Algoritmos
Estructuras de Datos I

Facultad de Ciencias Exactas y Tecnología
Universidad Nacional de Tucumán
2025

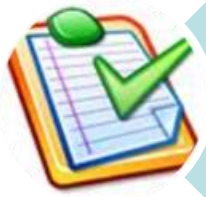
Complejidad
Notación O grande(1)

Elección de un algoritmo

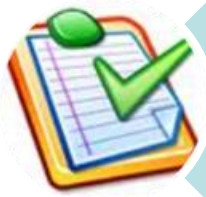
Cuando se resuelve un problema, frecuentemente se debe elegir entre distintos algoritmos.

¿Sobre que base se elige?.

Dos metas contradictorias:



Un algoritmo *simple*, fácil de entender, codificar y de rastrear errores.



Un algoritmo que haga *uso eficiente* de los recursos de la computadora.

Rendimiento de un programa

Hay criterios para juzgar un programa directamente relacionados con la performance del mismo:

- **tiempo de calculo**
- **requisitos de almacenamiento**

Definiciones:

- La ***complejidad de espacio*** es la cantidad de memoria que necesita para realizar toda su ejecución.
- La ***complejidad de tiempo*** es la cantidad de tiempo de computador que necesita para completar su ejecución.

Rendimiento de un programa

Por lo tanto el análisis del rendimiento abarcará un análisis de la :

Complejidad de espacio del programa.

Complejidad de tiempo del programa.

Complejidad de espacio de un Programa

El espacio necesario para un programa es la suma:

Una ***parte fija*** que es independiente de las características de la entrada y la salida. Esta parte incluye el espacio para las instrucciones (el espacio para el código), el espacio para variables simples y variables con componentes de tamaño fijo, espacio para constantes etc.

Una ***parte variable*** que consiste del espacio necesario para variables con componentes cuyo tamaño depende de la instancia particular del problema que se está resolviendo, espacio para variables dinámicas referenciadas y espacio para el stack de recursión.

Complejidad de tiempo de un Programa

El tiempo de un programa P, es la suma del tiempo de **compilación** y del tiempo de **ejecución.**

El tiempo de compilación no depende de las características de la instancia.

Se supone que un programa compilado se ejecutará varias veces sin recompilarlo.

Por lo tanto lo mas significativo será el ***tiempo de ejecución.***

Tiempo de ejecución de un Programa

Para dar una medida del tiempo de ejecución de un programa hay que analizar varios factores, entre ellos los mas importantes son:

- ✓ La ***entrada*** del programa, los datos de entrada, su tamaño.
- ✓ El ***algoritmo*** aplicado.
- ✓ La calidad del ***código generado por el compilador*** usado para crear la imagen objeto.
- ✓ La naturaleza y velocidad de las ***instrucciones de máquina*** usadas para ejecutar el programa.

Complejidad de Tiempo de un programa

- Se llama **T(n)** al *tiempo de ejecución de un programa con una entrada de tamaño n*.
- **T(n)** es la llamada ***complejidad de tiempo del programa***.
- T(n) depende del algoritmo usado.
- Se dice que el tiempo de ejecución es proporcional a T(n), la constante de proporcionalidad depende de la computadora.

Tiempo de ejecución de un algoritmo

- Se denota también 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 la máquina RAM.

Tiempo de ejecución de un algoritmo

- Qué pasa con el tiempo de ejecución T(n) estimado cuando ese algoritmo se implementa en un programa y se ejecuta en una computadora real?

- Y si se cambia de lenguaje y/o de máquina?

- Una respuesta a estas preguntas viene dada por el *principio de invariancia* que afirma que dos implementaciones distintas de un mismo algoritmo no diferirán en su eficiencia en más de alguna constante multiplicativa.

