

PREDA - UNED

Programación y Estructuras de Datos Avanzadas

Capítulo 4

Divide y vencerás

Estrategia

1. Descomposición del problema en subproblemas de su mismo tipo o naturaleza
 - Disminuir la complejidad y ¿paralelizar?
 - Para casos sencillos se aporta solución trivial (base de la recursión)
2. Resolución recursiva de los subproblemas
3. Combinación, si procede, de las soluciones de los subproblemas

Elementos

- Trivial y solución-trivial:
 - ¿Cuándo se convierte el problema en **trivial**?
Cuando una función pueda resolverlo sin descomponerlo
- Descomponer:
 - **Dividir** el problema en subproblemas más pequeños que el inicial
 - El tipo de sucesión formada por los sucesivos tamaños del problema y el nº de subproblemas determinan la **complejidad**
- Combinar:
 - Aplicando el principio de inducción, se dan los subproblemas por resueltos y lo que queda es cómo **combinar** las soluciones de los subproblemas para obtener la solución del problema final
 - Si no hay que realizar combinación de soluciones el tipo de problema se conoce como **reducción**

Esquema general

```
fun DyV(problema)
    si trivial(problema) entonces
        dev solución-trivial
    sino hacer
         $\{p_1, p_2, \dots, p_k\} \leftarrow \text{descomponer}(\text{problema})$ 
        para  $i \in (1..k)$  hacer
             $s_i \leftarrow \text{DyV}(p_i)$ 
        fpara
    fsi
    dev combinar( $s_1, s_2, \dots, s_k$ )
ffun
```

Ejemplo

- Búsqueda binaria en un vector ordenado:

```
fun Bbinaria(i,j:entero;v: vector [1..N] de entero; x:entero): booleano
    var
        m:entero
    var
        si i = j entonces
            si v[i] = x entonces
                dev verdadero
            sino
                dev falso
            fsi
        sino
            m ← (i+j) div 2
            si x ≤ v[m] entonces
                bbinaria(i,m,v,x)
            sino
                bbinaria(m+1,j,v,x)
            fsi
    ffun
```

Ejemplo de problema de reducción
(sin combinación de soluciones parciales)

Ejercicio de examen

5. Para resolver determinado problema hemos diseñado un algoritmo de tipo divide y vencerás. ¿Cuál de las siguientes afirmaciones es falsa?:
- (a) La resolución debe alcanzar un caso trivial que se pueda resolver sin realizar nuevas descomposiciones.
 - (b) El problema se resuelve por divisiones sucesivas en subproblemas, que pueden ser de mayor o de menor tamaño que el de partida.
 - (c) Los algoritmos basados en este esquema pueden requerir un paso de combinación de las soluciones parciales.
 - (d) El algoritmo de la búsqueda binaria aplica la estrategia divide y vencerás.

Ejercicio de examen

5. Para resolver determinado problema hemos diseñado un algoritmo de tipo divide y vencerás. ¿Cuál de las siguientes afirmaciones es falsa?:
- (a) La resolución debe alcanzar un caso trivial que se pueda resolver sin realizar nuevas descomposiciones.
 - (b) El problema se resuelve por divisiones sucesivas en subproblemas, que pueden ser de mayor o de menor tamaño que el de partida.
 - (c) Los algoritmos basados en este esquema pueden requerir un paso de combinación de las soluciones parciales.
 - (d) El algoritmo de la búsqueda binaria aplica la estrategia divide y vencerás.

El tamaño de los subproblemas
no puede ser mayor

Ejercicio de examen

4. Con respecto al esquema divide y vencerás indique cuál de las siguientes afirmaciones es verdadera:
- (a) En algunos casos permite la paralelización de la resolución de los subproblemas del mismo tipo.
 - (b) El coste de este tipo de algoritmos depende del número de subproblemas en los que se descompone el problema pero no de la reducción del tamaño en las sucesivas llamadas recursivas.
 - (c) Siempre es necesario combinar las soluciones de los subproblemas.
 - (d) Este esquema aplica el principio de deducción sobre los diversos ejemplares del problema.

Ejercicio de examen

4. Con respecto al esquema divide y vencerás indique cuál de las siguientes afirmaciones es verdadera:

- (a) En algunos casos permite la paralelización de la resolución de los subproblemas del mismo tipo.
- (b) El coste de este tipo de algoritmos depende del número de subproblemas en los que se descompone el problema pero no de la reducción del tamaño en las sucesivas llamadas recursivas.
- (c) Siempre es necesario combinar las soluciones de los subproblemas.
- (d) Este esquema aplica el principio de deducción sobre los diversos ejemplares del problema.

Ejercicio de examen

1. Dado el siguiente algoritmo:

```
función X(int n)
    si n>0 entonces
        escribir "n";
        X(n-1);
        escribir "n-1";
    fsi
```

Indica cuál de las siguientes sería la salida al ejecutar la llamada “X(5)”:

- a. 5 4 3 2 1 0 4 3 2 1 0.
- b. 5 4 3 2 1 0 1 2 3 4.
- c. 5 4 3 2 1 0 1 2 3 4 0.
- d. 5 4 3 2 1 0 4 3 2 1.

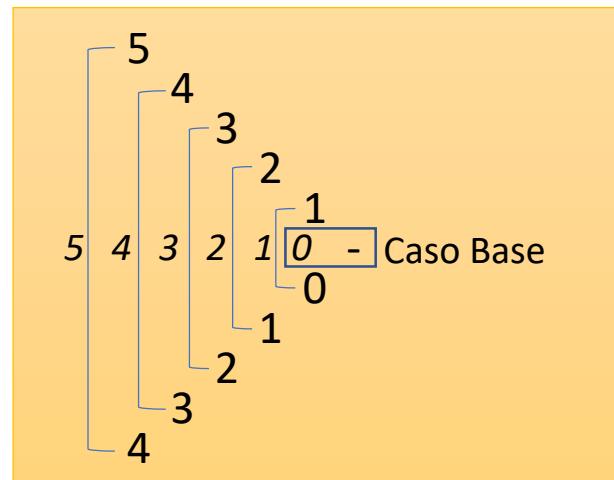
Ejercicio de examen

1. Dado el siguiente algoritmo:

```
función X(int n)
    si n>0 entonces
        escribir "n";
        X(n-1);
        escribir "n-1";
    fsi
```

Indica cuál de las siguientes sería la salida al ejecutar la llamada “X(5)”: 

- a. 5 4 3 2 1 0 4 3 2 1 0.
- b. 5 4 3 2 1 0 1 2 3 4.
- c. 5 4 3 2 1 0 1 2 3 4 0.
- d. 5 4 3 2 1 0 4 3 2 1.



