Ejercicio 1. Divide y vencerás

[ITIS/ITIG, Septiembre 2008] Se dispone de un vector de secuencias de vídeo *video*[1..*N*] en el que cada elemento contiene dos atributos: *duración* (en segundos) y *secuencia* (contiene el fragmento de vídeo). Se quiere ordenar este vector en orden creciente de *duración*.

Sin embargo, no es posible utilizar directamente uno de los algoritmos de ordenación conocidos, ya que las secuencias de vídeo almacenadas en el atributo secuencia ocupan gran cantidad de memoria y se quiere evitar mover estos datos de forma innecesaria durante el proceso de ordenación.

Diseña un algoritmo basado en la técnica **Divide y Vencerás** que proporcione un vector ordenado de secuencias de vídeo, de forma que realice una sola copia de los datos contenidos en el atributo *secuencia* de cada elemento del vector de origen *video*[1..*N*] al vector ordenado. Detalla lo siguiente:

- (0,5 puntos) las estructuras de datos utilizadas
- (2 puntos) el código **completo** del algoritmo

Solución ejercicio 1

```
proc videos(video[1..n],sol[1..n])
   crear vp[1..n]
   //vp[i] tiene dos atributos: duracion y la posicion en la que se encuentra en video[i]
   desde i ← 1 hasta n hacer
      vp[i].duracion ← video[i].duracion ; vp[i].pos ← i
   fin desde
   mergesort(vp)
   desde i ← 1 hasta n hacer
      sol[i] ← video[vp[i].pos]
   fin desde
fin proc
```

Solución ejercicio 1 (Cont.)

```
proc mergesort(vp[1..n])
   h ← ⌊ n/2 ⌋ ; m ← n-h
   crear U[1..h], V[1..m]
   // U[ ] y V[ ] tienen la misma estructura que vp[ ]
   si n>1 entonces
      U[1..h] ← vp[1..h]
      V[1..m] ← vp[h+1..n]
      mergesort(U)
      mergesort(V)
      combinar(U,V,S)
   fin si
fin proc
```

Solución ejercicio 1 (Cont.)

```
proc combinar(U[1..h],V[1..m],S[1..h+m])
   i ← 1 ; j ← 1 ; k ← 1
   mientras i ≤ h Y j ≤ m hacer
      si U[i].duracion < V[j].duracion entonces
         S[k] ← U[i] ; i ← i+1
      si no
         S[k] ← V[j] ; j ← j+1
      fin si
      k ← k+1
   fin mientras
   si i>h entonces S[k..h+m] ← V[j..m]
   si no S[k..h+m] ← U[i..h]
fin proc
```

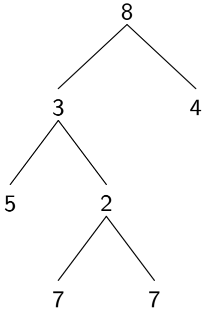
Ejercicio 2. Divide y vencerás

[ITIS/ITIG, Junio 2008] Un montículo ascendente es un árbol binario en el que cada nodo es mayor que sus hijos (si existen). Consideremos un montículo de m nodos con valores positivos implementado mediante un vector de la siguiente manera. Dado un nodo que se encuentra en la posición i del vector:

- El hijo izquierdo (si existe) está en la posición $2i$.
- El hijo derecho (si existe) está en la posición $2i + 1$.
- Si la posición i corresponde a un hijo inexistente, esta posición contiene el valor 0.

Diseña una función utilizando la técnica *Divide y Vencerás* que determine si un árbol binario implementado mediante un array de P elementos contiene un montículo ascendente.

Ejemplo de implementación de un árbol binario mediante un array:



Array:

8	3	4	5	2	0	0	0	0	7	7

Solución ejercicio 2

```
fun chequear_monticulo(i,v[1..P])
  monticulo ← cierto
  si 2i ≤ P entonces
    si (v[i]>0 ∧ v[i]>v[2i]) ∨ (v[i]=0 ∧ v[2i]=0) entonces
      monticulo ← chequear_monticulo(2i,v)
    si no
      monticulo ← falso
    fin si
    si monticulo ∧ 2i+1 ≤ P entonces
      si (v[i]>0 ∧ v[i]>v[2i+1]) ∨ (v[i]=0 ∧ v[2i+1]=0) entonces
        monticulo ← chequear_monticulo(2i+1,v)
      si no
        monticulo ← falso
      fin si
    fin si
  fin si
  devolver monticulo
fin fun
```

Ejercicio 3. Divide y vencerás

[ITIS/ITIG, Junio 2007] Dada una matriz cuadrada M, cuya dimensión es potencia de 2, diseña un algoritmo que calcule su traspuesta M^t, mediante la técnica Divide y Vencerás.

