Algoritmia y Complejidad

Tema 3. Algoritmos Divide y Vencerás.

1.- Técnica de diseño de algoritmos D y V.

- Descomponer el caso a resolver en subcasos más pequeños del mismo problema.
- Resolver independientemente cada subcaso.
- Combinar los resultados para construir la solución del caso original.

Este proceso se suele aplicar recursivamente.

La eficiencia de esta técnica depende de cómo se resuelvan los subcasos.

1.- Técnica de diseño de algoritmos D y V.

Esquema en 3 etapas:

División. Divide el problema original en *k* subproblemas de menor tamaño

Conquistar. Estos subproblemas se resuelven independientemente:

- Directamente si son simples.
- Reduciendo a casos más simples (típicamente de forma recursiva)

Combinar. Se combinan sus soluciones parciales para obtener la solución del problema original.

El caso más frecuente: $k=2$

1.- Técnica de diseño de algoritmos D y V.

```
funcion DivideYVenceras (c: tipocaso) : tiposolucion;
   var
         x1,..., xk : tipocaso;
         y1,...,yk: tiposolucion;
   si c es suficientemente simple entonces
         devolver solucion_simple(c)
   sino
         descomponer c en x1, ..., xk
         para i← 1 hasta k hacer
                  yi ← DivideYVenceras (xi)
         fpara
         devolver combinar_solucion_subcasos(y1, ..., yk)
   fsi
ffuncion
```

2.- Determinación del umbral.

- El umbral será un n_0 tal que cuando el tamaño del caso a tratar sea menor o igual, se resuelva con un algoritmo básico, es decir, no se generen más llamadas recursivas.
- La determinación del umbral óptimo, es un problema complejo.
- Dada una implementación particular, puede calcularse empíricamente.

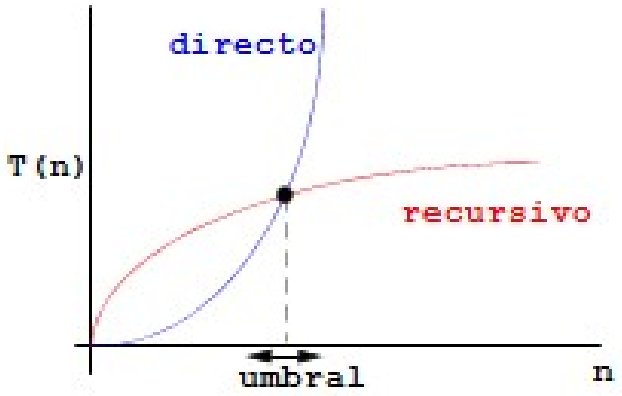
2.- Determinación del umbral.

Utilizaremos el algoritmo directo cuando el tamaño del problema sea menor que el umbral elegido

Tiene gran influencia en el tiempo de ejecución del algoritmo

- **aunque no en su ritmo de crecimiento**

El valor apropiado estará cerca del tamaño para el que el tiempo empleado por el algoritmo recursivo se iguala con el utilizado por el algoritmo directo



Puede obtenerse teóricamente o realizando medidas de tiempos

3.- Coste computacional.

Sea *n* el tamaño de nuestro caso original y sea *k* el número de subcasos:

Existe una constante *b* tal que el tamaño de los *k* subcasos es aproximadamente *n/b*

Si *g(n)* es el tiempo requerido por el algoritmo divide y vencerás en casos de tamaño *n*, *sin contar el tiempo necesario para realizar las llamadas recursivas* ⇒

$$t(n) = k*t(n/b) + g(n)$$

Donde *t(n)* es el tiempo total requerido por el algoritmo divide y vencerás, siempre que *n* sea lo suficientemente grande.

3.- Coste computacional.

Si existe un entero ***p*** tal que $g(n) \in \Theta(n^p)$, entonces podemos concluir que:

$$T(n)=\begin{cases} \Theta(n^p) & k<b^p \\ \Theta(n^p \log n) & k=b^p \\ \Theta(n^{\log_b k}) & k>b^p \end{cases}$$

n **= tamaño del problema**

k **= nº de subcasos**

n/b **= tamaño de los subcasos**

Esta solución está extraída del ejemplo 4.7.13 del libro "Fundamentos de Algoritmia" de Barssard y Bratley.

