

## División y Conquista: Presentación

Teoría de Algoritmos I (75.29 / 95.06)

Ing. Víctor Daniel Podberezski

✉ [vpodberezski@fi.uba.ar](mailto:vpodberezski@fi.uba.ar)

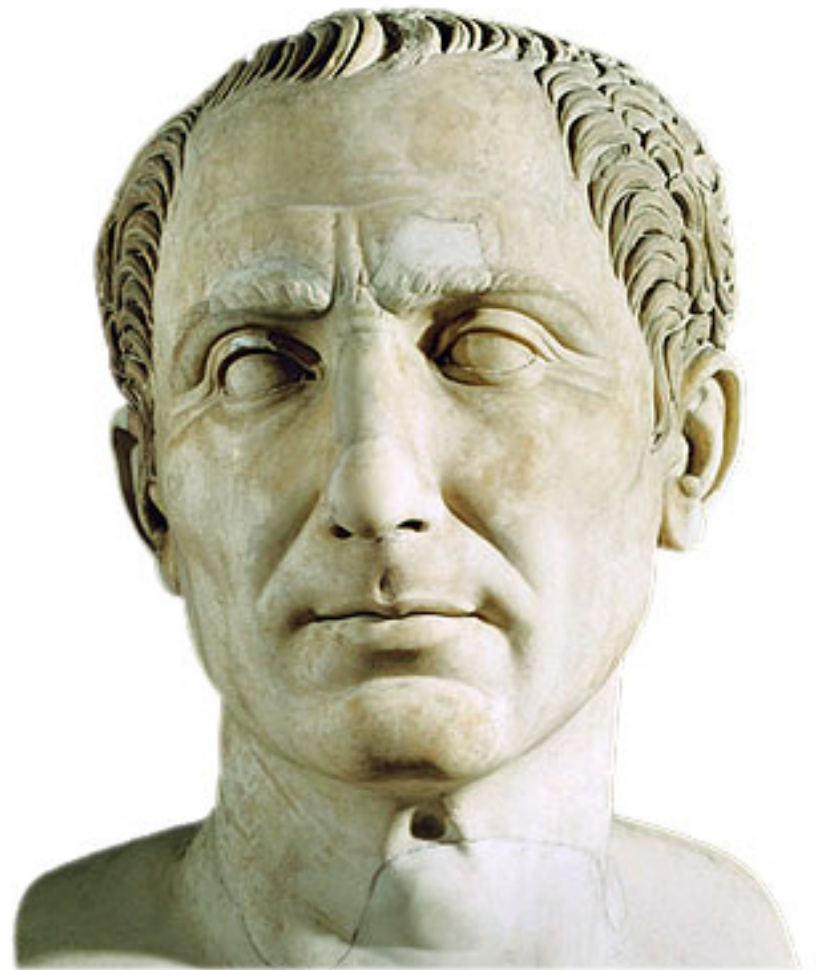
# Dīvide et īmpera

## Máxima política y militar

atribuida al emperador romano Julio Cesar.

## Utilizado en algoritmos matemáticos

desde hace siglos (Euclides, Gauss, ...)



# División y conquista

## Conjunto de técnicas algorítmicas

en el que se divide el problema en subproblemas de igual naturaleza y menor tamaño  
se los conquista (resuelve) en forma recursiva (hasta un caso base)  
y se combina los resultados en una solución general

## Generalmente se pueden aplicar a problemas

donde la solución por fuerza bruta ya tiene una complejidad polinómica

**Analizar su complejidad requiere resolver una relación de recurrencia**

# Relación de recurrencia

## Ecuación que define una secuencia recursiva

Cada término de la secuencia es definido como una función de términos anteriores

$$T_n = F(T_{n-1}, T_{n-2}, \dots)$$

Existe uno o varios términos base o iniciales desde los cuales se calculan los siguientes.

# Plantilla básica

1. Dividir el problema en “q” subproblemas de tamaño reducido al original.
2. Resolver cada subproblema por separado mediante recursión
3. Combinar el resultado de los subproblemas.

# Ejemplo: MergeSort

## Algoritmo de ordenamiento

utiliza el divide y vencerás

## Propuesto por John von Neumann en 1945

(Según Knuth en “Art of computing Programming”)

### MERGESORT(A)

Si  $\text{size } A == 2 \rightarrow$  comparar y devolver ordenado

$A1 = (\text{size } A)/2$  primeros elementos de A

$A2 = (\text{size } A)/2$  últimos elementos de A

Retornar MERGE (MERGESORT(A1), MERGESORT(A2))

# MergeSort – Análisis de Complejidad (cont.)

**Se debe resolver la relación de recurrencia**  
para poder calcular la complejidad.

## **3 formas básicas de resolverlas:**

“Desenrollarla”

“Adivinar” y verificar: Inducción

Método del maestro

# MergeSort – Análisis de Complejidad

Sea  $T(n)$  el peor caso de tiempo de ejecución

para la resolución del problema de “n” elementos.

↙  $f(N) = O(1) \leq c$  (para un  $c > 0$ )

Sea DIV el proceso de dividir el problema en 2 subproblemas.

Sea UNI el proceso de unir el resultado de de los 2 subproblemas

↙  $MERGE \rightarrow f(N) = O(N) \leq c n$  (para un  $c > 0$ )

$$T(n) \leq 2 * T(n/2) + DIV + UNI \quad \text{y} \quad T(2) \leq c$$

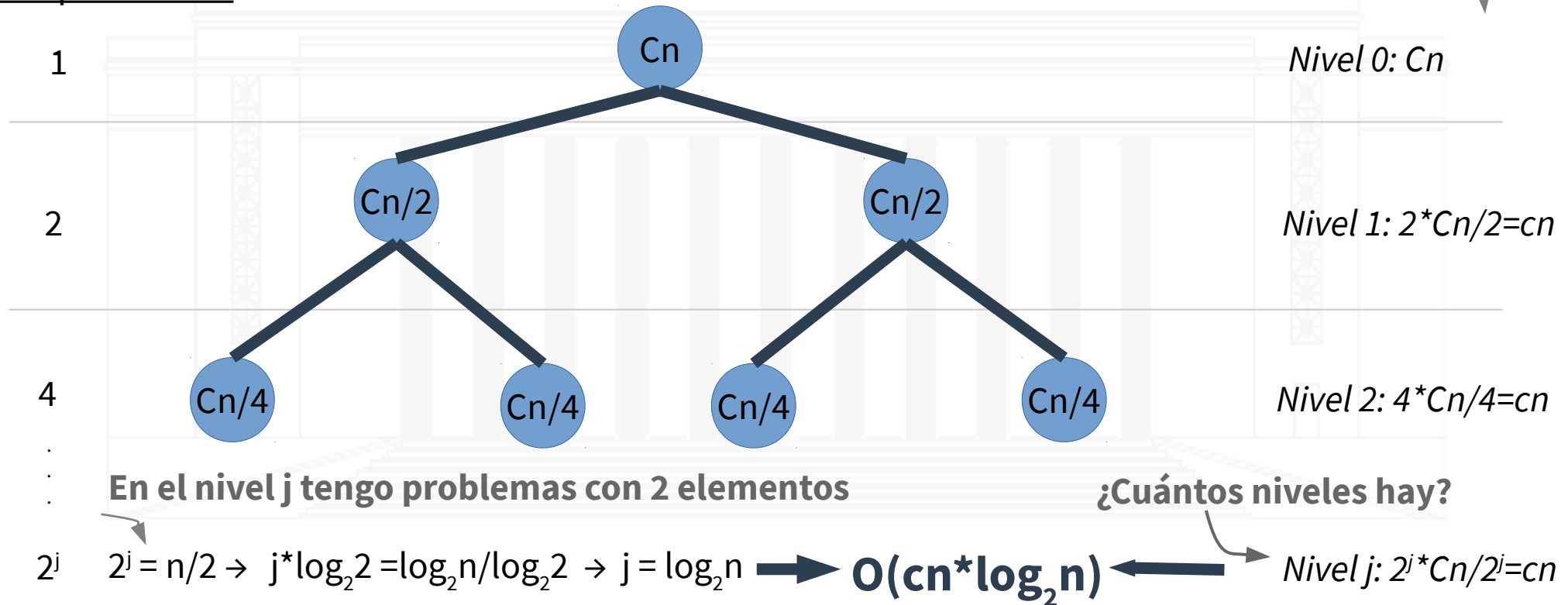
$$\begin{cases} T(n) \leq 2 * T(n/2) + cn & n > 2 \\ T(2) \leq d & n = 2 \end{cases}$$



# Desenrollando la recurrencia

## Analizar los primeros niveles de la misma

Subproblemas:



# Validación del resultado

Probar en la recurrencia la validez del resultado

$$T(2) \leq c$$

$$T(n) \leq 2 * T(n/2) + cn$$

$$T(n=2) = cn \log_2 n = 2c$$

$$T(2) \leq d < 2c$$

$$\begin{aligned} T(n) &\leq 2c(n/2) \log_2(n/2) + cn \\ &= cn[(\log_2 n) - 1] + cn \\ &= (cn \log_2 n) - cn + cn \\ &= cn \log_2 n. \end{aligned}$$

# Desenrollando - Matemáticamente

## Relación de recurrencia

$$T(n) = 2T(n/2) + cn$$

$$T(2) = d$$

$$T(n/2) = 2T(n/4) + n/2$$

$$T(n/4) = 2T(n/8) + n/4$$

...

$$T(n) = 2 [ 2 ( 2 \{ T(n/16) + n/8 \} + n/4 ) + n/2 ] + n$$

$$T(n) = 2^k * 1 + \sum_{i=0}^k \frac{2^i * n}{2^i}$$

# Desenrollando – Matemáticamente (cont.)

## Continuemos

vemos que  $k = \log n$

$$T(n) = 2^k * 1 + \sum_{i=0}^k \frac{2^i * n}{2^i}$$

$$T(n) = 2^{\log(n)} + n * \sum_{i=0}^{\log(n)} \left(\frac{2}{2}\right)^i$$

$$T(n) = 2^{\log(n)} + n * \sum_{i=0}^{\log(n)} 1^i$$

$$T(n) = 2^{\log(n)} + n \log(n)$$

$$T(n) = n^{\log(2)} + n \log(n) \quad \longrightarrow \quad T(n) = O(n \log n)$$

$$a^{\log_2 n} = n^{\log_2 a}$$

# “Adivinar” y verificar

Podemos calcular  $T(n)$  probando...

