Instituto Politécnico Nacional

Escuela Superior de Cómputo

Análisis de algoritmos

Tema 05: Análisis de algoritmos no recursivos

M. en C. Edgardo Adrián Franco Martínez
http://www.eafranco.com
edfrancom@ipn.mx
@edfrancom edgardoadrianfrancom

Contenido

- Análisis de algoritmos no recursivos
- La notación de Landau *O*
  - La notación *O*
  - Principio de invarianza del análisis asintótico
- Reglas prácticas del análisis de algoritmos
- Ordenes más comunes de los algoritmos
- Comportamiento de las funciones
- Análisis por bloques de algoritmos iterativos
  - Regla 1: Secuencia de instrucciones
  - Regla 2: Decisiones
  - Regla 3: Ciclos
  - Consideraciones especiales
- Logaritmos
  - Propiedades de los logaritmos
- Ejemplos
  - Ejemplo 01: Ordenamiento por intercambio
  - Ejemplo 02: Multiplicación de matrices de n x n
  - Ejemplo 03: Algoritmo de Horner
  - Ejemplo 04: Fibonacci (Iterativo)
  - Ejemplo 05: Búsqueda binaria

Análisis de algoritmos no recursivos

- El análisis de complejidad de los algoritmos no recursivos (**iterativos**) se realiza bajo los principios del **peor caso** y generalmente devolverá la cota superior ajustada del orden de este ($\boldsymbol{O}$).

- En principio se considera válido **cuando solo se desea obtener una cota para valores grandes de *n***, basándose en que se cumple el **principio de invarianza del análisis asintótico**.

- Para los algoritmos iterativos es únicamente necesario conocer los órdenes de complejidad $\boldsymbol{O}$ de las tres estructuras de control que todo algoritmo iterativo puede emplear.

La notación de Landau (***O***)

- Se dice que la función $f(n)$ "es de orden O $g(n)$" [**O(g(n))**], si existen constantes positivas c y n_0 tales que $|f(n)| \leq c\,|g(n)|$ cuando $n \geq n_0$

- Ejemplos:
  - $n+5$ es $O(n)$ pues $n+5 \leq 2n$ para toda $n \geq 5$
  - $(n+1)^2$ es $O(n^2)$ pues $(n+1)^2 \leq 4n^2$ para $n \geq 1$
  - $(n+1)^2$ **NO** es $O(n)$ pues para cualquier $c > 1$ no se cumple que $(n+1)^2 \leq c*n$

La notación O

- La notación O proporciona una cota superior para la tasa de crecimiento de una función.

- La siguiente tabla muestra la relación asintótica de la notación de orden O.

	$f(n)$ es $O(g(n))$	$g(n)$ es $O(f(n))$
$g(n)$ crece más	Sí	No
$f(n)$ crece más	No	Sí
Igual crecimiento	**Sí**	**Sí**

Principio de invarianza del análisis asintótico

- Cambios en el entorno HW o SW afectan a factores constantes pero no al orden de complejidad ***O(f(n))***

- El análisis de la eficiencia es **asintótico** → sólo es válido para tamaños de problema suficientemente grandes lo que hace valido el principio de invarianza.

"Dos implementaciones de un mismo algoritmo no diferirán más que en una constante multiplicativa"

Si $f_1(n)$ y $f_2(n)$ son los tiempos consumidos por dos implementaciones de un mismo algoritmo, se verifica que:

$$\exists\, c, d \in \mathbf{R},$$

$$f_1(n) \leq c \cdot f_2(n)$$

$$f_2(n) \leq d \cdot f_1(n)$$

Reglas prácticas del análisis de algoritmos

- **Operaciones primitivas**: tienen complejidad constante **O(1)**
- **Secuencia de instrucciones**: máximo de la complejidad de cada instrucción (regla de la suma).
- **Condiciones simples**: operaciones necesarias para evaluar la condición más las requeridas para ejecutar la consecuencia (peor caso).
- **Condiciones alternativas**: operaciones necesarias para evaluar la condición más las operaciones requeridas para ejecutar el mayor número de operaciones de las consecuencias (peor caso).