Coste

- **a**: número de subproblemas
- **b**: factor de reducción de tamaño
- **Expresión polinómica**: coste de la función de combinación
(si $k=0$ es problema de reducción)

- Depende de cuántas descomposiciones se realicen y del tipo de decrecimiento de las sucesivas llamadas recursivas a los subproblemas
 - Progresión geométrica

$$T(n) = \begin{cases} cn^k & , \text{ si } 1 \leq n < b \\ aT(n/b) + cn^k & , \text{ si } n \geq b \end{cases}$$

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

- Progresión aritmética

$$T(n) = \begin{cases} cn^k & , \text{ si } 1 \leq n < b \\ aT(n-b) + cn^k & , \text{ si } n \geq b \end{cases}$$

$$T(n) \in \begin{cases} \Theta(n^k) & , \text{ si } a < 1 \\ \Theta(n^{k+1}) & , \text{ si } a = 1 \\ \Theta(a^{n/b}) & , \text{ si } a > 1 \end{cases}$$

Coste

- **a**: número de subproblemas
- **b**: factor de reducción de tamaño
- **Expresión polinómica**: coste de la función de combinación
(si $k=0$ es problema de reducción)

- Depende de cuántas descomposiciones se realicen y del tipo de decrecimiento de las sucesivas llamadas recursivas a los subproblemas
 - Progresión geométrica

$$T(n) = \begin{cases} cn^k & , \text{ si } 1 \leq n < b \\ aT(n/b) + cn^k & , \text{ si } n \geq b \end{cases}$$

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

En el caso de la búsqueda binaria:

- ¿a?
- ¿b?
- ¿k?

¿T(n)?

< 1
= 1
> 1

Coste

- **a**: número de subproblemas
- **b**: factor de reducción de tamaño
- **Expresión polinómica**: coste de la función de combinación
(si $k=0$ es problema de reducción)

- Depende de cuántas descomposiciones se realicen y del tipo de decrecimiento de las sucesivas llamadas recursivas a los subproblemas
 - Progresión geométrica

$$T(n) = \begin{cases} cn^k & , \text{ si } 1 \leq n < b \\ aT(n/b) + cn^k & , \text{ si } n \geq b \end{cases}$$

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

En el caso de la búsqueda binaria:

- **a = 1** (nos quedamos solo con la mitad donde puede estar el elemento)
- **b = 2**
- **k = 0**

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

Ejercicio de examen

4. Se tiene que resolver un problema de tipo divide y vencerás de tamaño n donde el decrecimiento del problema es en progresión aritmética a razón de dos llamadas recursivas con tamaño $n-1$, siendo el resto de las operaciones realizadas en el algoritmo de coste $O(k)$ con k constante. El coste del algoritmo es:
- a) $O(n^2)$
 - b) $O(2^n)$
 - c) $O(n^2 \cdot \log n)$
 - d) $O(2n)$

Ejercicio de examen

4. Se tiene que resolver un problema de tipo divide y vencerás de tamaño n donde el decrecimiento del problema es en progresión aritmética a razón de dos llamadas recursivas con tamaño $n-1$, siendo el resto de las operaciones realizadas en el algoritmo de coste $O(k)$ con k constante. El coste del algoritmo es:

- a) $O(n^2)$
- b) $O(2^n)$
- c) $O(n^2 \cdot \log n)$
- d) $O(2n)$

Progresión
geométrica

$$T(n) = \begin{cases} cn^k & , \text{ si } 1 \leq n < b \\ aT(n/b) + cn^k & , \text{ si } n \geq b \end{cases}$$

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

Progresión
aritmética

$$T(n) = \begin{cases} cn^k & , \text{ si } 1 \leq n < b \\ aT(n-b) + cn^k & , \text{ si } n \geq b \end{cases}$$

$$T(n) \in \begin{cases} \Theta(n^k) & , \text{ si } a < 1 \\ \Theta(n^{k+1}) & , \text{ si } a = 1 \\ \Theta(a^{n/b}) & , \text{ si } a > 1 \end{cases}$$

Ejercicio de examen

4. Se tiene que resolver un problema de tipo divide y vencerás de tamaño n donde el decrecimiento del problema es en progresión aritmética a razón de dos llamadas recursivas con tamaño $n-1$, siendo el resto de las operaciones realizadas en el algoritmo de coste $O(k)$ con k constante. El coste del algoritmo es:

- a) $O(n^2)$
- b) $O(2^n)$
- c) $O(n^2 \cdot \log n)$
- d) $O(2n)$

Progresión
geométrica

$$T(n) = \begin{cases} cn^k & , \text{ si } 1 \leq n < b \\ aT(n/b) + cn^k & , \text{ si } n \geq b \end{cases}$$

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

Progresión
aritmética

$$T(n) = \begin{cases} cn^k & , \text{ si } 1 \leq n < b \\ aT(n-b) + cn^k & , \text{ si } n \geq b \end{cases}$$

$$T(n) \in \begin{cases} \Theta(n^k) & , \text{ si } a < 1 \\ \Theta(n^{k+1}) & , \text{ si } a = 1 \\ \Theta(a^{n/b}) & , \text{ si } a > 1 \end{cases}$$

Ejercicio de examen

4. Se tiene que resolver un problema de tipo divide y vencerás de tamaño n donde el decrecimiento del problema es en progresión aritmética a razón de dos llamadas recursivas con tamaño $n-1$, siendo el resto de las operaciones realizadas en el algoritmo de coste $O(k)$ con k constante. El coste del algoritmo es:

- a) $O(n^2)$
- b) $O(2^n)$
- c) $O(n^2 \cdot \log n)$
- d) $O(2n)$

$$a = 2; b = 1; k = 0 \rightarrow ?$$

Progresión geométrica

$$T(n) = \begin{cases} cn^k & , \text{ si } 1 \leq n < b \\ aT(n/b) + cn^k & , \text{ si } n \geq b \end{cases}$$

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

Progresión aritmética

$$T(n) = \begin{cases} cn^k & , \text{ si } 1 \leq n < b \\ aT(n-b) + cn^k & , \text{ si } n \geq b \end{cases}$$

$$T(n) \in \begin{cases} \Theta(n^k) & , \text{ si } a < 1 \\ \Theta(n^{k+1}) & , \text{ si } a = 1 \\ \Theta(a^{n/b}) & , \text{ si } a > 1 \end{cases}$$

Ejercicio de examen

4. Se tiene que resolver un problema de tipo divide y vencerás de tamaño n donde el decrecimiento del problema es en progresión aritmética a razón de dos llamadas recursivas con tamaño $n-1$, siendo el resto de las operaciones realizadas en el algoritmo de coste $O(k)$ con k constante. El coste del algoritmo es:

- a) $O(n^2)$
b) $O(2^n)$
c) $O(n^2 \cdot \log n)$
d) $O(2n)$

$$a = 2; b = 1; k = 0 \rightarrow O(2^n)$$

Progresión geométrica

$$T(n) = \begin{cases} cn^k & , \text{ si } 1 \leq n < b \\ aT(n/b) + cn^k & , \text{ si } n \geq b \end{cases}$$

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

Progresión aritmética

$$T(n) = \begin{cases} cn^k & , \text{ si } 1 \leq n < b \\ aT(n-b) + cn^k & , \text{ si } n \geq b \end{cases}$$

$$T(n) \in \begin{cases} \Theta(n^k) & , \text{ si } a < 1 \\ \Theta(n^{k+1}) & , \text{ si } a = 1 \\ \Theta(a^{n/b}) & , \text{ si } a > 1 \end{cases}$$

