Algoritmos y estructuras de datos II
Complejidad I

Carlos Gustavo Lopez Pombo
(Charlie)

Departamento de Computación,
Facultad de ciencias exactas y naturales,
Universidad de Buenos Aires

Complejidad

Motivación

Desde un punto de vista **práctico** no todo algoritmo que satisface una especificación da lo mismo.

Veamos un ejemplo...

Complejidad

Motivación

```
void max_min (int *datos, int cant, int &max, int &min){
        max = datos[0];
        for (int i = 1; i < cant; i++)
            if (max < datos[i]) then max = datos[i];
        min = datos[0];
        for (int i = 1; i < cant; i++)
            if (min > datos[i]) then min = datos[i];
}
```

Complejidad

Motivación

```
void max_min (int *datos, int cant, int &max, int &min){
        max = datos[0], min = datos[0];
        for (int i = 1; i < cant; i++){
            if (max < datos[i]) then max = datos[i];
            if (min > datos[i]) then min = datos[i];
        }
}
```

Complejidad

Existen dos tipos de **complejidad**, la **temporal** y **espacial**, y sirven para saber cuanto nos cuesta resolver un problema en tiempo y espacio, respectivamente.

En la materia sólo nos preocuparemos por la **complejidad temporal**

¿Siempre podemos abstraernos de la complejidad espacial?

Complejidad

¿Cómo podemos medir el costo temporal de un algoritmo?

1. **Usando un cronómetro** Se suele llamar **wall time**. Lo **bueno** es que nos dice objetivamente cuánto tarda, lo **malo** es que depende de factores completamente ajenos al algoritmo en sí: la máquina y los datos. *Ni siquiera es confiable entre dos ejecuciones consecutivas*.

2. **¡Contando operaciones elementales!** Se suele llamar **complejidad algorítmica**. Se trata de acotar la cantidad de **operaciones elementales** que toma resolver un problema en función del tamaño de la entrada.

3. **Usando un medidor de recursos** Se suele llamar **CPU time**. Lo **bueno** es que nos dice cuántos recursos utilizamos, lo **malo** es que depende de la computadora específica; luego si mañana cambio de computadora, cambia el comportamiento.

Complejidad

¿Cómo podemos medir el costo temporal de un algoritmo?

Debiéramos encontrar una métrica que sea independiente de la “máquina” en la que se ejecuta...

... e incluso del lenguaje en el que está implementado

Complejidad

¿Cuánto tiempo nos cuesta resolver el problema de buscar el máximo y el mínimo con este algoritmo?

```
void max_min (int *datos, int cant, int &max, int &min){
    max = datos[0];                                          c1
    for (int i = 1; i < cant; i++)                           cant*c2
        if (max < datos[i]) then max = datos[i];             cant*c3
    min = datos[0];                                          c4
    for (int i = 1; i < cant; i++)                           cant*c5
        if (min > datos[i]) then min = datos[i];             cant*c6
}
```

Complejidad

¿Cuánto tiempo nos cuesta resolver el problema de buscar el máximo y el mínimo con este algoritmo?

```
void max_min (int *datos, int cant, int &max, int &min){
max = datos[0], min = datos[0];                          2*c1'
for (int i = 1; i < cant; i++){                          cant*c2'
     if (max < datos[i]) then max = datos[i];            cant*c3'
     if (min > datos[i]) then max = datos[i];            cant*c4'
    }
}
```

Complejidad

¿Cuál de los dos algoritmos es mejor, el que tarda *c1+cant*c2+cant*c3+c1+cant*c2+cant*c3*, o el que tarda 2**c1*'+*cant*c2*'+2**cant*c3*'?

¿Qué es lo que expresan las constantes?

Las constantes reflejan el costo de las operaciones elementales en el lenguaje y plataforma puntual en la que se ejecuta el algoritmo

Aquello de lo que queremos abstraernos...

Complejidad

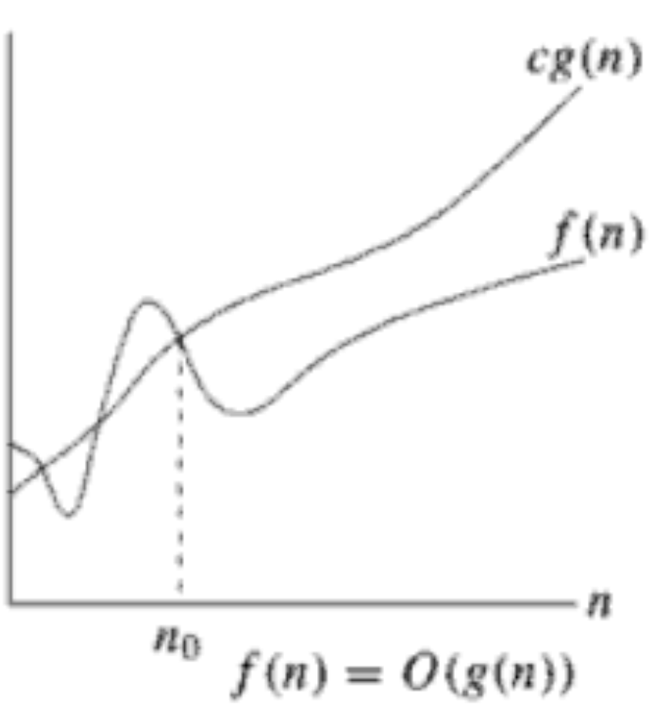
Clases de funciones

Para poder abstraernos de las constantes, deberemos poder clasificar las funciones de acuerdo a su razón de crecimiento...

