Complejidad algorítmica

Clase práctica

Algoritmos y Estructuras de Datos

Departamento de Computación

DEPARTAMENTO DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

1^er cuatrimestre 2025

Menú del día

Cotas de complejidad - $\mathcal{O}, \Omega, \Theta$

Definición

Sea $g : \mathbb{N} \to \mathbb{R}$. Entonces:

$$\mathcal{O}(g) \stackrel{\text{def}}{=} \{f : \mathbb{N} \to \mathbb{R} \mid (\exists\ n_0 \in \mathbb{N},\ c \in \mathbb{R}_{>0})\ \ f(n) \leq c \cdot g(n)\ \ \forall n \geq n_0\}$$

Definición

Sea $g : \mathbb{N} \to \mathbb{R}$. Entonces:

$$\Omega(g) \stackrel{\text{def}}{=} \{f : \mathbb{N} \to \mathbb{R} \mid (\exists\ n_0 \in \mathbb{N},\ c \in \mathbb{R}_{>0})\ \ f(n) \geq c \cdot g(n)\ \ \forall n \geq n_0\}$$

Definición

Sea $g : \mathbb{N} \to \mathbb{R}$. Entonces:

$$\Theta(g) \stackrel{\text{def}}{=} \{f : \mathbb{N} \to \mathbb{R} \ \mid\ f \in O(g) \land f \in \Omega(g)\}$$

Ejercicio

Demostrar que $n^2 + 5n + 3 \in \Omega(n)$, pero $n^2 + 5n + 3 \notin \mathcal{O}(n)$

Ejercicio

$$n^2 + 5n + 3 \in \Omega(n)$$

Queremos ver que:

$$\exists c \in \mathbb{R}_{>0}, \exists n_0 \in \mathbb{N} \text{ tal que: } n^2 + 5n + 3 \geq cn, \quad \forall n \geq n_0$$

Dividiendo ambos lados por n (con $n > 0$):

$$\frac{n^2 + 5n + 3}{n} \geq c$$

$$n + 5 + \frac{3}{n} \geq c, \quad \forall n \geq n_0$$

n + 5 + $\frac{3}{n}$ es una función creciente, con $c = 1$ y $n \geq 1$ se cumple.
$\implies n^2 + 5n + 3 \in \Omega(n)$

Ejercicio

$$n^2 + 5n + 3 \notin \mathcal{O}(n)$$

Supongamos que:

$$n^2 + 5n + 3 \in \mathcal{O}(n) \Rightarrow \exists c > 0, \exists n_0 \in \mathbb{N} \text{ tal que:}$$

$$n^2 + 5n + 3 \leq cn, \quad \forall n \geq n_0$$

Dividiendo ambos lados por n:

$$n + 5 + \frac{3}{n} \leq c, \quad \forall n \geq n_0$$

Pero $n + 5 + \frac{3}{n} \to \infty$ cuando $n \to \infty$, no puede existir c constante que lo cumpla

$$\Rightarrow n^2 + 5n + 3 \notin \mathcal{O}(n)$$

Propiedades – $\Diamond$ es "comodín" de $\mathcal{O}, \Omega, \Theta$

1. Toda f cumple $f \in \Diamond(f)$. Reflexiva
2. $f \in \Diamond(g) \implies k \cdot f \in \Diamond(g)$ (con k cte.)
3. Regla de la suma:
   $f_1 \in \Diamond(g) \wedge f_2 \in \Diamond(h) \implies f_1 + f_2 \in \Diamond(g+h) = \Diamond(\max\{g, h\})$
4. Regla del producto:
   $f_1 \in \Diamond(g) \wedge f_2 \in \Diamond(h) \implies f_1 \cdot f_2 \in \Diamond(g \cdot h)$

3 y 4 corresponden al **álgebra de órdenes**. Además 4 implica 2.