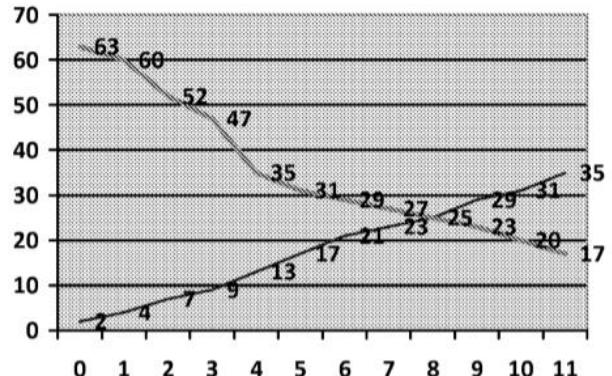
Ejercicio de examen

3. Considere dos vectores f y g de n elementos que representan los valores que toman dos funciones en el intervalo $[0..n-1]$. Los dos vectores están ordenados, pero el primero f es un vector estrictamente creciente ($f[0] < f[1] < \dots < f[n-1]$), mientras g es un vector estrictamente decreciente ($g[0] > g[1] > \dots > g[n-1]$). Las curvas que representan dichos vectores se cruzan en un punto concreto, y lo que se desea saber es si dicho punto está contenido entre las componentes de ambos vectores, es decir, si existe un valor i tal que $f[i] = g[i]$ para $0 \leq i \leq n - 1$.

La figura muestra un ejemplo en el que las gráficas se cruzan en $i=8$ que se corresponde con el valor 25 en ambas curvas.

Se busca un algoritmo que compruebe si el punto de cruce está contenido en las componentes de ambos vectores. ¿Cuál de las siguientes opciones es cierta?

- (a) Se puede encontrar un algoritmo recursivo de coste logarítmico.
- (b) El algoritmo más eficiente que se puede encontrar es $O(n)$.
- (c) El algoritmo más eficiente que se puede encontrar es $O(n^2)$
- (d) Ninguna de las anteriores.



Ejercicio de examen

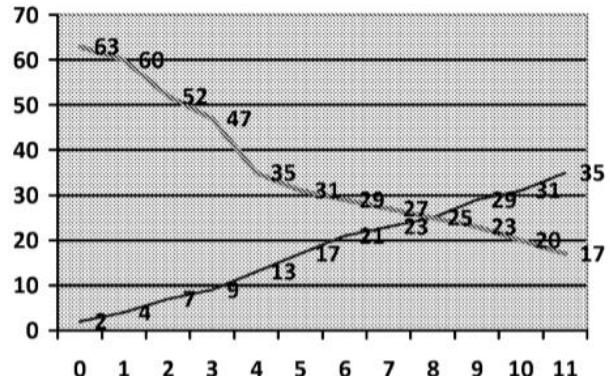
3. Considere dos vectores f y g de n elementos que representan los valores que toman dos funciones en el intervalo $[0..n-1]$. Los dos vectores están ordenados, pero el primero f es un vector estrictamente creciente ($f[0] < f[1] < \dots < f[n-1]$), mientras g es un vector estrictamente decreciente ($g[0] > g[1] > \dots > g[n-1]$). Las curvas que representan dichos vectores se cruzan en un punto concreto, y lo que se desea saber es si dicho punto está contenido entre las componentes de ambos vectores, es decir, si existe un valor i tal que $f[i] = g[i]$ para $0 \leq i \leq n - 1$.

La figura muestra un ejemplo en el que las gráficas se cruzan en $i=8$ que se corresponde con el valor 25 en ambas curvas.

Se busca un algoritmo que compruebe si el punto de cruce está contenido en las componentes de ambos vectores. ¿Cuál de las siguientes opciones es cierta?

- (a) Se puede encontrar un algoritmo recursivo de coste logarítmico.
- (b) El algoritmo más eficiente que se puede encontrar es $O(n)$.
- (c) El algoritmo más eficiente que se puede encontrar es $O(n^2)$
- (d) Ninguna de las anteriores.

Búsqueda binaria



Ejercicio de examen

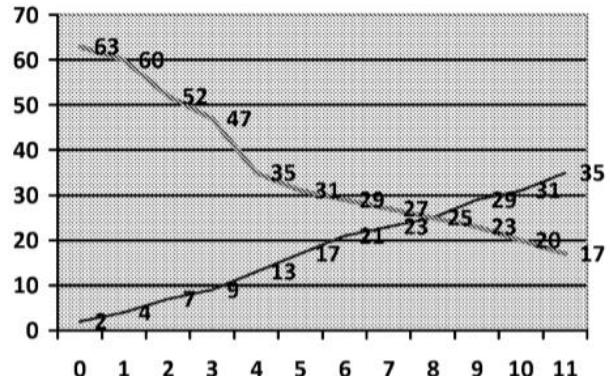
3. Considere dos vectores f y g de n elementos que representan los valores que toman dos funciones en el intervalo $[0..n-1]$. Los dos vectores están ordenados, pero el primero f es un vector estrictamente creciente ($f[0] < f[1] < \dots < f[n-1]$), mientras g es un vector estrictamente decreciente ($g[0] > g[1] > \dots > g[n-1]$). Las curvas que representan dichos vectores se cruzan en un punto concreto, y lo que se desea saber es si dicho punto está contenido entre las componentes de ambos vectores, es decir, si existe un valor i tal que $f[i] = g[i]$ para $0 \leq i \leq n - 1$.

La figura muestra un ejemplo en el que las gráficas se cruzan en $i=8$ que se corresponde con el valor 25 en ambas curvas.

Se busca un algoritmo que compruebe si el punto de cruce está contenido en las componentes de ambos vectores. ¿Cuál de las siguientes opciones es cierta?

- (a) Se puede encontrar un algoritmo recursivo de coste logarítmico.
(b) El algoritmo más eficiente que se puede encontrar es $O(n)$.
(c) El algoritmo más eficiente que se puede encontrar es $O(n^2)$
(d) Ninguna de las anteriores.

Búsqueda binaria



Ejercicio de examen

2. Se tiene un vector V de números enteros no repetidos y ordenados de menor a mayor. Se desea comprobar si existe algún elemento del vector que coincida con su índice ($V[i]=i$). Indica cuál de las siguientes afirmaciones es cierta:
- (a) La solución más eficiente se obtiene mediante programación y dinámica y tiene un coste de $O(n)$.
 - (b) Es posible resolver el problema en tiempo logarítmico mediante divide y vencerás.
 - (c) La solución más eficiente tiene un coste $O(n^2)$.
 - (d) Ninguna de las anteriores es cierta.

Ejercicio de examen

2. Se tiene un vector V de números enteros no repetidos y ordenados de menor a mayor. Se desea comprobar si existe algún elemento del vector que coincida con su índice ($V[i]=i$). Indica cuál de las siguientes afirmaciones es cierta:
- (a) La solución más eficiente se obtiene mediante programación y dinámica y tiene un coste de $O(n)$.
 - (b) Es posible resolver el problema en tiempo logarítmico mediante divide y vencerás.
 - (c) La solución más eficiente tiene un coste $O(n^2)$.
 - (d) Ninguna de las anteriores es cierta.

