# TEMA 3. ALGORITMOS DIVIDE Y VENCERÁS

## 1. Técnica de diseño de algoritmos divide y vencerás.

En los algoritmos de divide y vencerás se trata de descomponer el caso a resolver en subcasos más pequeños del mismo problema. Posteriormente se resuelve independientemente cada subcaso y se combinan los resultados para construir la solución el caso original. Este proceso se suele aplicar recursivamente y la eficiencia de esta técnica depende de cómo se resuelvan los subcasos. Se trata, por tanto, de un esquema en 3 etapas:

1) Dividir. Divide el problema original en k subproblemas de menor tamaño
2) Conquistar. Estos subproblemas se resuelven independientemente:
   - Directamente si son simples.
   - Reduciendo a casos más simples (típicamente de forma recursiva)
3) Combinar. Se combinan sus soluciones parciales para obtener la solución del problema original.

El caso más frecuente corresponde con k=2, es decir, se divide el problema en la mitad.

El seudocódigo es como sigue:

```
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
```

Para que el enfoque divide y vencerás merezca la pena, se deben cumplir las siguientes condiciones:

- Debe ser posible descomponer el caso a resolver en subcasos.

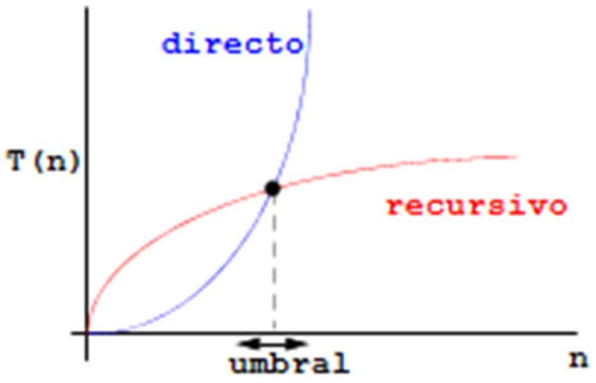
- La formulación recursiva nunca resuelva el mismo subproblema más de una vez.
- Se debe poder componer la solución a partir de las soluciones de los subcasos de un modo eficiente.
- Los subejemplares deben ser, aproximadamente, del mismo tamaño.

## 2. Determinación del umbral.

Cuando el problema sea lo suficientemente pequeño no se realizan más divisiones del mismo. Este límite lo marca el 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. También puede emplearse una técnica híbrida para su cálculo, que consiste en lo siguiente:

1) Obtener las ecuaciones de recurrencia del algoritmo (estudio teórico).
2) Determinar empíricamente los valores de las constantes que se utilizan en dichas ecuaciones para la implementación concreta que estamos empleando.
3) El umbral óptimo se estima hallando el tamaño n del caso para el cual no hay diferencia entre aplicar directamente el algoritmo clásico o pasar a un nivel más de recursión.

En todo caso, se utilizará el algoritmo directo cuando el tamaño del problema sea menor que el umbral elegido.




## 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, entonces:

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

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

Si existe un entero p tal que $g(n) \in O(n^p)$, se puede concluir que:

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

## 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 $x \in T$.

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
```

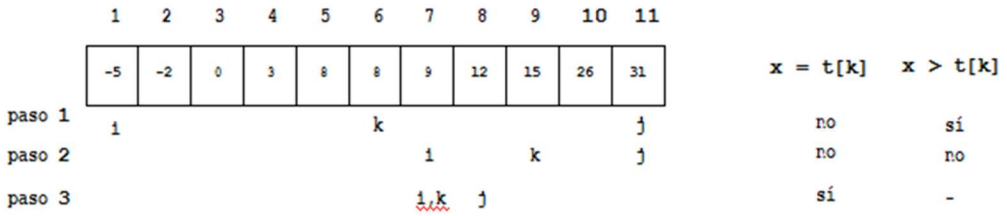
Identificación del problema con el esquema divide y vencerás:

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

Se trata de una de las aplicaciones más sencillas de divide y vencerás. Realmente no se va dividiendo el problema, sino que se va reduciendo su

tamaño en cada paso. Es un algoritmo divide y vencerás de reducción o simplificación.

Ejemplo: búsqueda de x=9.




Pasos del algoritmo de búsqueda binaria:

1) Se compara x con el elemento que está en k= (i+j)/2
2) Si T[k]≥x la búsqueda continúa en T[i..k]
3) 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. El pseudocódigo sería:

