Algoritmos y Estructuras de Datos II
Clase 8

Carreras:

Licenciatura en Informática

Ingeniería en Informática

2024

Divide & Conquer
Algoritmo de Mezcla por Fusión

Merge-sort

Propuesto en 1945 por John Von Newmann

Idea:

- Si el arreglo tiene 0 o 1 elemento está ordenado.
- Sino Dividir el arreglo en 2 partes.
- Ordenar cada parte recursivamente
- Mezclar las 2 partes ya ordenadas para conseguir el arreglo ordenado.

Divide & Conquer
Algoritmo de Mezcla por Fusión

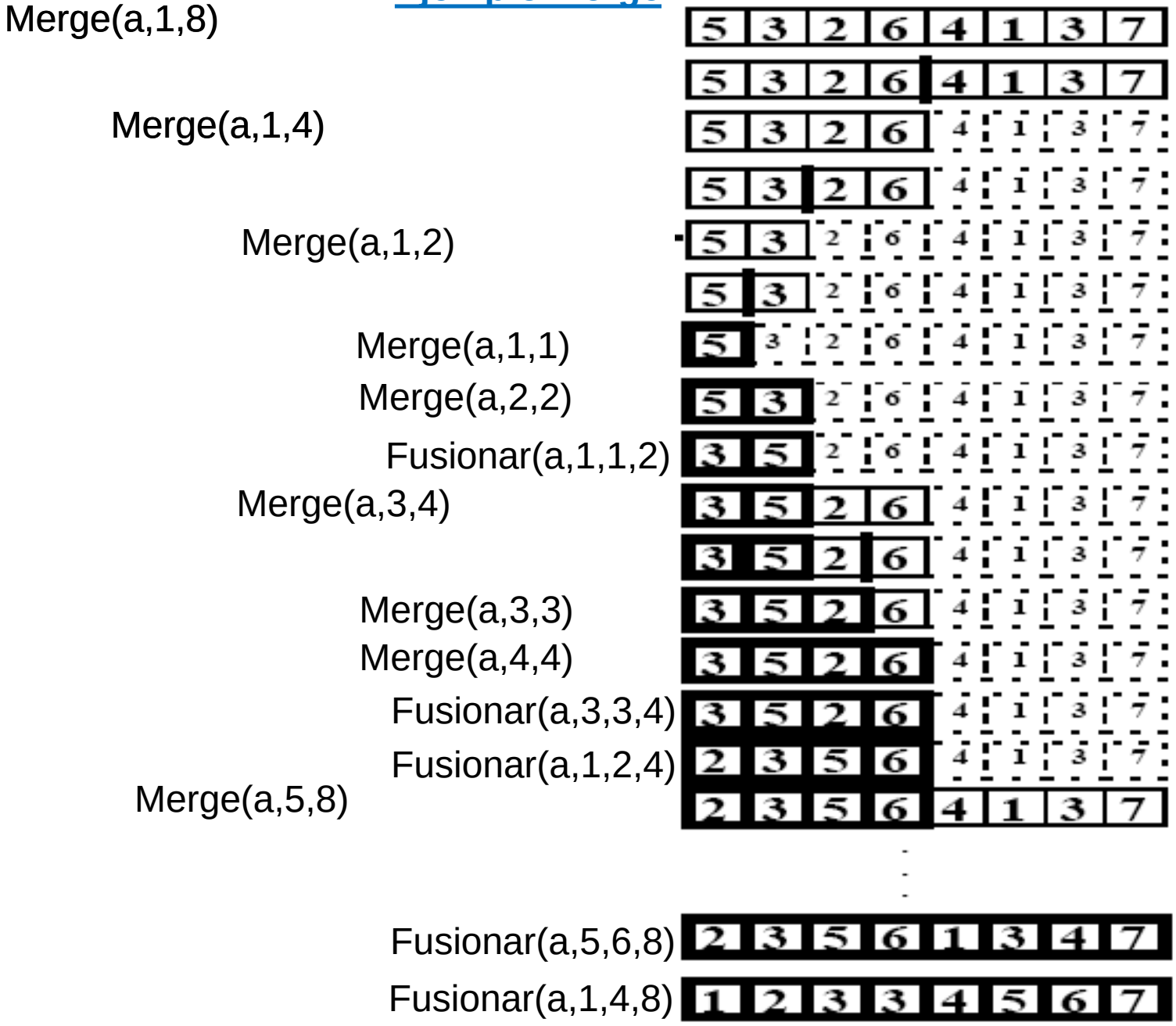
```
Algoritmo Merge(a,izq,der)

Entrada: a: arreglo de n items
   izq, der: números enteros positivos correspondientes a los
   indices entre los cuales se ordenan las componentes del arreglo
Salida: a: arreglo de n items ORDENADO entre izq y der

Si  der>izq entonces
      m←(der+izq)/2
      Merge(a,izq,m)
      Merge(a,m+1,der)
      Fusionar(a,izq,m,der)
Fin
```