- **Bucle con iteraciones fijas**: multiplicar el número de iteraciones por la complejidad del cuerpo (regla del producto).

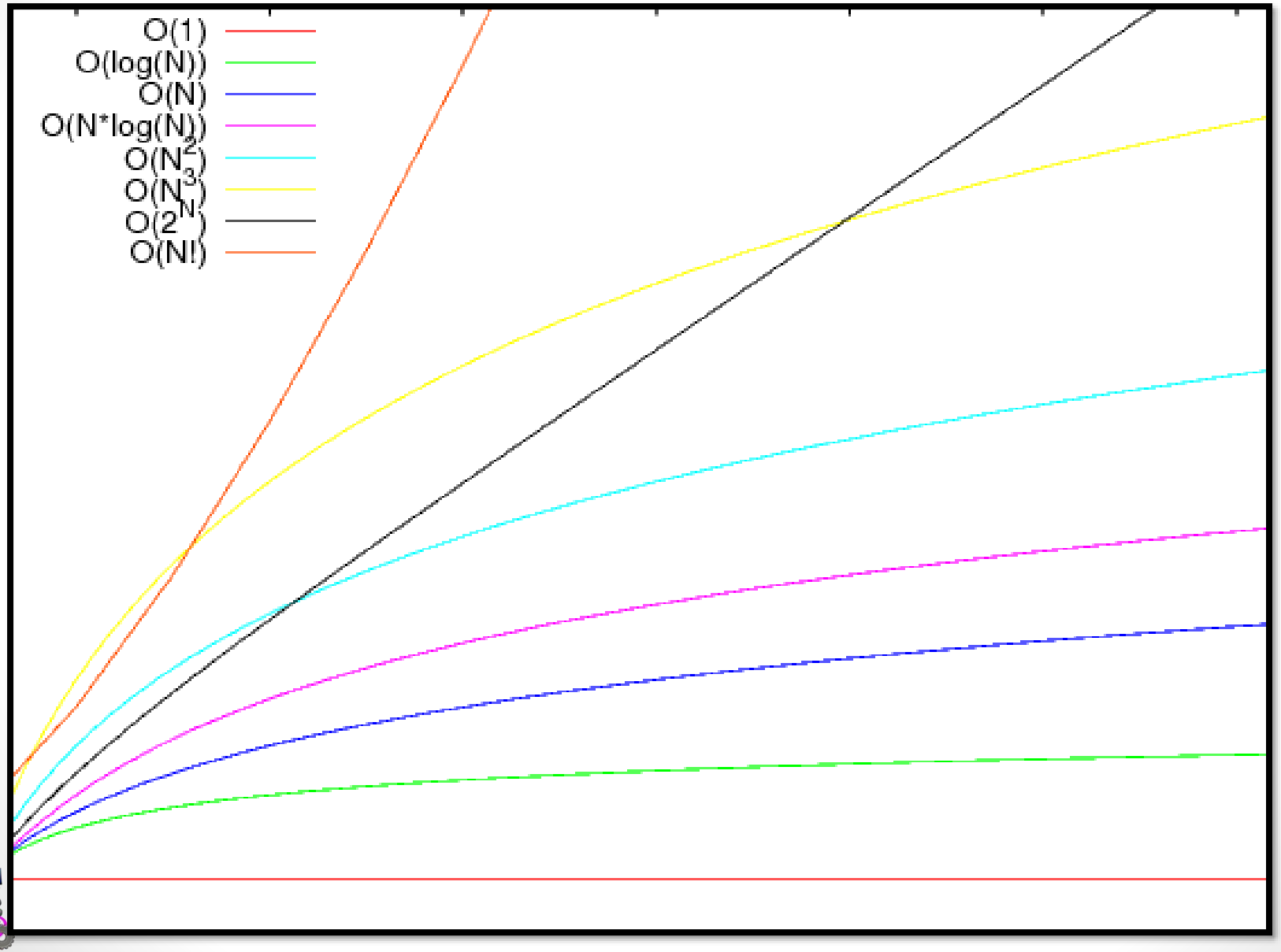
- **Bucle con iteraciones variables**: Igual pero poniéndose en el peor caso (ejecutar el mayor número de iteraciones posible).

