Diseño y Análisis de Algoritmos ICI-522

Sergio Hernández.
PhD computer science

Departamento de Computación e Informática
Universidad Católica del Maule.
shernandez@ucm.cl

Algoritmos v/s Programas

Tipos de Problemas

- Los programas resuelven problemas, pero existen muchas formas de resolver un problema. Cómo saber cuándo un programa es mejor que otro?

Algoritmos v/s Programas

Tipos de Problemas

- Los programas resuelven problemas, pero existen muchas formas de resolver un problema. Cómo saber cuándo un programa es mejor que otro?
- El análisis de algoritmos es la manera de determinar los requerimientos de tiempo y espacio requeridos por un programa.

Análisis de Algoritmos (Knuth, 1968)

- En los años 60s, el científico computacional Donald Knuth publica una serie de libros que sientan las bases de la disciplina.
- La idea es calcular el costo de cada operación unitaria y luego sumar el costo total de un algoritmo y calcular el tiempo de ejecución.

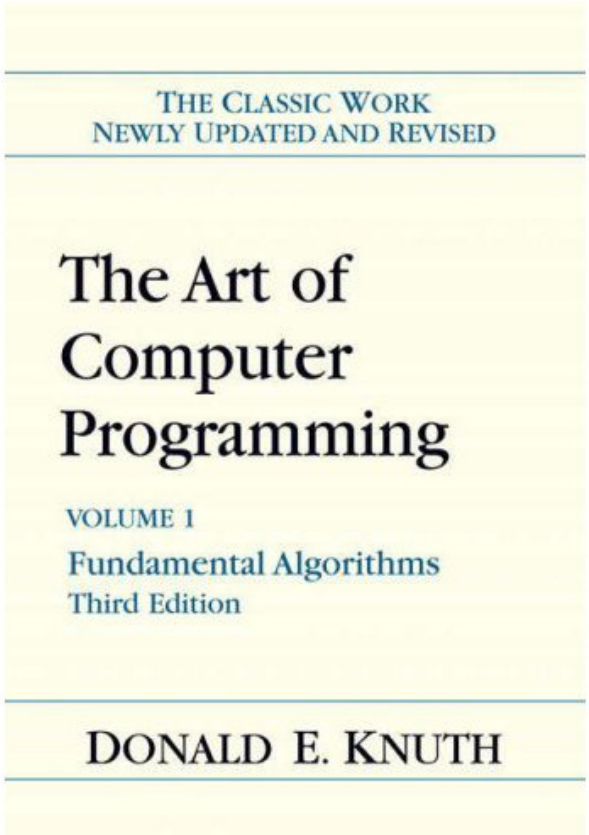


Figura: https://commons.wikimedia.org/wiki/File:ArtOfComputerProgramming.jpg

Análisis de Algoritmos (Cormen et.al., 2009)

- En la practica, el desempeño no solamente depende de las implementaciones sino que también de los casos de estudio.
- Debido a esto, una perspectiva más realista es considerar el **peor caso** (cota superior del tiempo de ejecución para cualquier entrada).
- Ahora debemos enfocarnos en qué tan rápido crece una función con el tamaño de la entrada. A esto lo llamamos la **tasa de crecimiento** del tiempo de ejecución.