Muy importante el dato de que V está ordenado y contiene enteros no repetidos

Ejercicio de examen

2. Se tiene un vector V de números enteros no repetidos y ordenados de menor a mayor. Se desea comprobar si existe algún elemento del vector que coincida con su índice ($V[i]=i$). Indica cuál de las siguientes afirmaciones es cierta:
- (a) La solución más eficiente se obtiene mediante programación y dinámica y tiene un coste de $O(n)$.
 - (b) Es posible resolver el problema en tiempo logarítmico mediante divide y vencerás.
 - (c) La solución más eficiente tiene un coste $O(n^2)$.
 - (d) Ninguna de las anteriores es cierta.

Muy importante el dato de que V está ordenado y contiene enteros no repetidos:

- Podemos aplicar la misma estrategia que para la búsqueda binaria

Ejercicio de examen

6. Dado el siguiente algoritmo:

```
// Precondición: i pertenece a {1,2,3}
hanoi(n,i,j) {
    si n=1 entonces escribe "Mover de " i "hasta" j
    sino {
        hanoi(n-1, i , 6-i-j)
        hanoi(1 , i , j)
        hanoi(n-1, 6-i-j, j)
    }
}
```

El coste asintótico temporal pertenece al orden:

- (a) $O(2n)$
- (b) $O(2^n)$
- (c) $O(\log_2 n)$
- (d) $O(n^2)$

Ejercicio de examen

6. Dado el siguiente algoritmo:

```
// Precondición: i pertenece a {1,2,3}
hanoi(n,i,j) {
    si n=1 entonces escribe "Mover de " i "hasta" j
    sino {
        hanoi(n-1, i , 6-i-j)
        hanoi(1 , i , j)
        hanoi(n-1, 6-i-j, j)
    }
}
```

El coste asintótico temporal pertenece al orden:

- (a) $O(2n)$
- (b) $O(2^n)$
- (c) $O(\log_2 n)$
- (d) $O(n^2)$

a = ?
b = ?
k = ?

Ejercicio de examen

6. Dado el siguiente algoritmo:

```
// Precondición: i pertenece a {1,2,3}
hanoi(n,i,j) {
    si n=1 entonces escribe "Mover de " i "hasta" j
    sino {
        hanoi(n-1, i , 6-i-j)
        hanoi(1 , i , j)
        hanoi(n-1, 6-i-j, j)
    }
}
```

El coste asintótico temporal pertenece al orden:

-
- (a) $O(2n)$
 - (b) $O(2^n)$
 - (c) $O(\log_2 n)$
 - (d) $O(n^2)$

$a = 2$ (el caso $n=1$ es siempre trivial)
 $b = 1$ (progresión aritmética: $n-1$)
 $k = 0$

Ordenación por fusión (Mergesort)

fun Fusionar (U:vector [1..n+1] de entero,V:vector [1..m+1] de entero,T:vector [1..m+n] de entero)

var
 i,j: natural

fvar

i,j \leftarrow 1

$U[n+1], V[m+1] \leftarrow \infty$

para $k \leftarrow 1$ **hasta** $m+n$ **hacer**

si $U[i] < V[j]$

entonces $T[k] \leftarrow U[i]$

$i \leftarrow i + 1$

sino $T[k] \leftarrow V[j]$

$j \leftarrow j + 1$

fsi

fpara

ffun

fun Mergesort (T: vector [1..n] de entero): vector [1..n] de entero

var

 U: vector [1..n] de entero, V: vector [1..n] de entero

fvar

si trivial(n) **entonces** Insertar(T[1..n])

sino

$U[1..1 + \lfloor n/2 \rfloor] \leftarrow T[1..\lfloor n/2 \rfloor]$

$V[1..1 + \lceil n/2 \rceil] \leftarrow T[1 + \lceil n/2 \rceil..n]$

 Mergesort(U)

 Mergesort(V)

 Fusionar(U,V,T)

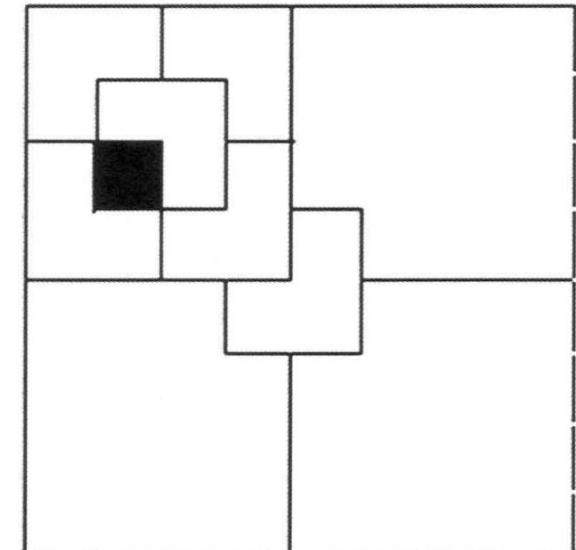
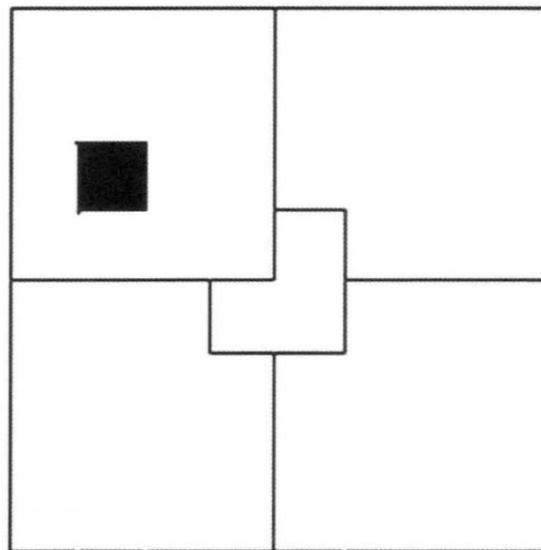
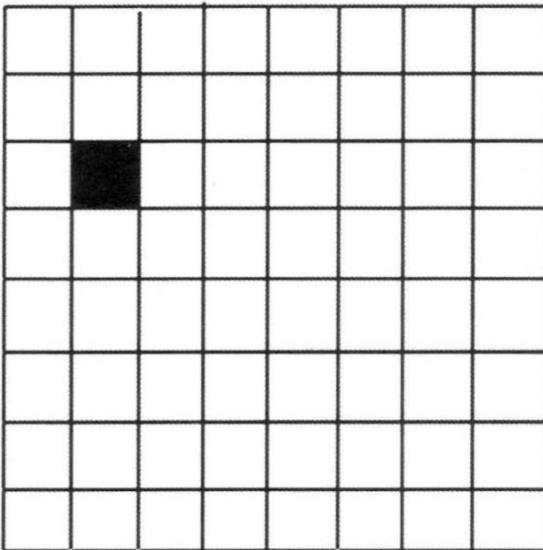
fsi

ffun

$$T(n) = 2T(n/2) + cn \quad \text{con } a = b = 2 \text{ y } k = 1, \text{ es decir, } t(n) \in \Theta(n \log n).$$

