Algoritmos y Estructuras de Datos II
Clase 3

Carreras:

Licenciatura en Informática

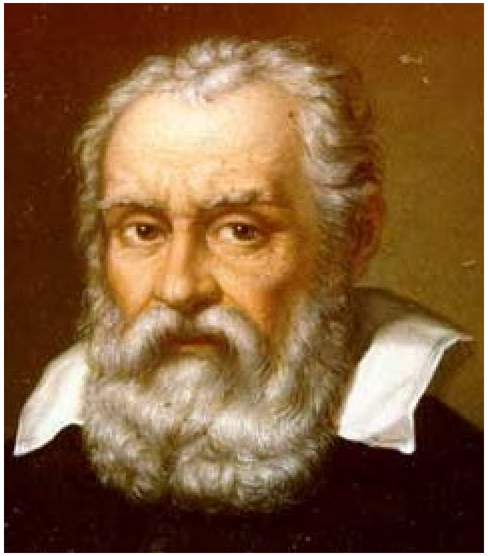
Ingeniería en Informática

2024

Unidad II

"La Matemática es el Alfabeto con el cual Dios ha Escrito el Universo"

Galileo Galilei



Galileo Galilei (Pisa, 15 de febrero de 1564- Florencia, 8 de enero de 1642), fue astrónomo, filósofo, matemático y físico. Se lo considera el «padre de la ciencia».

Análisis de Algoritmos

Los algoritmos se pueden escribir, entender y estudiar de manera independiente del lenguaje y de la maquina en que se vayan a implementar.

Usando el modelo de maquina RAM de computación , se pueden contar cuantos pasos hace un algoritmo para una instancia particular de ejecución.

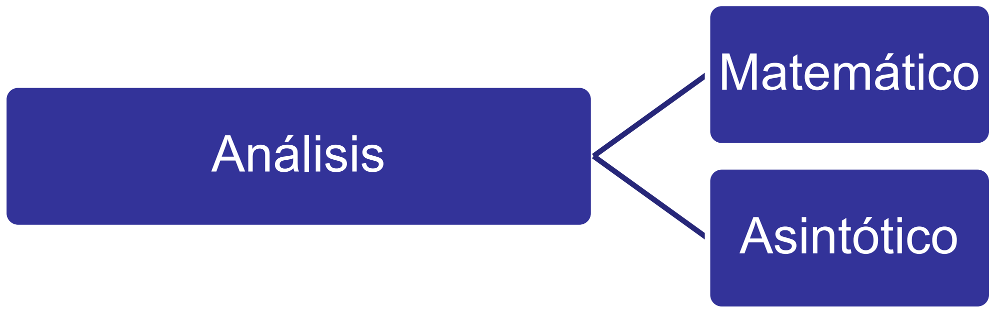
Sin embargo para saber si el algoritmo es eficiente o no hay que probarlo con ***todas*** las instancias posibles.

Análisis de Algoritmos

- Se denota con T(n) el tiempo de ejecución de un **algoritmo** para una entrada de tamaño n.

- T(n) es el número de instrucciones ejecutadas por el algoritmo en una computadora ideal, por ejemplo con el modelo RAM.

- Para analizar el tiempo de ejecución de un algoritmo, se pueden hacer distintos tipos de *análisis de algoritmos*:



Análisis Matemático de Algoritmos

El *análisis matemático de algoritmos* es independiente de la implementación, pero tiene varios inconvenientes:

- La imposibilidad de determinar, para muchos problemas y cada una de las posibles entradas, la cantidad exacta de instrucciones ejecutadas.
- La dificultad para determinar T(n) *exactamente*.

Análisis Asintótico de Algoritmos

El *análisis asintótico de algoritmos* es una técnica derivada del análisis matemático de algoritmos basada en dos conceptos fundamentales:

