

PREDA - UNED

Programación y Estructuras de Datos Avanzadas

Ordenación por fusión (Mergesort)

- Estrategia:
 - Dividir sucesivamente el vector en dos mitades hasta que su ordenación sea trivial y luego fusionar
- Ordenación trivial:
 - Ordenación por inserción: Se recorre el vector 1 a 1 insertando el elemento en el sitio que le corresponda a su izquierda (la parte izquierda está ordenada y se van añadiendo los de la derecha 1 a 1)
- Combinación:
 - Recorrer ambos subvectores ordenados seleccionando para la solución el menor en cada caso, moviendo el puntero solo en el subvector del elemento seleccionado

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

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

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

Mergesort(U)

Mergesort(V)

Fusionar(U, V, T)

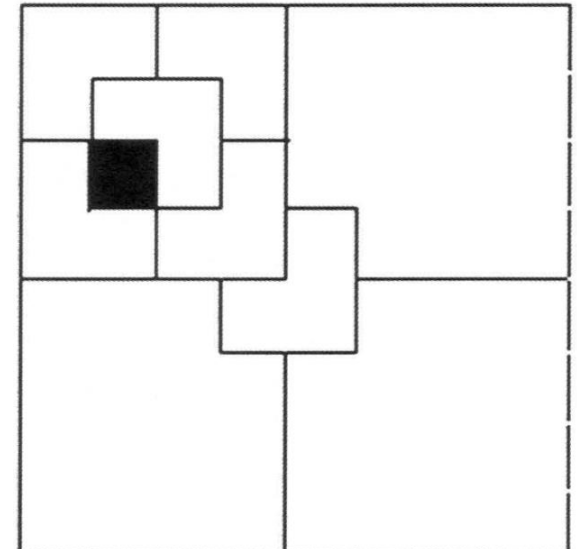
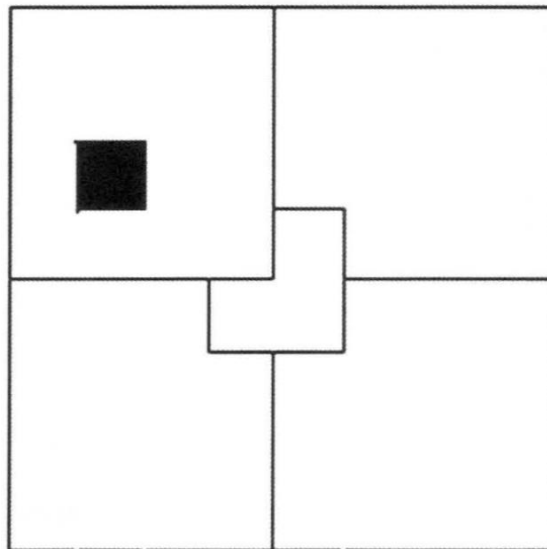
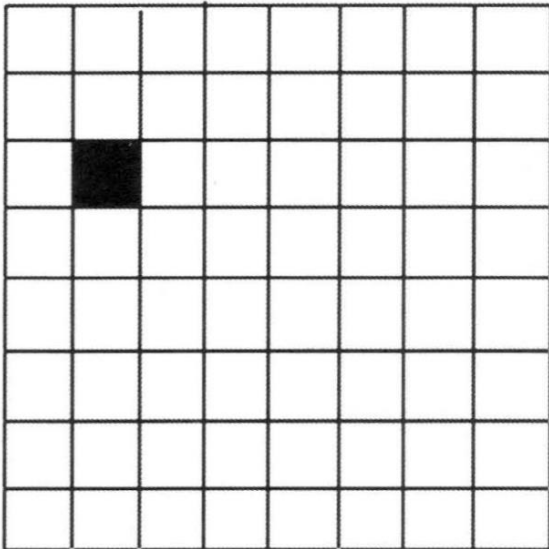
fsi

ffun

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

Puzzle tromino

- Objetivo: rellenar la cuadrícula de trominos sin solaparlos y cubriéndola 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



