

Técnicas de diseño de algoritmos

Divide y conquista

Divide y Conquista

Esquema algorítmico

```
tipoResultado DivideyConquista (tipoDato X)
{ tipoDato  $x_1, x_2, \dots, x_k$ ;
  tipoResultado  $y_1, y_2, \dots, y_k$ ;
  if (simple(x)) return solución (x);
  else {  $x_1 = \text{Parte}_1(x)$ ;
         $x_2 = \text{Parte}_2(x)$ ;
        ...
         $x_k = \text{Parte}_k(x)$ ;
  return Combinar ( DivideyConquista ( $x_1$ ), DivideyConquista ( $x_2$ ), ...
                    DivideyConquista ( $x_k$ ));
}
```

Divide y Conquista.

Consideraciones

La solución de un problema se obtiene combinando la solución de subproblemas idénticos al problema original

- No se resuelve el mismo problema más de una vez
- El tamaño de las entradas de las partes es una fracción del tamaño de la entrada del problema original

Para lograr algoritmos eficientes

- Convienen problemas balanceados
- Las funciones $Parte_1$, $Parte_2, \dots$ y $Combina$ deben ser eficientes!

Divide y Conquista.

Un problema...

Supongamos que queremos invertir en acciones de una empresa.

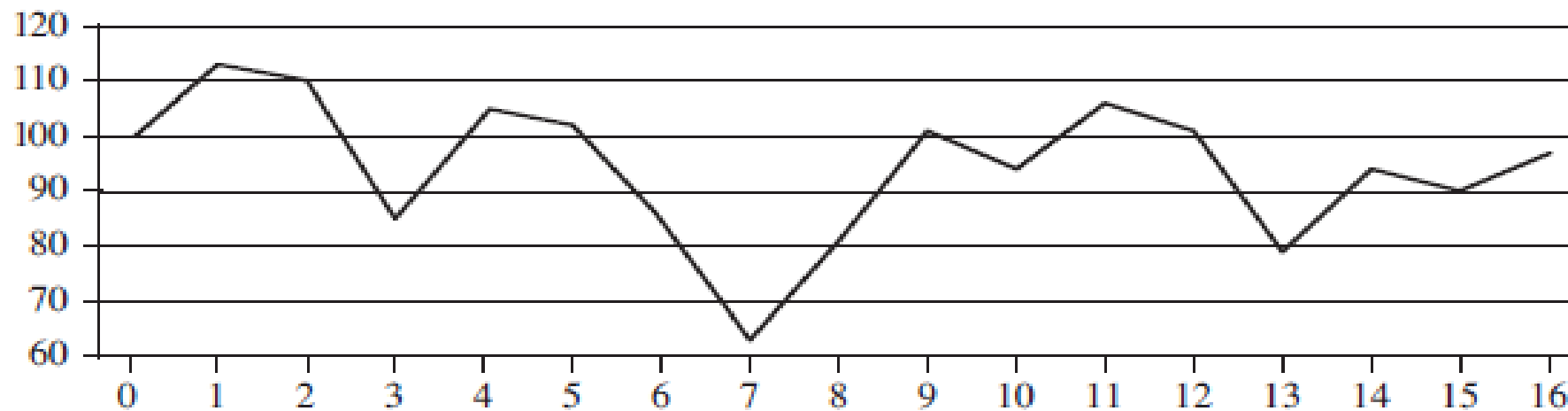
El precio de los activos de la empresa es volátil. Algunos tipos de activos experimentan periodos de volatilidad alta y baja.

La volatilidad puede permitir obtener beneficio si se vende en los picos y se compra en las bajas.

