ANÁLISIS DE ALGORITMOS

Algoritmos y Programación

Javier Miranda

Escuela de Ingeniería Informática
Universidad de Las Palmas de Gran Canaria

22 de Noviembre de 2023

REPASO

Análisis Asintótico de un algoritmo

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

REPASO

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

1) ¿ Cómo definimos N ?

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

REPASO

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

REPASO

Reglas Generales para el Análisis Asintótico de algoritmos iterativos

- 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**

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

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

$$t(n) = a_m n^m + \ldots + a_1 n + a_0$$

$t(n) \in O(n^m)$ si $a_m > 0$

REPASO

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**

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

- 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

Ejemplo: Último Máximo

```
def search_last_max (v, n):
    max = v[0]
    pos = 0;
    i = 1

    while i < n:
        if v[i] >= max:
            max = v[i]
            pos = i
        i = i + 1

    return max, pos
```

Ejemplo: Último Máximo

```
def search_last_max (v, n):
    max = v[0]                    O(1)
    pos = 0;                      O(1)
    i = 1                         O(1)

    while i < n:
        if v[i] >= max:
            max = v[i]            O(1)
            pos = i               O(1)
        i = i + 1                 O(1)

    return max, pos               O(1)
```

Ejemplo: Último Máximo

```
def search_last_max (v, n):
    max = v[0]
    pos = 0;
    i = 1

    while i < n:
        if v[i] >= max:
            max = v[i]
            pos = i
        i = i + 1

    return max, pos
```

O(1)

O(1)

O(1)

O(1)

Ejemplo: Último Máximo

```
def search_last_max (v, n):
    max = v[0]
    pos = 0;                          <------------------  O(1)
    i = 1

    while i < n:
        if v[i] >= max:               <------------------  O(1)
            max = v[i]
            pos = i                   <------------------  O(1)
        i = i + 1                     <------------------  O(1)

    return max, pos                   <------------------  O(1)
```

Ejemplo: Último Máximo

```
def search_last_max (v, n):
    max = v[0]
    pos = 0;
    i = 1

    while i < n:
        if v[i] >= max:
            max = v[i]
            pos = i
        i = i + 1

    return max, pos
```

O(1)

O(1)

O(1)

O(1)

Ejemplo: Último Máximo

```
def search_last_max (v, n):
    max = v[0]
    pos = 0;
    i = 1

    while i < n:
        if v[i] >= max:
            max = v[i]
            pos = i
        i = i + 1

    return max, pos
```

O(1)

O(1)

O(1)

Ejemplo: Último Máximo

```
def search_last_max (v, n):
    max = v[0]
    pos = 0;                          O(1)
    i = 1

    while i < n:                      O(n)
        if v[i] >= max:               O(1)
            max = v[i]
            pos = i
        i = i + 1

    return max, pos                   O(1)
```

Ejemplo: Último Máximo

```
def search_last_max (v, n):
    max = v[0]
    pos = 0;
    i = 1

    while i < n:
        if v[i] >= max:
            max = v[i]
            pos = i
        i = i + 1

    return max, pos
```

O(1)

O(n)

O(1)

Ejemplo: Último Máximo

```
def search_last_max (v, n):
    max = v[0]
    pos = 0;
    i = 1

    while i < n:
        if v[i] >= max:
            max = v[i]
            pos = i
        i = i + 1

    return max, pos
```

O(1)

O(n)

O(1)

$\in$ **O(n)**

Ejemplo: Producto número por Matriz

```
def product_number_square_matrix (k, matrix, n):
    for i in range(n-1):
        matrix[i, i] = k * matrix[i, i]

        for j in range(i+1,n):
            matrix[i, j] = k * matrix[i, j]
            matrix[j, i] = k * matrix[j, i]

    matrix[n-1,n-1] = k * matrix[n-1,n-1]
    return
```

Ejemplo: Producto número por Matriz

```
def product_number_square_matrix (k, matrix, n):
    for i in range(n-1):
        matrix[i, i] = k * matrix[i, i]                 O(1)

        for j in range(i+1,n):
            matrix[i, j] = k * matrix[i, j]             O(1)
            matrix[j, i] = k * matrix[j, i]             O(1)

    matrix[n-1,n-1] = k * matrix[n-1,n-1]               O(1)
    return                                              O(1)
```

Ejemplo: Producto número por Matriz

```
def product_number_square_matrix (k, matrix, n):
    for i in range(n-1):
        matrix[i, i] = k * matrix[i, i]                O(1)

        for j in range(i+1,n):
            matrix[i, j] = k * matrix[i, j]            O(1)
            matrix[j, i] = k * matrix[j, i]

    matrix[n-1,n-1] = k * matrix[n-1,n-1]              O(1)
    return
```