Ayuda:

Puede ser útil utilizar la siguiente convención: $M[i..j, k..l]$ es la submatriz obtenida al considerar las filas que van desde la i hasta la j y las columnas que van desde la k hasta la l.

Notas:

- Se puede considerar que la función que devuelve la parte entera de un número está implementada: $ent(x)$.
- Se puede utilizar el operador asignación ($\longleftarrow$) para realizar operaciones entre arrays y/o estructuras.

Solución ejercicio 3

fun Trasponer(M[i..j, k..l])
    si j-1 = i **entonces**
        // tamaño 2x2
        aux ← M[i,l] ; M[i,l] ← M[j,k] ; M[j,k] ← aux
    si no
        crear $\mathrm{M}_{aux}[1..\frac{j-i}{2},1..\frac{l-k}{2}]$
        $\mathrm{M}_{aux} \leftarrow \mathrm{M}[\ ent(\frac{i+j}{2})+1..j, k..ent(\frac{k+l}{2})]$
        $\mathrm{M}[ent(\frac{i+j}{2})+1..j,k..ent(\frac{k+l}{2})] \leftarrow \mathrm{M}[\ i..ent(\frac{i+j}{2}),ent(\frac{k+l}{2})+1..l]$
        $\mathrm{M}[\ i..ent(\frac{i+j}{2}),ent(\frac{k+l}{2})+1..l] \leftarrow \mathrm{M}_{aux}$
        Trasponer($\mathrm{M}[i..ent(\frac{i+j}{2}),k..ent(\frac{k+l}{2})]$)
        Trasponer($\mathrm{M}[ent(\frac{i+j}{2})+1..j,k..ent(\frac{k+l}{2})]$)
        Trasponer($\mathrm{M}[i..ent(\frac{i+j}{2}),ent(\frac{k+l}{2})+1..l]$)
        Trasponer($\mathrm{M}[ent(\frac{i+j}{2})+1..j,ent(\frac{k+l}{2})+1..l]$)
    fin si
fin fun

Solución ejercicio 3 (Cont.)

- Si la matriz M es de la forma:

$$M = \begin{pmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{pmatrix}$$

- Las líneas 9, 10 y 11 producen el siguiente resultado

$$M' = \begin{pmatrix} M_{11} & M_{21} \\ M_{12} & M_{22} \end{pmatrix}$$

Ejercicio 4. Divide y vencerás

[GV00], p. 136. **Subsecuencia de suma máxima.** Dados *n* enteros cualesquiera $a_1, a_2, \ldots, a_n$, mecesitamos calcular el valor de la expresión:

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

que calcula el máximo de las sumas parciales de elementos consecutivos. Como ejemplo, dados 6 números enteros

$$(-2, \quad 11, \quad -4, \quad 13, \quad -5, \quad -2)$$

la solución al problema es 20 (suma de a_2 hasta a_4). Deseamos implementar un algoritmo Divide y Vencerás de complejidad *nlogn* que resuelva el problema. ¿Existe algún otro algoritmo que lo resuelva en menos tiempo?

Solución ejercicio 4

- Para resolver este problema aplicando la técnica Divide y Vencerás, seguiremos la siguiente idea:
- Dividir el problema en tres subproblemas más pequeños, sobre cuyas soluciones construiremos la solución final.
- La subsecuencia de suma máxima puede encontrarse en tres lugares:
  1. En la primera mitad del vector
  2. En la segunda mitad del vector
  3. Contiene el punto medio y se encuentra en ambas mitades

Solución ejercicio 4

- Para resolver este problema aplicando la técnica Divide y Vencerás, seguiremos la siguiente idea:
- Dividir el problema en tres subproblemas más pequeños, sobre cuyas soluciones construiremos la solución final.
- La subsecuencia de suma máxima puede encontrarse en tres lugares:
  1. En la primera mitad del vector
  2. En la segunda mitad del vector
  3. Contiene el punto medio y se encuentra en ambas mitades
- Los casos 1. y 2. se resuelven recursivamente (max_1, max_2)
- Respecto al tercer caso:
  - se calcula la subsecuencia máxima de la primera mitad que contine a su último elemento (aux_max_1)
  - se calcula la subsecuencia máxima de la segunda mitad que contine a su primer elemento (aux_max_2)
  - concatenar las dos secuencias anteriores con el elemento central para construir la subsecuencia máxima que contiene al elemento central ($max_3 = aux\_max_1 + aux\_max_2$)
- Solución $= max\{max_1, max_2, max_3\}$