Podemos comprar en un determinado día y vender en otro y se conocen las previsiones de los precios de los activos

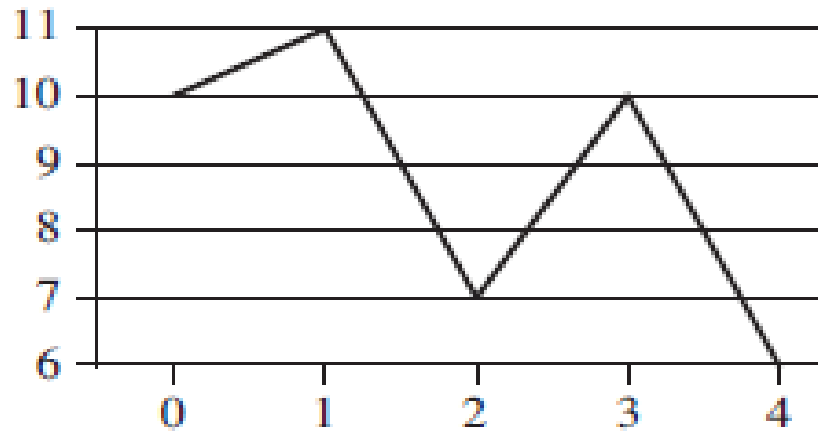
Divide y Conquista.

Día	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Precio	100	113	110	85	105	102	86	63	81	101	94	106	101	79	94	90	97
diferencia		13	-3	-25	20	-3	-16	-23	18	20	-7	12	-5	-22	15	-4	7



Divide y Conquista.

Analicemos un caso simple...

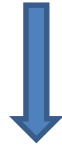


	0	1	2	3	4
	10	11	7	10	6
:		1	-4	3	-4

Conviene comprar el día 2 en 7\$ y vender el 3 en 10\$

Divide y Conquista.

Un algoritmo de “fuerza bruta” podría generar todos los pares de compra y venta. En n días hay $\text{comb}(n, 2)$ pares



$O(n^2)$

Es posible mejorarlo?



Divide y Conquista.

Comprar el día 7 a 63\$ y vender el 11 a \$106, el beneficio es 43

Encontrar el sub-arreglo de suma máxima

A[8]..A[11] y la suma de los elementos es (18 + 20 -7 +12)

Día	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Precio	100	113	110	85	105	102	86	63	81	101	94	106	101	79	94	90	97
diferencia		13	-3	-25	20	-3	-16	-23	18	20	-7	12	-5	-22	15	-4	7

Divide y Conquista.

Un algoritmo de “fuerza bruta” podría generar todos los pares de compra y venta. En n días hay $\text{comb}(n, 2)$ pares



$O(n^2)$


Es posible mejorarlo?



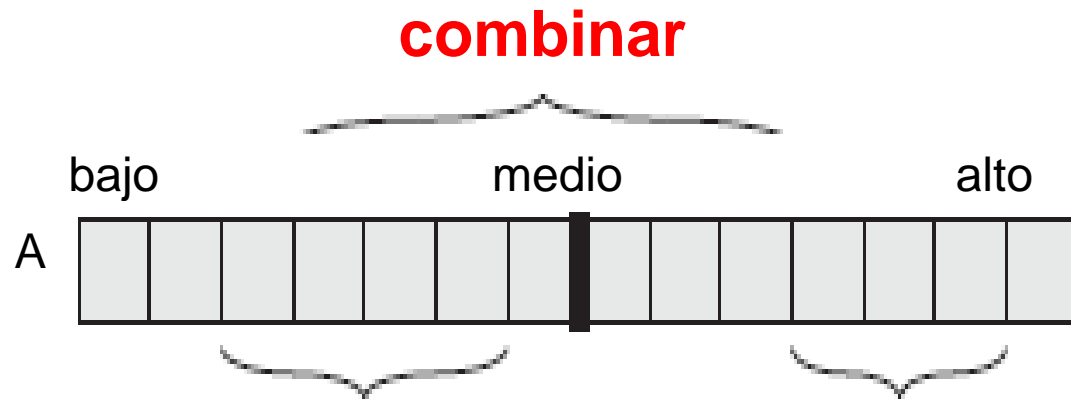
Divide y Conquista.

