Pares más cercanos

Pares más cercanos

Entrada: Un arreglo *X* de *N* puntos en un plano cartesiano bidimensional

Salida: La menor distancia euclidiana entre los pares de puntos de *X*

Primera solución: búsqueda por fuerza bruta!

```
function closestPairs(P, N):
   minD = ∞
   for i = 0 to N-2:
      for j =i+1 to N-1:
         if dist(P_i, P_j) < mindD:
            minD = dist(P_i, P_j)
   return minD
```

¿Cuál es la eficiencia? $O(N^2)$

¿Se puede hacer mejor?

Pares más cercanos

Pensemos primero que pasaría sino fueran puntos 2D sino 1D. En este caso el número de pares también es cuadrático, pero el problema se puede resolver fácilmente si se ordenan los puntos según su única dimensión.

De esta manera:

Ordenamiento = $O(N*log(N))$ + Procedimiento lineal $O(N) = O(N*log(N))$

¿Cómo podemos aplicar un principio similar en el caso 2D?

Solución mediante Divide & Vencerás

```
P = P.sort() //según coordernada X

closestPair(0, N-1)

function closestPair(i, j):
   if i = j: //un punto
      return ∞
   else if j-i = 1 //un único par
      return dist(P_i, P_j)
   else:
      dL = closestPair(i, (i+j)/2) //mitad izquierda del subarreglo
      dR = closestPair(1+(i+j)/2, j) //mitad derecha del subarreglo
      delta = min(dL, dR)
      dS = closestSplitPair(i, j, delta)
      return min(delta, dS)
```

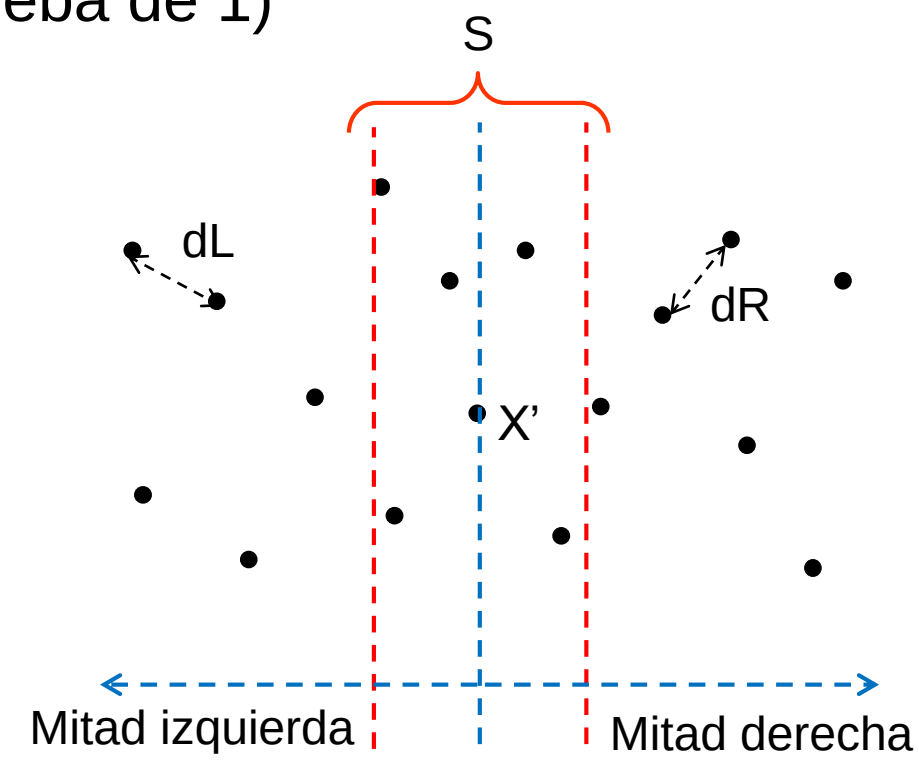
Solución mediante Divide & Vencerás

```
closestSplitPair(i, j, delta){
   x’ = coordenada X del punto (i+j)/2
   S = puntos de P ubicados entre los índices i,j y en [x’-delta, x’+delta]
   SY = S.sort() //según coordenada Y
   minD = delta
   for p = 0 to SY.lenght – 1:
      q = p+1
      while q < SY.lenght – 1 and q <= p+7:
          if dist(SY_p, SY_q) < minD
              minD = dist(SY_p, SY_q)
          q++
   return minD
```

