
UNIVERSIDAD DE GRANADA

Departamento de Ciencias de la Computación e Inteligencia Artificial

# Reto 1: Eficiencia

**J. Fdez-Valdivia**
Dpto. Ciencias de la Computación e Inteligencia Artificial
E.T.S. de Ingenierías Informática y de Telecomunicación
Universidad de Granada

## Estructuras de Datos

Grado en Ingeniería Informática
Doble Grado en Ingeniería Informática y Matemáticas
Doble Grado en Ingeniería Informática y ADE

**QUINTÍN MESA ROMERO 2º DGIIM**

*1.- Usando la* ***notación O,*** *determinar la eficiencia de las siguientes* `funciones`:

**(a)**

```
  void eficiencia1(int n)
{
  int x=0; int i,j,k;
          for(i=1; i<=n; i+=4)
               for(j=1; j<=n; j+=[n/4])                              |
                    for(k=1; k<=n; k*=2)
                            x++;
}
```

***(b)***

```
  int eficiencia2 (bool existe)
   {
      int sum2=0; int k,j,n;

      if (existe)
            for(k=1; k<=n; k*=2)
                 for(j=1; j<=k; j++)
                         sum2++;
      else
             for(k=1; k<=n; k*=2)
               for(j=1; j<=n; j++)
                         sum2++;
    return sum2;
   }
```

***(c)***

```
 void eficiencia3 (int n)              |    void eficiencia4 (int n)
{                                      |      {
  int j; int i=1; int x=0;             |      int j; int i=2; int x=0;
   do{                                 |       do{
      j=1;                             |           j=1;
      while (j <= n){                  |           while (j <= i){
         j=j*2;                        |               j=j*2;
         x++;                          |               x++;
      }                                |           }
      i++;                             |           i++;
    }while (i<=n);                     |         }while (i<=n);
}                                      |       }
```

*2.- Considerar el siguiente segmento de código con el que se pretende buscar un entero* ***x*** *en una lista de enteros L de tamaño n (el bucle* ***for*** *se ejecuta* ***n veces****):*

```
    void eliminar (Lista L, int x)
       {
          int aux, p;
          for (p=primero(L); p!=fin(L);)
              {
                 aux=elemento (p,L);
                 if (aux==x)
                    borrar (p,L);
                 else p++;
              }
       }
```

Analizar la eficiencia de la función **eliminar** si:

(a) **primero es O(1)** y **fin, elemento y borrar son O(n)**. ¿Cómo mejorarías esa eficiencia con un ligero cambio en el código?

(b) **primero, elemento y borrar son O(1)** y **fin es O(n)**. ¿Cómo mejorarías esa eficiencia con un ligero cambio en el código?

(c ) **todas las funciones son O(1)**. ¿Puede en ese caso mejorarse la eficiencia con un ligero cambio en el código?

***Consideraciones:***

*1.- El reto es **individual***

*2.- la solución deberá entregarse obligatoriamente en un fichero pdf (se sugiere como nombre reto1.pdf)*

*3.- Si la solución es correcta, se puntuará con 0.2 para la evaluación contínua*

*4.- El plazo límite de entrega es el 3 de Octubre a las 23.55h*

*1.- Usando la* ***notación O,*** *determinar la eficiencia de las siguientes* `funciones`:

**(a)**

```
void eficiencia1(int n)
{
  int x=0; int i,j,k;                          ⇒ O(1)
        3) for(i=1; i<=n; i+=4)                ⇒ O(n)
             2) for(j=1; j<=n; j+=[n/4])       ⇒ O(4) ≡ O(1)
                  1) for(k=1; k<=n; k*=2)      ⇒ O(log2 n)
                       x++;                    ⇒ O(1)
}
```

Regla del producto → $\Rightarrow O(\log_2 n)$ | $\Rightarrow O(\log_2 n)$ → Regla del producto → $\Rightarrow O(n \log_2 n)$

Para determinar la eficiencia de esta función se han tenido en cuenta la eficiencia de las operaciones simples ( $O(1)$ ), el número de iteraciones de cada bucle, así como también se ha hecho uso de la Regla del Producto.