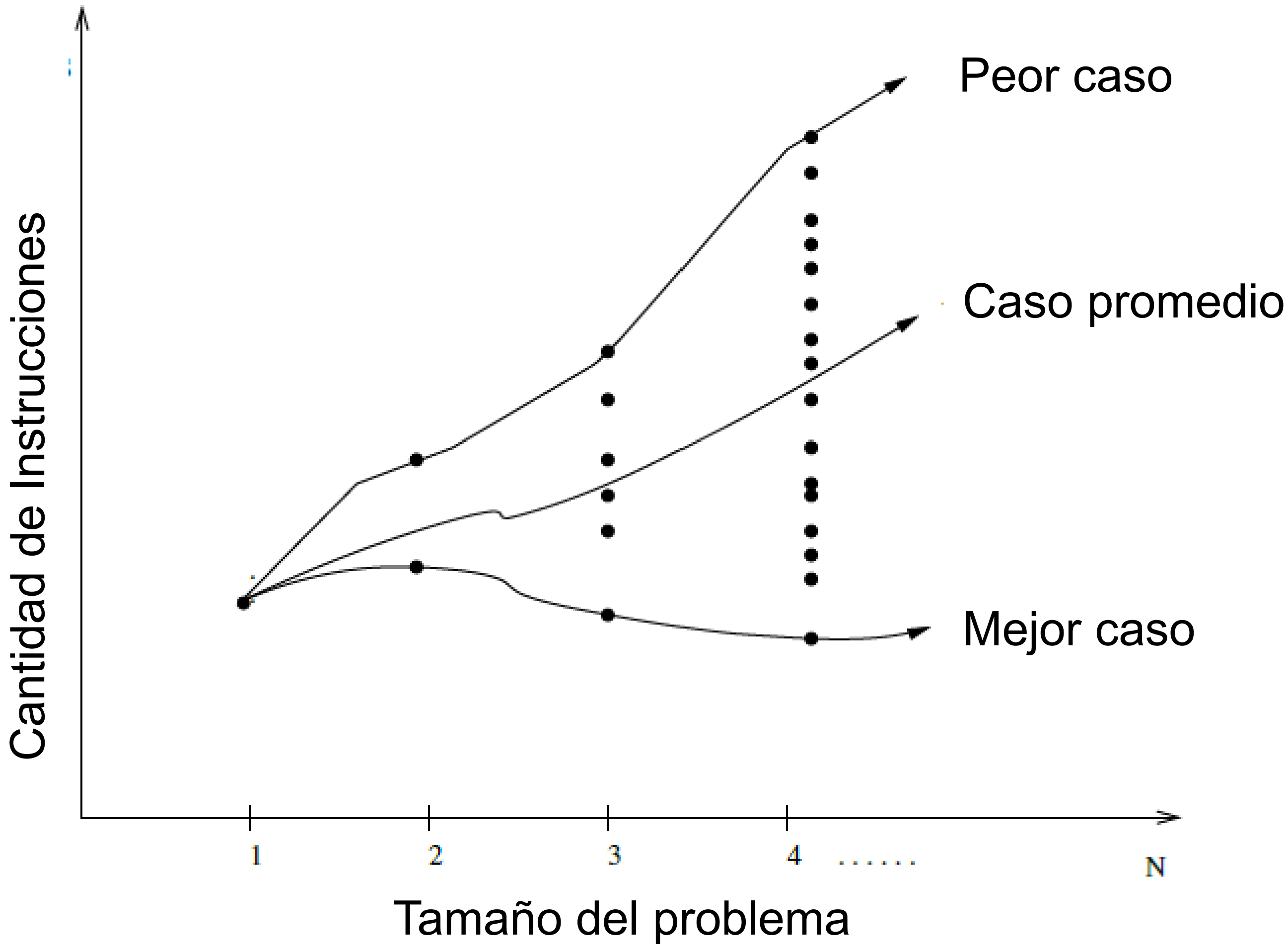
- la caracterización de datos de entrada
- la complejidad asintótica.

- Peor caso: Los casos de datos de entrada que maximizan la cantidad instrucciones ejecutadas por un algoritmo.

- Mejor caso: Los casos de datos de entrada que minimizan la cantidad de instrucciones ejecutadas por un algoritmo.

- Caso promedio: El valor medio de la cantidad de instrucciones ejecutadas por un algoritmo. Se debe tener en cuenta la distribución probabilística de los datos de entrada que se manejan.

Análisis Asintótico de Algoritmos



Análisis Asintótico de Algoritmos

Complejidad del peor caso de un algoritmo: es la función definida como el máximo numero de pasos ejecutados para cualquier instancia de tamaño n.

Complejidad del mejor caso: es la función definida como el mínimo numero de pasos ejecutados para cualquier instancia de tamaño n.

Complejidad del caso promedio: es la función definida como el promedio del numero de pasos ejecutados para cualquier instancia de tamaño n.