4.- Búsqueda Binaria.

Sea T[1..n] un vector ordenado tal que

$T[j] \geq T[i]$ siempre que $n \geq j \geq i \geq 1$

y sea x un elemento.

El problema consiste en buscar x en T.

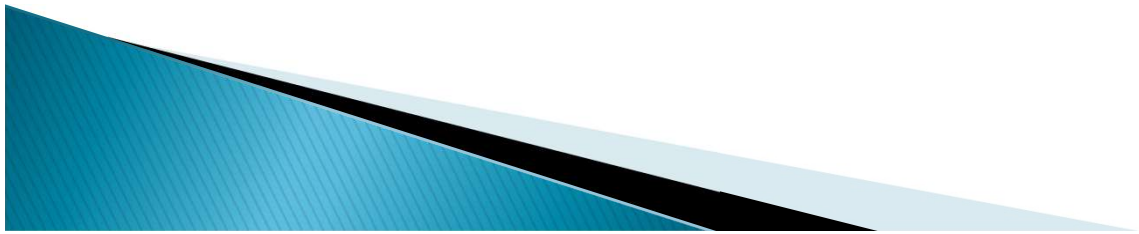
Es decir, queremos hallar un i tal que

$n \geq i \geq 1$ y $x = T[i]$, si $x \in T$.

4.- Búsqueda Binaria.

El algoritmo secuencial de búsqueda estará en el orden de O(*n*).

```
método búsquedaSecuencial(entero[1..n] t, entero x)
   desde i:=1 hasta n hacer
         si t[i] = x entonces
                  retorna i          // encontrado
         fsi
   fhacer
   retorna -1    // no encontrado
fmétodo
```



4.- Búsqueda Binaria.

Identificación del problema con el esquema

- **División**: El problema se puede descomponer en subproblemas de menor tamaño ($k = 1$).
- **Conquistar**: No hay soluciones parciales, la solución es única.
- **Combinar**: No es necesario.

4.- Búsqueda Binaria.

Se trata de una de las aplicaciones más sencillas de DyV

- **realmente no se va dividiendo el problema sino que se va reduciendo su tamaño a cada paso**
  - **algoritmos DyV denominados de reducción o simplificación**

Ejemplo: búsqueda de x=9

	1	2	3	4	5	6	7	8	9	10	11
	-5	-2	0	3	8	8	9	12	15	26	31
paso 1	i					k					j
paso 2							i		k		j
paso 3							i,k	j			

x = t[k]	x > t[k]
no	sí
no	no
sí	-

- **En cada paso (hasta que `x=t[k]` o `i>j`)**
  - **si `x > t[k]` ⇒ `i = k+1`**
  - **si `x < t[k]` ⇒ `j = k-1`**
  - **`k = (i + j) / 2`**

4.- Búsqueda Binaria.

Pasos del algoritmo de búsqueda binaria:

- Se compara x con el elemento que está en $k = (i+j)/2$
- Si T[k] $\geq$ x la búsqueda continúa en T[$i..k$]
- Si T[k] $<$ x la búsqueda continúa en T[$k+1..j$]
- Estos pasos continuarán hasta localizar x, o determinar que no está en T.

4.- Búsqueda Binaria.

```
método búsquedaBinaria(entero[1..n] t, entero i, entero j, entero x)
   // calcula el centro
   k := (i + j)/2
   // caso directo
   si i > j entonces
         retorna -1          // no encontrado
   fsi
   si t[k] = x entonces
         retorna k           // encontrado
   fsi
   // caso recursivo
   si x > t[k] entonces
         retorna búsquedaBinaria(t, k+1, j, x)
   sino
         retorna búsquedaBinaria(t, i, k-1, x)
   fsi
fmétodo
```

4.- Búsqueda Binaria.