- **Llamadas a subprogramas**, funciones o métodos: Operaciones de asignación de cada parámetro más las operaciones de ejecución del cuerpo, más el número de operaciones del retorno.

Ordenes más comunes de los algoritmos

- **O(1) Constante**
- **O(n) Lineal**
- **$O(n^2)$ Cuadrático**
- **$O(n^3)$ Cúbico**
- **$O(n^m)$ Polinomial**
- **O(log(n)) Logarítmico**
- **O(nlog(n)) nlog (n)**
- **$O(m^n)$ exponencial**
- **O(n!) factorial**

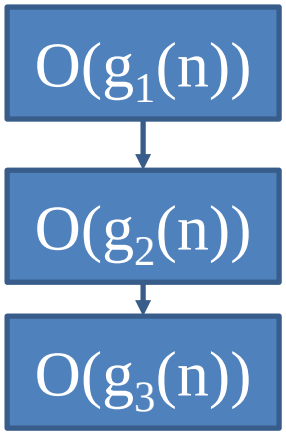
Comportamiento de las funciones



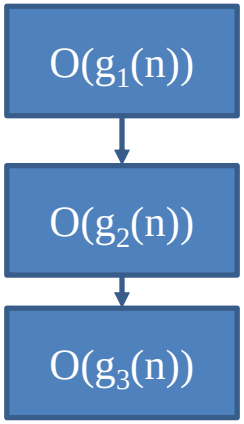
Análisis por bloques de algoritmos iterativos

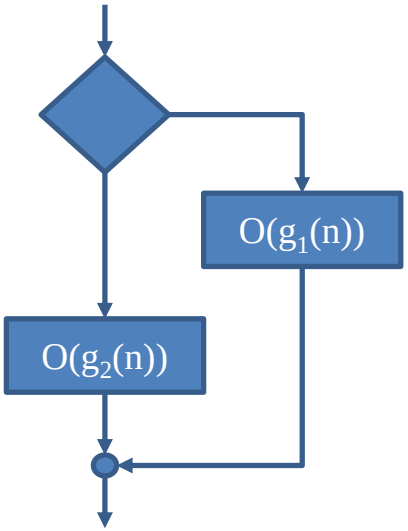
- El análisis de complejidad por bloques proporciona un importante medio para hacer un análisis más dinámico, que se basa principalmente en el conocimiento del orden de los bloques de instrucciones.

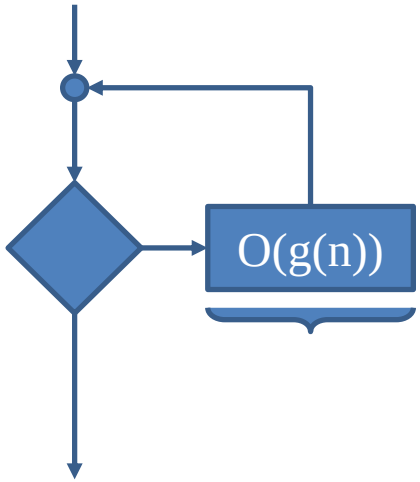
- Un bloque de instrucciones puede ser un algoritmo del cual conocemos su complejidad computacional. Para determinar el orden de un bloque, es necesario tener en mente el orden de las estructuras de control más usuales.



