Diseño y Análisis de Algoritmos

DDDIVETEMA 4. DIVIDE Y VENCERÁS

Universidad
Rey Juan Carlos

Tema 2. Divide y vencerás

Introducción

- Motivación. Multiplicación de enteros grandes
- Esquema de la técnica
- Aspectos del diseño
- Aplicaciones de Divide y Vencerás
  - Búsqueda binaria
  - Ordenación
  - Multiplicación de matrices cuadradas

Tema 2. Divide y vencerás

Motivación

- Multiplicación de enteros de *n* cifras. Algoritmo clásico

$1234*5678 = 1234*(5*10^3+6*10^2+7*10^1+8*10^0) =$

$1234*5*10^3+1234*6*10^2+1234*7*10^1+1234*8*10^0$

- Operaciones básicas
  - Multiplicaciones de dígitos sencillas: O(1)
  - Sumas de dígitos sencillas: O(1)
  - Desplazamientos: O(1)

  Eficiencia del algoritmo: $O(n^2)$

Tema 2. Divide y vencerás

Motivación

- Multiplicación de enteros de *n* cifras. Algoritmo Divide y Vencerás

$$1234 = 12 * 10^2 + 34$$

$$5678 = 56 * 10^2 + 78$$

$$1234*5678 = (12 * 10^2 + 34)*(56 * 10^2 + 78) =$$

$$(12*56)*10^4+(12*78+34*56)*10^2+(34*78)*10^0$$

- Se reduce **una** multiplicación de 4 cifras a **cuatro** multiplicaciones de 2 cifras, **tres** sumas y **dos** desplazamientos

Motivación

- Multiplicación de enteros de *n* cifras. Algoritmo Divide y Vencerás. **Versión sencilla**
- **Dividir**

$X=12345678 = x_i * 10^4 + x_d, \quad x_i = 1234, x_d = 5678$

$Y=24680135 = y_i * 10^4 + y_d, \quad y_i = 2468, y_d = 0135$

- **Combinar**

$X*Y=(x_i * 10^4 + x_d)*(y_i * 10^4 + y_d) =$

$= (x_i * y_i)*10^8 + (x_i*y_d + x_d*y_i)*10^4 + (x_d * y_d)*10^0$

Motivación

- Multiplicación de enteros de *n* cifras. Algoritmo Divide y Vencerás. **Versión sencilla**
- **En general**

$X = x_i * 10^{n/2} + x_d$

$Y = y_i * 10^{n/2} + y_d$

Combinar

$X*Y=(x_i * 10^{n/2} + x_d)*(y_i * 10^{n/2} + y_d) =$

$= (x_i * y_i)*10^n + (x_i*y_d + x_d*y_i)*10^{n/2} + (x_d * y_d)*10^0$

Tema 2. Divide y vencerás

Esquema de la técnica

```
funcion multiplica (X,Y,n: Entero) : Entero;
  si esSuficientementeSimple(n) entonces
    devolver X*Y
  sino
    {xi, xd} ← obtener(X);              //DIVIDIR
    {yi, yd} ← obtener(Y);
    z1 = multiplica(xi,yi,n/2);
    z2 = multiplica(xi,yd,n/2);
    z3 = multiplica(xd,yi,n/2);
    z4 = multiplica(xd,yd,n/2);
    aux = suma(z2,z3);                  //COMBINAR
    z1 = desplazaIzda(z1,n)
    aux = desplazaIzda(aux,n/2)
    z = suma(z1,aux);
    z = suma(z,z4);
    devolver z
  fsi
ffuncion
```

Tema 2. Divide y vencerás

Esquema de la técnica

```
funcion multiplica (X,Y,n: Entero) : Entero;        EFICIENCIA
  si esSuficientementeSimple(n) entonces            O(1)
    devolver X*Y O(1)
  sino
    {xi, xd} ← obtener(X);                           O(n)
    {yi, yd} ← obtener(Y);                           O(n)
    z1 = multiplica(xi,yi,n/2);                      T(n/2)
    z2 = multiplica(xi,yd,n/2);                      T(n/2)
    z3 = multiplica(xd,yi,n/2);                      T(n/2)
    z4 = multiplica(xd,yd,n/2);                      T(n/2)
    aux = suma(z2,z3);                               O(n)
    z1 = desplazaIzda(z1,n);                         O(n)
    aux = desplazaIzda(aux,n/2);                     O(n)
    z = suma(z1,aux);                                O(n)
    z = suma(z,z4);                                  O(n)
   devolver z;                                       O(1)
  fsi
ffuncion
```

