

UNIVERSIDAD NACIONAL DE INGENIERÍA

Análisis y Diseño de Algoritmos

Selección en tiempo lineal, Karatsuba y el par más cercano

Más divide y vencerás

Gerson Garrido

Semana 3

Hoy

1. Quickselect: partir y seguir por un solo lado
2. La mediana de medianas: un pivote con garantía
3. Una recurrencia con dos tamaños, y por qué da $O(n)$
4. Karatsuba: tres productos en lugar de cuatro
5. El par más cercano: combinar mirando la franja

> **Lectura**
>
> Apuntes de la semana 3. Erickson, secs. 1.8 y 1.9. CLRS, *Medians and Order Statistics*. Código en `code/week-03`.

SECCIÓN 01

Quickselect

01



El k-ésimo menor

k-ésimo menor

El valor que queda en la posición k si el arreglo se ordena. Contamos desde 0: $k = 0$ es el mínimo, $k = n - 1$ el máximo. La **mediana** es $k = (n - 1)$ div 2.

- Ordenar y leer: $\Theta(n \log n)$. Calcula el lugar de todos, cuando pedimos uno.
- Para el mínimo basta una pasada. ¿Y para la mediana?
- Se puede en $O(n)$, en el peor caso.

Partir en tres, seguir por un lado

pseudocódigo

```
algoritmo quickselect(A, k)
    mientras |A| > 1 hacer
        pivote ← A[0]
        L, E, G ← partir(A, pivote)
        si k < |L| entonces
            A ← L
        sino si k < |L| + |E| entonces
            devolver pivote
        sino
            k ← k - |L| - |E|
            A ← G
        fin si
    fin mientras
    devolver A[0]
fin algoritmo
```

- L: menores que el pivote.
- E: iguales.
- G: mayores.

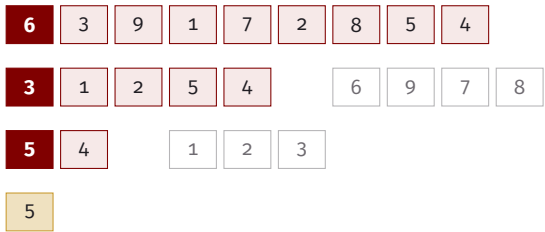
Invariante

$0 \leq k < |A|$, y el k-ésimo menor del A actual es la respuesta.

Ordenado, A es: todo L, todo E, todo G. La posición k cae en uno de los tres. **Termina:** el pivote queda en E, así que A se acorta.

Quickselect, paso a paso

$A = 6, 3, 9, 1, 7, 2, 8, 5, 4$, $k = 4$. Respuesta esperada: 5.



$9 + 5 + 2 = 16$ comparaciones, una por elemento en cada partición.

El peor caso: $\Theta(n^2)$

build/quickselect_trace

```
arreglo ordenado, k = n - 1, primer elemento como pivote
       n   comparaciones   n(n+1)/2 - 1
    1000          500499         500499
    2000         2000999        2000999
    4000         8001999        8001999
    8000        32003999       32003999
```

- El mínimo como pivote: L vacío.
- Tamaños $n, n-1, \ldots, 2$.
- Es un conteo, no un tiempo: no depende de la máquina.
- Depende de k: ordenado con $k = 0$ cuesta n. Al revés con $k = 0$ es el malo.

Todo depende del pivote

Y una regla fija siempre tiene una entrada mala.

Qué necesitamos en realidad

Supongamos un pivote que, en $O(n)$, deje siempre a lo sumo αn en cada lado, con $\alpha < 1$ fijo.

$$cn + c\alpha n + c\alpha^2 n + \cdots \leq \frac{cn}{1-\alpha} = O(n)$$

- No hace falta la mediana exacta.
- Con $\alpha = 0{,}99$ también sería lineal, con una constante enorme.

Idea clave

Basta un pivote que siempre descarte una **fracción constante**. Una suma geométrica hace el resto.

