# Ejercicio 1. Examen Febrero 2009 (Ing. Inf.)

**Ejercicio 1 (1,2 puntos).**
Se conocen dos algoritmos recursivos para resolver determinado problema que tiene como entrada un vector:

1. El primer algoritmo divide los datos de entrada en tres vectores de igual tamaño, realiza una llamada recursiva sobre cada uno de estos vectores, y por último realiza un proceso sobre los datos de entrada con coste cuadrático.
2. El segundo algoritmo divide los datos de entrada en dos vectores iguales, realiza una serie de llamadas recursivas sobre estos vectores, y finalmente realiza un proceso sobre los datos de entrada con coste lineal.

Estudiando la complejidad de ambos algoritmos, se ha comprobado que el primero está en un orden de complejidad asintótica $\Theta$ menor que el segundo. ¿Cuántas llamadas recursivas realiza al menos el segundo algoritmo? Razona la respuesta.

# Solución ejercicio 1. Examen Febrero 2009 II

- Las recurrencias que expresan la complejidad de ambos problemas son:
  - $T_1(n) = 3T_1(n/3) + p_1(n)$, donde $p_1(n) \in \Theta(n^2)$
  - $T_2(n) = KT_2(n/2) + p_2(n)$, donde $p_2(n) \in \Theta(n)$
- Por tanto, aplicando el teorema de la división, tenemos que $T_1(n) \in \Theta(n^2)$.
- Para que $T_2(n)$ tenga un orden de complejidad peor que el de $T_1(n)$, podemos aplicar el mismo teorema, con el siguiente resultado:
  - Si $K < 2$, $T_2(n) \in \Theta(n)$
  - Si $K = 2$, $T_2(n) \in \Theta(n \ log \ n)$
  - Si $K > 2$, $T_2(n) \in \Theta(n^{log \ K})$
- El orden de complejidad es peor que el de $T_1(n)$ sólo si nos encontramos en el tercer caso.
- Para saber el valor exacto de K, debe ocurrir que $\Theta(n^2) \subset \Theta(n^{log \ K})$. Por tanto, $K$ debe ser mayor a 4.

# Ejercicio 2. Examen Febrero 2009 (Ing. Inf.)

**Ejercicio 2 (2,5 puntos).**
En un lejano país, los estudios de Master se cursan únicamente por la Universidad a distancia. Estos estudios constan de un total de N asignaturas, y cada alumno ha de aprobar al menos M de ellas ($M \leq N$) para obtener el título de Master. En cuanto al sistema de evaluación, se tienen las siguientes reglas:

- Para aprobar una asignatura hay que hacer un examen presencial escrito en una de las sedes de la universidad.
- Los exámenes se celebran una sola vez al año, en una única semana, conociendo el día y hora de comienzo, así como la duración del examen.
- Para poder examinarse de una asignatura, es obligatorio:
  a) registrarse, por lo que se pide a los alumnos que entren en el aula correspondiente unos minutos antes de la hora de comienzo del examen.
  b) permanecer en el aula hasta que termine el examen.

# Examen febrero 2009 II (cont.)

**Ejercicio 2 (cont.)**

Suponiendo conocidos los datos de cada examen (fecha y hora de comienzo, duración y tiempo que se tarda en registrarse):

- (0,5 puntos) desarrolla una estrategia voraz que ayude a un estudiante a decidir a qué exámenes va a poder asistir, teniendo en cuenta que el objetivo es examinarse del máximo número de asignaturas para aumentar la probabilidad de obtener el título de Master en un curso académico.
- (1,5 puntos) Diseña el algoritmo que implemente esta estrategia.
- (0,5 puntos) Demuestra que la estrategia es óptima.

# Solución ejercicio 2. Examen febrero 2009 II

- Este problema es muy parecido al ejercicio 3 de la hoja de ejercicios propuestos y el ejercicio 12.8 que aparece en [MOV04].
- La diferencia más importante es que hay que tener en cuenta el tiempo de inscripción.
- La estrategia óptima sería elegir primero el examen que termine antes y que no se solape con el anterior.
- La entrada del algoritmo es un array de N exámenes, donde cada elemento tiene tres atributos:
  - Hora de inicio
  - duración
  - Tiempo de inscripción
- La solución vendrá dada por un array $Sol[1, \ldots, n_1]$, con $n_1 \leq N$ donde $Sol[i]$ es el índice del examen que el alumno ha de realizar en el i-esimo lugar.