- $f \in \Diamond(g) \wedge g \in \Diamond(h) \implies f \in \Diamond(h)$ Transitiva
- $f \in \Diamond(g) \implies \Diamond(f) \subseteq \Diamond(g)$
- $\Diamond(f) = \Diamond(g) \iff f \in \Diamond(g) \wedge g \in \Diamond(f)$

  Como $f \in \Theta(g) \implies g \in \Theta(f)$ Simétrica
  - $\Theta(f) = \Theta(g) \iff f \in \Theta(g)$

Ejercicio

Demostrar que $n^2 + 5n + 3 \in \Omega(n)$, pero $n^2 + 5n + 3 \notin \mathcal{O}(n)$

Ejemplo con Propiedades

$$n^2 + 5n + 3 \in \Omega(n),$$

Usamos propiedades:

- Sea $f(n) = n^2$, $g(n) = 5n$, $h(n) = 3$.
- Por **reflexividad**, $f \in \Omega(f)$, $g \in \Omega(g)$ y $h \in \Omega(h)$
- Por **propiedades de las constantes**, $5n \in \Omega(n)$ y $3 \in \Omega(1)$
- Por la **regla de la suma**:

$$n^2 + n + 1 \in \Omega(n^2 + n + 1) = \Omega(\text{máx}\{n^2, n, 1\}) = \Omega(n^2)$$

Como $\Omega(n^2) \subseteq \Omega(n)$ entonces $n^2 + 5n + 3 \in \Omega(n)$

Ejemplo con Propiedades

$$n^2 + 5n + 3 \notin \mathcal{O}(n),$$

Usamos propiedades:

- Sea $f(n) = n^2$, $g(n) = 5n$, $h(n) = 3$.
- Por la **regla de la suma**:

$$n^2 + n + 1 \in O(n^2 + n + 1) = O(\text{máx}\{n^2, n, 1\}) = O(n^2)$$

La regla de la suma nos da la cota superior (o inferior) más ajustada. Como $\mathcal{O}(n^2) \subsetneq \mathcal{O}(n)$, entonces $n^2 + 5n + 3 \notin \mathcal{O}(n)$.

Ejercicio

Demostrar que:

- $\mathcal{O}(n^2) \cap \Omega(1) \neq \Theta(n)$

Ejercicio

$$\mathcal{O}(n^2) \cap \Omega(1) \neq \Theta(n)$$

Vamos a mostrar que existe al menos una función en la intersección

$$\mathcal{O}(n^2) \cap \Omega(1)$$

que no pertenece a $\Theta(n)$.

- $\mathcal{O}(n^2)$: funciones que no crecen más rápido que n^2
- $\Omega(1)$: funciones que están acotadas inferiormente por la función constante
- $\Theta(n)$: funciones que crecen linealmente

Ejemplo: Con $f(n) = n^2$

- $f \in \mathcal{O}(n^2)$ y $f \in \Omega(1)$,
- $f \notin \Theta(n)$
- por lo que $f \in \mathcal{O}(n^2) \cap \Omega(1)$ pero $f \notin \Theta(n)$. Los conjuntos no son iguales

Ejercicios

Decidir si son verdaderas o falsas y justificar:

1. $2^n \in \mathcal{O}(1)$
2. $2^n \in O(n!)$
3. $\Omega(n) \subseteq \mathcal{O}(n^2)$
4. $\mathcal{O}(n) \subseteq \Omega(n)$

$$2^n \in \mathcal{O}(1)$$

FALSO. Supongamos que:

$$2^n \in \mathcal{O}(1) \Rightarrow \exists c > 0, \exists n_0 \in \mathbb{N} \text{ tal que:}$$

$$2^n \leq c1, \quad \forall n \geq n_0$$

Pero $2^n \to \infty$ cuando $n \to \infty$, no puede existir c constante que lo cumpla

$$\Rightarrow 2^n \notin \mathcal{O}(1)$$

$$2^n \in \mathcal{O}(n!)$$

VERDADERO
Queremos ver que:

$$\exists c \in \mathbb{R}_{>0}, \exists n_0 \in \mathbb{N} \text{ tal que: } 2^n \leq cn!, \quad \forall n \geq n_0$$

$2^n = \prod_{i=1}^{n} 2$ y $n! = \prod_{i=1}^{n} i$, hay n términos de cada lado

$$2 * 2 * 2 * 2 * ..... * 2 * 2 \leq c * (1 * 2 * 3 * 4 * ... * n - 1 * n), \quad \forall n \geq n_0$$

Todos los términos de $n!$ son mayor o iguales a los de 2^n excepto por el primero.
Con $c = 2$,

$$2 * 2 * 2 * 2 * ..... * 2 * 2 \leq \mathbf{2} * 2 * 3 * 4 * ... * n - 1 * n), \quad \forall n \geq 1$$

