Clase 23 - Algoritmos y Complejidad Computacional

IIC1001 - Algoritmos y Sistemas Computacionales

Cristian Ruz – cruz@uc.cl

Jueves 4-Junio-2026

Departamento de Ciencia de la Computación
Escuela de Ingeniería
Pontificia Universidad Católica de Chile

Contenidos

Contacto

Temas

Algoritmos

Contenidos

Contacto

Temas

Algoritmos

Contacto

🐐

vicente.cabra@uc.cl

Vicente Cabra

Ayudante jefe



🐸

alejandro.tapia@uc.cl

Alejandro Tapia

Ayudante jefe



🛼

gabriel.sandoval@uc.cl

Gabriel Sandoval

Ayudante jefe



Contenidos

Contacto

Temas

Algoritmos

Temas del curso

Sistemas computacionales

- Representación datos, números y codificación
- Funcionamiento hardware, procesadores y memoria.
- Funcionamiento de sistemas operativos
- Funcionamiento de Internet
- Herramientas computacionales: github + latex

Algoritmos

- Algoritmos y resolución de problemas
- Eficiencia algorítmica
- Estructuras secuenciales y ordenamiento
- Grafos y árboles

Próximos eventos

Viernes 5-Junio: Recorrección presencial I1 y ACs

- 13:30-15:40
- Sala H1

Martes 9-Junio: Algoritmos de ordenamiento

Jueves 11-Junio: Actividad 6

- En clase

Viernes 12-Junio: Notas I2

Contenidos

Contacto

Temas

Algoritmos

Complejidad algorítmica

¿Cuánto demora este algoritmo?

```
def f (a):
    b = a * 42
    c = a // 50
    d = b + c
    return d
```

Complejidad algorítmica

¿Cuánto demora este algoritmo?

```
def g ():
    n = 10
    s = 0
    for i in range(10):
        s += i
    return s
```

Complejidad algorítmica

¿Cuánto demora este algoritmo?

```
def h (n):
    s = 0
    for i in range(n):
        s += i
    return s
```

Complejidad algorítmica

¿Cuánto demora este algoritmo?

```
def buscar_fijo (a, x):
    n = 50
    for i in range(n):
        if a[i] == x:
            return True
    return False
```

Complejidad algorítmica

¿Cuánto demora este algoritmo?

```
def buscar_fijo (a, x):
    n = len(a)
    for i in range(n):
        if a[i] == x:
            return True
    return False
```

Complejidad algorítmica

¿Cuánto demora este algoritmo?

```
def ciclo1 (a):
    n = len(a)
    for i in range(n):
        s = 0
        for j in range(10):
            s += j
        a[i] = s
    return a
```

Complejidad algorítmica

¿Cuánto demora este algoritmo?

```
def ciclo2 (a):
    n = len(a)
    for i in range(n):
        s = 0
        for j in range(i):
            s += j
        a[i] = s
    return a
```

Complejidad algorítmica

¿Cuánto demora este algoritmo?

```
def ciclo3 (a):
    n = len(a)
    for i in range(n):
        for j in range(n):
            a[i][j] = a[i][j] * 2
    return a
```

Complejidad algorítmica

¿Cuánto demora este algoritmo?

```
def busqueda_binaria(n, lista):
    inicio = 0
    fin = len(lista)-1
    while inicio <= fin:
        medio = (inicio + fin) // 2
        if lista[medio] == n:
            return True
        if n < lista[medio]:
            fin = medio - 1
        elif n > lista[medio]:
            inicio = medio + 1
    return False
```

Complejidad algorítmica

Algorithm 4 Selection Sort

```
1: for i = 1 to n − 1 do
2:     min = i
3:     for j = i + 1 to n do
4:         // Find the index of the i^th smallest element
5:         if A[j] < A[min] then
6:             min = j
7:         end if
8:     end for
9:     Swap A[min] and A[i]
10: end for
```

Complejidad algorítmica

```
INSERTION-SORT(A)
1  for j = 2 to A.length
2      key = A[j]
3      // Insert A[j] into the sorted sequence A[1 .. j − 1].
4      i = j − 1
5      while i > 0 and A[i] > key
6          A[i + 1] = A[i]
7          i = i − 1
8      A[i + 1] = key
```