SECCIÓN 02

Mediana de medianas

02

El algoritmo *(median of medians)*

pseudocódigo

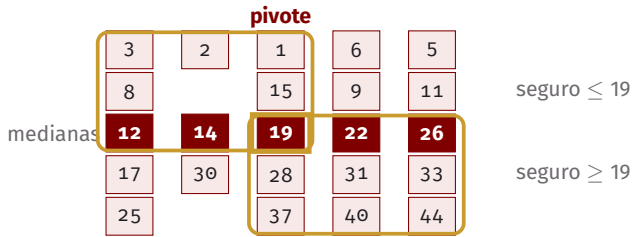
```
algoritmo seleccionar(A, k)
    si |A| ≤ 5 entonces
        ordenar A
        devolver A[k]
    fin si
    M ← medianas_de_grupos(A)
    pivote ← seleccionar(M, (|M| - 1) div 2)
    L, E, G ← partir(A, pivote)
    si k < |L| entonces
        devolver seleccionar(L, k)
    sino si k < |L| + |E| entonces
        devolver pivote
    sino
        devolver seleccionar(G, k - |L| - |E|)
    fin si
fin algoritmo
```

1. Grupos de 5 por posición: $\lceil n/5 \rceil$ grupos. El último puede ser corto.
2. La mediana inferior de cada grupo.
3. Con esas medianas, **recursión**: su mediana inferior es el pivote.
4. Partir y seguir por un lado.

La correctitud no depende de qué pivote sea, solo de que esté en A. Lo que depende de él es el costo.

Una vuelta con 23 elementos

12 3 25 8 17 | 40 6 31 22 9 | 15 37 1 28 19 | 44 11 33 5 26 | 14 30 2



Grupos ordenados, y columnas ordenadas por su mediana. El grupo corto es el de 14.
Reales: 13 elementos ≤ 19 y 11 elementos ≥ 19. La cota promete 3,9 con $n = 23$.

El lema

Lema

Sea $m = \lceil n/5 \rceil$ y p la mediana inferior de las m medianas. Hay al menos $\frac{3n}{10} - 3$ elementos $\leq p$, y al menos $\frac{3n}{10} - 3$ elementos $\geq p$. Entonces $|L| \leq \frac{7n}{10} + 3$ y $|G| \leq \frac{7n}{10} + 3$.

Técnica: contar. Usamos posiciones, no valores, y los repetidos no molestan.

- **Medianas:** p tiene lugar $\lceil m/2 \rceil$. Son $\lceil m/2 \rceil$ medianas $\leq p$ y $\lfloor m/2 \rfloor + 1 \geq \lceil m/2 \rceil$ medianas $\geq p$.
- **Por grupo:** un grupo completo da 3 elementos de ese lado, contando su mediana.
- **Grupo corto:** a lo sumo uno. No le contamos nada. Quedan $3(\lceil m/2 \rceil - 1)$ elementos.
- **Cuenta:** $\lceil m/2 \rceil \geq m/2 \geq n/10$, así que hay al menos $3(n/10 - 1) = 3n/10 - 3$.
- **Tamaños:** ninguno de los $\geq p$ está en L: $|L| \leq n - (3n/10 - 3)$. Igual con G. ■

Por qué partir en tres

Partir en dos, menores y mayores o iguales:

- Todos iguales: todos del mismo lado.
- Cada vuelta quita solo uno.
- Al menos cuadrático, con el pivote "bueno".

Partir en tres:

- E absorbe todos los iguales.
- El lema habla de $\leq p$ y $\geq p$.
- Vale con cualquier cantidad de repetidos.

Ojo con el -3

"Al menos $3n/10$" es solo aproximado. Con $n = 23$: $3n/10 = 6{,}9$, y el lema promete $3{,}9$. El -3 paga el grupo corto y el redondeo.

El lema, medido

build/pivot_bound

```
n = 1000, cota 3n/10 - 3 = 297
entrada                <= piv   >= piv
al azar                   507      494
ordenado                  498      503
todos iguales            1000     1000
tres valores              673      659
construida                300      701
```

- Las entradas comunes quedan cerca de la mitad.
- `construida`: la mitad de los grupos tiene sus tres menores entre los más chicos de todos. Llega a 300.
- El $3/10$ no se puede mejorar para esta regla, salvo por la constante.

La recurrencia

Sea $T(n)$ el mayor costo con **a lo sumo** n elementos. Con $n \geq 6$:

- Medianas de los grupos y partición: $O(n)$.
- Recursión sobre las medianas: $\lceil n/5 \rceil$.
- Recursión sobre un lado: a lo sumo $\frac{7n}{10} + 3$.