Tema 2. Divide y vencerás

Motivación

- Multiplicación de enteros de *n* cifras. Algoritmo Divide y Vencerás. **Versión sencilla**

$$T(n) = 4T(n/2) + n \Rightarrow O(n^2)$$

(fórmula de cálculo de complejidad para algoritmos recursivos)

- **Cuello de botella.** Número de multiplicaciones de tamaño *n*/2
- **Mejorar la eficiencia.** Reducir el número de multiplicaciones

Esquema de la técnica

- Eficiencia de los algoritmos divide y vencerás

$$T(n) = aT\left(\frac{n}{b}\right) + g(n) \text{ con } g(n) \in O\left(n^k\right), y$$
$$a \geq 1, b \geq 2, k \geq 0, c > 0$$

$$T(n) = \begin{cases} O\left(n^k\right), & a < b^k \\ O\left(n^k \log_b n\right) & a = b^k \\ O\left(n^{\log_b a}\right) & a > b^k \end{cases}$$

- *a*: sub-problemas generados
- *b*: divisor del problema
- *k*: complejidad del método de combinación

Tema 2. Divide y vencerás

Motivación

- **Multiplicación** de enteros de *n* cifras. Algoritmo **Divide y Vencerás**

$r = (x_i+x_d)*(y_i+y_d)= x_i*y_i + (x_i*y_d + x_d*y_i) + x_d*y_d$

$p = x_i*y_i \qquad q = x_d*y_d$

$X*Y=(x_i* y_i)*10^n + (x_i*y_d + x_d*y_i)*10^{n/2} + (x_d* y_d)*10^0$

$X*Y= p*10^n + (r-p-q)*10^{n/2} + q*10^0$

- Reducción a **tres** multiplicaciones de tamaño *n*/2

Tema 2. Divide y vencerás

Esquema de la técnica

```
funcion multiplicaDV(X,Y,n: Entero): Entero;
  si esSuficientementeSimple(n) entonces
    devolver X*Y
  sino
     {xi, xd} ← obtener(X);              //DIVIDIR
    {yi, yd} ← obtener(Y);
    s1 = suma(xi,xd);
    s2 = suma(yi,yd);
    p = multiplicaDV(xi,yi,n/2);
    q = multiplicaDV(xd,yd,n/2);
    r = multiplicaDV(s1,s2,n/2);
    aux = suma(r,-p,-q);                 //COMBINAR
    aux = desplazaIzda(aux,n/2)
    p = desplazaIzda(p,n)
    z = suma(p,aux,q);
     devolver z
  fsi
ffuncion
```

Tema 2. Divide y vencerás

Esquema de la técnica

```
funcion multiplicaDV(X,Y,n: Entero) : Entero;      EFICIENCIA
  si esSuficientementeSimple(n) entonces           O(1)
    devolver X*Y                                   O(1)
  sino
    {xi, xd} ← obtener(X);                          O(n)
    {yi, yd} ← obtener(Y);                          O(n)
    s1 = suma(xi,xd);                               O(n)
    s2 = suma(yi,yd) ;                              O(n)
    p = multiplicaDV(xi,yi,n/2);                    T(n/2)
    q = multiplicaDV(xd,yd,n/2);                    T(n/2)
    r = multiplicaDV(s1,s2,n/2);                    T(n/2)
    aux = suma(r,-p,-q);                            O(n)
    aux = desplazaIzda(aux,n/2);                    O(n)
    p = desplazaIzda(p,n);                          O(n)
    z = suma(p,aux,q);                              O(n)
    devolver z;                                    O(1)
  fsi
ffuncion
```