Encontrar el sub-arreglo de un arreglo de enteros de suma máxima

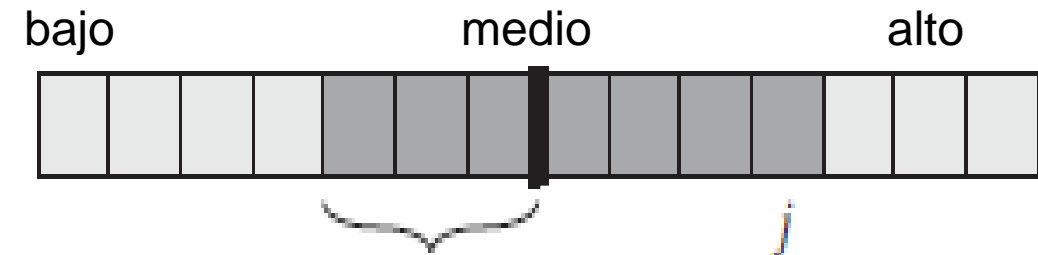
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	13	-3	-25	20	-3	-16	-23	18	20	-7	12	-5	-22	15	-4	7



Divide y Conquista.



Parte 1



Parte 2

Divide y conquista

Pseudo-código maximoSubarreglo

ENTRADA (A, bajo, alto)

SALIDA (bajoSol, altoSol, sumaSol)

// CASO BASE

if (alto== bajo)

return (bajo, alto, A[bajo]);

// CÁLCULO DE SUBPROBLEMAS

else

{ medio = (alto + bajo) / 2;

// Subproblema- Parte1

(bajolzq, altolzq,sumalzq) =
maximoSubarreglo(A, bajo, medio);

// Subproblema-Parte 2

(bajoDer, altoDer,sumaDer) =
maximoSubarreglo(A,medio+1, alto);

// Combinar soluciones



Divide y Conquista

// Combinar soluciones

```
(bajoComb, altoComb, sumaComb) =  
    SolucionCombinada( A, bajo, medio, alto);  
  
If ((sumalzq > sumaDer ) and ( sumalzq > sumaComb))  
    return (bajolzq, altolzq, sumalzq);  
If ((sumaDer >= sumalzq) and (sumaDer >= sumaComb))  
    return (bajoDer, altoDer, sumaDer);  
else return( bajoComb, altoComb, sumaComb);
```

Divide y Conquista.

Pseudo-código SolucionCombinada

Entrada (A, bajo, medio, alto)

Salida = (maxIzq, maxDer, sumaComb)

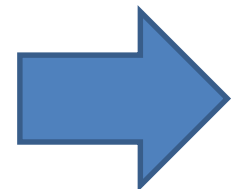
```
sumalzq = -  $\infty$  ;  
sumaDer = 0;  
for (i=1; i <= medio; i++)  
{ suma += A[medio-i +1];  
  if (suma > sumalzq)  
  { sumalzq = suma;  
    maxIzq = medio -i +1;  
  }  
}
```

```
sumaDer = -  $\infty$ ;  
suma = 0;  
for (j = medio + 1; j <= alto; j++)  
  suma += A[j];  
If (suma > sumaDer)  
{ sumaDer = suma;  
  maxDer = j;  
}
```

```
sumaComb = sumalzq +  
sumaDer
```

```
void SolucionMedio (int a[], unsigned int bajo,unsigned int medio,unsigned int alto,  
    unsigned int & bajomedio, unsigned int & altomedio, int & cantmedio)  
{  
    int sumaizq = ValorMínimoInicial;  
    int suma =0;  
    unsigned int i =0;  
    while (i <= medio)  
        {suma += a[medio -i];  
        if ( suma > sumaizq)  
            {sumaizq = suma; bajomedio= medio -i;}  
        i++;  
    }  
    int sumader = ValorMinimoInicial;  
    suma = 0;  
    unsigned int j=medio +1;  
    while ( j <= alto)  
        {suma += a[j];  
        if (suma > sumader)  
            {sumader = suma;  
            altomedio =j;}  
        j++;  
    }  
    cantmedio = sumaizq + sumader;  
}
```

```
void MayorSubA(int a[], unsigned int bajo, unsigned int alto, unsigned int & bajomax,  
               unsigned int & altomax, int & cant)  
{  
    unsigned int bajoizq ,altoizq, bajoder, altoder, bajomedio,altomedio;  
    int cantmedio, cantder,cantizq;  
  
    if (bajo == alto)  
        {bajomax= bajo;  
         altomax= alto;  
         cant = a[bajo];  
        }  
    else  
    {  
        unsigned int medio = (alto+bajo)/2;  
        MayorSubA(a, bajo , medio, bajoizq, altoizq,cantizq);  
        MayorSubA(a, medio +1, alto, bajoder, altoder,cantder);  
        SolucionMedio(a,bajo,medio,alto,bajomedio,altomedio,cantmedio);  
    }
```