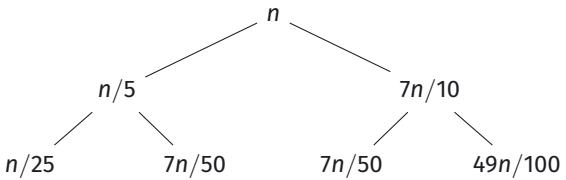
$$T(n) \leq T(\lceil n/5 \rceil) + T(\lfloor 7n/10 \rfloor + 3) + cn$$

Sin redondeos: $T(n) \leq T(n/5) + T(7n/10) + cn$.

Error común

Aplicar el teorema maestro. Pide a subproblemas **del mismo tamaño**, y aquí hay dos distintos. "Tomo $b = 5$": da $\Theta(n)$ por casualidad, y con $T(n/3) + T(2n/3) + n$ daría $\Theta(n)$ también, y es falso.

La intuición: el árbol



$$n \cdot (1 + \tfrac{9}{10} + (\tfrac{9}{10})^2 + \dots) = 10n$$

La clave

$\frac{1}{5} + \frac{7}{10} = \frac{9}{10} < 1$. Juntos son más pequeños que el problema, y da igual que sean desiguales. Manda la raíz.

Sin redondeos: $T(n) \leq 10n$

Afirmación. Si $T(n) \leq T(n/5) + T(7n/10) + n$, con $T(n) \leq 10n$ en los valores pequeños, entonces $T(n) \leq 10n$.

Técnica: inducción fuerte.

$$\begin{aligned} T(n) &\leq 10 \cdot \frac{n}{5} + 10 \cdot \frac{7n}{10} + n \\ &= 2n + 7n + n \\ &= 10n \end{aligned}$$

Se prueba la constante exacta, como la semana pasada. El 10 es $1/(1 - 9/10)$.

Pero ignora los redondeos, y las inducciones se rompen justo por esos detalles.

La prueba completa

Teorema. `seleccionar` cuesta $O(n)$ en el peor caso.

- **Pequeños:** para $n < 80$, $T(n)$ es finito. Sea a su máximo. El programa ordena solo con $n \leq 5$; el 80 es de la demostración.
- **Constante:** $d = \max(20c, a)$. Afirmamos $T(n) \leq dn$ para $n \geq 1$.
- **Base:** si $1 \leq n < 80$, $T(n) \leq a \leq an \leq dn$.
- **Paso,** $n \geq 80$. Los dos tamaños son menores que n:

$$T(n) \leq d\left(\tfrac{n}{5} + 1\right) + d\left(\tfrac{7n}{10} + 3\right) + cn = d\left(\tfrac{9n}{10} + 4\right) + cn$$

- Con $n \geq 80$, $4 \leq n/20$, así que $\frac{9n}{10} + 4 \leq \frac{19n}{20}$:

$$T(n) \leq dn - \tfrac{dn}{20} + cn \leq dn \quad \text{porque } d \geq 20c. \blacksquare$$

Por qué 5 y no 3

Grupos de 3. Cada grupo da 2 de cada lado. Hay al menos $n/3 - 2$ de cada lado.

$$T(n) \leq T(n/3) + T(2n/3) + cn, \qquad \tfrac{1}{3} + \tfrac{2}{3} = 1$$

- El árbol ya no baja: cada nivel completo suma cn, con $\Theta(\log n)$ niveles. $\Theta(n \log n)$.
- **Grupos de 7:** $\frac{1}{7} + \frac{5}{7} = \frac{6}{7} < 1$. También sirve, con más trabajo por grupo.

Con precisión

Con grupos de 3, **este argumento** solo da $O(n \log n)$. No probamos que ningún algoritmo con grupos de 3 sea lineal: hay variantes modificadas que lo logran.

