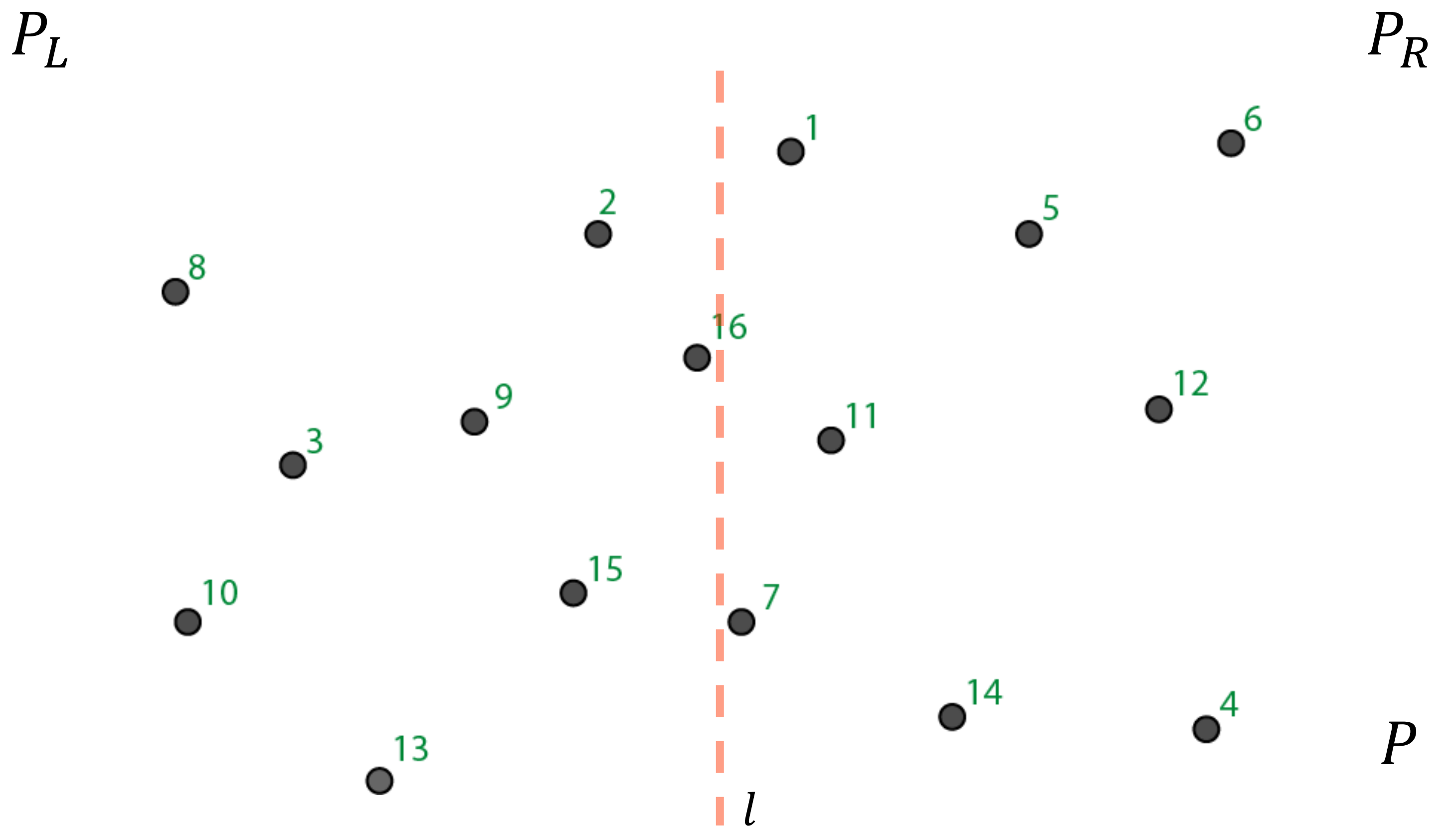
- Dividir en dos:

$P_L = [8,10,3,13,9,15,2,16]$ $P_R = [7,1,11,14,5,12,4,6]$



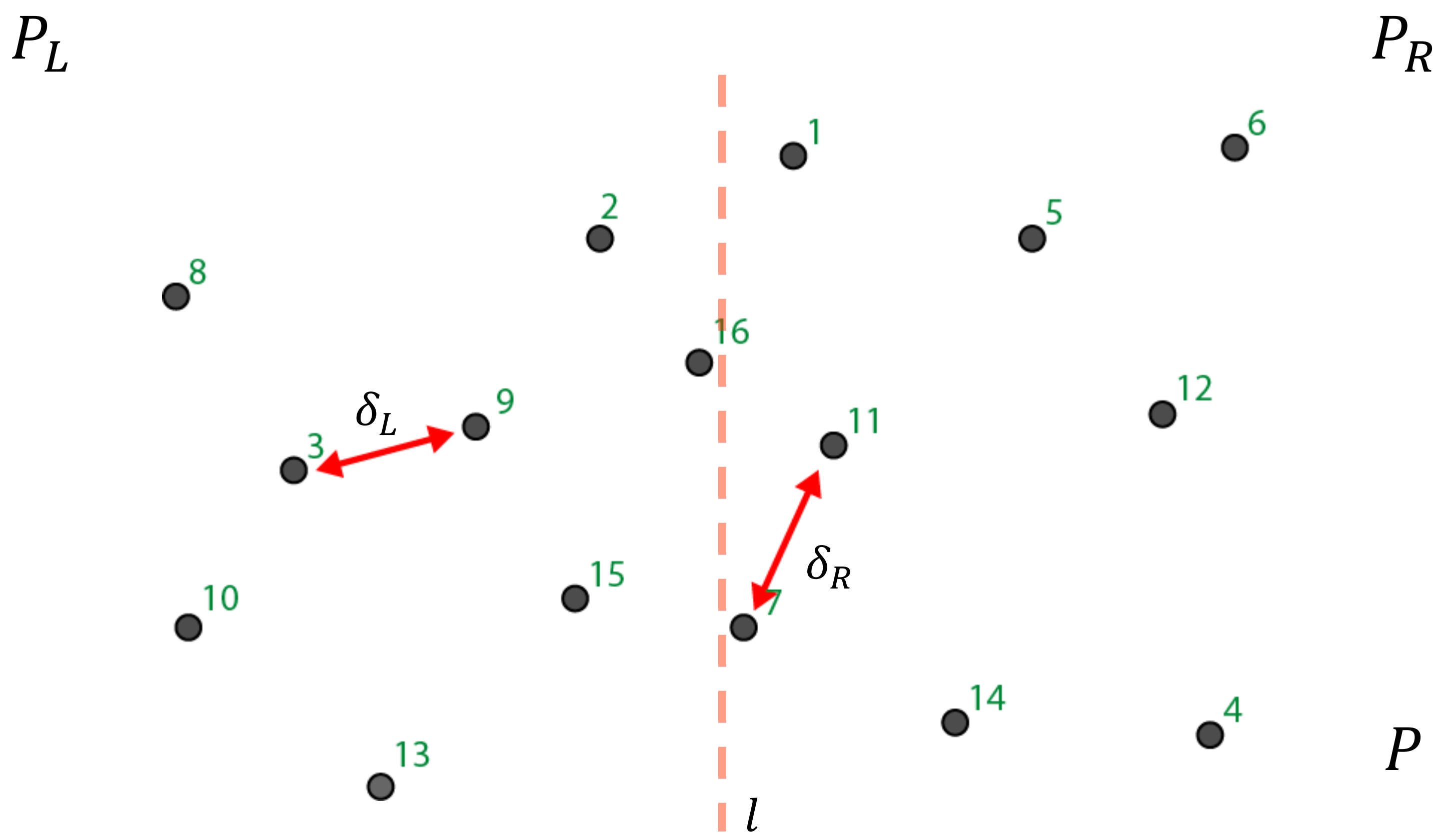
Ejemplo: par de puntos más cercano

- **Conquer:** Encontrar el par de puntos más cercano en ambas mitades (recursivo)



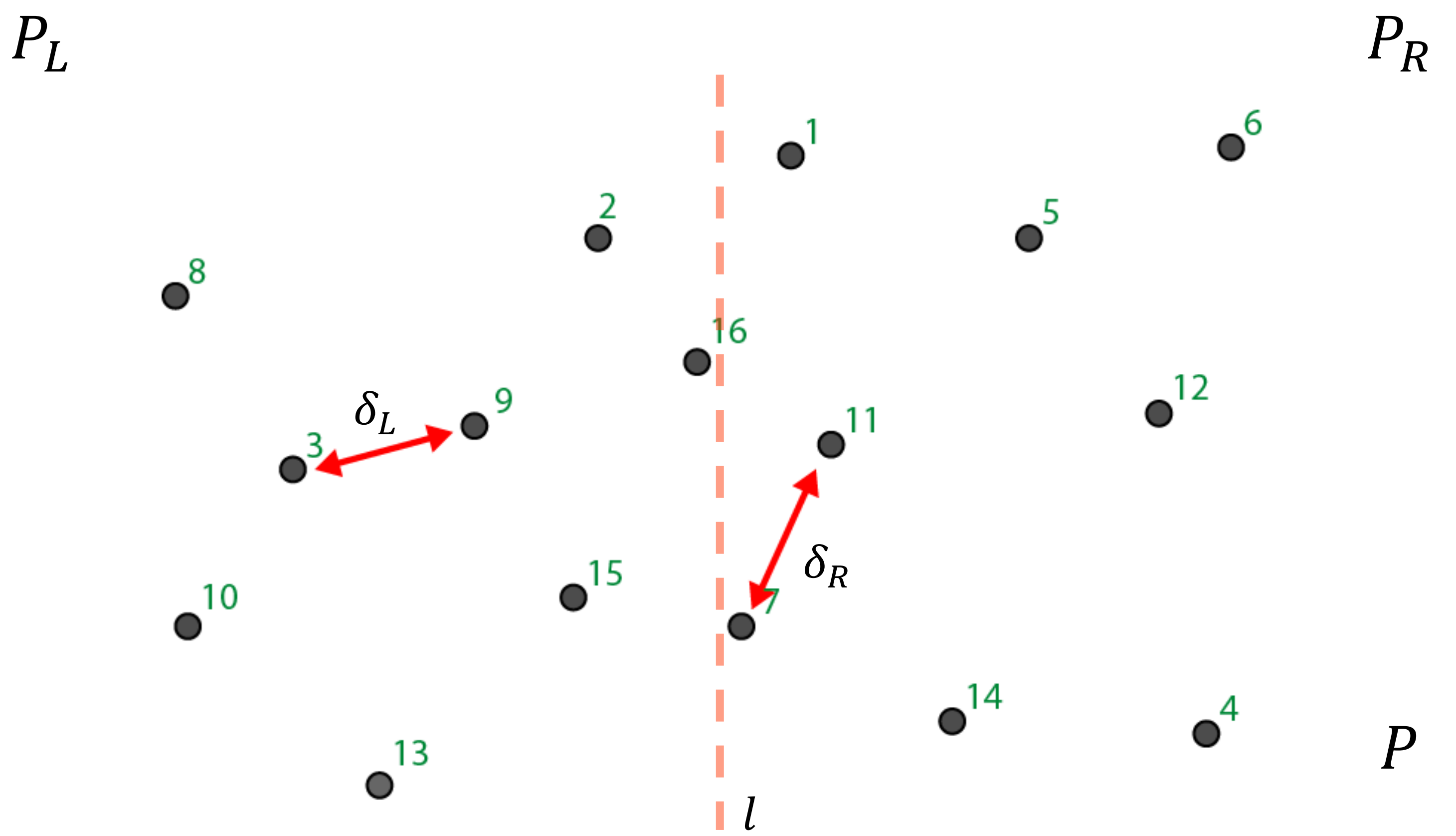
Ejemplo: par de puntos más cercano

- **Conquer:** Encontrar el par de puntos más cercano en ambas mitades (recursivo)