Tema 2. Divide y vencerás

Motivación

- Multiplicación de enteros de *n* cifras. Algoritmo Divide y Vencerás. **Versión sencilla**

$$T(n) = 3T(n/2) + n \Rightarrow O(n^{\log_2 3}) = O(n^{1.585})$$

(fórmula de cálculo de complejidad para algoritmos recursivos)

Tamaño	Básico n^2	DyV $n^{1.585}$
n = 10	0.1 ms	0.04 ms
n = 100	10 ms	1.48ms
n = 1000	1 s	57 ms
n = 10000	100 s	2 s

Tema 2. Divide y vencerás

Esquema de la técnica

- La técnica de DyV consiste en:
  - **Descomponer** el caso a resolver en sub-casos más pequeños del mismo problema
  - **Resolver** independientemente cada sub-caso
  - **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 sub-casos

Tema 2. Divide y vencerás

Esquema de la técnica

Esquema en 3 etapas:

- **División**. Divide el problema original en *k* sub-problemas de menor tamaño
- **Conquistar**. Estos sub-problemas 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

Tema 2. Divide y vencerás

Esquema de la técnica

```
funcion divideYVenceras (c: TipoCaso): TipoSolucion;
  var
    x1,..., xk : TipoCaso;
    y1,...,yk: TipoSolucion;
  si esSuficientementeSimple(c) entonces
    devolver solucionSimple(c)
  sino
    {x1, ..., xk} ← descomponer(c)
    para i ← 1 hasta k hacer
      yi ← divideYVenceras (xi)
    fpara
    devolver combinarSolucionSubcasos(y1, ..., yk)
  fsi
ffuncion
```

Esquema de la técnica

- Eficiencia de los algoritmos divide y vencerás

$$T(n) = aT\left(\frac{n}{b}\right) + g(n) \text{ con } g(n) \in O\left(n^k\right), y$$
$$a \geq 1, b \geq 2, k \geq 0, c > 0$$

$$T(n) = \begin{cases} \Theta\left(n^k\right), & a < b^k \\ \Theta\left(n^k \log_b n\right) & a = b^k \\ n^{\log_b a} & a > b^k \end{cases}$$

- *a*: sub-problemas generados
- *b*: divisor del problema
- *k*: complejidad del método de combinación

Tema 2. Divide y vencerás

Aspectos del diseño

- El número de sub-ejemplares, *k*, suele ser **pequeño** e **independiente** del caso particular a resolver
- Cuando $k = 1$, el esquema divide y vencerás pasa a llamarse **reducción** o **simplificación**
- Algunos algoritmos de DyV pueden necesitar que el **primer** sub-ejemplar esté **resuelto** antes de formular el segundo sub-ejemplar.

Aspectos del diseño

Para que el enfoque DyV se pueda aplicar, se deben **cumplir** las siguientes **condiciones**:

- Debe ser posible **descomponer** el caso a resolver en sub-casos.
- Se debe poder **componer** la solución a partir de las soluciones de los sub-casos de un modo **eficiente**
- Los sub-ejemplares deben ser, aproximadamente, del mismo **tamaño**.

Aspectos relevantes para el diseño

- Algoritmo recursivo: **división** en sub-problemas y **combinación** eficiente de soluciones parciales
- Algoritmo específico: para resolver problemas de **tamaño reducido**
- Determinación del umbral: ¿**cuándo** parar la descomposición recursiva y aplicar el algoritmo específico?