Las dos recurrencias, con números

build/recurrence_table

```
         n    T5(n)/n    T3(n)/n
     10000       6.31       9.29
     40000       7.05      11.61
    160000       7.63      13.76
    640000       8.10      15.89
   2560000       8.49      18.11
  10240000       8.79      20.27
mayor T5(n)/n con n hasta 10240000: 8.79
```

- $T_5(n) = T_5(\lceil n/5 \rceil) + T_5(\lfloor 7n/10 \rfloor + 3) + n$, con $T_5(n) = n$ si $n < 80$. T_3, igual con sus fracciones.
- $T_5(n)/n$ crece cada vez menos. $T_3(n)/n$ sube casi lo mismo por cada factor 4: la huella de un logaritmo.
- Una tabla no demuestra nada. La garantía la da la prueba, y la tabla ayuda a detectar errores en ella.

En C++

select.h

```cpp
inline int select_linear(std::vector<int> a, std::size_t k, long long& comparisons) {
    assert(k < a.size());
    if (a.size() <= 5) {
        sort_group(a, comparisons);
        return a[k];
    }
    int pivot = median_of_medians(a, comparisons);
    Parts parts = partition3(a, pivot, comparisons);
    if (k < parts.less.size()) {
        return select_linear(std::move(parts.less), k, comparisons);
    }
    if (k < parts.less.size() + parts.equal.size()) {
        return pivot;
    }
    return select_linear(std::move(parts.greater), k - parts.less.size() - parts.equal.size(),
      comparisons);
}
```

En la práctica

build/select_bench

n	cmp.azar/n	cmp.mdm/n	sort ms	nth ms	azar ms	mdm ms
125000	6.48	8.27	6.8	0.8	1.2	5.3
250000	3.67	8.35	14.5	2.0	2.5	11.1
500000	4.12	8.36	32.7	3.1	5.4	24.4
1000000	2.00	8.38	65.3	7.3	10.8	48.1
2000000	3.31	8.41	141.9	14.9	22.0	91.2
4000000	3.14	8.41	295.5	28.8	39.3	177.6
8000000	3.62	8.43	616.5	63.7	95.7	365.8

Con `-O2`, sin sanitizers, mediana de 7 corridas; los tiempos dependen de la máquina y del momento.

- `cmp.mdm/n` se queda cerca de 8,4: lineal, con constante mayor que 1.
- Aun así pierde: unas 5 a 8 veces más lenta que `nth_element`, y solo le gana a ordenar.

`std::nth_element`

- El estándar pide costo lineal **en promedio**.
- En esta biblioteca, g++ 14: pivote como mediana de tres, y tras $2\lfloor \log_2 n \rfloor$ particiones cambia a un método con heap. Peor caso $O(n \log n)$.
- Un pivote al azar protege de otro modo: ninguna entrada fija es mala de antemano. Su análisis pide Probabilidad.
- ¿Para qué la mediana de medianas? Prueba que $O(n)$ en el peor caso es posible, y es garantía cuando alguien puede construir la entrada mala.

En la práctica

Si necesitan una mediana en C++, usen `std::nth_element`.

SECCIÓN 03

Karatsuba

03

Multiplicar números de n dígitos

La escuela: cada dígito con cada dígito. n^2 productos: $\Theta(n^2)$.

Dividir en dos, con $m = n \operatorname{div} 2$ dígitos en la parte baja:

$$x = a \cdot 10^m + b, \qquad y = c \cdot 10^m + d$$

$$xy = ac \cdot 10^{2m} + (ad + bc) \cdot 10^m + bd$$

Cuatro productos de $n/2$ dígitos:

$$T(n) = 4T(n/2) + O(n) = \Theta(n^{\log_2 4}) = \Theta(n^2)$$

No ganamos nada

$4 \cdot (n/2)^2 = n^2$. Dividir no ayuda por sí solo. Hacen falta **menos subproblemas**.

Tres productos en lugar de cuatro

No necesitamos ad y bc por separado, sino su suma.

$$(a+b)(c+d) = ac + ad + bc + bd$$

$$ad + bc = (a+b)(c+d) - ac - bd$$

Tres productos:

- $p_1 = ac$
- $p_2 = bd$
- $p_3 = (a+b)(c+d)$

$$xy = p_1 10^{2m} + (p_3 - p_1 - p_2)10^m + p_2$$

$$T(n) = 3T(n/2) + O(n) = \Theta(n^{\log_2 3}) \approx \Theta(n^{1,585})$$

El algoritmo

pseudocódigo

```
algoritmo karatsuba(x, y)
    n ← máximo entre los dígitos de x y los de y
    si n ≤ 3 entonces
        devolver multiplicar_escuela(x, y)
    fin si
    m ← n div 2
    a, b ← parte alta y baja de x, con m en la baja
    c, d ← parte alta y baja de y, con m en la baja
    p1 ← karatsuba(a, c)
    p2 ← karatsuba(b, d)
    p3 ← karatsuba(a + b, c + d)
    devolver p1·10^(2m) + (p3 - p1 - p2)·10^m + p2
fin algoritmo
```

¿Por qué $n \leq 3$?

