UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA	**Programación III** Solución **Prueba Presencial**	**Prueba Presencial** **Original** Septiembre de 2005 Duración: **2 horas** Material permitido: **NINGUNO**

Cuestión 1 (2 puntos). Suponga que N personas numeradas de 1 a N deben elegir por votación a una entre ellas. Sea V un vector en el que la componente V[i] contiene el número del candidato que ha elegido el votante i. ¿Qué algoritmo utilizarías para determinar con un coste lineal si una persona ha obtenido más de la mitad de los votos?

Podría utilizarse una estrategia similar a la que emplea para ordenación el algoritmo de la *casilla* (página 80 del libro), pero trasladado al conteo de elementos. La siguiente función almacena en el vector *votos* el número de votos que va sumando cada candidato votado. Devuelve un valor TRUE si hay alguno con mayoría indicando de qué candidato se trata:

```
función contar_votos(V[1..N) dev (booleano, elegido)
        votos[1..N]=0;
        para i=1 hasta N hacer
                votos[V[i]]=votos[V[i]]+1;
                si votos[V[i]]>N/2 entonces devolver (TRUE,V[i]) fsi;
        fpara
        devolver (FALSE,0);
```

Cuestión 2 (1 punto). En el contexto de elegir un esquema algorítmico para resolver un problema de optimización con restricciones, ¿cuándo se puede resolver mediante un esquema voraz y en qué casos sería necesario utilizar un esquema de ramificación y poda?

Para resolverlo con un esquema voraz es necesario que exista una función de selección de candidatos y una función de factibilidad que decida si se acepta o rechaza el candidato, de manera que la decisión es irreversible. Si no es posible encontrar las funciones de selección y de factibilidad de manera que se garantice que la elección de un candidato lleva a la solución óptima, entonces optaríamos por otro esquema como el de ramificación y poda.

Cuestión 3 (2 puntos). Sea $T(n)=4n^2-3n+2$ el tiempo de ejecución de un algoritmo. Demuestra si es cierta o falsa cada una de las siguientes afirmaciones (0.5 puntos cada una):

Recordamos la regla del límite:

$$\lim_{n\to\infty}\frac{f(n)}{g(n)}=\begin{cases} c & entonces \quad f(n)\in\Theta(g(n)) \\ 0 & entonces \quad f(n)\in O(g(n)),\ f(n)\notin\Theta(g(n)) \\ +\infty & entonces \quad f(n)\in\Omega(g(n)),\ f(n)\notin\Theta(g(n)) \end{cases}$$

a) $T(n) \notin O(n^2 \ln n)$

$$\lim_{n\to\infty}\frac{4n^2-3n+2}{n^2\ln n}=\lim_{n\to\infty}\frac{8n-3}{2n\ln n+n}=\frac{8}{2\ln n+3}=0$$

Luego pertenece al orden indicado y la cuestión a) es **FALSA**

b) $T(n) \notin O(n^3)$

$$\lim_{n\to\infty}\frac{4n^2-3n+2}{n^3}=\lim_{n\to\infty}\frac{8n-3}{3n^2}=\frac{8}{6n}=0$$

Por lo que T(n) pertenece a $O(n^3)$ y la cuestión b) es **FALSA**

c) $T(n) \in \Omega(n \log n)$

$$\lim_{n\to\infty} \frac{4n^2 - 3n + 2}{n \ln n} = \lim_{n\to\infty} \frac{8n - 3}{\ln n + 1} = \frac{8}{\frac{1}{n}} = +\infty$$

con lo que c) es **CIERTO**

d) $T(n) \in O(n^2)$

$$\lim_{n\to\infty} \frac{4n^2 - 3n + 2}{n^2} = \lim_{n\to\infty} \frac{8n - 3}{2n} = \frac{8}{2} = 4 \in \Re^+$$

En este caso, al ser $T(n) \in \Theta(n^2)$ se cumple también que $T(n) \in O(n^2)$ luego d) es **CIERTO**

Problema (5 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. Implementa un programa cuya función principal siga el esquema divide y vencerás, 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 del problema debe incluir, por este orden:

1. Descripción del esquema divide y vencerás y su aplicación al problema (0.5 puntos).

   El esquema general Divide y Vencerás consiste en:
   1. Descomponer el ejemplar en subejemplares del mismo tipo que el original
   2. Resolver independientemente cada subejemplar
   3. Combinar los resultados para construir la solución del ejemplar original

   Más formalmente:

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

2. Algoritmo completo a partir del refinamiento del esquema general (3 puntos)

```
función contar(V[1..N], i, j) dev (tiene_mayoría, candidato, num_votos)
    si (i==j) entonces
            dev (TRUE, V[i], 1)
    si no
            /* Descomponer */
            (tiene_mayoría1, candidato1, num_votos1)=contar(V, i, (i+j)÷2);
            (tiene_mayoría2, candidato2, num_votos2)=contar(V, (i+j)÷2+1, j);
            /* Recombinar */
            si tiene_mayoría1 entonces
                    para k desde (i+j)÷2+1 hasta j hacer
                            si es_igual(V[k], candidato1) entonces
                                    num_votos1=num_votos1+1;
                            fsi
                    fpara
                    si (num_votos1>(j-i+1)/2) entonces
                            devolver (cierto, candidato1, num_votos1);
                    fsi
            fsi
            si tiene_mayoría2 entonces
                    para k desde i hasta (i+j)÷2 hacer
                            si es_igual(V[k], candidato2) entonces
                                    num_votos2=num_votos2+1;
                            fsi
                    fpara
                    si (num_votos2>(j-i+1)/2) entonces
                            devolver (cierto, candidato2, num_votos2);
                    fsi
            fsi
            devolver (falso,"",0);
    fsi
```

Llamada inicial *contar(V,1,N)*;

3. Estudio del coste del algoritmo desarrollado (1.5 puntos)

Planteamos la ecuación de recurrencia

$T(n) = 2 \cdot T(n \div 2) + n \div 2 + n \div 2 = 2 \cdot T(n \div 2) + n$

La reducción del problema se realiza mediante división, cuyos casos son los siguientes:

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

donde:

- n: tamaño del problema
- a: número de llamadas recursivas
- n/b: tamaño del subproblema
- n^k: coste de las instrucciones que no son llamadas recursivas

Aplicado a nuestro problema:

$c \cdot n^k=n$; $k=1$; $b=2$; $a=2$; $b^k=2^1=2$

$a=b^k$ luego $T(n) \in \Theta(n \log n)$