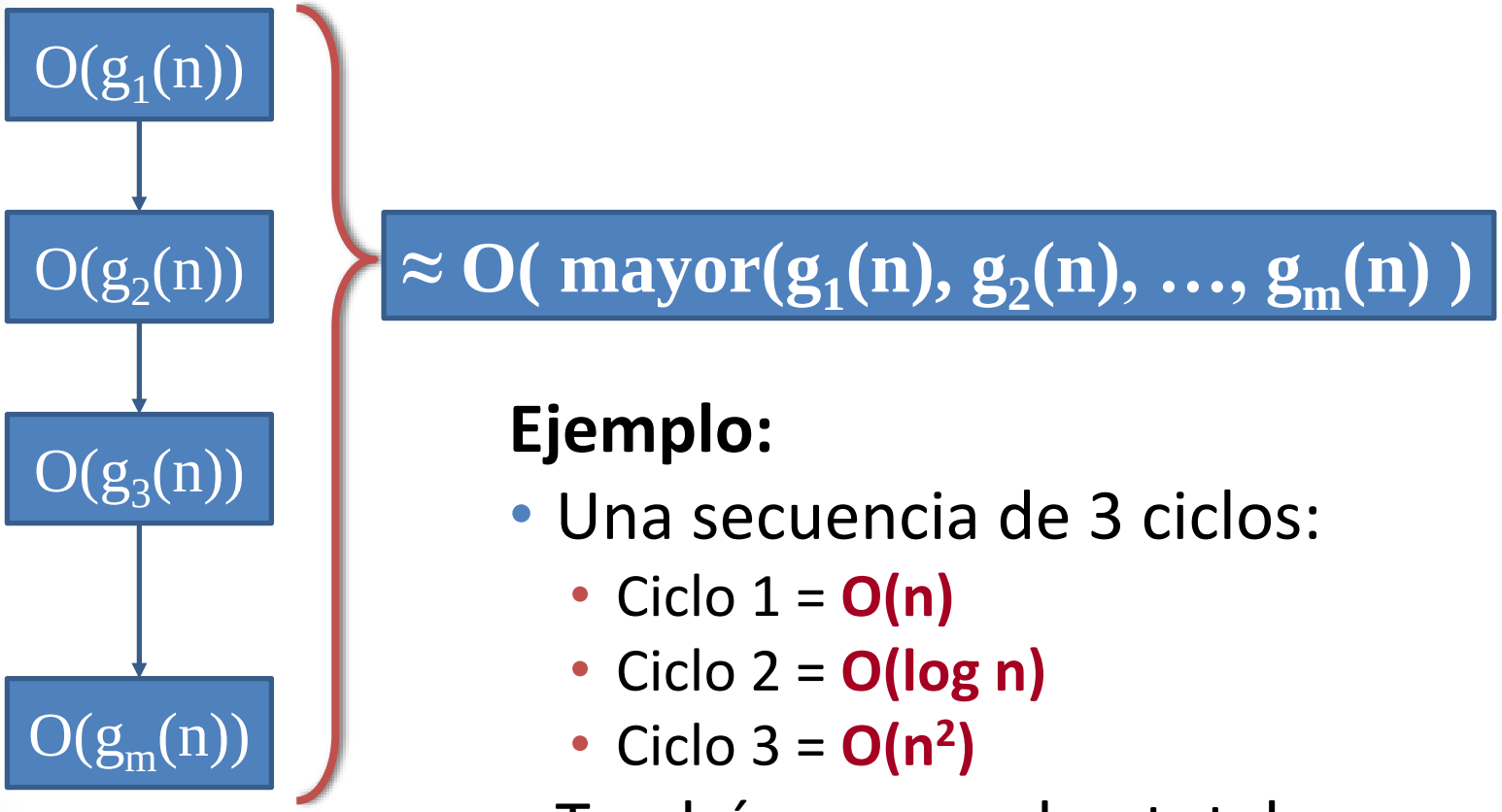
- El análisis por bloques implica el análisis de:
  1. **Secuencias de instrucciones**
  2. **Condicionales**
  3. **Ciclos**