$a + b$ puede tener un dígito más: $99 + 9 = 108$. Hasta $\lceil n/2 \rceil + 1$ dígitos.

Con $n = 3$ eso da 3: no baja. Con $n \geq 4$ sí: $\lceil n/2 \rceil + 1 \leq n - 1$.

1234×5678

$a = 12,\ b = 34,\ c = 56,\ d = 78,\ m = 2$

$p_1 = ac = 672$

$p_2 = bd = 2652$

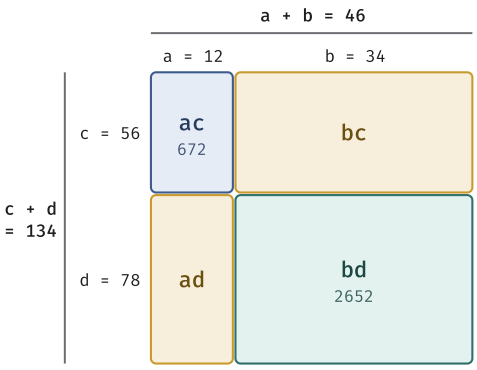
$p_3 = (a + b)(c + d) = 46 \times 134 = 6164$

134 tiene un dígito más

$ad + bc = 6164 - 672 - 2652 = 2840$

$672 \cdot 10^4 + 2840 \cdot 10^2 + 2652 = \mathbf{7\,006\,652}$

Por qué alcanzan tres productos



Los tres productos

$ac = 672$

$bd = 2652$

$(a + b)(c + d) = 46 \times 134 = 6164$

La suma de las dos piezas de oro

$ad + bc = 6164 - 672 - 2652 = 2840$

El resultado

$672 \cdot 10^4$	6 720 000
$2840 \cdot 10^2$	284 000
2652	2 652
	7 006 652

ad y bc no se calculan por separado: solo hace falta su suma.

El costo, con el dígito de más

$$T(n) \leq 3T(\lceil n/2 \rceil + 1) + cn \quad (n \geq 4)$$

- Cambio de variable: $S(n) = T(n+3)$. Como $\lceil (n+3)/2 \rceil + 1 \leq \lceil n/2 \rceil + 3$:

$$S(n) \leq 3S(\lceil n/2 \rceil) + c(n+3)$$

- S ya no tiene el $+1$: $O(n^{\log_2 3})$. Y $T(n) \leq S(n)$.
- En el árbol, el nivel i trabaja $cn(3/2)^i$. **Mandan las hojas:** $3^{\log_2 n} = n^{\log_2 3}$.
- $\log_2 3 = 1{,}58496\ldots$; 1,585 es su redondeo.
- Ahorrar un subproblema cambia el **exponente**. La misma idea da Strassen: 7 productos en vez de 8.

Contar: la teoría predice bien

build/karatsuba_count

n	escuela	razon	karatsuba	razon
16	256	0.00	120	0.00
32	1024	4.00	453	3.77
64	4096	4.00	1313	2.90
128	16384	4.00	3864	2.94
256	65536	4.00	12038	3.12
512	262144	4.00	35686	2.96
1024	1048576	4.00	107748	3.02
2048	4194304	4.00	320861	2.98
4096	16777216	4.00	962380	3.00
8192	67108864	4.00	2895612	3.01
16384	268435456	4.00	8695871	3.00
32768	1073741824	4.00	26057885	3.00
65536	4294967296	4.00	78165329	3.00

Productos de un dígito, números al azar, umbral 3. No dependen de la máquina.

- Al duplicar n, $\times 4$ contra $\times 3$. Y $2^{\log_2 3} = 3$.