Análisis Asintótico de Algoritmos

Sea D el *conjunto de datos* de entrada de tamaño n para un algoritmo, y sea $I \in D$ un elemento cualquiera.

Sea $T(I)$ la *cantidad instrucciones* ejecutadas por el algoritmo para procesar la entrada I.

Se definen entonces las siguientes funciones:

- Complejidad del peor caso: $F(n) = \max \{T(I) : I \in D, |I| = n\}$
- Complejidad del mejor caso: $G(n) = \min \{T(I) : I \in D, |I| = n\}$
- Complejidad del caso promedio: $P(n) = \Sigma_{I \in D,} [p(I) * T(I)]$, donde $p(I)$ es la probabilidad de que ocurra la entrada I de tamaño n.

Análisis Asintótico de Algoritmos

Consejo: Considerar siempre el peor caso, ya que representa una cota superior para la cantidad de trabajo realizado por un algoritmo

Idea fundamental: tratar de encontrar una función F(n), fácil de calcular y conocida, que acote asintóticamente el orden de crecimiento de la función T(n)

Objetivo: Estudiar la eficiencia asintótica de algoritmos: Cómo se incrementa la cantidad de trabajo realizado por un algoritmo a medida que se incrementa el tamaño de la entrada (con valores "*suficientemente grandes*")

Herramienta: Para realizar este análisis se necesitan herramientas especiales: *las notaciones asintóticas*

Notación asintótica O grande

Introducida por el matemático alemán Paul Gustav Heinrich Bachmann en su libro *Analytische Zahlentheorie* (1892) y popularizada por otro matemático alemán Edmund Landau.

Características:

- Se usa para funciones que dependen de un entero positivo ***n.***
- *Simplifica* el análisis de los algoritmos ignorando los detalles que no impacten en su comparación con otros algoritmos.
- Es una notación muy conveniente para trabajar con aproximaciones: permite *aproximar un valor* de una cantidad en lugar de su valor exacto.
- Describe un concepto que ocurre frecuentemente y *suprime los detalles* de la información que es irrelevante.
- Se puede *manipular algebraicamente* conociendo sus propiedades.
- Permite *definir un límite superior* al crecimiento de una función.
- Establece un “*orden*” entre funciones, que permite compararlas.
- *Clasifica* las funciones de tiempo de los algoritmos para que puedan ser comparadas.

Notación O grande

Definición:

Sea R* reales no negativos

Sea f: N → R* una función arbitraria

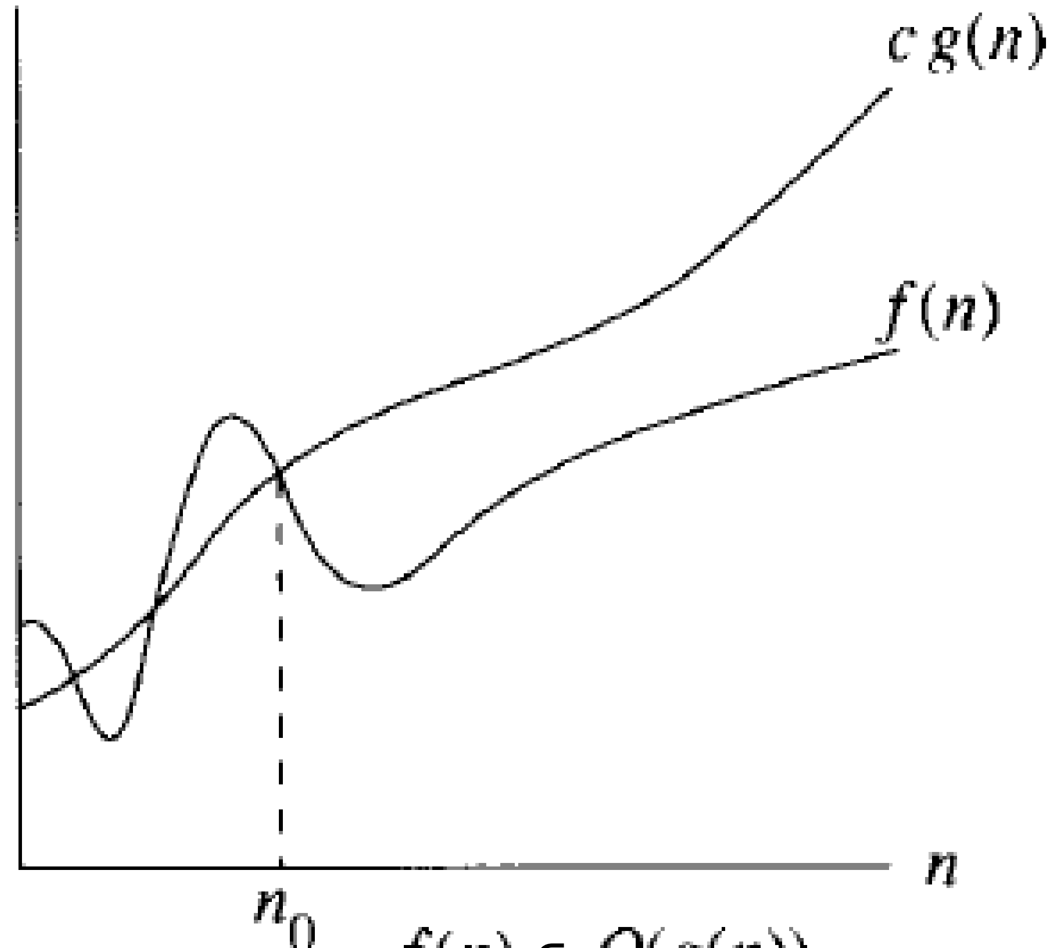
Se define O grande de f(n) como el conjunto de todas las aplicaciones:

$$\mathbf{O(f(n)) = \{\ t : N \rightarrow R^* \ / \ (\exists\ c \in R^+)\ (\exists\ n_0 \in N):}$$
$$\mathbf{0 \leq t(n) \leq c\ .\ f(n) \quad \forall\ n \geq n_0\ \}}$$

Si una función **g pertenece a O(f(n))** se dice que:
***g es de orden f(n)*.**

Notación O grande

En un gráfico:



$f(n) \in O(g(n))$

Propiedades de la notación O grande

- $f(n) \in O(f(n))$
- $f(n) \in O(g\,(n)) \Rightarrow O(f(n)) \subset O(g(n))$
- $f(n) \in O(g(n)) \Rightarrow c.f(n) \in O(g(n)) \quad \forall\ c \in R+$
- $f1(n) \in O(g(n))$ y $f2(n) \in O(g(n)) \Rightarrow f1(n) + f2(n) \in O(g(n))$
- $f(n) \in O(g(n))$ y $g(n) \in O(f(n)) \Leftrightarrow O(f(n)) = O(g(n))$
- $f(n) \in O(g1(n))$ y $f(n) \in O(g2(n)) \Rightarrow f(n) \in O(\min\,(g1(n),g2(n))\,)$

- **Transitiva:**
  $f(n) \in O(g(n))$ y $g(n) \in O(h(n)) \Rightarrow f(n) \in O(h(n))$

- **Regla de la suma**:
  $f1(n) \in O(g1(n))$ y $f2(n) \in O(g2(n)) \Rightarrow f1(n)+f2(n) \in O(\max\,(g1(n),g2(n)))$

- **Regla del producto**:
  $f1(n) \in O(g1(n))$ y $g2(n) \in O(g2(n)) \Rightarrow f1(n).f2(n) \in O(\,g1(n).g2(n)\,)$

Propiedades de la notación O grande

Propiedad:

$f1(n) \in O(g1(n))$ y $f2(n) \in O(g2(n)) \Rightarrow f1(n)+f2(n) \in O(g1(n)+g2(n))$

Demostración:

Vale que:

Para todo $n \geq n1$, $f1(n) \leq c1\ g1(n)$

Para todo $n \geq n2$, $f2(n) \leq c2\ g2(n)$

Sea $n0 = \max\{n1,n2\}$, entonces:

Para todo $n \geq n0$, $f1(n)+f2(n) \leq c1\ g1(n) + c2\ g2(n)$

Sea $c0 = \max\{c1,c2\}$, entonces:

Para todo $n \geq n0$, $f1(n)+f2(n) \leq c0\ (g1(n) + g2(n))$

Propiedades de la notación O grande

Propiedad, regla de la suma:
$f1(n) \in O(g1(n))$ y $f2(n) \in O(g2(n)) \Rightarrow f1(n)+f2(n) \in O(\max(g1(n), g2(n)))$

Demostración:
Vale que:

Para todo $n \geq n1$, $f1(n) \leq c1\ g1(n)$
Para todo $n \geq n2$, $f2(n) \leq c2\ g2(n)$

Sea $n0 = \max\{n1, n2\}$ y sea $c0 = c1+c2$, entonces:
Para todo $n \geq n0$,

$$
\begin{aligned}
f1(n) + f2(n) &\leq c1\ g1(n) + c2\ g2(n) \\
&\leq (c1+c2)(\max\{g1(n), g2(n)\}) \\
&= c0\ (\max\{g1(n), g2(n)\})
\end{aligned}
$$

Notación O grande

Notación O grande aplicada a los algoritmos:

- Sea T(n) el tiempo de ejecución de un algoritmo para una entrada de tamaño n.

- Se dice que ***T(n) es de orden f(n)*** o que ***T(n) es O(f(n))*** si existen dos constantes: c(real) y n_0(natural) tales que:

  $$T(n) \leq c.\ f(n) \text{ cuando } n \geq n_0$$

- Cuando se dice que T(n) es O(f(n)) se está garantizando que la función f(n) es una cota superior para el crecimiento de T(n).

Principio de Invariancia

- Dadas dos implementaciones de un mismo algoritmo: $I1$ e $I2$ que emplean $T1(n)$ y $T2(n)$ segundos respectivamente, el *Principio de Invariancia* afirma que existe una constante real $c>0$ y un número natural $n0$, tales que para todo $n \geq n0$ se verifica que:

$$T1(n) \leq c.T2(n)$$

- Entonces: el tiempo de ejecución de dos implementaciones distintas de un algoritmo dado no va a diferir más que en una constante multiplicativa.

Notación asintótica

- Con esto se puede definir que un algoritmo necesita un tiempo del orden de $T(n)$ si existen una constante real $c>0$ y una implementación I del algoritmo que necesita menos que $cT(n)$, para todo n tamaño de la entrada.

- En notación asintótica se expresa entonces que el algoritmo pertenece a $O(T(n))$

$$\mathbf{T(n) \in O(T(n))}$$

Notación O grande
Ejemplos:

Verdadero o Falso

- $4n+1 \in O(n)$?
- $2n \in O(1)$?
- $n(n+1)^2 \in O(n^3)$?
- $2^{n+1} \in O(2^n)$?
- $2^{2n} \in O(2^n)$?
- $\log_2 8^n \in O(n)$?
- $4^{\log_2 n} \in O(n^2)$?

$4n+1 \in O(n)$?

$4n+1 \leq c.n$ para $n \geq n_0$

Dividir por n>0 ambos miembros de la desigualdad:

$(4n+1)/n \leq c.n/n$ para $n \geq n_0$

$4+1/n \leq c$ para $n \geq n_0$

Vale para: $c=5$, $n_0=1$

Vale para: $c=4.5$, $n_0=2$ o…

Etc.

➔ $4n+1 \in O(n)$

$2n \in O(1)$?

$2n \leq c.1$ para $n \geq n_0$

$2n \leq c$ para $n \geq n_0$

2n crece con n $\Rightarrow$ no se puede acotar para ninguna constante c cuando n crece.

➔ $2n \notin O(1)$

$n(n+1)^2 \in O(n^3)$?

$n(n+1)^2 \leq c.n^3$ para $n \geq n_0$

$n^3 + 2.n^2 + n \leq c.n^3$ para $n \geq n_0$

Dividir por $n^3 > 0$ ambos miembros de la desigualdad:

$1 + 2/n + 1/n^2 \leq c$ para $n \geq n_0$

De modo que: si $c=4$, $n_0=1$

$n(n+1)^2 \leq 4n^3$ para $n \geq 1$

➔ $n(n+1)^2 \in O(n^3)$

Propiedades de exponeciales

- Para todo a>0, m y n valen las siguientes identidades:

$$
\begin{aligned}
a^0 &= 1, \\
a^1 &= a, \\
a^{-1} &= 1/a, \\
(a^m)^n &= a^{mn}, \\
(a^m)^n &= (a^n)^m, \\
a^m\, a^n &= a^{m+n}.
\end{aligned}
$$

$2^{n+1} \in O(2^n)$?