// Combinar soluciones

```
if ((cantizq >= cantder) and (cantizq >= cantmedio))
{
    bajomax=bajoizq;
    altomax=altoizq;
    cant = cantizq;
}
else
    if ((cantder >= cantizq) and (cantder >= cantmedio))
    {
        bajomax=bajoder;
        altomax=altoder;
        cant = cantder;
    }
    else
    {
        bajomax=bajomedio;
        altomax=altomedio;
        cant = cantmedio;
    }
}
```

Divide y conquista

Complejidad temporal

Algoritmo de “fuerza bruta”

$O(n^2)$

Algoritmo por Divide y Conquista

$O(n \log n)$

Ejercicio propuesto

Implementar en C++

Determinar la complejidad temporal

Análisis de eficiencia comparativo entre el algoritmo de “fuerza bruta” y el implementado por Divide y Conquista

Divide y conquista

Algoritmo de Strassen

Sean dos matrices A y B de dimensiones $2^k \times 2^k$ particionadas en 4 submatrices de $2^{k-1} \times 2^{k-1}$.

La matriz producto $C = A \times B$ puede calcularse a partir del algoritmo propuesto por Volker Strassen

A_{11}	A_{12}
A_{21}	A_{22}

×

B_{11}	B_{12}
B_{21}	B_{22}

=

C_{11}	C_{12}
C_{21}	C_{22}

Divide y conquista

Algoritmo de Strassen

$$I = (A_{11} + A_{22}) (B_{11} + B_{22})$$

$$II = (A_{21} + A_{22}) B_{11}$$

$$III = A_{11} (B_{12} - B_{22})$$

$$IV = A_{22} (B_{21} - B_{11})$$

$$V = (A_{11} + A_{12}) B_{22}$$

$$VI = (A_{21} - A_{11}) (B_{11} + B_{12})$$

$$VII = (A_{12} - A_{22}) (B_{21} + B_{22})$$

$$C_{11} = I + IV - V + VII$$

$$C_{12} = III + V$$

$$C_{21} = II + IV$$

$$C_{22} = I + III - II + VI$$

I, II, III, IV, V, VI, VII son productos de matrices de dimensión $2^{k-1} \times 2^{k-1}$ que pueden ser resueltos recursivamente por el Algoritmo de Strassen

Algoritmo de Strassen

Secuencia de cálculo

	ESPACIO AUXILIAR	RESULTADOS
I	AUX1, AUX2	P1
II	AUX1	P2
III	AUX1	P3
IV	AUX1	P4
V	AUX1	P5
VI	AUX1, AUX2	P6
VII	AUX1, AUX2	P7
C11, C12, C21, C22		



Algoritmo de Strassen

Secuencia de cálculo:

I, II, III, IV, VI, VII, C11, C12, C21, C22

ESPACIO AUXILIAR $(2^K) \in O(9^K)$

¿Cómo realizar un uso más eficiente de espacio?

- Utilizar el espacio asignado al resultado del nivel para almacenar resultados parciales
- Cambiar la secuencia de cálculo

Algoritmo de Strassen

Otra secuencia de cálculo

- Sea MR el espacio asignado a la matriz resultado del nivel de dimensión 2^k .
- Sus submatrices, de dimensión $2^{k-1} \times 2^{k-1}$ son MR 11, MR 12, MR 21 y MR 22

Algoritmo de Strassen

Secuencia de cálculo

	ESPACIO AUXILIAR	RESULTADOS
I	AUX1, AUX2	AUX3
VII	AUX1, AUX2	MR 11
VI	AUX1, AUX2	MR 22
I + VII		MR 11
I + VI		MR 22
II	AUX1	MR21

Algoritmo de Strassen

	ESPACIO AUXILIAR	RESULTADO
III	AUX1	MR 12
III – II		AUX1
(I + VI) + (III – II)		MR 22
IV	AUX1	AUX2
II + IV		MR 21
V	AUX1	AUX3
IV – V		AUX2
(I + VII) + (IV – V)		MR 11
III + V		MR 12

Secuencia de cálculo: I, VII, I + VII, VI, I+VI, II, III, III – II, (I+VI) + (III – II), IV, II + IV, V, IV – V, (I +VII) + (IV-V), III + V

ESPACIO AUXILIAR (2^K) $\in O(3^K)$

Representación de matrices

Las funciones *Parte₁*, *Parte₂*,... y *Combina* deben ser eficientes!