Se invoca con: **`Merge (a,1,n)`**

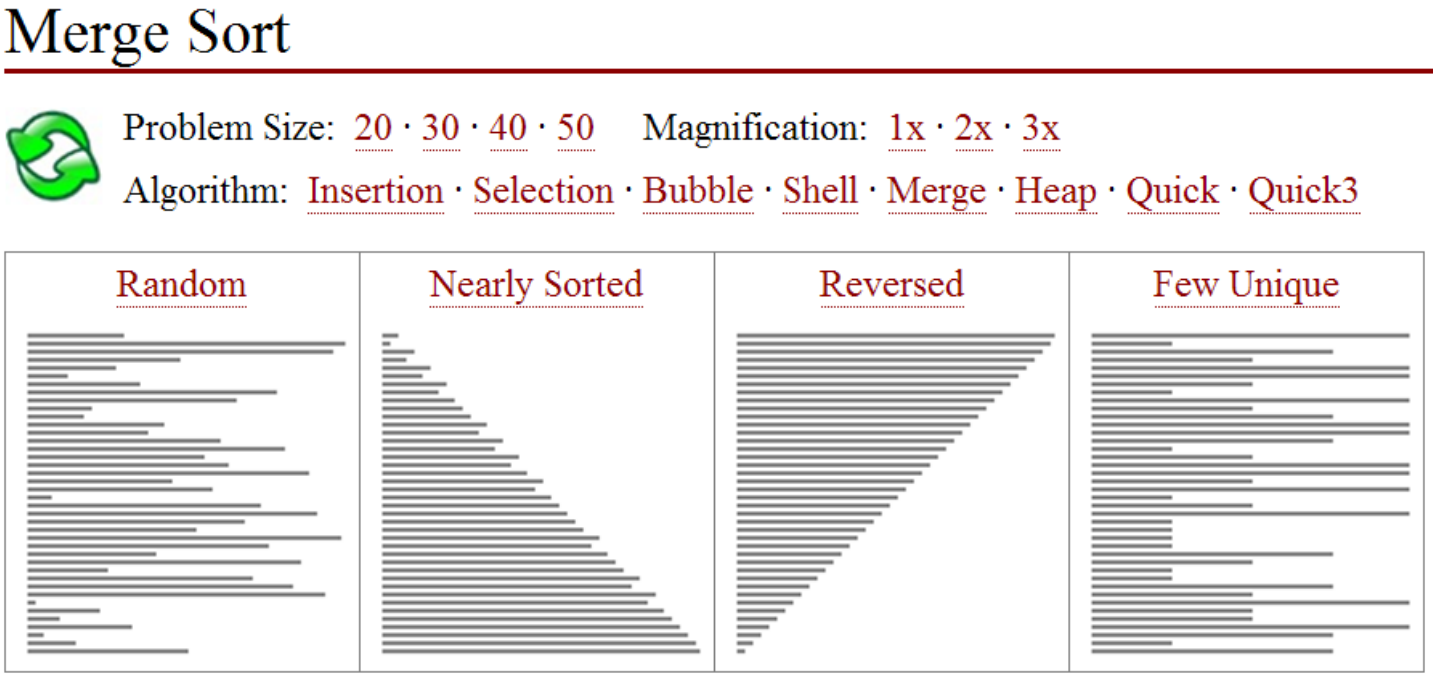
Divide & Conquer
Algoritmo de Mezcla por Fusión

```
Algoritmo Fusionar (a,izq,m,der)
     // Fusiona la parte: a(izq,m) con: a(m+1,der)
    Auxiliar b(max)
    Para i=izq,m hacer
       b(i) ← a(i)
    Para j=m, der-1 hacer
       b(der+m-j) ← a(j+1)
    i←izq
    j←der
    Para k=izq, der
      Si b(i)< b(j) entonces
              a(k)←b(i)
              i←i+1
      sino
              a(k)← b(j)
              j←j-1
Fin
```

Ejemplo Merge



Divide & Conquer
Algoritmo de Mezcla por Fusión



https://www.toptal.com/developers/sorting-algorithms/merge-sort

Divide & Conquer
Algoritmo de Mezcla por Fusión

Costos:

- **Fusionar** $\in \Theta(n)$
- **Merge** $\in$ **?**

$$T(n) = a\,T(n/b) + c\,n^k \quad (teorema)$$
$$T(n) = 2T(n/2) + c\,n^1 \quad si\ n>0$$

Según el teorema:

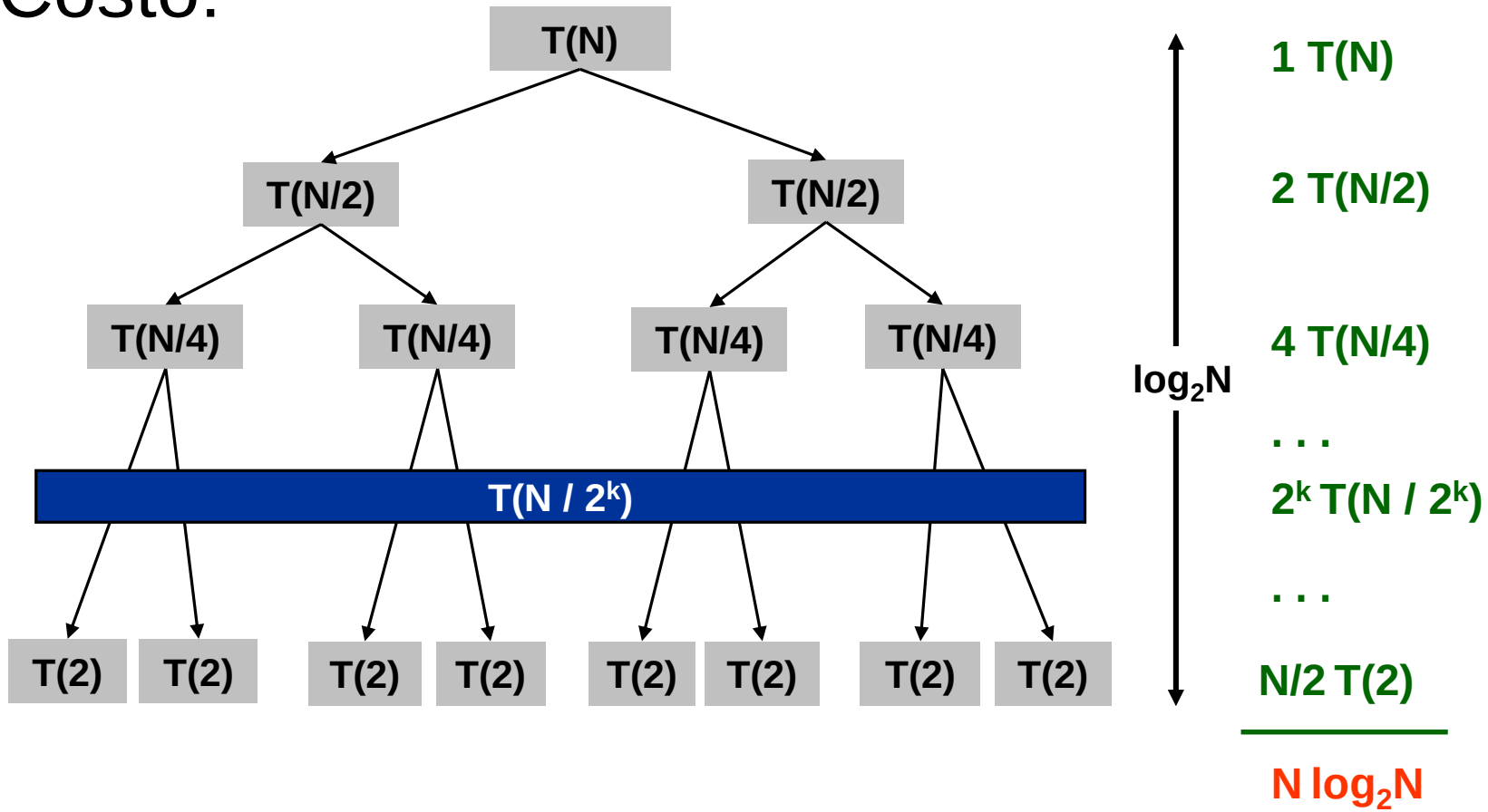
si: $a = b^k$ $\quad T(n) \in \Theta(n^k \log_b n)$

como: $2 = 2^1$ $\quad T(n) \in \Theta(n^1 \log_2 n)$

Merge $\in \Theta(n \log_2 n)$

Divide & Conquer
Algoritmo de Mezcla por Fusión

Costo:



Divide & Conquer
Búsqueda del máximo y del mínimo

Aplicando algoritmo clásico se recorre el vector secuencialmente:

```
Funcion MaxMin(A,n,max,min)
max ← A(1)
min ← A(1)
Para i=2,n hacer
      Si A(i) > max entonces
            max ← A(i)
      sino Si A(i) < min entonces
            min ← A(i)
      fin Si
fin Para
Fin
```

Esta función tiene 4 parámetros dos de entrada: **A** y **n** y dos de salida: **max** y **min**

Divide & Conquer
Búsqueda del máximo y del mínimo

Número de operaciones-algortimo clásico

Comparaciones:

El caso **más *favorable*** para el numero de comparaciones es cuando se cumple que **`A(i)>max`** repetido para **`i=2,n`**, lo que resulta en:

`(n-1)` comparaciones.

El caso **más *desfavorable*** se da cuando para **`i=2,n`** es siempre **`A(i)≤max`** (cuando el máximo sea **`A(1)`**), entonces se tiene además las otras comparaciones: **`A(i)<min`** lo que resulta en: **`2(n-1)`** comparaciones.

```
Funcion MaxMin(A,n,max,min)
max ← A(1)
min ← A(1)
Para i=2,n hacer
      Si A(i) > max entonces
            max ← A(i)
      sino Si A(i) < min entonces
            min ← A(i)
      fin Si
fin Para
Fin
```

Divide & Conquer
Búsqueda del máximo y del mínimo

Número de operaciones-algortimo clásico.

Asignaciones:

El numero **más *favorable*** de asignaciones se da cuando nunca **`A(i)>max`** ni **`A(i)<min`** entonces se tiene solo 2 asignaciones.

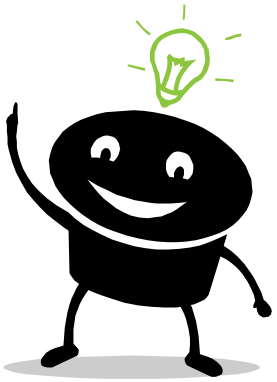
El caso **más *desfavorable*** es que se realice siempre una asignación dentro de la iteración, esto resulta un total de **`(n+1)`** asignaciones

```
Funcion MaxMin(A,n,max,min)
max ← A(1)
min ← A(1)
Para i=2,n hacer
      Si A(i) > max entonces
            max ← A(i)
      sino Si A(i) < min entonces
            min ← A(i)
      fin Si
fin Para
Fin
```

Divide & Conquer
Búsqueda del máximo y del mínimo

Aplicando D&C: se divide por la mitad hasta tener uno o dos elementos:

```
Funcion MaxMinD&C(A,n,i,j,max,min)
  Si i < j-1 entonces    // 3 o mas elementos
        k ← (i+j)/2
        MaxMinD&C(A,n,i,k,max1,min1)
        MaxMinD&C(A,n,k+1,j,max2,min2)
        Si max1 > max2 entonces
                max ← max1
        sino
                max ← max2
        Si min1 < min2 entonces
                min ← min1
        sino
                min ← min2
  sino  Si i = j-1 entonces     // 2 elementos
                    Si A(i) < A(j) entonces
                            max ← A(j) ;   min ← A(i)
                    sino
                            max ← A(i) ;   min ← A(j)
        sino    // 1 solo elemento
                max ← A(i) ;   min ← max
Fin
```



Divide & Conquer
Búsqueda del máximo y del mínimo

Para calcular el número de operaciones se considera que n es una potencia de 2.