- Como se ha visto el tiempo requerido por un algoritmo divide y vencerás es de la forma:

$$t(n) = k*t(n/b)+g(n)$$

- Para este algoritmo cada llamada genera una llamada recursiva (k=1), el tamaño del subproblema es la mitad del problema original (b=2) y sin considerar la recurrencia el resto de las operaciones son O(1), luego g(n) es $O(1)=O(n^0)$ (p=0). Estamos en el caso $k=b^p$ $(1=2^0)$, luego t(n) es $O(n^p \log n)=O(n^0 \log n)=O(\log n)$.

5.- Ordenación.

Dado un vector T[1..*n*], inicialmente desordenado, lo ordenaremos aplicando la técnica de Divide y Vencerás partiendo el vector inicial en dos subvectores más pequeños.

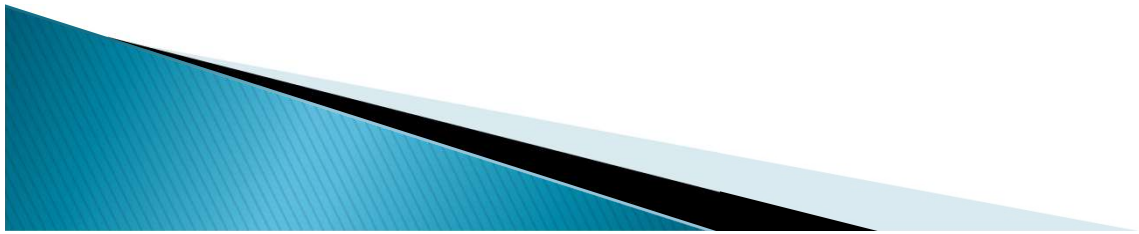
Utilizaremos dos técnicas:

- Ordenación por mezcla (mergesort)
- Ordenación rápida (quicksort)

5.- Ordenación. Mergesort.

Identificación del problema con el esquema:

- **División**: El problema se puede descomponer en subproblemas de menor tamaño ($k = 2$)
- **Conquistar**: Los subvectores se siguen dividiendo hasta alcanzar un tamaño umbral.
- **Combinar**: Dependerá del tipo de algoritmo.

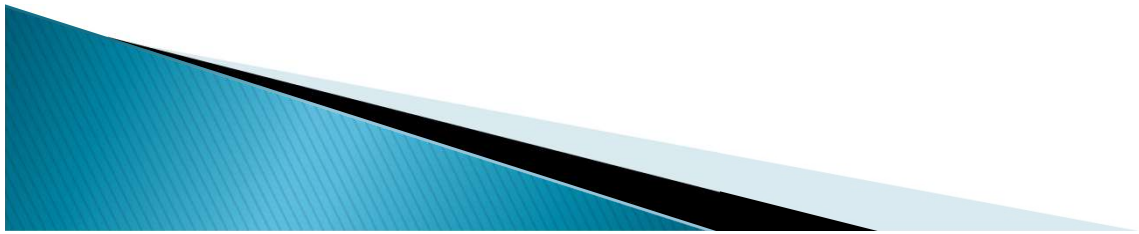


5.- Ordenación. Mergesort.

```
procedimiento fusionar(U[1..m+1],V[1..n+1],T[1..m+n])
        {Fusiona las matrices ordenadas U[1..m+1] y V[1..n+1]
        almacenándolas en T[1..m+n]. U[1..m+1] y V[1..n+1] se
        utilizan como centinelas}
        i,j:=1
        U[m+1],V[n+1]:=∞
        para k=1 hasta m+n hacer
                si U[i] <V[j] entonces
                        T[k]=U[i]
                        i=i+1
                sino
                        T[k]=V[j]
                        j=j+1
```

5.- Ordenación. Mergesort.

```
procedimiento ordenarporfusión(T[1..n])
        si n es suficientemente pequeño entonces ordenar(T)
        sino
                U[1..n/2]=T[1..n/2]
                V[1..n/2]=T[1+n/2..n]
                ordenarporfusión(U[1..n/2])
                ordenarporfusión(V[1..n/2])
                fusionar(U,V,T)
```



5.- Ordenación. Mergesort.