Complejidad algorítmica

```
QUICKSORT(A, p, r)
1  if p < r
2     then q ← PARTITION(A, p, r)
3          QUICKSORT(A, p, q − 1)
4          QUICKSORT(A, q + 1, r)
```

To sort an entire array A, the initial call is QUICKSORT(A, 1, $length[A]$).

```
PARTITION(A, p, r)
1  x ← A[r]
2  i ← p − 1
3  for j ← p to r − 1
4       do if A[j] ≤ x
5            then i ← i + 1
6                 exchange A[i] ↔ A[j]
7  exchange A[i + 1] ↔ A[r]
8  return i + 1
```

Complejidad algorítmica

¿Qué algoritmo es más rápido?

¿Cómo determinamos qué algoritmo se comporta mejor?

¿De qué depende?

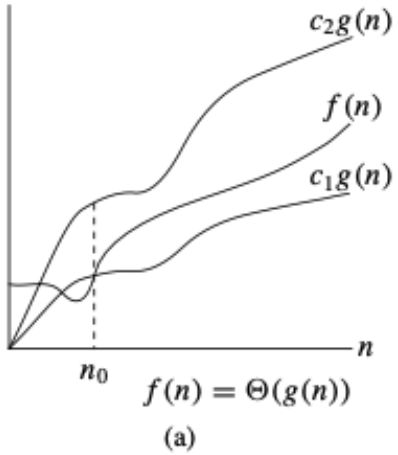
Complejidad algorítmica

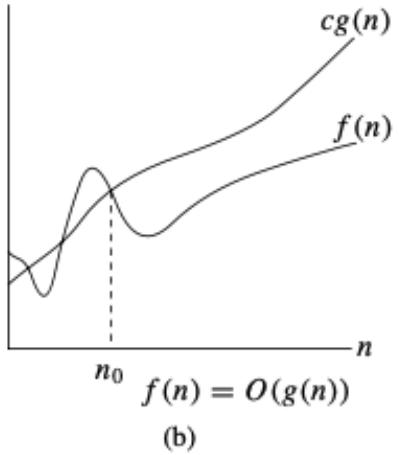
Notación asintótica nos permite establecer el comportamiento de un algoritmo “en el largo plazo”

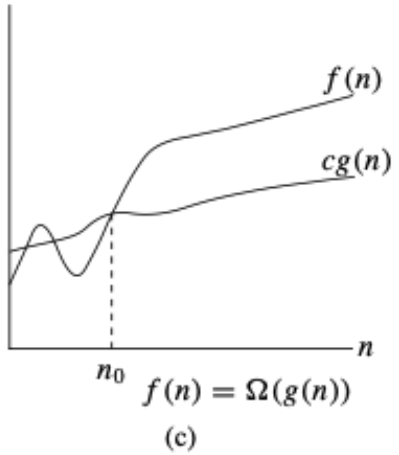
$$\Theta(g(n)) = \{f(n) : \text{there exist positive constants } c_1, c_2, \text{ and } n_0 \text{ such that } 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ for all } n \geq n_0\}.^{1}$$

Complejidad algorítmica

Notación asintótica nos permite establecer el comportamiento de un algoritmo “en el largo plazo”





