UNIVERSIDAD DE GRANADA

UNIVERSIDAD DE GRANADA

2°C

GRADO EN INGENIERÍA INFORMÁTICA

Reto 1: Eficiencia

Autor: Blanca Abril González

Asignatura: Estructura de Datos

Índice

1. Ejercicio 1

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

a)

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

En la linea 1 es O(1) ya que se esta produciendo una asignación en x = 0. En la linea 5 es O(1). En la linea 4 se esta recorriendo solo una parte de n por lo tanto sería O(log_2 n). En la linea 3 es siempre 4. En la linea 2 es N.

Por lo tanto $n * 4 * log_2 n$ es $O(n * log_2 n)$.

b)

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

En la linea 1 es O(1) ya que se esta produciendo una asignación en sum2 = 0.
Primero miramos el if, en la linea 4 que vemos que recorre los valores de k y k en la linea 3 no recorre todos los valores de n. Por lo tanto es O($log_2(n)$)
Ahora miramos el else,en la linea 8 vemos que se recorren todos los valores de n por lo tanto es O(n) y en la linea 7 se recorren O($log_2(n)$). Por lo tanto la eficiencia es $O(n * log_2(n))$.

Por lo tanto la eficiencia es: $O(n * log_2(n))$ porque se coge el peor caso que en este caso es el else.

c)

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

En este caso, el do while es O(n). En la linea 3 se produce una asignación por lo tanto es O(1). En la linea 4 y 5 se puede ver que j no coge todos los valores por lo tanto es O($log_2(n)$).

Por lo tanto la eficiencia final es $O(n * log_2(n))$.

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

En este caso, el do while tenemos un O(n) porque en el peor caso se recorre todo n. En la linea 3 tenemos un O(1) porque se ha producido una asignación. Dentro del while, en la linea 5 hace que el while de dentro se reduzca a O($log_2(n)$) .

Por lo tanto en el peor caso, la eficiencia final es $O(n * log_2(n))$.

2. Ejercicio 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++;
        }
}
```

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

No se puede mejorar con un solo cambio por lo tanto la eficiencia es $O(n^3)$.

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

La eficiencia sin realizar ningún cambio sería $O(n^2)$ pero tras el cambio sería O(n).

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

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

No puede modificarse más ya que si todos son O(1) siempre va a salir O(n).