- Coste computacional:

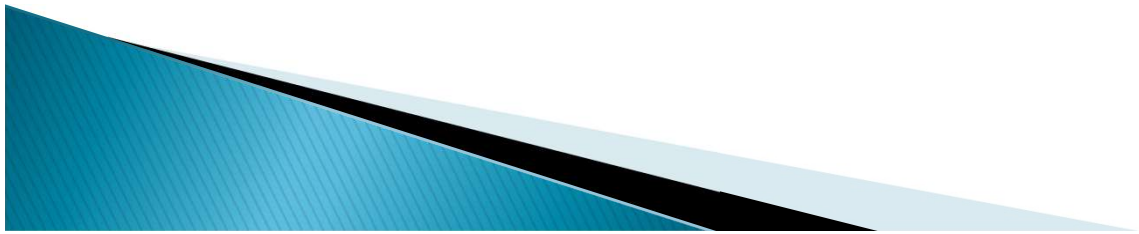
$$T(n) = \begin{cases} O(n^p) & k < b^p \\ O(n^p \log n) & k = b^p \\ O(n^{\log_b k}) & k > b^p \end{cases}$$

- En mergesort k=2 (dos subproblemas), b=2 (problemas de tamaño mitad) y p=1 (la complejidad de ordenar el caso sencillo es O(n)), entonces $k=b^p$ y, por tanto, el problema es O(nlogn).

5.- Ordenación. Quicksort.

Pasos del Algoritmo:

- Escoger un elemento de la matriz a ordenar como **pivote**.
- Se parte el vector a ambos lados del pivote. Los elementos mayores que el pivote se almacenan a la derecha del mismo y los demás a la izquierda.
- El vector se ordena mediante llamadas recursivas a este algoritmo.



5.- Ordenación. Quicksort.

```
procedimiento pivote(T[1..j],var b)
        {Permuta los elementos de la matriz T[i..j] y proporciona un valor b tal
        que, al final i≤b≤j; T[k]≤p para todo i≤k<b, T[b]=p y T[k]>p para todo
        b<k≤j en donde p es el valor inicial de T[i]}
        p=T[i]
        k=i
        b=j+1
        repetir k=k+1 hasta que T[k]>p o k≥j
        repetir b=b-1 hasta que T[b]≤p
        mientras k<b hacer
                intercambiar T[k] y T[b]
                repetir k=k+1 hasta que T[k]>p
                repetir b=b-1 hasta que T[b]≤p
        intercambiar T[i] y T[b]
```

5.- Ordenación. Quicksort.

```
procedimiento quicksort(T[i..j])
        {Ordena la submatriz T[i..j] por orden no decreciente}
        si j-i es suficientemente pequeño entonces ordenar(T[i..j])
        sino
                pivote(T[i..j],b)
                quicksort(T[i..l-1])
                quicksort(T[l+1..j])
```

6.- Potenciación de enteros.

Dados los enteros positivos *a y n*, se trata de calcular a^n.

Solución ingenua:

```
función pot0(a,n)
        r:=1
        para i:=1 hasta n hacer r:=r*a
        devuelve r
```

6.- Potenciación de enteros.

Si “r:=r*a” se considera “operación elemental”, el coste está en O(*n*).

Pero NO es así (p.e., 15^{17} no cabe en 8 bytes).

Como “r:=r*a” no es elemental, el inconveniente de esta solución es su coste (el bucle se ejecuta *n* veces). Si se quiere aplicar a un entero *a* grande (de *m* cifras), el coste es prohibitivo.

6.- Potenciación de enteros.

Sean a y n 2 enteros. Si m es el tamaño de a, la operación $a^{n,}$

- Si se utiliza el algoritmo clásico de la multiplicación el coste es polinómico: $O(m^2n^2)$

- Usando el algoritmo divide y vencerás, el coste se reduce a: $O(m^{log3}n^2)$

6.- Potenciación de enteros.

Existe un algoritmo más eficiente para la potenciación de enteros.

Por ejemplo, con dos multiplicaciones y elevar al cuadrado cuatro veces se obtiene:

$$x^{25} = \left(\left(\left(x^2 x \right)^2 \right)^2 \right)^2 x$$

Nótese que reemplazando en

$$x^{25} = \left(\left(\left(x^2 \times x \right)^2 \times 1 \right)^2 \times 1 \right)^2 \times x$$

6.- Potenciación de enteros.

```
función expoDV(a,n)
                si n=1 entonces devolver a
                si n es par entonces devolver[expoDV(a,n/2)]2
                devolver a*expoDV(a,n-1)
```

7.- Multiplicación de Matrices.

