

# **Algoritmos II**

## **Fundamentos de Computación**

**Diego Caro**

**2021-1**

# Resultados de aprendizaje

- Describir el funcionamiento de algoritmos recursivos
- Describir mecanismo funcionamiento algoritmos de ordenación
- Comprender conceptos de tiempo de ejecución, peor caso y mejor caso

# Recordatorio: funciones

- Secuencia de instrucciones diseñadas para resolver una **tarea** específica.
- El objetivo es reutilizarla en programas (o pseudocódigo) cada vez que se desea resolver esa **tarea**.
- La función está definida por un nombre, una entrada, una salida, y la secuencia de instrucciones.
  - La entrada es lo que deseamos procesar (pueden ser números, texto, lista de elementos, etc...)
  - La salida es el resultado de computar las instrucciones con la entrada.
  - La salida se especifica con la palabra **retornar**.
- Es análogo a funciones matemáticas!

Función en matemáticas

$$f(x) = x^2$$

Función en pseudocódigo

```
def f(x):  
    return x*x
```

**¿Como calcular el número  
máximo para  $N$  números?**

1. Sea  $X$  una lista  $X_1, X_2, \dots, X_N$ ;  $X_i \in \mathbb{R}$

→ 2. Leer  $N$  números y dejarlos en la lista  $X$

3.  $\text{max} = \mathbf{Maximo}(X)$

4. print “El máximo numero en  $X$  es”,  $\text{max}$

1. def  $\text{Maximo}(Z)$ :

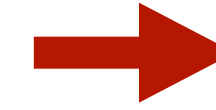
2.     Sea  $m = -\infty$  un número real negativo

3.     **for**  $i = 1$  **to**  $N$ :

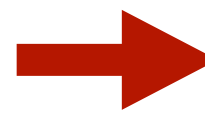
4.         **if**  $Z_i > m$ :

5.              $m = Z_i$

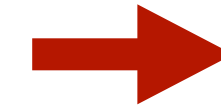
6.     **return**  $m$



$X$	$\text{max}$
5,9,7	-

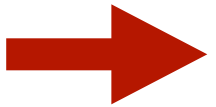
- 
1. Sea  $X$  una lista  $X_1, X_2, \dots, X_N$ ;  $X_i \in \mathbb{R}$
  2. Leer  $N$  números y dejarlos en la lista  $X$
  3.  $\text{max} = \mathbf{Maximo}(X)$
  4. print “El máximo numero en  $X$  es”,  $\text{max}$

1. def Maximo( $Z$ ):
2.   Sea  $m = -\infty$  un número real negativo
3.   **for**  $i = 1$  **to**  $N$ :
4.     **if**  $Z_i > m$ :
5.        $m = Z_i$
6.   **return**  $m$





$X$	$\text{max}$
5,9,7	-
5,9,7	?????

1. Sea  $X$  una lista  $X_1, X_2, \dots, X_N$ ;  $X_i \in \mathbb{R}$
2. Leer  $N$  números y dejarlos en la lista  $X$
3.  $\text{max} = \mathbf{Maximo}(X)$
4. print “El máximo numero en  $X$  es”,  $\text{max}$



X	max
5,9,7	-
5,9,7	?????

- 
1. def **Maximo**(Z):
  2.   Sea  $m = -\infty$  un número real negativo
  3.   **for**  $i = 1$  **to**  $N$ :
  4.     **if**  $Z_i > m$ :
  5.        $m = Z_i$
  6.   **return**  $m$



Z	m	i	$Z_i$	$Z_i > m$
5,9,7	-	-	-	-

- 1. Sea  $X$  una lista  $X_1, X_2, \dots, X_N$ ;  $X_i \in \mathbb{R}$
- 2. Leer  $N$  números y dejarlos en la lista  $X$
- 3.  $\text{max} = \text{Maximo}(X)$
- 4. print “El máximo numero en  $X$  es”,  $\text{max}$

X	max
5,9,7	-
5,9,7	?????

- 1. def Maximo( $Z$ ):
- 2. Sea  $m = -\infty$  un número real negativo
- 3. for  $i = 1$  to  $N$ :
- 4. if  $Z_i > m$ :
- 5.  $m = Z_i$
- 6. return  $m$

Z	m	i	$Z_i$	$Z_i > m$
5,9,7	-	-	-	-
5,9,7	$-\infty$	-	-	-



- 1. Sea  $X$  una lista  $X_1, X_2, \dots, X_N; X_i \in \mathbb{R}$
- 2. Leer  $N$  números y dejarlos en la lista  $X$
- 3.  $\text{max} = \text{Maximo}(X)$
- 4. print “El máximo numero en  $X$  es”,  $\text{max}$

X	max
5,9,7	-
5,9,7	?????

