UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA	**Programación III** Solución	**Prueba Presencial** **Segunda Semana** Enero - Febrero de 2005 Duración: **2 horas** Material permitido: **NINGUNO**

Cuestión 1 (2 puntos). Demuestra cuál es el orden exacto de complejidad en función de la variable n, del siguiente programa:

```
1       procedimiento lona (n: entero, j: entero)
2           para i desde 1 hasta n div 4 hacer
3               j:=j+i;
4           fpara
5           si n>1 entonces
6               lona(n-2,j);
7               escribir "j";
8               lona(n-2,n-j);
9           fsi
```

Planteamos una recurrencia con reducción del problema mediante sustracción

n: tamaño del problemas
a: número de llamadas recursivas
b: reducción del problema en cada llamada recursiva
n-b: tamaño del subproblema
n^k: coste de las instrucciones que no son llamadas recursivas

Recurrencia:

$$T(n)=\begin{cases} c\cdot n^k & \text{si } 0\le n<b \\ a\cdot T(n-b)+c\cdot n^k & \text{si } n\ge b \end{cases}$$

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

$a=2$; $b=2$; $n^k=n$; $k=1$

Por tanto, $T(n)\in \Theta(2^{n \text{ div } 2})$

Nota: $\Theta(2^{n \text{ div } 2})$ no es equivalente a $\Theta(2^n)$, puesto que se le está aplicando la raíz cuadrada y no el producto de una constante.

La resolución del problema debe incluir, por este orden:

1. Elección razonada del esquema algorítmico
2. Descripción del esquema usado e identificación con el problema
3. Estructuras de datos
4. Algoritmo completo a partir del refinamiento del esquema general
5. Estudio del coste

Cuestión 2 (1 punto). ¿En qué se diferencia una búsqueda ciega en profundidad y un esquema de vuelta atrás? Pon un ejemplo.

El esquema de vuelta atrás es una búsqueda en profundidad pero en la que se articula un mecanismo para detener la búsqueda en una determinada rama (poda). Para alcanzar una solución final es necesario que los pasos intermedios sean soluciones parciales al problema. Si un determinado nodo no es una solución parcial, entonces la solución final no se puede alcanzar a partir de dicho nodo y la rama se poda. Esto se implementa en el esquema de vuelta atrás mediante la función *condiciones_de_poda* o función de factibilidad.

El problema de colocar N reinas en un tablero de NxN sin que se amenacen entre sí es un problema que se puede resolver mediante vuelta atrás, puesto que en cada nivel de la búsqueda se establece un problema parcial: en el nivel i se trata de colocar i reinas en un tablero de NxN sin que se amenacen entre sí. Si un nodo no cumple esta condición, por muchas más reinas que se coloquen a continuación nunca se va a encontrar una solución final.

Cuestión 3 (2 puntos). Demuestra por inducción que el algoritmo de Dijkstra halla los caminos mínimos desde un único origen hasta todos los demás nodos del grafo.

(Solución en el libro de texto, Brassard, pág. 225).

Problema (5 puntos). Utiliza el esquema de divide y vencerás para implementar una función que tome un vector de enteros y le dé estructura de montículo con el menor coste posible.

1. Elección del esquema (0 puntos)

No es necesario razonar la elección del esquema puesto que viene impuesto en el problema.

2. Esquema general (0.5 puntos)

    1. Descomponer el ejemplar en subejemplares del mismo tipo que el ejemplar original
    2. Resolver independientemente cada subejemplar (subproblema)
    3. Combinar los resultados para construir la solución del ejemplar original

```
Función DivideVencerás(X)
    si suficiente_pequeño(X) entonces
            dev subalgoritmo_básico(X)
    si no
            Descomponer(X, X1..Xn)
            para i=1 hasta n hacer
                    Yi:=DivideVencerás(Xi);
            fpara
            Recombinar(Y1..Yn, Y)
            dev Y
    fsi
```

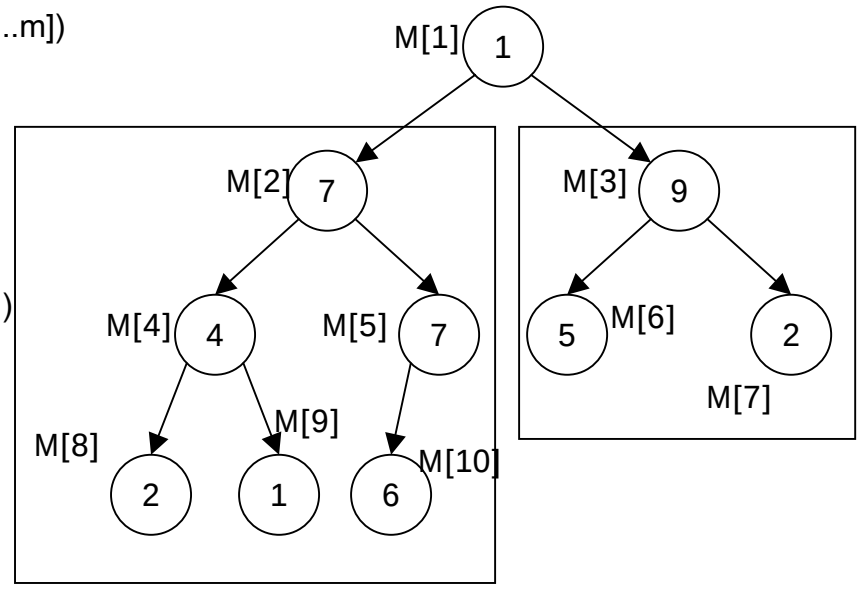
3. Estructuras de datos (0 puntos)