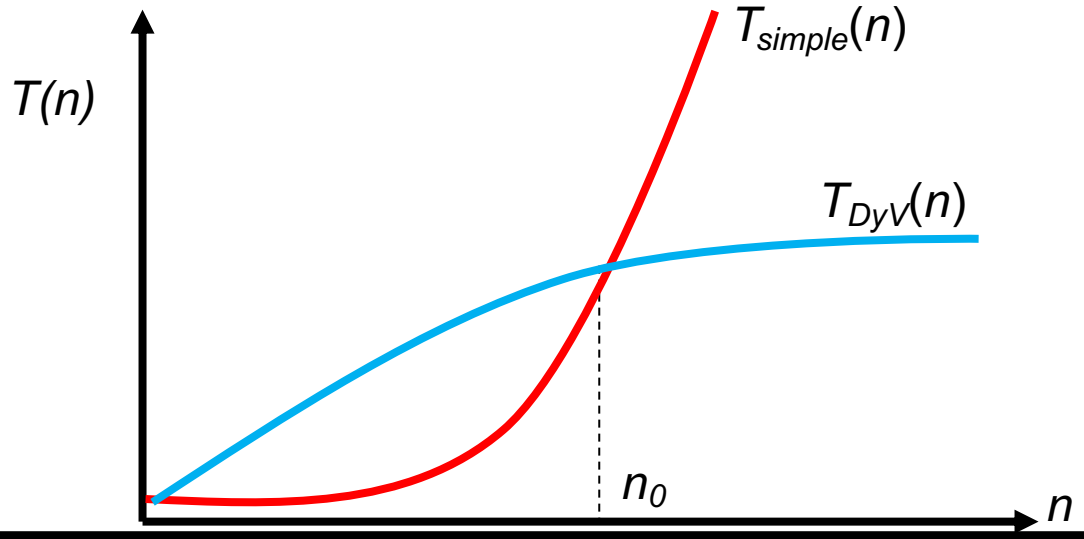
Aspectos del diseño

¿Cómo se determina el umbral?

- Depende de la **implementación**
- No hay a priori valor **máximo** (solo se aplica el método simple) ni **mínimo** (en tamaños pequeños suele ser más eficiente el método simple)
- La determinación del umbral óptimo, es un problema **complejo** (depende de la implementación)

Método experimental

- **Implementar** el algoritmo simple y el DyV
- **Resolver** el problema para distintos valores de *n*
- Presumiblemente a valores **altos** de *n*, el DyV será más **rápido** que el simple



Método teórico

- Determina el punto de cruce entre el algoritmo simple y el DyV

$T(n) = h(n)$ si $n \leq n_0$

$T(n) = aT(n/b) + g(n)$ si $n > n_0$

- El umbral *óptimo* será cuando se igualen

$h(n_0) = ah(n_0/b) + g(n_0)$ Despejar n_0

Método híbrido

- Obtener las **ecuaciones de recurrencia** del algoritmo (estudio teórico).
- Determinar **empíricamente** los valores de las constantes ocultas
- Probar valores alrededor del umbral **óptimo**

Ejemplo: multiplicación de enteros grandes

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

$g(n) = 16n$

$h(n) = n^2$

Si tomamos n = 1024

$n_0 = 1 \Rightarrow$ t(n) =31m45s (división hasta tamaño 1)

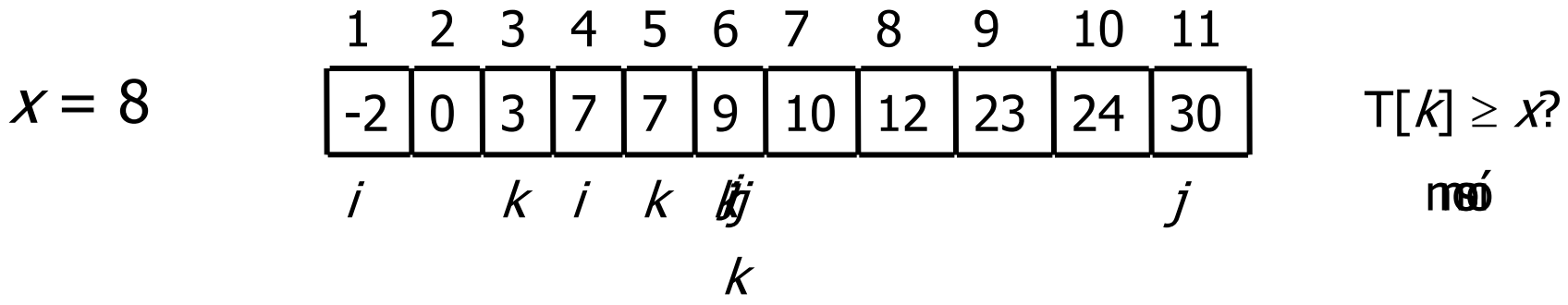
$n_0 = \infty \Rightarrow$ t(n) =17m29s (si no se divide)