- 1. def Maximo( $Z$ ):
- 2.   Sea  $m = -\infty$  un número real negativo
- 3.   **for**  $i = 1$  **to**  $N$ :
- 4.     **if**  $Z_i > m$ :
- 5.        $m = Z_i$
- 6.   **return**  $m$

Z	m	i	$Z_i$	$Z_i > m$
5,9,7	-	-	-	-
5,9,7	$-\infty$	-	-	-
5,9,7	$-\infty$	1	5	

- 1. Sea  $X$  una lista  $X_1, X_2, \dots, X_N$ ;  $X_i \in \mathbb{R}$
- 2. Leer  $N$  números y dejarlos en la lista  $X$
- 3.  $\text{max} = \text{Maximo}(X)$
- 4. print “El máximo numero en  $X$  es”,  $\text{max}$

X	max
5,9,7	-
5,9,7	?????

- 1. def Maximo( $Z$ ):
- 2.   Sea  $m = -\infty$  un número real negativo
- 3.   **for**  $i = 1$  **to**  $N$ :
- 4.       **if**  $Z_i > m$ :
- 5.          $m = Z_i$
- 6.   **return**  $m$

Z	m	i	$Z_i$	$Z_i > m$
5,9,7	-	-	-	-
5,9,7	$-\infty$	-	-	-
5,9,7	$-\infty$	1	5	V

- 1. Sea  $X$  una lista  $X_1, X_2, \dots, X_N$ ;  $X_i \in \mathbb{R}$
- 2. Leer  $N$  números y dejarlos en la lista  $X$
- 3.  $\text{max} = \text{Maximo}(X)$
- 4. print “El máximo numero en  $X$  es”,  $\text{max}$

X	max
5,9,7	-
5,9,7	?????

- 1. def Maximo( $Z$ ):
- 2.   Sea  $m = -\infty$  un número real negativo
- 3.   **for**  $i = 1$  **to**  $N$ :
- 4.     **if**  $Z_i > m$ :
- 5.        $m = Z_i$
- 6.   **return**  $m$

Z	m	i	$Z_i$	$Z_i > m$
5,9,7	-	-	-	-
5,9,7	$-\infty$	-	-	-
5,9,7	$-\infty$	1	5	V
5,9,7	5	1	5	V

- 1. Sea  $X$  una lista  $X_1, X_2, \dots, X_N; X_i \in \mathbb{R}$
- 2. Leer  $N$  números y dejarlos en la lista  $X$
- 3.  $\text{max} = \text{Maximo}(X)$
- 4. print “El máximo numero en  $X$  es”,  $\text{max}$

X	max
5,9,7	-
5,9,7	?????

- 1. def Maximo( $Z$ ):
- 2.   Sea  $m = -\infty$  un número real negativo
- 3.   **for**  $i = 1$  **to**  $N$ :
- 4.     **if**  $Z_i > m$ :
- 5.        $m = Z_i$
- 6.   **return**  $m$

Z	m	i	$Z_i$	$Z_i > m$
5,9,7	-	-	-	-
5,9,7	$-\infty$	-	-	-
5,9,7	$-\infty$	1	5	V
5,9,7	5	1	5	V
5,9,7	5	2	9	

- 1. Sea  $X$  una lista  $X_1, X_2, \dots, X_N; X_i \in \mathbb{R}$
- 2. Leer  $N$  números y dejarlos en la lista  $X$
- 3.  $\text{max} = \text{Maximo}(X)$
- 4. print “El máximo numero en  $X$  es”,  $\text{max}$

X	max
5,9,7	-
5,9,7	?????