```
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
```

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, se ordenará aplicando la técnica de Divide y Vencerás partiendo el vector inicial en dos subvectores más pequeños. Se utilizarán dos técnicas:

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

### 5.1. Ordenación utilizando mergesort.

Pasos del algoritmo:

1. Se divide el vector en dos mitades
2. Se ordenan esas dos mitades recursivamente
3. Se fusionan en un solo vector ordenado.

Para poder aplicar este algoritmo necesitaríamos un procedimiento eficiente para fusionar dos matrices ordenadas que sería como sigue:

```
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
```

Una vez tenemos este procedimiento el algoritmo de ordenar por fusión será:

```
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)
```

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).

## 5.2. Ordenación utilizando quicksort.

En este caso se escoge un elemento de la matriz a ordenar como pivote y se divide 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. A la hora de seleccionar el pivote cualquier elemento es válido, por lo que lo normal es escoger el primer elemento del vector, aunque el pivote óptimo es la mediana de los elementos del vector.

En el algoritmo quicksort una parte importante es colocar el pivote en la posición adecuada para lo cual se utiliza este procedimiento:

```
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]
```

Con este procedimiento el algoritmo de ordenación quedaría:

```
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])
```

Para calcular el coste se asume que los subproblemas están equilibrados y, en ese caso, el coste es O(nlogn).

## 6. Potenciación de enteros.

Se trata una vez dados los enteros positivos a y n, se trata de calcular $a^n$. Una solución ingenua sería:

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

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: $O(m^2n^2)$, mientras que utilizando algoritmo divide y vencerás, el coste se reduce a: $O(m^{log3}n^2)$.

Sin embargo, existe un algoritmo más eficiente para la potenciación de enteros, considerando que:

$$x^{25} = (((x^2x)^2)^2)^2x$$

Teniendo en cuenta esto se puede escribir un algoritmo de potenciación de tipo divide y vencerás:

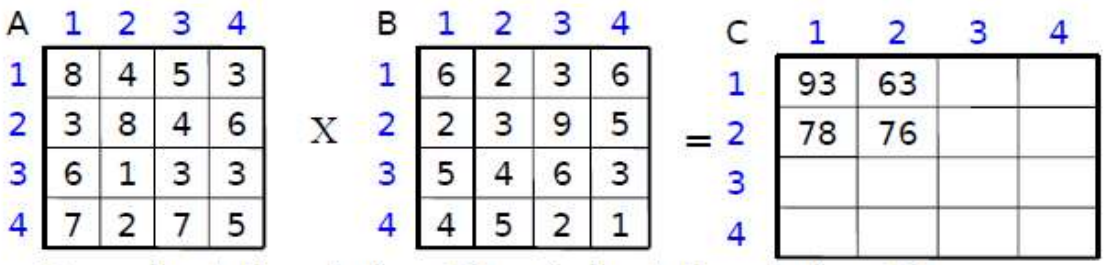
función expoDV(a,n)
si n=1 entonces devolver a
si n es par entonces devolver[expoDV(a,n/2)]$^2$
devolver a*expoDV(a,n-1)

## 7. Multiplicación de matrices.

Sean A y B dos matrices de tamaño nxn, se desea calcular su producto aplicando el método de divide y vencerás. Dos posibilidades:

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