$2^{n+1} \leq c.2^n$ para $n \geq n_0$

$2.2^n \leq c.2^n$ para $n \geq n_0$

Dividir por $2^n > 0$ ambos miembros de la desigualdad:

$2 \leq c$ para $n \geq n_0$

De modo que si: si $c=2$, $n_0=1$

$2^{n+1} \leq 2.2^n$ para $n \geq 1$

➔ $2^{n+1} \in O(2^n)$

$2^{2n} \in O(4^n)$

$2^{2n} \leq c.4^n$ para $n \geq n_0$

$2^{2n} \leq c.(2^2)^n = c.\ 2^{2n}$ para $n \geq n_0$

Dividir por $2^{2n} > 0$ ambos miembros de la desigualdad:

$1 \leq c$ para $n \geq n_0$

De modo que si: si $c=1$, $n_0=1$

$2^{2n} \leq 4^n$ para $n \geq 1$

➔ $2^{2n} \in O(4^n)$

Propiedades de logaritmos

- Para todo a>0, b>0, c>0 y n>0 valen las identidades:

$$a = b^{log_b a}$$

$$log_c(ab) = log_c a + log_c b$$

$$log_c \frac{a}{b} = log_c a - log_c b$$

$$log_b a^n = n\, log_b a$$

$$log_b a = \frac{log_c a}{log_c b}$$

$$log_b \frac{1}{a} = -log_b a$$

$$log_b a = \frac{1}{log_a b}$$

$$a^{log_b n} = n^{log_b a}$$

$\log_2 8^n \in O(n)$?

$\log_2 8^n \leq c\, n$ para $n \geq n_0$

Usando: $\boxed{\log_b a^n = n \log_b a}$

$n \log_2 8 \leq c\, n$ para $n \geq n_0$

$\log_2 8 \leq c$ para $n \geq n_0$

$3 \leq c$ $n_0=1$

De modo que si: $c=3$, $n_0=1$

$\log_2 8^n \leq 3\, n$ para $n \geq 1$

➔ $\log_2 8^n \in O(n)$

$4^{\log_2 n} \in O(n^2)$?

$4^{\log_2 n} \leq c.n^2$ para $n \geq n_0$

Usando: $\boxed{a^{log_b n} = n^{log_b a}}$

$n^{\log_2 4} \leq c.n^2$ para $n \geq n_0$

$n^2 \leq c.n^2$ para $n \geq n_0$

De modo que si: $c=1$, $n_0=1$

$n^2 \leq 1.n^2$ para $n \geq 1$

➔ $4^{\log_2 n} \in O(n^2)$

Notación O grande
Ejercicios:

Demostrar que:

- $3n^2+100\,n+4 \notin O(1)$
- $3n^2+100\,n+4 \notin O(n)$
- $3n^2+100\,n+4 \in O(n^2)$
- $3n^2+100\,n+4 \in O(n^3)$

Complejidad Computacional

- Complejidad en el ***peor caso*** (*es siempre significativo el peor caso?)*
- Complejidad en el ***caso promedio*** (*porqué no se usa este enfoque siempre?)*
- Complejidad en el ***mejor caso*** (*es significativo el mejor caso?)*

Ejemplo del casino.

Algoritmo de clasificación

Algoritmo InserciónDirecta(A,n)

Entrada: A, arreglo de (1.. MAX) elementos de tipoitem
n: nro. entero de datos a ordenar en el arreglo
Salida: A, arreglo ordenado.
Auxiliar: i,j:entero; proximo: tipoitem


```
P1. PARA  i  = 2 ,n  HACER
          proximo ← A(i)
          j ←  i
P2.       MIENTRAS ( j > 1   AND   proximo < A(j-1) ) HACER
                    A(j) ←  A(j-1)
                    j ← j-1
          A(j) ←  proximo
P3. FIN
```

Ejemplo: inserción directa

Arreglo A : **5 2 4 6 1 3**, n=6

		próximo				
i=2	5	2	4	6	1	3
			próximo			
i=3	2	5	4	6	1	3
				próximo		
i=4	2	4	5	6	1	3
					próximo	
i=5	2	4	5	6	1	3
						próximo
i=6	1	2	4	5	6	3
A ordenado:	1	2	3	4	5	6

Ejemplo: inserción directa

Análisis del número de movimientos:

```
P1. PARA  i  = 2 ,n  HACER
          proximo ← A(i)
          j ←  i
P2.       MIENTRAS ( j > 1   AND   proximo < A(j-1) ) HACER
                   A(j) ←  A(j-1)
                   j ← j-1
          A(j) ←  proximo
P3. FIN
```

