ESTRUCTURA DE DATOS. RETO 1: EFICIENCIA

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)
      for(i=1; i<=n; i+=4)
        for(j=1; j<=n; j+=[n/4])
          for(k=1; k<=n; k*=2)
            x++;                               ] O(1)
}
```
] $O(\log_2(n))$] $O(\log_2(n))$] $O(n\log_2(n))$] $O(n\log_2(n))$

Eficiencia de la función void eficiencia1 (int n): $O(n\log_2(n))$

b)
```
int eficiencia2 (bool existe)
{
  int sum2=0; int k,j,n;                       ] O(1)
  if (existe)
     for(k=1; k<=n; k*=2)
        for(j=1; j<=k; j++)
           sum2++;                             ] O(1)
  else
     for(k=1; k<=n; k*=2)
        for(j=1; j<=n; j++)
           sum2++;                             ] O(1)
  return sum2;                                 ] O(1)
}
```
] $O(k)$] $O(n\log_2(n))$ $\circledast_1$

] $O(n)$] $O(n\log_2(n))$

] $O(n\log_2(n))$

Eficiencia de la función int eficiencia2 (bool existe): $O(n\log_2(n))$

$\circledast_1$ En el peor de los casos k=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))
      j=j*2;                        O(1)
      x++;                          O(1)
    }
    i++;                            O(1)
  } while (i<=n);                                   O(n log2(n))
}
```

Eficiencia de la función void eficiencia3 (int n) : $O(n \log_2(n))$

d) void eficiencia4 (int n)

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

Eficiencia de la función void eficiencia4 (int n) : $O(n \log_2(n))$

⊛2 En el peor de los casos 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 solo 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)   ] O(n)
      else p++;                        ] O(1)   ] O(n)   ] O(n²)   ] O(n²)
    }
}
```

Eficiencia de la función void eliminar (Lista L, int x): $O(n^2)$

Para mejorar la eficiencia del código, almacenaría antes del bucle for el valor de la función fin(L) en una variable de tipo entera. De esta forma, no se tendría que ejecutar dicha función en cada iteración del bucle for. Luego, la condición del bucle for pasaría a tener $O(1)$ y no $O(n)$, pero la función eliminar seguirá siendo $O(n^2)$ ya que el cuerpo del bucle sigue siendo $O(n)$ y el bucle se repite n veces.

Recordemos que en un bucle for:

$$O(Ini(n)) + O(Con(n)) + O(Ite(n)) \cdot [O(Cu(n)) + O(Inc(n)) + O(Con(n))] \quad (1)$$

donde:

$O(Ini(n))$: Inicialización

$O(Con(n))$: Condición

$O(Ite(n))$: Iteración

$O(Cu(n))$: Cuerpo

$O(Inc(n))$: Incremento

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;                                   ] O(1)
  for (p = primero(L); p != fin(L);)            ] O(n)
  {
    aux = elemento(p, L);                       ] O(1)
    if (aux == x)                               ] O(1)
      borrar(p, L);                             ] O(1)   ] O(1)   ] O(n²)   ] O(n²)
    else p++;                                   ] O(1)
  }
}
```

Eficiencia de la función void eliminar (Lista L, int x): $O(n^2)$

Para el cálculo de la eficiencia del bucle for hemos usado (1). Para mejorar la eficiencia del código, almacenaría antes del bucle for el valor de la función fin(L) en una variable de tipo entera. De esta forma, no se tendría que ejecutar dicha función en cada iteración del bucle for. Luego, la condición del bucle for pasaría a ser $O(1)$ y no $O(n)$, y así todo el bucle for sería $O(n)$ y no $O(n^2)$, siendo, por tanto, la función eliminar $O(n)$ y no $O(n^2)$.

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;                                   O(1)
  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)   O(1)   O(1)   O(n)
      else p++;                                 O(1)
    }
}
```

Eficiencia de la función void eliminar (Lista L, int x): $O(n)$

Para el cálculo de la eficiencia del bucle for hemos usado (1). En este caso no puede mejorarse la eficiencia del código con un solo cambio ya que la función eliminar es $O(n)$ debido a las n iteraciones del bucle for y eso no puede cambiar ya que, si pensamos en el peor de los casos, el elemento a eliminar se encontrará en la última posición de la lista L.