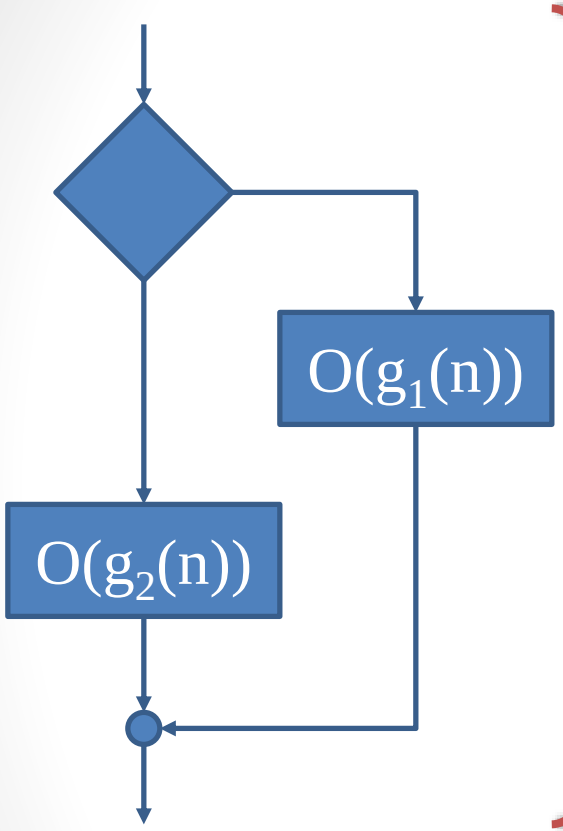
Regla 1: Secuencia de instrucciones



Ejemplo:

- Una secuencia de 3 ciclos:
  - Ciclo 1 = **O(n)**
  - Ciclo 2 = **O(log n)**
  - Ciclo 3 = **$O(n^2)$**
- Tendrá como orden total…
  - **$O(n^2)$**.

Regla 2: Decisiones

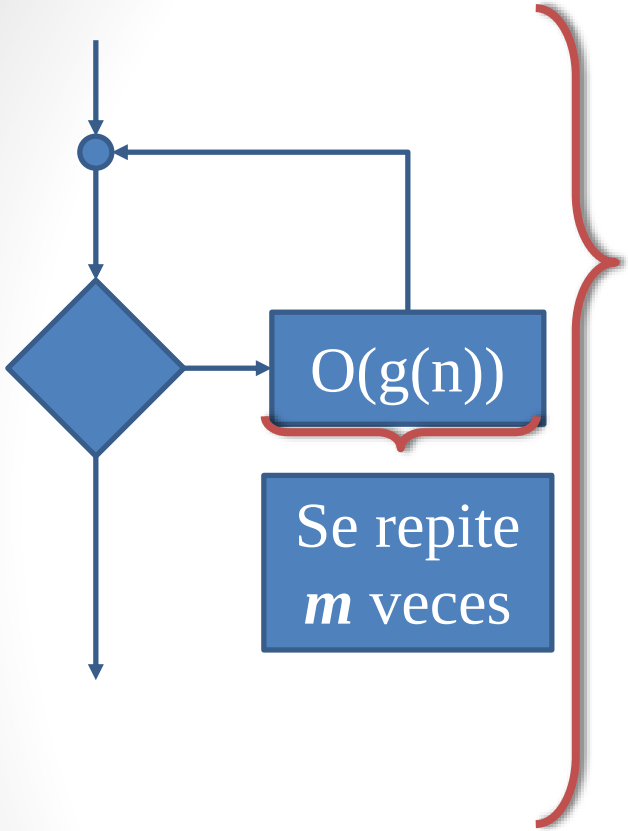


$$\approx O(\text{ mayor}(g_1(n), g_2(n)))$$

Ejemplo:

- Una decisión con:
  - Bloque verdadero= **O(n log n)**
  - Bloque falso= **O(log n)**
- Tendrá como orden total…
  - **O(n log n)**.

Regla 3: Ciclos



≈ O(m * g(n))

Ejemplo:

- Un ciclo cuya instrucción:
  - Tiene un **O(log n)**
  - Se repite **n/2** veces
- Tendrá como orden total…
  - **O(½ n log n) = O(n log n)**.

Consideraciones especiales

- En decisiones y ciclos anidados:
  - Analizar el código desde la instrucción más interna hacia el más externa.

- Tips para los ciclos:
  - ¿"Normalmente" cuál es el orden de la instrucción interna?
    - Si la variable de control se incrementa o decremento con un valor constante: **Orden LINEAL**.
    - Si la variable de control se multiplica o divide por un valor constante: **Orden LOGARÍTIMICO**.

