Análisis de Algoritmos

Programación de Estructuras de Datos y Algoritmos Fundamentales (TC1031)

M.C. Xavier Sánchez Díaz
sax@tec.mx

Tecnológico de Monterrey

Outline

¿Qué es un *algoritmo*?

Eficiencia de un algoritmo

Introducción

Q: Si tuviéramos dos algoritmos que resuelven el mismo problema. . . ¿cómo sabemos cuál de ellos nos conviene utilizar?

A: Aquél que sea el más *eficiente* (casi siempre)

Ya sea hablando de **tiempo de ejecución** o bien en **uso de espacio en memoria**.

¿Qué es más barato? ¿Tiempo de procesamiento o almacenamiento?

Eficiencia de un algoritmo

Introducción

Q: Si tuviéramos dos algoritmos que resuelven el mismo problema. . . ¿cómo sabemos cuál de ellos nos conviene utilizar?

A: Aquél que sea el más *eficiente* (casi siempre)

Ya sea hablando de **tiempo de ejecución** o bien en **uso de espacio en memoria**.

¿Qué es más barato? ¿Tiempo de procesamiento o almacenamiento?

Eficiencia de un algoritmo
Introducción

Q: Si tuviéramos dos algoritmos que resuelven el mismo problema. . . ¿cómo sabemos cuál de ellos nos conviene utilizar?

A: Aquél que sea el más *eficiente* (casi siempre)

Ya sea hablando de **tiempo de ejecución** o bien en **uso de espacio en memoria**.

¿Qué es más barato? ¿Tiempo de procesamiento o almacenamiento?

Tiempo de Ejecución
Introducción

El tiempo de ejecución de un algoritmo depende de muchos factores:

1. **Entrada del programa**. No es lo mismo ordenar una lista con 10 elementos que una de 1 millón de elementos.
2. **Calidad del código compilado**. Existen algunas instrucciones más rápidas que otras, y algunos compiladores más veloces que otros.
3. **Implicaciones de hardware**. Algunos procesadores son más rápidos que otros.
4. **Complejidad del problema**. Es más difícil multiplicar dos matrices que ordenar una lista.

Tiempo de Ejecución

Introducción

El tiempo de ejecución de un algoritmo depende de muchos factores:

1. **Entrada del programa**. No es lo mismo ordenar una lista con 10 elementos que una de 1 millón de elementos.
2. **Calidad del código compilado**. Existen algunas instrucciones más rápidas que otras, y algunos compiladores más veloces que otros.
3. **Implicaciones de hardware**. Algunos procesadores son más rápidos que otros.
4. **Complejidad del problema**. Es más difícil multiplicar dos matrices que ordenar una lista.

Tiempo de Ejecución

Introducción

El tiempo de ejecución de un algoritmo depende de muchos factores:

1. **Entrada del programa**. No es lo mismo ordenar una lista con 10 elementos que una de 1 millón de elementos.
2. **Calidad del código compilado**. Existen algunas instrucciones más rápidas que otras, y algunos compiladores más veloces que otros.
3. **Implicaciones de hardware**. Algunos procesadores son más rápidos que otros.
4. **Complejidad del problema**. Es más difícil multiplicar dos matrices que ordenar una lista.

Tiempo de Ejecución

Introducción

El tiempo de ejecución de un algoritmo depende de muchos factores:

1. **Entrada del programa**. No es lo mismo ordenar una lista con 10 elementos que una de 1 millón de elementos.
2. **Calidad del código compilado**. Existen algunas instrucciones más rápidas que otras, y algunos compiladores más veloces que otros.
3. **Implicaciones de hardware**. Algunos procesadores son más rápidos que otros.
4. **Complejidad del problema**. Es más difícil multiplicar dos matrices que ordenar una lista.

Tiempo de Ejecución

Introducción

El tiempo de ejecución de un algoritmo depende de muchos factores:

1. **Entrada del programa**. No es lo mismo ordenar una lista con 10 elementos que una de 1 millón de elementos.
2. **Calidad del código compilado**. Existen algunas instrucciones más rápidas que otras, y algunos compiladores más veloces que otros.
3. **Implicaciones de hardware**. Algunos procesadores son más rápidos que otros.
4. **Complejidad del problema**. Es más difícil multiplicar dos matrices que ordenar una lista.

Tiempo de Ejecución

Introducción

El tiempo de ejecución de un algoritmo depende de muchos factores:

1. **Entrada del programa**. No es lo mismo ordenar una lista con 10 elementos que una de 1 millón de elementos.
2. **Calidad del código compilado**. Existen algunas instrucciones más rápidas que otras, y algunos compiladores más veloces que otros.
3. **Implicaciones de hardware**. Algunos procesadores son más rápidos que otros.
4. **Complejidad del problema**. Es más difícil multiplicar dos matrices que ordenar una lista.

La complejidad **temporal** de un algoritmo hace referencia al tiempo requerido por un algoritmo para ejecutarse, expresado con base en una **función** que depende del tamaño del problema.