Con $c \geq 2$ y $n \geq 1 \implies 2^n \in \mathcal{O}(n!)$

$$\Omega(n) \subseteq \mathcal{O}(n^2)$$

FALSO. Damos un contraejemplo, necesitamos una función que esté en $Omega(n)$ pero no en $\mathcal{O}(n^2)$

- $\mathcal{O}(n^2)$: funciones que no crecen más rápido que n^2
- $\Omega(n)$: funciones que crecen igual o más rápido que n

Ejemplo: Con $f(n) = n^5$

- $f \in \Omega(n)$
- $f \notin \mathcal{O}(n^2)$
- por lo que $f \in \Omega(n^2)$ pero $f \notin \mathcal{O}(n^2)$. Por lo tanto, $\Omega(n) \nsubseteq \mathcal{O}(n^2)$

$$\mathcal{O}(n) \subseteq \Omega(n)$$

FALSO. Damos un contraejemplo, necesitamos una función que esté en $\mathcal{O}(n)$ pero no en $\Omega(n)$

- $\mathcal{O}(n)$: funciones que no crecen más rápido que n
- $\Omega(n)$: funciones que crecen igual o más rápido que n

Ejemplo: Con $f(n) = 1$

- $f \in \mathcal{O}(n)$
- $f \notin \Omega(n)$
- por lo que $f \in \Omega(n)$ pero $f \notin \mathcal{O}(n)$. Por lo tanto, $\mathcal{O}(n) \nsubseteq \Omega(n)$

Propiedad

Dadas las funciones $f,\ g : \mathbb{N} \to \mathbb{R}$. Si existe

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \ell \in \mathbb{R}_{\geq 0} \cup \{+\infty\}$$

Entonces:

- $f \in \Theta(g) \iff 0 < \ell < +\infty$
- $f \in \mathcal{O}(g)$ y $f \notin \Omega(g) \iff \ell = 0$
- $f \in \Omega(g)$ y $f \notin \mathcal{O}(g) \iff \ell = +\infty$

Ejercicio

Demostrar que $n^2 + 5n + 3 \in \Omega(n)$, pero $n^2 + 5n + 3 \notin \mathcal{O}(n)$

Ejercicio

$$n^2 + 5n + 3 \in \Omega(n), \quad \text{pero} \quad n^2 + 5n + 3 \notin \mathcal{O}(n)$$

Usando el límite:

$$\lim_{n\to\infty} \frac{n^2 + 5n + 3}{n} = \lim_{n\to\infty} \left(n + 5 + \frac{3}{n} \right) = +\infty$$

Conclusión:

- $\ell = +\infty \Rightarrow f \in \Omega(n) \wedge f \notin \mathcal{O}(n)$
- Por lo tanto: $n^2 + 5n + 3 \in \Omega(n)$ pero no en $\mathcal{O}(n)$

Primeros pasos – Análisis de complejidad

Precondición: $|A| > 0$ (arreglo no vacío)

Algorithm $\textsc{BusquedaSecuencial}(A : \texttt{arreglo(nat)},\ e : \texttt{nat}) \longrightarrow \texttt{bool}$

var $i : nat, n : nat$
$n \leftarrow tam(A)$
$i \leftarrow 0$
mientras $i < n \wedge A[i] \neq e$ **hacer**
    $i \leftarrow i + 1$
devolver $(i < n)$

- $f_{mejor}(n)$ y $f_{peor}(n)$: **cantidad de operaciones** realizadas para un arreglo de tamaño n en mejor y peor caso respectivamente.

Primeros pasos – Análisis del **peor** caso

Precondición: $|A| > 0$ (arreglo no vacío)

Algorithm $\textsc{BusquedaSecuencial}(A : \texttt{arreglo(nat)}, e : \texttt{nat}) \longrightarrow \texttt{bool}$

1: **var** $i : nat, n : nat$
2: $n \leftarrow tam(A)$
3: $i \leftarrow 0$ $\triangleright$ línea 2-3: 3
4: **mientras** $i < n \wedge A[i] \neq e$ **hacer** $\triangleright$ 4
5: $\quad i \leftarrow i + 1$ $\triangleright$ 2
6: **devolver** $(i < n)$ $\triangleright$ 2

- $f_{mejor}(n)$ y $f_{peor}(n)$: **cantidad de operaciones** realizadas para un arreglo de tamaño n en mejor y peor caso respectivamente.