Ej:  $T(n) = O(n)$ ,  $O(n^2)$ ,  $O(n \log n)$

$$T(n) \leq 2 * T(n/2) + cn \quad n > 2$$

$$O(n) : kn \leq 2 * kn/2 + cn \leq (k+c)n$$

$$O(n^2) : kn^2 \leq 2 * kn^2/2 + cn \leq kn^2 + cn$$

$$O(n \log n) : kn \log_2 n \leq 2k(n/2) \log_2 (n/2) + cn =$$

$$= kn * (\log_2 n - 1) + cn = kn \log_2 n - kn + cn$$

Mejor!

$$kn \log_2 n \leq kn \log_2 n + (c-k)n$$

sii  $c=0$  **X**  
para todo  $c \geq 0$  **✓**

sii  $k \leq c$  **✓**

# Corolario

Cualquier función que satisfaga:

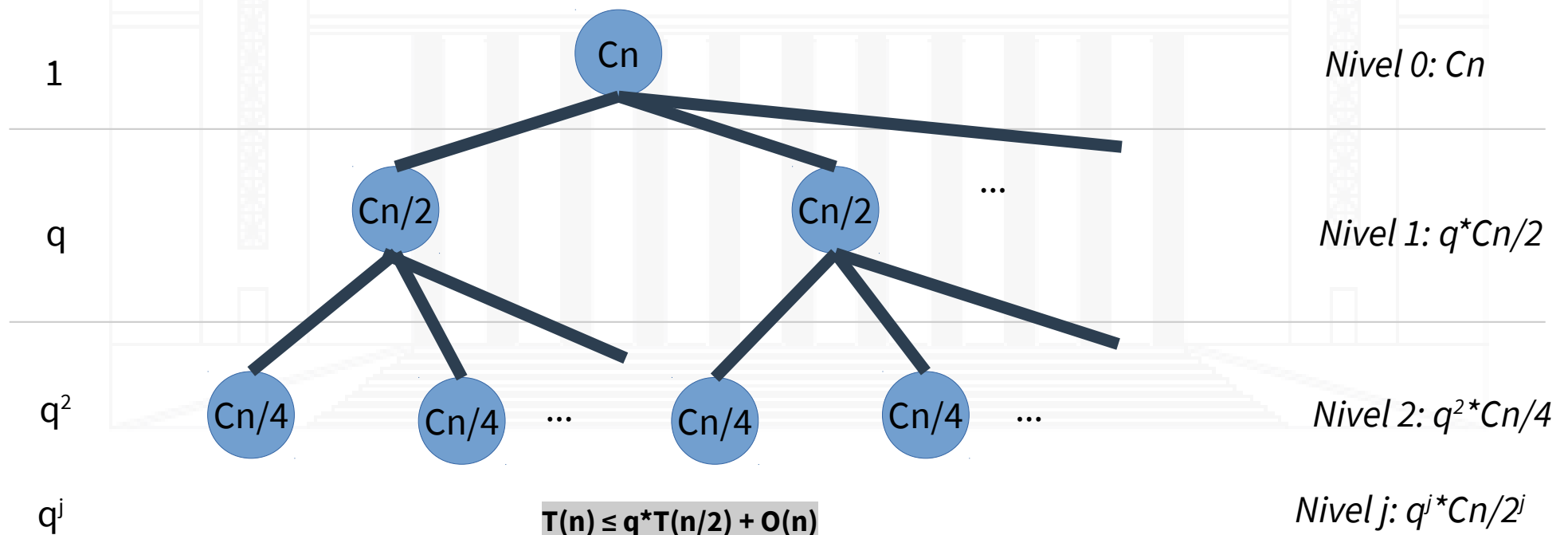
$$\begin{aligned}T(n) &\leq 2 * T(n/2) + cn \quad n > 2 \\T(2) &\leq c\end{aligned}$$

cumple con  $O(n \log n)$  cuando  $n > 1$

# Subdivisión en mas de 2 partes

Que pasa si en vez de dividir el problema en 2 subproblemas se hace en “q”?

Si  $q > 2$



# Análisis de complejidad

Podemos explorar la recurrencia como:

$r = q/2 > 1 \rightarrow$  podemos usar la suma geométrica

$$\sum_{i=0}^x r^i = \frac{1-r^{(x+1)}}{1-r}$$

$$T(n) \leq \sum_{j=0}^{\log_2 n - 1} \left(\frac{q}{2}\right)^j cn$$

$$T(n) \leq cn \frac{(1-r^{\log_2 n})}{1-r} \leq cn \frac{r^{\log_2 n}}{r-1}$$

$$n^{\log_2 r}$$

$$\text{Si } q=3 \rightarrow O(n^{1.59})$$

$$\text{Si } q=4 \rightarrow O(n^2)$$

$$T(n) \leq \frac{c}{\frac{q}{2}-1} n n^{\log_2 \frac{q}{2}} = \frac{c}{\frac{q}{2}-1} n^{1+\log_2 q-1}$$

$$\rightarrow O(n^{\log_2 q})$$

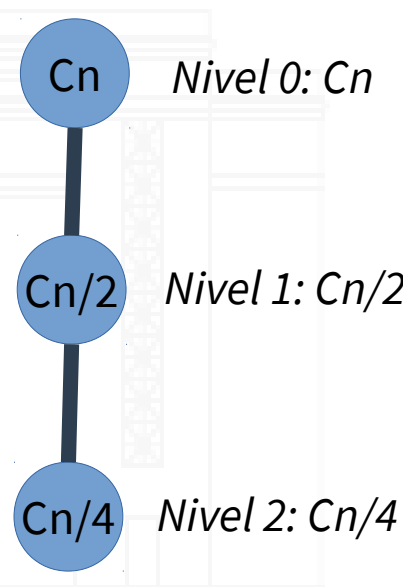


# Subdivisión en solo 1 problema

Si  $q=1$

$$T(n) \leq \sum_{j=0}^{\log_2 n - 1} \frac{cn}{2^j}$$

podemos usar la suma geométrica

$$T(n) \leq 2cn \quad \longrightarrow \quad O(n)$$


$$T(n) \leq q \cdot T(n/2) + O(n)$$

# El método del maestro

**Presentado por Jon Bentley, Dorothea Haken y James B. Saxe**

en el paper “A General Method for Solving Divide-and-Conquer Recurrences ”  
en 1980.

**Nombrado originalmente "unifying method"**

Popularizado como "Master Theorem" por Cormen, Leiserson, Rivest, and Stein



Presentación realizada en Septiembre de 2020

## División y conquista: Teorema maestro - Ejemplos

Teoría de Algoritmos I (75.29 / 95.06)

Ing. Víctor Daniel Podberezski

✉ [vpodberezski@fi.uba.ar](mailto:vpodberezski@fi.uba.ar)

# Teorema maestro

## Sean

$a \geq 1$  y  $b \geq 1$  constantes,

$f(n)$  una función,

$T(n) = aT(n/b) + f(n)$  una recurrencia con  $T(0) = \text{cte}$

## Entonces

- 1) Si  $f(n) = O(n^{\log_b a - e})$ ,  $e > 0 \Rightarrow T(n) = \Theta(n^{\log_b a})$
- 2) Si  $f(n) = \Theta(n^{\log_b a}) \Rightarrow T(n) = \Theta(n^{\log_b a} * \log n)$
- 3) Si  $f(n) = \Omega(n^{\log_b a + e})$ ,  $e > 0 \Rightarrow T(n) = \Theta(f(n))$

$\forall af(n/b) \leq cf(n)$ ,  $c < 1$  y  $n \gg$

# Ejemplo 1

$$T(n) = 9 T(n/3) + n$$

$$a = 9 \quad b = 3 \quad f(n) = n$$

## Probamos

$$\text{Caso 2: } f(n) = \Theta(n^{\log_b a}) \Rightarrow T(n) = \Theta(n^{\log_b a} * \log n)$$

$$n \stackrel{?}{=} \Theta(n^{\log_3 9}) \stackrel{?}{=} \Theta(n^2)$$



$$\text{Caso 1: } f(n) = O(n^{\log_b a - e}), e > 0 \Rightarrow T(n) = \Theta(n^{\log_b a})$$

$$n \stackrel{?}{=} O(n^{\log_3 9 - e}) \stackrel{?}{=} O(n^{2-e}) \quad \text{Si } e = 1 \rightarrow n = O(n^{2-1}) = O(n)$$

$$\Rightarrow T(n) = \Theta(n^2)$$

# Ejemplo 2

$$T(n) = T(2n/3) + 1$$

$$a = 1 \quad b = 3/2 \quad f(n) = 1$$

## Probamos

$$\text{Caso 2: } f(n) = \Theta(n^{\log_b a}) \Rightarrow T(n) = \Theta(n^{\log_b a} * \log n)$$

$$1 \stackrel{?}{=} \Theta(n^{\log_{3/2} 1}) \stackrel{?}{=} \Theta(n^0)$$

$$\Rightarrow T(n) = \Theta(n^{\log_{3/2} 1} * \log n)$$

$$\Rightarrow T(n) = \Theta(\log n)$$

# Ejemplo 3

$$T(n) = 3 T(n/4) + n \log n$$

$$a=3 \quad b=4 \quad f(n) = n \log n$$

## Probamos

$$\text{Caso 2: } f(n) = \Theta(n^{\log_b a}) \Rightarrow T(n) = \Theta(n^{\log_b a} * \log n)$$

$$n \log n \stackrel{?}{=} \Theta(n^{\log_4 3}) \stackrel{?}{=} \Theta(n^{0,793}) \quad \mathbf{X}$$

$$\text{Caso 1: } f(n) = O(n^{\log_b a - e}), e > 0 \Rightarrow T(n) = \Theta(n^{\log_b a})$$

$$n \log n \stackrel{?}{=} O(n^{0,793 - e}) \stackrel{?}{=} O(n^{2 - e}) \quad \mathbf{X}$$