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**

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

- 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

Análisis de bucles dependientes

- **Ejemplo**

```
for i in range(1,n-1):
    ... # O(1)

    for j in range(i+1,n):
        ...  # O(1)
```

El número de iteraciones del bucle interno depende del bucle externo.

$$\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}1=\sum_{i=1}^{n-1}(n-i)=(n-1)+(n-2)+\cdots+1=$$

$$=\frac{(n-1+1)(n-1)}{2}=\frac{n^2-n}{2}\leq\frac{n^2}{2}\in O\left(n^2\right)$$

Ejemplo: Producto número por Matriz

```
def product_number_square_matrix (k, matrix, n):
  for i in range(n-1):                              O(n^2)
      matrix[i, i] = k * matrix[i, i]

      for j in range(i+1,n):
          matrix[i, j] = k * matrix[i, j]
          matrix[j, i] = k * matrix[j, i]

  matrix[n-1,n-1] = k * matrix[n-1,n-1]             O(1)
  return
```

$\in$ **$O(n^2)$**

Análisis de Bucles: ¡Cuidado con el paso!

Hemos dicho que:

> El Orden de un bucle es igual al orden de la suma de sus iteraciones.

¿ Cual es el coste de ejecución de este bucle ?

```
int i = n;
while (i > 0) {
    ...            // O(1)
    i = i / 2;
}
```

$\in$ **O (log N)**

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

¿ Podemos simplificar un poco más ? Si

- No es necesario calcular el Orden de todas las operaciones.
- Es suficiente con determinar cuál es la operación elemental que se ejecuta el mayor número de veces (**operación crítica**)

Identificamos la **operación crítica** de nuestro algoritmo
…. y calculamos su coste de ejecución

Análisis Asintótico Abreviado

Último máximo: Análisis Abreviado

```
def search_last_max (v, n):
    max = v[0]
    pos = 0;
    i = 1

    while i < n:
        if v[i] >= max:
            max = v[i]
            pos = I
        i = i + 1

    return max, pos
```

O(n)

Operación crítica

$\in$ **O(n)**

Producto de Matrices: A. Abreviado

```
def product_square_matrixes (A, B, C, n):
    for i in range(n):                                   O(n³)
        for j in range(n):                               O(n²)
            C[i, j] = 0

            for k in range(n):                           O(n)
                C[i, j] = C[i, j] + A[i, k] * B[k, j]

    return
```

Operación crítica

$\in O(n^3)$

Análisis de algoritmos de ordenación iterativos

- Ordenación por selección
- Ordenación por inserción
- Burbuja

Visualización de algoritmos de ordenación

https://visualgo.net/en/sorting
http://www.sorting-algorithms.com/
http://www.cs.usfca.edu/~galles/visualization/ComparisonSort.html

https://www.youtube.com/watch?v=kPRA0W1kECg

Ordenación por Selección

- Mientras queden elementos en la lista de entrada:
  - Buscamos el menor elemento
  - Lo extraemos de la lista de entrada y lo añadimos a la lista de salida

i	Input List					Output List				
0	D	B	A	E	C	A				
1	D	B	E	C		A	B			
2	D	E	C			A	B	C		
3	D	E				A	B	C	D	
4	E					A	B	C	D	E

Ordenación por Selección (versión creando una nueva lista)

```
def selection_sort_new_list(A):
    out_list=[]
    aux_list=A.copy()   # Why is the list copied here?

    while len(aux_list) > 0:
        # Find the minimum of the input list
        pos_minimum = 0
        for j in range(1, len(aux_list)):
            if aux_list[j] < aux_list[pos_minimum]:
                pos_minimum = j

        # Remove the minimum from the list
        # ... and append it to the new list
        out_list.append(aux_list.pop(pos_minimum))

    return out_list
```

Ordenación por Selección (versión in place)

```
def selection_sort_in_place(A):
    for i in range(len(A)):
        # Find the minimum of the input list
        pos_minimum = i
        for j in range(i+1, len(A)):
            if A[j] < A[pos_minimum]:
                pos_minimum = j

        # Swap the found minimum item with item i
        A[i], A[pos_minimum] = A[pos_minimum], A[i]
    return
```

https://visualgo.net/en/sorting

In Place: La función ordena directamente la lista de entrada.