Comparaciones:

$T(n)=2T(n/2)+2$ si $n>2$ y n potencia de 2

$T(2)=1$

Se reescribe la recurrencia:

$$T(n)-2T(n/2)=2$$

Haciendo el cambio de variable $n=2^k$ se tiene:

$$T(2^k)-2T(2^{k-1})= 2$$

Llamando $t_k=T(2^k)$ queda la ecuación: $t_k -2\, t_{k-1}= 2$

Divide & Conquer
Búsqueda del máximo y del mínimo

El polinomio característico es: $(x-2)(x-1)$ con soluciones 2 y 1.
Entonces:

$$t_k = c_1 2^k + c_2 1^k = c_1 2^k + c_2$$

Volviendo a que $T(n) = T(2^k) = t_k$ queda la ecuación como:

$$T(n) = c_1 n + c2$$

Con las condiciones iniciales:

$n=2$	$T(2)=1$	$2c_1 + c_2 = 1$
$n=4$	$T(4)=4$	$4c_1 + c_2 = 4$

De donde: $c_1 = 3/2$ y $c_2 = -2$

$$\boldsymbol{T(n) = 3/2n - 2 \text{ , } T(n) \in \Theta(n)}$$

Divide & Conquer
Búsqueda del máximo y del mínimo

Asignaciones:

$T(n)=2T(n/2)+2$ si $n>2$ y n potencia de 2

$T(2)=2$

La recurrencia se resuelve como antes cambiando las condiciones iniciales. Con las condiciones iniciales:

$n=2$	$T(2)=2$	$2c_1+c_2=2$
$n=4$	$T(4)=6$	$4c_1+c_2=6$

De donde $c_1=2$ y $c_2=-2$

$T(n)= 2n- 2$

$T(n) \in \Theta(n)$

Divide & Conquer
Búsqueda del máximo y del mínimo

Conclusión:

Aunque los dos métodos tengan el mismo costo de crecimiento del tiempo de ejecución en notación Ogrande, la elección de un método o de otro depende del ***costo*** de las operaciones de *comparación* y de *asignación*.

	MaxMin	MaxMinD&C
Comparaciones	2 (n-1)	3/2n-2
Asignaciones	(n+1)	2n-2

- Si se considera que las asignaciones son más costosas, el método directo es preferible.
- Si las comparaciones son más costosas que las asignaciones puede ser preferible el método D&C.

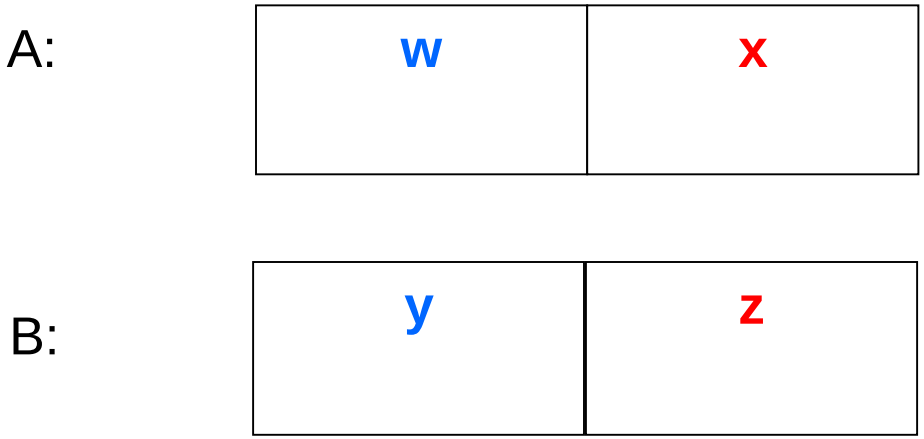
Hay que tener en cuenta que el método D&C es recursivo y dependiendo de la implementación esto puede ser un costo adicional en memoria y en tiempo de ejecución.-

Divide & Conquer
Multiplicación de enteros grandes

Considere la multiplicación de dos enteros A y B de ***n dígitos*** cada uno: AxB

El algoritmo de la escuela primaria implica la multiplicación dígito a dígito de cada número por las del otro número. Esto implica n^2 multiplicaciones y luego una suma.

Un enfoque Divide & Conquer divide A y B en dos enteros cada uno de n/2 dígitos:



Divide & Conquer
Multiplicación de enteros grandes

Ejemplo de multiplicación de dos números: AxB con algoritmo D&C

Se necesita que los números tengan igual número de cifras, y que este número de cifras sea una potencia de 2.

Ej. AxB = 2345 x 1234 =

	A	B	Desplazar a izquierda	Resultado						
1)	23	12	4	2	7	6	-	-	-	-
2)	23	34	2			7	8	2	-	-
3)	45	12	2			5	4	0	-	-
4)	45	34	0				1	5	3	0
Sumar				2	8	9	3	7	3	0

Divide & Conquer
Multiplicación de enteros grandes

En este ejemplo se ha pasado de multiplicar números de 4 cifras a multiplicar números de 2 cifras. Cada una de estas multiplicaciones de 2 cifras a su vez se puede resolver con la misma metodología. De este modo se puede operar de modo que las multiplicaciones se reduzcan a varias multiplicaciones de 1 cifra y sumas.