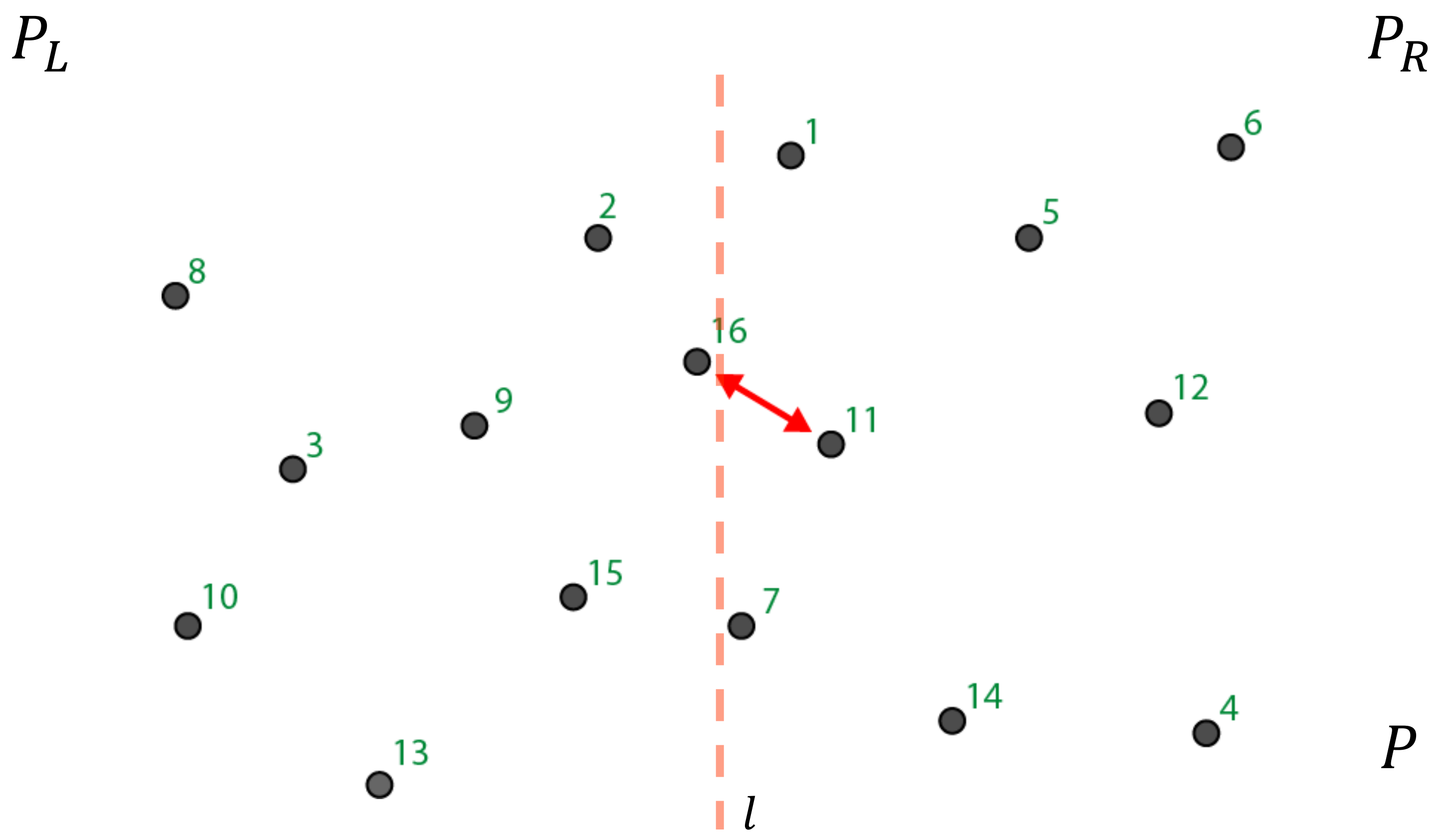
Ejemplo: par de puntos más cercano

- **Combine?** Podría retornar $\delta = \min(\delta_L, \delta_R)$



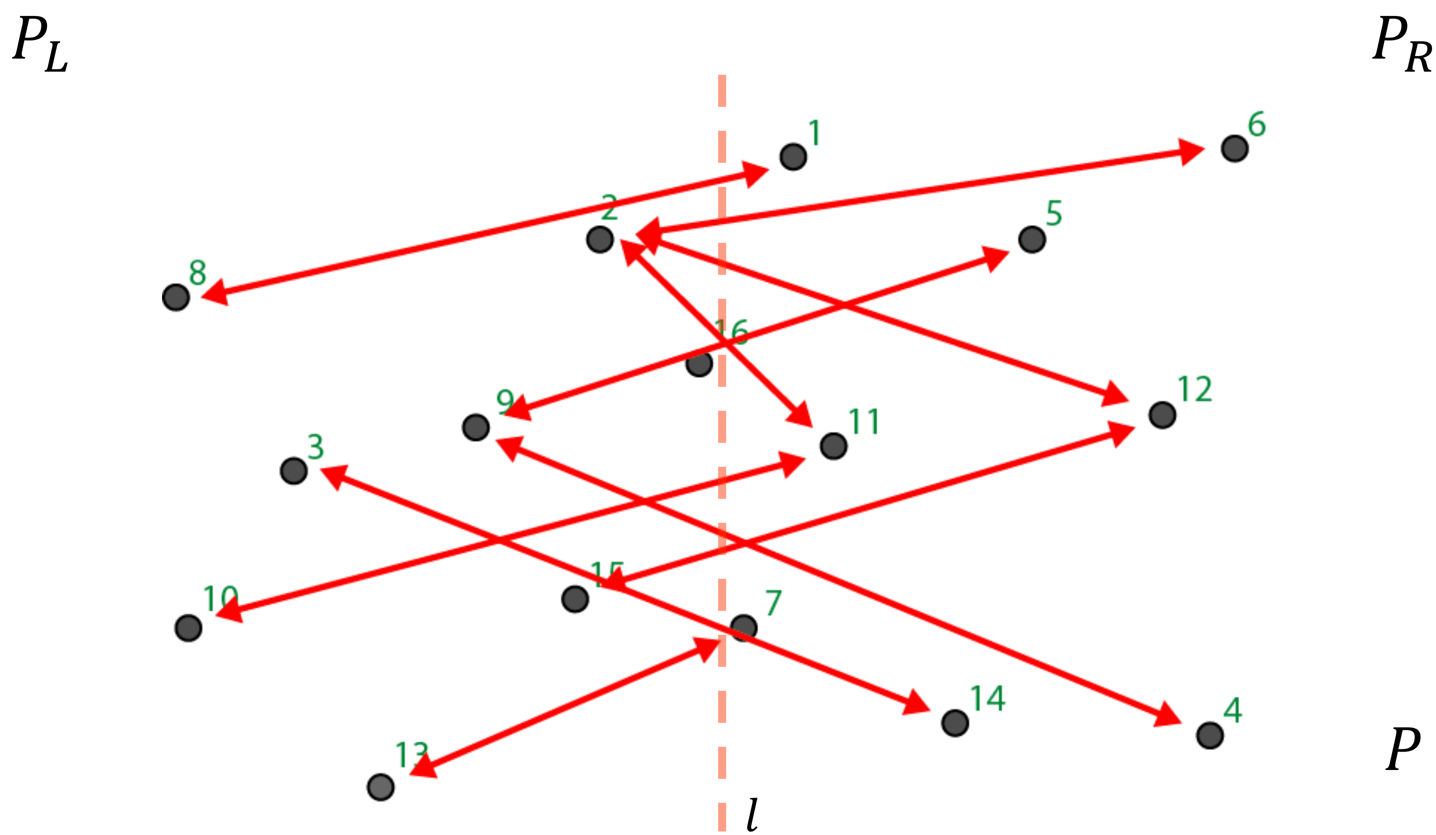
Ejemplo: par de puntos más cercano

- **Combine?** Podría retornar $\delta = \min(\delta_L, \delta_R)$ pero…



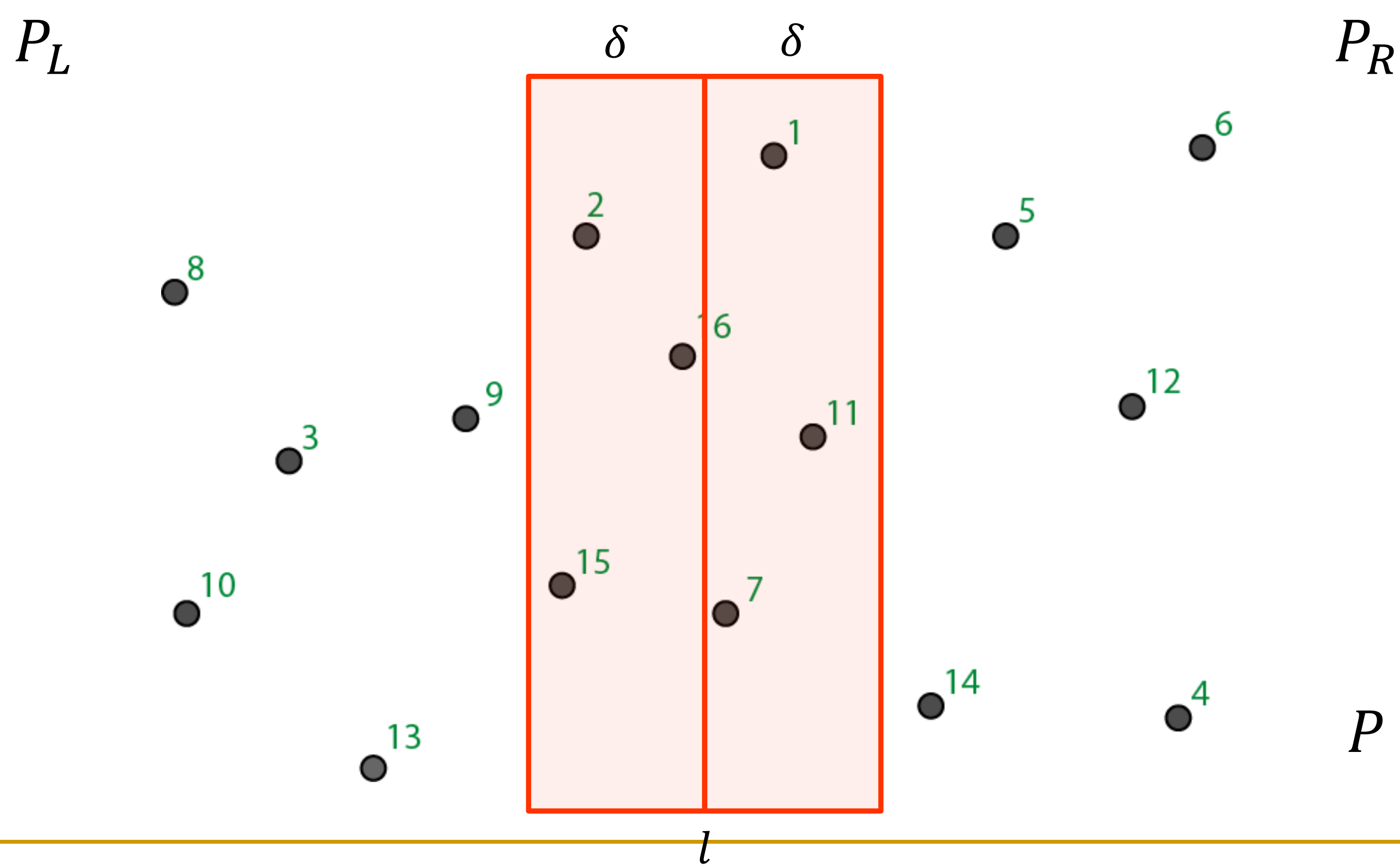
Ejemplo: par de puntos más cercano