Puzzle tromino

- Objetivo: llenar la cuadrícula de trominos sin solaparlos y cubriendola totalmente salvo la casilla marcada
- Método DyV (n debe ser potencia de 2):
 - Dividir en 4 partes y colocar un tromino solapado con los cuadrantes sin la casilla marcada, utilizando las partes del tromino como casillas marcadas en la nueva iteración
 - Caso base: cuadrante de 2x2 con una casilla marcada



Pu

fun Tromino(*T*: matriz [1..n,1..n] de natural, *n,m*: natural)

var

T₁, T₂, T₃, T₄ : matriz [1..n,1..n] de natural

fvar

- O
- C
- M

si *n* = 2 **entonces**

T \leftarrow colocaTromino(*T,m*)

dev *T*

sino

m' \leftarrow esquina cuadrante con casilla negra

colocaTromino(*T,m'*)

T₁, T₂, T₃, T₄ \leftarrow dividir *T* en 4 cuadrantes

tromino(*T₁, n/2, m₁*)

tromino(*T₂, n/2, m₂*)

tromino(*T₃, n/2, m₃*)

tromino(*T₄, n/2, m₄*)

fsi

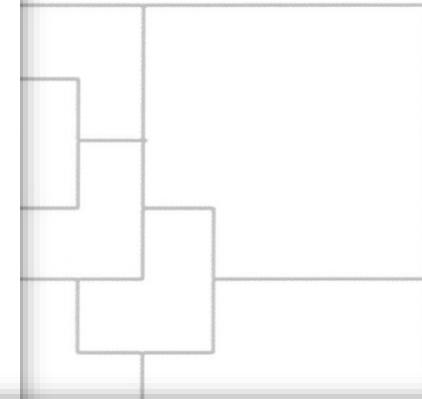
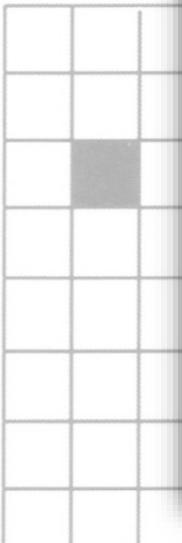
T \leftarrow combinar(*T₁, T₂, T₃, T₄*)

dev *T*

ffun

- **T**: tablero
- **n**: tamaño del tablero
- **m**: posición de la casilla marcada

o como casillas



$$T(n) = 4T(n/2) + c, \text{ con } b = 2, a = 4 \text{ y } k = 0$$

cuando $a > b^k$ es $\Theta(n^{\log_b a})$, es decir $\Theta(n^2)$

Ordenación rápida (Quicksort)

- Proceso:
 - Toma un elemento cualquiera del vector denominado **pivote**
 - Normalmente el primero
 - Toma los valores del vector que son menores que el pivote y forma un subvector, e igual con los valores mayores o iguales
 - Una forma es recorrer de izq a dcha hasta encontrar un elemento mayor que el pivote y de dcha a izq hasta encontrar uno menor, e intercambiarlos, deteniéndonos al cruzarnos
 - Se invoca recursivamente al algoritmo para cada subvector

	Mergesort	Quicksort
Descomposición	Trivial	No trivial
Combinación	No trivial	Trivial

Ordenación rápida (Quicksort)

```
fun Pivotar (T:vector [i..j] de entero)
    var
        p,k:entero
    fvar
        Añadir:
        , pivot: natural
    p ← T[i]
    k ← i; l ← j + 1
    repetir k ← k + 1 hasta T[k] > p ∨ k ≥ j
    repetir l ← l - 1 hasta T[l] ≤ p
    mientras k < l hacer
        intercambiar(T,k,l)
        repetir k ← k + 1 hasta T[k] > p
        repetir l ← l - 1 hasta T[l] ≤ p
    fmientras
    intercambiar(T,i,l)
ffun
```

```
fun Quicksort (T[i..j])
    si trivial(j-i) entonces Insertar(T[i..j])
    sino
        Pivotar(T[i..j],l);
        Quicksort(T[i..l - 1]);
        Quicksort(T[l + 1..j])
    fsi
ffun
```

Añadir:
var
l:natural
fvar

Coste: O(n^2)
Pero el mejor en el caso promedio

5	6	9	3	4	1
5	1	9	3	4	6
5	1	4	3	9	6
3	1	4	5	9	6

Ejercicio de examen

5. En relación a los algoritmos de ordenación, ¿cuál de las siguientes afirmaciones es **verdadera**?
- (a) La ordenación mediante el algoritmo Heapsort tiene un coste $O(\log n)$.
 - (b) El coste del algoritmo Quicksort en el caso peor es de orden $O(n \log n)$.
 - (c) El coste del algoritmo de ordenación mergesort es $O(n)$.
 - (d) Todas las anteriores son falsas.

Ejercicio de examen

5. En relación a los algoritmos de ordenación, ¿cuál de las siguientes afirmaciones es **verdadera**?
- (a) La ordenación mediante el algoritmo Heapsort tiene un coste $O(\log n)$.
 - (b) El coste del algoritmo Quicksort en el caso peor es de orden $O(n \log n)$.
 - (c) El coste del algoritmo de ordenación mergesort es $O(n)$.
 - (d) Todas las anteriores son falsas.
- 

Ejercicio de examen

3. Se desea implementar el algoritmo de ordenación rápida (quicksort) para aplicarlo a vectores que están casi ordenados. A la hora de elegir el elemento pivote para la partición de los subvectores, ¿Cuál sería la elección más adecuada para este caso concreto?
 - a. El primer elemento del subvector.
 - b. El último elemento del subvector.
 - c. El elemento que se encuentra en la posición media del subvector.
 - d. La elección del elemento pivote no influye en el rendimiento del algoritmo.

Ejercicio de examen

3. Se desea implementar el algoritmo de ordenación rápida (quicksort) para aplicarlo a vectores que están casi ordenados. A la hora de elegir el elemento pivote para la partición de los subvectores, ¿Cuál sería la elección más adecuada para este caso concreto?
- a. El primer elemento del subvector.
 - b. El último elemento del subvector.
 - c. El elemento que se encuentra en la posición media del subvector.
 - d. La elección del elemento pivote no influye en el rendimiento del algoritmo.
- 

Ejercicio de examen

4. ¿Cuál de las siguientes afirmaciones es falsa?

- (a) El algoritmo de ordenación por fusión (*mergesort*) es $O(n \log n)$.
- (b) La eficiencia del algoritmo de ordenación rápida (*quicksort*) es independiente de que el pivote sea el elemento de menor valor del vector.
- (c) El algoritmo de ordenación rápida en el caso peor es $O(n^2)$.
- (d) El algoritmo de ordenación basada en montículos (*heapsort*) es $O(n \log n)$.

Ejercicio de examen

4. ¿Cuál de las siguientes afirmaciones es falsa?

- (a) El algoritmo de ordenación por fusión (*mergesort*) es $O(n \log n)$.
- (b) La eficiencia del algoritmo de ordenación rápida (*quicksort*) es independiente de que el pivote sea el elemento de menor valor del vector.
- (c) El algoritmo de ordenación rápida en el caso peor es $O(n^2)$.
- (d) El algoritmo de ordenación basada en montículos (*heapsort*) es $O(n \log n)$.