Medir: otra pregunta

build/karatsuba_bench

```
microsegundos por multiplicacion
       n      escuela     umbral 3    umbral 128    esc/u3  esc/u128
    1024        528.8       4812.1         369.8      0.11      1.43
    2048       1918.8      14877.1        1162.2      0.13      1.65
    4096       7401.4      52581.2        4370.2      0.14      1.69
    8192      32221.0     145868.5       11165.9      0.22      2.89
   16384     120437.3     381234.3       38112.1      0.32      3.16
   32768     599088.0    1155659.7       92233.5      0.52      6.50
   65536    2330401.8    4419787.0      321476.4      0.53      7.25
  131072    9431325.0   12666355.1      789361.8      0.74     11.95
  262144   58873990.7   57706431.0     4047510.4      1.02     14.55
```

`-O2`, sin sanitizers. Depende de la máquina y de esta implementación. Desde $n = 1024$; el resto, en los apuntes.

- Umbral 3: pierde hasta más de cien mil dígitos. En otra corrida, 1,07 en $n = 131072$.
- Umbral 128: gana desde $n = 2048$. Entre 256 y 1024 las dos corridas no coinciden.

Por qué se cambia a la escuela con n pequeño

n = 16384

umbral	microseg
3	563765.5
8	203841.3
16	144803.1
32	94903.5
64	61352.2
128	59770.8
256	70581.4
512	73600.5
1024	81137.0

- Un producto de la escuela: una instrucción en un ciclo.
- Karatsuba: reservas, copias, sumas, restas.
- De 3 a 128 se ahorra un factor de 9 a 14. Entre 64 y 512, casi no importa cuál.
- El número no se traslada: otra base u otra implementación tendrán otro cruce. Se traslada el fenómeno.

Bibliotecas como GMP combinan la escuela, Karatsuba y métodos mejores para números enormes.

SECCIÓN 04

El par más cercano

04

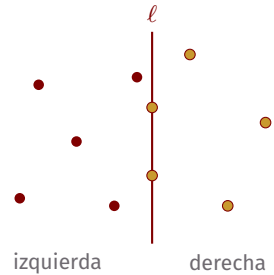
El problema

$n \geq 2$ puntos en el plano. ¿Cuál es la menor distancia entre dos de ellos?

- Los puntos **pueden repetirse**: dos posiciones con el mismo punto están a 0.
- Trabajamos con d^2: compara igual y no hay decimales.
- Fuerza bruta: $\Theta(n^2)$.
- En una dimensión: ordenar y mirar vecinos. En dos no alcanza: ordenar por una coordenada separa puntos cercanos.

Dividir por posición

- Ordenar por *x*, una sola vez.
- Partir por posición: $mid = n$ div 2, ℓ = la *x* del punto en *mid*.
- Izquierda: $x \leq \ell$. Derecha: $x \geq \ell$. **Puede haber puntos sobre la línea en las dos.**
- $\delta = \text{mín}(\delta_I, \delta_D)$. Faltan los pares que cruzan.



Error común

Partir por valor, "$x < \ell$". Con todos sobre una vertical la izquierda queda vacía y la recursión no termina.

La franja

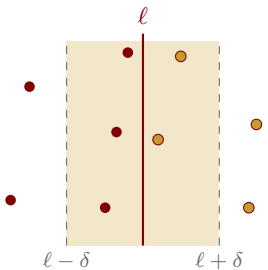
Si p está a la izquierda, q a la derecha y $d(p, q) < \delta$:

$$\ell - p.x \leq q.x - p.x \leq d(p, q) < \delta$$

Igual para q. Los dos están en la franja.

Franja

Los puntos con $|x - \ell| < \delta$. Con cuadrados: $(x - \ell)^2 < \delta^2$.



No basta con los más cercanos a la línea

build/strip_demo, experimento 2

```
linea x = 1
mejor izquierda = 10001, mejor derecha = 10001, delta^2 = 10001
franja por y: (1,-100)D (-2,0)I (2,0)D (-1,100)I
distancia^2 entre los mas cercanos a la linea: 40004
fuerza bruta = 16
```

`I` y `D` marcan la mitad de cada punto.