Análisis del tiempo de ejecución de un Programa

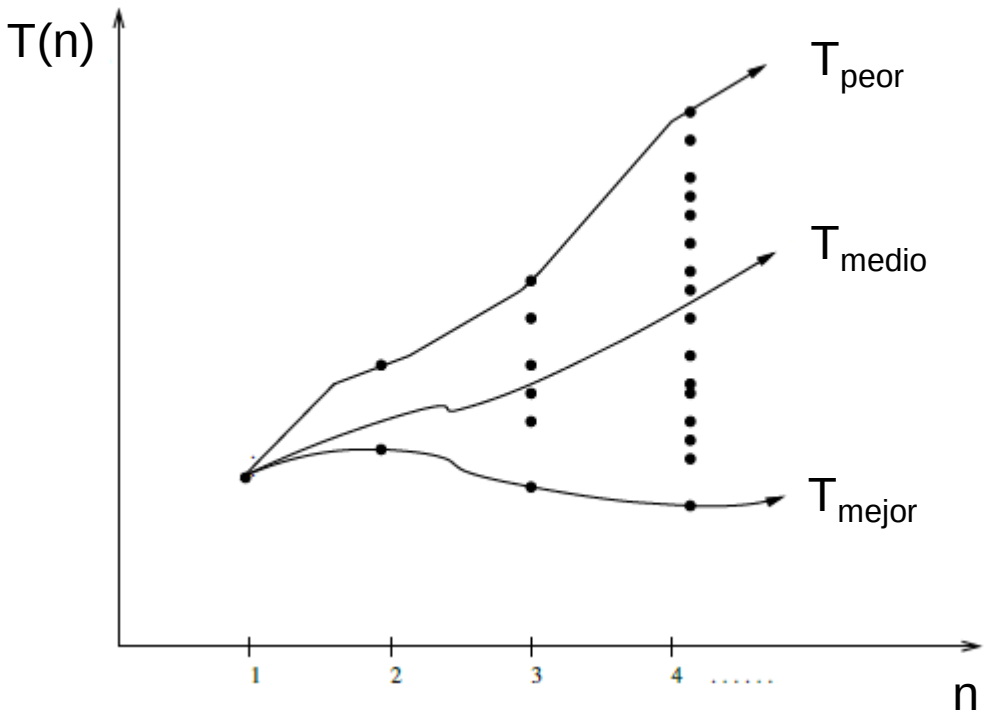
Para algunos programas el ***costo es uniforme*** para distintas muestras de tamaño n: el tiempo de ejecución de un algoritmo no varía para todas las entradas del mismo tamaño.

Para muchos programas el ***costo no es uniforme*** para distintas muestras de tamaño n: se puede encontrar que los tiempos de ejecución de un algoritmo varían para entradas del mismo tamaño.

Análisis del tiempo de ejecución de un Programa

Habrá entonces:

- $T_{mejor}(n)$,
- $T_{medio}(n)$
- $T_{peor}(n)$



Análisis del tiempo de ejecución de un Programa

Cuál elegir entre:

- $T_{mejor}(n)$
- $T_{medio}(n)$
- $T_{peor}(n)$

Se debe definir T(n) como el ***peor*** tiempo de ejecución, esto es, el máximo sobre todas las posibles muestras de tamaño n.

Ej1. Algoritmo para ***encontrar el mayor elemento*** de un arreglo A[1..n], n≥1.

Algoritmo: Máximo
Entrada: **A**[1..n], : vector de números enteros,
n:entero
Salida: **max**: entero

P1. Leer(A,n)
P2. max ← A_1
P3. Para i desde 2 hasta n hacer
Si max < A_i entonces
max ← A_i
P4. Escribir (max)
P5. Fin

Costo uniforme*: hace siempre (n-1) comparaciones*

$T_{mejor}(n) \equiv T_{peor}(n)$

Ej2. Algoritmo de ***Búsqueda Secuencial*** para determinar la pertenencia de un elemento **x** a un arreglo A[1..n], n≥1.

Algoritmo: Búsqueda Secuencial
Entrada: **A**[1..n] : vector de números enteros,
n:entero ; **x**:entero
Salida: encontrado: bool

P1. Leer(A,n,x)
P2. encontrado ← falso
P3. i ← 1
P4. Mientras (i ≤ n AND $A_i \neq x$) hacer
i ← i+1
P5. encontrado ← (i ≤ n)
P6. Escribir (encontrado)
P7. Fin

Costo no uniforme*: mejor caso: 1 comparación ,*
peor caso: n comparaciones.