Cálculo del elemento mayoritario

fun Mayoritario(i,j:natural;v:vector [1..n] de natural): entero

var

 m:natural

 s₁,s₂:entero

fvar

si i = j **entonces**

dev v[i]

sino

 m \leftarrow (i + j) \div 2

 s₁ \leftarrow Mayoritario(i,m,v)

 s₂ \leftarrow Mayoritario(m+1,j,v)

dev Combinar(s₁,s₂,v)

ffun

$$T(n) = 2T(n/2) + cn$$

$$a = b = 2 \text{ y } k = 1$$

$$t(n) \in \Theta(n \log n)$$

fun Combinar (a,b:entero;v:vector [1..n] de natural):entero

si a = -1 \wedge b = -1 **entonces dev** -1 **fsi**

si a = -1 \wedge b \neq -1 **entonces dev** ComprobarMayoritario(b,v) **fsi**

si a \neq -1 \wedge b = -1 **entonces dev** ComprobarMayoritario(a,v) **fsi**

si a \neq -1 \wedge b \neq -1 **entonces**

si ComprobarMayoritario(a,v) = a **entonces dev** a

sino si ComprobarMayoritario(b,v) = b **entonces dev** b **fsi**

fsi
fun

fun ComprobarMayoritario (x:natural;v:vector [1..n] de natural):entero

var

 c:natural

Añadir parámetros i y j

fvar

 c \leftarrow 1

para k \leftarrow 1 **hasta** n **hacer**

si v[k]=x **entonces** c \leftarrow c + 1 **fsi**

fpara

si c > (j - i + 1) / 2 **entonces dev** c **sino dev** -1 **fsi**

ffun

Liga de equipos

- $n = 2^k$ equipos crean una liga con las siguientes condiciones:
 - cada equipo puede jugar un partido al día
 - la liga debe celebrarse en $n-1$ días
 - se dispone de suficientes campos de juego

- Elementos:

- Caso trivial:
 - sólo dos equipos e_i y e_j juegan en $n-1 = 2-1 = 1$ días (el primero) $\rightarrow T[i, j] = d$
- Descomponer:
 - si el rango inicial es $[1..n]$, la partición sería $[1..n/2]$ y $[n/2+1..n]$
 - generalizando: con $[i..j]$ y $m = (i+j-1)/2$ la partición es $[i..m]$ y $[m+1..j]$
- Combinar:
 - suponemos que los conjuntos de equipos $c_A = \{e_1, \dots, e_{n/2}\}$ y $c_B = \{e_{n/2+1}, \dots, e_n\}$ han jugado ya todos entre sí en los primeros $n/2-1$ días
 - los equipos de c_A y c_B deben jugar en los $n/2$ días restantes, empezando el día $n/2$ jugando e_1 con $e_{n/2+1}$, e_2 con $e_{n/2+2}$, ..., $e_{n/2}$ con e_n , al día siguiente jugando e_1 con $e_{n/2+2}$, e_2 con $e_{n/2+3}$, ..., $e_{n/2}$ con $e_{n/2+1}$ y así sucesivamente

	e_1	e_2	...	e_n
e_1	-	3	...	1
e_2	3	-	...	1
...
e_n	1	4		-

Liga de equipos

fun Torneo(i,j,d:natural)

var

 m,n: natural;

fvar

 m \leftarrow (i+j-1)/2;

 n \leftarrow j-i+1;

si n = 2 **entonces**

 T[i,j] \leftarrow d;

sino

 Torneo(i,m,d)

 Torneo(m+1,j,d)

 Combinar(i,j,d)

fsi

ffun

 día n/2 jugando e₁ con e_{n/2+1}, e₂ con e_{n/2+2}, ..., e_{n/2} con e_n, al dia siguiente

$$T(n) = 2T(n/2) + cn^2 \text{ con } a = 2, b = 2 \text{ y } k = 2 \text{ con lo que siendo } a < b^k \rightarrow O(n^2)$$

fun Combinar(i,j,d:natural)

var

 m,n,s,t: natural

 a,b: natural

fvar

 m \leftarrow (i+j-1)/2

 n \leftarrow j-i+1

para s \leftarrow 0 **hasta** n/2-1 **hacer**

para t \leftarrow 0 **hasta** n/2-1 **hacer**

 a \leftarrow i+t

 b \leftarrow m+1+((t+s) **mod** (n/2))

 T[a,b] \leftarrow s+d+n/2-1

fpara

fpara

ffun

Ejercicio de examen

5. El problema de la liga de equipos consiste en que dados n equipos con $n = 2^k$, se desea realizar una liga de forma que cada equipo puede jugar un partido al día y la liga debe celebrarse en $n-1$ días, suponiendo que existen suficientes campos de juego. El objetivo sería realizar un calendario que indique el día que deben jugar cada par de equipos. Si este problema se resuelve con un esquema divide y vencerás, considerando que el caso trivial se da cuando la liga consta de 2 equipos, la descomposición se realiza dividiendo el problema en dos partes similares, y la combinación se produce dando los subproblemas por resueltos y aplicando el principio de inducción. Indica de entre los siguientes, cuál sería el coste mínimo de dicho algoritmo.
- (a) $\Theta(n)$.
 - (b) $\Theta(n^2)$.
 - (c) $\Theta(n \log n)$.
 - (d) $\Theta(n^2 \log n)$.

Ejercicio de examen

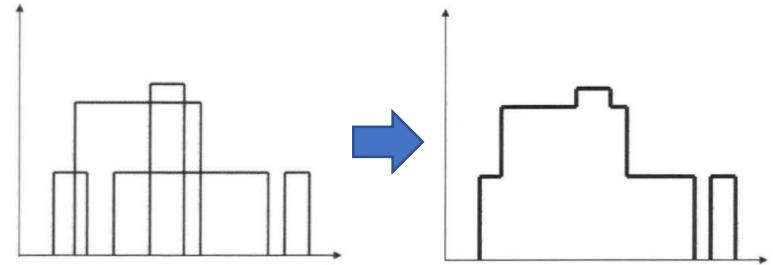
5. El problema de la liga de equipos consiste en que dados n equipos con $n = 2^k$, se desea realizar una liga de forma que cada equipo puede jugar un partido al día y la liga debe celebrarse en $n-1$ días, suponiendo que existen suficientes campos de juego. El objetivo sería realizar un calendario que indique el día que deben jugar cada par de equipos. Si este problema se resuelve con un esquema divide y vencerás, considerando que el caso trivial se da cuando la liga consta de 2 equipos, la descomposición se realiza dividiendo el problema en dos partes similares, y la combinación se produce dando los subproblemas por resueltos y aplicando el principio de inducción. Indica de entre los siguientes, cuál sería el coste mínimo de dicho algoritmo.