- 1. def Maximo( $Z$ ):
- 2.   Sea  $m = -\infty$  un número real negativo
- 3.   for  $i = 1$  to  $N$ :
- 4.     if  $Z_i > m$ :
- 5.        $m = Z_i$
- 6.   return  $m$

Z	m	i	$Z_i$	$Z_i > m$
5,9,7	-	-	-	-
5,9,7	$-\infty$	-	-	-
5,9,7	$-\infty$	1	5	V
5,9,7	5	1	5	V
5,9,7	5	2	9	V

- 1. Sea  $X$  una lista  $X_1, X_2, \dots, X_N; X_i \in \mathbb{R}$
- 2. Leer  $N$  números y dejarlos en la lista  $X$
- 3.  $\text{max} = \text{Maximo}(X)$
- 4. print “El máximo numero en  $X$  es”,  $\text{max}$

X	max
5,9,7	-
5,9,7	?????

- 1. def Maximo( $Z$ ):
- 2.   Sea  $m = -\infty$  un número real negativo
- 3.   **for**  $i = 1$  **to**  $N$ :
- 4.     **if**  $Z_i > m$ :
- 5.        $m = Z_i$
- 6.   **return**  $m$

Z	m	i	$Z_i$	$Z_i > m$
5,9,7	-	-	-	-
5,9,7	$-\infty$	-	-	-
5,9,7	$-\infty$	1	5	V
5,9,7	5	1	5	V
5,9,7	5	2	9	V
5,9,7	9	2	9	V

- 1. Sea  $X$  una lista  $X_1, X_2, \dots, X_N$ ;  $X_i \in \mathbb{R}$
- 2. Leer  $N$  números y dejarlos en la lista  $X$
- 3.  $\text{max} = \text{Maximo}(X)$
- 4. print “El máximo numero en  $X$  es”,  $\text{max}$

$X$	$\text{max}$
5,9,7	-
5,9,7	?????

- 1. def Maximo( $Z$ ):
- 2.   Sea  $m = -\infty$  un número real negativo
- 3.   for  $i = 1$  to  $N$ :
- 4.     if  $Z_i > m$ :
- 5.        $m = Z_i$
- 6.   return  $m$

$Z$	$m$	$i$	$Z_i$	$Z_i > m$
5,9,7	-	-	-	-
5,9,7	$-\infty$	-	-	-
5,9,7	$-\infty$	1	5	V
5,9,7	5	1	5	V
5,9,7	5	2	9	V
5,9,7	9	2	9	V
5,9,7	9	3	7	F

- 1. Sea  $X$  una lista  $X_1, X_2, \dots, X_N$ ;  $X_i \in \mathbb{R}$
- 2. Leer  $N$  números y dejarlos en la lista  $X$
- 3.  $\text{max} = \text{Maximo}(X)$
- 4. print “El máximo numero en  $X$  es”,  $\text{max}$

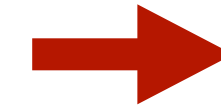
X	max
5,9,7	-
5,9,7	?????

- 1. def Maximo(Z):
- 2.   Sea  $m = -\infty$  un número real negativo
- 3.   for  $i = 1$  to  $N$ :
- 4.     if  $Z_i > m$ :
- 5.        $m = Z_i$
- 6.   return  $m$

Z	m	i	$Z_i$	$Z_i > m$
5,9,7	-	-	-	-
5,9,7	$-\infty$	-	-	-
5,9,7	$-\infty$	1	5	V
5,9,7	5	1	5	V
5,9,7	5	2	9	V
5,9,7	9	2	9	V
5,9,7	9	3	7	F



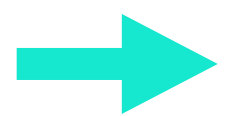
1. Sea  $X$  una lista  $X_1, X_2, \dots, X_N$ ;  $X_i \in \mathbb{R}$
2. Leer  $N$  números y dejarlos en la lista  $X$
3.  $\text{max} = \mathbf{Maximo}(X)$
4. print “El máximo numero en  $X$  es”,  $\text{max}$



X	max
5,9,7	-
5,9,7	9



1. def Maximo( $Z$ ):
2.   Sea  $m = -\infty$  un número real negativo
3.   **for**  $i = 1$  **to**  $N$ :
4.     **if**  $Z_i > m$ :
5.        $m = Z_i$
6.   **return**  $m$

