Ejercicios Eficiencia (FAL)

A.L.K.

Octubre 2020

1. *La siguiente función implementa un algoritmo de búsqueda secuencial con centinela en un array de enteros (el centinela es -1). Analiza la complejidad en tiempo de dicho algoritmo en el peor y el mejor caso*

```
bool busca(const int[] elems, int elem)
{
    int i = 0;
    while (elems[i] != elem && elems[i] != -1)
    {
        i++;
    }
    return elems[i] != -1;
}
```

Mejor caso: el elemento buscado es el primer elemento del array. El coste del algoritmo sería:

- K_0 = Coste de asignación y de comprobaciones del bucle while

Para k >0 T(n) = K (complejidad constante)

$$\lim_{n\to\infty} \frac{k}{1} = k > 0 \to \Theta(1) \left\{ \begin{array}{c} \Omega = 1 \\ 0 = 1 \end{array} \right\}$$

Peor caso: el elemento no está en el array. El bucle dará tantas vueltas como elementos haya con un coste constante K_1 por vuelta. Los costes serán:

- K_0 = coste de inicialización y finalización
- $n * K_1$ = n veces el coste del bucle

$$T(n) = K_0 + n * K_1 (complejidad\ n\ lineal)$$

$$\lim_{n\to\infty} \frac{K_0 + n * K_1}{n} = \lim_{n\to\infty} K_1 + \frac{K_0}{n} = K_1 > 0 \to \Theta(n)$$

2. *Considera la siguiente función:*

```
void insertar(tSecuencia &sec, int elem)
{
    int i = sec.nelems while (i > 0 && sec.datos[i - 1] > elem)
    {
        sec.datos[i] = sec.datos[i - 1];
        i--;
    }
    sec.datos[i] = elem;
    sec.nelems++;
}
```

Esta función implementa el algoritmo de inserción de un elemento en una secuencia de enteros con posibles repeticiones, representada mediante el tipo:

```
typedef struct tSecuencia
{
    int *datos;
    int nelems;
};
```

(la funcion supone que en datos hay espacio suficiente). Analizar la complejidad en tiempo de dicho algoritmo en el peor y mejor caso.

Mejor caso: el elemento es el mayor de la lista y por tanto se inserta al final, por tanto su complejidad será constante por los costes de inicialización y finalización.

Peor caso: el elemento es menor que todos los de la lista y el algoritmo se ejecuta n veces

$$T(n) = K_1 * n + K_0 \rightarrow T(n) \in H(n)$$

3. Dada una secuencia de enteros **S**, se dice que la posición de i es singular cuando su valor coincide con la suma de todos los valores que lo preceden (es decir, cuando $S = \sum_{j=0}^{i-1} S_j$). A continuación, se presentan cuatro algoritmos que determinan el número de posiciones singulares de una secuencia, almacenada en las primeras n posiciones de un array a:

Algoritmo 1:

```
unsigned int num_singulares(int a[], unsigned int n)
{
    int result = 0;
    for (int i = 0; i < n; i++)
    {
        int suma = 0;
        for (int j = 0; j < i; j++)
        {
            suma += a[j];
        }
        if (a[i] == suma)
            result++;
    }
    return result;
}
```

Coste bucle interno en cada interacción del externo: $i * K_0$(comprobación, suma e incremento). i varía de 0 a n-1.

Coste total del bucle interno: $\sum_{i=0}^{n-1} i * K_0 = K_0 * \frac{n(n-1)}{2}$

- $K_1 \rightarrow$ no se incrementa result
- $K_2 \rightarrow$ se incrementa result
- $K_3 \rightarrow$ coste inicialización y finalización

Mejor caso:

$$E_0(n) = K_0 \frac{n(n-1)}{2} + k_3 + n * K_1 \rightarrow \lim_{n \to \infty} \frac{\frac{k_0 * n^2}{2} + n(K_1 - \frac{K_0}{2}) + K_3}{n^2} = \frac{K_0}{2} > 0 \in \Theta(n^2)$$

Peor caso:

$$E_1(n) = K_0 \frac{n(n-1)}{2} + k_3 + n * K_2 \rightarrow \lim_{n \to \infty} \frac{\frac{k_0 * n^2}{2} + n(K_2 - \frac{K_0}{2}) + K_3}{n^2} = \frac{K_0}{2} > 0 \in \Theta(n^2)$$

$$E_0(n) \leq T(n) \leq E_1 \rightarrow \left\{ \begin{array}{c} T(n) \in \Omega(n^2) \\ T(n) \in 0(n^2) \end{array} \right\} \Rightarrow T(n) \in \Theta(n^2)$$

