

Algoritmia y Complejidad

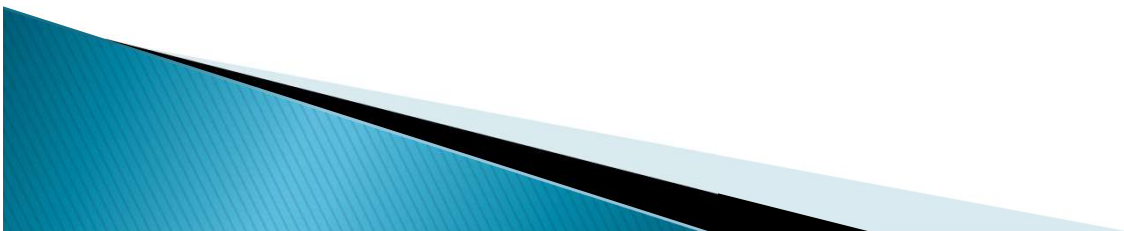
Tema 1. Introducción a la algoritmia

INTRODUCCIÓN A LA ALGORITMIA

Introducción, conceptos básicos.

Algoritmo.

- ▶ Descripción abstracta y ordenada de todas las acciones que se deben realizar, así como la descripción de los datos que deben ser manipulados por dichas acciones, para llegar a la solución de un problema.
- ▶ Debe:
 - Ser independiente tanto del lenguaje de programación en que se exprese, como del ordenador en que se vaya a ejecutar.
 - Ser claro y sencillo.
 - Indicar el orden de realización de cada paso.
 - Tener un número finito de pasos (así como un principio y un fin).
 - Ser flexible (para facilitar su mantenimiento).
 - Estar definido (de manera que si se sigue un algoritmo N veces con los mismos datos de entrada, se debe obtener el mismo resultado N veces).



INTRODUCCIÓN A LA ALGORITMIA

❖ Un ejemplo muy sencillo:

Multiplicación de dos enteros positivos
con lápiz y papel.

– En España:

$$\begin{array}{r} 981 \\ 1234 \\ \hline 3924 \\ 2943 \\ 1962 \\ 981 \\ \hline 1210554 \end{array}$$

– En Inglaterra:



$$\begin{array}{r} 981 \\ 1234 \\ \hline 981 \\ 1962 \\ 2943 \\ 3924 \\ \hline 1210554 \end{array}$$

– Ambos métodos son muy similares, los llamaremos algoritmo “clásico” de multiplicación.

INTRODUCCIÓN A LA ALGORITMIA

- Algoritmo de multiplicación “a la rusa”:

✓	981	1234	1234
	490	2468	
✓	245	4936	4936
	122	9872	
✓	61	19744	19744
	30	39488	
✓	15	78976	78976
✓	7	157952	157952
✓	3	315904	315904
✓	1	631808	631808
			<hr/> 1210554

- ♦ Ventaja: no hay que almacenar los productos parciales.
- ♦ Sólo hay que saber sumar y dividir por 2.

INTRODUCCIÓN A LA ALGORITMIA

- Todavía otro algoritmo:

- ♦ De momento, exigimos que ambos números tengan igual nº de cifras y que éste sea potencia de 2.

Por ejemplo: 0981 y 1234.

- ♦ En primer lugar, partimos ambos números por la mitad y hacemos cuatro productos:

	multiplicar	desplazar	resultado
1)	09 12	4	108 ····
2)	09 34	2	306 ··
3)	81 12	2	972 ··
4)	81 34	0	2754
			<hr/> 1210554

dobles del nº de cifras	nº de cifras
-------------------------	--------------

Es decir, hemos reducido un producto de nºs de 4 cifras en cuatro productos de nºs de 2 cifras, varios desplazamientos y una suma.

INTRODUCCIÓN A LA ALGORITMIA

- ♦ Los productos de números de 2 cifras pueden hacerse con la misma técnica. Por ejemplo, 09 y 12:

	multiplicar		desplazar	resultado
1)	0	1	2	0 . .
2)	0	2	1	0 .
3)	9	1	1	9 .
4)	9	2	0	18
				<hr/> 108

- Es un ejemplo de algoritmo que utiliza la técnica de "divide y vencerás".