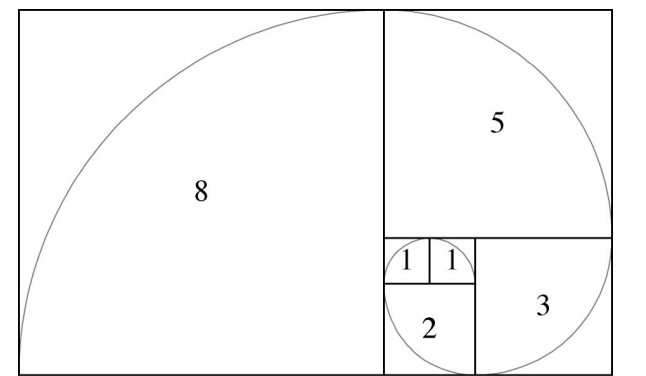


# Funciones recursivas

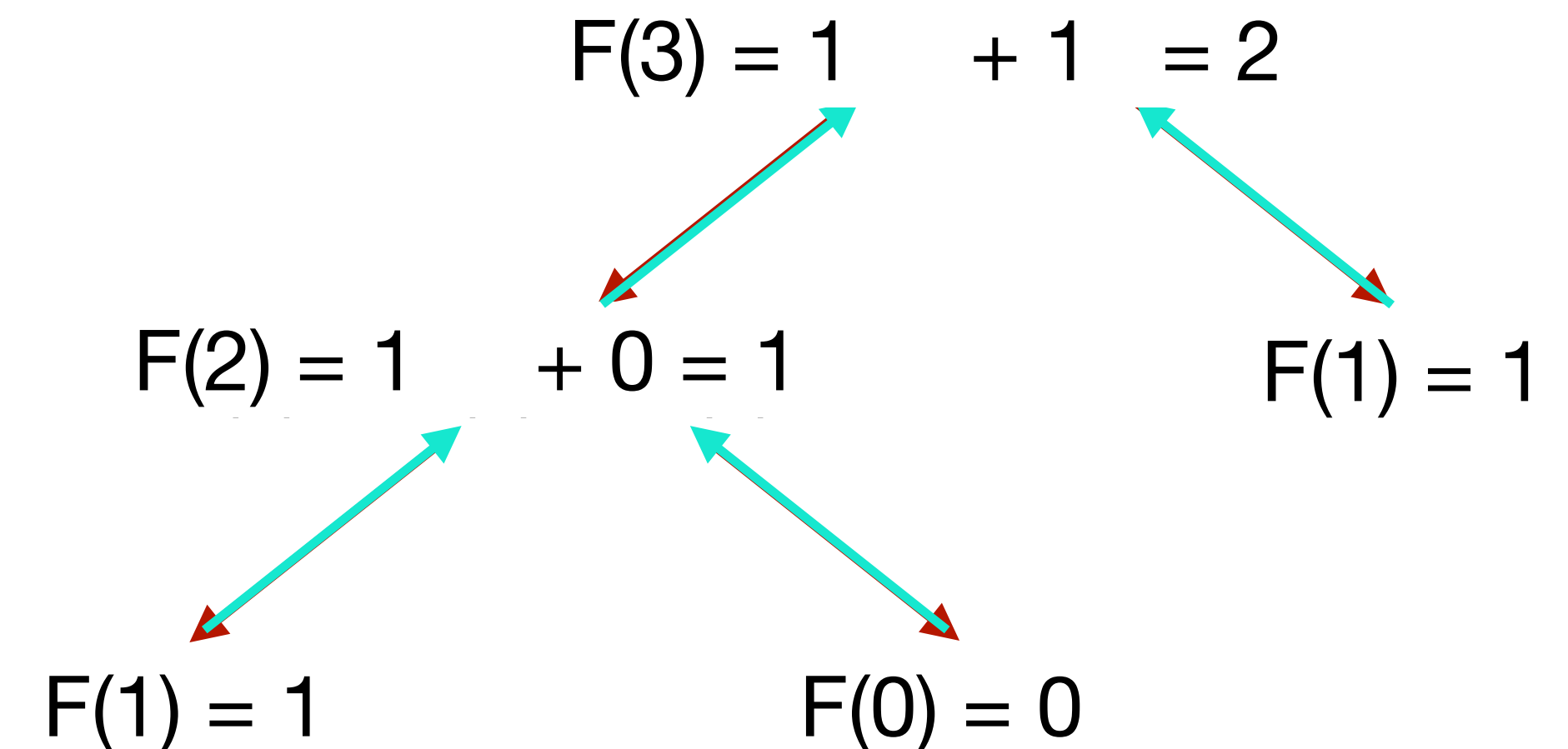
- Algoritmo que calcula una salida de forma recursiva, es decir, ejecutándose a si mismo.
- Leonardo Fibonacci, matemático italiano conocido por su famosa secuencia:
  - 0, 1, 1, 2, 3, 5, 8, 13, 21, 34...
- Definición recursiva:

$$F_n = \begin{cases} F_{n-1} + F_{n-2} & \text{if } n > 1 \\ 1 & \text{if } n = 1 \\ 0 & \text{if } n = 0. \end{cases}$$

Espiral de Fibonacci:



¿Cuánto es F(3)?