## Ejemplo 3 (cont.)

$$T(n) = 3 T(n/4) + n \log n$$

$$a = 3 \quad b = 4 \quad f(n) = n \log n$$

### Probamos

$$\text{Caso 3: } f(n) = \Omega(n^{\log_b a + e}), e > 0 \Rightarrow T(n) = \Theta(f(n))$$

$$n \log n \stackrel{?}{=} \Omega(n^{0,793+e}) \quad \text{Si } e = 0,1 \rightarrow n \log n = \Omega(n^{0,893})$$

$$\exists c < 1, n \gg 1 / a * f(n/b) \leq c * f(n)$$

$$3(n/4 * \log(n/4)) \leq c * n * \log n$$

$$\text{Si } c = 3/4 \Rightarrow 3/4 n * \log(n/4) \leq 3/4 n * \log n$$

$$\Rightarrow T(n) = \Theta(n \log n)$$

# Ejemplo 4

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

$$a = 2 \quad b = 2 \quad f(n) = n \log n$$

## Probamos

Caso 2:  $f(n) = \Theta(n^{\log_b a})$

$$n \log n = \Theta(n)$$



Caso 1:  $f(n) = O(n^{\log_b a - e})$ ,  $e > 0$

$$n \log n = O(n^{1-e})$$



Caso 3:  $f(n) = \Omega(n^{\log_b a + e})$ ,  $e > 0$

$$n \log n = \Omega(n^{1+e})$$



No se puede!



Presentación realizada en Abril de 2020

## División y conquista: Contando inversiones

Teoría de Algoritmos I (75.29 / 95.06)

Ing. Víctor Daniel Podberezski

✉ [vpodberezski@fi.uba.ar](mailto:vpodberezski@fi.uba.ar)

# Problema

## Sean

Un conjunto de  $n$  “elementos”

Dos listas ordenadas de los “ $n$ ” elementos

## Queremos

Tener una medida de semejanza / diferencia entre las dos listas.

# Diferencias entre las preferencias

## Llamaremos

A y B a las listas de preferencias.

## Utilizaremos

El orden de aparición de los elementos de A para identificar a los diferencias.

$A=1,2,\dots,n$

## Por otro lado

Tendremos la lista de B que posiblemente esté “fuera de orden”

Ej:  $B=2,1,4,7,n,\dots$

(Si B esta en orden, entonces representa el mismo orden de preferencia que A)

# Mensurar las diferencias

## Buscaremos

Poder mensurar que nos diga que tan lejos esta B de estar ordenada en forma ascendente.

## Si

$b_i < b_{i+1}$  para todo  $i$ , entonces A y B son iguales

Queremos que el valor de “diferencia” sea igual a cero

## A medida

Que B esté “más mezclado” el valor de diferencia debe aumentar

# Inversiones

## Utilizaremos

El concepto de “inversiones” para medir cuando desordenado esta la lista B

**Dos elementos  $b_i, b_j$  con  $i < j$  están invertidos**

Si  $b_i > b_j$

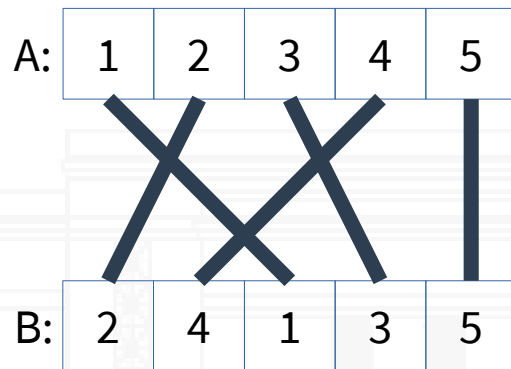
**Podemos – mediante fuerza bruta – calcular la cantidad de inversiones**

Comparando cada posición con todas las siguientes  $\rightarrow O(n^2)$

¿Podemos hacerlo mejor?



# Ejemplo

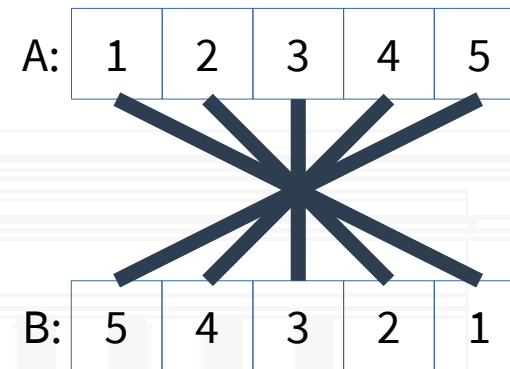


## Podemos ver que

1 está “invertido” con 2 elementos: 2 y 4  $\rightarrow$  (2,1) (4,1)

4 está “invertido” con 1 elemento: 3  $\rightarrow$  (4,3)

**En total hay 3 inversiones**



1 está invertido con 4 elementos

2 está invertido con 3 elementos

3 está invertido con 2 elementos

4 está invertido con 1 elementos

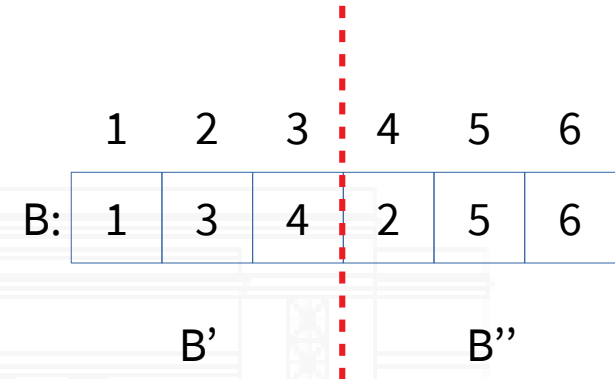
**En total hay 10 inversiones**

$$\binom{n}{2}$$

# Idea de la solución

## Si la lista de preferencias esta semi-ordenada en mitades

Con ausencia de inversiones dentro de las mitades de la lista

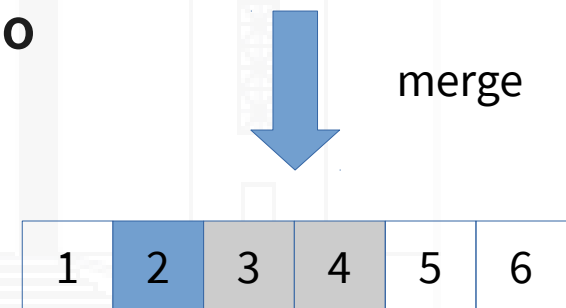


## Las inversiones se encuentran

entre elementos de la primera mitad con elementos de la segunda.

## Podemos contar las inversiones realizando un proceso de merge

Contando - cada vez que se inserta un elemento de la segunda mitad - cuantos elementos de la primera mitad quedan por insertarse