Primeros pasos – Análisis del **peor** caso

Precondición: $|A| > 0$ (arreglo no vacío)

Algorithm $\textsc{BusquedaSecuencial}(A : \texttt{arreglo(nat)},\ e : \texttt{nat}) \longrightarrow \texttt{bool}$

1: **var** $i : nat, n : nat$
2: $n \leftarrow tam(A)$
3: $i \leftarrow 0$ $\qquad\triangleright$ línea 2-3: 3
4: **mientras** $i < n \wedge A[i] \neq e$ **hacer** $\qquad\triangleright (n+1) \cdot 4$
5: $\quad i \leftarrow i + 1$ $\qquad\triangleright 2$
6: **devolver** $(i < n)$ $\qquad\triangleright 2$

- $f_{mejor}(n)$ y $f_{peor}(n)$: **cantidad de operaciones** realizadas para un arreglo de tamaño n en mejor y peor caso respectivamente.

Primeros pasos – Análisis del **peor** caso

Precondición: $|A| > 0$ (arreglo no vacío)

Algorithm $\textsc{BusquedaSecuencial}(A : \texttt{arreglo(nat)}, e : \texttt{nat}) \longrightarrow \texttt{bool}$

var $i : nat, n : nat$
$n \leftarrow tam(A)$
$i \leftarrow 0$ ▷ línea 2-3: 3
mientras $i < n \wedge A[i] \neq e$ **hacer** ▷ ciclo: $(n+1) \cdot 4 + 2 \cdot n$
    $i \leftarrow i + 1$ ▷ 2
devolver $(i < n)$ ▷ 2

- $f_{mejor}(n)$ y $f_{peor}(n)$: **cantidad de operaciones** realizadas para un arreglo de tamaño n en mejor y peor caso respectivamente.

Primeros pasos – Análisis del **peor** caso

Precondición: $|A| > 0$ (arreglo no vacío)

Algorithm $\textsc{BusquedaSecuencial}(A : \texttt{arreglo(nat)}, e : \texttt{nat}) \longrightarrow \texttt{bool}$

```
1: var i : nat, n : nat
2: n ← tam(A)
3: i ← 0                                          ▷ línea 2-3: 3
4: mientras i < n ∧ A[i] ≠ e hacer                 ▷ ciclo: (n + 1) · 4 + 2 · n
5:     i ← i + 1                                   ▷ 2
6: devolver (i < n)                                ▷ 2
```

- $f_{mejor}(n)$ y $f_{peor}(n)$: **cantidad de operaciones** realizadas para un arreglo de tamaño n en mejor y peor caso respectivamente.
- $f_{peor}(n) = 3 + (n+1)\ 4 + 2n + 2 =$

Primeros pasos – Análisis del **peor** caso

Precondición: $|A| > 0$ (arreglo no vacío)

Algorithm BUSQUEDASECUENCIAL(A : `arreglo(nat)`, e : `nat`) $\longrightarrow$ `bool`

1: **var** $i : nat, n : nat$
2: $n \leftarrow tam(A)$
3: $i \leftarrow 0$ $\quad\triangleright$ línea 2-3: 3
4: **mientras** $i < n \wedge A[i] \neq e$ **hacer** $\quad\triangleright$ ciclo: $(n+1) \cdot 4 + 2 \cdot n$
5: $\quad i \leftarrow i + 1$ $\quad\triangleright$ 2
6: **devolver** $(i < n)$ $\quad\triangleright$ 2

- $f_{mejor}(n)$ y $f_{peor}(n)$: **cantidad de operaciones** realizadas para un arreglo de tamaño n en mejor y peor caso respectivamente.
- $f_{peor}(n) = 3 + (n+1)\ 4 + 2n + 2 = \ 3 + 4 + 2 + \ (4 + 2)\ n$

Primeros pasos – Análisis del **peor** caso

Precondición: $|A| > 0$ (arreglo no vacío)

Algorithm $\textsc{BusquedaSecuencial}(A : \texttt{arreglo(nat)}, e : \texttt{nat}) \longrightarrow \texttt{bool}$