La eficiencia de los algoritmos.

Métodos para evaluar la eficiencia.

- ▶ La unidad para medir la eficiencia de los algoritmos la vamos a encontrar a partir del Principio de Invarianza.

“dos implementaciones diferentes de un mismo algoritmo no diferirán en eficiencia mas que, a lo sumo, en una constante multiplicativa”

- ▶ Esto es válido, independientemente del ordenador usado, del lenguaje de programación empleado y de la habilidad del programador (supuesto que no modifica el algoritmo).
- ▶ El hecho de que un tiempo de ejecución dependa del input nos indica que ese tiempo debe definirse en función de dicho input (o del tamaño del input).
- ▶ Por tanto $T(n)$ es el tiempo de ejecución de un programa para un input de tamaño n , y también para el del algoritmo en el que se basa.



Notación Asintótica.

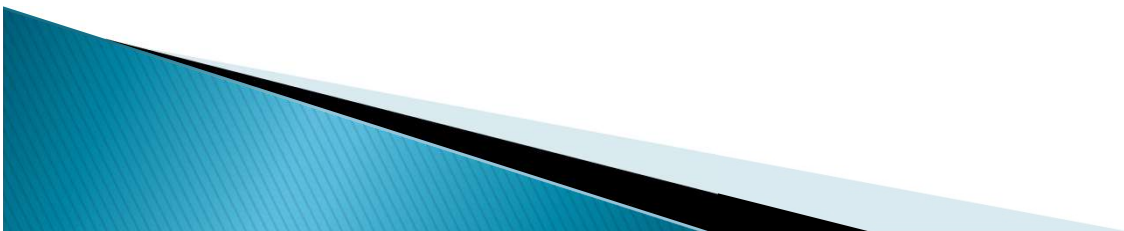
Un algoritmo con un tiempo de ejecución $T(n)$ se dice que es de orden $O(f(n))$ si existe una constante positiva c y un número entero n_0 tales que para todo $n \geq n_0$ entonces $T(n) \leq cf(n)$

Por ejemplo, si $T(0) = 1$, $T(1) = 4$ y $T(n) = (n+1)^2$. Entonces $T(n)$ es $O(n^2)$, puesto que si tomamos $n_0 = 1$ y $c = 4$, se verifica que
Para todo $n \geq 1$ entonces $(n+1)^2 \leq 4n^2$

Si un algoritmo tiene un tiempo de ejecución $O(f(n))$, a $f(n)$ le llamaremos Tasa de Crecimiento.

$O(f(n))$ (que se lee "el orden de $f(n)$ ") es el conjunto de todas las funciones $t(n)$ acotadas superiormente por un múltiplo real positivo de $f(n)$, dado que n es suficientemente grande (mayor que algún umbral n_0).

- ▶ Cuando $T(n)$ es $O(f(n))$, estamos dando es una cota superior para el tiempo de ejecución, que siempre referiremos al peor caso.



Análisis teórico del tiempo de ejecución.

Regla de la suma.

Supongamos, en primer lugar, que $T1(n)$ y $T2(n)$ son los tiempos de ejecución de dos segmentos de programa, $P1$ y $P2$, que $T1(n)$ es $O(f(n))$ y $T2(n)$ es $O(g(n))$. Entonces el tiempo de ejecución de $P1$ seguido de $P2$, es decir $T1(n) + T2(n)$, es $O(\max(f(n), g(n)))$.

Por ejemplo, tenemos un algoritmo constituido por 3 etapas, en el que cada una de ellas puede ser un fragmento arbitrario de algoritmo con bucles y ramas. Supongamos sus tiempos respectivos $O(n^2)$, $O(n^3)$ y $O(n \log n)$. Entonces

El tiempo de ejecución de las dos primeras etapas ejecutadas secuencialmente es $O(\max(n^2, n^3))$, es decir $O(n^3)$.

El tiempo de ejecución de las tres juntas es $O(\max(n^2, n^3, n \log n))$, es decir $O(n^3)$.

