Alejandro Rubio Martínez

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

(a)

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

Vemos como la eficiencia en la línea 3 es O(1). Ahora calculamos la del bucle de las líneas 4-7, y para ello primero vemos el de la 5-7, que a su vez necesitamos primero el de la 6-7. Este último vemos que tiene una eficiencia de O(log2 n). El bule de la línea 5-7 se realiza siempre 4 veces, por lo que sería O(4), pero multiplicado por el de su interior no queda O(log2 n). Con esto vemos que el bucle de la 4-7 itera n veces, por lo que al multiplicarlo con lo de su interior obtenemos una eficiencia de O(nlog2 n), que será la de toda la función al sumar con las otras funciones.

(b)

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

La eficiencia de las líneas 3 y 12 son O(1). Por tanto la eficiencia de la función será la del condicional, que tendrá la eficiencia de su peor caso. Analizamos primero el primer caso. La condición es O(1). En su cuerpo encontramos un bucle for con otro anidado de eficiencia O(n) en su peor caso, dejándonos así eficiencia de O(nlog2 n) al multiplicarlo con el for de la 5. Veamos ahora el else, para el que nos volvemos a encontrar otros dos bucles for anidados, el de la línea 10 con una eficiencia O(n) que al multiplicarlo con el de la línea 9 obtenemos de nuevo O(nlog2 n), quedándonos esta como la eficiencia de la función.

(c)

```
1. void eficiencia3 (int n)
2. {
3.       int j; int i=1; int x=0;
4.       do {
5.              j=1;
6.              while (j <= n) {
7.                     j=j*2;
8.                     x++;
9.              }
10.             i++;
```

```
11.     } while (i <= n);
```

Vemos que la línea 3 es de eficiencia O(1). Analicemos ahora el while de la línea 4-10. Para ello vemos que la línea 5 es O(1), y pasamos a analizar el bucle de la línea 6-9. El cuerpo del mismo es de eficiencia O(1), pero itera log2 n veces por lo que tiene eficiencia O(log2 n). Con esto sabemos que el primer bucle tiene una eficiencia O(nlog2 n) pues itera n veces, que será la eficiencia de la función.

(d)

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

Vemos que la línea 3 tiene eficiencia O(1), así que pasemos a analizar la eficiencia del bucle de la línea 4-12. La línea 5 y 11 son de eficiencia O(1), así que veamos la eficiencia del while de las líneas 6-9. Este en el peor de sus casos tendrá una eficiencia O(log2 n), y así ya sabemos que el primer bucle tiene una eficiencia O(nlog2 n), que será la de la función en total.

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):

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

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

Tenemos que la línea 3 es O(1). Para ello veamos la eficiencia del bucle for de la línea 4-10. Tenemos que la inicialización es O(1) y la comprobación es O(n). El cuerpo tiene un if que es O(n) y la línea 6 que también es O(n). Con esto nos queda que es O(n) en total al sumarlo y al multiplicarlo por la cabecera nos queda O(n^2), que será la eficiencia de la función completa.

No hay forma de mejorar la eficiencia del código, pero sí es recomendable crear una variable int f=fin(L); y sustituirla en la comprobación del bucle.

(b) primero, elemento y borrar son O(1) y fin es O(n). ¿Cómo mejorarías esa eficiencia con un solo cambio en el código?
En este caso la función de la inicialización del bucle tenemos que es O(1), pero al seguir siendo O(n) la comprobación no nos mejora la eficiencia. Ahora en el cuerpo del bucle todo es O(1) así que el bucle al multiplicar nos queda O(n^2) pues se itera n veces, que será también la eficiencia de la función.

En este caso si realizamos el cambio sugerido en el apartado (a) si obtendremos una mejora de la eficiencia a un caso O(n).

c) todas las funciones son O(1). ¿Puede en ese caso mejorarse la eficiencia con un solo cambio en el código?
En este caso todas las líneas del código son O(1), pero el bucle for iterará n veces siendo de eficiencia tipo O(n), que será la de la función.

En este caso no podemos reducir la eficiencia de la función pues para ello tendríamos que realizar un cambio del planteamiento del algoritmo para no hacer un bucle for de n iteraciones.