Ejemplo 01: Ordenamiento por intercambio

```
for (int i=1; i<n; i++)
    for (int j=i+1; j<=n;j++)
        if (a[ j ] < a[ i ])
            intercambia(a[ i ], a[ j ]);
```

→ **O(1)** → **O(1)**

Regla 2: Decisiones = mayor de las 2 ramas

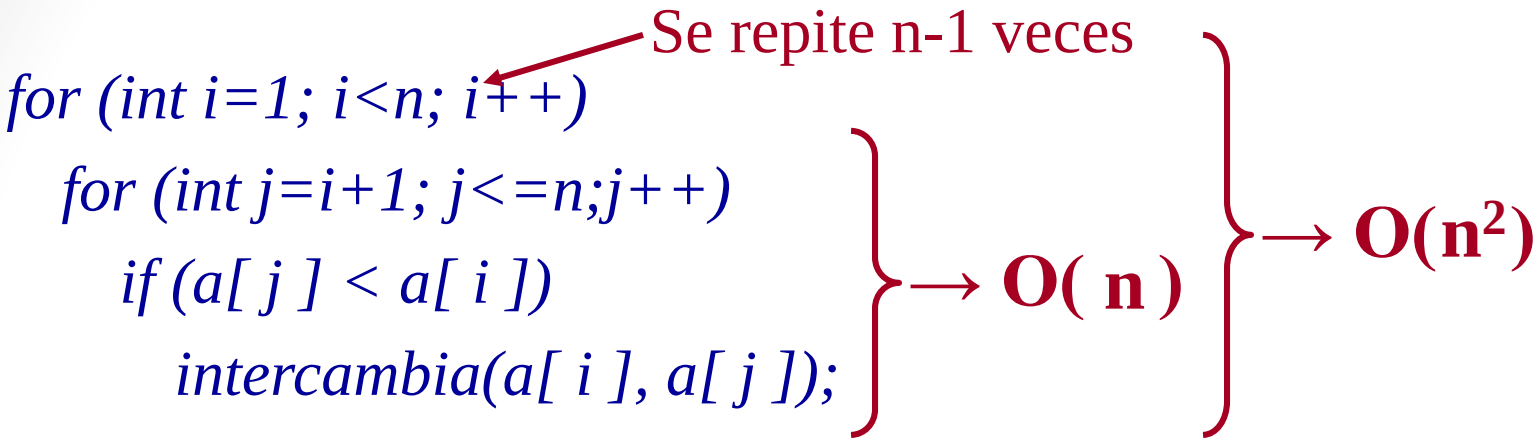
```
for (int i=1; i<n; i++)
    for (int j=i+1; j<=n;j++)
        if (a[ j ] < a[ i ])
            intercambia(a[ i ], a[ j ]);
```

Peor caso: se repite n-1 veces

→ O(1)

→ O(n)

*Regla 3: Ciclos = # veces * orden de la instrucción interna*



*Regla 3: Ciclos = # veces * orden de la instrucción interna*

Análisis espacial

- $fe(n) = espacio\ de(a)$

- $fe(n) = n\ es\ O(n)$

49	12	56	90	2	5	11	32	22	7	99	02	35	1

Ejemplo 02: Multiplicación de matrices de n x n

$$\begin{pmatrix} a_{11} & a_{12} \dots a_{1n} \\ a_{21} & a_{22} \dots a_{2n} \\ \dots & \dots \dots \dots \\ a_{n1} & a_{n2} \dots a_{nn} \end{pmatrix} \mathbf{X} \begin{pmatrix} b_{11} & b_{12} \dots b_{1n} \\ b_{21} & b_{22} \dots b_{2n} \\ \dots & \dots \dots \dots \\ b_{n1} & b_{n2} \dots b_{nn} \end{pmatrix} = \begin{pmatrix} c_{11} & c_{12} \dots c_{1n} \\ c_{21} & c_{22} \dots c_{2n} \\ \dots & \dots \dots \dots \\ c_{n1} & c_{n2} \dots c_{nn} \end{pmatrix}$$

$$c_{11} = a_{11}*b_{11}+a_{12}*b_{21}+\dots+a_{1n}*b_{n1}$$
$$c_{12} = a_{11}*b_{12}+a_{12}*b_{22}+\dots+a_{1n}*b_{n2}$$
$$\dots$$
$$c_{21} = a_{21}*b_{11}+a_{22}*b_{21}+\dots+a_{2n}*b_{n1}$$
$$\dots$$
$$c_{nn} = a_{n1}*b_{1n}+a_{n2}*b_{2n}+\dots+a_{nn}*b_{nn}$$