- **Combine?** Podría retornar $\delta = \min(\delta_L, \delta_R)$ pero…

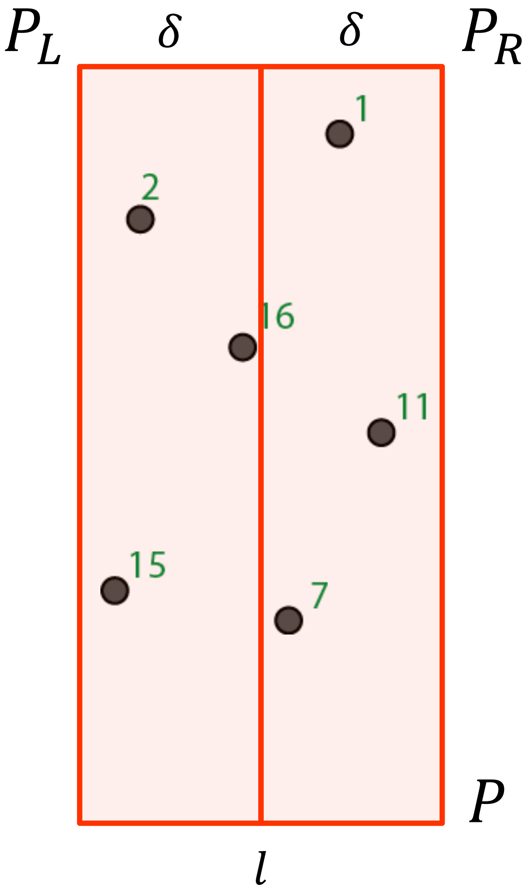


Ejemplo: par de puntos más cercano

- ¿Qué sabemos? La distancia es $\delta = \min(\delta_L, \delta_R)$ o menor



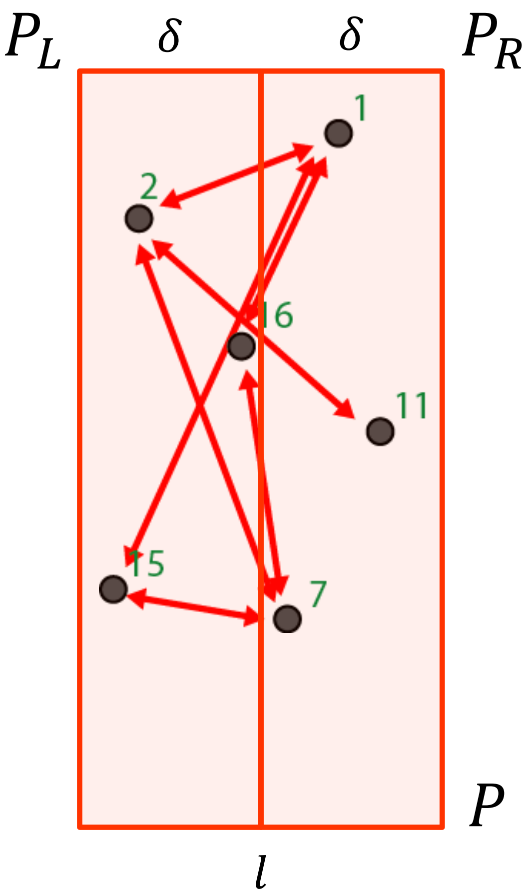
Ejemplo: par de puntos más cercano



- Creamos un arreglo Y' con los puntos de las bandas verticales ordenados por la coordenada y:

$$Y' = [1,2,16,11,15,7]$$

Ejemplo: par de puntos más cercano

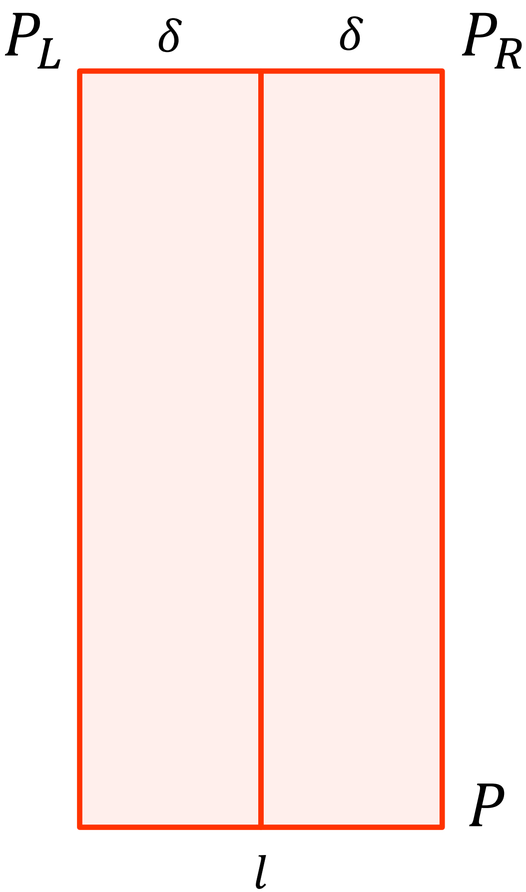


- Creamos un arreglo Y' con los puntos de las bandas verticales ordenados por la coordenada y:

$$Y' = [1,2,16,11,15,7]$$

- Para cada punto p en Y' buscamos otros puntos (también en Y') que estén a una distancia δ' menor a δ.

Ejemplo: par de puntos más cercano

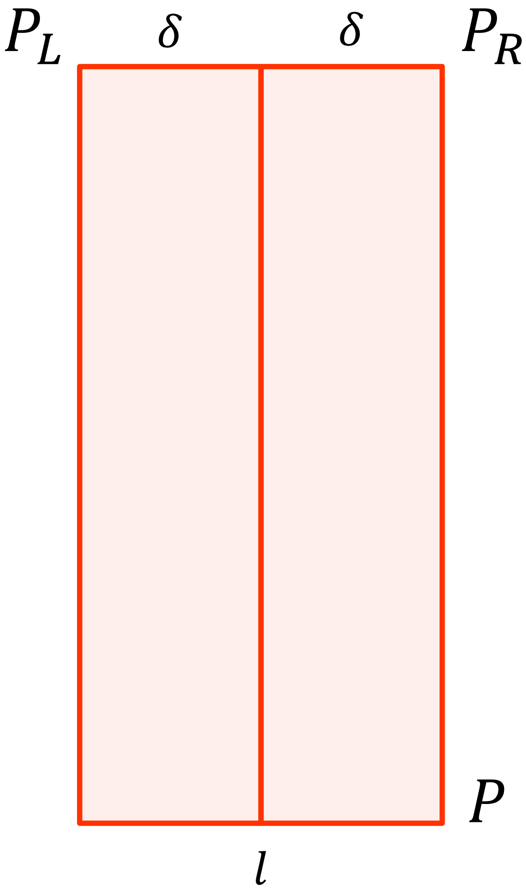


- Creamos un arreglo Y' con los puntos de las bandas verticales ordenados por la coordenada y:

$$Y' = [1,2,16,11,15,7]$$

- Para cada punto p en Y' buscamos otros puntos (también en Y') que estén a una distancia δ' menor a δ.
- Si $\delta' < \delta$, retornamos δ', y sino δ

Ejemplo: par de puntos más cercano

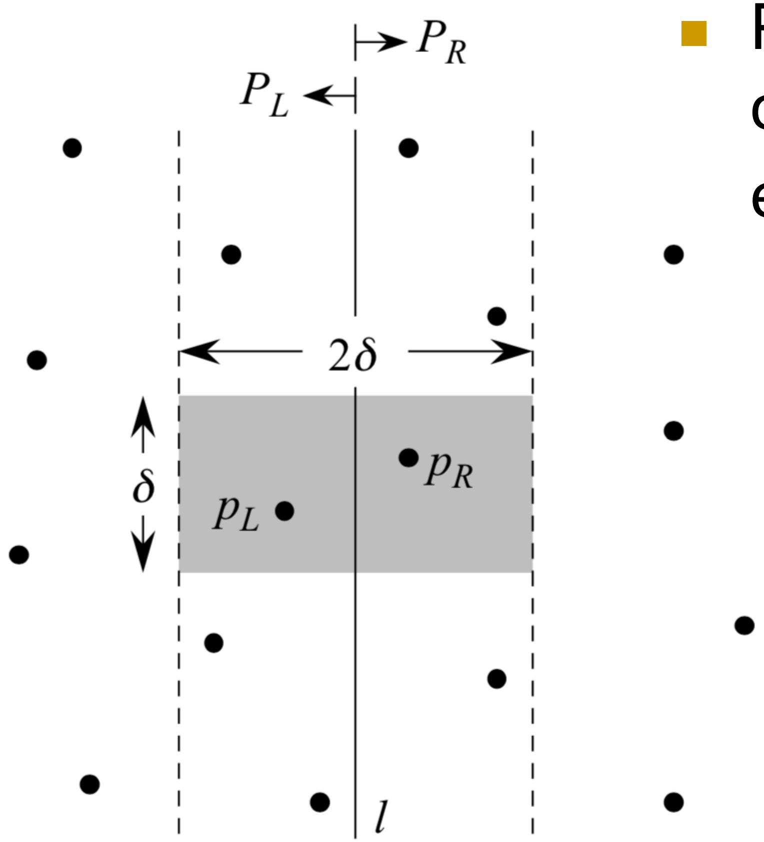


- Creamos un arreglo Y' con los puntos de las bandas verticales ordenados por la coordenada y:

$$Y' = [1,2,16,11,15,7]$$

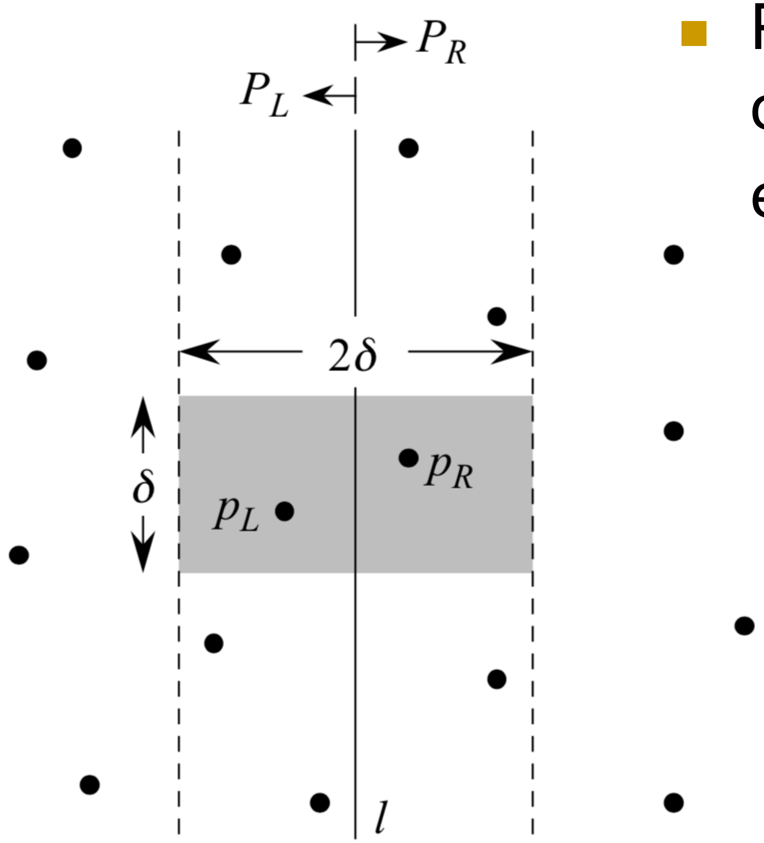
- Para cada punto p en Y' buscamos otros puntos (también en Y') que estén a una distancia δ' menor a δ.
- Si $\delta' < \delta$, retornamos δ', y sino δ

Ejemplo: par de puntos más cercano

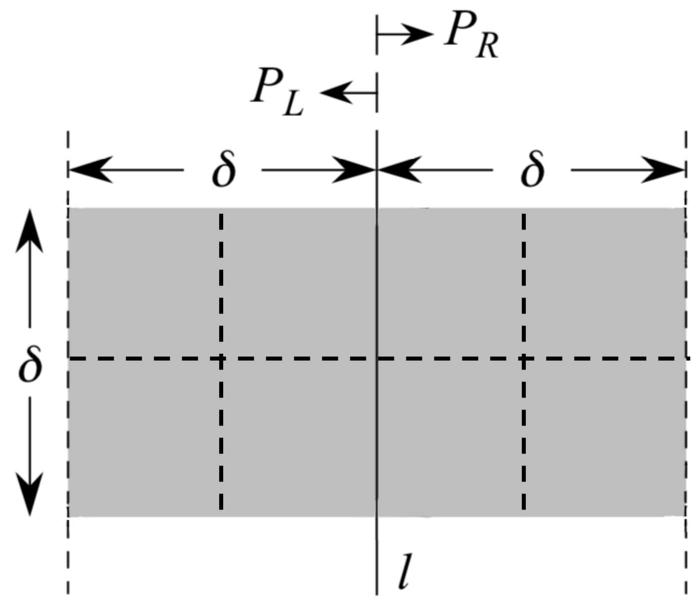


- Para cada punto p en Y' buscamos otros puntos (también en Y') que estén a una distancia δ' menor a δ.

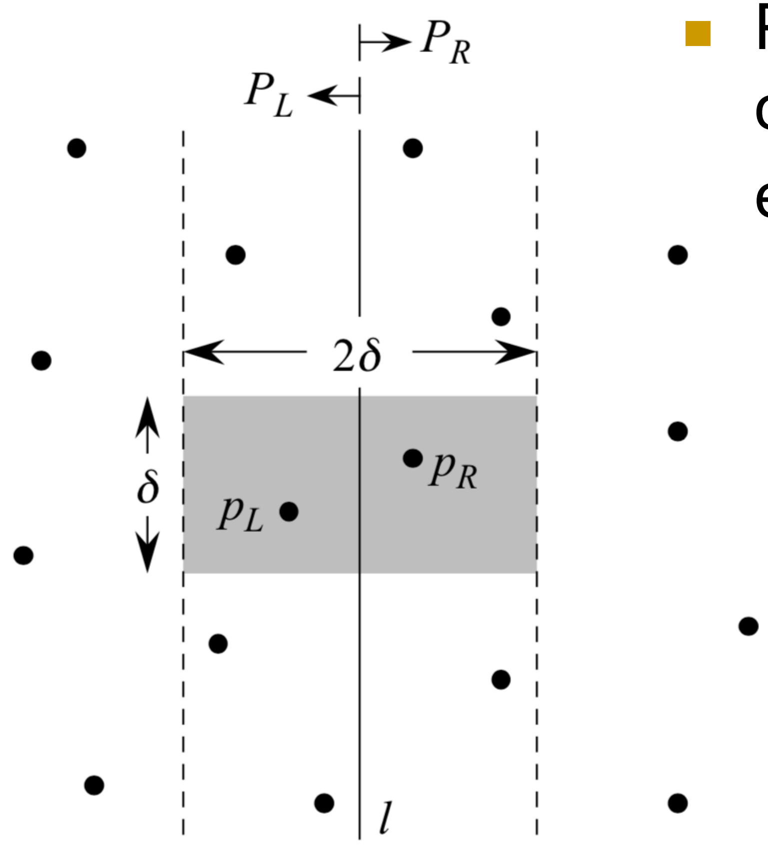
Ejemplo: par de puntos más cercano



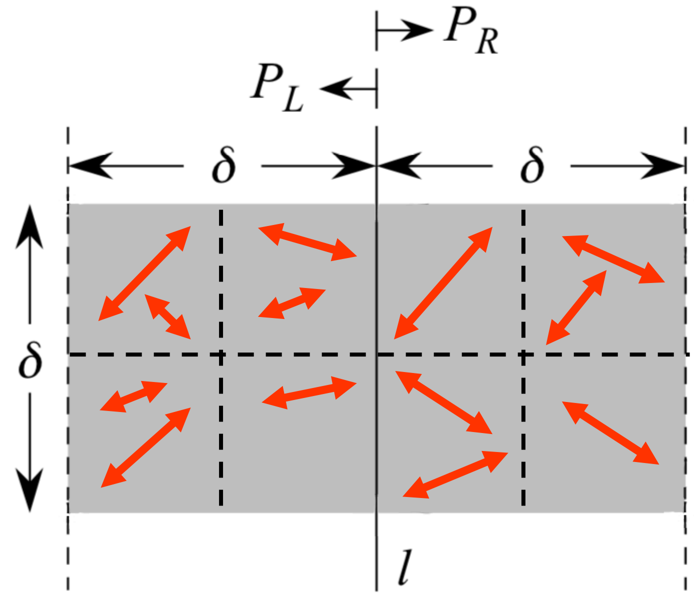
- Para cada punto p en Y' buscamos otros puntos (también en Y') que estén a una distancia δ' menor a δ.