Sean A y B dos matrices de tamaño n x n, se desea calcular su producto aplicando el método de divide y vencerás.

Dos aproximaciones:

- Cuando n es potencia de 2
- Cuando n no es potencia de 2: Añadir filas y columnas de ceros hasta que lo sea.

7.- Multiplicación de Matrices cuadradas. Potencia de 2

A	1	2	3	4
1	8	4	5	3
2	3	8	4	6
3	6	1	3	3
4	7	2	7	5

X

B	1	2	3	4
1	6	2	3	6
2	2	3	9	5
3	5	4	6	3
4	4	5	2	1

=

C	1	2	3	4
1	93	63		
2	78	76		
3				
4				

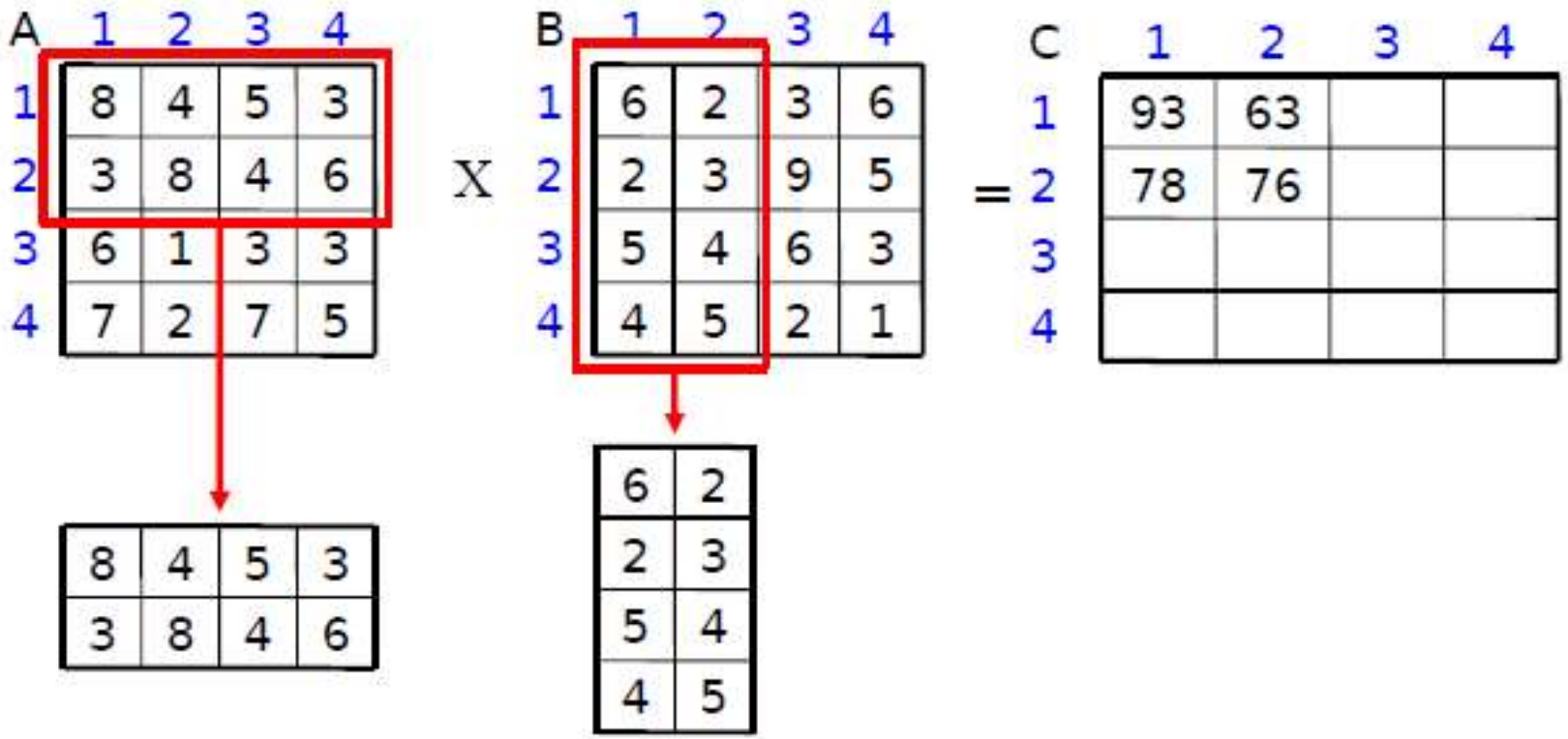
$$C_{11} = A_{11} * B_{11} + A_{12} * B_{21} + A_{13} * B_{31} + A_{14} * B_{41}$$

