Algoritmos, parte II

Ejercicio 1.1.

```
for(i=0; i<n; i++)
    # alguna instrucción
```

R: El ciclo se ejecuta $n+1$ veces → la función de tiempo es orden $O(n)$

Ejercicio 1.2.

```
for(i=n; i>0; i--)
    # alguna instrucción
```

R: El ciclo se ejecuta $n+1$ veces → la función de tiempo es orden $O(n)$

Ejercicio 1.3.

```
for(i=0; i<n; i+=2)
    # alguna instrucción
```

i	i después	# ejec.
0	2	1
2	4	2
4	6 ÷2	3

i después: $n \xrightarrow{\div 2}$ # ejec.: $\lfloor n/2 \rfloor$

$$f(n) = \begin{cases} n/2 & n \text{ par} \\ (n+1)/2 & n \text{ impar} \end{cases} \longrightarrow O(n)$$

R: El ciclo se ejecuta $n/2$ veces → la función de tiempo es orden $O(n)$

Ejercicio 2.

```
for(i=0; i<n; i++)
    for(j=0; j<i; j++)
        #alguna instrucción
```

i	j
0	0
1	1
2	2
3	3
⋮	⋮
n	n

$$1+2+3+\ldots+n = \frac{n(n+1)}{2}$$

R: El ciclo se ejecuta $n(n+1)/2$ veces → la función de tiempo es orden $O(n^2)$

Ejercicio 3.

```
p=0                    i=3
for(i=1; p<=n; i++)
    for(j=0; j<i; j++)
        p=p+i          0<2
        p=0+1
```

i	j	p	
1	1	1	
2	2	3	+2
3	3	6	+3
4	4	10	+4

núm. de veces que ejecutamos

$a(n) = a(n-1) + n$

$a(n) = r^n \longrightarrow$ EDFH: $r^n = r^{n-1} \longrightarrow r^n(r-1) = 0$

$r = 1 \longrightarrow a_c = C_1$ Proponemos:

$a_p = (An+B)n = An^2 + Bn$

$An^2 + Bn = A(n-1)^2 + B(n-1) + n$

$An^2 + Bn = An^2 - 2An + A + Bn - B + n$

$2An = n \quad A - B = 0$

$A = \frac{1}{2} \quad A = B = \frac{1}{2} \quad \therefore a(n) = C_1 + \frac{1}{2}(n^2 + n)$

R: El código se ejecuta $n(n+1)/2 + c_1$ veces → la función de tiempo es orden $O(n^2)$

Ejercicio 4.

```
for(i=1; i<n; i=i*2)
    #alguna instruccion
```

i	i después
1	2
2	2^2
4	2^3
8	2^4
⋮	⋮

$2^K = n$

$K = \log_2 n$

R: El código se ejecuta $\log_2 n$ veces → la función de tiempo es orden $O(\log_2 n)$

Ejercicio 5.

```
for(i=n; i>=1; i=i/2)
    #alguna instruccion
```

i	i después
n	$n/2$
$n/2$	$n/2^2$
$n/4$	$n/2^3$
⋮	⋮

$\frac{n}{2^K} = 1 \longrightarrow n = 2^K \longrightarrow K = \log_2 n$

R: El código se ejecuta $\log_2 n$ veces → la función de tiempo es orden $O(\log_2 n)$

Ejercicio 6.

```
for(i=0; i*i<n; i++)
    #alguna instruccion
```

i	$i \times i$
0	0
1	1
2	4
⋮	
K	K^2

$K^2 = n \longrightarrow K = \sqrt{n}$

Ejercicio 7.

```
p=0
for(i=1; i<n; i=i*2)
    p++

for(j=1; j<p;j=j*2)
    #alguna instruccion
```

$\longrightarrow p = \log_2 n$

$\longrightarrow \log_2 p = \log_2(\log_2 n)$

R: El código se ejecuta $\log_2(\log_2 n)$ veces → la función de tiempo es orden $O(\log_2 \log_2 n)$

Ejercicio 8.

```
for(i=0; i<n; i++)
    for(j=1; j<n;j=j*2)
        #alguna instruccion
```

$\longrightarrow n+1$

$\longrightarrow \log_2 n$

R: El código se ejecuta $(n+1) \log_2 n$ veces → la función de tiempo es orden $O(n \log_2 n)$

Clases de funciones

Para la mayoría de los algoritmos, las funciones de tiempo pueden identificarse con alguna de las siguientes:

$O(1)$ — orden constante
$O(\log n)$ — orden logarítmico
$O(n)$ — orden lineal
$O(n \log n)$ — orden n logarítmo de n
$O(n^2)$ — orden cuadrático
$O(n^3)$ — orden cúbico
$O(2^n)$ — orden exponencial

Podemos «ordenar» las funciones de tiempo según su comportamiento asintótico:

$$O(1) < O(\log n) < O(n) < O(n \log n) < O(n^2) < O(n^3) < \ldots < O(2^n) < O(3^n)$$

⚠️ Este es el objeto de estudio de la complejidad de algoritmos.