# Funciones recursivas

- Cómo construir una función recursiva: definir casos
  - **Caso Recursivo:** ¿con que valores la función se llama a si misma?
  - **Caso Base:** ¿con qué valores se detiene la función?
- Puede haber más de un caso base, y más de un caso recursivo.
- Debes asegurarte que la función termine con al menos un caso base.

$$F_n = \begin{cases} F_{n-1} + F_{n-2} & \text{if } n > 1 & \text{Caso recursivo} \\ 1 & \text{if } n = 1 & \text{Caso base} \\ 0 & \text{if } n = 0. & \text{Caso base} \end{cases}$$

```
1. def F(n):
2.     if n > 1:
3.         return F(n-1) + F(n-2)
4.     if n == 1:
5.         return 1
6.     if n == 0:
7.         return 0
```

$$\text{Factorial}(n) = \begin{cases} n \times \text{Factorial}(n - 1) & \text{si } n > 1 \\ 1 & \text{si } n == 1 \end{cases}$$

**Caso recursivo**  
**Caso base**

Factorial(3) = 3 \* Factorial(2)

Factorial(2) = 2 \* Factorial(1)

Factorial(1) = 1

1. **def** Factorial(n):
2.     **if** n > 1: **Caso recursivo**
3.         **return** n\*Factorial(n-1)
4.     **if** n == 1: **Caso base**
5.         **return** 1



# Fractales

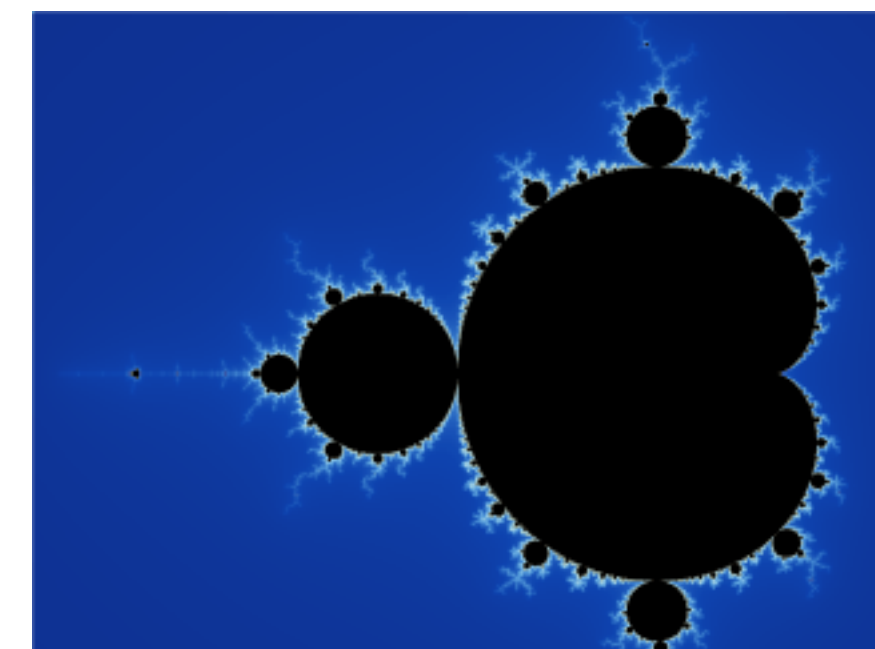
- Benoit Mandelbrot acuñó el término Fractal para referirse a:
  - "...un objeto geométrico que puede dividirse en partes, cada una de las cuales es una copia (aproximada) de la original...."
- Sirven para modelar la naturaleza (y estudiarla), para compresión, arte, etc...
- Y también son objetos recursivos!



Brócoli fractal (es real!)