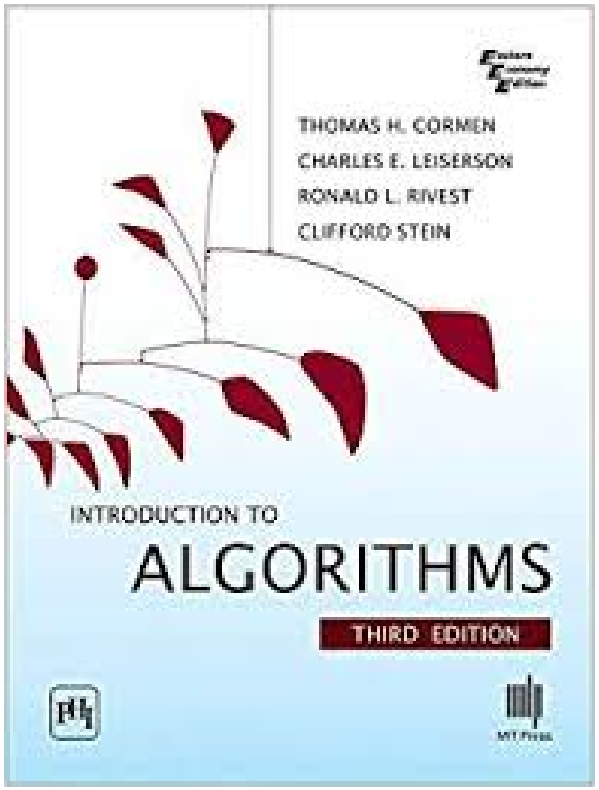


Figura:
https://mitpress.mit.edu/books/introduction-algorithms-third-edition

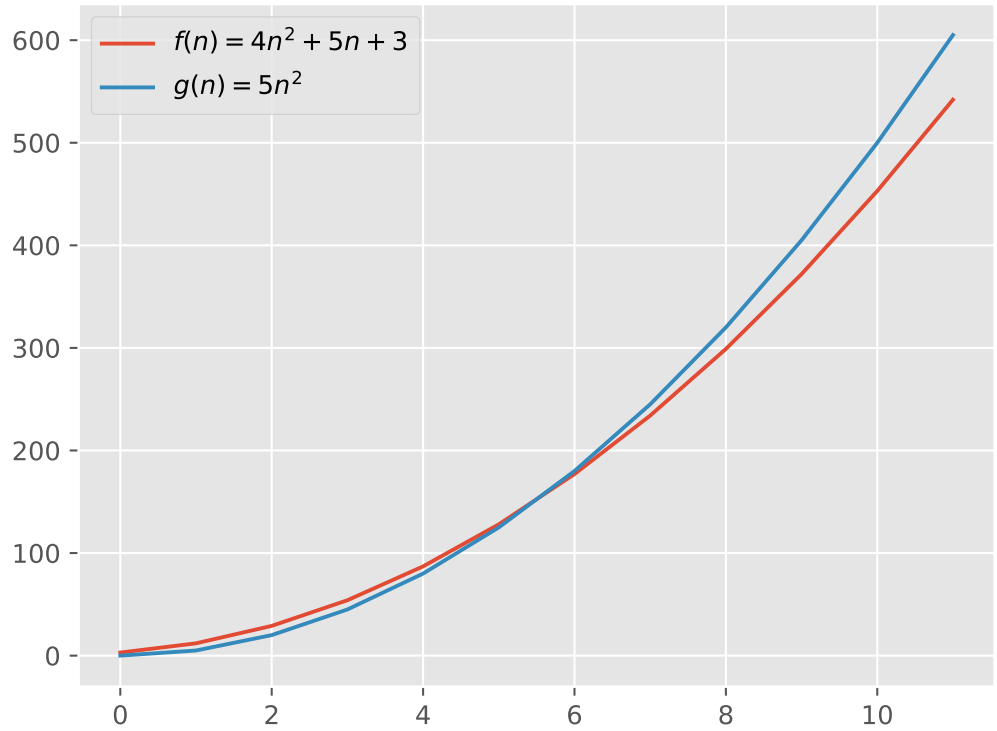
Notación O grande

$f(n) = O(g(n))$

Se dice que una función $f(n)$ está acotada por $g(n)$ si $f(n) \leq k \times g(n)$ para todo $n \geq n_0$ donde k y n_0 son constantes.

n	$f(n)$	$g(n)$
1	12	5
2	29	20
3	54	45
4	87	80
5	128	125
6	177	180

$n_0 = 6, k = 5$



Notación O grande

Cotas

- $f(n) = O(g(n))$ significa que $kg(n)$ es una cota superior para $f(n)$.

Notación O grande

Cotas

- $f(n) = O(g(n))$ significa que $kg(n)$ es una cota superior para $f(n)$.
- $f(n) = \Omega(g(n))$ significa que $kg(n)$ es una cota inferior para $f(n)$.

Notación O grande

Cotas

- $f(n) = O(g(n))$ significa que $kg(n)$ es una cota superior para $f(n)$.
- $f(n) = \Omega(g(n))$ significa que $kg(n)$ es una cota inferior para $f(n)$.
- $f(n) = \Theta(g(n))$ significa que $k_1 g(n)$ es una cota inferior y $k_2 g(n)$ es una cota superior para $f(n)$.

Relaciones

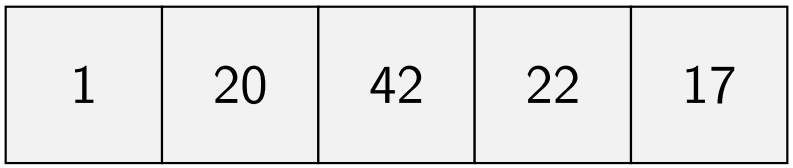
$$O(f(n)) + O(g(n)) \rightarrow O(\max(f(n), g(n)))$$
$$\Omega(f(n)) + \Omega(g(n)) \rightarrow \Omega(\max(f(n), g(n)))$$
$$\Theta(f(n)) + \Theta(g(n)) \rightarrow \Theta(\max(f(n), g(n)))$$

Ordenamiento por Seleccion

```
def selection_sort(collection):
  length = len(collection)
  for i in range(length):
    least = i
    for k in range(i + 1, length):
      if collection[k] < collection[least]:
        least = k
      collection[least], collection[i] = (collection[i], collection[least])
  return collection
```

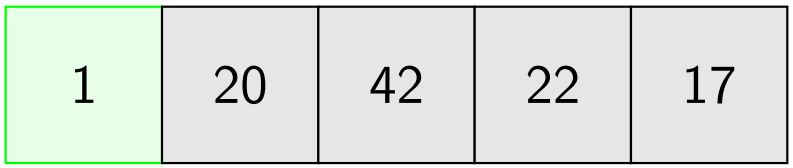
Ordenamiento por Seleccion

```
def selection_sort(collection):
  length = len(collection)
  for i in range(length):
    least = i
    for k in range(i + 1, length):
      if collection[k] < collection[least]:
        least = k
      collection[least], collection[i] = (collection[i], collection[least])
  return collection
```