# Solución ejercicio 2. Examen febrero 2009 II (cont.)

```
proc Master(E[1..n], Sol[1..n])
   crear F[1..n], I[1..n] //vectores de hora fin y de índices
   desde i ← 1 hasta n hacer F[i] ← E[i].ini+E[i].duracion ; I[i] ← i
   Ordenar(E,I) //ordena los índices de I de forma creciente respecto a E
   Sol[1] ← I[1]
   desde i ← 2 hasta n hacer Sol[i] ← 0
   j ← 2, i ← 1
   mientras j ≤ N hacer
      si E[I[j]].ini - E[I[j]].inscrip > E[Sol[i]].ini + E[Sol[i]].duracion entonces
         Sol[i+1] ← E[j]
         i ← i+1
      fin si
      j ← j+1
   fin mientras
fin proc
```

# Solución ejercicio 2. Examen febrero 2009 II (cont.)

- Para demostrar que esta estrategia es óptima, vamos a comparar la solución que nos da este algoritmo, $Sol_x = x_1, x_2, \ldots, x_t$ con cualquier otra solución, $Sol_y = y_1, y_2, \ldots, y_s$.
- Sean $E[a].ini$ y $E[a].fin$ la hora de inicio y de fin de del examen a, respectivamente (la hora de fin se puede calcular fácilmente como la hora de inicio más la duración).
- Suponemos que los exámenes están ordenadas por hora de fin.
- Debe demostrarse que $t \geq s$.
- Supongamos que los primeros elementos de ambas secuencias son iguales. Sea $k$ el primer elemento de ambas secuencias tal que $x_k \neq y_k$.
- Por la forma de seleccionar los exámenes del algoritmo voraz, $E[x_k].fin \leq E[y_k].fin$.
- Podemos demostrar por inducción que esto ocurre para todo $j = k, k+1, \ldots, \text{mín}(s,t)$, $E[x_j].fin \leq E[y_j].fin$:

# Solución ejercicio 2. Examen febrero 2009 II (cont.)

- **caso base:** acabamos de ver que se cumple para $j = k$.
- **hipótesis de inducción:** se cumple para $m > k$.
- **paso de inducción:** si $E[x_m].fin \leq E[y_m].fin$, entonces el siguiente exámen que elige el algoritmo voraz, $x_{m+1}$ será el que acabe primero y que no se solape con $x_m$,
  $E[x_m].fin \leq E[x_{m+1}].ini - E[x_{m+1}].inscrip < E[x_{m+1}].fin$.
- Además, el siguiente examen de la selección $Y$ también debe cumplir que $E[y_m].fin \leq E[y_{m+1}].ini - E[y_{m+1}].inscrip$.
- Por tanto,
  $E[x_m].fin \leq E[y_m].fin \leq E[y_{m+1}].ini - E[y_{m+1}].inscrip < E[y_{m+1}].fin$,
  y entonces $E[x_{m+1}].fin \leq E[y_{m+1}].fin$, puesto que el algoritmo voraz elige el examen que termina antes, y si no existe ningún examen adecuado que termine antes que $y_{m+1}$, el algoritmo puede elegir éste.

# Solución ejercicio 2. Examen febrero 2009 II (cont.)

- Por tanto, esto también se cumple para $u = \text{mín}(s, t)$: $E[x_u].fin \leq E[y_u].fin$.
- Vamos a ver que $u = s$ (y por tanto $s \leq t$) utilizando reducción al absurdo:
  - Supongamos que $s > t$. En este caso, $E[y_t].fin \leq E[y_{t+1}].ini - E[y_{t+1}].inscrip$
  - Por la estrategia voraz elegida, el algoritmo voraz hubiera elegido también a $y_{t+1}$, puesto que $E[x_t].fin \leq E[y_t].fin \leq E[y_{t+1}].ini - E[y_{t+1}].inscrip$.
  - Por tanto, $X$ tiene más de $t$ elementos, lo que es una contradicción.