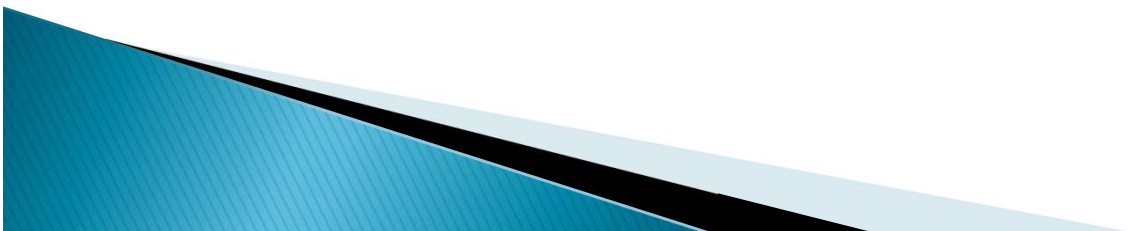


Análisis teórico del tiempo de ejecución.

Regla del producto.

Si $T1(n)$ y $T2(n)$ son los tiempos de ejecución de dos segmentos de programa, $P1$ y $P2$, $T1(n)$ es $O(f(n))$ y $T2(n)$ es $O(g(n))$, entonces $T1(n) T2(n)$ es $O(f(n) g(n))$.

De esta regla se deduce que $O(cf(n))$ es lo mismo que $O(f(n))$ si c es una constante positiva, así que por ejemplo $O(n^2/2)$ es lo mismo que $O(n^2)$.



Teoría del cálculo de la eficiencia.

Eficiencia Lineal. Operaciones Elementales

- La medición del **tiempo** de un algoritmo se realizará en función del número de **operaciones elementales** (Oes) que este realiza.
- Una **OE** es aquella cuyo consumo de tiempo de ejecución puede acotarse por una constante, es decir, que no depende en ninguna medida del tamaño ejemplar que se esté resolviendo.
- Para simplificar, se considera el coste de una **OE** o de una instrucción compuesta únicamente por Oes, como **coste 1**, es decir $OE \in O(1)$.



Teoría del cálculo de la eficiencia.

Eficiencia Lineal. Operaciones Elementales

► Las siguientes operaciones, si se aplican sobre datos de tipo básico son operaciones elementales:

- **Operaciones Aritméticas básicas:** +, -, *, /, DIV, MOD, ABS, ...;
- **Operaciones lógicas:** Y, O, NO, XOR;
- **Operaciones de Orden:** <, ≤, >, ≥, ≠;
- **Lectura o escritura;**
- **Asignación de valores;**
- **La instrucción de pseudocódigo Devolver.**

Teoría del cálculo de la eficiencia.

Eficiencia Lineal. Instrucción Condicional

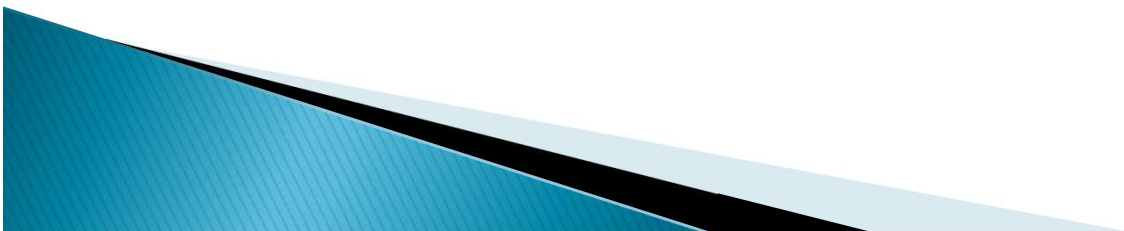
- ▶ Son instrucciones del tipo:

`Si cond entonces InstrV si no InstrF fsi`

- ▶ El coste de la instrucción completa es:

$$\text{coste}(\text{cond}) + \text{Max}\{\text{coste}(\text{InstrV}), \text{coste}(\text{InstrF})\}$$

- ▶ Siempre y cuando las instrucciones *InstrV* e *InstrF* puedan ocurrir con aproximadamente la misma frecuencia. En caso de que la condición *cond* tenga un valor lógico mucho más frecuente que el otro, en vez de tomar el máximo de las dos ramas se usará únicamente el del caso más habitual.



Teoría del cálculo de la eficiencia.