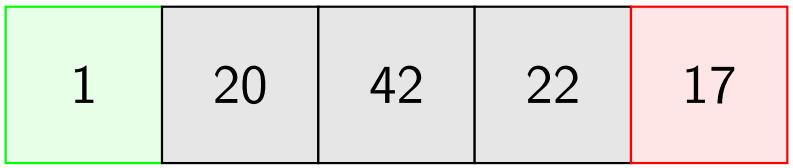
Ordenamiento por Seleccion

```
def selection_sort(collection):
  length = len(collection)
  for i in range(length):
    least = i
    for k in range(i + 1, length):
      if collection[k] < collection[least]:
        least = k
      collection[least], collection[i] = (collection[i], collection[least])
  return collection
```



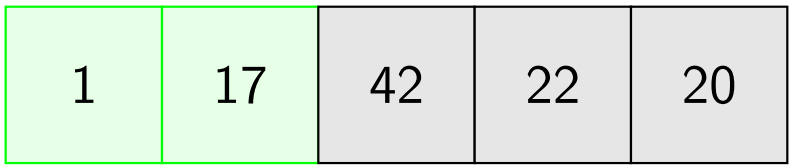
Ordenamiento por Seleccion

```
def selection_sort(collection):
  length = len(collection)
  for i in range(length):
    least = i
    for k in range(i + 1, length):
      if collection[k] < collection[least]:
        least = k
      collection[least], collection[i] = (collection[i], collection[least])
  return collection
```



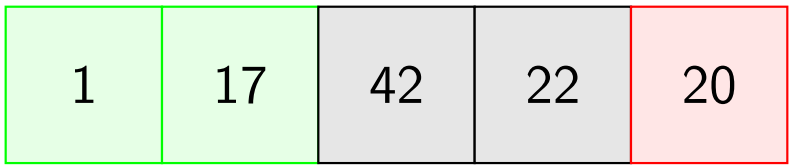
Ordenamiento por Seleccion

```
def selection_sort(collection):
  length = len(collection)
  for i in range(length):
    least = i
    for k in range(i + 1, length):
      if collection[k] < collection[least]:
        least = k
      collection[least], collection[i] = (collection[i], collection[least])
  return collection
```



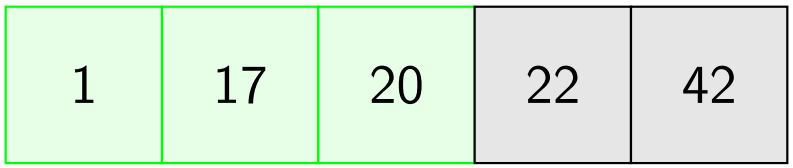
Ordenamiento por Seleccion

```
def selection_sort(collection):
  length = len(collection)
  for i in range(length):
    least = i
    for k in range(i + 1, length):
      if collection[k] < collection[least]:
        least = k
      collection[least], collection[i] = (collection[i], collection[least])
  return collection
```



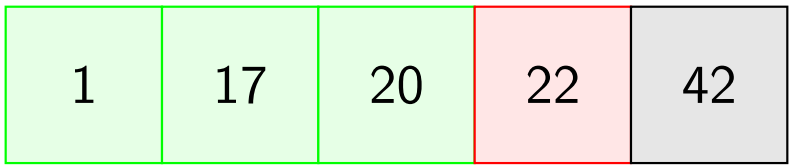
Ordenamiento por Seleccion

```
def selection_sort(collection):
  length = len(collection)
  for i in range(length):
    least = i
    for k in range(i + 1, length):
      if collection[k] < collection[least]:
        least = k
      collection[least], collection[i] = (collection[i], collection[least])
  return collection
```



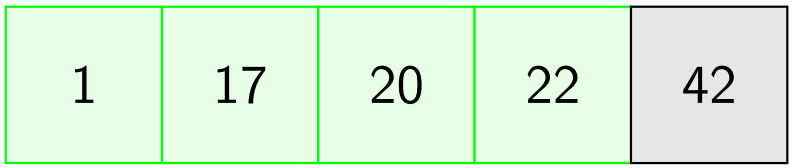
Ordenamiento por Seleccion

```
def selection_sort(collection):
  length = len(collection)
  for i in range(length):
    least = i
    for k in range(i + 1, length):
      if collection[k] < collection[least]:
        least = k
      collection[least], collection[i] = (collection[i], collection[least])
  return collection
```



Ordenamiento por Seleccion

```
def selection_sort(collection):
  length = len(collection)
  for i in range(length):
    least = i
    for k in range(i + 1, length):
      if collection[k] < collection[least]:
        least = k
      collection[least], collection[i] = (collection[i], collection[least])
  return collection
```



Ordenamiento por Seleccion

```
def selection_sort(collection):
  length = len(collection)
  for i in range(length):
    least = i
    for k in range(i + 1, length):
      if collection[k] < collection[least]:
        least = k
      collection[least], collection[i] = (collection[i], collection[least])
  return collection
```