- En este algoritmo hay dos iteraciones, una exterior fija (P1) y una interior condicional (P2).
- Para el ciclo exterior se realizan siempre 2 movimientos

  proximo ← A(i)

  A(j) ← proximo

Ejemplo: inserción directa

Análisis del número de movimientos:

```
P1. PARA  i  = 2 ,n  HACER
          proximo ← A(i)
          j ← i
P2.       MIENTRAS ( j > 1   AND   proximo < A(j-1) ) HACER
                   A(j) ←  A(j-1)
                   j ← j-1
          A(j) ←  proximo
P3. FIN
```

- El número de movimientos del ciclo interior (que puede o no ser ejecutado) se denotará con d_i.
- El número total de movimientos M para la clasificación de n datos es:

$$M = \sum_{i=2}^{n} (2 + d_i) = 2(n-1) + \sum_{i=2}^{n} d_i$$

Ejemplo: inserción directa

Arreglo A :	**5**	**2**	**4**	**6**	**1**	**3,**	n=6
		próximo					
i=2	5	2	4	6	1	3	$d_2=1$
			próximo				
i=3	2	5	4	6	1	3	$d_3=1$
				próximo			
i=4	2	4	5	6	1	3	$d_4=0$
					próximo		
i=5	2	4	5	6	1	3	$d_5=4$
						próximo	
i=6	1	2	4	5	6	3	$d_6=3$
A ordenado:	1	2	3	4	5	6	

Ejemplo: inserción directa

Mejor caso, ocurre cuando los datos ya están ordenados.

En este caso no hay movimientos en el ciclo interior:

$$\sum_{i=2}^{n} d_i = 0$$

Y el número total de movimientos: $M = 2(n-1) + \sum_{i=2}^{n} d_i$

Queda en:

$$\boxed{M = 2(n-1) \in O(n)}$$

Ejemplo: inserción directa

Peor caso, cuando los datos están en orden inverso.

En este caso:

$$d_2 = 1$$
$$d_3 = 2$$
$$\dots$$
$$d_n = n - 1$$

Así sumando: $$\sum_{i=2}^{n} d_i = (1 + n - 1)\frac{(n-1)}{2} = n\frac{(n-1)}{2}$$

Número total de movimientos $M = 2(n-1) + \sum_{i=2}^{n} d_i$ resulta:

$$\boxed{M = 2(n-1) + n\frac{(n-1)}{2} = \frac{1}{2}(n-1)(n+4) \in O(n^2)}$$

Ejemplo: inserción directa

Caso promedio: en el paso i cuando se considera el próximo, ya se ha ordenado antes $(i-1)$ datos.

Si el próximo es el más grande, no hay movimientos en el ciclo interior ($d_i=0$). La probabilidad de que el próximo sea el más grande es $1/i$.

Si el próximo es el segundo mayor de los i números habrá 1 intercambio ($d_i=1$) y así sucesivamente.

La probabilidad de que el próximo sea el j-ésimo más grande para $1 \leq j \leq i$ es también $1/i$. Intercambios: $d_i=j-1$.

Por ultimo si el próximo es el más chico de los i números se realizan $i-1$ intercambios ($d_i=i-1$).

Ejemplo: inserción directa

Caso promedio: $M = \sum_{i=2}^{n} (2 + d_i).p(i)$

Se calcula $(2+di).p(i)$ con $d_i=0,1,\ldots,\ i\text{-}1$:

$$\frac{2}{i} + \frac{3}{i} + \frac{4}{i} + \cdots + \frac{i+1}{i} = \sum_{j=1}^{i} \frac{j+1}{i} = \frac{i+3}{2}$$

El número total de movimientos:

$$\boxed{M = \sum_{i=2}^{n} \frac{i+3}{2} = \frac{1}{2}\left(\sum_{i=2}^{n} i + \sum_{i=2}^{n} 3 \right) = \frac{1}{4}(n-1)(n+8) \in O(n^2)}$$

Ejemplo: inserción directa

En resumen, el *número de movimientos* del algoritmo de inserción directa:

- Mejor caso: $2(n-1) \in O(n)$
- Caso promedio: $\frac{1}{4}(n-1)(n+8) \in O(n^2)$
- Peor caso: $\frac{1}{2}(n-1)(n+4) \in O(n^2)$