$T_{mejor}(n) << T_{peor}(n)$

Eficiencia de algoritmos

Dos enfoques para el análisis de eficiencia de los algoritmos:

- El enfoque ***experimental*** (a posteriori) consiste en programar todos los algoritmos y probarlos con diferentes instancias con la ayuda de la computadora. Luego medir los tiempos de ejecución.
- El enfoque ***teórico*** (a priori) consiste en determinar matemáticamente la cantidad de recursos (tiempo de ejecución, espacio en memoria, etc.) necesarios para cada algoritmo en función del tamaño de la instancia considerada.

Eficiencia de algoritmos

Las ventajas del análisis teórico son:

- No depende ni de la computadora usada ni del lenguaje de programación, ni siquiera de la habilidad del programador.
- Ahorra el tiempo de programar algoritmos ineficientes, así como el tiempo el tiempo de máquina necesario para probarlos.
- Permite además estudiar la eficiencia de un algoritmo cuando se estudia instancias de cualquier tamaño.

Eficiencia de algoritmos

- El análisis requerido para estimar los recursos que usa un algoritmo es un problema teórico, y por ello se necesita un marco formal para su tratamiento.

- Se va a presentar una notación asintótica que permite comparar el orden de crecimiento de T(n), donde n es el tamaño de la entrada del algoritmo.

- Por el ***principio de invarianza*** se puede pensar **T(n) como el número de instrucciones en una computadora ideal.**

Notación O grande

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:

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

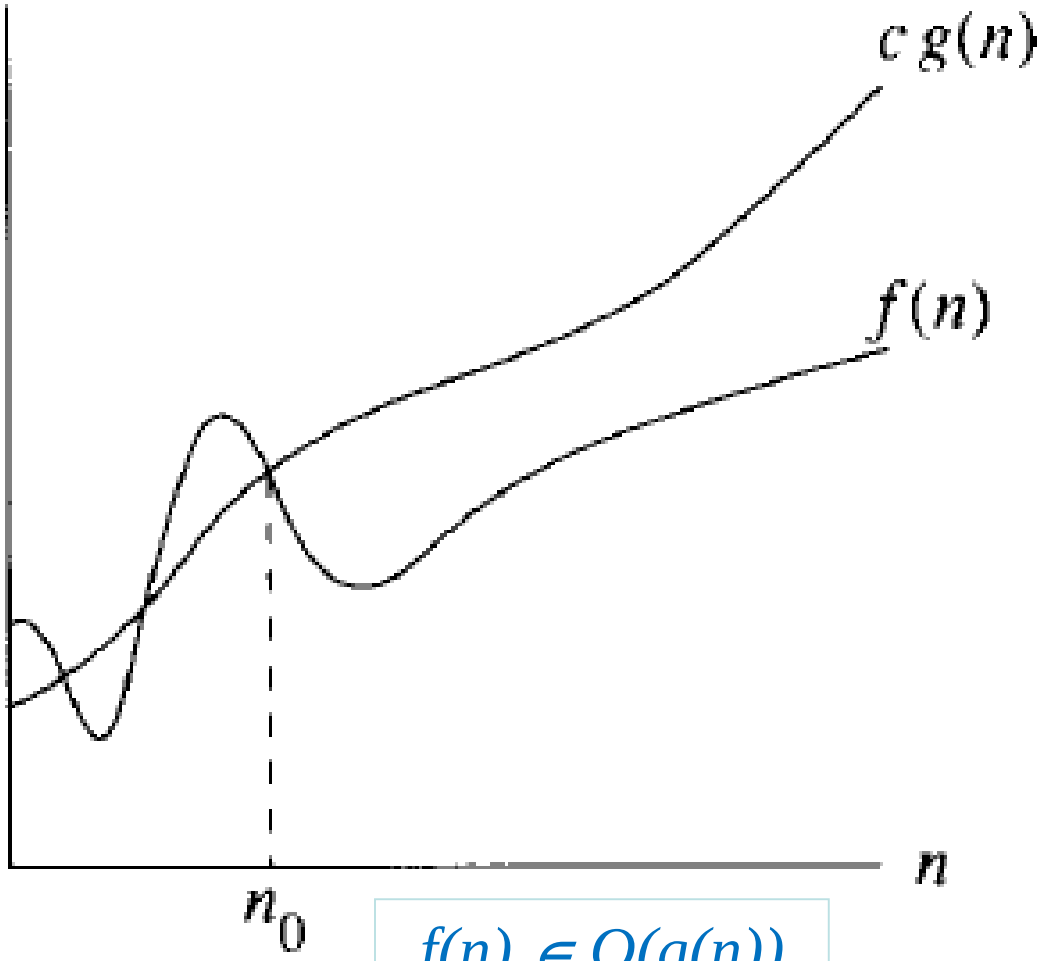
Si una función g ∈ a O(f(n)) se dice que:

g es O grande de f(n)

o que: *g es de orden f(n)*.