Curva de Koch



Conjunto de Mandelbrot

# Fractal recursivo: Curva de Koch

1. Start with a line.



2. Divide the line into three equal parts.



3. Draw an equilateral triangle (a triangle where all the sides are equal) using the middle segment as its base.



4. Erase the base of the equilateral triangle (the middle segment from step 2).

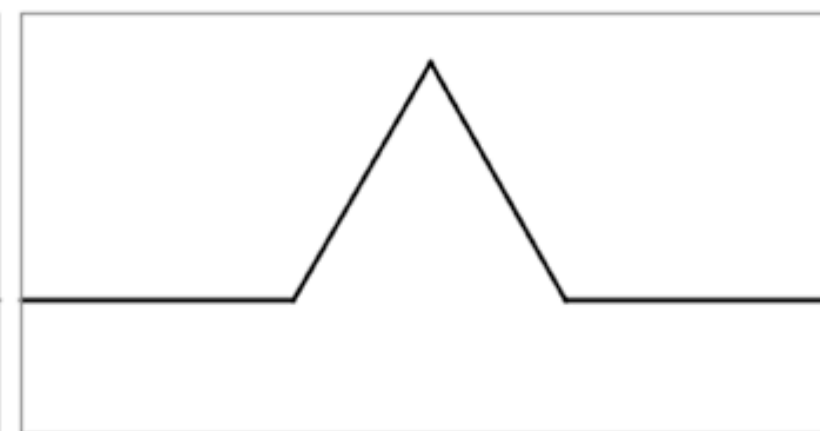


5. Repeat steps 2 through 4 for the remaining lines again and again and again.

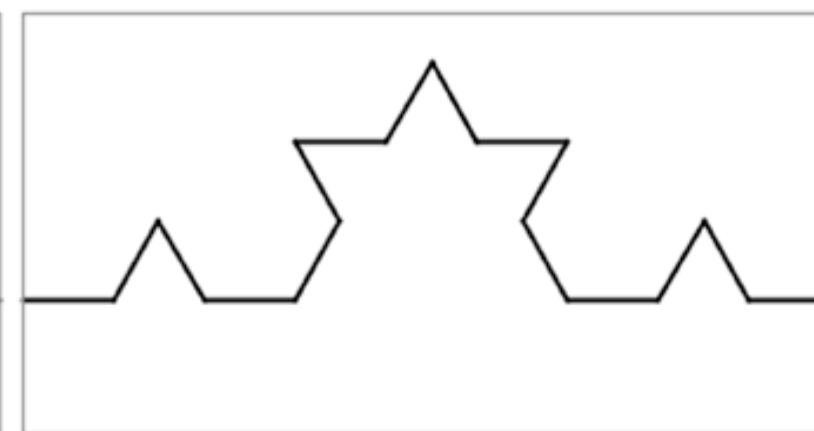
Inicio



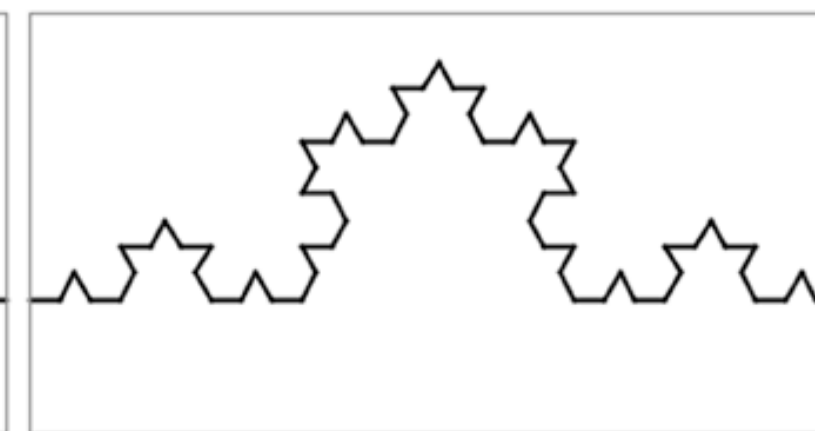
1ra recursión



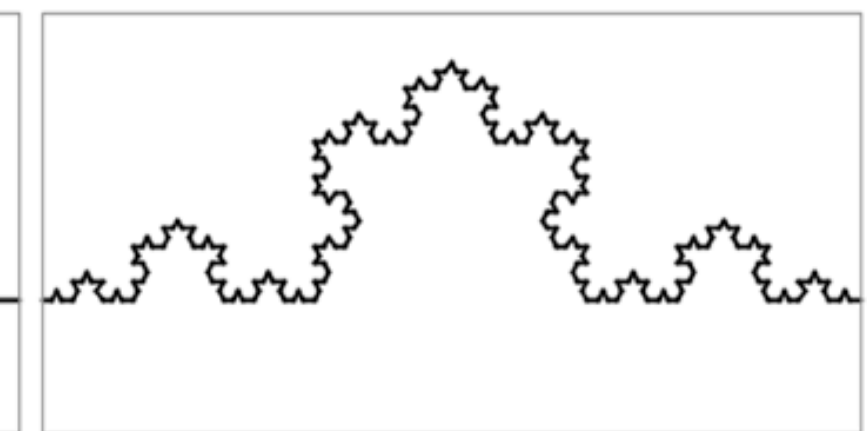
2da recursión



3ra recursión



4ta recursión



# Actividad

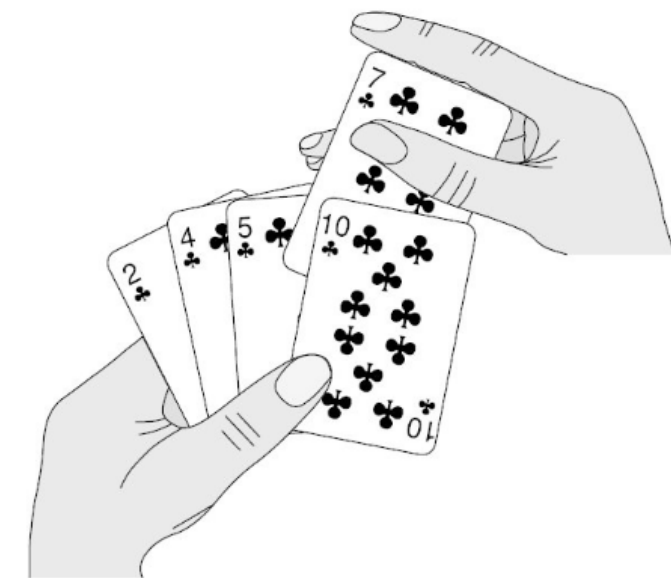
- Cómo sería el pseudocódigo de la función recursiva que permite dibujar la curva de Koch?
- Asuma que puede dibujar líneas entre dos puntos.
- Utilice la estrategia de resolución de problemas de George Polya.
  - **Hint:** parta por hacer el caso base y los casos recursivos!

# Ordenamiento por Inserción

- Idea: estrategia similar al ordenar cartas para formar una escala en el juego de naipes **carioca**.
- Es un algoritmo iterativo,
  - Mantiene una sublista de elementos ordenados en cada iteración.
  - Inserta un nuevo elemento, y lo mueve hasta hacer que la sublista se mantenga ordenada.
  - Y así sucesivamente hasta que se ha insertado el último elemento de la lista.

a:

6 5 3 1 8 7 2 4





# Ordenamiento por Inserción

- **Iteración i:** intercambiar repetidamente  $a_i$  si el elemento a la izquierda es menor
- **Propiedad:** después de i-ésima iteración, los elementos entre  $a_0$  y  $a_i$  están ordenados.