$$C_{12} = A_{11} * B_{12} + A_{12} * B_{22} + A_{13} * B_{32} + A_{14} * B_{42}$$

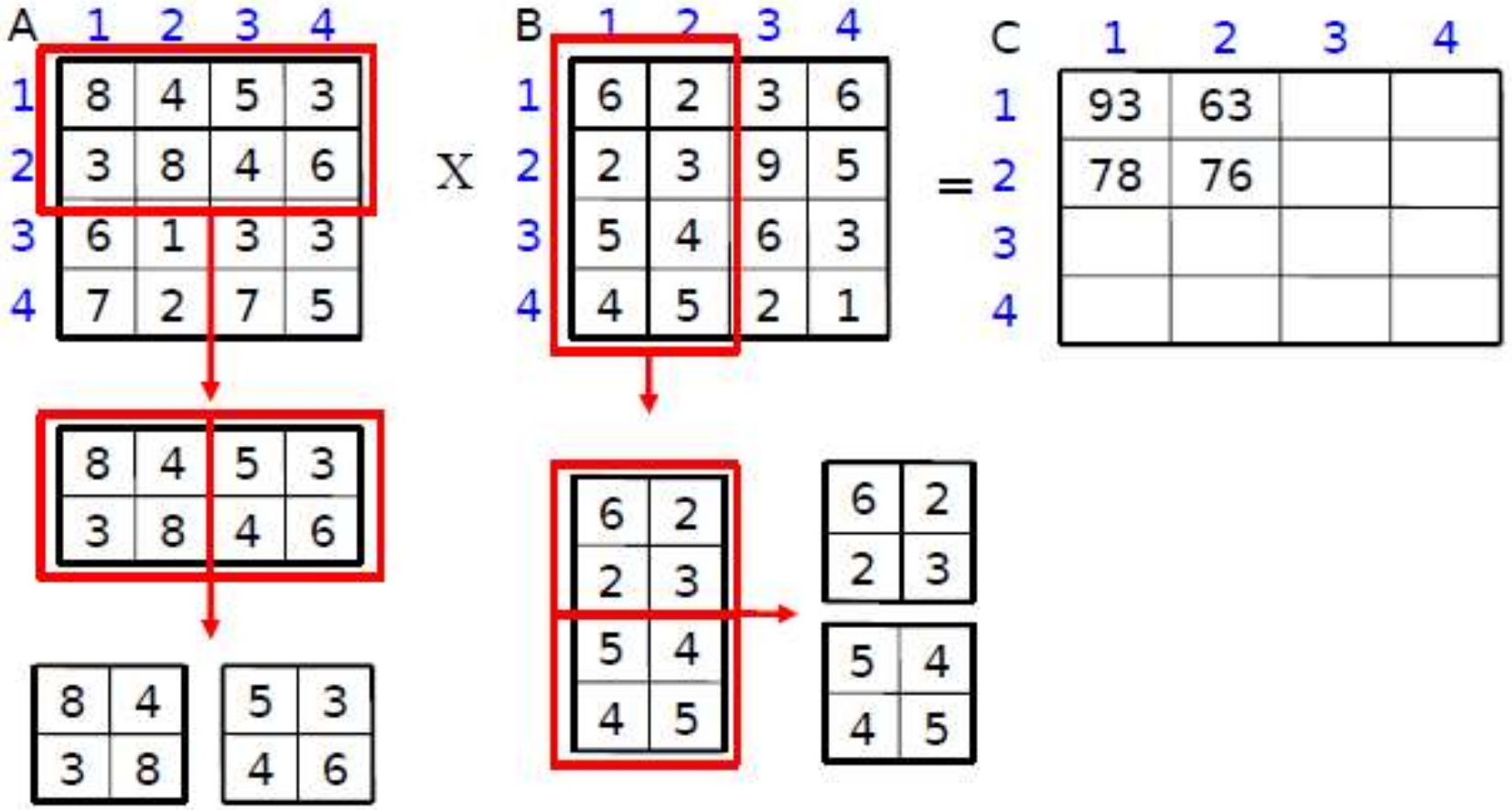
$$C_{21} = A_{21} * B_{11} + A_{22} * B_{21} + A_{23} * B_{31} + A_{24} * B_{41}$$

