

# Programación Avanzada – Complejidad algorítmica

---

Diciembre, 2016



# EFICIENCIA Y COMPLEJIDAD

Para medir el rendimiento y comportamiento de un algoritmo se considera:

- Simplicidad
- Uso eficiente de los recursos

# EFICIENCIA

La eficiencia se puede medir en base a dos parámetros:

**El espacio** = memoria que utiliza.

**El tiempo**= lo que tarda en ejecutarse.

Estos parámetros sirven para determinar el coste de la solución-algoritmo.

Permiten comparar algoritmos entre sí, => más adecuado

# EFICIENCIA TEMPORAL

El **tiempo de ejecución** de un algoritmo depende de:

- Datos de entrada suministrados.
- El compilador para crear el programa objeto, la naturaleza y rapidez de las instrucciones máquina del procesador
- Complejidad intrínseca del algoritmo.



# TIEMPO

Se posee dos estudios del tiempo:

- **A Priori:** Proporciona una medida teórica, que consiste en obtener una función que acote (por arriba o por abajo) el tiempo de ejecución del algoritmo para unos valores de entrada dados.
- **A Posteriori:** Ofrece una medida real, consistente en medir el tiempo de ejecución del algoritmo para unos valores de entrada dados y en un ordenador concreto.

# Algunas consideraciones

- **Tamaño de la entrada:** Número de componentes sobre los que se va a ejecutar el algoritmo.
- **Unidad de tiempo:** No puede ser expresada en una unidad de tiempo concreta.
- **Notación:**  $T(n)$

El tiempo de ejecución de un algoritmo para una entrada de tamaño  $n$ .

# TIEMPO DE EJECUCIÓN

- $T(n)$ : Indica el número de instrucciones ejecutadas por un ordenador idealizado.
- Independiente del ordenador a utilizar. (a priori).



# PRINCIPIO DE INVARIANZA

Dado un algoritmo y dos implementaciones suyas  $I_1$  e  $I_2$ , que tardan  $T_1(n)$  y  $T_2(n)$  segundos respectivamente, el Principio de Invarianza afirma que :

- Existe una constante real  $c > 0$  y un número natural  $n_0$  tales que para todo  $n \geq n_0$  se verifica que  $T_1(n) \leq cT_2(n)$ .

En otras palabras, el tiempo de ejecución de dos implementaciones distintas de un algoritmo dado no difiere más que en una constante multiplicativa



# CASOS

En muchos programas el tiempo de ejecución es en realidad una función de la entrada específica, y no sólo del tamaño de ésta.

Casos posibles:

- Coste Peor.
- Coste Mejor.
- Coste promedio.



# TIEMPO DE EJECUCIÓN

- $T(n)$  Es una función que mide el número de operaciones elementales que realiza el algoritmo para un tamaño de entrada dado.

# OPERACIONES ELEMENTALES

OE: Son aquellas que el ordenador realiza en tiempo acotado por una constante. Estas pueden ser:

- Operaciones aritméticas básicas.
- Asignaciones a variables de tipo predefinido por el compilador.
- Saltos (llamadas a funciones y procedimientos, retorno desde ellos, etc.)
- Las comparaciones lógicas.
- El acceso a estructuras indexadas básicas, como son los vectores y matrices.

**Cada una de ellas se contará como 1 OE**



# EJEMPLO

```
int  Buscar(int vector, int c , int n) {  
    int j;  
    j:=0;  
    while ( (a[j]<c)  &&  (j<n)  )  
        j:=j+1;  
    if a[j]=c then  
        return j;  
    else  return 0;  
}
```

**1 OE:** 1 asignación

**4 OE:** 2 comparaciones, 1 acceso al vector  
1 and.

**2 OE:** 1 incremento, 1 asignación.

**2 OE:** 1 condición, 1 acceso.

**1 OE:** si la condición se cumple

**1 OE:** si la condición es falsa

# Coste mejor

```
int  Buscar(int vector[10], int c , int n) {
```

```
    int j;
```

```
1:   j:=0;
```

1 OE: 1 asignación

```
2:   while ((a[j]<c) && (j<n) )
```

2 OE: Solo la mitad

```
3:       j:=j+1;
```

```
4:       if a[j]=c then
```

2 OE: 1 condición, 1 acceso.

```
5:           return j;
```

1 OE: si la condición se cumple

```
6:       else return 0;
```

```
}
```

**(Tn)=1+2+2+1=6**

# Coste mejor razón

1: Se ejecuta.

2: Se ejecuta sólo la primera parte de la condición, ya que se evalúa de izquierda a derecha y corta la siguiente, ya que no necesita que se evalúe los otros términos.