Solución ejercicio 4

- Si tenemos la lista:

-2	11	-4	13	-5	-2

- Dividimos en 2

-2	**11**	-4

$max_1 = 11\ (a_2 \longrightarrow a_2)$

13	-5	-2

$max_2 = 13\ (a_4 \longrightarrow a_4)$

Solución ejercicio 4

- Si tenemos la lista:

-2	11	-4	13	-5	-2

- Dividimos en 2

-2	**11**	-4

$max_1 = 11 \ (a_2 \rightarrow a_2)$

13	-5	-2

$max_2 = 13 \ (a_4 \rightarrow a_4)$

- Calculamos la subsecuencia máxima que contiene al último y al primero de las dos mitades

-2	**11**	**-4**

$aux\_max_1 = 7 \ (a_2 \rightarrow a_3)$

13	-5	-2

$aux\_max_2 = 13 \ (a_4 \rightarrow a_4)$

- Así, $max_3 = aux\_max_1 + aux\_max_2 = 7 + 13 = 20 \ (a_2 \rightarrow a_4)$
- Solución : maximo $\{max_1, max_2, max_3\} = 20 \ (a_2 \rightarrow a_4)$

Solución ejercicio 4

- El algoritmo resultante es:

fun SumaMax(V[1..n], prim, ult)
  si prim = ult **entonces devolver** V[prim]
  mitad $\leftarrow \lfloor$ (prim + ult) / 2 $\rfloor$
  $max_1 \leftarrow$ SumaMax(V, prim , mitad) /* primer caso */
  $max_2 \leftarrow$ SumaMax(V, mitad+1 , ult) /* segundo caso */
  suma $\leftarrow$ 0 , $aux\_max_1 \leftarrow -\infty$ /* tercer caso */
  desde i=mitad **hasta** prim **hacer**
    suma $\leftarrow$ suma + V[i]
    $aux\_max_1 \leftarrow$ maximo $\{suma, aux\_max_1\}$
  fin desde
  suma $\leftarrow$ 0 , $aux\_max_2 \leftarrow -\infty$
  desde i=mitad+1 **hasta** ult **hacer**
    suma $\leftarrow$ suma + V[i]
    $aux\_max_2 \leftarrow$ maximo $\{suma, aux\_max_2\}$
  fin desde
  devolver maximo $\{max_1, max_2, aux\_max_2 + aux\_max_1\}$
fin fun

Solución ejercicio 4

- Análisis de complejidad:
  El tiempo de ejecución $T(n)$ del algoritmo anterior viene dado por la ecuación de recurrencia:
  $$T(n) = \begin{cases} 7 & si\ \ n = 1 \\ 2 \cdot T(n/2) + C \cdot n & si\ \ n > 1 \end{cases}$$
  donde C es una constante

Ejercicio 5. Divide y vencerás

[GV00], p. 110. **Búsqueda ternaria.** Se trata de decidir si existe un elemento dado X en un vector de enteros. Plantear un algoritmo con la siguiente estrategia: En primer lugar comparar el elemento dado X con el elemento que se encuentra en la posición $n/3$ del vector. Si este es menor que el elemento X, entonces lo compara con el elemento que se encuentra en la posición $2n/3$, y si no coincide con X, busca recursivamente en el correspondiente subvector de tamaño $1/3$ del original.

- ¿Qué complejidad tiene este algoritmo?
- Compáralo con el de búsqueda binaria.

Solución ejercicio 5

- El algoritmo resultante es:

```
fun busqueda_ternaria(V[1..n], x, inf, sup)
   si inf >= sup entonces devolver V[sup]= x
   terc ← ⌊ (sup - inf +1) / 3 ⌋
   si x = V[inf+terc] entonces
      devolver cierto
   si no si x < V[inf+terc] entonces
      devolver busqueda_ternaria(V, x, inf, inf+terc-1)
   si no si  x = V[sup-terc]  entonces
      devolver cierto
   si no si  x < V[sup-terc]  entonces
      devolver busqueda_ternaria(V, x, inf+terc, sup-terc)
   si no
      devolver busqueda_ternaria(V, x, sup-terc+1, sup)
   fin si
fin fun
```

Solución ejercicio 5 (Cont.)