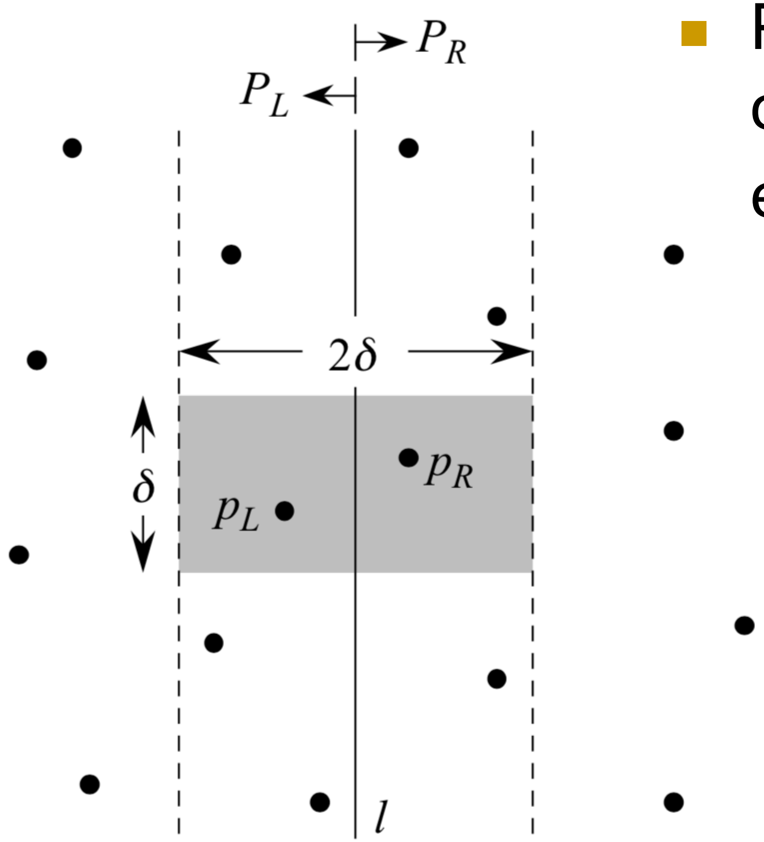
Ejemplo: par de puntos más cercano



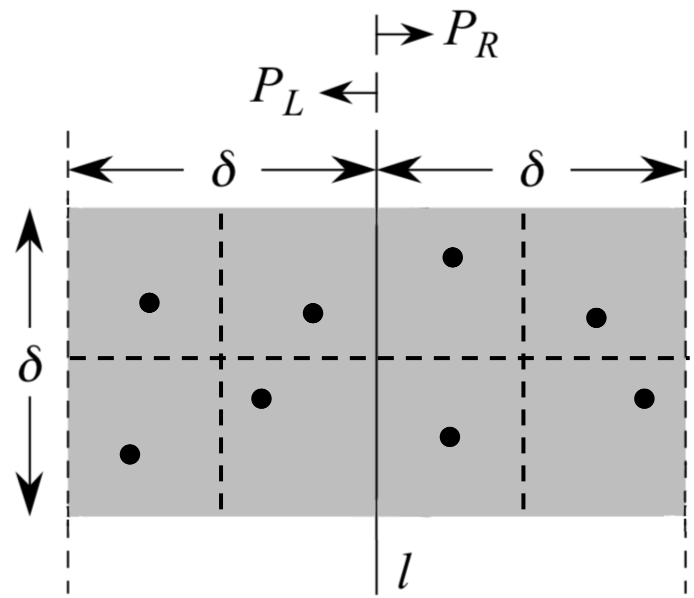
- Para cada punto p en Y' buscamos otros puntos (también en Y') que estén a una distancia δ' menor a δ.



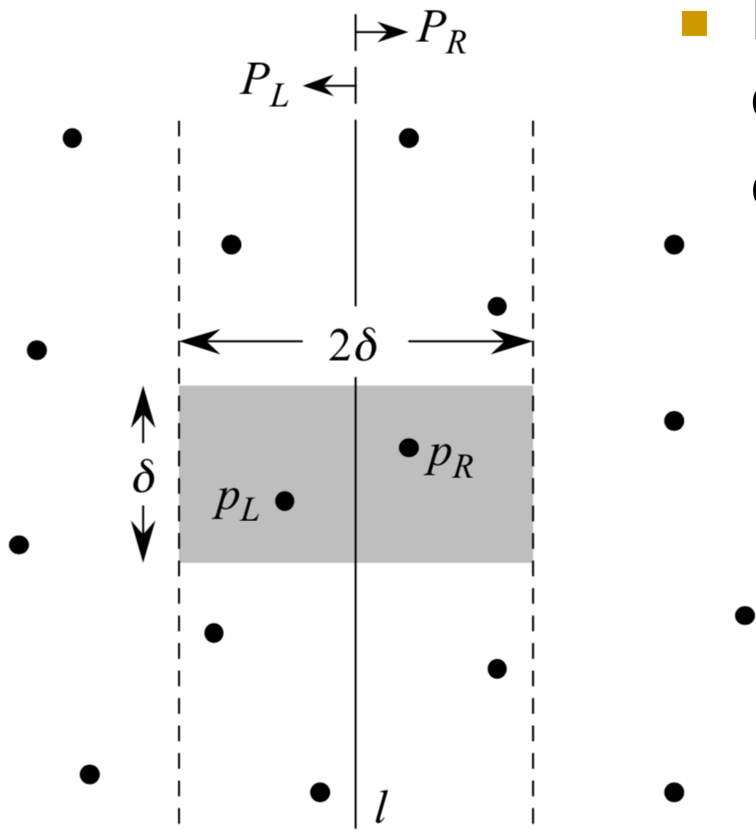
Ejemplo: par de puntos más cercano



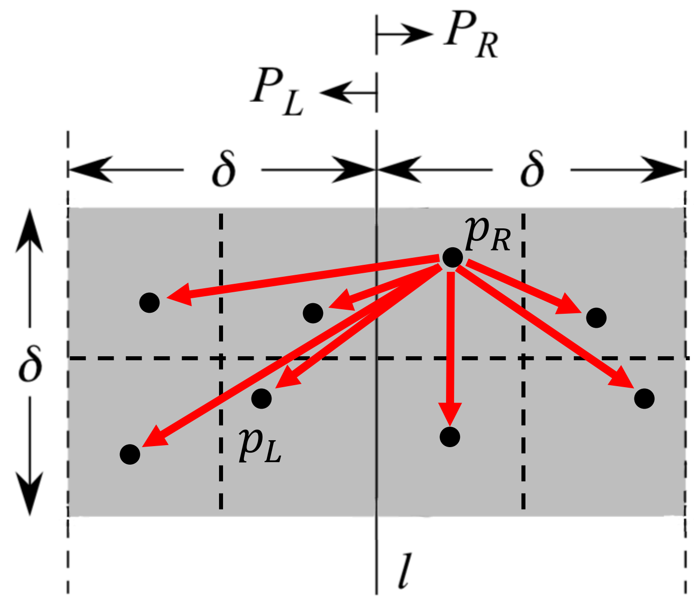
- Para cada punto p en Y' buscamos otros puntos (también en Y') que estén a una distancia δ' menor a δ.