4-6: Se evaluará y se ejecutará la correspondiente.



# Coste peor

```
int  Buscar(int vector[10], int c , int n) {
```

```
    int j;
```

1 OE: 1 asignación

```
1:   j:=0;
```

```
2:   while ((a[j]<c) && (j<n))
```

4 OE

Bucle se repite n-1 veces

```
3:       j:=j+1;
```

1más

2 OE

```
4:       if a[j]=c then
```

2 OE: 1 condición, 1 acceso.

```
5:           return j;
```

```
6:       else return 0;
```

1 OE: si la condición se cumple

```
}
```

$$T(n) = 1 + \left( \left( \sum_{i=1}^{n-1} (4+2) \right) + 4 \right) + 2 + 1$$

# Coste peor

- Se ejecuta la línea 1.
- El bucle se repite  $n-1$  veces hasta que se cumpla la segunda condición.
- Cada iteración del bucle contiene las líneas 2 y 3, más una ejecución adicional de la línea 2 que ocasiona la salida del bucle.
- Después se evalúa la condición de la línea 4.
- Acaba al ejecutarse la línea 6.



# Coste medio

El bucle se ejecutará un número de veces entre 0 y  $n-1$ .

Se supone que cada una tiene la misma probabilidad de suceder. Se supone a priori que son equi-probables y cada una tiene una probabilidad de  $1/n$

El número medio de veces que se efectúa el bucle es de

$$\sum_{i=0}^{n-1} i \frac{1}{n} = \frac{n-1}{2}.$$

Entonces:

$$T(n) = 1 + \left( \left( \sum_{i=1}^{(n-1)/2} (4+2) \right) + 2 \right) + 2 + 1$$

# Observaciones

- Caso mejor: Cuando el elemento está en la primera posición del vector.
- Caso peor: Cuando el elemento no está en el vector.
- Caso medio: Cuando cada posición del vector tiene la misma probabilidad de poseer el valor o que no se encuentre.



# Coste esperado

- $T_{max}(n)$ : Representa la complejidad temporal en el peor de los casos.
- $T_{min}(n)$ : Representa la complejidad en el mejor de los casos posibles.
- $T_{med}(n)$ : Expresa la complejidad temporal en el caso promedio. Para su cálculo se suponen que todas las entradas son equiprobables.

$$T_{min}(n) \leq T(n) \leq T_{max}(n)$$

# Reglas básicas para el cálculo del número de oe

- OE posee el orden 1. la constante  $c$  del principio de Invarianza dependerá de la implementación particular, pero en este caso su valor será 1.
- El tiempo de ejecución de una secuencia consecutiva de instrucciones se calcula sumando los tiempos de ejecución de cada una de las instrucciones.
- El tiempo de ejecución de la sentencia case es  $T = T(C) + \max\{T(S1), T(S2), \dots, T(Sn)\}$ :  $T(C)$  incluye el tiempo de comparación con  $v1, v2, \dots, vn$ .
- El tiempo de ejecución de la sentencia if then else es.  $T = T(C) + \max\{T(S1), T(S2)\}$

# Reglas básicas para el cálculo del número de oe

- El tiempo de ejecución de un bucle de sentencia while, es

$$T = T(C) + (\text{num. instrucciones}) * (T(S) + T(C)).$$

$T(C)$  y  $T(S)$ : pueden variar en cada iteración.


- El tiempo de ejecución de las otras estructuras repetitivas deben expresarse como un while.

# Reglas básicas para el cálculo del número de oe

- El tiempo de ejecución de una llamada a una función  $F(P1..Pn)$  es 1 por la llamada, más el tiempo de evaluación de los parámetros  $P1...Pn$ , más el tiempo que tarde en ejecutarse  $F$ .

$$T=1+T(P1)+...+T(Pn)+T(F)$$

NOTA: Se contabiliza como OE la copia de los argumentos a la pila de parámetros que se pasen por valor. Si son por referencia no se contabiliza.



# Reglas básicas para el cálculo del número de oe

- El tiempo de ejecución de las llamadas a procedimientos recursivos va a dar lugar a ecuaciones en recurrencia.



# Tarea:

- Crear un algoritmo que permita identificar el valor máximo de un vector.
- Calcular el  $T(n)$ , según las reglas analizadas.



# Fuentes

- <http://www.lcc.uma.es/~av/Libro/CAP1.pdf>
- <http://www.infor.uva.es/~jvalvarez/docencia/tema5.pdf>
- 2015, Universidad Nacional de Colombia,  
<http://www.virtual.unal.edu.co/cursos/sedes/manizales/4060024/Lecciones/Capitulo%20II/rbasicas.htm>