Reto 1: Eficiencia

Pablo Olivares Martínez

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

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

```
//Apartado b)
int eficiencia2 (bool existe)
{
    int sum2=0; | O(1)
    int k,j,n;
    if (existe)                        -|               -|
    for(k=1; k<=n; k*=2)                | O(n*log(n))    |
        for(j=1; j<=k; j++)-|O(n)       |                |
            sum2++; |O(1)  -|          -|                |
                                                         | O(n*log(n))
    else                               -|                |
        for(k=1; k<=n; k*=2)            | O(n*log(n))    |
            for(j=1; j<=n; j++)-|O(n)|                   |
                sum2++; |O(1)  -|   -|                   |
                                                         |
    return sum2;                       -| O(1)          -|
}
```

```
//Apartado c)
void eficiencia3 (int n)                                          -|
{                                                                  |
    int j; int i=1; int x=0; | O(1) //cada asignación              |
    do{                                              -|            |
        j=1; | O(1)                                   |            |
        while (j <= n){          -|                   |            | O(n*log(n))
            j=j*2; | O(1)         | O(log(n))        | O(n*log(n))|
            x++;   | O(1)         |                  |            |
        }                        -|                   |            |
        i++; | O(1)                                   |            |
    }while (i<=n);                                  -|           -|
}
```

```
//Apartado d)
void eficiencia4 (int n)
 -|
```

```
{
  |
    int j;
  |
    int i=2; |O(1)
  |
    int x=0; | O(1)
  |
    do{                                                                    -|
  |
        j=1;     | O(1)                                                     |
             |
        while (j <= i){          -|                                       |
  |
            j=j*2;  | O(log(n))  | O(1*log(n)) //While es de 1 a 2 => O(1)   |
O(n*log(n)) |
            x++;    | O(1)       |                                        |
  |
        }                        -|                                       |
  |
        i++;                      | O(1)                                   |
  |
    }while (i<=n);                                                         -|
 -|
}

//Para la eficiencia final, es O(n*log(n)) ya que aplicando la regla de la suma
con la asignación de valores de arriba, sale este resultado (no lo he puesto
directamente en su sitio porque no cabía).
```

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:

Antes de comenzar el ejercicio, me gustaría aclarar que he obviado escribir O(1) en los condicionales *if* y *else* ya que las funciones que se encuentran en su interior son de peor o igual eficiencia que éstas y así dejar un texto más limpio.

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

```
void eliminar (Lista L, int x)
{
    int aux, p;
    for (p=primero(L); p!=fin(L);)   | O(n)   -|
    {                                          |
        aux=elemento (p,L); |O(n)   -|         |
        if (aux==x)         -|       | O(n)    | O(n²)
            borrar (p,L);   -|O(n)   |         |
        else p++;            |O(1)   -|        -|
    }
}
```

Para mejorar la eficiencia del código, sustituiría la función fin(L) del bucle por una variable con el mismo valor para evitar llamar a la función cada vez que se repita el bucle, aunque en efectos reales apenas supone una notable mejoría.

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

```
void eliminar (Lista L, int x)
{
    int aux, p;
    for (p=primero(L); p!=fin(L);)   | O(n)   -|   //fin(L) es O(n)
    {                                          |
        aux=elemento (p,L); |O(1)   -|         |
        if (aux==x)         -|       | O(1)    | O(n²)
            borrar (p,L);   -|O(1)   |         |
        else p++;            |O(1)   -|        -|
    }
}
```

En este caso, la mejora la realizaría de la misma forma que en el caso anterior, pero al tener un peor rendimiento fin(L) en el caso b, finalmente nos quedaría O(n) gracias a sacar la función y sustituirla por una variable, lo que supondría una buena mejora en la eficiencia.

(c) todas las funciones son O(1). ¿Puede en ese caso mejorarse la eficiencia con un solo 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)    | O(n)
            borrar (p,L);   -|O(1)   |         |
        else p++;            |O(1)   -|        -|
    }
}
```

Al ser la eficiencia de todas las funciones del bucle O(1), por la regla de la suma obtenemos que es de eficiencia O(1), y por la del producto con el bucle *for*, obtenemos que la eficiencia es O(n). Esta eficiencia es muy improbable que se pueda mejorar con tan solo un cambio, pues tendríamos que sustituir el bucle ya que la eficiencia de las funciones que contiene es muy buena (constante).