Ejemplo:

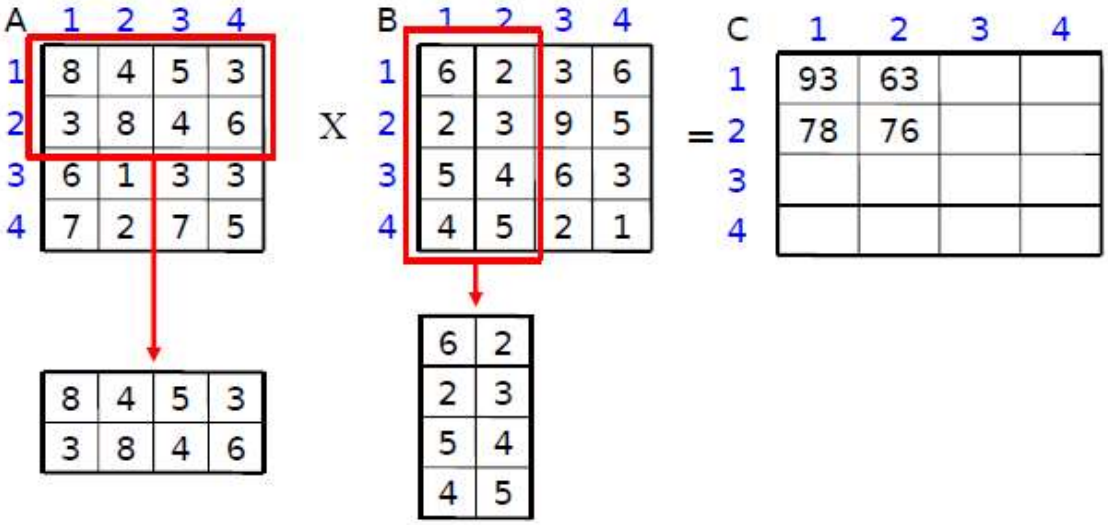


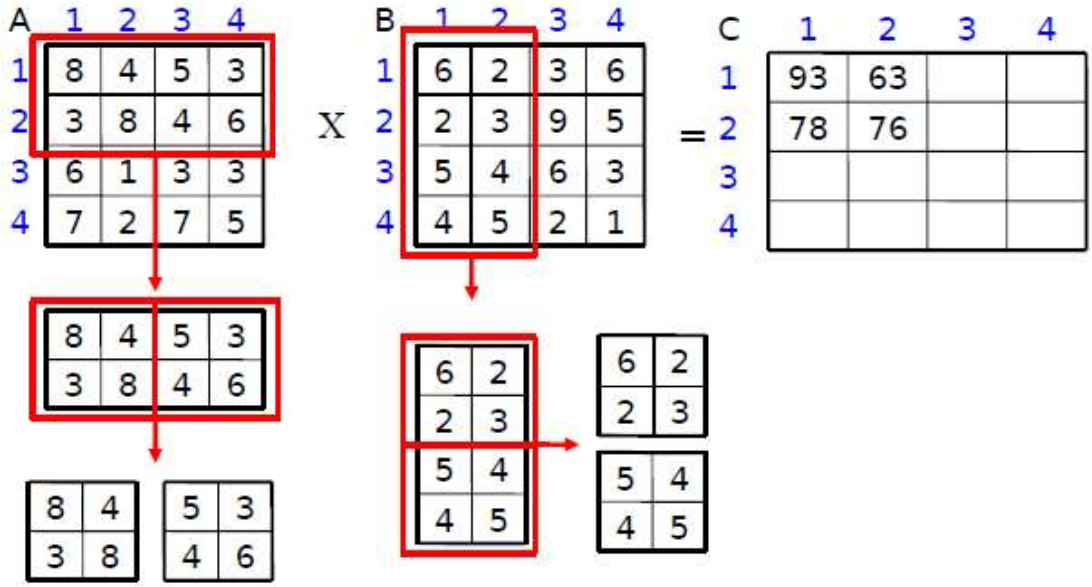

$$C_{11} = A_{11} * B_{11} + A_{12} * B_{21} + A_{13} * B_{31} + A_{14} * B_{41}$$

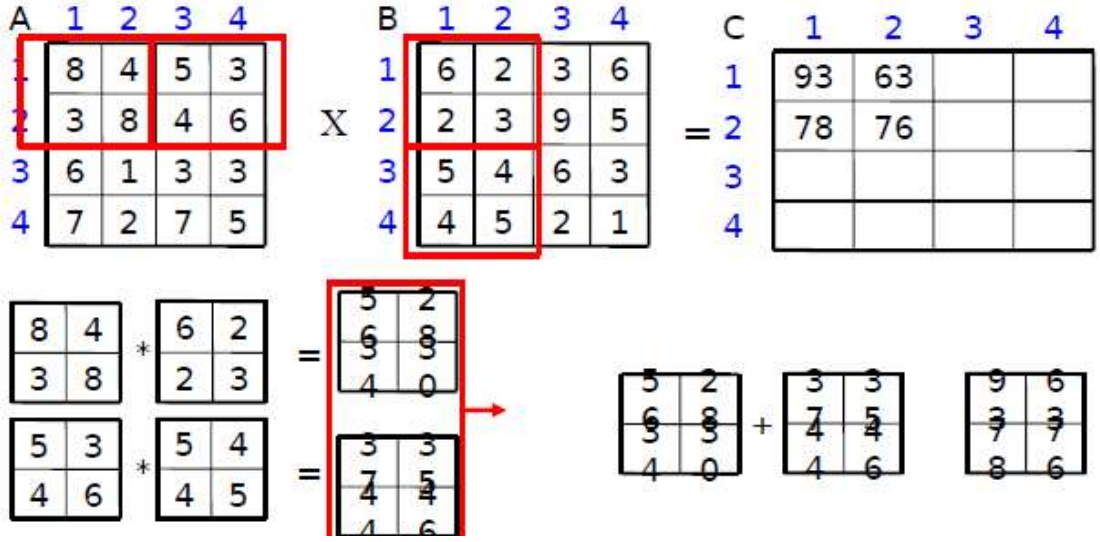
$$C_{12} = A_{11} * B_{12} + A_{12} * B_{22} + A_{13} * B_{32} + A_{14} * B_{42}$$

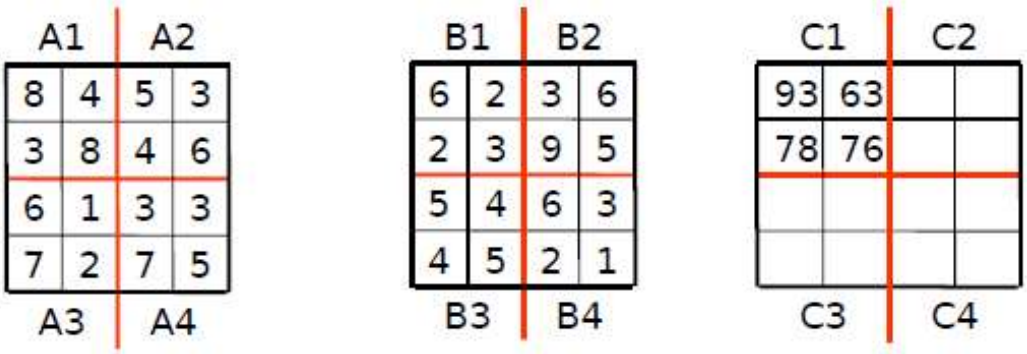
$$C_{21} = A_{21} * B_{11} + A_{22} * B_{21} + A_{23} * B_{31} + A_{24} * B_{41}$$

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


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
8 4 5 3
3 8 4 6
6 2
2 3
5 4
4 5

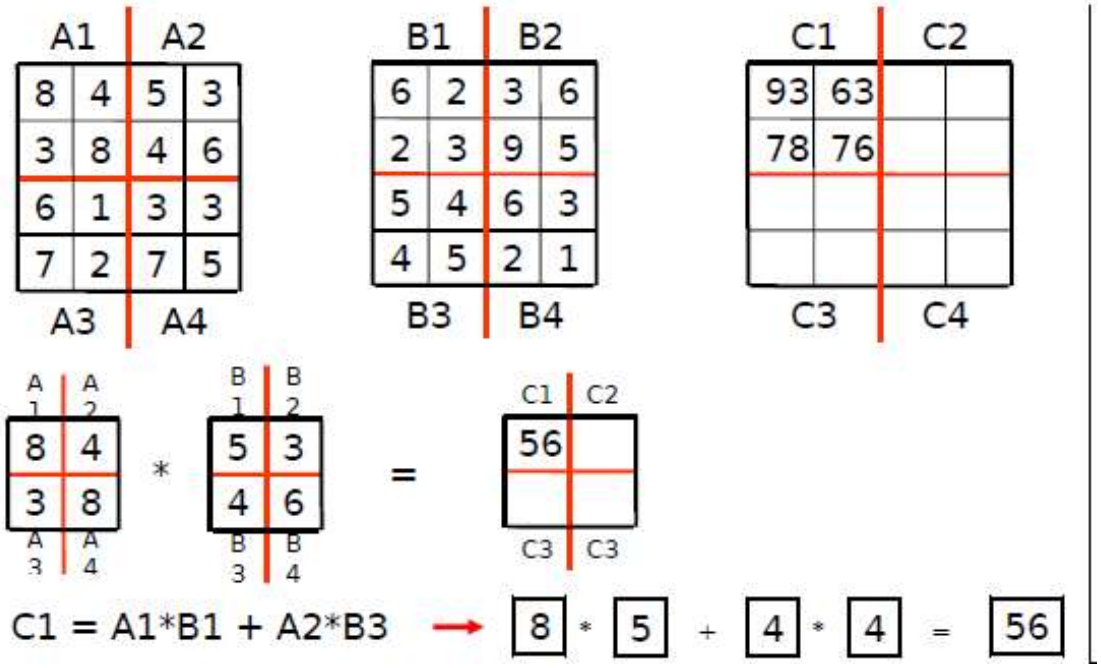

A 1 2 3 4
1 8 4 5 3
2 3 8 4 6
3 6 1 3 3
4 7 2 7 5
X
B 1 2 3 4
1 6 2 3 6
2 2 3 9 5
3 5 4 6 3
4 4 5 2 1
= C 1 2 3 4
1 93 63
2 78 76
3
4
8 4 5 3
3 8 4 6
8 4
3 8
5 3
4 6
6 2
2 3
5 4
4 5


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
8 4
3 8
* 6 2
2 3
=
5 3
4 6
* 5 4
4 5
=
+




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

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




Identificación del problema con el esquema divide y vencerás:

- 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

El pseudocódigo sería:

```
procedimiento multiplica(A,B:tipomatriz; var C:tipomatriz)
var A11,A12,A21,A22, B11,B12,B21,B22, C11,C12,C21,C22:
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
```