Para ello, lo que se hace es determinar cotas ajustadas por encima y por debajo que permiten reflejar esta razón.

Complejidad

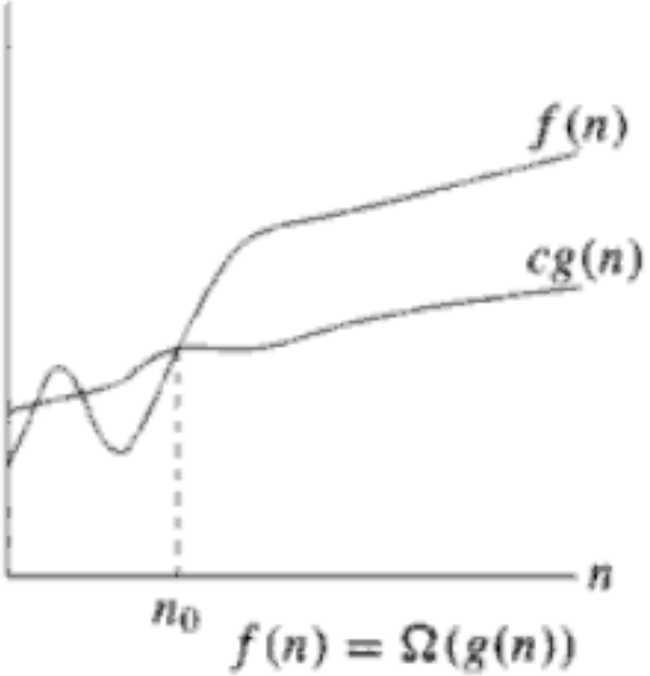
La clase $O(g(n))$



$$O(g(n)) = \{f(n)|(\exists c, x_o)(\forall x_0 \leq x)(f(x) < c * g(x))\}$$

Complejidad

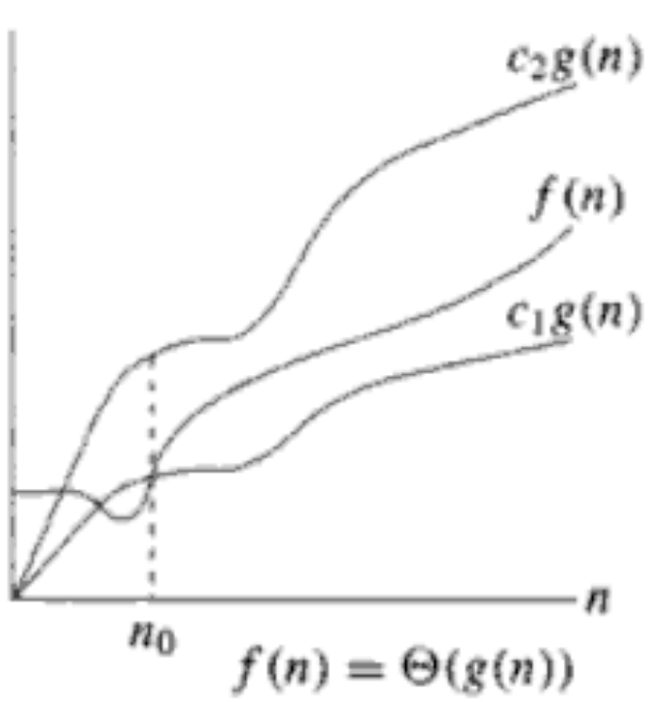
La clase $\Omega(g(n))$



$$\Omega(g(n)) = \{f(n)|(\exists c, x_o)(\forall x_0 \leq x)(c * g(x)) < f(x)\}$$

Complejidad

La clase $\Theta(g(n))$



$$\Theta(g(n)) = \{f(n)|(\exists c_1, c_2, x_o)(\forall x_0 \leq x)(c_1 * g(x)) < f(x) \leq c_2 * g(x))\}$$

Complejidad

Complejidad de un algoritmo

Luego, dada la función de costo de un algoritmo particular, se puede probar su pertenencia a una clase de funciones determinada.

Veamos un ejemplo...

Repaso

- Presentamos el concepto naïf de costo temporal de un algoritmo
- Formalizamos varias nociones de complejidad de un algoritmo usando clases de funciones
- Presentamos un pequeño ejemplo de cómo usarlas.

¡Es todo por hoy!