23 x 12 =

	A	B	Desplazar a izquierda	Resultado		
1)	2	1	2	2	-	-
2)	2	2	1		4	-
3)	3	1	1		3	-
4)	3	2	0			6
Sumar				2	7	6

Divide & Conquer
Multiplicación de enteros grandes

$A = w10^{n/2} + x$

$B = y10^{n/2} + z$

$AxB=(w10^{n/2}+x).(y10^{n/2}+z)= w.y\ 10^{n} + (w.z+x.y)\ 10^{n/2}+x.z$

Se ha transformado la multiplicación de AxB en 4 multiplicaciones de enteros de n/2 dígitos, tres sumas y dos desplazamientos (potencias de 10).

La recurrencia es de la forma:

$T(1)=1$

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

Lo que resulta en **$T(n)\in O(n^2)$** que no resulta ser una mejora frente al algoritmo tradicional.

Divide & Conquer
Multiplicación de enteros grandes

Este algoritmo que necesita 4 multiplicaciones y una suma en cada paso, no es más rápido que el algoritmo clásico, pero puede admitir algunas mejoras.

A= 2345 w= 23 x= 45
B= 1234 y= 12 z= 34

$A = 10^2 w + x$
$B = 10^2 y + z$

$A \times B = (10^2 w + x).(10^2 y + z) = 10^4 w.y + 10^2 (w.z + x.y) + x.z$

necesita calcular 4 multiplicaciones: w.y, w.z, x.y, x.z

Divide & Conquer
Multiplicación de enteros grandes

$AxB = 10^4\mathbf{w.y}+10^2(\mathbf{w.z+x.y})+\mathbf{x.z}$

Sea:

$p = w.y \qquad q = x.z$

$r = (w+x).(y+z) = w.y+(w.z+x.y)+x.z$

De donde:

$(w.z+x.y) = r - w.y - x.z = r - p - q$

De modo que el producto se puede escribir en función de la suma de estas 3 multiplicaciones p, q, r:

$AxB = 10^4p + 10^2(r - p - q) + q$

Divide & Conquer
Multiplicación de enteros grandes

De modo que el producto se puede escribir en función de la suma de estas 3 multiplicaciones:

$$AxB = 10^4p + 10^2(r - p - q) + q$$

En números:

p=23x12=276

q=45x34=1530

r=68x46=3128

AxB=2760000+132200+1530=2893730

Este algoritmo hace una multiplicación menos, pero hace 4 sumas más. Vale la pena hacer 4 sumas más para ahorrar una multiplicación?.

La respuesta es: "***no*** cuando los números son chicos como en este ejemplo".

Divide & Conquer
Multiplicación de enteros grandes

El algoritmo obtenido puede multiplicar dos números de n cifras en un tiempo: $T(n)=3T(n/2) + c.n$ cuando n es par y suficientemente grande.

Resolviendo esta recurrencia se obtiene que:

$$T(n) \in O(n^{\log_2 3})$$

$$T(n) \in O(n^{1.585})$$

Esto representa una mejora del algoritmo clásico que es $O(n^2)$.

Divide & Conquer
Multiplicación de enteros grandes

Hasta aquí se consideraron varias suposiciones que hay que generalizar.

Los números de *longitud impar* se multiplican fácilmente partiéndolos del modo más parejo posible.

Un número de n cifras con n impar se parte en dos números con las siguientes cantidades de cifras:

$$\left\lceil \frac{n}{2} \right\rceil \qquad \left\lfloor \frac{n}{2} \right\rfloor$$

Divide & Conquer
Multiplicación de enteros grandes

Otro problema se presenta cuando para calcular r se tiene una multiplicación de una cifra más que en las de p y de q.

Por ejemplo:

w=56	x=78
y=67	z=89

p=w.y=56x67

q=x.z=78x69

r=(w+x).(y+z)=134x156

El tamaño de los factores de r no supera $\left\lceil \frac{n}{2} \right\rceil + 1$ cifras.

Divide & Conquer
Multiplicación de enteros grandes

- Para analizar el tiempo de ejecución en este caso se considera que para n es suficientemente grande, el algoritmo reduce la multiplicación de dos números de tamaño menor o igual a n (en cantidad de cifras) a tres multiplicaciones p, q y r de números de tamaños menores o iguales a $\lceil \frac{n}{2} \rceil$, $\lfloor \frac{n}{2} \rfloor$ y $\lceil \frac{n}{2} \rceil + 1$ cifras respectivamente. Además se requieren algunas operaciones que son O(n).
- Por lo tanto existe una constante c real y positiva tal que:

$$T(n) \leq T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + T\left(\left\lceil \frac{n}{2} \right\rceil\right) + T\left(\left\lceil \frac{n}{2} \right\rceil + 1\right) + cn$$,para n suficientemente grande.

Esto da lugar a decir que: $T(n) \in O(n^{\log_2 3}) = O(n^{1.585})$

Divide & Conquer
Multiplicación de enteros grandes

Otro problema es multiplicar números A y B de ***distintos tamaños***.

Sean: m y n a las cantidades de cifras respectivas de A y de B. Si m y n no difieren en más de un factor 2, se puede rellenar el operando más pequeño con ceros adelante para hacerlos de la misma longitud.

Si se supone: $m \leq n$.

El algoritmo D&C con relleno requiere: $T(n) \in O(n^{\log_2 3})$

El algoritmo clásico requiere: $T(n,m) \in O(n.m)$

Se puede demostrar que el algoritmo D&C es más lento que el clásico cuando: $m < n^{1/2}$.

Divide & Conquer
Multiplicación de enteros grandes

Se puede combinar los dos algoritmos para obtener un algoritmo mejor. Idea: Segmentar el operando más largo B, en bloques de tamaño m y usar el algoritmo D&C para multiplicar A por cada bloque de B, de manera de usarlo para multiplicar bloques de igual tamaño. El producto final AxB se obtiene entonces fácilmente mediante sumas y desplazamientos simples.

