



ANÁLISIS DE ALGORITMOS

Algoritmos y Programación
Javier Miranda

Escuela de Ingeniería Informática
Universidad de Las Palmas de Gran Canaria

15 de Noviembre de 2023



Donald Knuth

Motivación

¿ Cómo podemos medir la eficiencia de los algoritmos que hemos visto hasta ahora ?

BFS

DFS

Dijkstra

Kruskal

Prims

MergeSort

QuickSort

Knapsack 0/1 (greedy)

Knapsack 0/1 (programación dinámica)

Se denomina **ejemplar** de un problema a cada uno de los posibles casos que se pueden dar como datos iniciales del problema.



Motivación

¿ Cómo podemos medir la eficiencia de los algoritmos que hemos visto hasta ahora ?

BFS

DFS

Dijkstra

Kruskal

Prims

MergeSort

QuickSort

Knapsack 0/1 (greedy)

Knapsack 0/1 (programación dinámica)



*Estrategia
empírica*

Se denomina **ejemplar** de un problema a cada uno de los posibles casos que se pueden dar como datos iniciales del problema.



Motivación

¿ Cómo podemos medir la eficiencia de los algoritmos que hemos visto hasta ahora ?

BFS

DFS

Dijkstra

Kruskal

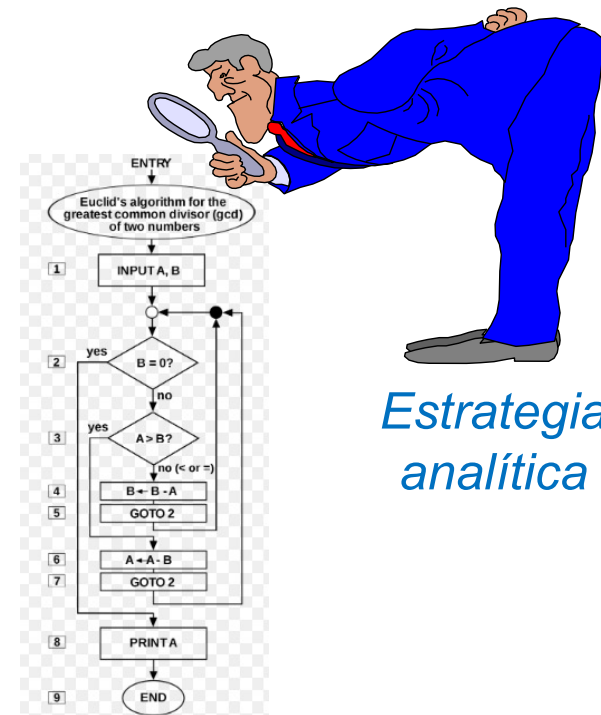
Prims

MergeSort

QuickSort

Knapsack 0/1 (greedy)

Knapsack 0/1 (programación dinámica)



Se denomina **ejemplar** de un problema a cada uno de los posibles casos que se pueden dar como datos iniciales del problema.

Motivación

¿ Qué significa este tipo de documentación ?

List

Operación	Time Complexity
Copy	$O(n)$
Append	$O(1)$
Pop	$O(1)$
Insert	$O(n)$
Remove	$O(n)$
Sort	$O(n \log n)$

Dict

Operación	Time Complexity
Copy	$O(n)$
Set	$O(1)$
Get	$O(1)$
Delete	$O(1)$

Podemos
analizar

Tiempo de ejecución (*Time Complexity*)

Memoria consumida (*Space Complexity*)



Motivación

- Necesitamos soporte para analizar algoritmos que tenga las siguientes características:
 - Permita razonar sobre la eficiencia de los algoritmos
 - Facilite la comparación de algoritmos, especialmente al procesar grandes ejemplares
 - Evite detalles dependientes de la arquitectura, el lenguaje o el compilador