Notación O grande

En un gráfico:



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

Notación O grande

- $f(n) \in O(f(n))$
- $f(n) \in O(g(n)) \Rightarrow O(f(n)) \subset O(g(n))$
- Si $f(n) \in O(g(n)) \Rightarrow c.f(n) \in O(g(n)) \quad \forall\, c \in R+$
- Si $f(n) \in O(g(n))$ y $h(n) \in O(g(n)) \Rightarrow f(n) + h(n) \in O(g(n))$
- $f(n) \in O(g(n))$ y $g(n) \in O(f(n)) \Leftrightarrow O(f(n)) = O(g(n))$
- $f(n) \in O(g(n))$ y $f(n) \in O(h(n)) \Rightarrow f(n) \in O(\min(g(n),h(n)))$

- **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**:
  $f1(n) \in O(g(n))$ y $f2(n) \in O(h(n)) \Rightarrow f1(n)+f2(n) \in O(\max(g(n),h(n)))$

- **Regla del producto**:
  $f1(n) \in O(g(n))$ y $f2(n) \in O(h(n)) \Rightarrow f1(n).f2(n) \in O(g(n).h(n))$

Notación O grande
Ejemplos:

Verdadero o Falso

- $3n+2 \in O(n)$?
- $10n+100 \in O(n)$?
- $3n^2 \in O(n)$?
- $(n+1)^2 \in O(n^2)$?
- $\log_4 n \in O(\log_2 n)$?
- $\log_2 n \in O(\log_4 n)$?
- $2^n \in O(3^n)$?

Ejemplo: $3n+2 \in O(n)$?

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

Dividir por n>0 ambos miembros de la desigualdad:

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

$3+2/n \leq c$ para $n \geq n_0$

Vale para: $c=5$, $n_0=1$ o para: $c=4$, $n_0=2$ o…

Etc.

➔ $3n+2 \in O(n)$

Ejemplo: $10n+100 \in O(n)$?

ya que:

$$10n+100 \leq 20n \qquad \text{para } n \geq 10$$

Vale para: $c=20$, $n_0=10$

➔ $10n+100 \in O(n)$

Ejemplo: $3n^2 \in O(n)$?

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

$3n \leq c$ para $n \geq n_0$

3n crece con n $\Rightarrow$ no se puede acotar para ninguna constante c cuando n crece.

➔ $3n^2 \notin O(n)$

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

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

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

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

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

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

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

➔ $(n+1)^2 \in O(n^2)$

Ejemplo: $\log_4 n \in O(\log_2 n)$?

Recordar que: $$log_a n = \frac{log_b n}{log_b a}$$

Entonces ya que:

$\log_4 n = \log_2 n / \log_2 4 = \log_2 n / 2$ para $n \geq 1$

Si $c=0.5$, $n_0=1$:

$\log_4 n = 0.5 \log_2 n$ para $n \geq 1$

➔ $\log_4 n \in O(\log_2 n)$

Ejemplo: $\log_2 n \in O(\log_4 n)$?

Ya que:

$\log_2 n = 2 \log_4 n$ para $n \geq 1$

$c=2$, $n_0=1$

➔ $\log_2 n \in O(\log_4 n)$

Ejemplo: $2^n \in O(3^n)$?

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

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

$(2/3)^n \leq c$ para $n \geq n_0$

para: $c=2/3$, $n_0=1$

$2^n \leq 2/3 \, . \, 3^n$ para $n \geq 1$

➔ $2^n \in O(3^n)$

Notación O grande
Ejercicios:

Verdadero o Falso

- $4n+1 \in O(n^2)$?
- $n - 1 \in O(1)$?
- $\log_2 n \in O(n)$?
- $2n+ \log_4 n \in O(n^2)$?
- $\sqrt{n} \in O(n)$?
- $3^n \in O(2^n)$?