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

var

T_1, T_2, T_3, T_4 :matriz [1..n,1..n] de natural

fvar

si $n = 2$ **entonces**

$T \leftarrow \text{colocaTromino}(T,m)$

dev T

sino

$m' \leftarrow \text{esquina cuadrante con casilla negra}$

$\text{colocaTromino}(T,m')$

$T_1, T_2, T_3, T_4 \leftarrow \text{dividir } T \text{ en 4 cuadrantes}$

$\text{tromino}(T_1, n/2, m_1)$

$\text{tromino}(T_2, n/2, m_2)$

$\text{tromino}(T_3, n/2, m_3)$

$\text{tromino}(T_4, n/2, m_4)$

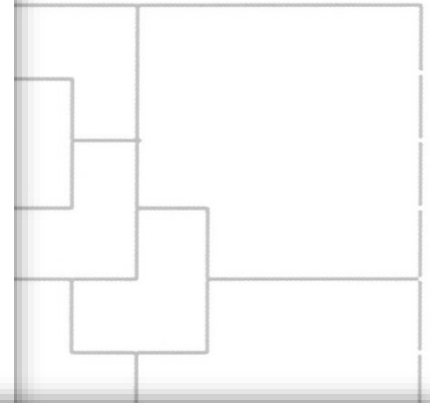
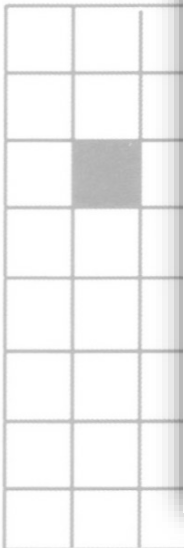
fsi

$T \leftarrow \text{combinar}(T_1, T_2, T_3, T_4)$

dev T

ffun

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



$T(n) = 4T(n/2) + c$, con $b = 2$, $a = 4$ y $k = 0$
cuando $a > b^k$ es $\Theta(n^{\log_b a})$, es decir $\Theta(n^2)$

Ordenación rápida (Quicksort)

	Mergesort	Quicksort
Descomposición	Trivial	No trivial
Combinación	No trivial	Trivial

- 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

Ordenación rápida (Quicksort)

```
fun Pivotar (T:vector [i..j] de entero)
  var
    p,k:entero
  fvar
     $p \leftarrow T[i]$ 
     $k \leftarrow i; l \leftarrow j + 1$ 
    repetir  $k \leftarrow k + 1$  hasta  $T[k] > p \vee k \geq j$ 
    repetir  $l \leftarrow l - 1$  hasta  $T[l] \leq p$ 
    mientras  $k < l$  hacer
      intercambiar(T,k,l)
      repetir  $k \leftarrow k + 1$  hasta  $T[k] > p$ 
      repetir  $l \leftarrow l - 1$  hasta  $T[l] \leq p$ 
    fmientras
      intercambiar(T,i,l)
ffun
```