Motivación

- Necesitamos soporte para analizar algoritmos que tenga las siguientes características:
 - Permita razonar sobre la eficiencia de los algoritmos
 - Facilite la comparación de algoritmos, especialmente al procesar grandes ejemplares
 - Evite detalles dependientes de la arquitectura, el lenguaje o el compilador
- La idea general es suprimir:
 - Constantes (demasiado dependientes del Sistema)
 - Terminos de menor grado (irrelevantes para ejemplares grandes)

Ejemplos: $10n^2 + 3n + 2 \rightarrow \text{Orden } (n^2)$

$6n \log_2 n + 6 \rightarrow \text{Orden } (n \log n)$

Cota Superior: Notación O

Sea $f : N \rightarrow R^*$ una función cualquiera y $R^* = R^+ \cup \{0\}$
 $O(f(n)) = \left\{ t : N \rightarrow R^* \mid \left(\exists c \in R^+ \right) \left(\exists n_0 \in N \right) \left(\forall n \geq n_0 \right) \left(t(n) \leq cf(n) \right) \right\}$

- **$O(f(n))$** se lee del orden de $f(n)$

Cota Superior: Notación O

Sea $f : N \rightarrow R^*$ una función cualquiera y $R^* = R^+ \cup \{0\}$

$$O(f(n)) = \left\{ t : N \rightarrow R^* \mid \left(\exists c \in R^+ \right) \left(\exists n_0 \in N \right) \left(\forall n \geq n_0 \right) \left(t(n) \leq cf(n) \right) \right\}$$

- Es el conjunto de todas las funciones $t(n)$ no negativas, ...

Cota Superior: Notación O

Sea $f : N \rightarrow R^*$ una función cualquiera y $R^* = R^+ \cup \{0\}$

$$O(f(n)) = \left\{ t : N \rightarrow R^* \left(\left(\exists c \in R^+ \right) \left(\exists n_0 \in N \right) \left(\forall n \geq n_0 \right) \left(t(n) \leq cf(n) \right) \right) \right\}$$

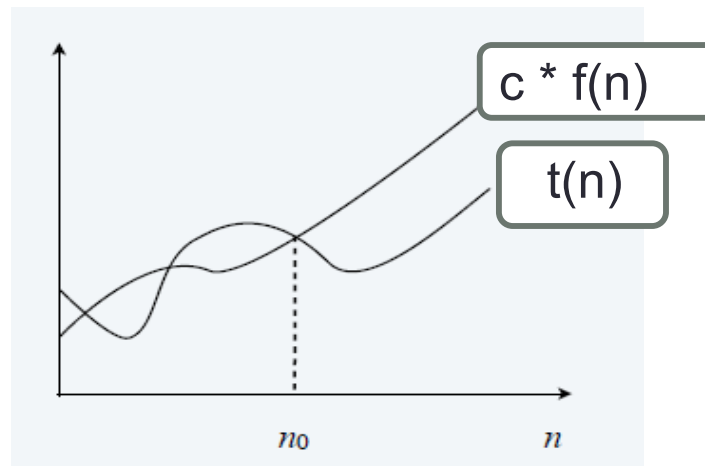
- Es el conjunto de todas las funciones $t(n)$ no negativas, acotadas superiormente por un múltiplo real positivo de $f(n)$

Cota Superior: Notación O

$$\text{Sea } f : N \rightarrow R^* \text{ una función cualquiera y } R^* = R^+ \cup \{0\}$$

$$O(f(n)) = \left\{ t : N \rightarrow R^* \mid \left(\exists c \in R^+ \right) \left(\exists n_0 \in N \right) \left(\forall n \geq n_0 \right) \left(t(n) \leq cf(n) \right) \right\}$$

- Es el conjunto de todas las funciones $t(n)$ no negativas, acotadas superiormente por un múltiplo real positivo de $f(n)$ para valores de n suficientemente grandes (es decir, a partir de un cierto umbral n_0 en adelante).