¿Cómo representar a las matrices?

Problemas recursivos



Representación recursiva de los datos

Representación de matrices

a11	a12	a21	a22	a13	a14	a23	a24	a31	a32	a41	a42	a33	a34	a43	a44
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Arreglos unidimensionales

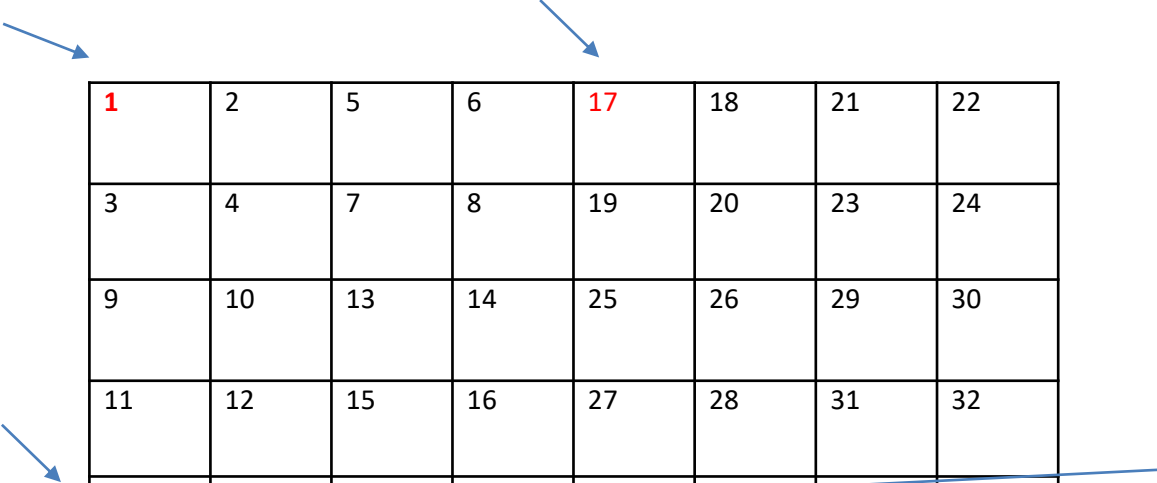
Matriz

a11	a12	a13	a14
a21	a22	a23	a24
a31	a32	a33	a34
a41	a42	a43	a44

Índice en el arreglo

1	2	5	6
3	4	7	8
9	10	13	14
11	12	15	16

Representación de matrices



The diagram shows an 8x8 matrix A. Blue arrows indicate recursive partitioning: one arrow points to the top-left 4x4 quadrant, another to the top-right 4x4 quadrant, and a third to the bottom-left 4x4 quadrant. The matrix contains sequential integers from 1 to 64, with some cells highlighted in red: 1, 17, 33, and 49.

1	2	5	6	17	18	21	22
3	4	7	8	19	20	23	24
9	10	13	14	25	26	29	30
11	12	15	16	27	28	31	32
33	34	37	38	49	50	53	54
35	36	39	40	51	52	55	56
41	42	45	46	57	58	61	62
43	44	47	48	59	60	63	64

$$\text{índice}(A_{11}) = \text{índice}(A)$$

$$\text{índice}(A_{12}) = \text{índice}(A) + (n/2)^2$$

$$\text{Índice}(A_{21}) = \text{índice}(A) + 2(n/2)^2$$

$$\text{Índice}(A_{22}) = \text{índice}(A) + 3(n/2)^2$$

Se almacenan
recursivamente
 $A_{11}, A_{12}, A_{21}, A_{22}$

Matriz A dimensión $n \times n$ ($n = 2^k$)

Algoritmo de Strassen

¿Cómo aplicarlo con productos de matrices generales, no necesariamente cuadradas y de dimensión $2^k \times 2^k$?