Añadir:
, pivote: natural

Añadir:
pivote $\leftarrow 1$

```
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

<u>5</u>	6	9	3	4	1
<u>5</u>	1	9	3	4	6
<u>5</u>	1	4	3	9	6
3	1	4	<u>5</u>	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 $O(n \log n)$.

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 (sin ida-vuelta)
 - 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 (dentro de sus grupos) 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	-	...	4
...
e_n	1	4		-

Liga de equipos

Combinar:

• n

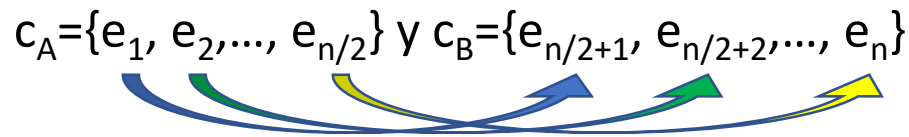
a) Partimos de $c_A = \{e_1, \dots, e_{n/2}\}$ y $c_B = \{e_{n/2+1}, \dots, e_n\}$

Ya han jugado entre sí Ya han jugado entre sí

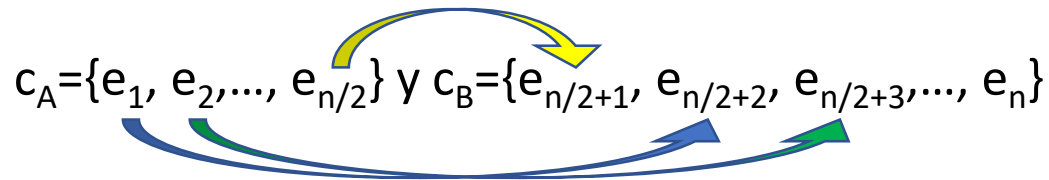
• E

b) Ahora creamos los partidos entre equipos de c_A y c_B :

- días 1..n-1: partidos anteriores dentro de c_A y c_B , ver a)
- día n



- día n+1



...

es:

..	e_n
..	1
..	1 4
..	...
..	-

$j] = d$

$e_n\}$ han
días
o el día
jugando

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
```

```
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
```

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


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

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.

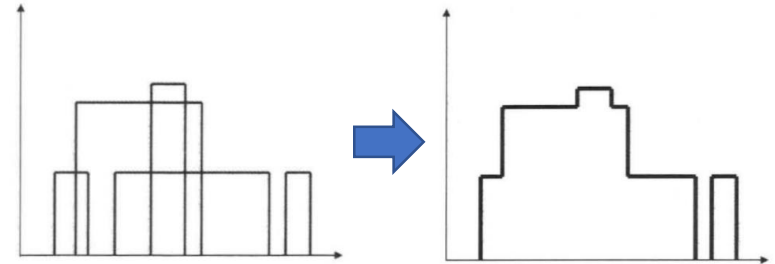
- 
- (a) $\Theta(n)$.
 - (b) $\Theta(n^2)$.
 - (c) $\Theta(n \log n)$.
 - (d) $\Theta(n^2 \log n)$.

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

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 abscisa 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 línea 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

$s_1 \leftarrow \text{Edificios}(C,i,m)$

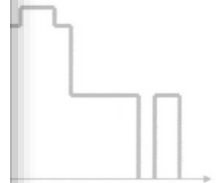
$s_2 \leftarrow \text{Edificios}(C,m+1,j)$

$s \leftarrow \text{Combinar}(s_1,s_2)$

dev s

fsi

ffun



izonte)
nde

a)

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 línea 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

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 max

k \leftarrow k+1

fmientras

S \leftarrow S_x \cup S_h

dev S

ffun

- TipoSkyline: concatenación de posiciones, s = (x₁, h₁, x₂, h₂, ...) representa transiciones en un edificio y la línea de horizonte

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

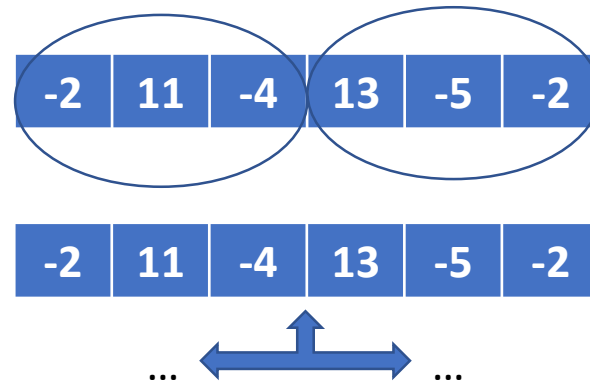
calcular todas las
posibles sumas
("fuerza bruta")

```

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)
 - Pivotar(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)