Podemos usar la misma idea para expresar también su **complejidad espacial**, es decir el **espacio en memoria** que necesita dicho algoritmo...

La complejidad **temporal** de un algoritmo hace referencia al tiempo requerido por un algoritmo para ejecutarse, expresado con base en una **función** que depende del tamaño del problema.

Podemos usar la misma idea para expresar también su **complejidad espacial**, es decir el **espacio en memoria** que necesita dicho algoritmo...

Ejemplo
Complejidad

z = 0;	$\longrightarrow$	1 asignación
for x = 1; x $\leq n$; x++ **do**	$\longrightarrow$	1 asignación + $(n+1)$ comparaciones
for y = 1; y $\leq n$; y++ **do**	$\longrightarrow$	$n(n+2) = n^2 + 2n$
z = z + a[x, y];	$\longrightarrow$	$n \times n = n^2$
end for	$\longrightarrow$	$2n^2$ (incremento + 1 `goto` implícito)
end for	$\longrightarrow$	n (`goto` implícito cuando y sea `false`)
	$\longrightarrow$	$2n$ (incremento + `goto` implícito)
	$\longrightarrow$	1 (`goto` implícito cuando x sea `false`)

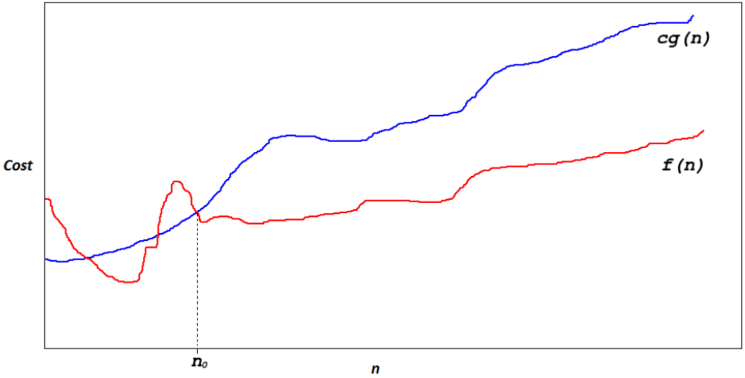
$$\text{Total} = 4n^2 + 6n + 4$$

Notación asintótica: $\mathcal{O}(g)$

Orden Asintótico

Definición 1

Sea $g\colon \mathbb{N} \to \mathbb{R}^+$. $\mathcal{O}(g)$ es el conjunto de funciones $f\colon \mathbb{N} \to \mathbb{R}^+$ tal que para cualquier constante $c \in R^+$ y alguna $n_0 \in \mathbb{N}$, $f(n) \leq cg(n) \forall n \geq n_0$.



Ejemplos

Notación asintótica

- $g = n + 5$ pertenece al grupo de funciones $\mathcal{O}(n)$ pues $n + 5 \leq 2n$ para cada $n \geq 5$.
- $g = (n + 1)^2$ es del grupo $\mathcal{O}(n^2)$ porque $(n + 1)^2 \leq 4n^2$ para cada $n \geq 1$.
- $g = n + 1)^2$ no es del grupo $\mathcal{O}(n)$ porque para cualquier $c > 1$ no se cumplirá nunca que $(n + 1)^2 \leq cn$.

Otra manera de verlo

Orden asintótico

Podemos pensar en la notación de $\mathcal{O}(g)$ como un "*a lo mucho*":

$g = n + 5$ crece a lo mucho como n

pues la notación $\mathcal{O}$ impone un **límite superior** (*upper bound*).

Otra manera de verlo

Orden asintótico

Podemos pensar en la notación de $\mathcal{O}(g)$ como un "*a lo mucho*":

$g = n + 5$ crece a lo mucho como n

pues la notación $\mathcal{O}$ impone un **límite superior** (*upper bound*).

Podemos pensar en la notación de $\mathcal{O}(g)$ como un "*a lo mucho*":

$g = n + 5$ crece a lo mucho como n

pues la notación $\mathcal{O}$ impone un **límite superior** (*upper bound*).

Notación asintótica: $\Omega(g)$ y $\Theta(g)$

Orden asintótico

- $\Omega(g)$ se usa como un **límite inferior** (*lower bound*), es decir que una función $f \in \Omega(g)$ es una función que crece al menos tan rápido como g.

- $\Theta(g)$ se usa para definir un grupo de funciones del **mismo orden**, es decir que una función $f \in \Theta(g)$ es una función que crece igual de rápido como g.

Órdenes más comunes
Orden Asintótico

- $\mathcal{O}(1)$ que es constante
- $\mathcal{O}(\log n)$ que es logarítmico
- $\mathcal{O}(n)$ que es *lineal*
- $\mathcal{O}(n \log n)$
- $\mathcal{O}(n^2)$ que es *cuadrático*
- $\mathcal{O}(n^2 logn)$
- $\mathcal{O}(n^m)$ que es *polinomial*
- $\mathcal{O}(m^n)$ que es *exponencial*
- $\mathcal{O}(n!)$ que es *factorial*