Tiempo de Ejecución III

Cálculo de T(n) y O(n)

Enunciado

- Calcular el T(n) de countOverlap para el **peor caso**, detallando los pasos seguidos para llegar al resultado.
- Calcular el O(n) de la función del punto anterior justificando con la **definición de Big-OH**.

```java
public static int recu(int[] array, int count, int len) {
    if (len == 0)
        return 0;
    else
    if (array[len - 1] == count)
        return 1 + recu(array, count, len - 1);
    else
        return recu(array, count, len - 1);
}

public static void countOverlap(int[] arrayA, int[] arrayB) {
    int count = 0, calc = 0, tam = 0;
    if (arrayA.length == arrayB.length) {
        tam = arrayA.length;
        for (int i = 0; i < arrayA.length; i++)
            for (int j = 0; j < arrayB.length; j++)
                if (arrayA[i] == arrayB[j]) {
                    count++;
                    calc = calc + recu(arrayA, count, tam)
                            + recu(arrayB, count, tam);
                }
    }
    System.out.println("count:" + count + "-calc:" + calc);
}
```

Método recursivo

Método iterativo

T(n) de "countOverlap"

```java
public static void countOverlap(int[] arrayA, int[] arrayB) {
    int count = 0, calc = 0, tam = 0;
    if (arrayA.length == arrayB.length) {
        tam = arrayA.length;
        for (int i = 0; i < arrayA.length; i++)
            for (int j = 0; j < arrayB.length; j++)
                if (arrayA[i] == arrayB[j]) {
                    count++;
                    calc = calc + recu(arrayA, count, tam)
                            + recu(arrayB, count, tam);
                }
    }
    System.out.println("count:" + count + "-calc:" + calc);
}
```

Tiempo constante

Sumatorias

Tiempo constante + ?

T(n) de “recu”

```
public static int recu(int[] array, int count, int len) {
    if (len == 0)
        return 0;
    else
    if (array[len - 1] == count)
        return 1 + recu(array, count, len - 1);
    else
        return recu(array, count, len - 1);
}
```

len se reduce en 1 hasta llegar al caso base, por lo tanto **len** es nuestro **N**

$$T'(n) = \begin{cases} c & ,n = 0 \\ d + T'(n-1), & n > 0 \end{cases}$$

$$T'(n) = \begin{cases} c & , n = 0 \\ d + T'(n-1), & n > 0 \end{cases}$$

(paso 1) $T'(n) = d+T'(n-1)$

(paso 2) $T'(n) = d+d+T'(n-2)$

(paso 3) $T'(n) = d+d+d+T'(n-3)$

....

(paso i) $T'(n) = id+T'(n-i)$

$$n - i = 0 \rightarrow i = n$$

$$= nd + T'(n-n) \rightarrow nd + T'(0) \rightarrow nd + c$$

$T'(n) = dn + c$

T(n) de countOverlap

- Debemos considerar que ambos arrays son de igual tamaño (peor caso).
- El tamaño del array es nuestro N

$$T(n) = b + \sum_{i=0}^{n-1} \left(\sum_{j=0}^{n-1} (f + 2(dn + c)) \right) (sea\ g = f + 2c)$$

Las sumatorias evalúan N veces. El contenido de sumatoria no se ve afectado por el índice, puedo o no reescribir los índices de la sumatoria

$$= b + \sum_{i=1}^{n} (n(2dn + g))$$

$$= b + n(n)(2dn + g)$$

$$= b + 2dn^3 + gn^2$$

O(n) de countOverlap

Siendo $\mathbf{T(n) = b + d2n^3 + gn^2}$, vamos a demostrar que tiene $O(n^3)$, por definición si se cumple

$$\mathbf{b + d2n^3 + gn^2 \;\;<= k * n^3 \qquad\qquad k>0 \text{ y } n>=n_0}$$

$b \qquad <= k0 * n^3 \qquad\qquad k0 = b \;\; \text{y para } n_0 = 1$

$2dn^3 \;\; <= k1 * n^3 \qquad\qquad k1 = 2d \text{ y para } n_0 = 0$

$gn^2 \quad <= k2 * n^3 \qquad\qquad k2 = g \;\; \text{y para } n_0 = 0$

Luego…

$b + 2dn^3 + gn^2 <= (k0 + k1 + k2\ )\ n^3$

$b + 2dn^3 + gn^2 <= k * n^3 \qquad\qquad k = k0+k1+k2 = b + 2d + g \text{ y para } n_0 = 1$

Por lo tanto, $b + 2dn^3 + gn^2$ es $O(n^3)$

¿Cuál es el O(n) de la siguiente expresión?

$$T(n) = 4n + 100Log_2 n - Log_4 n$$

Utilizando Big-Oh debemos probar que

$$4n+100Log_2 n-Log_4 n \leq k* n \qquad k > 0, \forall n \geq n_0$$

Debemos verificar la desigualdad para cada uno de los términos

$$\mathbf{4}\boldsymbol{n}\mathbf{+100}\boldsymbol{Log_2 n - Log_4 n} \leq \mathbf{k * }\boldsymbol{n}\ \mathbf{k>0}\ ,\ \forall \boldsymbol{n \geq n_0}$$

$4n \leq k1 * n$ $\quad$ $k1=4\ y\ n_0 = 0$

$100 Log_2 n \leq k2 * n$ $\quad$ $k2=100\ y\ n_0=1$

Otra opción:

$100 Log_2 n \leq k2 * n$, $k2=1\ y\ n_0=1000$

- No es necesario considerar $-Log_4 n$ puesto que es negativo. Si logramos acotar el resto de la expresión, estaremos acotando el resto de la expresión "–" algo.

$$4n \quad \leq 4n, \qquad k1=4 \; y \; \forall n_0$$

$$100Log_2n \leq 100n, \qquad k2=100 \; y \; n_0=1$$

$$4n+100Log_2n \leq 4n+100n$$

$$4n+100Log_2n \leq 104n$$

$$4n+100Log_2n \leq k * n, \qquad k=104 \; y \; n_0=1$$

$$\boxed{4n+100Log_2n-Log_4n \leq k*n, \qquad k=104 \; y \; n_0=1}$$