- Cerca de la línea por cada lado: $(-1, 100)$ y $(1, -100)$, a distancia^2 40004.
- La respuesta, 16, está en $(-2, 0)$ y $(2, 0)$: más lejos de la línea, pero a la misma altura.
- Y toda la entrada cabe en la franja. El truco es el orden por y.

El lema de los 7 siguientes

Lema

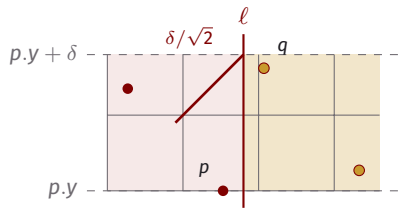
Sea $\delta > 0$. Dos puntos de una misma mitad están siempre a distancia $\geq \delta$. Sea S la franja $|x - \ell| < \delta$, ordenada por y, con cualquier desempate. Si $i < j$ y $d(S[i], S[j]) < \delta$, entonces $j - i \leq 7$.

Técnica: principio del palomar, cubriendo una caja con cuadrados pequeños.

Todo el algoritmo descansa aquí: comparar con 7 vecinos en vez de con todos convierte un paso $\Theta(n^2)$ en $\Theta(n)$.

La demostración

1. Sean $p = S[i]$, $q = S[j]$. Entonces $q.y - p.y < \delta$, y todos los puntos entre ellos tienen $y \in [p.y, p.y + \delta)$.
2. **Por mitad, no por geometría.** Un punto de la izquierda cae en $[\ell - \delta, \ell] \times [p.y, p.y + \delta]$. Uno de la derecha, en $[\ell, \ell + \delta] \times$ lo mismo.
3. Cuatro celdas de lado $\delta/2$ cubren cada cuadrado. Su diagonal es $\delta/\sqrt{2} < \delta$.
4. En una celda cabe a lo sumo un punto de **su** mitad: dos de ellos estarían a menos de δ.
5. A lo sumo $4 + 4 = 8$ puntos: $j - i + 1 \leq 8$. ■



Puntos sobre la línea y repetidos

- **Sobre la línea:** nada especial, porque se clasifica por mitad. Todos sobre una vertical: todos en la franja si $\delta > 0$, y el lema sigue valiendo.
- **Repetidos dentro de una mitad:** esa mitad da $\delta = 0$. La franja queda vacía, y el 0 ya está listo. No se usa el lema.
- **Repetidos, uno en cada mitad:** $\delta > 0$. Los dos tienen $x = \ell$, están en la franja, a distancia $0 < \delta$. El lema los encuentra.
- **Franja estricta:** con $|x - \ell| = \delta$ un punto queda afuera sin riesgo. Solo buscamos pares estrictamente mejores que δ.
- **Sobre el 7:** alcanza, y eso es lo que se necesita. No afirmamos que sea el menor posible, y otros libros usan otra constante según cómo cubran la caja.

Pocos vecinos no alcanza

build/strip_demo, experimentos 3 y 4

```
linea x = 0
mejor izquierda = 2564, mejor derecha = 65, delta^2 = 65
franja por y: (-1,0)I (8,1)D (0,2)D
solo con el siguiente: 65
con los 7 siguientes: 5
fuerza bruta = 5

linea x = 1
mejor izquierda = 12301, mejor derecha = 11908, delta^2 = 11908
franja por y: (0,0)I (-99,50)I (99,51)D (1,99)D
con los 1 siguientes: 11908
con los 2 siguientes: 11908
con los 3 siguientes: 9802
fuerza bruta = 9802
```

- Con 1 vecino falla: un punto lejano se mete entre los dos del par.
- Con 2 también.
- Con 7 está demostrado.

El algoritmo

pseudocódigo

```
algoritmo resolver(P, lo, hi)
    si hi - lo ≤ 3 entonces
        d ← el menor d²(P[i], P[j]) del rango
        ordenar P[lo .. hi - 1] por y
        devolver d
    fin si
    mid ← (lo + hi) div 2
    linea ← P[mid].x
    d ← mín(resolver(P, lo, mid), resolver(P, mid, hi))
    mezclar por y, dejando P[lo .. hi - 1] ordenado por y
    S ← los puntos con (x - linea)² < d, en ese orden
    para i desde 0 hasta |S| - 1 hacer
        para j desde i + 1 hasta mín(i + 7, |S| - 1) hacer
            d ← mín(d, d²(S[i], S[j]))
        fin para
    fin para
    devolver d
fin algoritmo
```

- Ordenar por *x* una vez, antes de llamar.
- `linea` antes de las llamadas: ellas reordenan el rango.
- Cada llamada deja su rango ordenado por *y*.