Este proceso se puede realizar en  $O(n)$

# Idea de la solución (cont.)

## El resultado del proceso de merge y conteo

Sera una lista ordenada

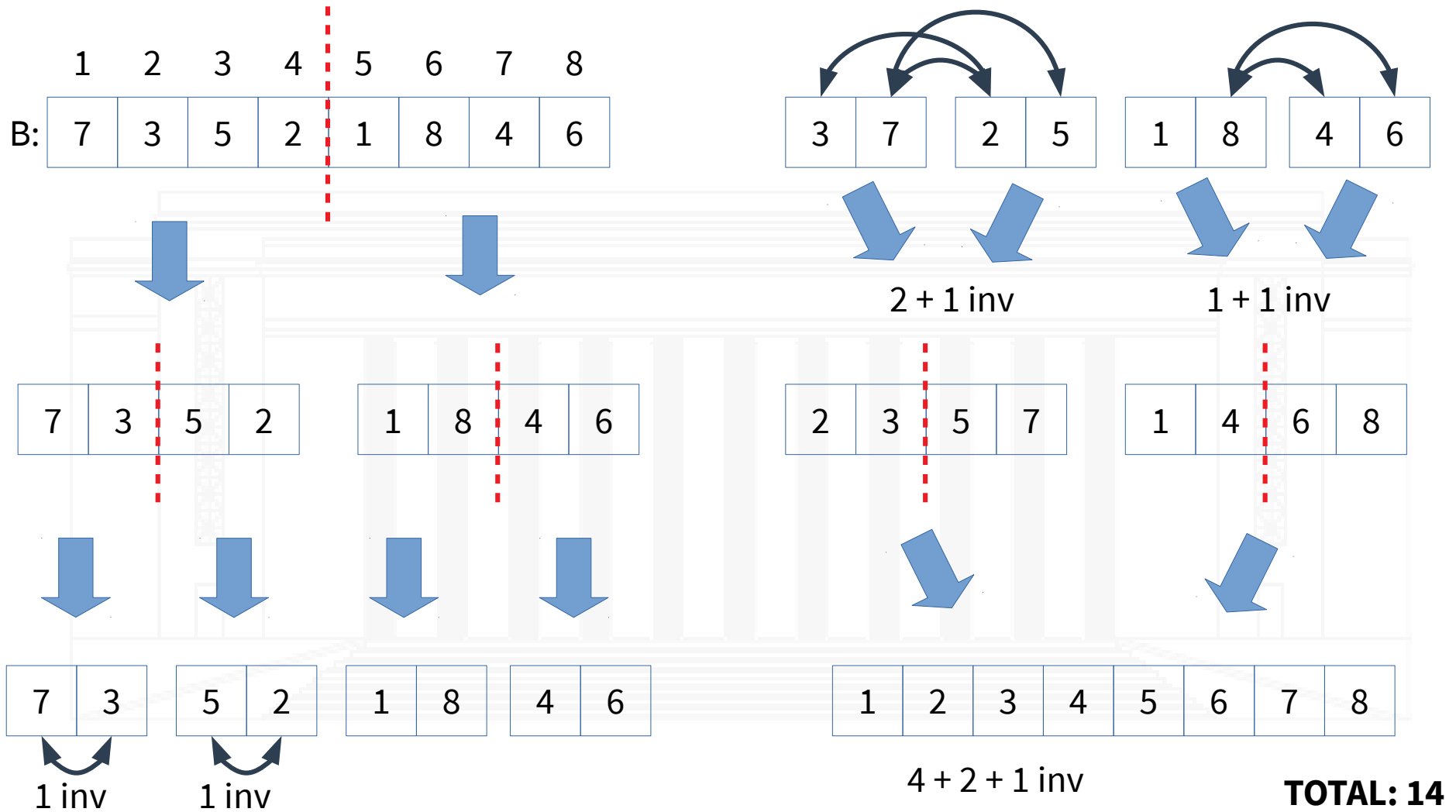
## Lamentablemente no podemos suponer

Que las mitades estén ordeandas

## Pero, podemos partir estas partes recursivamente para hacerlo.

La suma de todas las inversiones contadas en cada uno de los subproblemas corresponden a la cantidad total de inversiones en la lista

# Ejemplo



# Pseudocódigo

```
Ordenar-Contar(L)
  Si  $|L|=1$ 
    Retornar (0,L) // No hay inversiones

  Sino
    Sea A los techo( $n/2$ ) primeros elementos de L
    Sea B los piso( $n/2$ ) restantes elementos de L

    (ra,A) = Ordenar-Contar(A)
    (rb,B) = Ordenar-Contar(B)

    (r,L) = Merge-Contar(A,B)

  Retornar (r+ra+rb,L)
```

# Pseudocódigo (cont.)

```
Merge-Contar(A,B)
  Sea L lista
  inv =0
  j=0, i=0          //punteros a la lista A y B
  Repetir
    a = A[i], b = B[j]
    Si a>b
      L[i+j]=a , i++
    Sino
      L[i+j]=b , j++
      inv+= (|A| - i)
  Mientras i<|A| y j<|B|
  Desde i hasta |A|-1
    L[i+j]=A[i]
    inv+=|B|
  Desde j hasta |B|-1
    L[i+j]=B[i]
  Retornar (inv,L)
```

# Complejidad

## Cada problema con $n$ elementos

Se divide en 2 subproblemas de  $n/2$  elementos

## La unión de los resultados

Se construye recorriendo una vez los  $n$  elementos  $\rightarrow O(n)$

## Podemos expresar la recurrencia como

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

## Utilizando el teorema maestro

Nos queda una complejidad temporal  $T(n) = O(n \log n)$



Presentación realizada en Septiembre de 2020



# División y Conquista: Puntos más cercanos en el plano

Teoría de Algoritmos I (75.29 / 95.06)

Ing. Víctor Daniel Podberezski

✉ [vpodberezski@fi.uba.ar](mailto:vpodberezski@fi.uba.ar)

# Problema

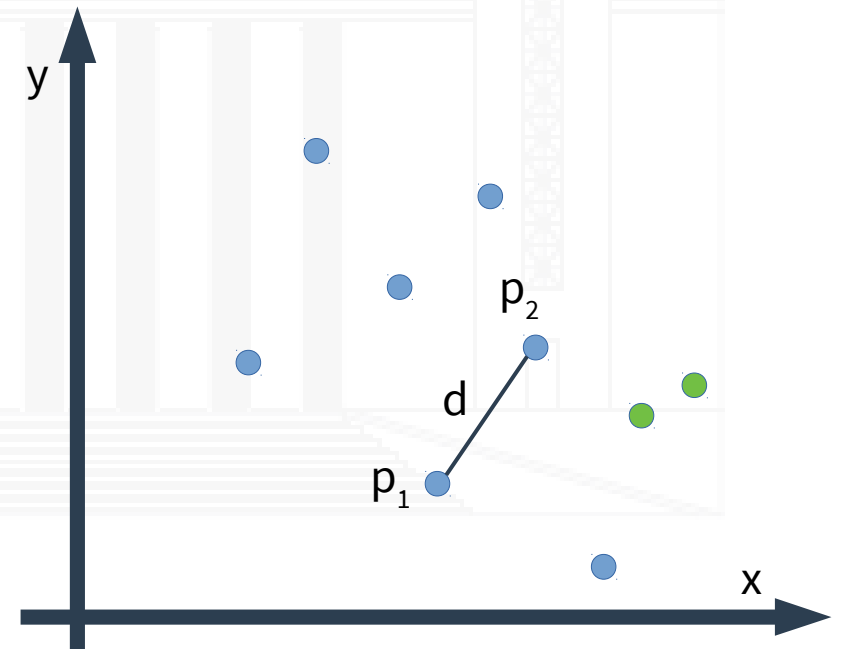
## Sea

$P$  un conjunto de “ $n$ ” puntos en el plano

$d(p_1, p_2)$  la función distancia entre  $p_1=(x_1, y_1)$ ,  $p_2=(x_2, y_2) \in P$

## Queremos

Encontrar los puntos más cercanos en  $P$



# Solución por fuerza bruta

## Para buscar los puntos más cercanos

Tengo que calcular las distancia entre cada punto y el resto de ellos

Quedarme con aquellos 2 de menor distancia

## Teniendo n puntos

Para el primer punto calculo n-1 distancias

Para el segundo calculo n-2 distancias

...

Para el punto n calculo 0 distancias

## Realizo en total $n*(n-1)/2$

Complejidad  $O(n^2)$

¿Lo puedo hacer mejor?

# Simplificación: puntos cercanos en una recta

## Si reducimos

el problema a solo 1 dimensión

## Podemos ordenar los puntos

De mayor a menor  $\rightarrow O(n \log n)$



## Luego por cada punto (iniciando por el menor)

Calcular la distancia al punto siguiente  $\rightarrow O(n)$

Recordar los puntos si es la distancia menor encontrara hasta el momento

## ¿Podemos hacer algo similar con los puntos en el plano?

(no directamente... pero utilizaremos el ordenamiento de los puntos como parte de la solución)

# Solución utilizando División y conquista

**El algoritmo utilizando división y conquista fue propuesto por**

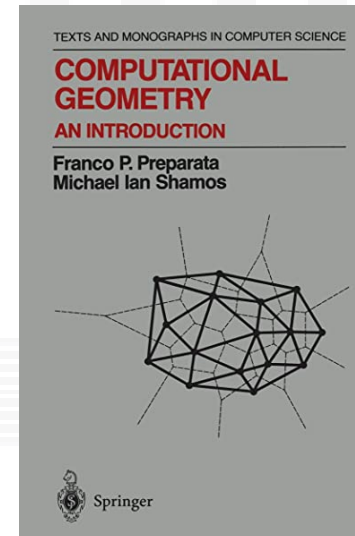
Michael. I. Shamos en 1975

**Lo explica en detalle en su libro de 1985**

“Computational Geometry. An introduction”

**Con su coautor**

Preparata, Franco P



Michael I. Shamos

# Preliminares

## Asumiremos que

No existe  $p_1=(x_1,y_1), p_2=(x_2,y_2) \in P / x_1 = x_2$  o  $y_1 = y_2$

(Este requerimiento se puede eliminar modificando ligeramente el algoritmo)

## Crearemos

$P_x$ : almacenando los puntos ordenados de acuerdo a la coordenada  $x$

$P_y$ : almacenando los puntos ordenados de acuerdo a la coordenada  $y$

## Para cada punto $p \in P$

almacenaremos en que posición aparece en  $P_x$  y  $P_y$

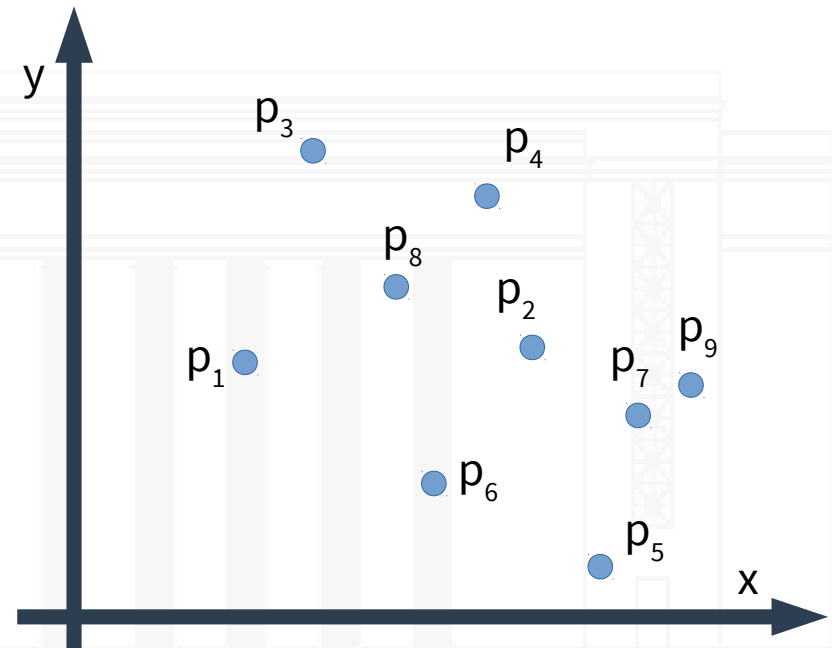
# Ejemplo

## En el ejemplo

$P: p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_9$

$P_x: p_1, p_3, p_8, p_6, p_4, p_2, p_5, p_7, p_9$

$P_y: p_5, p_6, p_7, p_9, p_1, p_2, p_8, p_4, p_3$



# División: Sub problemas

## Llamaremos

Q al set de puntos que se encuentra en las primeras  $\lceil n/2 \rceil$  posiciones de  $P_x$