Si observamos bien, podemos ver que la función consta de tres bucles for anidados. El cuerpo del bucle (1) : x++ , es lo mismo que x = x+1 y, por tanto, su eficiencia es $O(1)$. Por otra parte, vemos que el bucle comienza en k=1 y en cada iteración k *=2, es decir, parará cuando y solo cuando $2^k > n \iff$ lleve a cabo $\log_2 n$ iteraciones. Por tanto, concluimos que la eficiencia de este bucle es $O(\log_2 n)$. El bucle 2, si observamos bien, realiza 4 iteraciones pues j += n/4 mientras $j \leq n \iff \frac{n}{4} \cdot T = n \iff T = 4$. La eficiencia total del bucle 2 anidado con el 1 es, por la regla del producto, $O(\log_2 n)$ ; eficiencia bucle (2): $O(1)$

Por su parte, el bucle 3 lleva a cabo $\frac{n}{4}$ iteraciones por el hecho de que en cada iteración i+=4 , hasta que $i \leq n$ con i = 1, luego, la eficiencia de ese bucle es $O(n/4) = O(n)$.

En conclusión, dado el anidamiento de los tres bucles, la eficiencia total de la función, aplicando la Regla del Producto, es $O(n \log_2 n)$

**(b)**

```
int eficiencia2 (bool existe)
 {
    int sum2=0; int k,j,n;  ⇒ O(1)

    if (existe)
          for(k=1; k<=n; k*=2) ⇒ O(log2n)
               for(j=1; j<=k; j++) ⇒ O(k)
                     sum2++; ⇒ O(1)
    else
           for(k=1; k<=n; k*=2) ⇒ O(log2n)
             for(j=1; j<=n; j++) ⇒ O(n)
                     sum2++; ⇒ O(1)
  return sum2; ⇒ O(1)
 }
```

⇒ O(k) → Regla del Producto → ⇒ O(k log₂ n)

⇒ O(n) → Regla del producto → ⇒ O(n log₂ n)

⇒ O(n log₂ n) ← Regla de la Suma

En este caso estamos ante una función la cual distinguimos tres partes: la primera son las declaraciones y la asignación, que, al ser operaciones simples tienen eficiencia O(1). La segunda es una estructura condicional; analicémosla:

•) If ⇒ Constituido por dos bucles for anidados. El cuerpo del más interno es sum2++, que claramente tiene eficiencia O(1). Dicho bucle comienza en j=1, siempre que j ≤ k, haciendo en cada iteración j++, luego, está claro que el bucle se ejecuta k veces, por tanto, la eficiencia es O(k). Por su parte, el bucle más externo hace en cada iteración k*=2 con k=1 y k≤n luego, se estará ejecutando hasta que $2^k > n \iff$ el bucle realiza $\log_2 n$ iteraciones. Por tanto, aplicando la regla del producto, obtenemos que la eficiencia del If es $O(k \log_2 n)$.

•) else ⇒ Lo constituyen dos for anidados tales que el más interno tiene por cuerpo sum2++, es decir, un cuerpo de eficiencia O(1) y, claramente, realiza n iteraciones, por el razonamiento anterior para el if. Luego, dicho bucle tiene eficiencia O(n). Por su parte, el otro bucle lleva a cabo, por la misma razón que en el caso del bucle más externo del if, $\log_2 n$ iteraciones. Por consiguiente, aplicando la regla del producto, obtenemos que la parte else tiene una eficiencia $O(n \log_2 n)$

La tercera parte (última en el return sum2 ⇒ O(1)

En conclusión, si aplicamos la regla de la suma (y teniendo en cuenta que debemos coger el peor de los casos), dado que $n \geq K$, deducimos que la eficiencia de la función es $O(n \log_2 n)$

**(c)**

```
void eficiencia3 (int n)
{
  int j; int i=1; int x=0;          ⇒ O(1)
   do{
      j=1;                          ⇒ O(1)
      while (j <= n){               } ⇒ O(log2 n)   } ⇒ O(n log2 n)
         j=j*2;                     ⇒ O(1)
         x++;                       ⇒ O(1)
      }
      i++;                          ⇒ O(1)
    }while (i<=n);
}
```