# Ejercicio 3. Examen Febrero 2009 (Ing. Inf.)

**Ejercicio 3 (2 puntos).**
Un aparcacoches dispone de un trozo de carretera de longitud $L$ para aparcar $N$ posibles coches. Por cada coche que logre aparcar cobrará "la voluntad" a su dueño. Idealmente las distancias entre coches aparcados puede ser cero. Suponiendo conocidas las longidudes de los coches (en metros) y las cantidades que cobra por aparcar cada coche, determina qué coches debe aparcar para lograr el mayor beneficio. Detalla lo siguiente:

- (0,5 puntos) Justifica adecuadamente la metodología que resuelve el problema con una menor complejidad.
- (1,5 puntos) Especifica el procedimiento y/o función que resuelva el problema.

# Solución ejercicio 3. Examen febrero 2009 II

- Este problema es similar al problema de la mochila 0-1.
- En el tema de algoritmos voraces vimos un algoritmo para resolver el problema de la mochila, pero no proporciona resultados óptimos en el caso de la mochila 0-1.
- Se podría resolver utilizando un algoritmo de *backtracking*, pero su complejidad es exponencial sobre el número de vehículos, y por ello puede ser mucho menos eficiente que un algoritmo de programación dinámica.
- Por tanto, lo más adecuado es utilizar la técnica de programación dinámica.
- Además, hay que tener en cuenta que debe devolverse la lista de vehículos seleccionados.
- La expresión recurrente es:

$$C[i,j] = \begin{cases} 0 & si\ j < 0 \\ 0 & si\ i = 1\ y\ j < L_i \\ c_i & si\ i = 1\ y\ j \geq L_i \\ \max(C[i-1,j], c_i + C[i-1,j-L_i]) & \text{en otro caso} \end{cases}$$

# Solución ejercicio 3. Examen febrero 2009 II (cont.)

**proc** coches(Long[1..N], Cant[1..N], L, Coches)
  **crear** C[1..N,0..L]
  **desde** i $\leftarrow$ 1 **hasta** N **hacer** $C[i,0] \leftarrow 0$
  **desde** j $\leftarrow$ 1 **hasta** N **hacer**
    **desde** l $\leftarrow$ 1 **hasta** L **hacer**
      **si** $j = 1 \wedge l < Long[1]$ **entonces** $C[j,l] \leftarrow 0$
      **si no si** $j = 1$ **entonces** $C[j,l] \leftarrow Cant[j]$
      **si no si** $l < Long[j]$ **entonces** $C[j,l] \leftarrow C[j-1,l]$
      **si no si** $C[j-1,l] \geq Cant[j] + C[j-1,l-Long[j]]$ **entonces**
        $C[j,l] \leftarrow C[j-1,l]$
      **si no**
        $C[j,l] \leftarrow Cant[j] + C[j-1,l-Long[j]]$
      **fin si**
    **fin desde**
  **fin desde**
  seleccion(C,Long,Coches)
**fin proc**

# Solución ejercicio 3. Examen febrero 2009 II (cont.)

```
proc seleccion(C[1..N,0..L],Long[1..N],Coches[1..N])
   c ← 1
   i ← n
   j ← Cal
   mientras i > 0 Y j > 0 hacer
      si D[i,j]=D[i-1,j] entonces
         i ← i-1
      si no
         Coches[c] ← i
         c ← c + 1
         j ← j-Long[i]
         i ← i-1
      fin si
   fin mientras
   desde i ← c hasta N hacer m[i] ← -1
fin proc
```

- También se puede utilizar un algoritmo de programación dinámica que genere los elementos seleccionados (en azul en el algoritmo siguiente) a la vez que rellena la matriz intermedia.

