UNIVERSIDAD DE MÁLAGA

Análisis y Diseño de Algoritmos

Parcial 1 (Septiembre 2025)

Nombre: **Grupo:**

1. (10 ptos) Hemos diseñado dos algoritmos que encuentran la primera vez que aparece un elemento *x* en un array dado.

```
int indexOf_v1(int[] a, int x, int izq, int der) {
     int i = izq;
     while (i <= der  && a[i]!= x) {
          i++;
     }
     if (i == der+1) {
          i = -1;
     }
     return i;}

int indexOf_v2(int[] a, int x, int izq, int der) {
     int res = -1;

     if (izq <= der) {
          int m = (izq + der)/2;
          if (x == a[m]) {
               if (m > izq && a[m-1] == x) {
                    res = indexOf_v2(a,x,izq,m-1);
               }else {
                    res = m;
               }
          }else if (x < a[m]) {
               res = indexOf_v2(a,x,izq,m-1);
          }else {
               res = indexOf_v2(a,x,m+1,der);
          }
     }
     return res;}
```

a) *(2 ptos)* Determinar la función de complejidad de *indexOf_v1* en el peor caso, considerando como tamaño de entrada el número de elementos del rango [izq, der] y contabilizando sólo las comparaciones (<,>,==,!=, <=,>=).
b) *(1 pto)* Indicar el orden de crecimiento exacto de la función del apartado a) y justificarlo matemáticamente.
c) *(2 ptos)* Determinar la ecuación de recurrencia que determina la función de complejidad de *indexOf_v2*.
d) *(2.5 pto)* Resolver la recurrencia del apartado c) para obtener una expresión cerrada equivalente.
e) *(1.5 pto)* Utilizar el Teorema Maestro para obtener el orden de crecimiento exacto de la función de complejidad del apartado c).
f) *(1 pto.)* Indicar qué algoritmo es más eficiente y cuál es más complejo y justificarlo matemáticamente.

En el peor caso, no se encontrará el elemento, por lo que el bucle itera el número máximo de veces, hasta que la última evaluación de la condición de control se evalúa a falso porque falla (i<=der) (evaluación en conrtocircuito)

$$T_1(n) = \sum_{i=izq}^{der} 2 + 1 + 1 = (der - izq + 1) \cdot 2 + 2 = 2 \cdot n + 2$$

En este caso $T_1(n) \in \Theta(n)$.
ya que $\lim_{n\to\infty h} \frac{2n+2}{n} = \lim_{n\to\infty h} 2 + \frac{2}{n} = 2$

El peor caso del algoritmo recursivo se da cuando el array sólo contiene el valor que buscamos. En ese caso siempre entrará por la primera rama del if y hará llamadas recursivas hasta el final. Por ello,

$$T_2(n) = 4 + T\left(\frac{n}{2}\right) \qquad si\ n > 0;$$

Como casos base tenemos una llamada al algoritmo con un rango vacío. En este caso $T_2(0) = 0$.
Podemos suponer que el peor caso anterior va a ir consumiendo el rango de valores hasta que quede un solo elemento. Podría ser *x* y se harían *3* comparaciones (evaluación en cortocircuito ya que *m > izq* sería falso). Por ello, caso $T_2(1) = 3$.
Para resolver la recurrencia vamos a hacer el cambio de variable: $n = 2^k$

$$T_2(2^k) = 4 + T\left(\frac{2^k}{2}\right) = T(2^{k-1}) + 4$$

Renombrando $T_2(2^k)$ por F(k), tenemos

$$F(k) = F(k-1) + 4$$

cuyo polinomio característico es $(x-1)\cdot(x-1) = (x-1)^2$
Por ello, la expresión cerrada equivalente a F(k) es

$$F(k) = (\alpha_1 \cdot k + \alpha_2) \cdot 1^k = \alpha_1 \cdot k + \alpha_2$$

Deshaciendo los cambios queda,

$$T_2(2^k) = \alpha_1 \cdot k + \alpha_2$$
$$T_2(n) = \alpha_1 \cdot \log_2 n + \alpha_2$$

Podemos obtener el valor de las constantes, usando T(2)=4+T(1)=7

$$T(1) = 3 = \alpha_1 \log_2 1 + \alpha_2 = \alpha_1 \cdot 0 + \alpha_2 = \alpha_2$$
$$T(2) = 7 = \alpha_1 \cdot \log_2 2 + \alpha_2 = \alpha_1 + \alpha_2$$

Por lo que la expresión final es $T_2(n) = 4 \cdot \log_2 n + 3$

Podríamos aplicar el teorema maestro reducido a la ecuación de recurrencia para obtener su orden exacto.
a = 1, b = 2 y f(n) es un polinomio de grado 0.
$a = 1 = 2^0 = b^d \Rightarrow T_2(n) \in \Theta(\log n)$

Para determinar qué algoritmo es más eficiente vamos a comparar sus funciones de complejidad (o sus órdenes de crecimiento)

$$\lim_{n\to\infty} \frac{2n+2}{4\log n + 3} = \frac{\infty}{\infty} = \lim_{n\to\infty} \frac{n}{4/n} = \lim_{n\to\infty} n^2 = \infty$$

Por tanto, $T_1(n) \in \Omega(T_2(n)) \wedge T_1(n) \notin \Theta(T_2(n))$, es decir, el algoritmo indexOf_v1 es más complejo que la segunda versión, que es la más eficiente.