- 
- (a) $\Theta(n)$.
 - (b) $\Theta(n^2)$.
 - (c) $\Theta(n \log n)$.
 - (d) $\Theta(n^2 \log n)$.

$$T(n) = 2T(n/2) + cn^2 \text{ con } a = 2, b = 2 \text{ y } k = 2 \text{ con lo que siendo } a < b^k \rightarrow O(n^2)$$

Skyline de una ciudad

- Elementos:
 - Caso trivial:
 - realizar el skyline de un edificio
 - Descomponer:
 - el conjunto de edificios se divide en dos mitades iguales
 - Combinar:
 - la entrada a este algoritmo son dos soluciones (2 líneas de horizonte) y se fusionan eligiendo la mayor ordenada para cada abcisa donde haya edificios
- Estructuras de datos
 - Edificios: triadas del tipo $e_1 = (x_1, x_2, h)$ (inicio, final, y altura)
 - TipoSkyline: concatenación de posiciones y alturas en esas posiciones, $s = (x_1, h_1, x_2, h_2, \dots, x_k, h_k)$ en donde cada par x_i, h_i representa transiciones entre un edificio y otro, o bien entre un edificio y la linea de horizonte

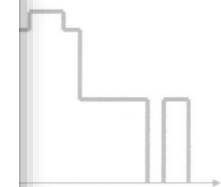


Skyline de una ciudad

```
fun Edificios(C: vector [1..n] de edificio; i,j:natural): TipoSkyline
    var
        m,n: natural
    fvar
        m  $\leftarrow$  (i+j-1)/2
        n  $\leftarrow$  j-i+1
        si  $n = 1$  entonces
            dev convierte_edificio_en_skyline(C[i])
        sino
             $s_1 \leftarrow$  Edificios(C,i,m)
             $s_2 \leftarrow$  Edificios(C,m+1,j)
             $s \leftarrow$  Combinar( $s_1, s_2$ )
            dev s
        fsi
    ffun
```

• El resultado es un skyline que se divide en dos partes: la parte izquierda (s_1) y la parte derecha (s_2). La parte izquierda (s_1) es el skyline de los edificios entre la posición i y la mitad ($(i+j-1)/2$). La parte derecha (s_2) es el skyline de los edificios entre la mitad ($(i+j-1)/2$) y la posición j . Si $n = 1$, el resultado es el skyline de un solo edificio en la posición i .

• Es importante notar que el resultado es un skyline, lo que significa que es una lista de posiciones (x_i) y alturas (h_i) que representan las transiciones entre los edificios y la línea de horizonte.



izonte)
nde

a)

;

, h_i

posiciones, $s = (x_1, h_1, x_2, h_2, \dots, x_k, h_k)$ en donde cada par x_i, h_i representa transiciones entre un edificio y otro, o bien entre un edificio y la linea de horizonte

fun Combinar(s1,s2: TipoSkyline)

var

i,j,k: natural

fvar

n \leftarrow s1.longitud()

m \leftarrow s2.longitud()

$s1_x \leftarrow$ ExtraeAbscisa(s1)

$s1_h \leftarrow$ ExtraeAlturas(s1)

$s2_x \leftarrow$ ExtraeAbscisa(s2)

$s2_h \leftarrow$ ExtraeAlturas(s2)

i \leftarrow 1; j \leftarrow 1

k \leftarrow 1; S \leftarrow []

mientras ($i \leq n$) \vee ($j \leq m$) **hacer**

x \leftarrow min($s1_x[i], s2_x[j]$)

si $s1_x[i] < s2_x[j]$ **entonces**

max \leftarrow max($s1_h[i], s2_h[j - 1]$)

i \leftarrow i+1

sino

~~Entradas: trazos del tipo~~

- TipoSkyline: concatenación de posiciones, s = ($x_1, h_1, x_2, h_2, \dots$) representa transiciones entre la silueta de un edificio y la linea de horizonte

dad

si $s1_x[i] > s2_x[j]$ **entonces**

max \leftarrow max($s1_h[i - 1], s2_h[j]$)

j \leftarrow j+1

sino

max \leftarrow max($s1_h[i], s2_h[j]$)

i \leftarrow i+1

j \leftarrow j+1

fsi

fsi

$S_x[k] \leftarrow x$

$S_h[k] \leftarrow \text{max}$

k \leftarrow k+1

fmientras

$S \leftarrow S_x \cup S_h$

dev S

ffun

Ejercicio de examen

Problema (4 puntos).

Se tienen 3 palos verticales y n discos agujereados por el centro. Los discos son todos de diferente tamaño y en la posición inicial están insertados en el primer palo ordenados en tamaños en sucesión decreciente desde la base hasta la altura. El problema consiste en pasar los discos del 1^{er} al 3^{er} palo, utilizando el segundo como auxiliar, observando las siguientes reglas:

- a) Se mueven los discos de 1 en 1.
- b) Nunca un disco puede colocarse encima de uno menor que éste.

La resolución de este problema debe incluir, por este orden:

1. Elección del esquema más apropiado, el esquema general y explicación de su aplicación al problema (0,5 puntos).
2. Algoritmo completo a partir del refinamiento del esquema general (3 puntos solo si el punto 1 es correcto). Si se trata del esquema voraz debe hacerse la demostración de optimalidad.
3. Estudio del coste del algoritmo desarrollado (0.5 puntos solo si el punto 1 es correcto).

“Tower of Hanoi”



The program for solving this problem would be:

```
void main()
{
    int n = 4;                                /* Number of discs = 4 */
    hanoi('A','B','C', n);
}

void hanoi(char From, char To, char Other, int n)
{
    if (n == 0) return;
    hanoi(From, Other, To, n-1);
    printf("Moving disc from peg %c to %c\n", From, To);
    hanoi(Other, To, From, n-1);
}
```

The output of the program (which moves 4 discs) would be:

```
Moving disc from peg A to C
Moving disc from peg A to B
Moving disc from peg C to B
Moving disc from peg A to C
Moving disc from peg B to A
Moving disc from peg B to C
Moving disc from peg A to C
Moving disc from peg A to B
Moving disc from peg C to B
Moving disc from peg C to A
Moving disc from peg B to A
Moving disc from peg C to B
Moving disc from peg A to C
Moving disc from peg A to B
Moving disc from peg C to B
```



Ejercicio de examen

Problema (4 puntos).

Un vector de n números naturales tiene un “elemento mayoritario” si hay al menos $n/2 + 1$ ocurrencias de dicho elemento en el vector. Diseñar y programar un algoritmo que compruebe esta propiedad con el menor coste posible.

NOTA: No se podrán usar estructuras de datos adicionales.

La resolución del problema debe incluir, por este orden:

1. Elección del esquema más apropiado, el esquema general y explicación de su aplicación al problema (0,5 puntos)
2. Descripción de las estructuras de datos necesarias (0,5 puntos solo si el punto 1 es correcto)
3. Algoritmo completo a partir del refinamiento del esquema general (2,5 puntos solo si el punto 1 es correcto)
4. Estudio del coste del algoritmo desarrollado (0,5 puntos solo si el punto 1 es correcto)

Ejercicio de examen

Problema (4 puntos).

