# Reto 1. Eficiencia

**Daniel Alconchel Vázquez**

**EJERCICIO 1**

```
1  //Apartado a
2  void eficiencia1(int n){
3      int x=0;    // O(1)
4      int i,j,k;  // O(1)
5      for(i=1; i<=n; i+=4)
6          for(j=1; j<=n; j+=n/4)
7              for(k=1; k<=n; k*=2)
8                  x++;    // O(1)
9  }
```

Para este primer algoritmo nos centramos en los bucles, ya que el resto (incluido las condiciones de dentro de los bucles) son O(1). Analizando los bucles encontramos:

- El primer bucle (línea 5) claramente tiene eficiencia O(n) cuando el número de datos tiende a infinito
- El segundo bucle (línea 6) solo se ejecuta unas 4 veces, por lo que su eficiencia es O(1)
- En tercer bucle (línea 7) vemos que, debido a la forma en la que aumenta el contador k (k*=2), el bucle realizará $log_2(n)$ cuando $n \to \infty$, lo que equivale a O(log(n))

Como los bucles están anidados, usamos la regla del producto, donde obtenemos $O(n) \cdot O(1) \cdot O(log(n)) = O(nlog(n))$

Esta última eficiencia corresponde con la del algoritmo.

```
1  //Apartado b
2  int eficiencia2(bool existe){
3      int suma2 = 0;  //O(1)
4      int j,k,n;  //O(1)
5
6      if(existe)
7          for(k=1; k<=n; k*=2)
8              for(j=1; j<=k; j++)
9                  sum2++; //O(1)
10     else
11         for(k=1; k<=n; k*=2)
12             for(j=1; j<=n; j++)
13                 sum2++; //O(1)
14     return sum2;    //O(1)
15 }
```

Notemos que en este algoritmo encontramos dos bloques de código independientes, el bloque del if y el bloque del else, por lo que la eficiencia del algoritmo será la del bloque de mayor complejidad.

Para comenzar, todas las asignaciones y condiciones de los bucles for son O(1), por lo que para determinar que bloque tiene mayor complejidad nos centraremos, nuevamente, en los bucles. En este caso, el primer bucle de cada bloque es el mismo, por lo que nos fijamos en el segundo. Claramente, se puede observar que el segundo bucle del bloque else hace mayor número de iteraciones que el del bloque if, por lo que el bloque else será el de mayor complejidad. Centrándonos en el observamos:

- Primer bucle tiene una eficiencia de O(nlog(n)) (Mismo razonamiento que en el ejercicio anterior)
- Segundo bucle (línea 12) realiza n iteraciones, por lo que su eficiencia será O(n)

Nuevamente, aplicando la regla del producto, ya que los bucles están anidados, la eficiencia del bloque else, así como de la totalidad del algoritmo, será, nuevamente, O(nlog(n))

```
1  //Apartado c
2  void eficiencia3(int n){
3      int j;  // O(1)
4      int i = 1;  // O(1)
5      int x = 0;  // O(1)
6      do{
7          j=1;    // O(1)
8          while(j <= n){
9              j=j*2;  // O(1)
10             x++;    // O(1)
11         }
12         i++;    // O(1)
13     }while(i<=n);
14 }
```

Al igual que en los casos anteriores, la complejidad del algoritmo se encuentra en el número de iteraciones que realizan los bucles.

- Notemos que el bucle while que se encuentra anidado es similar a los bucles for que hemos visto ateriormente, cuya eficiencia era $log_2 n$, por lo que su eficiencia será O(nlog(n))
- El bucle do while realiza n iteraciones, por lo que su eficiencia será O(n)

Nuevamente, usando la regla del producto, obtenemos que la eficiencia del algoritmo es O(nlog(n))

```
1  //Apartado d
2  void eficiencia4(int n){
3      int j;      // O(1)
4      int i=2;    // O(1)
5      int x=0:    // O(1)
6      do{
7          j=1;    // O(1)
8          while(j <= i){
9              j=j*2;  // O(1)
10             x++;    // O(1)
11         }
12         i++;    // O(1)
13     }while(i<=n);
14 }
```

Nuevamente, la complejidad del algoritmo se encuentra en el número de iteraciones que realizan los bucles.