Cota Superior: Notación O

Sea $f : N \rightarrow R^*$ una función cualquiera y $R^* = R^+ \cup \{0\}$

$$O(f(n)) = \left\{ t : N \rightarrow R^* \mid \left(\exists c \in R^+ \right) \left(\exists n_0 \in N \right) \left(\forall n \geq n_0 \right) \left(t(n) \leq cf(n) \right) \right\}$$

- Es el conjunto de todas las funciones $t(n)$ no negativas, acotadas superiormente por un múltiplo real positivo de $f(n)$ para valores de n suficientemente grandes (es decir, a partir de un cierto umbral n_0 en adelante).

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

Cota Superior: Notación O

Sea $f : N \rightarrow R^*$ una función cualquiera y $R^* = R^+ \cup \{0\}$
 $O(f(n)) = \left\{ t : N \rightarrow R^* \mid \left(\exists c \in R^+ \right) \left(\exists n_0 \in N \right) \left(\forall n \geq n_0 \right) \left(t(n) \leq cf(n) \right) \right\}$

Ejemplo: $t(n) = 3n + 12$

¿ Cual es la cota superior más cercana ?



Cota Superior: Notación O

Sea $f : N \rightarrow R^*$ una función cualquiera y $R^* = R^+ \cup \{0\}$

$$O(f(n)) = \left\{ t : N \rightarrow R^* \mid \left(\exists c \in R^+ \right) \left(\exists n_0 \in N \right) \left(\forall n \geq n_0 \right) \left(t(n) \leq cf(n) \right) \right\}$$

Ejemplo: $t(n) = 3n + 12$

... eligiendo $f(n) = n$
 $c = 4$

Por tanto $t(n) \in O(n)$

$n_0 \rightarrow$

	$t(n)$	$4 \cdot f(n)$
N	$3n+12$	$4n$
1	15	4
2	18	8
3	21	12
...
...
10	42	40
11	45	44
$n_0 \rightarrow 12$	48	48
13	51	52
...

Cota Superior: Notación O

Sea $f : N \rightarrow R^*$ una función cualquiera y $R^* = R^+ \cup \{0\}$
 $O(f(n)) = \left\{ t : N \rightarrow R^* \mid \left(\exists c \in R^+ \right) \left(\exists n_0 \in N \right) \left(\forall n \geq n_0 \right) \left(t(n) \leq cf(n) \right) \right\}$

Ejemplo: $t(n) = 3n + 12$

... eligiendo $f(n) = n$
 $c = 4$

Por tanto $t(n) \in O(n)$

En general, dado un polinomio con coeficiente de mayor grado positivo, en análisis asintótico nos quedamos con el término de mayor exponente:

$$t(n) = a_m n^m + \dots + a_1 n + a_0 \quad t(n) \in O(n^m) \text{ si } a_m > 0$$

Cota Superior: Notación O

Sea $f : N \rightarrow R^*$ una función cualquiera y $R^* = R^+ \cup \{0\}$
 $O(f(n)) = \left\{ t : N \rightarrow R^* \mid \left(\exists c \in R^+ \right) \left(\exists n_0 \in N \right) \left(\forall n \geq n_0 \right) \left(t(n) \leq cf(n) \right) \right\}$

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



Cota Superior: Notación O

Sea $f : N \rightarrow R^*$ una función cualquiera y $R^* = R^+ \cup \{0\}$
 $O(f(n)) = \left\{ t : N \rightarrow R^* \mid \left(\exists c \in R^+ \right) \left(\exists n_0 \in N \right) \left(\forall n \geq n_0 \right) \left(t(n) \leq cf(n) \right) \right\}$