El tiempo total de ejecución está dominado por la necesidad de efectuar $\left\lceil \frac{n}{m} \right\rceil$ multiplicaciones de números de m cifras,
cada una requiere un tiempo que es: $O(m^{\log_2 3})$

El tiempo total de ejecución para multiplicar un número de n cifras por otro de m cifras es: $O(n.m^{\log_2(3/2)})$ cuando $m \leq n$.

Divide & Conquer
Multiplicación de enteros grandes

Por lo tanto este método es de orden de complejidad: $T(n,m) \in O(n.m^{0.585})$

Es menor que el tradicional: $T(n,m) \in O(n.m)$

¿ Por qué entonces no se enseña y se usa en las escuelas?.

Existen dos razones importantes para no usarlo:
1) es menos intuitivo que el método clásico, a pesar de ser más eficiente,
2) las constantes de proporcionalidad que se obtienen hacen que el método sea más eficiente que el tradicional a partir de números de más de 500 bits y los números que en general multiplicamos son menores que ese tamaño.-

Divide & Conquer
Problema del número mas cercano

Para una dimensión el problema del par mas cercano consiste en encontrar la diferencia de los dos números mas cercanos en un conjunto de n números reales.

Se puede suponer que los puntos están almacenados en un vector.

Un primer algoritmo básico

En este caso se puede considerar un algoritmo muy simple que:

- Ordenar el vector de manera creciente
- Secuencialmente se va comparando las diferencia de los pares de puntos adyacentes para buscar la mínima diferencia.

Costo del algoritmo básico: $T(n) \in \Theta(n \log_2 n) + \Theta(n)$

$T(n) \in \Theta(n \log_2 n)$

Divide & Conquer
Problema del número mas cercano

Un algoritmo con la metodología D&C:

- **Divide**: Se puede dividir en 2 partes el arreglo y encontrar la diferencia del par mas cercano en la primera mitad del arreglo y la diferencia del par mas cercano en la segunda mitad.

- **Conquer**: el par mas cercano de todo el arreglo será el mínimo entre 3 valores m1, m2 y d:
  - ✓ m1 = mínimo de la primera mitad del arreglo.
  - ✓ m2 = mínimo de la segunda mitad del arreglo.
  - ✓ d = diferencia entre el punto más a la izquierda de la segunda mitad del arreglo y el punto más a la derecha de la primera mitad del arreglo.

Divide & Conquer
Problema del número mas cercano

```
Funcion NumeroCercano(A,izq,der)
                :arreglo ordenado x int x int → real

Si izq = der entonces // 1 solo
       Retorna ∞
Sino
       Si der-izq = 1 entonces // 2
              Retorna A(der)-A(izq)
       Sino          // mas de dos
              m ← (izq + der)/2
              m1 ← NumeroCercano(A,izq,m)
              m2 ← NumeroCercano(A,m+1,der)
              Retorna MINIMO3( m1 , m2, A(m+1)-A(m))
Fin Si
Fin
```

Se invoca con NumeroCercano(A,1,n)

Divide & Conquer
Problema del número mas cercano

Tiempo de ejecución de **`NumeroCercano`**

La recurrencia de tiempo T(n) se puede calcular por:

$T(n)=2T(n/2)+c$ *si n>2 y n potencia de 2*

Resolviendo con el teorema master se tiene $a=2$, $b=2$, $k=0$, resulta $a>b^k$

De modo que **si** $a > b^k$: $T(n) \in \Theta(n^{\log_b a})$

En este caso : $T(n) \in \Theta(n^{\log_2 2})$, $T(n) \in \Theta(n)$

Considerando el tiempo de ordenar el arreglo : $T(n) \in \Theta(n \log_2 n)$

Resulta: $T(n) \in \Theta(n \log_2 n) + \Theta(n)$

$T(n) \in \Theta(n \log_2 n)$

Divide & Conquer
Problema del par mas cercano

Se ahora se consideran ahora n puntos del plano:

$P1 = (x1, y1), \ldots, Pn = (xn, yn)$ donde n es potencia de 2 (por simplicidad)

Se quiere encontrar los 2 puntos entre los n dados que tengan distancia mínima.

Se puede suponer que los puntos están ordenados en orden creciente de sus abscisas.

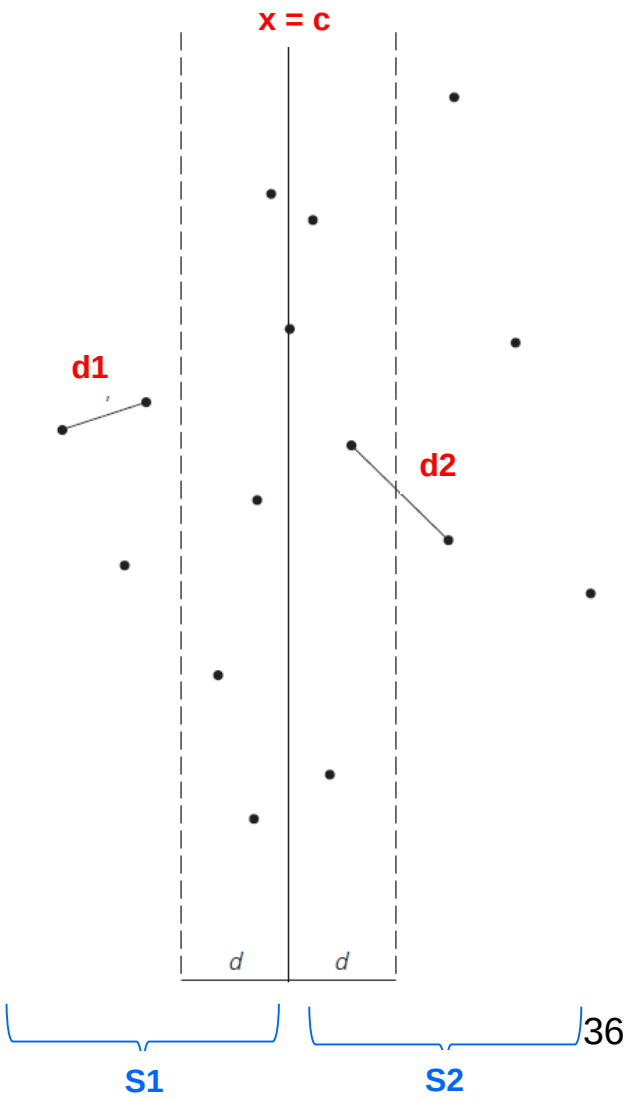
Con una estrategia D&C:

Para encontrar el par mas cercano se puede pensar en dividir los puntos del plano en dos subconjuntos S1 y S2 de $n/2$ puntos cada uno trazando una línea vertical x=c de modo que la mitad de los puntos queden a cada lado. (c puede ser la mediana de las abscisas)

Divide & Conquer
Problema del par mas cercano

En esta estrategia D&C:

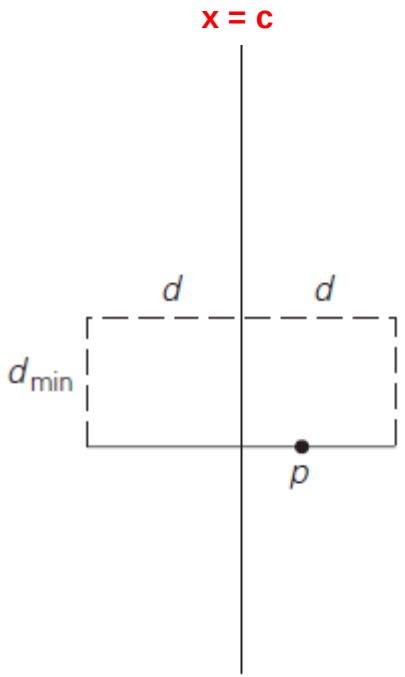
- Se puede ir encontrando recursivamente los puntos mas próximos en cada subconjunto S1 y S2.
- Sean d1 y d2 esas distancias mínimas de cada subconjunto y d=mínimo(d1,d2)



Divide & Conquer
Problema del par mas cercano

En esta estrategia D&C:

- d no será necesariamente la distancia mínima , porque podría haber un par de puntos de distancia mínima separados por la recta.
- Habrá que inspeccionar las distancias de los puntos en una franja de ancho 2d centrada en la recta x=c
- Entonces cuando se combinan las soluciones de los problemas mas chicos hay que tener en cuenta esta situación.-



Divide & Conquer
Problema del par mas cercano

Tiempo de ejecución:

La recurrencia de tiempo T(n) se puede calcular por:

$$T(n)=2T(n/2)+ c.n \qquad si\ n>2\ y\ n\ potencia\ de\ 2$$

Resolviendo con el teorema master se tiene $a=2$, $b=2$, $k=1$, resulta $a=b^k$

De modo que si $a = b^k$: $T(n) \in \Theta(n \log_2 n)$

Para ordenar los pares por las abscisas y ordenadas : $T(n) \in \Theta(n \log_2 n)$

Resulta: $T(n) \in \Theta(n \log_2 n)+ \Theta(n \log_2 n)$

$T(n) \in \Theta(n \log_2 n)$

Divide & Conquer
Multiplicación de matrices

Sean A y B dos matrices cuadradas de orden n y se quiere calcular el producto $C=A.B$

El algoritmo clásico de multiplicación se escribe en base a su definición. El elemento genérico de la matriz C está dado por la fórmula:

$$c_{i,j} = \sum_{k=1}^{n} a_{i,k} b_{k,j}$$

$$\begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \cdot \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nn} \end{bmatrix}$$

Divide & Conquer
Multiplicación de matrices

Aplicando Algoritmo *clásico*:

Para $i=1,n$ hacer
  Para $j=1,n$ hacer
    $c_{ij} \leftarrow 0$
    Para $k=1,n$ hacer
      $c_{ij} \leftarrow c_{ij} + a_{ik} \cdot b_{kj}$

Para cada elemento de la matriz C se necesitan en el orden de n operaciones elementales. Como son n^2 elementos para calcular, esto que resulta en:

Multiplicación clásica $\in \Theta(n^3)$

Divide & Conquer
Multiplicación de matrices

Aplicando la metodología *D&C*:

Cada una de las matrices de tamaño n se divide en n/2 en cada dimensión, lo que resulta 4 submatrices de tamaño n/2:

$$\begin{bmatrix} r & s \\ t & u \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \cdot \begin{bmatrix} e & f \\ g & h \end{bmatrix}$$

$$C = A \cdot B$$

Divide & Conquer
Multiplicación de matrices

Para calcular:

$$\begin{bmatrix} r & s \\ t & u \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \cdot \begin{bmatrix} e & f \\ g & h \end{bmatrix}$$

Se realizan los siguientes cálculos:

$$r = a.e + b.g$$

$$s = a.f + b.h$$

$$t = c.e + d.g$$

$$u = c.f + d.h$$

Total de operaciones: 8 multiplicaciones de matrices y 4 sumas de matrices que ahora son de tamaño $n/2$.