No se necesita ninguna estructura adicional a parte del vector de entrada.

4. Algoritmo completo (3 puntos)

Como se trata de un árbol binario, descomponemos el problema en los dos subárboles bajo el nodo i, es decir, el que tiene por raíz 2i y el que tiene por raiz 2i+1. Cada subárbol a su vez debe tener estructura de montículo lo que resulta un problema del mismo tipo que el original y, por tanto, corresponde a sendas llamadas recursivas, una para 2i y otra para 2i+1. Por último, hay que colocar la raíz i en su lugar. Para ello, hay que "hundirla".

```
Procedimiento crear_montículo(i, M[1..m])
    si 2i<m entonces
        crear_montículo(2i, M)
    fsi
    si (2i+1)<m entonces
        crear_montículo(2i+1, M)
    fsi
    si (2i≤m) entonces
        hundir(M, i)
    fsi
```



La llamada inicial será: *crear_montículo(1,M).*

Como puede observarse, dividir el vector en dos mitades M[1..m/2], M[(m/2)+1..m] y tratar de resolver cada una de ellas supone un error. Una solución que recorra completamente el vector y proceda a hundir o flotar cada elemento siempre tendrá mayor coste.

Ya que estamos en un esquema divide y vencerás, podemos plantear el procedimiento hundir como un problema de reducción. Únicamente se tiene que decidir si el nodo debe intercambiarse por alguno de sus hijos, y en caso afirmativo intercambiarse con el mayor de ellos y realizar la correspondiente llamada recursiva para seguir "hundiéndose".

```
Procedimiento hundir(T[1..n], i)
    hmayor:=i
    si (2i ≤ n) y (T[2i] > T[hmayor]) entonces
        hmayor=2i
```

```
fsi
si (2i < n) y (T[2i+1] > T[hmayor]) entonces
        hmayor=2i+1
fsi
si (hmayor > i) entonces
        intercambiar(T[i], T[hmayor])
        hundir(T[1..n], hmayor)
fsi
```

5. Estudio del coste (1.5 puntos)

El coste del procedimiento *hundir* se puede plantear mediante una recurrencia con reducción del problema por división, ya que *hundir* prosigue por uno de los dos subárboles y, por tanto, el problema se ha reducido a la mitad.

$$T(n)=\begin{cases} c\cdot n^k & \text{si } 0\le n<b \\ a\cdot T(n/b)+c\cdot n^k & \text{si } n\ge b \end{cases}$$

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

$a=1$; $b=2$; $c\cdot n^k=c$, $k=0$; $a=b^k$ luego $T(n)\in\Theta(\log n)$

El coste de *crear_montículo* puede expresarse, entonces mediante la siguiente recurrencia:

$T(n)=2\cdot T(n/2)+\log n$

Sin embargo, esta recurrencia no se puede resolver con la fórmula anterior puesto que la parte no recursiva no tiene una complejidad polinomial. Para resolverlo, vamos a utilizar un cambio de variable:

Sea h la altura del montículo de n nodos: $h=\log_2 n$. En cada llamada recursiva bajamos un nivel por lo que el problema se puede expresar mediante una recurrencia con reducción del problema por sustracción. Es decir, si en cada paso se baja un nivel, el problema se reduce en uno.

$$T(n)=\begin{cases} c\cdot n^k & \text{si } 0\le n<b \\ a\cdot T(n-b)+c\cdot n^k & \text{si } n\ge b \end{cases}$$

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

En el caso de *hundir* $T(h)=T(h-1)+c$, con a=1 y b=1, luego $T(h)\in\Theta(h)$, que deshaciendo el cambio de variable lleva a un tiempo en función de n: $T(n)\in\Theta(\log n)$ como habíamos demostrado antes.

Sin embargo, ahora ya podemos plantear la recurrencia para *crear_montículo:*

$T(h)=2\cdot T(h-1)+h$, donde a=2 y b=1 y, por tanto, $T(h)\in\Theta(2^h)$. Deshaciendo el cambio de variable:

$2^h=2^{\log_2 n}=n$, y $T(n)\in\Theta(n)$, que es el menor coste posible para dar estructura de montículo a un vector.

NOTA: cualquier solución que tenga mayor coste no será puntuada.