Principio de invariancia

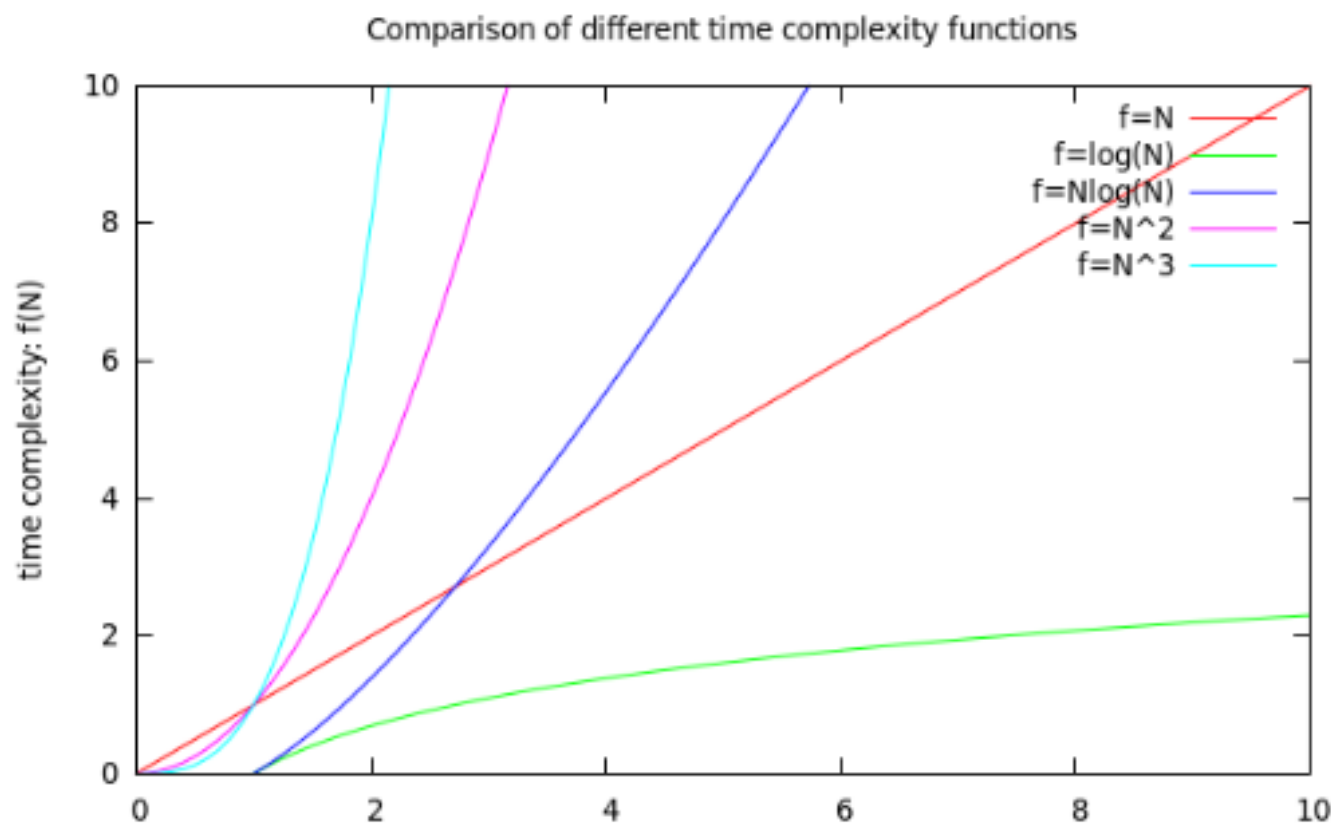
- Dadas dos implementaciones del mismo algoritmo con tiempos $t_1(n)$ y $t_2(n)$, para valores de n suficientemente grandes se cumple que: $t_1(n) \leq c * t_2(n)$



$$t_1(n) \in O(t_2(n))$$

Cota Superior: Notación O

Sea $f : N \rightarrow R^*$ una función cualquiera y $R^* = R^+ \cup \{0\}$

$$O(f(n)) = \left\{ t : N \rightarrow R^* \mid \left(\exists c \in R^+ \right) \left(\exists n_0 \in N \right) \left(\forall n \geq n_0 \right) \left(t(n) \leq cf(n) \right) \right\}$$


Exponencial
Cúbico
Cuadrático
Lineal
Logarítmico

Análisis Asintótico de un algoritmo

- 1. Definir N
- 2. Casos de estudio
- 3. Aplicar las reglas generales de análisis asintótico

1) ¿ Cómo definimos N ?