1: **var** $i : nat, n : nat$
2: $n \leftarrow tam(A)$
3: $i \leftarrow 0$ $\triangleright$ línea 2-3: 3
4: **mientras** $i < n \wedge A[i] \neq e$ **hacer** $\triangleright$ ciclo: $(n+1) \cdot 4 + 2 \cdot n$
5: $\quad i \leftarrow i + 1$ $\triangleright$ 2
6: **devolver** $(i < n)$ $\triangleright$ 2

- $f_{mejor}(n)$ y $f_{peor}(n)$: **cantidad de operaciones** realizadas para un arreglo de tamaño n en mejor y peor caso respectivamente.
- $f_{peor}(n) = 3 + (n+1)\,4 + 2n + 2 = 3 + 4 + 2 + (4 + 2)\,n$
  $= 9 + 6\,n$

Ejercicio

Elegir la o las opciones correcta:

- Si el algoritmo G es $O(n)$, siendo n el tamaño de la entrada:
  1. El peor caso de G tiene complejidad lineal
  2. La complejidad de G para cualquier caso está acotada superiormente por n
  3. El peor caso no puede ser $\Theta(log(n))$
  4. El peor caso es $O(n)$

Ejercicio

- Si el algoritmo G es $O(n)$, siendo n el tamaño de la entrada:
  1. El peor caso de G tiene complejidad lineal
     FALSO. $O(n)$ no implica que el peor caso sea exactamente lineal, $O(n)$ incluye también funciones de complejidad logarítmica o constante, por ejemplo.
  2. La complejidad de G para cualquier caso está acotada superiormente por n
     VERDADERO.
  3. El peor caso no puede ser $\Theta(log(n))$ FALSO. Si el peor caso tiene cota exacta logarítmica, cumple que está acotado superiormente por n
  4. El mejor caso es $O(n)$ VERDADERO. El peor caso (como todos) está acotado superiormente por n

Ejercicio: Dar complejidad en mejor y peor caso

Algorithm $\textsc{SumaProductosMenoresA100}(A : \texttt{arreglo(nat)}) \longrightarrow \texttt{nat}$

si $tam(A) = 0$ **entonces**
    devolver 0
var $i \leftarrow 0$, $suma \leftarrow 0$
mientras $i < tam(A)$ **hacer**
    var $j \leftarrow 0$
    mientras $j < tam(A)$ **hacer**
        si $A[i] \cdot A[j] < 100$ **entonces**
            $suma \leftarrow suma + A[i] \cdot A[j]$
        $j \leftarrow j + 1$
    $i \leftarrow i + 1$
devolver suma

Análisis de complejidad: SumaProductosMenoresA100

Algorithm SumaProductosMenoresA100(A)

```
si tam(A) = 0 entonces                       ▷ O(1)
    devolver 0                               ▷ O(1)
var i ← 0, suma ← 0                          ▷ O(1)
mientras i < tam(A) hacer                    ▷ Θ(n) iteraciones
    var j ← 0                                ▷ O(1)
    mientras j < tam(A) hacer                ▷ Θ(n) iteraciones
        si A[i] · A[j] < 100 entonces        ▷ O(1)
            suma ← suma + A[i] · A[j]        ▷ O(1)
        j ← j + 1                            ▷ O(1)
    i ← i + 1                                ▷ O(1)
devolver suma                                ▷ O(1)
```

¿Qué influye en la complejidad?

- Dos ciclos anidados que recorren el arreglo de tamaño n
- Dentro del segundo ciclo hay un *if* pero no afecta la complejidad total (si entra al if sólo hace operaciones de costo constante)

Para este algoritmo el peor y el mejor caso son iguales,
Complejidad:

$$T(n) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \Theta(1)$$

$$\Rightarrow T(n) = \Theta\left(\sum_{i=0}^{n-1} \sum_{j=0}^{n-1} 1\right) = \Theta\left(\sum_{i=0}^{n-1} (n-1)\right) = \Theta(n \cdot n) = \boxed{\Theta(n^2)}$$

Algorithm ALGORITMOQUEHACEALGO(A : `arreglo(nat)`)

```
i ← 0;
suma ← 1; count ← 0
mientras i ≤ tam(A) hacer
    si i ≠ A[i] entonces
        count ← count + 1
    j ← 1
    mientras j ≤ count hacer
        k ← 1
        mientras k ≤ tam(A) hacer
            suma ← suma + A[k]
            k ← k · 2
        j ← j + 1
    i ← i + 1
devolver suma
```

- ¿Cuándo es mejor caso?
- ¿Cuándo es peor caso?