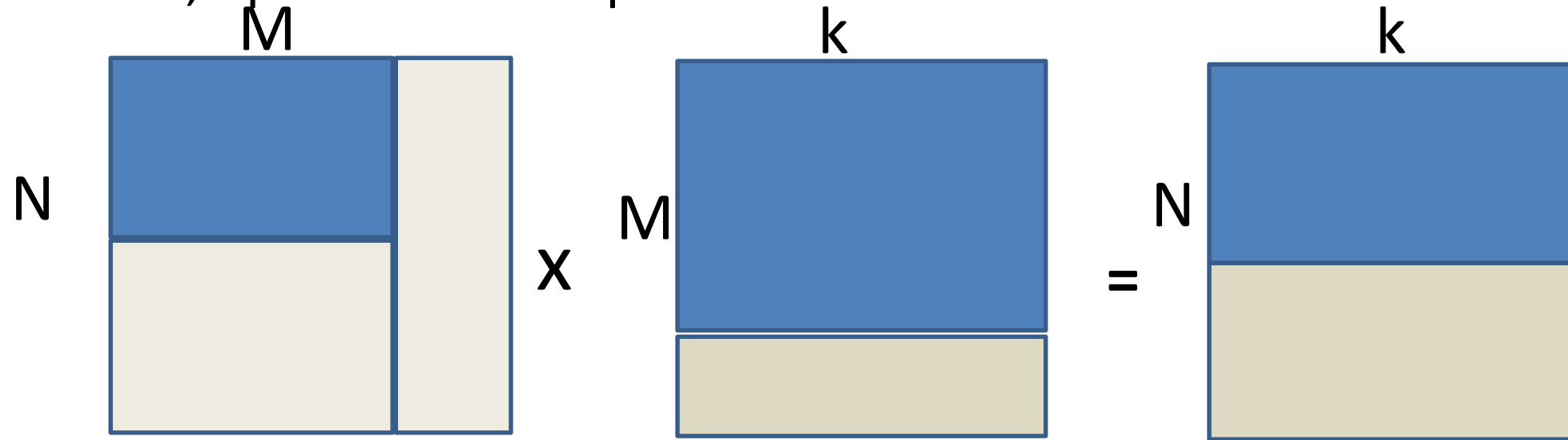
Combinar el algoritmo de Strassen con el tradicional

Siempre es posible encontrar dos enteros m y k (positivos o nulos) tal que $n = m 2^k$ y particionar en:

- $m \times m$ submatrices de $2^k \times 2^k$ que pueden ser resueltas por Strassen
- Ó
- suponer a las matrices cubiertas con ceros hasta la potencia de 2 más cercana, operando conceptualmente con los ceros