$h(n_0)= 3h(n_0/2)+g(n_0)$

$n_0^2 = (3/4)n_0^2 + 16\, n_0 \Rightarrow n_0 = 64 \Rightarrow$ t(n) =7m44s

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 el vector 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*)
- Aprovechar la propiedad de que el vector está **ordenado**



Identificación del **problema** con el **esquema**

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

Pasos del algoritmo de búsqueda binaria :

- Se compara x con el elemento que se encuentra en la posición $k = (i+j)/2$
- Si T[k] ≥ x la búsqueda continúa en T[*i*..*k*]
- Si T[k] < x la búsqueda continúa en T[*k*+i..*j*]
- Los pasos 1 y 2 continuarán hasta:
  - Localizar *x*, o alternativamente,
  - Determinar que no está en T

Tema 2. Divide y vencerás

Aplicaciones de DyV. Búsqueda binaria

```
funcion busquedabin(T[1..n],x): Boolean
{Búsq. Bin. de x en T[i..j] con T[i-1] < x <= T[j]}
   devolver binrec(T[1..n],x)
ffuncion

funcion binrec(T[i..j],x)
  si i > j entonces
    devolver false
  sino
    k ← (i + j)  ÷ 2
    si x = T[k] entonces
      devolver true
    si x <= T[k] entonces
      devolver binrec(T[i..k],x)
    sino
      devolver binrec(T[k + 1..j],x)
    fsi
  fsi
ffuncion
```

- **Cálculo de la eficiencia**

`binrec` $g(n) \in O(1) = O(n^0), p = 0$

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

- No combinamos soluciones: hay 1 caso $\Rightarrow$ k = 1
- En cada paso dividimos el tamaño del problema a la mitad $\Rightarrow$ b = 2

$$T(n) \in \Theta(n^p \log n) \Rightarrow T(n) \in \Theta(\log n)$$

Aplicaciones de DyV. Ordenación de vectores

Dado un vector T[1..n], inicialmente desordenado, lo ordenaremos aplicando la técnica de DyV partiendo el vector inicial en dos sub-vectores más pequeños.

- Utilizaremos dos técnicas:
  - Ordenación por mezcla (*mergesort*)
  - Ordenación rápida (*quicksort*)

Identificación del problema con el esquema

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

Pasos del Algoritmo:

- **Dividir** el vector en dos mitades
- **Ordenar** esas dos mitades recursivamente
- **Fusionarlas** en un solo vector ordenado.

Tema 2. Divide y vencerás

Aplicaciones de DyV. Ordenación de vectores

1	2	3	4	5	6	7	8	9	10	11	12
3	1	4	3	7	9	2	6	6	7	8	8

1	2	3	4	5	6
3	1	4	4	7	9

7	8	9	10	11	12
2	6	5	5	6	8

1	2	3
3	3	4

4	5	6
1	7	9

7	8	9
2	6	6

10	11	12
3	5	8

1
3

2	3
1	4

4
1

5	6
7	9

7
2

8	9
6	6

10
3

11	12
5	8

2	3
1	4

5	6
7	9

8	9
6	5

11	12
5	8

Tema 2. Divide y vencerás

Aplicaciones de DyV. Ordenación de vectores

```
procedimiento fusionar(U[1..m+1], V[1..n+1], T[1..m+n])
{Fusionar vectores ordenados U[1..m] y V[1..n] almacenándolas
   en T[1..m+n],U[m+1] y V[n+1] se utilizan como centinelas}
   i ← 1, 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
    fsi
   fpara
fprocedimiento
```

Tema 2. Divide y vencerás

Aplicaciones de DyV. Ordenación de vectores

```
procedimiento ordenarporfusión (T[1..n])
   si esSuficientementePequeño(n) entonces
    ordenar(T) {P. ej. Ord. por inserción}
   sino
    vector U[1..1 + ⌊n/2⌋], V[1..1 + ⌈n/2⌉],
    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)
   fsi
fprocedimiento
```