Algorithm ALGQUEHACEALGO(A)

```
i ← 1; j ← 1
suma ← 1; count ← 0
mientras i ≤ tam(A) hacer
    si i ≠ A[i] entonces
        count ← count + 1
    j ← 1
    mientras j ≤ count hacer
        k ← 1
        mientras k ≤ tam(A) hacer
            suma ← suma + A[k]
            k ← k · 2
        j ← j + 1
    i ← i + 1
devolver suma
```

$A[1]$ es la primera posición del arreglo.

Mejor caso: $A[i] = i$ siempre

$$T_{mejor} = \sum_{i=1}^{n} (\Theta(1)) = \Theta(n)$$

Algorithm $\textsc{AlgQueHaceAlgo}(A)$

$i \leftarrow 1;\ j \leftarrow 1$
$suma \leftarrow 1;\ count \leftarrow 0$
mientras $i \leq tam(A)$ **hacer**
    si $i \neq A[i]$ **entonces**
        $count \leftarrow count + 1$
    $j \leftarrow 1$
    mientras $j \leq count$ **hacer**
        $k \leftarrow 1$
        mientras $k \leq tam(A)$ **hacer**
            $suma \leftarrow suma + A[k]$
            $k \leftarrow k \cdot 2$
        $j \leftarrow j + 1$
    $i \leftarrow i + 1$
devolver suma

$A[1]$ es la primera posición del arreglo.

Peor caso: $A[i] \neq i$ siempre

Algorithm $\textsc{AlgQueHaceAlgo}(A)$

$i \leftarrow 1$; $j \leftarrow 1$
$suma \leftarrow 1$; $count \leftarrow 0$
mientras $i \leq tam(A)$ **hacer**
 si $i \neq A[i]$ **entonces**
  $count \leftarrow count + 1$
 $j \leftarrow 1$
 mientras $j \leq count$ **hacer**
  $k \leftarrow 1$
  mientras $k \leq tam(A)$ **hacer**
   $suma \leftarrow suma + A[k]$
   $k \leftarrow k \cdot 2$
  $j \leftarrow j + 1$
 $i \leftarrow i + 1$
devolver suma

$A[1]$ es la primera posición del arreglo.

Peor caso: $A[i] \neq i$ siempre

$$T_{peor}(n) = \sum_{i=1}^{n} \sum_{j=1}^{i} \log n$$

Algorithm ALGQUEHACEALGO(A)

$i \leftarrow 1$; $j \leftarrow 1$
$suma \leftarrow 1$; $count \leftarrow 0$
mientras $i \leq tam(A)$ **hacer**
  si $i \neq A[i]$ **entonces**
    $count \leftarrow count + 1$
  $j \leftarrow 1$
  mientras $j \leq count$ **hacer**
    $k \leftarrow 1$
    mientras $k \leq tam(A)$ **hacer**
      $suma \leftarrow suma + A[k]$
      $k \leftarrow k \cdot 2$
    $j \leftarrow j + 1$
  $i \leftarrow i + 1$
devolver suma

$A[1]$ es la primera posición del arreglo.

Peor caso: $A[i] \neq i$ siempre

$$T_{peor}(n) = \sum_{i=1}^{n} \sum_{j=1}^{i} \log n$$

$$= \log n \sum_{i=1}^{n} i$$

$$= \log n \frac{n(n+1)}{2}$$

$$= \frac{\log n}{2}(n^2 + n)$$

$$T_{peor} \in \Theta(n^2 \log n)$$

Algorithm ALGQUEHACEALGO(A)

```
i ← 1; j ← 1
suma ← 1; count ← 0
mientras i ≤ tam(A) hacer
    si i ≠ A[i] entonces
        count ← count + 1
    j ← 1
    mientras j ≤ count hacer
        k ← 1
        mientras k ≤ tam(A) hacer
            suma ← suma + A[k]
            k ← k · 2
        j ← j + 1
    i ← i + 1
devolver suma
```

$A[1]$ es la primera posición del arreglo.