- Podemos observar que el bucle do while realiza n iteraciones, por lo que su eficiencia sera O(n)
- El bucle while es parecido al anterior, solo que en vez de realizar $log_2 n$ iteraciones, realiza $log_2 i$ iteraciones, pero como $i \to n$, entonces la eficiencia del bucle while es O(log(n))

Nuevamente, usando la regla del producto, obtenemos que la eficiencia del algoritmo es O(nlog(n))

**EJERCICIO 2**

```
1  void eliminar(Lista L, int x){
2      int aux, p; //O(1)
3      for(p=primero(L); p!=fin(L);){
4          aux=elemento(p,L);
5          if(aux==x)  // La condición es O(1)
6              borrar (p,L);
7          else
8              p++;    //O(1)
9      }
10 }
```

**Apartado a**

Para este apartado tenemos que primero es O(1) y fin, elemento y borrar son O(n), por lo que :

- La condición del bucle for (línea 3), por la regla de la suma sería $O(O(n) + O(1)) = O(n)$ (Como las 3 funciones son independientes dentro del código, la eficiencia sigue siendo O(n))
- La eficiencia del código de dentro del bucle, por la regla de la suma, es O(n)

Esto hace que la eficiencia total del bucle, sin tener en cuenta el número de iteraciones, sea O(n)

- En el peor de los casos (no se cumple la condición del if), el bucle realizará n iteraciones, por lo que la eficiencia será de O(n)

Teniendo en cuenta que la eficiencia del bucle es O(n) y este se ejecuta n veces, por la regla del producto, la eficiencia resultante será de $O(n^2)$, que, además, corresponde con la del algoritmo, ya que el resto de código es independiente, por lo que usando la regla de la suma tendríamos $O(O(n^2) + O(n)) = O(n^2)$

Un posible cambio para mejorar el código es almacenar el valor de fin(L) en una variable, así evitamos llamar a la función en cada iteración pero esto no disminuye la complejidad del algoritmo, cuya eficiencia sigue siendo de $O(n^2)$.

```
1  void eliminar(Lista L, int x){
2      int aux, p; //O(1)
3      int fin = fin(L); //O(n)
4      for(p=primero(L); p!=fin;){ // O(1)
5          aux=elemento(p,L);  //O(n)
6          if(aux==x)  // La condición es O(1)
7              borrar (p,L);   //O(n)
8          else
9              p++;    //O(1)
10     }
11 }
```

**Apartado b**

Para este apartado tenemos que primero, elemento y borrar son O(1) y fin es O(n), por lo que:

- La condición del bucle for (línea 3), por la regla de la suma sería $O(O(n)+O(1))=O(n)$
- La eficiencia del código de dentro del bucle, por la regla de la suma, es O(1)

Esto hace que la eficiencia total del bucle, sin tener en cuenta el número de iteraciones, sea O(n)

- En el peor de los casos (no se cumple la condición del if), el bucle realizará n iteraciones, por lo que la eficiencia será de O(n)

Teniendo en cuenta que la eficiencia del bucle es O(n) y este se ejecuta n veces, por la regla del producto, la eficiencia resultante será de $O(n^2)$, que, además, corresponde con la del algoritmo, ya que el resto de código es independiente, por lo que usando la regla de la suma tendríamos $O(O(n^2)+O(n))=O(n^2)$

Al igual que el apartado aterior, un posible cambio para mejorar el código es almacenar el valor de fin(L) en una variable, así evitamos llamar a la función en cada iteración pero, en esta ocasión, si permite mejorar la eficiencia del código, ya que la eficiencia de la condición del bucle pasaría a ser O(1), por lo que, por la regla del producto, la eficiencia del bucle pasaría a ser O(n), por lo que tendriamos $O(O(n)+O(n))=O(n)$

```
1  void eliminar(Lista L, int x){
2      int aux, p; //O(1)
3      int fin = fin(L);   //  O(n)
4      for(p=primero(L); p!=fin;){ // O(1)
5          aux=elemento(p,L); // O(1)
6          if(aux==x)  // La condición es O(1)
7              borrar (p,L);   // o(1)
8          else
9              p++;    //O(1)
10     }
11 }
```

**Apartado c**

Para este apartado tenemos que todas las funciones son O(1), por lo que:

- La condición del bucle for (línea 3) es O(1)
- La eficiencia del código de dentro del bucle, por la regla de la suma, es O(1)

Esto hace que la eficiencia total del bucle, sin tener en cuenta el número de iteraciones, sea O(1)

- En el peor de los casos (no se cumple la condición del if), el bucle realizará n iteraciones, por lo que la eficiencia será de O(n)

Teniendo en cuenta que la eficiencia del bucle es O(1) y este se ejecuta n veces, por la regla del producto, la eficiencia resultante será de O(n), que, además, corresponde con la del algoritmo, ya que el resto de código es independiente, por lo que usando la regla de la suma tendríamos $O(O(n) + O(1)) = O(n)$

En este caso, ya que todas las funciones son O(1), es imposible mejorar la eficiencia del código.