- Análisis de la complejidad de este algoritmo en el caso peor
- El caso peor puede ocurrir cuando X es mayor a cualquier elemento de la lista
- En cada llamada recursiva, el tamaño del vector es un tercio del tamaño del vector en la llamada anterior
- Suponiendo que n es una potencia de 3, la complejidad del algoritmo es

$$W(n) = \begin{cases} 4 & si \;\; n = 1 \\ W(n/3) + 23 & si \;\; n > 1 \end{cases}$$

- Resolviendo esta ecuación de recurrencia, nos queda $W(n) = 23 log_3 n + 4 \in \Theta(lg \;\; n)$

Ejercicio 6. Divide y vencerás

[ITIG, Febrero 2008] En una instalación de producción de energía eólica se quieren sustituir algunos de los generadores existentes por otros de última generación.
La instalación está formada por una serie de generadores formando una hilera. Para reducir los costes de la sustitución, sólo puede sustituirse un conjunto de generadores consecutivos de la hilera.
Con el fin de amortizar lo antes posible la inversión, se pretende sustituir la mayor secuencia consecutiva de generadores que en el pasado han resultado en conjunto más rentables (ya que algunos no reciben suficiente viento y son deficitarios –con rentabilidad negativa). A tal fin, se dispone de la información de rentabilidad de cada uno de los generadores en el último año.

- Diseña mediante la técnica **divide y vencerás** un algoritmo eficiente que proporcione la secuencia de generadores consecutivos que en conjunto sean más rentables (2.5 puntos).
- Estudia la complejidad temporal del algoritmo proporcionado (0.5 puntos).

Solución Ejercicio 6

- Este problema es similar al ejercicio de la subsecuencia de suma máxima visto en clase
- La única diferencia es que, además de la suma, debemos devolver la secuencia. Para ello, basta con devolver los elementos inicial y final
- Para resolver este problema utilizando la técnica *divide y vencerás*, se puede subdividir el vector en dos, y calcular la subsecuencia de suma máxima de cada una de las dos partes
- la subsecuencia del vector original puede estar en cualquiera de las dos partes, o bien entre las dos
- Una versión sencilla del algoritmo es la siguiente:

Solución ejercicio 6 (cont.)

```
fun summax(A[ini..fin],iniMax,finMax)
  si ini=fin entonces
    iniMax ← ini ; finMax ← fin ; suma ← A[ini]
    devolver suma
  fin si
  mitad ← ⌊ ini + fin / 2 ⌋
  suma1 ← summax(A[ini..mitad],iniMax1,finMax1)
  suma2 ← summax(A[mitad+1..fin],iniMax2,finMax2)
  suma3 ← summitad(A[ini..fin],mitad,iniMax3,finMax3)
  si suma1 > suma2 entonces
    si suma1 > suma3 entonces
      iniMax ← iniMax1 ; finMax ← finMax1 ; return  suma1
    si no
      iniMax ← iniMax3 ; finMax ← finMax3 ; return  suma3
    fin si
  si no
    si suma2 > suma3 entonces
      iniMax ← iniMax2 ; finMax ← finMax2 ; return  suma2
    si no
      iniMax ← iniMax3 ; finMax ← finMax3 ; return  suma3
    fin si
  fin si
fin fun
```

Solución ejercicio 6 (cont.)

```
fun summitad(A[ini..fin],mitad,iniMax3,finMax3)
   suma3i ← −∞ ; suma3d ← −∞
   suma_aux ← 0 ; iniMax3 ← mitad ; finMax3 ← mitad
   desde i ← mitad hasta ini sumando -1 hacer
      suma_aux ← suma_aux + A[i]
      si suma_aux ≥ suma3i entonces
         suma3i ← suma_aux ; iniMax3 ← i
      fin si
   fin desde
   suma_aux ← 0
   desde i ← mitad+1 hasta fin hacer
      suma_aux ← suma_aux + A[i]
      si suma_aux ≥ suma3d entonces
         suma3d ← suma_aux ; finMax3 ← i
      fin si
   fin desde
   devolver suma3i + suma3d
fin fun
```

- La complejidad de summitad() está en $\Theta(n)$
- Por el teorema de reducción por división (caso $a = b^k$), la complejidad de summax() está en $\Theta(n\ lg\ n)$
- Se podría hacer de complejidad lineal, como se indica en [GV00].