Correctitud

Teorema. `resolver` devuelve la menor distancia^2 del rango, y lo deja ordenado por y.
Técnica: inducción fuerte sobre el tamaño.

- **Base:** con a lo sumo 3 puntos se miran todos los pares.
- **Paso:** sea D la verdadera respuesta. Todo lo que el algoritmo mira es la distancia de un par real, así que nunca baja de D^2.
- Si $D = \delta$: D^2 ya es el valor inicial de d.
- Si $D < \delta$: el par no está dentro de una mitad, porque ahí todos están a $\geq \delta$. Entonces cruza, y está en la franja, con $\delta > 0$.
- Por el lema, a lo sumo a 7 posiciones. El ciclo lo compara. ■

Costo

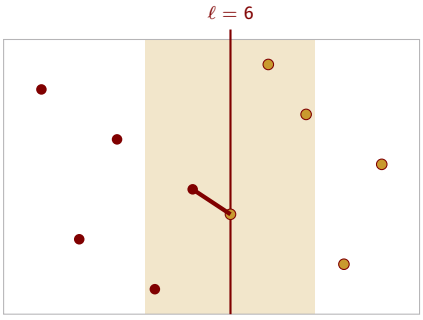
- Ordenar por x, una vez: $\Theta(n \log n)$.
- Mezcla, construir S, ciclo con a lo sumo 7 vecinos: $\Theta(n)$ por llamada.

$$T(n) = 2T(n/2) + \Theta(n) = \Theta(n \log n)$$

Error común

Ordenar la franja en cada nivel. Con la franja llena, $T(n) = 2T(n/2) + \Theta(n \log n) = \Theta(n \log^2 n)$. Hay que **mezclar**, y por eso cada llamada devuelve su rango ordenado.

Siguiendo un nivel



- $mid = 5$, $\ell = 6$. Izquierda: mejor $d^2 = 8$. Derecha: mejor $d^2 = 5$. $\delta^2 = 5$.
- Franja $(x - 6)^2 < 5$, por y: $(4, 1), (6, 4), (5, 5), (8, 8), (7, 10)$.
- $(6, 4)$ y $(5, 5)$: $d^2 = 2 < 5$. **Respuesta 2**, un par que cruza.

Los casos raros, antes de programar

Entrada	Qué pasa
Dos puntos	Caso base. Es la respuesta
Todos iguales	$\delta = 0$. La franja queda vacía. Respuesta 0
Par repetido, uno en cada mitad	$\delta > 0$. Los dos sobre la línea, en la franja, a 0
Todos sobre una vertical	Todos en la franja. Por posición, mitades de $n/2$
Todos sobre una horizontal	Las y empatan. Cualquier desempate sirve
Dos columnas pegadas	Franja llena en cada nivel. Los 7 vecinos mantienen el costo

Antes de declarar un tipo

Calculen el mayor valor posible de cada resta, cada producto y cada suma con los límites del problema.

Laboratorio: el par más cercano

1. Leer n puntos de coordenadas enteras e imprimir el cuadrado de la menor distancia.
2. Escribir la fuerza bruta, y usarla para generar pruebas al azar contra la versión rápida.
3. Pasar todos los casos, incluidos puntos repetidos y todos sobre una misma vertical.
4. Calcular si el cuadrado de la distancia cabe en `int` con los límites del enunciado.

> **Dónde**
>
> `code/week-03/lab/closest_pair`. `make test` corre el juez.

Para la próxima clase

- Leer los apuntes de la semana 3 y terminar el laboratorio.
- Preguntas de repaso de la guía: la recurrencia con 10n, 7 productos con trabajo extra $\Theta(n^2)$, y la mediana de medianas a mano.
- Próxima semana: una cota inferior para ordenar, y los primeros greedy.

Pregunta

Con grupos de 5, ¿qué línea del lema falla si partimos solo en menores y mayores o iguales? ¿Y por qué el algoritmo sigue siendo correcto aunque deje de ser lineal?