Análisis y diseño de algoritmos

COMPLEJIDAD TEMPORAL: CÁLCULO ANALÍTICO

Solución del ejercicio 1 de la práctica 3

Ejercicio 1

Realiza un análisis **analítico** de la complejidad temporal de la siguiente función del lenguaje C++. En el supuesto de que existan los casos mejor y peor identifica las instancias que pertenecen a cada caso y obtén las correspondientes cotas de complejidad.

```
1  int ejercicio1 (vector < int > & v){
2
3      int i,sum=0, n=v.size();
4
5      if (n>0){
6          int j=n;
7          while (j>0 and sum<100){
8              j=j/2;
9              sum=0;
10             for (i=j;i<n;i++)
11                 sum+=v[i];
12         }
13         return j;
14     }
15     else return -1;
16 }
```

Solución:

El tamaño del problema viene dado por el número de elementos del vector. Sea $n = \texttt{v.size()}$. El algoritmo presenta caso mejor y peor (¿puedes decir por qué?)

Complejidad temporal en el mejor de los casos:

Las instancias que pertencen al caso más favorable se caracterizan por ser vectores cuyos elementos, desde la mitad en adelante, suman al menos 100. Es decir, en el mejor de los casos están todos los vectores v, de cualquier tamaño n, tal que $\sum_{i=\frac{n}{2}}^{n-1} v_i \geq 100$. la complejidad temporal en este caso viene dada por:

$$c_i(n) = \sum_{i=\frac{n}{2}}^{n-1} 1 = n - \frac{n}{2} \in \Omega(n)$$

Complejidad temporal en el peor de los casos:
En el caso más desfavorable están (entre otros) todos los vectores cuyos elementos suman menos de 100. Es decir:

$$v \in Z^n \mid \sum_{i=0}^{n-1} v_i < 100$$

(nótese que en realidad hay algunas instancias más que también están en el caso peor -¿sabrías decir alguna más?-).

Para calcular la complejidad temporal puede resultar útil representar en una tabla los pasos de programa que cada una de las iteraciones del bucle `while` consume. Por otra parte, para averiguar el número total de iteraciones se incorpora una columna que contiene los valores que toma j al comienzo de cada iteración:

Iteración bucle `while`	j	Pasos en cada iteración
1	n	$\frac{n}{2}$
2	$\frac{n}{2}$	$\frac{n}{2} + \frac{n}{4}$
3	$\frac{n}{4}$	$\frac{n}{2} + \frac{n}{4} + \frac{n}{8}$
...	...	...
k	$\frac{n}{2^{k-1}} = 1$	$\sum_{j=1}^{k} \frac{n}{2^j}$

La complejidad del algoritmo viene dada por la suma de los pasos en todas y cada una de las iteraciones del bucle `while`. Además, de la segunda columna se deduce que realiza $\lfloor \log_2 n \rfloor + 1$ iteraciones. Por lo tanto:

$$c_s(n) = \sum_{k=1}^{\lfloor \log_2 n \rfloor + 1} \sum_{j=1}^{k} \frac{n}{2^j} = n \sum_{k=1}^{\lfloor \log_2 n \rfloor + 1} \sum_{j=1}^{k} \frac{1}{2^j}$$

Teniendo en cuenta que $\sum_{j=1}^{k} \frac{1}{2^j} \in \Theta(1)$ (se deja como ejercicio su demostración), se tiene:

$$c_s(n) = n \sum_{k=1}^{\lfloor \log_2 n \rfloor + 1} 1 \in O(n \log n)$$

Nota:

La tabla "iteraciones-pasos" podría desarrollarse, de forma equivalente, de la siguiente manera:

Iteración bucle `while`	Pasos en cada iteración
1	$n - \frac{n}{2}$
2	$n - \frac{n}{4}$
3	$n - \frac{n}{8}$
...	...
k	$n - \frac{n}{2^k}$

Siguiendo el mismo razonamiento anterior se tiene:

$$c_s(n) = \sum_{k=1}^{\lfloor \log_2 n \rfloor + 1} n - \frac{n}{2^k} = n \sum_{k=1}^{\lfloor \log_2 n \rfloor + 1} 1 - \frac{1}{2^k} \in O(n \log n)$$