Instrucción por Bucle

- ▶ Para los bucles no determinados, como en la instrucción:

`Repetir InstrR Hasta COND fRepetir`

- ▶ El coste se obtiene mediante:

$$coste(InstrR) + coste(cond) + \sum_{cond=Falso} (coste(InstrR) + coste(cond))$$

Teoría del cálculo de la eficiencia.

Eficiencia Lineal. Instrucción por Bucle

- ▶ Para los bucles determinados, como en la instrucción:

Desde var \leftarrow vini hasta vfin Hacer InstrD fDesde

- ▶ El coste se obtiene mediante:

$$\text{Max}\{\text{coste}(\text{vini}), \text{coste}(\text{vfin})\} + \sum_{var=vini}^{vfin} (\text{coste}(\text{InstD}))$$

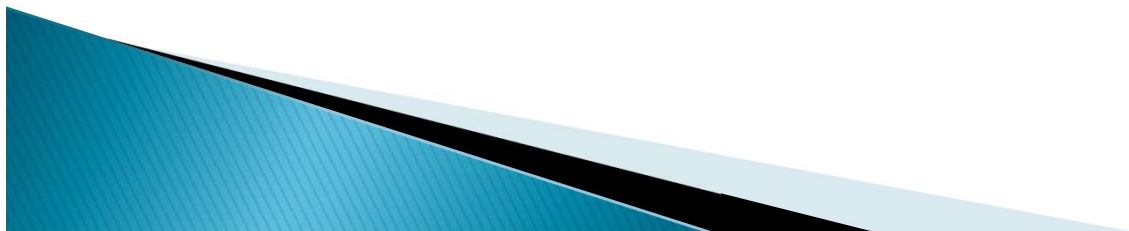
```
proc BucleFor1(E/S:entero1, entero2)
desde contador = 1 hasta entero2
    entero2 = entero2 / 2
    Escribir(entero1, entero2)
fdesde
fproc
```

```
proc BucleFor2(E/S:entero1, entero2)
desde contador = 1 hasta Potencia(entero2, 2)
    entero2 = entero2 / 2
    Escribir(entero1, entero2)
fdesde
fproc
```


Teoría del cálculo de la eficiencia.

Eficiencia Recursiva. Ecuación Característica

- ▶ Para obtener la eficiencia de un método recursivo se usará el método de la **Ecuación Característica**, que consiste en encontrar las raíces de la función matemática que representa el uso de recursos de un método recursivo.



Ejemplos.

Determinar la Función de eficiencia y el Orden de complejidad de los siguientes fragmentos de código:

```
for i := 1 to n do  
    for j := 1 to n do  
        A[i,j] := 0
```

```
for j := i+1 to n do  
    If A[j] < A[chico] then  
        chico := j
```

Ejemplos.

Determinar la Función de eficiencia y el Orden de complejidad del siguiente código:

```
If A[1,1] = 0 Then
    For i := 1 to n do
        For j := 1 to n do
            A[i,j] := 0
Else
    For i := 1 to n do
        A[i,i] := 1
```

Ejemplos.

Determinar la Función de eficiencia y el Orden de complejidad del siguiente código:

```
For i := 1 to n-1 do begin
  min := i
  For j := i+1 to n do
    If A[j] < A[min] Then
      min := j
  temp := A[min]
  A[min] := A[i]
  A[i] := temp
end
```

Ejemplos.

Determinar la Función de eficiencia y el Orden de complejidad del siguiente código:

```
Procedure Insert (T[1..n])  
  for i := 2 to n do  
    x := T[i];  
    j := i-1  
    while j > 0 and x < T[j] do  
      T[j+1] := T[j]  
      j := j-1  
    T[j+1] := x  
  end
```

Ejemplos.

Determinar la Función de eficiencia y el Orden de complejidad del siguiente código:

```
const dim = ...  
tipos tipovector = array[1..dim] de entero  
proc Ejercicio3(E n: entero; E/S v: tipovector)  
  var i: entero  
    si (n>0) y (n<=dim) entonces  
      Desde i ← 1 hasta (n-1) Hacer v[i] ← v[i+1] desde  
        Ejercicio(n-1, v)  
    fsi  
fproc
```