Ordenación por Selección (versión in place)

Análisis Asintótico del número de comparaciones

```
def selection_sort_in_place(A):
    for i in range(len(A)):
        # Find the minimum of the input list
        pos_minimum = i
        for j in range(i+1, len(A)):
            if A[j] < A[pos_minimum]:
                pos_minimum = j

        # Swap the found minimum item with item i
        A[i], A[pos_minimum] = A[pos_minimum], A[i]
    return
```

Operación crítica

$\in O(n^2)$

Ordenación por Selección (versión in place)

Análisis Asintótico del coste de movimiento de datos

```
def selection_sort_in_place(A):
    for i in range(len(A)):
        # Find the minimum of the input list
        pos_minimum = i
        for j in range(i+1, len(A)):
            if A[j] < A[pos_minimum]:
                pos_minimum = j

        # Swap the found minimum item with item i
        A[i], A[pos_minimum] = A[pos_minimum], A[i]
    return
```

Operación crítica

$\in$ **O(n)**

Ordenación por Inserción

- Recorremos los elementos de la lista de entrada
  - … y los insertamos en su posición correcta en la lista de salida

i	Input List					Output List				
0	D	B	A	E	C	D				
1		B	A	E	C	B	D			
2			A	E	C	A	B	D		
3				E	C	A	B	D	E	
4					C	A	B	C	D	E

Ordenación por Inserción (in place)

```
def insertion_sort_in_place(A):
    for i in range(1, len(A)):
        v = A[i]    # remember the value to be inserted

        # Go backward in list and shift to the right
        # if element > value to be inserted.
        j = i
        while j >= 1 and A[j-1] > v:
            A[j] = A[j-1]
            j -= 1

        # Insert the value at its correct position
        A[j] = v
    return
```

https://visualgo.net/en/sorting

Ordenación por Inserción

Análisis Asintótico del número de comparaciones

```
def insertion_sort_in_place(A):
    for i in range(1, len(A)):
        v = A[i]    # remember the value to be inserted

        # Go backward in list and shift to the right
        # if element > value to be inserted.
        j = i
        while j >= 1 and A[j-1] > v:
            A[j] = A[j-1]
            j -= 1

        # Insert the value at its correct position
        A[j] = v
    return
```