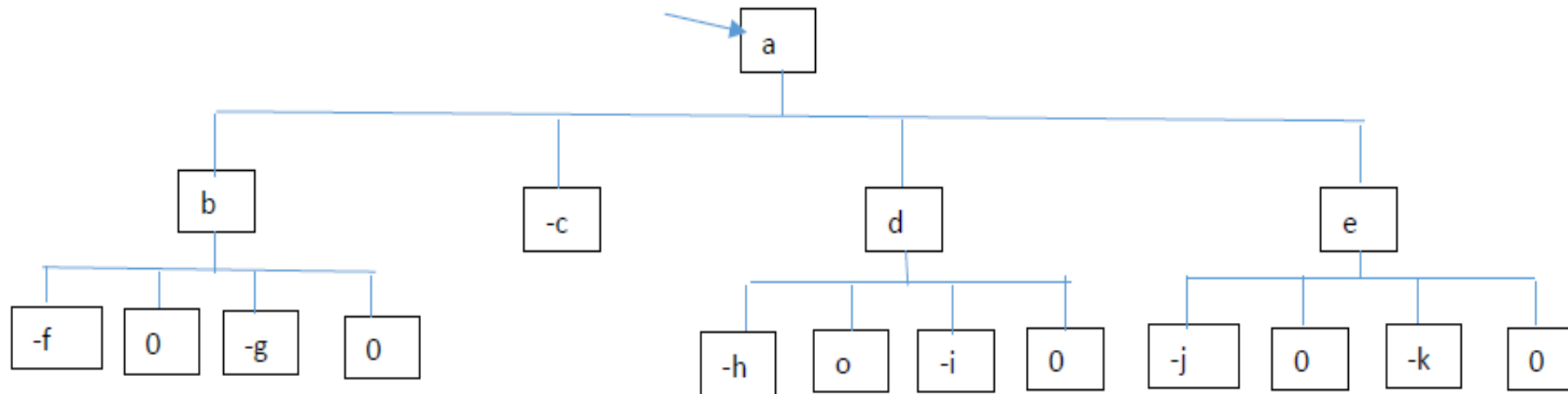
Algoritmo de Strassen

- Suponer a las matrices cubiertas con ceros hasta la potencia de 2 más cercana, operando conceptualmente con los ceros



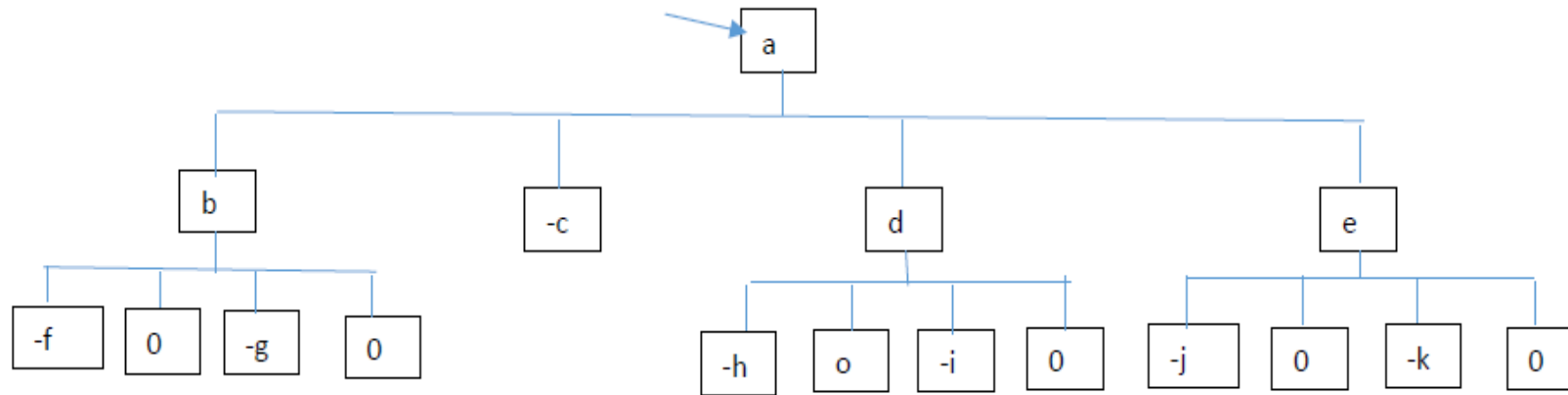
Se completa con ceros hasta la potencia de dos más cercana a
 $L = \text{MAX}(N, M, K)$

Matriz Quadtree



Clave = 0 REGIÓN NULA ; CLAVE > 0 REGIÓN NO HOMOGÉNEA;
CLAVE < 0 REGIÓN HOMOGÉNEA

Matriz Quadtree



Representación Matriz Quadtree

Arreglo de direcciones:

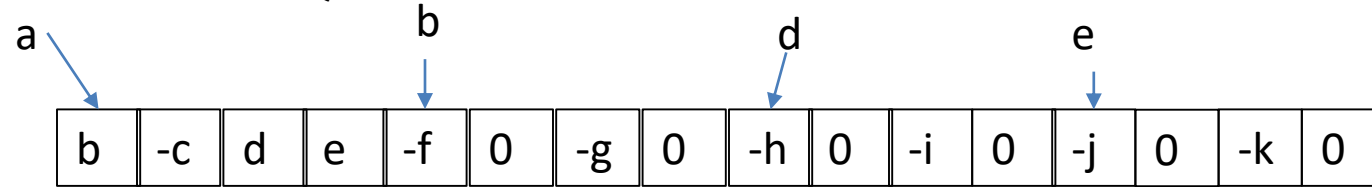
- Un valor positivo indica una matriz que vuelvo a particionar y representa el índice dentro del mismo arreglo donde se encuentran las particiones.
- Un valor negativo indica el índice del arreglo de valores donde está almacenada la información
- Un cero indica una región de ceros “conocidos”.

