Version1-Resuelto.pdf

NachoPiece

Análisis y diseño de algoritmos

2º Grado en Ingeniería del Software

Escuela Técnica Superior de Ingeniería Informática
Universidad de Málaga

Análisis y Diseño de Algoritmos

Parcial 1 (Noviembre 2020)

1. (3ptos) Escribir la especificación de una función e*lementoConMedia* que recibe un array de números enteros positivos y dos posiciones válidas *ini* y *fin*, y devuelve un valor booleano que indica si existe una posición que contiene la media de los valores en el rango de posiciones *[ini,fin]*.

$$\{\forall j: 0 \leq j < a.length: a[j] > 0 \wedge 0 \leq ini, fin < a.length\}$$

boolean elementoConMedia(int a[]) //b

$$\{b = [\exists i: 0 \leq i < a.length: a[i] = (\sum_{j=ini}^{fin} j)\}$$

2. (3 ptos) Sea T(n) = 9T(n/3) + n log n + n, la función de complejidad de un algoritmo. Se pide:

a) Resolver la ecuación de recurrencia y obtener el coste temporal exacto (sin calcular el valor de las constantes).

b) Indicar el orden de crecimiento del algoritmo, justificando matemáticamente la elección.

$$T(n) - 9T\left(\frac{n}{3}\right) = nlogn + n$$

Aplicamos el cambio de variable n = 3^k

$$T(3^k) - 9T\left(\frac{3^k}{3}\right) = 3^k log 3^k + 3^k; \;\; T(3^k) - 9T(3^{k-1}) = 3^k \cdot Cte \cdot k + 3^k \approx 3^k k + 3^k$$

Renombramos T(3^k) por F(k)

$$F(k) - 9F(k-1) = 3^k(k+1)$$

El polinomio característico es (x-9) $(x-3)^2$ = 0. Por tanto,

$$F(k) = a9^k + b3^k + ck3^k$$

Deshacemos los cambios

$$T(3^k) = a9^k + b3^k + ck3^k$$

$$T(n) = a9^{\log_3 n} + bn + cnlog_3 n = an^2 + bn + cnlog_3 n$$

El orden de crecimiento es $\Theta(n^2)$, ya que

$$\lim_{n\to\infty} \frac{T(n)}{n^2} = \lim_{n\to\infty} \frac{an^2 + bn + cnlog_3 n}{n^2} = \lim_{n\to\infty} \frac{an^2}{n^2} + \lim_{n\to\infty} \frac{bn}{n^2} + \lim_{n\to\infty} \frac{cnlog_3 n}{n^2}$$
$$= a + 0 + \lim_{n\to\infty} \frac{clog_3 n}{n}$$

Aplicando la regla de L'Hôpital:

$$\lim_{n\to\infty} \frac{T(n)}{n^2} = a + \lim_{n\to\infty} \frac{c \cdot cte \cdot 1/n}{1} = a + 0 = a$$

3. (4 ptos) Dado un array ordenado, se desea calcular el número de veces que aparece el valor x, siendo x unos de los valores presentes en el array.

a) Utilizar la estrategia Divide y Vencerás para diseñar un algoritmo de coste O(log n) que resuelva el problema indicado.

b) Analizar la complejidad del programa para demostrar que se ajusta al orden de crecimiento requerido. Se puede utilizar el Teorema Maestro si se considera necesario.

```java
public static int frecuencia(int[] a, int x) {
            int izq = buscarMasIzquierda(a, x, 0, a.length - 1);
            int der = buscarMasDerecha(a, x, 0, a.length - 1);
            return der - izq + 1;
      }

      // Precondición: x está en el array
      private static int buscarMasIzquierda(int[] a, int x, int izq,
int der) {
            int res = -1;

            if (izq == der) {// tiene que ser ese
                  res = izq;
            } else if (izq == der - 1) {
                  if (a[izq]== x) {
                        res = izq;
                  }else {
                        res = der;
                  }
            } else {
                  int m = (izq + der) / 2;
                  if (a[m] > x) {
                        res = buscarMasIzquierda(a, x, izq, m - 1);
                  } else if (a[m] < x) {
                        res = buscarMasIzquierda(a, x, m + 1, der);
                  } else {
                        if (a[m] != a[m - 1]) {
                              res = m;
                        } else {
                              res = buscarMasIzquierda(a, x, izq, m -
1);
                        }
                  }
            }
            return res;
      }

      //Precondición: x está en el array.
      //Eso hace que en el caso de 2 elementos, a[m]>x es falso porque
significaría que no está el elemento
      //lo cual contradice la precondición
      private static int buscarMasDerecha(int[] a, int x, int izq, int
der) {
            int res = -1;

            if (izq == der) {// tiene que ser ese
                  res = izq;
```

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

	}
```

buscarMasDerecha y buscarMasIzquierda tienen un coste similar. En ambos casos realizamos un número constante de operaciones elementales y hacemos una llamada recursiva para un array de la mitad de tamaño aproximadamente.
Por ello T(n) = T(n/2) + k.
Identificamos a = 1, b= 2 y f(n) es un polinomio de grado 0. Dado que $1 = 2^0$, por el teorema maestro podemos afirmar que el coste para ambas funciones es $T(n) \in \Theta(logn)$

La función frecuencia invoca una vez a cada función y realiza una operación aritmética. Por ello, su complejidad es
T(n) = log n + log n + k = 2 log n + k
Es decir $T(n) \in \Theta(logn)$