R al set de puntos que se encuentra en las ultimas  $\lfloor n/2 \rfloor$  posiciones de  $P_x$

## Para

Q, calcularemos  $Q_x$  y  $Q_y$

R, calcularemos  $R_x$  y  $R_y$

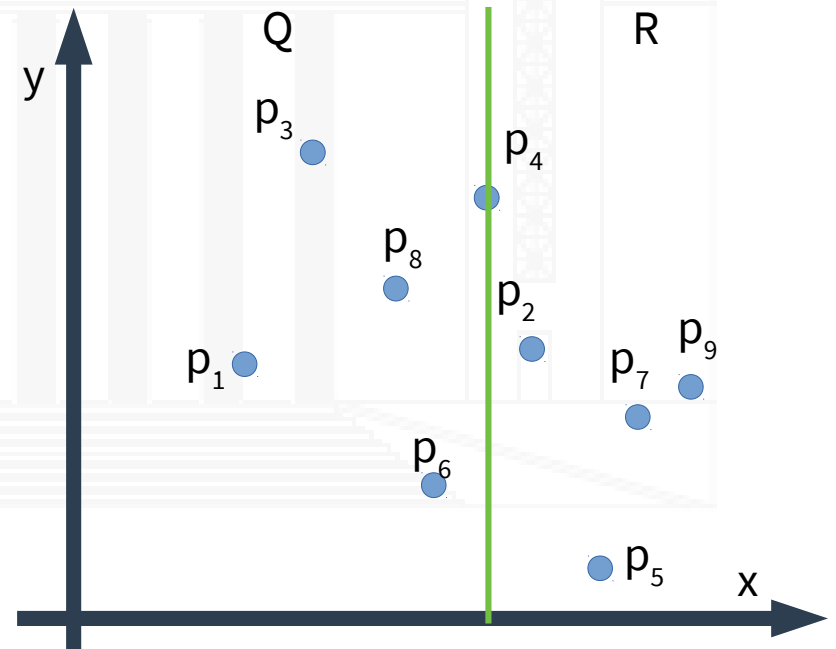
## cada punto $q \in Q$

almacenaremos en que posición aparece en  $Q_x$  y  $Q_y$

## cada punto $r \in R$

almacenaremos en que posición aparece en  $R_x$  y  $R_y$

(podemos hacerlo en  $O(n)$  utilizando  $P$ ,  $P_x$  y  $P_y$ )





# Conquista: Sub problemas

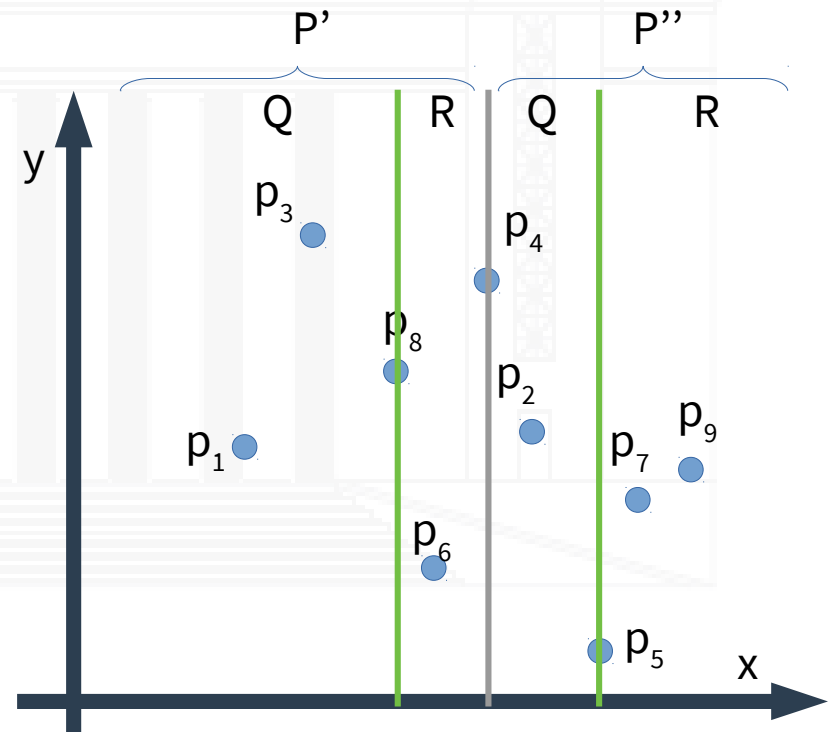
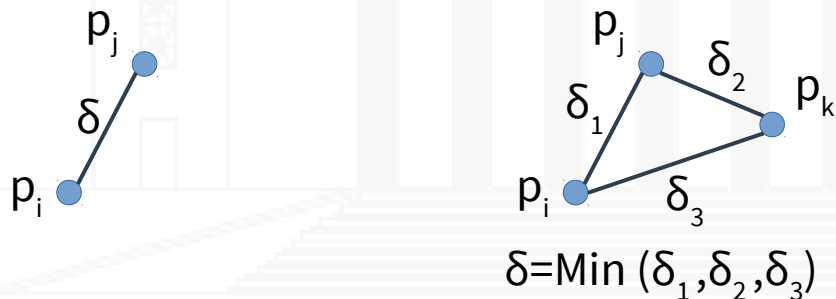
## Recursivamente

Resolveremos el problema de encontrar el par de puntos mas cercanos

Volviendo a dividir el subproblema en 2

## El problema base

Corresponde a un set de 2 o 3 puntos



# Combinación: Sub problemas

## Hasta el momento

Dividimos cada problema en 2 subproblemas de mitad de tamaño

Realizamos la división en  $O(n)$

## Para juntar los problemas

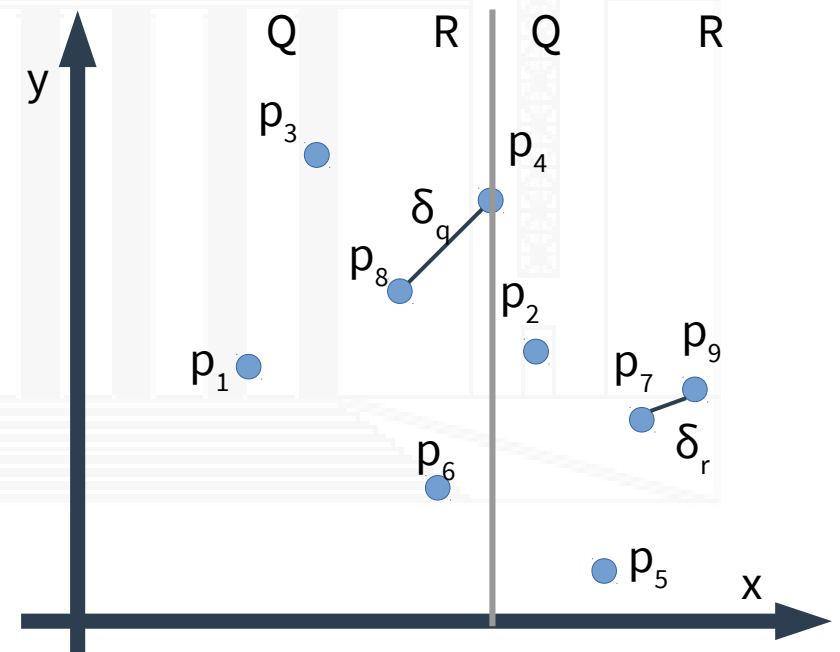
Comparamos los pares de puntos retornados por cada subproblema  $\rightarrow \delta = \min(\delta_q, \delta_r)$

Nos quedamos con el de menor distancia  $O(1)$

## Lo podemos expresar con la recurrencia

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

... que por teorema maestro:  $O(n \log n)$



# Combinación: Sub problemas

**Pero, pero... Hay un problema fundamental en nuestra solución.**

La distancia mínima podría darse entre el punto  $q \in Q$  y el punto  $r \in R$

## Al combinar

Debemos tener en cuenta los puntos de ambos subproblemas

## Comparar los puntos de un lado con los del otro

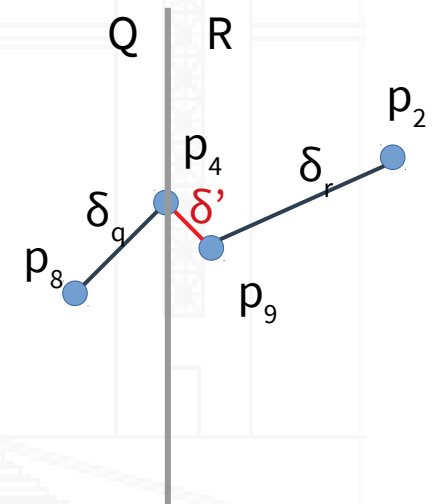
Quedando una recurrencia  $T(n) = 2T(n/2) + O(n^2)$

## Según teorema maestro quedando

Una complejidad  $T(n) = O(n^2)$

## Que indica que no es mejor que fuerza bruta

Debemos encontrar una forma más sencilla de combinar los subproblemas



# Puntos entre los subproblemas

## Llamemos

$x^*$  a la coordenada del punto Q mas a la derecha

L a la linea vertical con la ecuación  $x=x^*$

(L separa Q y R)