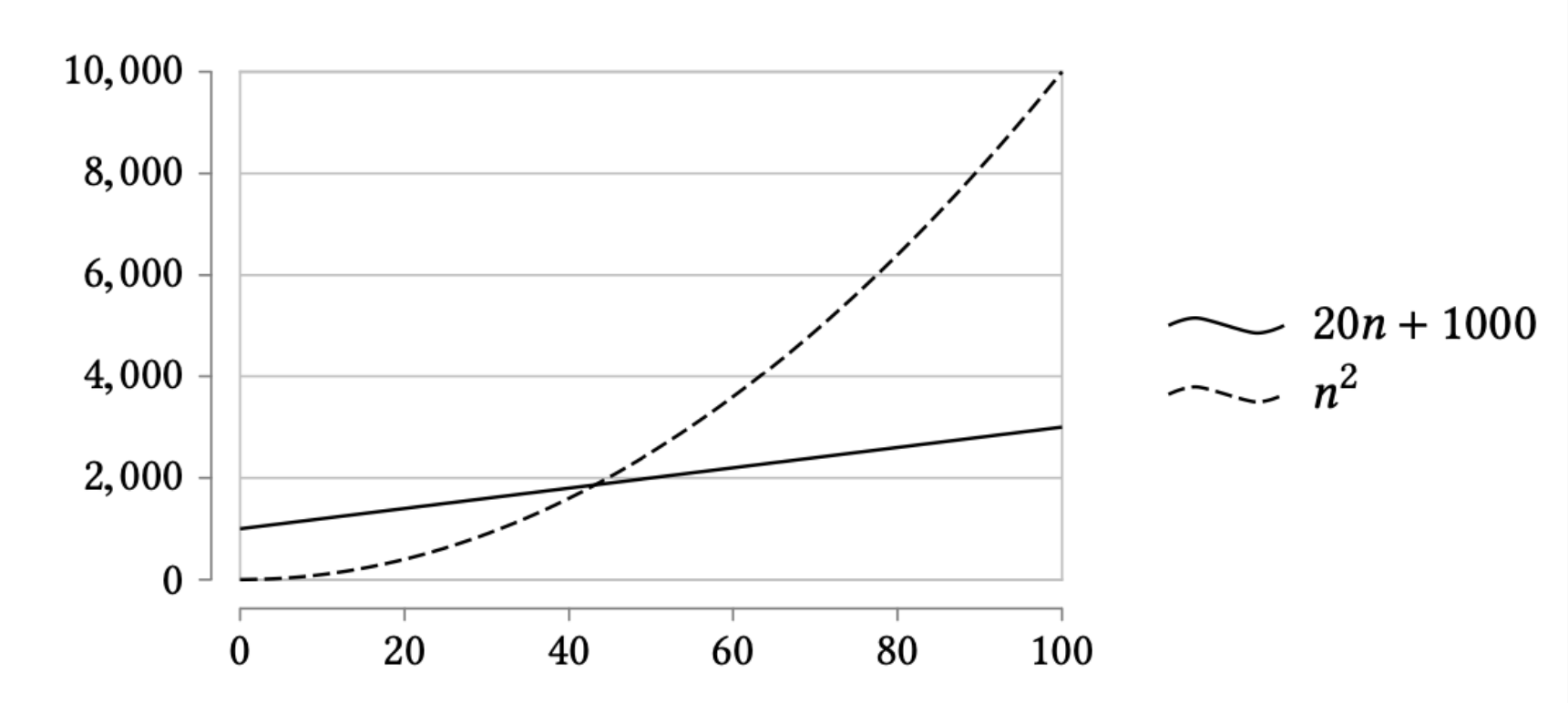


$$\Theta(g(n)) = \{f(n) : \text{there exist positive constants } c_1, c_2, \text{ and } n_0 \text{ such that } 0 \le c_1 g(n) \le f(n) \le c_2 g(n) \text{ for all } n \ge n_0\}.^1$$

$$O(g(n)) = \{f(n) : \text{there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \le f(n) \le cg(n) \text{ for all } n \ge n_0\}.$$

$$\Omega(g(n)) = \{f(n) : \text{there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \le cg(n) \le f(n) \text{ for all } n \ge n_0\}.$$

Complejidad algorítmica



Complejidad algorítmica

Growth of functions.

Function	Input size				
	1	10	100	1000	1,000,000
$\lg(n)$	0	3.32	6.64	9.97	19.93
n	1	10	100	1000	1,000,000
$n\ln(n)$	0	33.22	664.39	9965.78	1.9×10^7
n^2	1	100	10,000	1,000,000	10^{12}
n^3	1	1000	1,000,000	10^9	10^{18}
2^n	2	1024	1.3×10^{30}	10^{301}	$10^{10^{5.5}}$
$n!$	1	3,628,800	9.33×10^{157}	4×10^{2567}	$10^{10^{6.7}}$

Complejidad algorítmica