En esta función, en primer lugar aparece una primera línea de código en la cual hay declaraciones y asignaciones; operaciones simples con eficiencia $O(1)$. A continuación, tenemos un ciclo de while en el que hay anidado un ciclo while, a parte de dos operaciones simples ($j=1$ y $i++$) con eficiencia $O(1)$. Analicemos el while: el cuerpo del bucle se basa en dos operaciones simples que hacen que dicho cuerpo tenga eficiencia $O(1)$. En cuanto al número de iteraciones, observamos que $j=1$ con $j \leq n$, haciendo en cada iteración $j=j*2$, con lo cual esto me dice que el bucle parará cuando $2^i > n$, es decir, cuando se hagan $\log_2 n$ iteraciones. Luego, la eficiencia del bucle while es $O(\log_2 n)$. Como por su parte el bucle do while realiza $n$ iteraciones ($i=1$, $i \leq n$, $i++$), afirmamos que su eficiencia es $O(n)$. Como hay anidamiento de bucles aplicamos la regla del producto y obtenemos que la eficiencia de la función es $O(n \log_2 n)$.

**(d)**

```
void eficiencia4 (int n)
{
int j; int i=2; int x=0;            ⇒ O(1)
 do{
     j=1;                           ⇒ O(1)
     while (j <= i){                } ⇒ O(log2 i)   } ⇒ O(log2 i)   } ⇒ Σ_{i=2}^{n} log2 i
        j=j*2;                      ⇒ O(1)
        x++;                        ⇒ O(1)   } ⇒ O(1)
     }
     i++;                           ⇒ O(1)
   }while (i<=n);
 }
```

↳ n-1 iteraciones

La función de este apartado consta de una primera línea con operaciones simples que tiene eficiencia $O(1)$ y de un ciclo de while constituido por un ciclo while y dos sentencias de operaciones simples que, como tales, tienen eficiencia $O(1)$. Podemos determinar las iteraciones del bucle de while: $i=2$ mientras que $i \leq n$, con $i+1$; claramente realiza $n-1$ iteraciones. Centrémonos en el bucle while anidado en el de while. Dicho ciclo consta de dos sentencias en las que se llevan a cabo operaciones simples y, por consiguiente, tienen eficiencia $O(1)$, luego, el cuerpo del bucle tiene eficiencia $O(1)$. En cuanto al número de iteraciones del while, $j=1$, con $j \leq i$ y $j = j*2$, luego, se efectúan $\log_2 i$ iteraciones. En consecuencia la eficiencia de dicho ciclo es $O(\log_2 i)$. Como el de while realiza $n-1$ iteraciones yendo de $i=2$ hasta $n$, concluimos que la eficiencia total de la función es $O(\sum_{i=2}^{n} \log_2 i)$

Si calculamos $\sum_{i=2}^{n} \log_2 i$ obtenemos lo siguiente:

$\sum_{i=2}^{n} \log_2 i = \log_2 (n!)$. Por tanto, concluimos que la eficiencia es $O(\log_2 n)$

*2.- Considerar el siguiente segmento de código con el que se pretende buscar un entero **x** en una lista de enteros L de tamaño n (el bucle **for** se ejecuta **n veces**):*

```
void eliminar (Lista L, int x)
   {
      int aux, p;
      for (p=primero(L); p!=fin(L);)
          {
             aux=elemento (p,L);
             if (aux==x)
                borrar (p,L);
             else p++;
          }
   }
```

Analizar la eficiencia de la función **eliminar** si:

(a) **primero es O(1)** y **fin, elemento y borrar son O(n)**. ¿Cómo mejorarías esa eficiencia con un ligero cambio en el código?

```
void eliminar (Lista L, int x)
   {
      int aux, p; ⇒ O(1)
      for (p=primero(L); p!=fin(L);)   ⇑ O(n)
          {
             aux=elemento (p,L); ⇒ O(n)
             if (aux==x) ⇒ O(1)
                borrar (p,L); ⇒ O(n)
             else p++; ⇒ O(1)
          }
   }
```

⇒ O(n) (Regla de la Suma) ⇒ O(n²) (Regla del producto) ⇒ O(n²)

En primer lugar, esta función está constituida por una primera línea de código en la que se realizan declaraciones de variable que son operaciones simples que tienen eficiencia O(1). Contiene además un bucle for en el cual, tanto la asignación p = primero(L), como la condición del if ((i)(aux = x)) y el mismo cuerpo del else, se corresponden con operaciones simples de eficiencia O(1). Por su parte, la condición del bucle tiene eficiencia O(n) (porque fin es O(n)) y, los funciones elemento y borrar también lo son hacen que el cuerpo del bucle sea O(n), por la regla de la suma pero como se ejecuta n veces, tenemos que la eficiencia es $O(n^2)$.