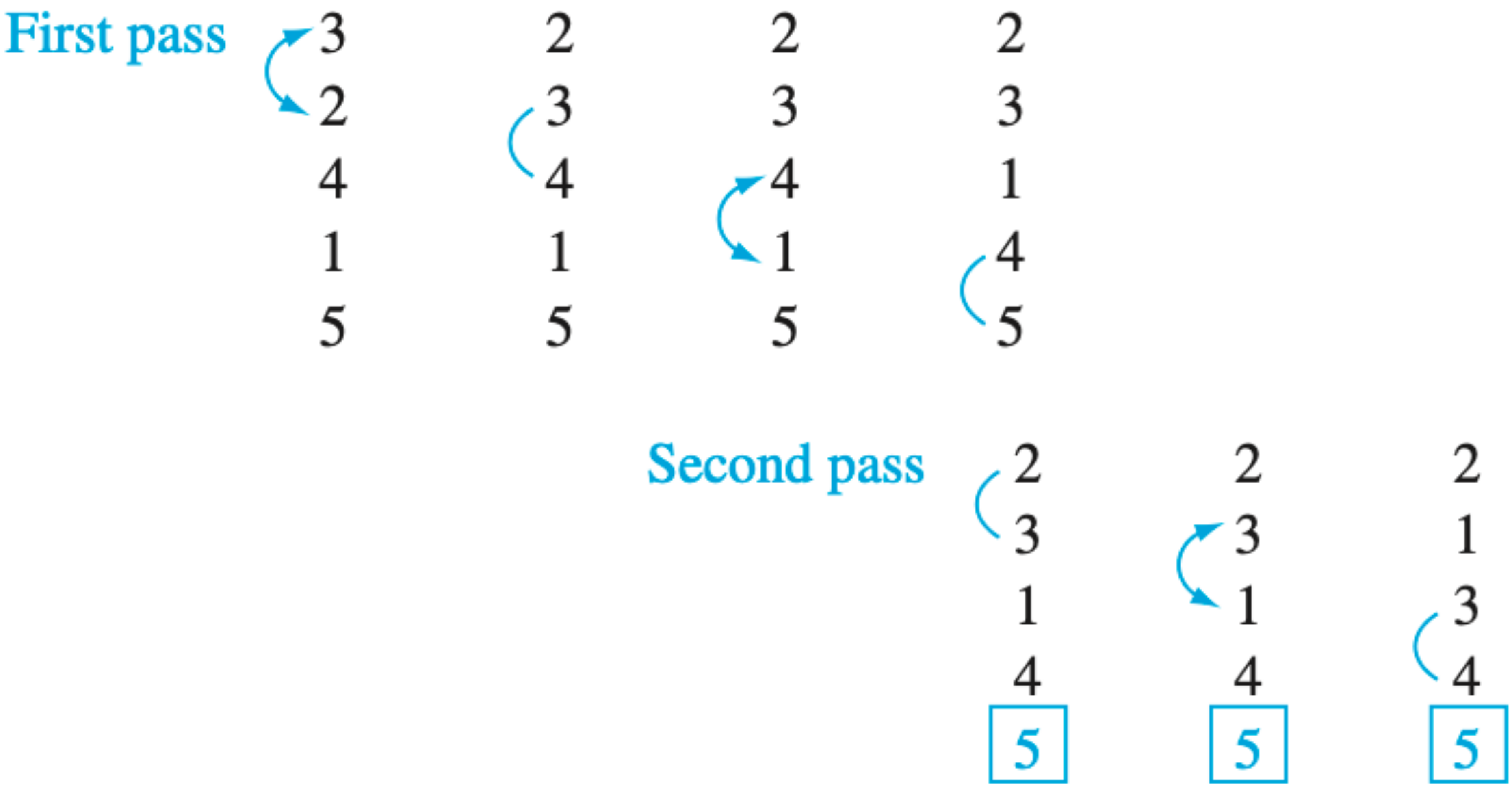
Operación crítica

Comparaciones

Mejor Caso: $\in$ **O(n)**

Peor Caso: $\in$ **O(n^2)**

Ordenación por Burbuja



https://visualgo.net/en/sorting

Ordenación por Burbuja

Análisis Asintótico del número de comparaciones

```
n = len(arr)

# Traverse through all array elements
for i in range(n):

    # Last i elements are already in place
    for j in range(0, n-i-1):

        # traverse the array from 0 to n-i-1
        # Swap if the element found is greater
        # than the next element
        if arr[j] > arr[j+1] :
            arr[j], arr[j+1] = arr[j+1], arr[j]
```

Operación crítica

https://www.geeksforgeeks.org/bubble-sort/

Comparaciones

Mejor Caso: $\in$ **$O(n^2)$**

Peor Caso: $\in$ **$O(n^2)$**

Ordenación por Burbuja (v2)

Análisis Asintótico del número de comparaciones

```
n = len(arr)

# Traverse through all array elements
for i in range(n):
    swapped = False

    # Last i elements are already
    #  in place
    for j in range(0, n-i-1):

        # traverse the array from 0 to
        # n-i-1. Swap if the element
        # found is greater than the
        # next element
        if arr[j] > arr[j+1] :
            arr[j], arr[j+1] = arr[j+1], arr[j]
            swapped = True

    # IF no two elements were swapped
    # by inner loop, then break
    if swapped == False:
        break
```

Versión optimizada

Operación crítica

Comparaciones

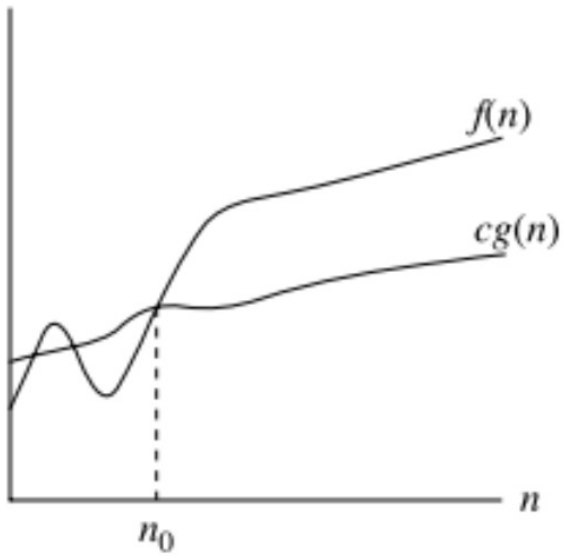
Mejor Caso: $\in$ **O(n)**

Peor Caso: $\in$ **O(n²)**

Otras notaciones para análisis de algoritmos: cota inferior

Ω-notation

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



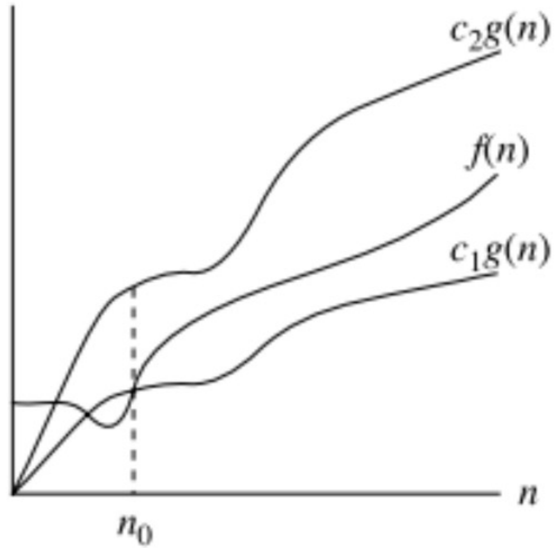
$g(n)$ is an ***asymptotic lower bound*** for $f(n)$.

Donald Knuth

Otras notaciones para análisis de algoritmos: cota ajustada

Θ-notation

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



$g(n)$ is an ***asymptotically tight bound*** for $f(n)$.

Donald Knuth

¿ Cómo analizamos algoritmos recursivos ?

Lo veremos en nuestra siguiente clase