Divide & Conquer
Multiplicación de matrices

La subdivisión de matrices se hace recursivamente hasta llegar a un caso base de tamaño predeterminado b. De esta manera el tiempo de ejecución resulta:

$$T(n) = \begin{cases} 2n^3 & si\ n \le b \\ 8T\left(\frac{n}{2}\right) + n^2 & en\ otro\ caso \end{cases}$$

de donde se tiene:

$$T(n) = c_1 n^3 + c_2 n^2$$

$$T(n) = \left(2 + \frac{1}{b}\right) n^3 - n^2$$

Que ***no es una mejora*** en el orden del tiempo de ejecución con respecto al algoritmo clásico, ya que es de orden $O(n^3)$.

Divide & Conquer
Multiplicación de matrices

Hacia finales de los años 60 ***Volker Strassen*** propuso una mejora de este algoritmo (*). Este descubrimiento fue un hito en la historia de D&C, aunque el algoritmo para multiplicar enteros largos se había descubierto una década antes.

La idea básica del algoritmo de Strassen es dividir las matrices $\boldsymbol{A}$ y $\boldsymbol{B}$ en 4 secciones cada una:

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \qquad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

Para obtener:

$$C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$$

(*) Strassen, V. "Gaussian Elimination is Not Optimal." *Numerische Mathematik* **13**, 354-356, 1969

Divide & Conquer
Multiplicación de matrices

Se trabaja reduciendo el número de multiplicaciones de 8 a 7 y aumentando el número de sumas de 4 a 18 usando las fórmulas:

$$P = (A_{11} + A_{22})(B_{11} + B_{22})$$
$$Q = (A_{21} + A_{22})B_{11}$$
$$R = A_{11}(B_{12} - B_{22})$$
$$S = A_{22}(B_{21} - B_{11})$$
$$T = (A_{11} + A_{12})B_{22}$$
$$U = (A_{21} - A_{11})(B_{11} + B_{12})$$
$$V = (A_{12} - A_{22})(B_{21} + B_{22})$$
$$C_{11} = P + S - T + V$$
$$C_{12} = R + T$$
$$C_{21} = Q + S$$
$$C_{22} = P + R - Q + U$$

Divide & Conquer
Multiplicación de matrices

Por lo visto es posible multiplicar dos matrices 2x2 empleando solamente ***siete*** multiplicaciones escalares.

A primera vista, este algoritmo no parece interesante ya que ahorra multiplicaciones pero utiliza un elevado número de sumas y restas en comparación con las cuatro sumas que bastan para el algoritmo clásico.

Sin embargo, cuando se multiplican matrices grandes, es mucho más rápido sumarlas que multiplicarlas y el ahorro es considerable.

Divide & Conquer
Multiplicación de matrices

Para analizar el tiempo de ejecución de este algoritmo se supone que n es una potencia de 2:

$$T(n) = \begin{cases} 7 + 18 & si \ n = 2 \\ 7T\left(\frac{n}{2}\right) + \frac{9}{2}n^2 & si \quad n > 2 \end{cases}$$

de donde resulta según el teorema: $T(n) \in O(n^{\log_2 7})$

o también : $T(n) \in O(n^{2.81})$

En este método el número de operaciones aritméticas varía al cambiar el tamaño del caso base.

El tamaño óptimo del caso base se puede estimar teórica o experimentalmente.

Divide & Conquer
Multiplicación de matrices

Desde este descubrimiento de Strassen, un cierto número de investigadores ha intentado mejorar este tiempo de ejecución.

En 1971, Hopcroft y Kerr demostraron que esto era imposible cuando no se puede usar la conmutatividad de la multiplicación.

Pasó casi una década y Victor Pan descubrió una forma de multiplicar matrices de 70x70 empleando 143.640 multiplicaciones escalares, (él método clásico hace 343.000 multiplicaciones) que está en en $O(n^{2.61})$.

Allí comenzó la guerra de los decimales. Se fueron descubriendo sucesivamente numerosos algoritmos cada vez más eficientes asintóticamente.

Por ejemplo al final de 1979 se sabía que era posible multiplicar matrices en un tiempo que estaba en $O(n^{2.521801})$.

Divide & Conquer
Multiplicación de matrices

En 1986, Strassen bajó los costos de su algoritmo a $O(n^{2.479})$.

Uno de los algoritmo más rápido es del año 1990, debido a Coppersmith y Winograd (*) que descubrieron que en teoría es posible multiplicar matrices nxn en un tiempo en $O(n^{2.376})$.

Al momento, las publicaciones del año 2020 y 2021 de Alman, investigador del MIT (**), aseguran un algoritmo de tiempo en $O(n^{2.3728})$.

(*) Coppersmith, D. and Winograd, S. "Matrix Multiplication via Arithmetic Programming." *J. Symb. Comput.* **9**, 251-280, 1990.

(**) Josh Alman y Virginia Vassilevska Williams «A refined laser method and faster matrix multiplication», en arXiv:2010.05846 [cs.DS], 12 de octubre de 2020.

Alman, J. & Williams, V. V. A refined laser method and faster matrix multiplication. In ACM-SIAM Symposium on Discrete Algorithms 522–539 (SIAM, 2021).

Divide & Conquer
Multiplicación de matrices

Sin embargo y como consecuencia de las constantes ocultas de la notación O grande, ninguno de los algoritmos hallados después del de Strassen tiene una aplicación práctica real.

Sólo el algoritmo de Strassen es quizás el único útil, aun teniendo en cuenta que incluso él no muestra su bondad hasta valores de n muy grandes.-

La cota inferior trivial de un algoritmo que multiplica matrices es $\Omega(n^2)$ que es el costo de operar con matrices.

La pregunta es: Podrá ser alcanzado por un algoritmo ?

Divide & Conquer

Trabajo Práctico no. 4