# Solución ejercicio 3. Examen febrero 2009 II (cont.)

**proc** coches(Long[1..N], Cant[1..N], L, Coches)
  **crear** C[1..N,0..L], S[1..N,0..L]
  **desde** i $\leftarrow$ 1 **hasta** N **hacer** $C[i,0] \leftarrow 0$ ; $S[i,0] \leftarrow \emptyset$
  **desde** j $\leftarrow$ 1 **hasta** N **hacer**
    **desde** l $\leftarrow$ 1 **hasta** L **hacer**
      **si** $j = 1 \wedge l < Long[1]$ **entonces** $C[j,l] \leftarrow 0$ ; $S[j,l] \leftarrow \emptyset$
      **si no si** $j = 1$ **entonces** $C[j,l] \leftarrow Cant[j]$ ; $S[j,l] \leftarrow j$
      **si no si** $l < Long[j]$ **entonces** $C[j,l] \leftarrow C[j-1,l]$ ; $S[j,l] \leftarrow S[j-1,l]$
      **si no si** $C[j-1,l] \geq Cant[j] + C[j-1, l-Long[j]]$ **entonces**
        $C[j,l] \leftarrow C[j-1,l]$ ; $S[j,l] \leftarrow S[j-1,l]$
      **si no**
        $C[j,l] \leftarrow Cant[j] + C[j-1, l-Long[j]]$
        $S[j,l] \leftarrow j \cup S[j-1, l-Long[j]]$
      **fin si**
    **fin desde**
  **fin desde**
  $Coches \leftarrow S[N,L]$
**fin proc**

# Ejercicio 4. Examen Febrero 2009 (Ing. Inf.)

**Ejercicio 4 (2,5 puntos)**.

Una empresa de autobuses realiza recorridos turísticos por los N pueblos de una región. Algunos de estos pueblos tienen un monumento de interés turístico mientras otros no. Se quieren realizar recorridos turísticos que no pasen dos veces por el mismo pueblo, pero que al final del viaje regresen al punto de partida. Son conocidas las distancias entre pueblos así como los pueblos que tienen monumento.

Diseña un algoritmo de vuelta atrás recursivo que calcule la ruta que permita visitar **el mayor número de pueblos con un monumento**, sin superar el número máximo de kilómetros KMAX que se pueden realizar en un día. Detalla lo siguiente:

1. (0,5 puntos) Declaración de tipos y/o variables para representar la información del problema y el árbol de búsqueda: significado de las aristas y de los niveles.
2. (1,75 puntos) El código del procedimiento.
3. (0,25 puntos) El programa llamador.

# Solución ejercicio 4. Examen febrero 2009 II

```
proc bus(D[1..n,1..n],M[1..n],KMAX,origen,vis[1..n],sol[1..n],numOpt,solAct[1..n],
     numActIni,kmActIni,etapa)
  desde v ← 1 hasta n hacer
    solAct[etapa] ← v
    si D[solAct[etapa-1],solAct[etapa]]< ∞ ∧ ¬ vis[v] entonces
      kmAct ← kmActIni + D[solAct[etapa-1],solAct[etapa]]
      si kmAct < KMAX entonces
        vis[v] ← cierto
        si M[v] entonces numAct ← numActIni+1
        si kmAct + D[solAct[etapa],origen] ≤ KMAX entonces
          si numAct > numOpt entonces numOpt ← numAct ; sol ← solAct
        fin si
        bus(D,M,KMAX,origen,vis,sol,numOpt,solAct,numAct,kmAct,etapa+1)
        vis[v] ← falso
      fin si
    fin si
  fin desde
  solAct[etapa] ← -1
fin proc
```

# Solución ejercicio 4. Examen febrero 2009 II (cont.)

**fun** llamador_bus(D[1..n,1..n],M[1..n],KMAX,origen,sol[1..n])
  **crear** solAct[1..n], vis[1..n]
  solAct[1] ← origen
  vis[origen] ← cierto
  **desde** i ← 2 **hasta** n **hacer** vis[i] ← falso ; solAct[i] ← -1
  numMon ← $-\infty$
  bus(D,M,KMAX,origen,vis,sol,numMon,solAct,0,0,2)
  **devolver** n umMon
**fin fun**