Algoritmo 2:

```
unsigned int num_singulares(int a[], unsigned int n)
{
    int resul = 0;
    int suma = 0;
    for (int i = 0; i < n; i++)
    {
        if (a[i] == suma)
            resul++;
        suma += a[i];
    }
    return resul;
}
```

- $K_0 \rightarrow$ result no se incrementa
- $K_1 \rightarrow$ result se incrementa
- $K_2 \rightarrow$ coste de inicialización y finalización

Mejor caso:

$$E_1(n) = n * K_0 + +K_2 \rightarrow \lim_{n\to\infty} \frac{n * K_0 + K_2}{n} = K_0 > 0 \in \Theta(n)$$

Peor caso:

$$E_1(n) = n * K_1 + +K_2 \rightarrow \lim_{n\to\infty} \frac{n * K_1 + K_2}{n} = K_1 > 0 \in \Theta(n)$$

$$E_0(n) \leq T(n) \leq E_1 \rightarrow \left\{ \begin{array}{c} T(n) \in \Omega(n) \\ T(n) \in 0(n) \end{array} \right\} \Rightarrow T(n) \in \Theta(n)$$

Algoritmo 3:

```
unsigned int num_singulares(int a[], unsigned int n)
{
    int result = 0;
    int suma = 0;
    int i = 0;
    int j = 0;
    while (i < n)
    {
        if (j == i)
        {
            if (a[i] == suma)
            {
                result++;
            }
            suma = 0;
            j = 0;
            i++;
        }
        else
        {
            suma += a[j];
            j++;
        }
    }
    return result;
}
```

Antes de incrementar i, hay que incrementar j i veces

- $K_0 \rightarrow$ coste iteración j, coste total: $K_0 \frac{n(n-1)}{2}$
- Iteraciones de i $\begin{array}{l} K_1 \rightarrow \textit{no se incrementa result} \\ K_2 \rightarrow \textit{no se incrementa result} \end{array}$
- $K_3 \rightarrow$ coste de inicialización y finalización

Mejor caso: Mejor caso:

$$E_0(n) = K_0 \frac{n(n-1)}{2} + K_1 + K_3 \rightarrow \lim_{n \to \infty} \frac{\frac{k_0 * n^2}{2} - n(\frac{K_0}{2}) + K_3 + K_1}{n^2} = \frac{K_0}{2} > 0 \in \Theta(n^2)$$

Peor caso:

$$E_1(n) = K_0 \frac{n(n-1)}{2} + k_3 + K_2 \rightarrow \lim_{n \to \infty} \frac{\frac{k_0 * n^2}{2} + n(\frac{K_0}{2}) + K_3 + K_2}{n^2} = \frac{K_0}{2} > 0 \in \Theta(n^2)$$

$E_0(n) \leq T(n) \leq E_1 \rightarrow \left\{ \begin{array}{c} T(n) \in \Omega(n^2) \\ T(n) \in 0(n^2) \end{array} \right\} \Rightarrow T(n) \in \Theta(n^2)$

Algoritmo 4:

```c
unsigned int num_singulares(int a[], unsigned int n)
{
    int result = 0;
    int suma = 0;
    int i = 0;
    while (i < n)
    {
        while (i < n && a[i] % 2 == 0)
        {
            if (a[i] == suma)
            {
                result++;
            }
            suma += a[i];
            i++;
        }
        if (i < n)
        {
            if (a[i] == suma)
            {
                result++;
            }
            suma += a[i];
            i++;
        }
    }
    return result;
}
```

Hay n incrementos de i como mucho.

- $K_{minima} \rightarrow$ Trabajo constante mínimo antes de cada incremento
- $K_{mxima} \rightarrow$ Trabajo constante máximo antes de cada incremento
- $K_i \rightarrow$ Trabajo constante al final del algoritmo

$m(n) = n * K_{min} + K_i \rightarrow$ estimación optimista T(n)
$p(n) = n * K_{max} + K_i \rightarrow$ estimación pesimista T(n)
$m(n), p(n) \in \Theta(n)$
$m(n) \leq T(n) \leq p(n) \rightarrow T(n) \in \Theta(n)$