UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA	**Programación III** Código de asignatura (marcar con una X): Sistemas: __ 402048 (plan viejo) __ 532044 (plan nuevo) Gestión: __ 41204- (plan viejo) __ 542046 (plan nuevo)	**Prueba Presencial** **Ordinario** Septiembre de 2006 Duración: **2 horas** Material permitido: **NINGUNO**

Apellidos: _____ DNI: _____

Nombre: _____ Centro donde entregó la práctica: _____ e-mail: _____

Cuestión 1 (2 puntos). Demuestra que el tiempo de ejecución en función de n del siguiente fragmento de código está acotado superiormente por $O(n^2)$ e inferiormente por $\Omega(n)$ (1.5 puntos). Demuestra también su orden exacto de complejidad (0.5 puntos).

```
para i desde 1 hasta n hacer
        para j desde 1 hasta n div i hacer
                escribir "i,j,k";
        fpara
fpara
```

Supón que el coste de "escribir" es constante.

Solución:

En primer lugar debemos plantear el tiempo de ejecución *T(n)* en función de *n*. Para ello, vamos a fijarnos en la instrucción barómetro (la instrucción que se ejecuta más veces) y vamos a contar cuántas veces se ejecuta. Puesto que todas las instrucciones tienen un coste constante, la complejidad del programa estará en el orden del número de veces que se ejecute la instrucción barómetro.

$$T(n) = \sum_{i=1}^{n} \sum_{j=1}^{n/i} 1 = \sum_{i=1}^{n} \frac{n}{i} = n \cdot \sum_{i=1}^{n} \frac{1}{i}$$

$\sum_{i=1}^{n} \frac{1}{i}$ es un sumatorio de n elementos menores o iguales que 1, por lo que su suma total será menor que n. Por tanto, $T(n) \leq n \cdot n$, o en otras palabras, *T(n)* está acotado superiormente por n^2:

$T(n) \in O(n^2)$ como queríamos demostrar.

Por otra parte, $\sum_{i=1}^{n} \frac{1}{i}$ es mayor que 1 y, por tanto $T(n) \geq n \cdot 1$, o en otras palabras, *T(n)* está acotado inferiormente por n:

$T(n) \in \Omega(n)$ como queríamos demostrar.

Para demostrar el orden exacto es necesario saber cómo crece la serie $\sum_{i=1}^{n}\frac{1}{i}$. Para ello, podemos aproximar la serie con la integral:

$\int_{1}^{n}\frac{1}{x}dx$ que corresponde al logaritmo natural de *n*. Por tanto, la serie crece tan rápidamente como el ln(n). Así pues, se pueden encontrar dos constantes *c* y d tal que T(n) esté acotado superiormente por *c·n·log(n)* e inferiormente por *d·n·log(n)*. En conclusión, el orden exacto de T(n) es *n·log(n)*:

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

Cuestión 2 (2 puntos). Escribe la salida al realizar la llamada "pinta(5)", dado el siguiente código (1.5 puntos):

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

Demuestra el coste computacional de la función "pinta" suponiendo que "escribir" tiene coste constante (0.5 puntos).

Solución:

La salida sería: 5 4 3 2 1 0 1 2 3 4

Para calcular el coste 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=1; b=1; n^k=1; k=0

Por tanto, $T(n) \in \Theta(n)$

Cuestión 3 (2 puntos). Dibuja cómo evolucionaría el siguiente vector al ordenarlo mediante el algoritmo de ordenación rápida (Quicksort). Indica únicamente cada una de las modificaciones que sufriría el vector.

6	5	1	2	3	4	7	8	9

Solución:

Si se toma como pivote el primer elemento:

6	5	1	2	3	4	7	8	9
4	5	1	2	3	6	7	8	9
4	3	1	2	5	6	7	8	9
2	3	1	4	5	6	7	8	9
2	1	3	4	5	6	7	8	9
1	2	3	4	5	6	7	8	9

Si se toma como pivote el elemento que ocupa la posición central, en este caso el primer pivote es el elemento que ocupa la posición quinta, que es el 3:

6	5	1	2	3	4	7	8	9
2	5	1	6	3	4	7	8	9
2	1	5	6	3	4	7	8	9
2	1	3	6	5	4	7	8	9
1	2	3	6	5	4	7	8	9
1	2	3	4	5	6	7	8	9

Problema (4 puntos). Desarrollar un programa que compruebe si es posible que un caballo de ajedrez mediante una secuencia de sus movimientos permitidos recorra todas las casillas de un tablero NxN a partir de una determinada casilla dada como entrada y sin repetir ninguna casilla. Se pide:

1. Determinar qué esquema algorítmico es el más apropiado para resolver el problema. Razonar la respuesta y escribir el esquema general (0.5 puntos)
2. Indicar que estructuras de datos son necesarias (0.5 puntos)
3. Desarrollar el algoritmo completo en el lenguaje de programación que utilizaste para la práctica (2.5 puntos)
4. Hallar el orden de complejidad del algoritmo desarrollado (0.5 puntos)

Solución:

Una solución en pseudocódigo se puede encontrar en la página 86 del libro de problemas de la asignatura: “Esquemas Algorítmicos: Enfoque Metodológico y Problemas Resueltos”. Julio Gonzalo Arroyo y Miguel Rodríguez Artacho. Cuadernos de la UNED.