Una caja con n bombones se considera “aburrida” si se repite un mismo tipo de bombón (por ejemplo, el bombón de “praliné”) más de $n/2$ veces. Programar un algoritmo que decida si una caja es “aburrida” y devuelva (en su caso) el tipo de bombón que le confirme dicha propiedad.

La resolución del problema debe incluir, por este orden:

1. Elección del esquema más apropiado, el esquema general y explicación de su aplicación al problema (0,5 puntos)
2. Descripción de las estructuras de datos necesarias (0,5 puntos solo si el punto 1 es correcto)
3. Algoritmo completo a partir del refinamiento del esquema general (2,5 puntos solo si el punto 1 es correcto). Si se trata del esquema voraz, debe realizarse la demostración de optimalidad. Si se trata del esquema de programación dinámica, deben proporcionarse las ecuaciones de recurrencia.
4. Estudio del coste del algoritmo desarrollado (0,5 puntos solo si el punto 1 es correcto)

Ejercicio de examen

Problema (4 puntos). Sea $V[1..N]$ un vector con la votación de unas elecciones. La componente $V[i]$ contiene el nombre del candidato que ha elegido el votante i . Se pide escribir un algoritmo que decida si algún candidato aparece en más de la mitad de las componentes (tiene mayoría absoluta) y que devuelva su nombre. Sirva como ayuda que para que un candidato tenga mayoría absoluta considerando todo el vector (al menos $N/2+1$ de los N votos), es condición necesaria pero no suficiente que tenga mayoría absoluta en alguna de las mitades del vector.

La resolución de este problema debe incluir, por este orden:

1. Elección razonada del esquema más apropiado de entre los siguientes: Voraz, Divide y Vencerás, Vuelta Atrás o Ramificación y Poda. Escriba la estructura general de dicho esquema e indique cómo se aplica al problema (0,5 puntos).
2. Descripción de las estructuras de datos necesarias (0,5 puntos solo si el punto 1 es correcto).
3. Algoritmo completo a partir del refinamiento del esquema general (2,5 puntos sólo si el punto 1 es correcto). Si se trata del esquema voraz debe hacerse la demostración de optimalidad.
4. Estudio del coste del algoritmo desarrollado (0,5 puntos solo si el punto 1 es correcto).

Ejercicio de examen

Problema (4 puntos). Resolver el problema de cálculo de la subsecuencia de mayor valor de un vector de enteros: Dado un vector $a[1..n]$ de enteros, se pide diseñar un algoritmo eficiente – mejor que $O(n^2)$ – que encuentre la subsecuencia de suma máxima dentro del vector.

$$\max_{1 \leq i \leq j \leq n} \left\{ \sum_{k=i}^j a_k \right\}$$

Por ejemplo, en el vector $a = [-2, 11, -4, 13, -5, -2]$ la solución al problema es $i=2$ y $j=4$, con resultado $a_2 + a_3 + a_4 = 20$.

La resolución de este problema debe incluir, por este orden:

1. Elección razonada del esquema más apropiado de entre los siguientes: Voraz, Divide y Vencerás, Vuelta Atrás o Ramificación y Poda. Escriba la estructura general de dicho esquema e indique cómo se aplica al problema (0,5 puntos).
2. Descripción de las estructuras de datos necesarias (0,5 puntos solo si el punto 1 es correcto).
3. Algoritmo completo a partir del refinamiento del esquema general (2.5 puntos sólo si el punto 1 es correcto). Si se trata del esquema voraz debe hacerse la demostración de optimalidad.
4. Estudio del coste del algoritmo desarrollado (0,5 puntos solo si el punto 1 es correcto).

Ejercicio de examen

Problema (4 puntos). Resolver el problema de cálculo de la subsecuencia de mayor valor de

```
PROCEDURE Sumamax(VAR a:vector;prim,ult:CARDINAL):CARDINAL;
  VAR izq,der,i:CARDINAL; max_aux,suma:INTEGER;
BEGIN
  max_aux:=0;
  FOR izq:=prim TO ult DO
    FOR der:=izq TO ult DO
      suma:=0;
      FOR i:=izq TO der DO
        suma:=suma+a[i];
      END;
      IF suma>max_aux THEN
        max_aux:=suma;
      END;
    END;
  END;
  RETURN max_aux
END Sumamax;
```

Solución trivial de
 $O(n^3) \rightarrow$ NO sirve

calcular todas las
posibles sumas
("fuerza bruta")

Ej

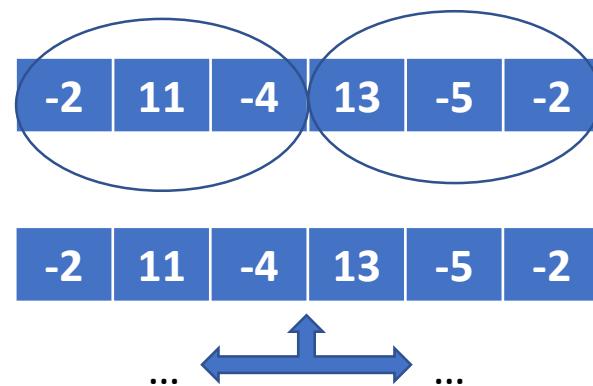
Probl
un ve
mejor

Por ej
resulta

La res

```
PROCEDURE Sumamax3(VAR a:vector;prim,ult:CARDINAL):CARDINAL;
  VAR mitad,i:CARDINAL;
      max_izq,max_der,suma,max_aux:INTEGER;
BEGIN
  (* casos base *)
  IF prim>ult THEN RETURN 0 END;
  IF prim=ult THEN RETURN Max2(0,a[prim]) END;
  mitad:=(prim+ult)DIV 2;
  (* casos 1 y 2 *)
  max_aux:=Max2(Sumamax3(a,prim,mitad),Sumamax3(a,mitad+1,ult));
  (* caso 3: parte izquierda *)
  max_izq:=0;
  suma:=0;
  FOR i:=mitad TO prim BY -1 DO
    suma:=suma+a[i];
    max_izq:=Max2(max_izq,suma)
  END;
  (* caso 3: parte derecha *)
  max_der:=0;
  suma:=0;
  FOR i:=mitad+1 TO ult DO
    suma:=suma+a[i];
    max_der:=Max2(max_der,suma)
  END;
  (* combinacion de resultados *)
  RETURN Max2(max_der+max_izq,max_aux)
END Sumamax3;
```

Podemos conseguir
una solución $O(n^2)$
con PD pero con
 $DyV O(n \log n)$



Resumen de ejemplos

- Búsqueda binaria en vector ordenado
 - Buscar en el centro y si no es seguir buscando en la mitad correcta
- Ordenación por fusión (Mergesort)
 - Dividir, ordenar, fusionar
- Puzzle tromino (con tablero de tamaño potencia de 2)
 - Dividir en 4 y colocar tromino en el centro orientado hacia la marca
- Ordenación rápida (Quicksort)
 - Pivatar(dividir según valores) y repetir con subvectores
- Cálculo del elemento mayoritario
 - Comprobación por mitades + combinación
- Liga equipos
 - División por 2 hasta tener 2 equipos + combinación (ventana deslizante)
- Skyline de una ciudad
 - División por 2 hasta tener 1 edificio + combinación (mayor ordenada)