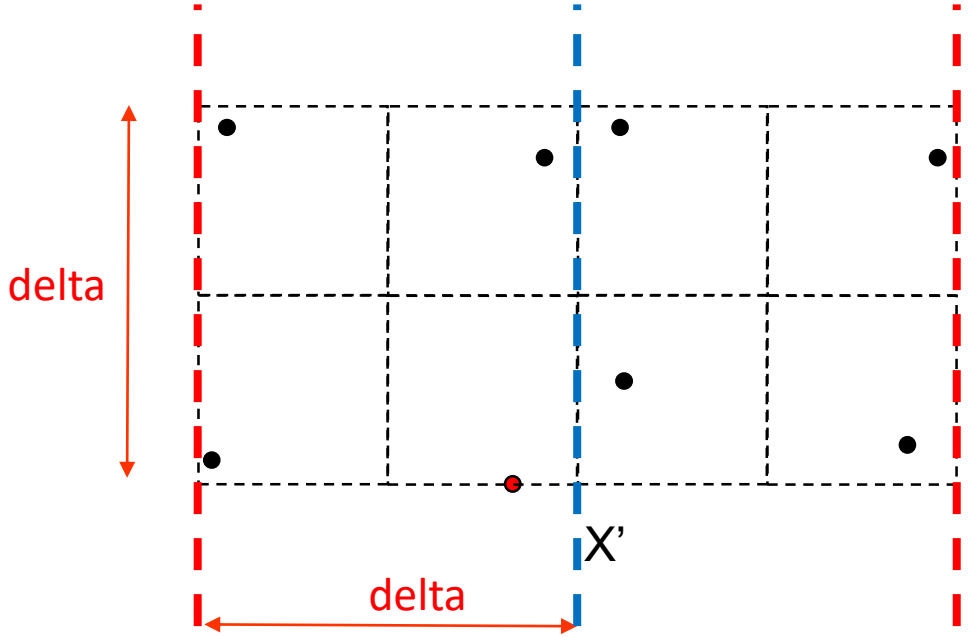
Dados X, Y tal que dist(X, Y) < delta, se necesita que se cumplan dos cosas para que este algoritmo funcione:

1) X, Y se encuentren en S
2) X, Y estén separados como máximo **7** posiciones el uno del otro en SY

Prueba de 1)



Prueba de 2)



Complejidad

```
closestSplitPair(i, j, delta){
   x’ = coordenada X del punto (i+j)/2
   S = puntos de P ubicados entre los índices i,j y en [x’-delta, x’+delta]
   SY = S.sort() //según coordenada Y
   minD = delta
   for i = 0 to SY.lenght – 1:
      j = i+1
      while j < SY.lenght – 1 and j <= i+7:
          if dist(SY_i, SY_j) < minD
              minD = dist(SY_i, SY_j)
          j++
   return minD
```

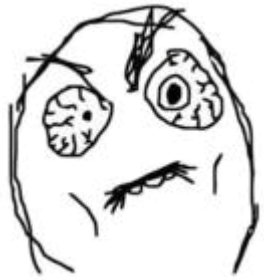
O(N)

*O(N*log(N))*

*O(N*7*C)* → *O(N)*

La complejidad resultante es $O(N*\log(N)^2)$

¿Se puede hacer mejor?



Si, al usar un procedimiento de divide + merge como el del *mergeSort* embebidos en las funciones *closestPair* y *closestSplitPair* para que el ordenamiento de SY sea *O(N)* con lo que la complejidad general sería *O(N*log(N))*