Ejemplo: par de puntos más cercano

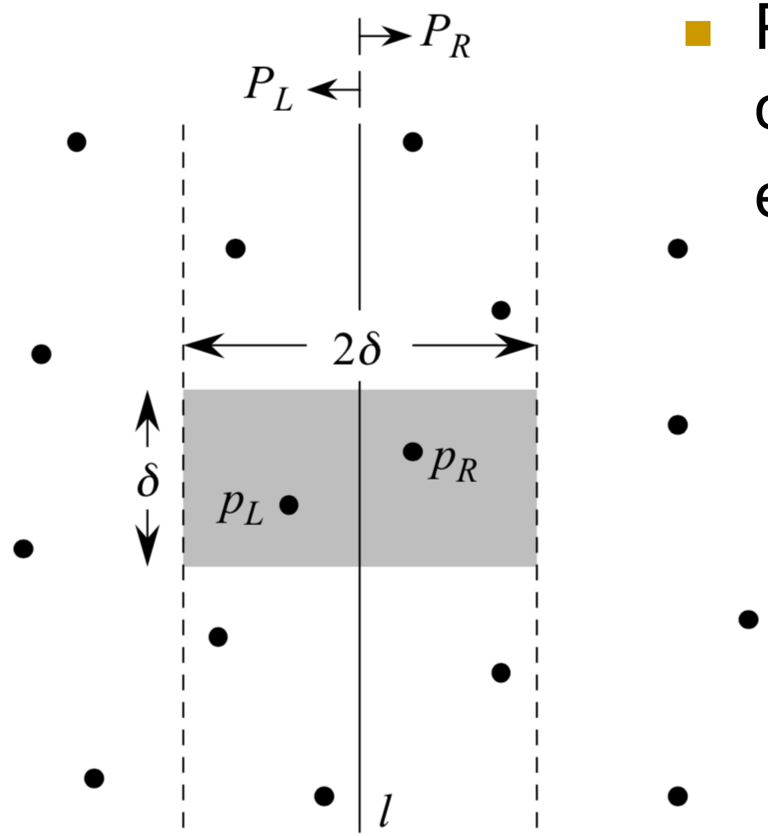


- Para cada punto p en Y' buscamos otros puntos (también en Y') que estén a una distancia δ' menor a δ.

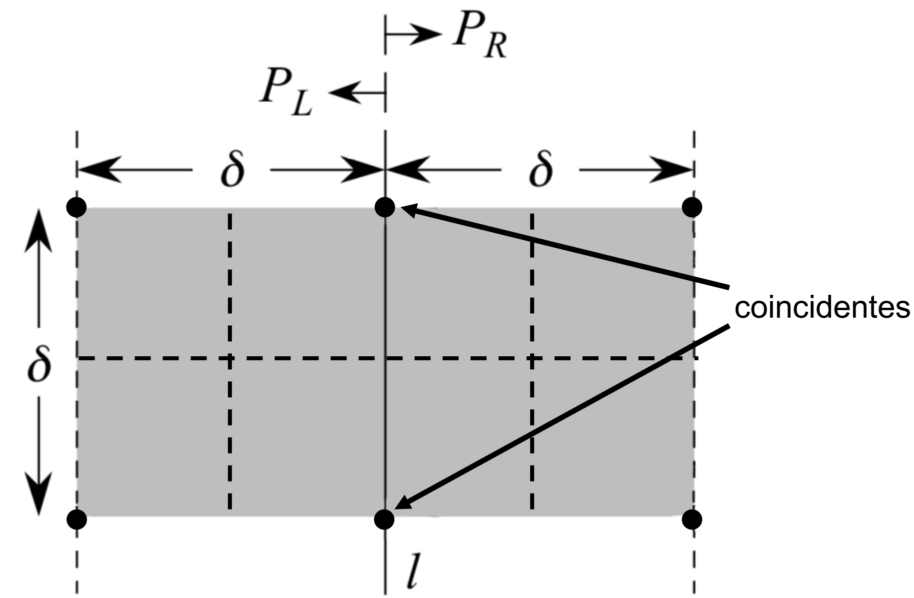


Máximo de 7 comparaciones

Ejemplo: par de puntos más cercano



- Para cada punto p en Y' buscamos otros puntos (también en Y') que estén a una distancia δ' menor a δ.



Caso extremo con 8 puntos

Ejemplo: par de puntos más cercano

- Aclaraciones generales:
  - Los llamados recursivos (***conquer***) se detienen cuando $|P| \leq 3$, de esa manera nos garantizamos que nunca vamos a intentar resolver sub-problemas que sólo contengan 1 punto.
  - Es necesario contar con dos versiones del arreglo de puntos P, uno ordenado según x, otro según y. Se los puede ***pre-ordenar*** en $O(n \lg n)$.
  - En cada llamado ***divide*** hay que preservar este orden para no aumentar la complejidad del algoritmo (cuidado con el arreglo ordenado según y). Se puede hacer en $O(n)$.

Ejemplo: par de puntos más cercano

- Análisis:

$$T(n) = \begin{cases} 2T\left(\frac{n}{2}\right) + O(n) & si\ n > 3 \\ O(1) & si\ n \leq 3 \end{cases}$$

Ejemplo: par de puntos más cercano

- Análisis:

$$T(n) = \begin{cases} 2T\left(\frac{n}{2}\right) + O(n) & si\ n > 3 \\ O(1) & si\ n \le 3 \end{cases}$$

- Podemos encontrar el par de puntos más cercanos en: $T(n) = O(n \lg n)$

Ejemplo: par de puntos más cercano

- Análisis:

$$T(n) = \begin{cases} 2T\left(\frac{n}{2}\right) + O(n) & si\ n > 3 \\ O(1) & si\ n \leq 3 \end{cases}$$

- Podemos encontrar el par de puntos más cercanos en: $T(n) = O(n \lg n)$
- Algoritmo generalizable a d-dimensiones con complejidad $T(n,d) = O(n\,(\lg n)^{d-1})$