1. → Definir N
2. Casos de estudio
3. Reglas de análisis

```
def Fib(n) {  
    if (n < 2)  
        return n  
    else  
        return Fib(n-2) + Fib(n-1)  
}
```

a) Algoritmos
numéricos

```
# Traverse through all array elements  
for i in range(len(A)):  
  
    # Find the minimum element in remaining  
    # unsorted array  
    min_idx = i  
    for j in range(i+1, len(A)):  
        if A[min_idx] > A[j]:  
            min_idx = j  
  
    # Swap the found minimum element with  
    # the first element  
    A[i], A[min_idx] = A[min_idx], A[i]
```

b) Algoritmos que recorren
estructuras de datos



1) ¿ Cómo definimos N ?

1. → Definir N
2. Casos de estudio
3. Reglas de análisis

*Definición formal: Número de **bits** necesarios para codificar el ejemplar*

```
def Fib(n) {
    if (n < 2)
        return n
    else
        return Fib(n-2) + Fib(n-1)
}
```

```
# Traverse through all array elements
for i in range(len(A)):

    # Find the minimum element in remaining
    # unsorted array
    min_idx = i
    for j in range(i+1, len(A)):
        if A[min_idx] > A[j]:
            min_idx = j

    # Swap the found minimum element with
    # the first element
    A[i], A[min_idx] = A[min_idx], A[i]
```

a) Algoritmos
numéricos

Valor máximo

b) Algoritmos que recorren
estructuras de datos

Número de elementos



1. Definir N
2. → Casos de estudio
3. Reglas de análisis

2) Casos de estudio

- Se denomina **ejemplar** de un problema a cada uno de los posibles casos que se pueden dar como datos iniciales del problema (por ejemplo, diferentes tamaños)
- ... pero si el algoritmo tiene distinto comportamiento con ejemplares del **mismo tamaño** analizamos también:
 - El **mejor caso**
 - El **peor caso**
 - El **caso promedio**

Por ejemplo, para analizar QuickSort

... y también puede interesarnos analizar un determinado tipo de operación.

Ejemplo: En algoritmos de ordenación:

- Comparaciones
- Intercambios

3) Reglas Generales para el Análisis Asintótico

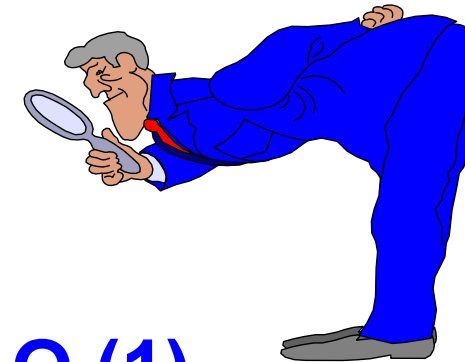
- El Orden de una operación elemental es 1 (por definición).
- El Orden de una secuencia de operaciones se calcula aplicando la **regla de la suma**.

Para cualesquiera dos funciones f y $g : N \rightarrow R^$*

$$O(f(n) + g(n)) = O(\max\{f(n), g(n)\})$$

Ejemplo: Intercambio

```
def swap (my_list, pos1, pos2):
O (1) → tmp = my_list[pos1]
O (1) → my_list[pos1] = my_list[pos2]
O (1) → my_list[pos2] = tmp
O (1) → return
```



O (1)

Python List

Operación	Time Complexity
Copy	O(n)
Append	O(1)
Get Item	O(1)
Set Item	O(1)
...	...
...	...

Ejemplo de cálculo del coste mediante notación asintótica

3) Reglas Generales para el Análisis Asintótico

- El Orden de una operación elemental es 1 (por definición)
- El Orden de una secuencia de operaciones se calcula aplicando la **regla de la suma**

$$\text{Para cualesquiera dos funciones } f \text{ y } g : N \rightarrow R^* \\ O(f(n) + g(n)) = O(\max\{f(n), g(n)\})$$

- El Orden de una sentencia condicional es igual al **máximo** del Orden de cada alternativa
- El Orden de un bucle es igual al Orden de la suma de sus iteraciones
- El Orden de una llamada a un subprograma es igual al Orden del subprograma llamado