Para mejorar la eficiencia de esta función lo que haría es declarar fuera del bucle una variable de tipo entero en la que guardaría el valor de la función fin, aplicada a la lista L.

con el objetivo de evitar que en cada iteración se llame a la función fin. El orden de eficiencia en cualquier caso seguiría siendo $n^2$, pues el cuerpo del bucle sigue siendo $O(n)$ y el bucle seguiría ejecutándose $n$ veces. A pesar de ser $O(n^2)$, la eficiencia claramente mejoraría.

(b) **primero, elemento y borrar son O(1)** y **fin es O(n)**. ¿Cómo mejorarías esa eficiencia con un ligero cambio en el código?

```
void eliminar (Lista L, int x)
   {
      int aux, p;
      for (p=primero(L); p!=fin(L);)
          {
             aux=elemento (p,L);
             if (aux==x)
                borrar (p,L);
             else p++;
          }
   }
```

⇒ O(1) (aux, p) ⇒ O(n) (for)
⇒ O(1) (aux=elemento)
⇒ O(1) (if)
⇒ O(1) (borrar)
⇒ O(1) (else p++)
⇒ O(1) (Regla de la Suma (Estructura condicional))
⇒ O(1) (Regla de la Suma)
⇒ $O(n^2)$
⇒ $O(n^2)$

En este caso, el bucle for de la función consta de un cuerpo que, en su totalidad, teniendo en cuenta las operaciones simples y lo que se nos indica en el enunciado, tiene eficiencia $O(1)$. La función fin, en la condición del bucle, es $O(n)$ y, dado que se toma como una sentencia más y teniendo en cuenta las $n$ iteraciones que se llevan a cabo, afirmamos que el bucle tiene eficiencia $O(n^2)$. Aplicando la regla de la suma, deducimos que la eficiencia de la función es $O(n^2)$.

Para mejorar la eficiencia de la función lo que haría es lo mismo que en el caso anterior: declaro fuera del bucle una variable entera en la que guardo el valor de fin(L) (int variable = fin(L)). De esta forma, tendría una sentencia fuera del bucle con eficiencia $O(1)$ y un bucle con eficiencia $O(n)$ (el cuerpo del bucle es $O(1)$ y $n$ iteraciones), luego, aplicando la Regla de la Suma, obtengo una eficiencia total de $O(n)$.

Con lo cual, claramente, hay una gran mejoría en cuanto a la eficiencia de la función.

(c ) **todas las funciones son O(1)**. ¿Puede en ese caso mejorarse la eficiencia con un ligero cambio en el código?

```
void eliminar (Lista L, int x)
   {
      int aux, p;
      for (p=primero(L); p!=fin(L);) ⇒ O(1)
          {
             aux=elemento (p,L); ⇒ O(1)
             if (aux==x) ⇒ O(1)
                borrar (p,L); ⇒ O(1)
             else p++; ⇒ O(1)
          }
   }
```

⇒ O(1) (Regla de la suma)

⇒ O(n) (Regla de la suma) } se repite n veces

⇒ O(n)

En este caso todas las sentencias y todas las funciones son O(1), luego, el cuerpo del bucle es O(1). Como se ejecuta n veces, deducimos que la eficiencia del bucle es O(n). Aplicando la regla de la suma con la eficiencia de la línea de declaraciones (O(1)), obtenemos que la función tiene eficiencia O(n).

Para mejorar la eficiencia de la función lo que yo haría es ahorrarme una asignación. En particular, aux = ejemplo (p, L). Evitaría declarar una variable auxiliar y trabajaría únicamente con el valor de aplicar la función ejemplo a p y L. Por otra parte, también declararía una variable auxiliar fue en la que guardar fin(L) para evitar llamarla las n iteraciones del bucle.
En definitiva, la eficiencia de la función se mantendría en O(n), pero dentro de esa familia, sería más eficiente que en el caso inicial.