Cálculo de la eficiencia

`mergesort` $g(n) \in \Theta(n) = \Theta(n^1), p = 1$

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

- Dos sub-problemas $\Rightarrow k = 2$
- En cada paso dividimos el tamaño del problema a la mitad $\Rightarrow$ $b = 2$

$$k = b^p \Rightarrow 2 = 2^1$$

$$T(n) \in \Theta(n^p \log n) \Rightarrow T(n) \in \Theta(n \log n)$$

Pasos del Algoritmo:

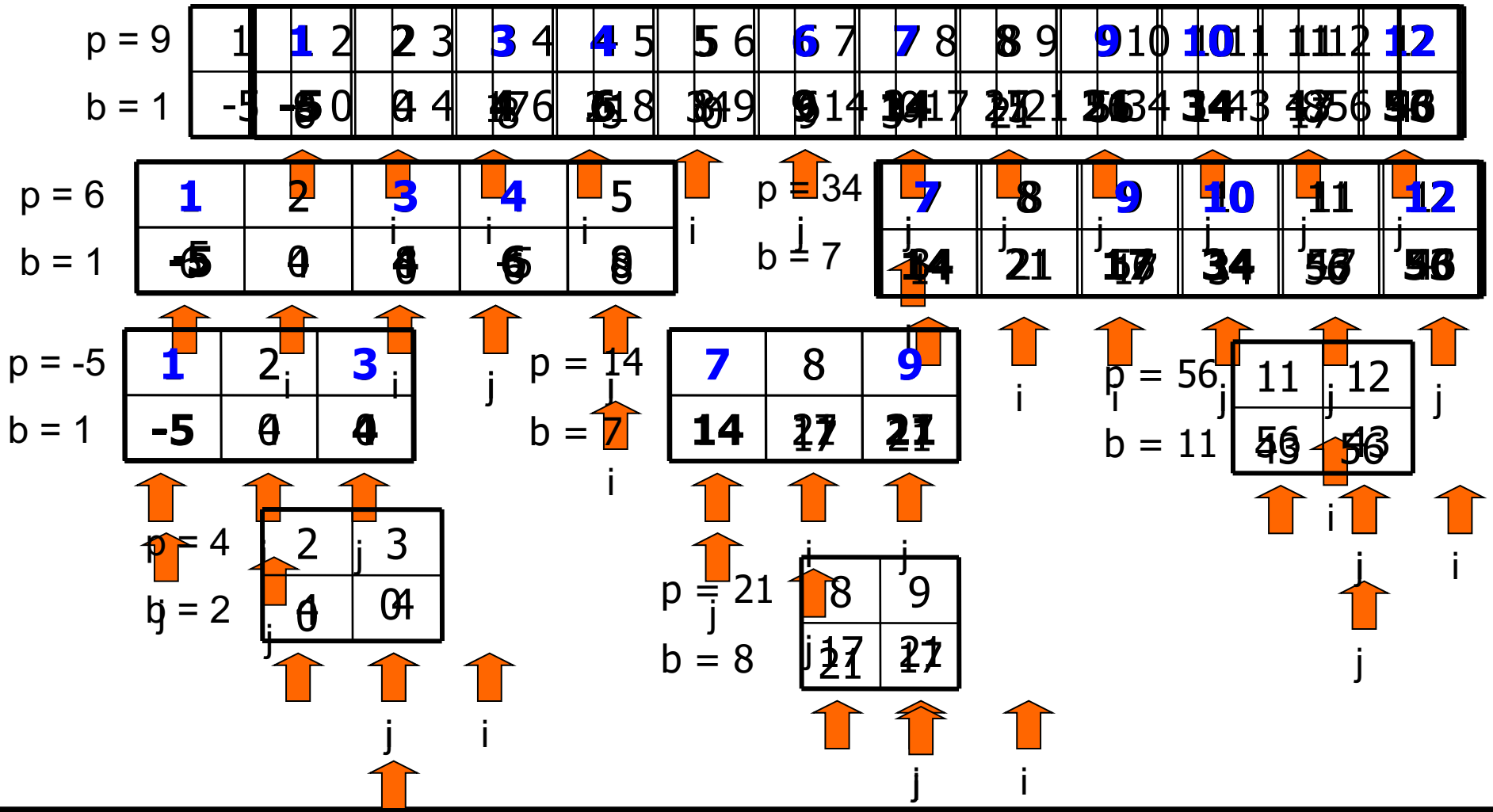
- Escoger un elemento de la matriz a ordenar como **pivote**
- Se **divide** el vector a ambos lados del pivote.
  - Elementos **mayores** que el pivote a la derecha
  - Elementos **menores** que el pivote a la izquierda
- El vector se ordena mediante llamadas recursivas a este algoritmo