$$C_{22} = A_{21} * B_{12} + A_{22} * B_{22} + A_{23} * B_{32} + A_{24} * B_{42}$$

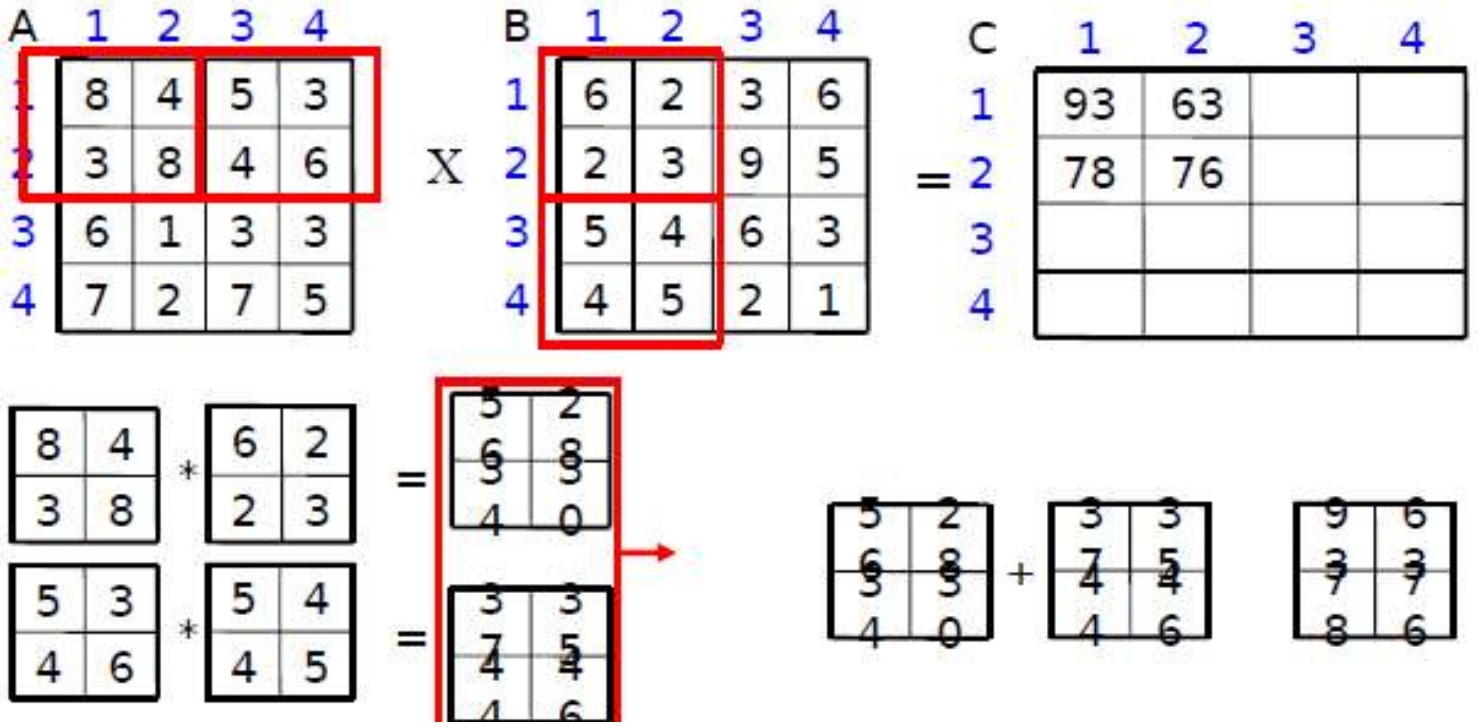
7.- Multiplicación de Matrices cuadradas. Potencia de 2