Arreglo de Valores: Almacena la información contenida en la matriz.

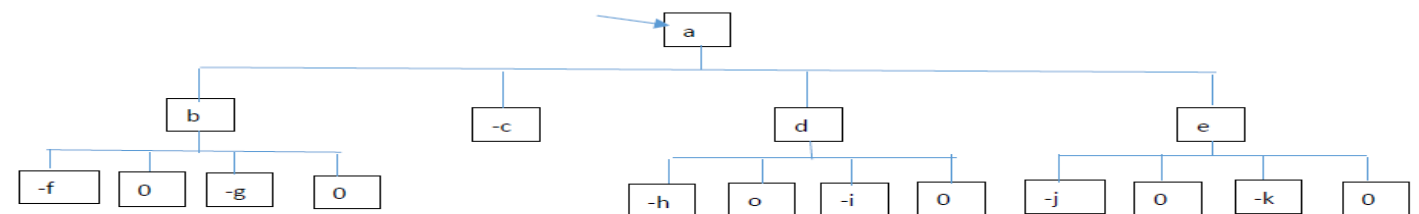
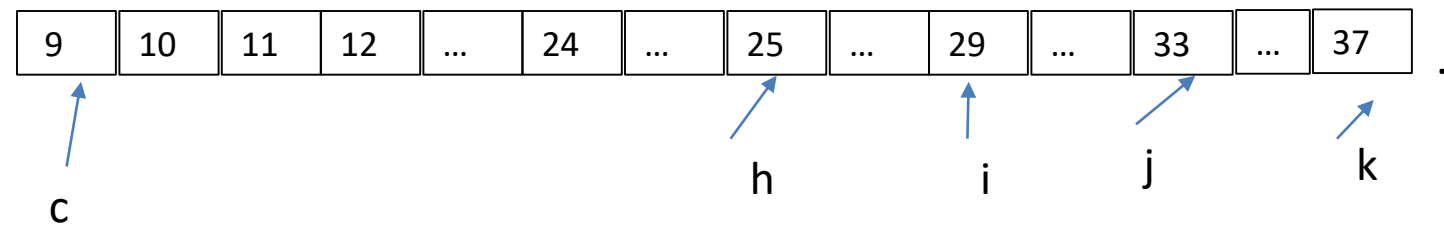
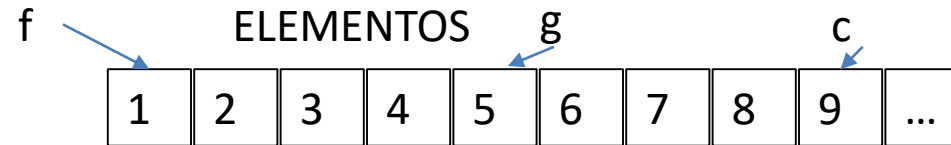
Matriz Quadtree

1	2	0	0	9	10	13	14
3	4	0	0	11	12	15	16
5	6	0	0	17	18	21	22
7	8	0	0	19	20	23	24
25	26	0	0	33	34	0	0
27	28	0	0	35	36	0	0
29	30	0	0	37	38	0	0
31	32	0	0	39	40	0	0

Matriz QUADTREE



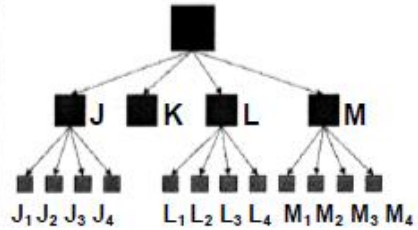
ELEMENTOS



Algoritmo de Strassen

Matrices ralas

J				K			
1	2	0	0	9	10	13	14
3	4	0	0	11	12	15	16
5	6	0	0	17	18	21	22
7	8	0	0	19	20	23	24
25	26	0	0	33	34	0	0
27	28	0	0	35	36	0	0
29	30	0	0	37	38	0	0
31	32	0	0	39	40	0	0
L				M			



Se representan mediante
QUADTREE