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

$O(n^3) \leftarrow$ *for i = 1 to n do*

$O(n^2) \leftarrow$ *for j = 1 to n do*

~~$O(1) \leftarrow$~~ *C[i,j] = 0;*

$O(n) \leftarrow$ *for k = 1 to n do*

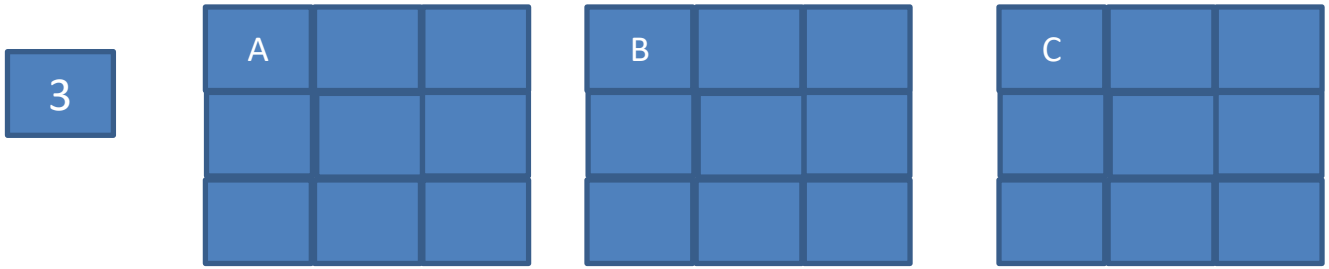
$O(1) \leftarrow$ *C[i,j] = C[i,j] + A[i,k]*B[k,j];*

22

Análisis espacial

- $fe(n) = espacio\ de(n) + \ espacio\ de\ (A, B, C)$

- $fe(n) \ = \ 1 \ + \ 3n^2 \ es \ O(n^2)$



Ejemplo 03: Algoritmo de Horner

- Sea A un vector de coeficientes y sea $P_n(z) = \sum_{i=0}^{n} A[i]\, z^i$, un polinomio de grado n; evaluado para un argumento real z:

- Encontrar la función complejidad, temporal y espacial, para el algoritmo que evalúa $P_n(z)$.