Selección del pivote:

- Cualquier elemento es válido
- Usualmente se escoge el que ocupa la primera posición del sub-vector
- Mejor pivote: la mediana

Tema 2. Divide y vencerás

Aplicaciones de DyV. Ordenación de vectores



Tema 2. Divide y vencerás

Aplicaciones de DyV. Ordenación de vectores

```
procedimiento pivote(T[i..j]; var b)
   p ← T[i]
   k ← i
   b ← j + 1
   repetir
      k ← k + 1
   hasta que T[k] > p o k >= j frepetir
   repetir
      b ← b - 1
   hasta que T[b] <= p frepetir
   mientras k < b hacer
      intercambiar(T[k], T[b])
      repetir
         k ← k + 1
      hasta que T[k] > p frepetir
      repetir
         b ← b - 1
      hasta que T[b] <= p frepetir
   fmientras;
   intercambiar(T[i], T[b])
fprocedimiento
```

Tema 2. Divide y vencerás

Aplicaciones de DyV. Ordenación de vectores

```
procedimiento ordenacion_rapida(T[i..j])
    {Ordena la submatriz T[i..j] por orden no decreciente}
    si esSuficientementePequeño(j - i) entonces
        ordenar (T[i..j]) {P.ej., Ord. por inserción}
    sino
        pivote(T[i..j],b);
        ordenacion_rapida (T[i..b-1])
        ordenación_rapida (T[b+1..j])
    fsi
fprocedimiento
```

Cálculo de la eficiencia de pivote:

- Todos los elementos de T son **diferentes** $\Rightarrow$ las n! permutaciones tienen la misma probabilidad
- El valor *b* devuelto por pivote(T[1..n],b) será un entero entre 1 y *n* con probabilidad 1/n
- pivote(T[1..n],b) requiere un tiempo $g(n) \in \Theta(n)$

Cálculo de la eficiencia de ordenación rápida:

- El orden inicial del vector es aleatorio
- Los elementos de T son diferentes en su mayoría
- Las n! permutaciones de sus elementos son igualmente probables

Cálculo de la eficiencia de ordenación rápida:

- Si se verifican las hipótesis anteriores $\Rightarrow$ Sub-problemas equilibrados con $O(n \log n)$
- Sub-problemas no estén del todo equilibrados, se asume $O(n \log n)$

Los casos peores $O(n^2)$:

- Vector está inicialmente ordenado
- Todos los elementos son iguales

$\Rightarrow$2 sub-problemas: uno de tamaño 0 y el segundo de tamaño j – i + 1

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:

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

Tema 2. Divide y vencerás

Aplicaciones de DyV. Multiplicación de matrices

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				

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

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

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

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

Tema 2. Divide y vencerás

Aplicaciones de DyV. Multiplicación de matrices

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	

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

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

Tema 2. Divide y vencerás

Aplicaciones de DyV. Multiplicación de matrices

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	

A1	A2
8	4
3	8
A3	A4

*

B1	B2
5	3
4	6
B3	B4

=

C1	C2
56	
C3	C3

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

Identificación del problema con el esquema

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

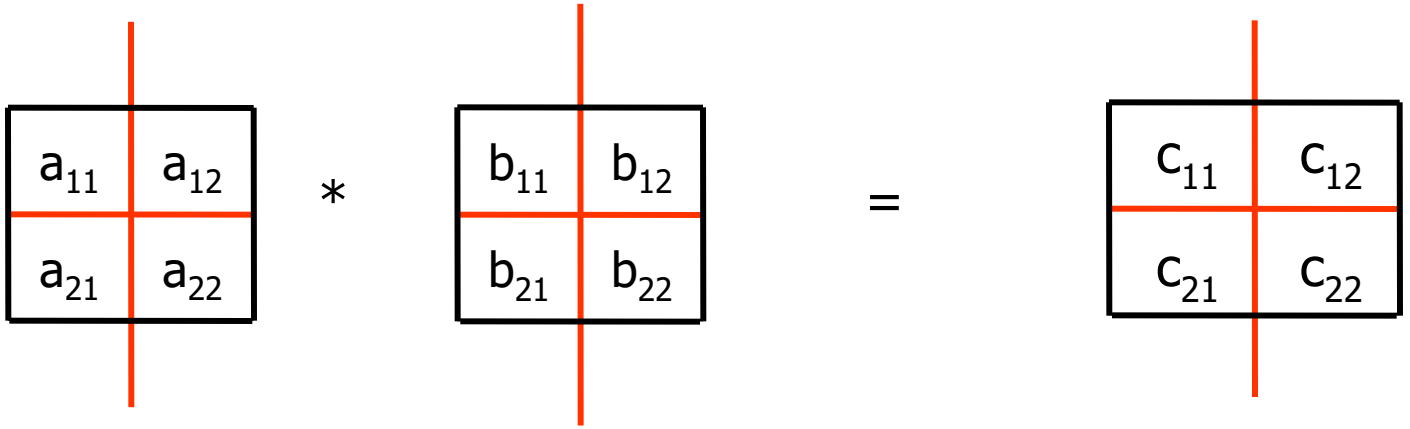
Tema 2. Divide y vencerás

Aplicaciones de DyV. Multiplicación de matrices

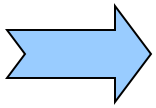
```
procedimiento multiplica (A,B: TipoMatriz; VAR C: TipoMatriz);
Var                    A11,A12,A21,A22,B11,B12,B21,B22,C11,C12,C21,C22,CAux:
   TipoMatriz;
   si tamaño(A) = 1 entonces
    C ← A*B
   sino
    DividirEnCuadrantes(A,A11,A12,A21,A22);
    DividirEnCuadrantes(B,B11,B12,B21,B22);
    para 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
```

Tema 2. Divide y vencerás

Aplicaciones de DyV. Multiplicación de matrices



$m_1 = (a_{21} + a_{22} - a_{11})(b_{22} - b_{12} + b_{11})$
$m_2 = a_{11}b_{11}$
$m_3 = a_{12}b_{12}$
$m_4 = (a_{11} - a_{21})(b_{22} - b_{12})$
$m_5 = (a_{21} - a_{22})(b_{12} - b_{11})$
$m_6 = (a_{12}-a_{21}+a_{11}-a_{22})b_{22}$
$m_7 = a_{22}(b_{11}-b_{22}-b_{11}-b_{21})$



Se reduce de 8 a 7 el número de productos necesarios

$c_{11} = m_2 + m_3$
$c_{12} = m_1 + m_2 + m_5 + m_6$
$c_{21} = m_1 + m_2 + m_4 + m_7$
$c_{22} = m_1 + m_2 + m_4 + m_5$

Aplicaciones de DyV. Multiplicación de matrices

- Si consideramos que cada componente a_{ij} y b_{ij} pueden ser matrices de *n*x*n* este algoritmo resulta interesante

Sea t(n) en tpo necesario para multiplicar dos matrices de tamaño *n* mediante el algoritmo de Strassen:

- t(n) = a t(n/b) + g(n)
- g(n) es suma de matrices de tamaño n:
- g(n) $\in \Theta$(n2)

Cálculo de la eficiencia

- `strassen` $g(n) \in O(n^p) = O(n^2), p = 2$

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

- Dos siete sub-problemas $\Rightarrow k = 7$
- En cada paso dividimos el tamaño del problema a la mitad $\Rightarrow b = 2$

$$k > b^p \Rightarrow 7 = 2^2$$

$$T(n) \in \Theta(n^{\log_b k}) \Rightarrow T(n) \in \Theta(n^{\lg(7)})$$