7.- Multiplicación de Matrices cuadradas. Potencia de 2



7.- Multiplicación de Matrices cuadradas. Potencia de 2



7.- Multiplicación de Matrices cuadradas. Potencia de 2

A1		A2	
8	4	5	3
3	8	4	6
6	1	3	3
7	2	7	5
A3		A4	

B1		B2	
6	2	3	6
2	3	9	5
5	4	6	3
4	5	2	1
B3		B4	

C1		C2	
93	63		
78	76		
C3		C4	

$$C1 = A1*B1 + A2*B3$$

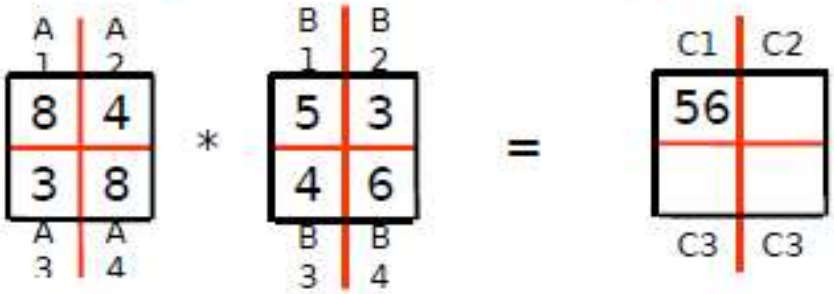
$$\begin{bmatrix} 8 & 4 \\ 3 & 8 \end{bmatrix} * \begin{bmatrix} 6 & 2 \\ 2 & 3 \end{bmatrix} + \begin{bmatrix} 5 & 3 \\ 4 & 6 \end{bmatrix} * \begin{bmatrix} 5 & 4 \\ 4 & 5 \end{bmatrix} = \begin{bmatrix} 93 & 63 \\ 78 & 76 \end{bmatrix}$$

7.- Multiplicación de Matrices cuadradas. Potencia de 2

A1		A2	
8	4	5	3
3	8	4	6
6	1	3	3
7	2	7	5
A3		A4	

B1		B2	
6	2	3	6
2	3	9	5
5	4	6	3
4	5	2	1
B3		B4	

C1		C2	
93	63		
78	76		
C3		C4	



C1 = A1*B1 + A2*B3 → 8 * 5 + 4 * 4 = 56

7.- Multiplicación de Matrices cuadradas. Potencia de 2

Identificación del problema con el esquema:

- **División**: El problema se puede descomponer en subproblemas de menor tamaño (*k = 4).*
- **Conquistar**: Los submatrices se siguen dividiendo hasta alcanzar un tamaño 1.
- **Combinar**: sumar los resultados intermedios.

7.- Multiplicación de Matrices cuadradas. Potencia de 2

```
procedimiento multiplica (A,B: TipoMatriz; VAR C: TipoMatriz);
Var  A11,A12,A21,A22,B11,B12,B21,B22,C11,C12,C21,C22,CAux:  TipoMatriz;

   si tamaño(A) = 1 entonces
    C ← A*B
   sino
    DividirEnCuadrantes(A,A11,A12,A21,A22);
    DividirEnCuadrantes(B,B11,B12,B21,B22);
    para i ← 1 hasta 2 hacer
        para j ← 1 hasta 2 hacer
            Inicializa(Cij)
            para k ← 1 hasta 2 hacer
                multiplica(Aik,Bkj,CAux);
                Suma(Cij,CAux)
            fpara
        fpara
    fpara
    ColocarCuadrantes(C,C11,C12,C21,C22);
   fsi
fprocedimiento
```