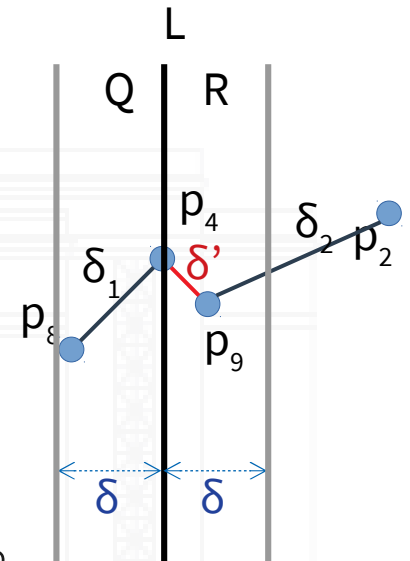
## Vemos que

si distancia mínima esta entre un punto  $q \in Q$  y un punto  $r \in R$

(su distancia tiene que ser menor a  $\delta = \min(\delta_q, \delta_r)$ )

## No pueden estar

a más distancia de  $\delta$  de la linea L



# Puntos entre los subproblemas (cont.)

## Nos interesa conocer aquellos puntos $S \subseteq P$

Que se encuentran a una distancia menor a  $\delta$  de  $L$

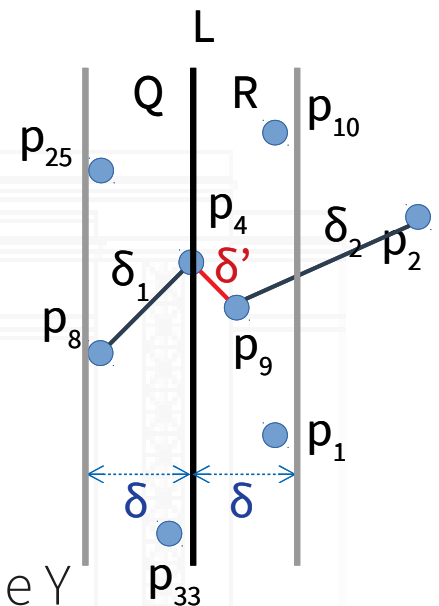
## Podemos

obtenerlos en  $O(n)$  Recorriendo  $P$

## Llamaremos

$S_y$  a la lista de elementos de  $S$  ordenadas por la coordenada de  $Y$

(Que podemos construir en  $O(n)$  mediante  $P_y$ )



$S = p_1, p_4, p_8, p_9, p_{10}, p_{25}, p_{33}$

$S_y = p_{33}, p_1, p_8, p_9, p_4, p_{25}, p_{10}$

# Puntos entre los subproblemas (cont.)

## Podemos ver que

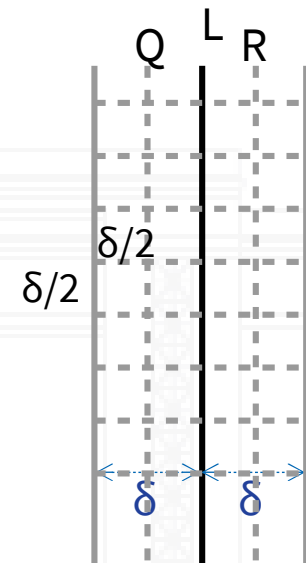
Si subdividimos el espacio entre  $x=L-\delta$  y  $x=L+\delta$   
(como se indica en la figura)

## En celdas

de  $\delta/2 \times \delta/2$

## Solo puede existir un punto P dentro de una celda

(de lo contrario la distancia mínima en Q o en R seria menor a  $\delta$ )



# Puntos entre los subproblemas (cont.)

## Cada punto $s \in S$ estará en una celda

Debemos comparar  $s$  unicamente con otros puntos de  $S$  que estén en celdas cercanas (a no mas de  $\delta$  de distancia)

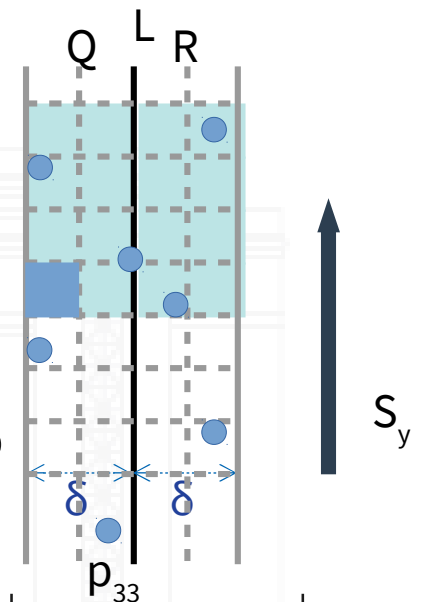
## Recorriendo $S$ en el orden de $S_y$

Podemos ver que unicamente tenemos que comparar a lo sumo con  $15^*$  puntos siguientes (o 15 celdas siguientes)

Es decir que hacemos  $c=15$  comparaciones por punto en  $S$  en solo una pasada

## En el peor escenario tenemos $15n$ comparaciones

Por lo tanto combinar el resultado es  $O(n)$



\* se puede demostrar que se puede reducir de 15 a 7

# Pseudocódigo

Puntos-cercanos(P)

Construir Px y Py //  $O(n \log n)$

$(p_0, p_1) = \text{puntos-cerc-rec}(Px, Py)$

puntos-cerc-rec(Px, Py)

Si  $|Px| \leq 3$

retornar  $(p_0, p_1)$  par mas cercano comparando todos los puntos

Sino

Construir Qx, Qy, Rx, Rz //  $O(n)$

$(q_0, q_1) = \text{puntos-cerc-rec}(Qx, Qy)$

$(r_0, r_1) = \text{puntos-cerc-rec}(Rx, Ry)$

$d = \min(\text{dist}(q_0, q_1), \text{dist}(r_0, r_1))$

$x' = \text{máxima coordenada } x \text{ de punto en } Q$



# Pseudocódigo (cont)

```
L = {(x,y) : x=x'}  
S = puntos de P a distancia d de L  
  
Construir Sy // O(n)  
Por cada punto s de Sy // O(n)  
    computar distancia con próximos 15 puntos de Sy  
    sea s, s' el par de puntos de menor distancia  
  
Si dist(s,s') < d  
    retornar (s,s')  
Sino si dist(q0, q1) < dist(r0, r1)  
    retornar (q0, q1)  
sino  
    retornar (r0, r1)
```



Presentación realizada en Septiembre de 2020

## División y conquista: Multiplicación - Karatsuba

Teoría de Algoritmos I (75.29 / 95.06)

Ing. Víctor Daniel Podberezski

✉ [vpodberezski@fi.uba.ar](mailto:vpodberezski@fi.uba.ar)

# Multiplicación de números enteros

**Sean**

$X$  y  $Y$  números enteros de  $n$  bits cada uno

**Queremos**

Calcular su multiplicación

# Un problema histórico

## Hace milenios la humanidad construyó algoritmos para multiplicar

Método egipcio

Método babilonio

Método japones (o chino)

Método hindú (o de celdillas)



# Método “tradicional”

## Sean X y Y de n bits (dígitos)

Por cada bit (dígito) lo multiplicamos por cada uno de los bits (dígitos) del otro numero

Encolumnamos los resultados y sumamos

## Realizamos

Orden de  $n^2$  multiplicaciones básicas

Mas un conjunto de sumas elementales

## Complejidad $O(n^2)$

$$\begin{array}{r} 4572 \\ \times 3169 \\ \hline 41148 \\ + 27432 \\ 4572 \\ 13716 \\ \hline 14488668 \end{array}$$

# Conjetura $n^2$

## Andrey Nikolaevich Kolmogorov

Sostuvo que no era posible una complejidad mejor  
(No era el único)

**Conjetura: Afirmación que se supone cierta**  
pero que no ha podido ser ni probada ni refutada.

**“Si existiese mejor método, ya tendría que haber sido encontrado”**



# El piso se rompe

## Anatoly Karatsuba

estudiante de 23 años en 1960

seminario de problemas matemáticos en cibernética,

Facultad de mecánica y matemáticas de la universidad de Moscú

Le bastó una semana para refutar la conjetura  $n^2$

Paper "Multiplication of Many-Digital Numbers by Automatic Computers"



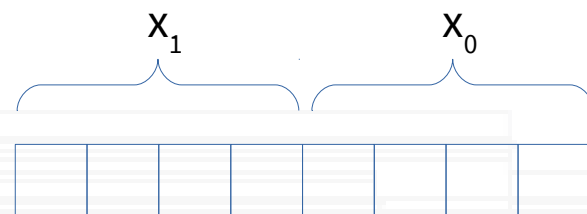


# Idea

## Representamos

$$X = x_1 * 2^{n/2} + x_0$$

$$Y = y_1 * 2^{n/2} + y_0$$



## Operamos

$$X * Y = (x_1 * 2^{n/2} + x_0) * (y_1 * 2^{n/2} + y_0)$$

$$X * Y = x_1 y_1 * 2^n + (x_0 y_1 + x_1 y_0) * 2^{n/2} + x_0 y_0$$

En vez de 1 multiplicación de  $n$  bits tenemos 4 multiplicaciones de  $n/2$  bits

# Idea (cont)

## Realizamos recursivamente la multiplicación

Hasta la multiplicación de 1 a 1 bits (o dígito)