$$p(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \cdots + a_n x^n,$$

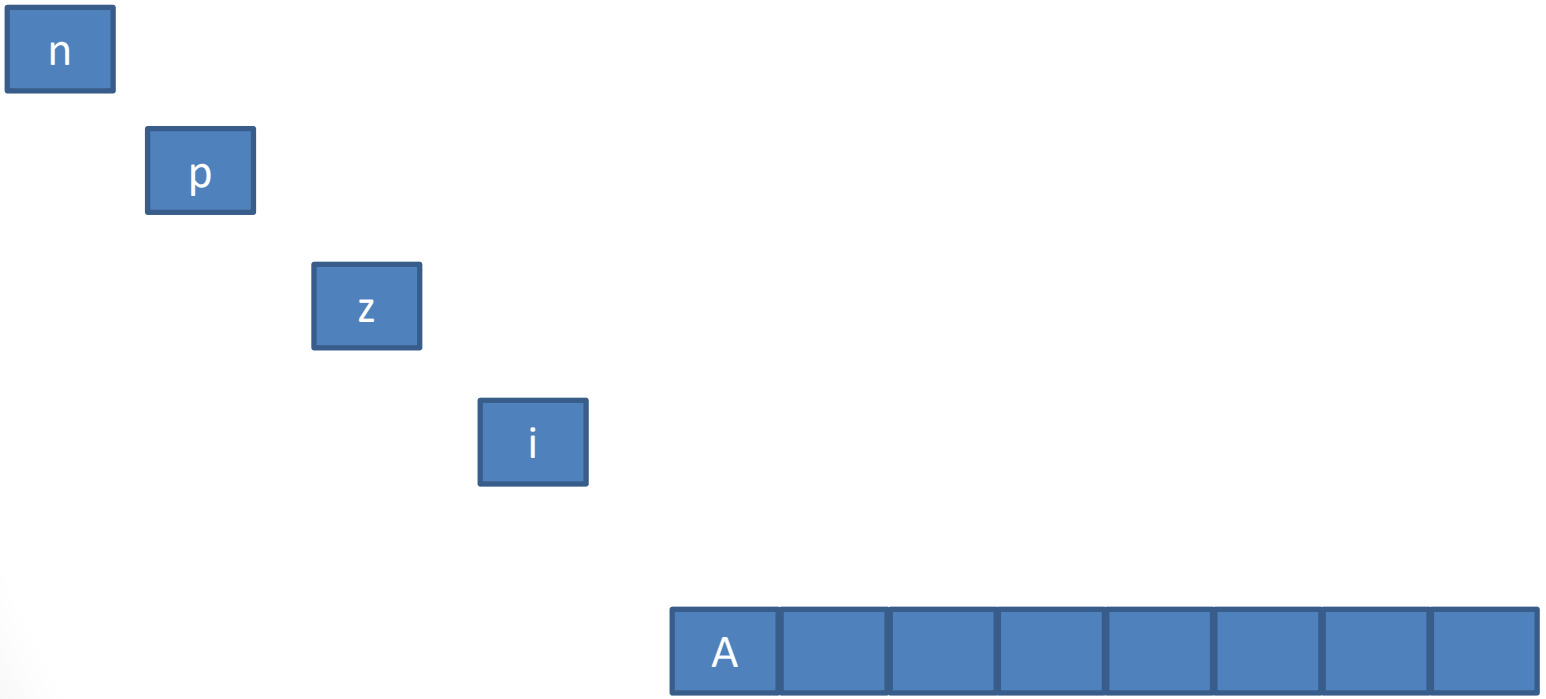
- **Algoritmo:** Método de Horner
  - **Tamaño del Problema**: ***n*** = grado del polinomio
  - **Operación básica:** La multiplicación * (Se realiza un número de veces del mismo orden al tamaño del problema)
  - **Caso:** El algoritmo hace el mismo número de operaciones en todos los casos.

```
proc Horner(n,z,A)
{
      polinomio=0;
      for(i=0;i<=n;i++)
      {
            polinomio=polinomio*z + A[n-i];*
      }
}
```

```
polinomio=0;
for(i=0; i<=n; i++)                          ← O( n )
        polinomio=polinomio*z + A[n-i];      ← O( 1 )
```

Análisis espacial

- $fe(n) = espacio\ de\ (n) + espacio\ de\ (polinomio) + espacio\ de\ (z) + espacio\ de\ (i) + espacio\ de\ (A)$
- $fe(n) = 1 + 1 + 1 + 1 + n = 4 + n\ es\ O(n)$



27

Ejemplo 04: Fibonacci (Iterativo)

anterior = 1;	**--> 1 Asignación**
actual = 1;	**--> 1 Asignación**
while (n>2){	**--> n-2 + 1 Condicionales**
aux = anterior + actual;	**--> n-2 Asignaciones**
anterior = actual;	**--> n-2 Asignaciones**
actual = aux;	**--> n-2 Asignaciones**
n = n - 1;	**--> n-2 Asignaciones**
}	**--> n-2+1 Saltos implicitos**
return (actual);	**--> 1 Asignación**

T(n) = 6n-7=O(n)

28

Ejemplo 05: Búsqueda binaria

```
BusquedaBinaria(A,n,dato)
        inferior=0;                          ← O( 1 )
        superior=n-1;
        while(inferior<=superior)
                centro=(superior+inferior)/2;
                if(A[centro]==dato)                    ← O(log 2n)
                                return centro;
                else if(dato < A[centro] )
                                superior=centro-1;
                else
                                inferior=centro+1;
        return -1;
```

29