Algoritmos

Definición. Un algoritmo es un conjunto de instrucciones o reglas definidas y no ambiguas, ordenadas y finitas que permiten resolver un problema (Wikipedia).

Algoritmos	Programas
Etapa de diseño	Etapa de implementación
Requieren conocimiento de la materia	Programadores
Escritos en cualquier lenguaje	Escritos en lenguaje de programación
Independientes de HW y SO	Dependientes de HW y SO

Características de un algoritmo

1. Entradas — 0 o más entradas
2. Salidas — al menos 1 salida
3. Definido — cada instrucción debe tener un significado claro, no ambiguo
4. Finito — debe terminar
5. Eficiencia — nada de instrucciones innecesarias

Descripción de un algoritmo

La descripción de un algoritmo se hace en tres niveles:
1. Descripción de alto nivel → explicación en manera verbal (lenguaje natural)
2. Descripción formal → se usa pseudocódigo para describir la secuencia de instrucciones
3. Implementación → se muestra un algoritmo en un lenguaje de programación específico

Ejemplo 1. Una descripción formal de un algoritmo.

```
swap(a,b){
    temp=a
    a=b
    b=temp
}
```

Análisis de algoritmos

El análisis de algoritmos es una parte importante de la teoría de complejidad computacional, que provee estimaciones teóricas de los recursos que necesita cualquier algoritmo. Estas estimaciones son muy útiles en la búsqueda de algoritmos eficientes.

1. Tiempo → función de tiempo $f(n)$
2. Espacio → función de espacio $s(n)$
3. Comunicaciones → cuántos datos se necesita transferir, por ejemplo, a un servicio *cloud*
4. Consumo de potencia
5. Uso de registros en CPU

🧑‍🏫 En este curso vamos a enfocarnos en el análisis de **tiempo** y **espacio** de un algoritmo.

Ejemplo 2. Análisis de un algoritmo. Encuentre la función de tiempo y espacio del siguiente algoritmo.

⚠️ Suponga que cada instrucción requiere 1 unidad de tiempo.

```
swap(a,b){
    temp=a
    a=b
    b=temp
}
```

Tiempo

1
1
1
+
$f(n) = 3$

Espacio

temp 1 word
a 1 "
b 1 "
+
$S(n) = 3$ words

⚠️ Las funciones de tiempo y espacio son **de orden constante**. Esto lo representamos como O(1).

Ejemplo 3. Encuentre la función de tiempo y espacio del siguiente algoritmo que calcula la suma de todos los elementos en un array.

```
sum(A, n){
    s=0
    for(i=0; i<n; i++)
        s = s+A[i]

    return s
}
```

Tiempo

	1
$i=0$	1
$i<n$	$n+1$
$i++$	$2n$
$s+A[i]$	$3n$

$f(n) = 6n+3$

i	
0	$< n$ ✓
1	$< n$ ✓
2	$< n$ ✓
⋮	
$n-1$	$< n$ ✓
n	$< n$ ✗

(n y $n+1$)

Espacio

S	1
i	1
n	1
A	n

$S(n) = n+3$

Las funciones de tiempo y espacio son de orden n. Esto lo escribimos O(n).

⚠️ De no haber escrito para `i++` $2n$ y para `s+A[i]` $3n$, la función de tiempo habría sido $f(n) = 3n + 3$, la cual **sigue siendo** de orden n. Según sea requerido por el analista, es posible *simplificar* el conteo de instrucciones ejecutadas.

Ejemplo 4. Encuentre la función de tiempo y espacio del siguiente algoritmo que calcula la suma de dos matrices cuadradas.

```
matrixAdd(A,B,n){
    for(i=0; i<n; i++)
        for(j=0; j<n; j++)
            C[i,j] = A[i,j]+B[i,j]

    return C
}
```

Tiempo

	$n+1$
	$n \times (n+1)$
	$n \times n$
	1

$f(n) = 2n^2+2n+2$

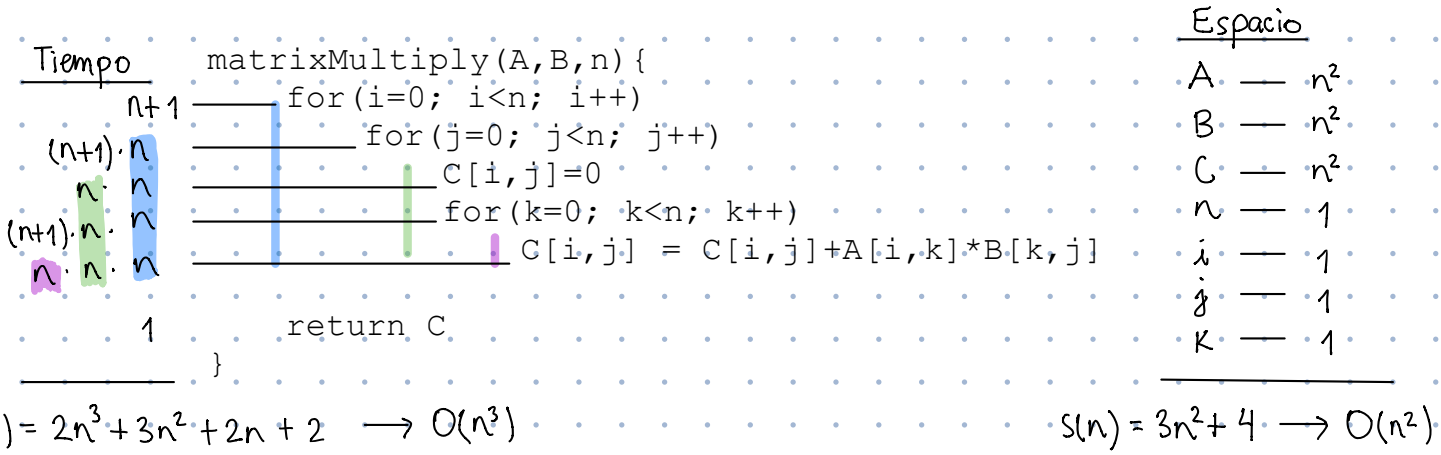
$O(n^2)$

Espacio

A	n^2
B	n^2
n	1
i	1
j	1
C	n^2

$S(n) = 3n^2+3 \longrightarrow O(n^2)$

Ejercicio 5. Encuentre la función de tiempo y espacio del siguiente algoritmo que calcula la multiplicación de dos matrices cuadradas.



⚠️ Cuando escribimos el orden de las funciones de tiempo y espacio estamos interesados únicamente en la **potencia** más alta de *n*, así que podemos «ignorar» los términos de menor grado.