## Relación de recurrencia:

$$T(n) = 4 T(n/2) + cn$$

## Por teorema maestro $O(n^2)$

... por el momento no mejora nada

# El aporte de Karatsuba

## Partimos de

$$X * Y = x_1 y_1 * 2^n + (x_0 y_1 + x_1 y_0) * 2^{n/2} + x_0 y_0$$

## Podemos ver que

$$(x_0 + x_1) * (y_0 + y_1) = x_1 y_1 + (x_0 y_1 + x_1 y_0) + x_0 y_0$$

Ya calculado

Lo que busco

$$(x_0 y_1 + x_1 y_0) = (x_0 + x_1) * (y_0 + y_1) - x_1 y_1 - x_0 y_0$$

Multiplicación de  $n/2$  bits

# Pseudocodigo

## Calculamos:

$X * Y =$

$$x_1 y_1 * 2^{n/2} +$$

$$[(x_0 + x_1) * (y_0 + y_1) - x_1 y_1 - x_0 y_0] * 2^{n/2} + x_0 y_0$$

Son 3 multiplicaciones de  $n/2$  bits

Mas sumas elementales

Cada recursión trabaja con  $n/2$  bits

Karatsuba(x,y)

$$x = x_1 * 2^{n/2} + x_0$$

$$y = y_1 * 2^{n/2} + y_0$$

Calcular  $x_1 + x_0$

Calcular  $y_1 + y_0$

$$P = \text{Karatsuba}(x_1 + x_0, y_1 + y_0)$$

$$x_1 y_1 = \text{Karatsuba}(x_1, y_1)$$

$$x_0 y_0 = \text{Karatsuba}(x_0, y_0)$$

$$\text{Retornar } x_1 y_1 2^n + (P - x_1 y_1 - x_0 y_0) 2^{n/2} + x_0 y_0$$

# Complejidad

## Relación de recurrencia

$$T(n) = 3T(n/2) + cn$$

$$T(1) = a$$

## Por teorema maestro

$$O(n^{\log_2 3}) = O(n^{1,59})$$

## Que es más “eficiente” que el algoritmo tradicional

Este método tiene mas sumas básicas, por lo que la ventaja se ve para n grandes

# Desenrollando la recursión

## Relación de recurrencia

$$T(n) = 3T(n/2) + cn$$

$$T(1) = a$$

$$T(n/2) = 3T(n/4) + n/2$$

$$T(n/4) = 3T(n/8) + n/4$$

...

$$T(n) = 3 [ 3 ( 3 \{ T(n/16) + n/8 \} + n/4 ) + n/2 ] + n$$

$$T(n) = 3^k * 1 + \sum_{i=0}^k \frac{3^i * n}{2^i}$$

## Continuemos

vemos que  $k = \log n$

$$T(n) = 3^k * 1 + \sum_{i=0}^k \frac{3^i * n}{2^i}$$

$$T(n) = 3^{\log(n)} + n * \sum_{i=0}^{\log(n)} \left(\frac{3}{2}\right)^i$$

$$T(n) = 3^{\log(n)} + \frac{n * (1 - (3/2)^{\log(n)+1})}{(1 - 3/2)}$$

$$T(n) = 3^{\log(n)} + 2n * ((3/2)^{\log(n)+1} - 1)$$

$$T(n) = 3^{\log(n)} + 3n * (3/2)^{\log(n)} - 2n$$

$$\sum_{i=0}^x r^i = \frac{1 - r^{(x+1)}}{1 - r}$$

Y continuamos...

$$T(n) = 3^{\log(n)} + 3n \frac{3^{\log(n)}}{2^{\log(n)}} - 2n$$

$$T(n) = 3^{\log(n)} + 3 * 3^{\log(n)} - 2n$$

$$T(n) = 4 * 3^{\log(n)} - 2n$$

$$T(n) = 4 * n^{\log 3} - 2n$$

Si  $n \rightarrow \infty$

$$O(n^{\log_2 3}) = O(n^{1,59})$$

$$a^{\log_2 n} = n^{\log_2 a}$$



# Evolución de la multiplicación

## Se puede hacer mejor?

Nueva conjetura de Schönhage-Strassen (1971)  $\rightarrow O(n \log n)$

### Avances en los métodos de multiplicación.

| Año    | $M(n)$                              | autor                 |
|--------|-------------------------------------|-----------------------|
| -2000? | $O(n^2)$                            | desconocido           |
| 1960   | $O(n^{\log_2 3})$                   | Karatsuba             |
| 1966   | $O(n \log n e^{\sqrt{2 \log_2 n}})$ | Toom-Cook             |
| 1971   | $O(n \log n \log \log n)$           | Schönhage-Strassen    |
| 2007   | $O(n \log n K^{\log^* n})$          | Fürer                 |
| 2018   | $O(n \log n 4^{\log^* n})$          | Harvey-van der Hoeven |

# Se ha llegado al máximo?

## David Harvey y Joris van der Hoeven presentan en 2019

Paper: “Integer multiplication in time  $O(n \log n)$ ”

<https://hal.archives-ouvertes.fr/hal-02070778/document>

## Se apoyan en la FFT pero la llevan a 1729 dimensiones

FFT: Fast Fourier transform

Técnica utilizada por primera vez por Schönhage y Strassen



Presentación realizada en Abril de 2020

# División y conquista: puntos extremos en polígonos

Teoría de Algoritmos I (75.29 / 95.06)

Ing. Víctor Daniel Podberezski

✉ [vpodberezski@fi.uba.ar](mailto:vpodberezski@fi.uba.ar)

# Obtención de extremos de un polígono

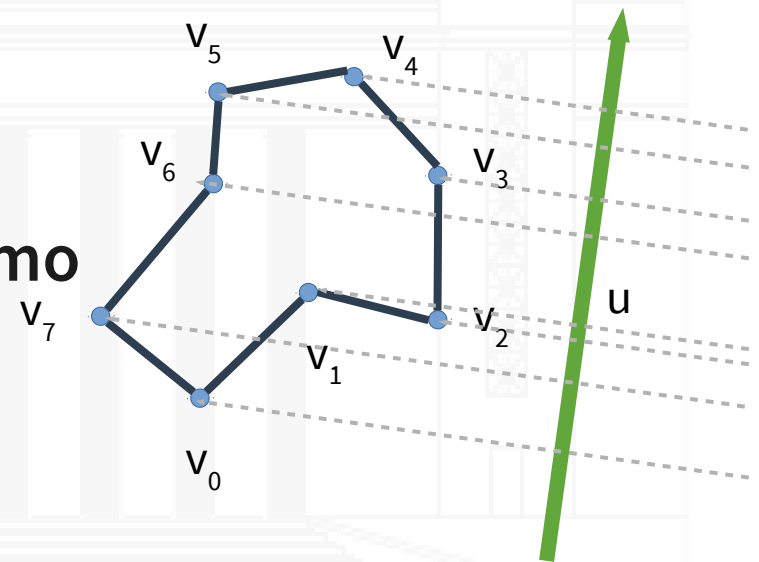
Sea un polígono de  $n$  vértices  $V=(v_0, v_1, \dots, v_{n-1}, v_n)$

con  $v_n = v_0$

en orden contra reloj

Queremos determinar el vértice extremo

(mayor o menor) a un eje  $u$



# Solución por fuerza bruta

## Por cada punto $v_i$

Realizamos la proyección ortogonal a  $u$

Verificamos si es mayor (o menor) al mayor (o menor) de los anteriores

## Complejidad algorítmica

$O(n)$

## Se puede hacer mejor?

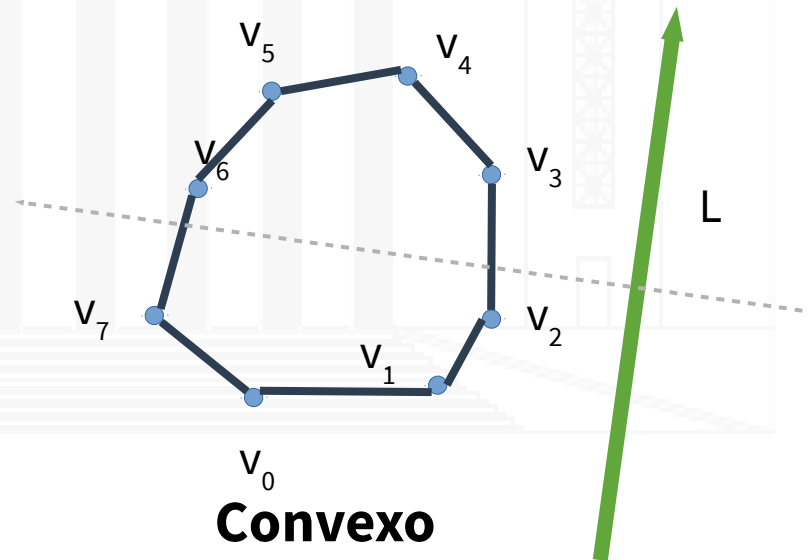
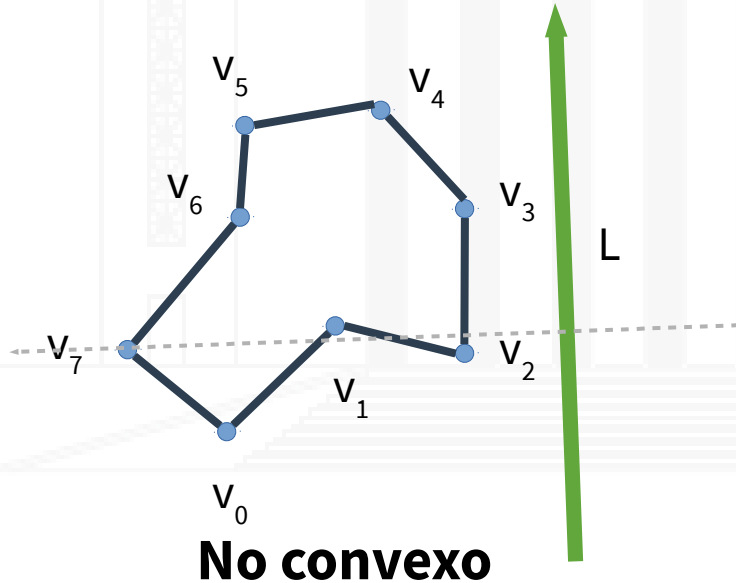
Si, aunque SOLO si el polígono es CONVEXO

# Polígonos convexos

## Un polígono es convexo si

Es monotónico para todo L segmento (Cualquier linea perpendicular a L cortará a los sumo 2 veces al polígono)

Sus ángulos interiores miden a lo sumo 180 grados



# Nomenclatura

## Llamaremos:

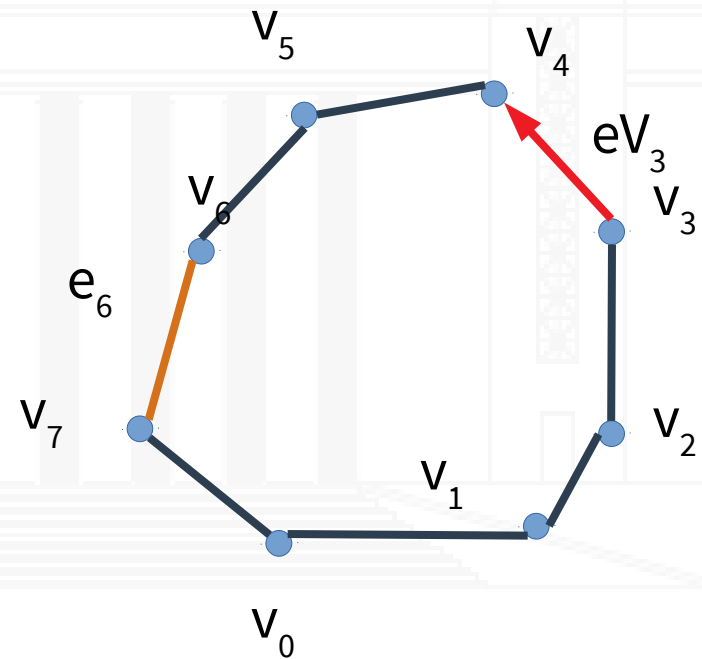
$e_i$  al  $i$ -ésimo segmento del vértice  $v_i$  a  $v_{i+1}$  para  $i=0$  a  $n-1$

$ev_i$  al vector  $V_{i+1} - V_i$

**$ev_i$  puede, en la proyección  $u$ ,**

Tener sentido positivo o negativo

“para arriba” o “para abajo”





# Idea

## Supongamos que

el vértice máximo se encuentra en el polígono entre los vértices  $V_a$  y  $V_b$

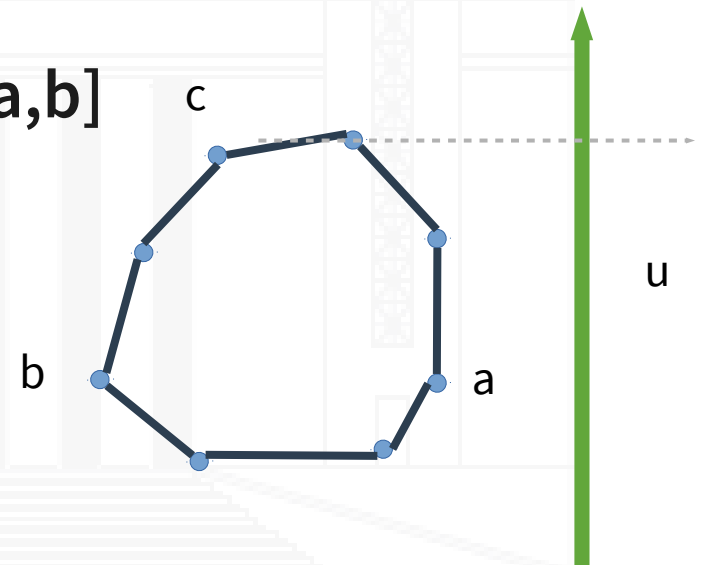
## Podemos extresar la línea poligonal como $[a,b]$

$$[a,b] = \{a, a+1, \dots, a+k=b \pmod{n}\} \quad k>0$$

## Seleccionamos un vértice $V_c$ entre $a$ y $b$

Dividimos en 2 líneas poligonales  $[a,c]$  y  $[c,b]$

El máximo estará en alguno de los 2 segmentos

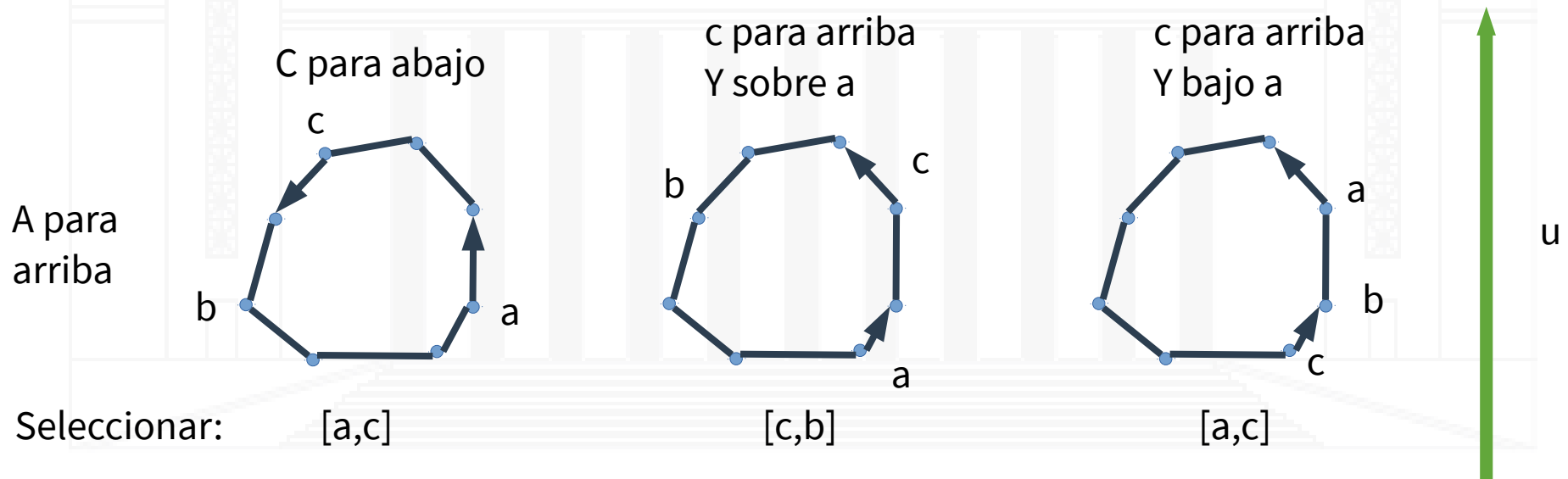


# Idea (cont.)

## Analizamos el vector $eV_a$ y $eV_c$

Tenemos 6 casos diferentes

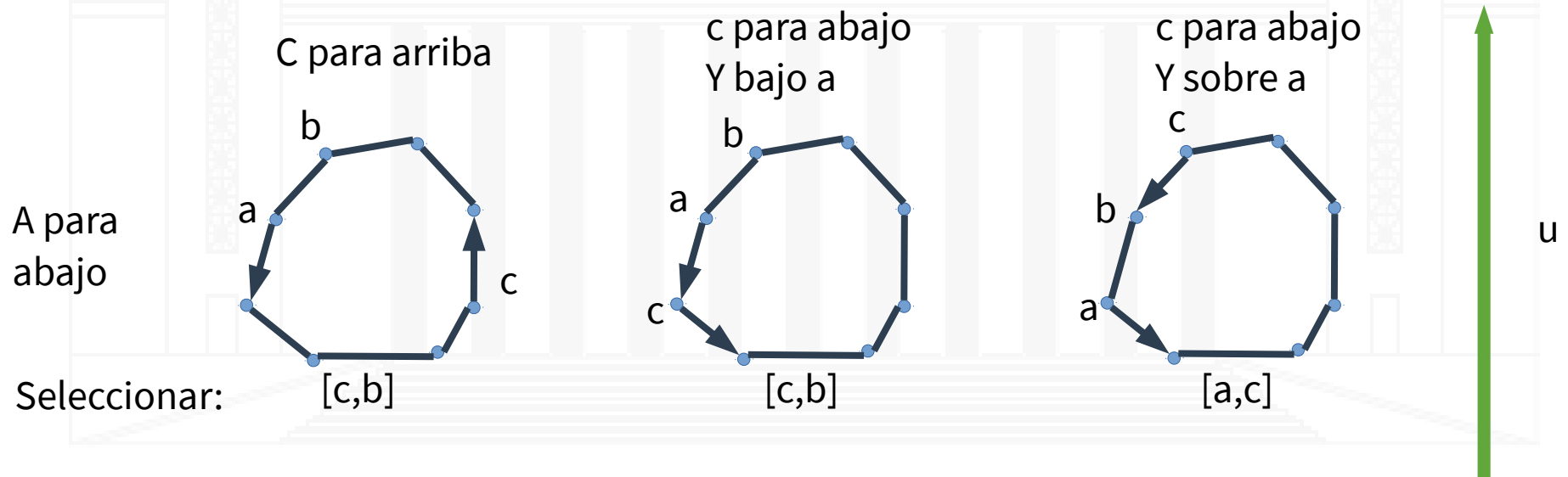
Los primeros 3



# Idea (cont.)

## Analizamos el vector $eV_a$ y $eV_c$ (cont.)

Los últimos 3



# Solución

## Iniciamos con

$a=0$  y  $b=0$  (todo el polígono)

$c=n/2$  (el punto intermedio)

## Iteramos

Si solo nos quedan 3 vértices compararlos uno a uno y obtener el máximo

Sino determinar cual de los casos aplica.

Actualizar  $a, b$  y  $c$  según corresponda

## Al finalizar tenemos el punto extremo

# Complejidad

## En cada iteración

Reducimos a la mitad la cantidad de vértices

Se convierte en un nuevo sub problema

Realizamos operaciones  $O(1)$

## Relación de recurrencia

$$T(n) = T(n/2) + c$$

## Por teorema maestro

$$O(\log n)$$



Presentación realizada en Abril de 2020