**Atención:** asuma que el texto se compara siguiendo orden lexicográfico (tal como está en un diccionario).

la	el	uno	los	las	una	tu	yo
1	2	3	4	5	6	7	8

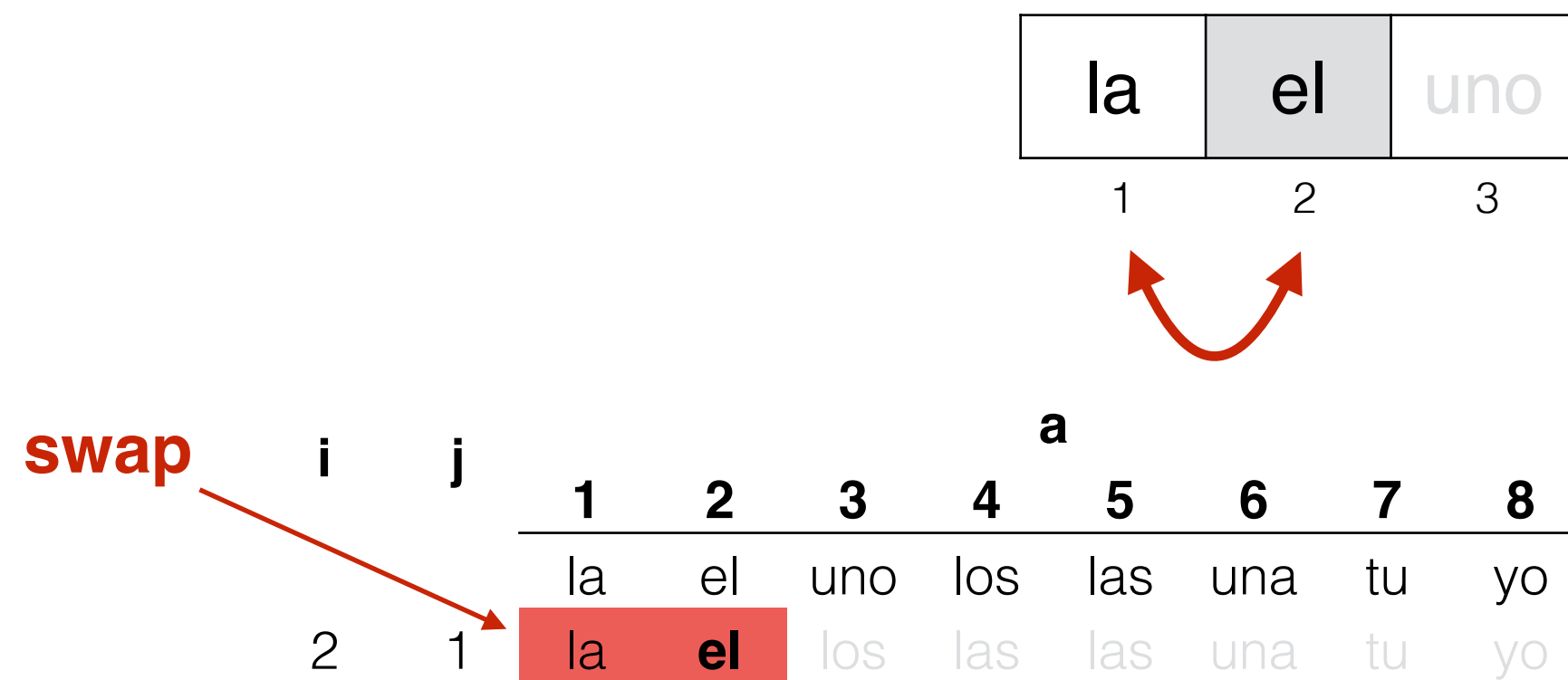
i	j	a							
		1	2	3	4	5	6	7	8
		la	el	uno	los	las	una	tu	yo

```
def insertionsort(A):  
    for i=2 to N:  
        k = Ai  
        j = i-1  
        while j >= 1 and Aj > k:  
            Aj+1 = Aj  
            Aj = k  
            j = j - 1
```

**Idea:** Insertar  $A_i$  en la sublista ordenada  $A_1$  hasta  $A_j$

# Ordenamiento por Inserción

- **Iteración i:** intercambiar repetidamente  $a_i$  si el elemento a la izquierda es menor
- **Propiedad:** después de  $i$ -ésima iteración, los elementos entre  $a_0$  y  $a_i$  están ordenados.



```
def insertionsort(A):
```

```
    for i=2 to N:
```

```
        k = Ai
```

```
        j = i-1
```

```
        while j >= 1 and Aj > k:
```

```
            Aj+1 = Aj
```

```
            Aj = k
```

```
            j = j - 1
```

**Idea:** Insertar  $A_i$  en la sublista ordenada  $A_1$  hasta  $A_j$

# Ordenamiento por Inserción

- **Iteración i:** intercambiar repetidamente  $a_i$  si el elemento a la izquierda es menor
- **Propiedad:** después de  $i$ -ésima iteración, los elementos entre  $a_0$  y  $a_i$  están ordenados.

el	la	uno	los	las	una	tu	yo
1	2	3	4	5	6	7	8

swap	i	j	a							
			1	2	3	4	5	6	7	8
			la	el	uno	los	las	una	tu	yo
	2	1	la	el	los	las	las	una	tu	yo
	3	2	el	la	uno	las	las	una	tu	yo

```
def insertionsort(A):
```

```
    for i=2 to N:
```

```
        k = Ai
```

```
        j = i-1
```

```
        while j >= 1 and Aj > k:
```

```
            Aj+1 = Aj
```

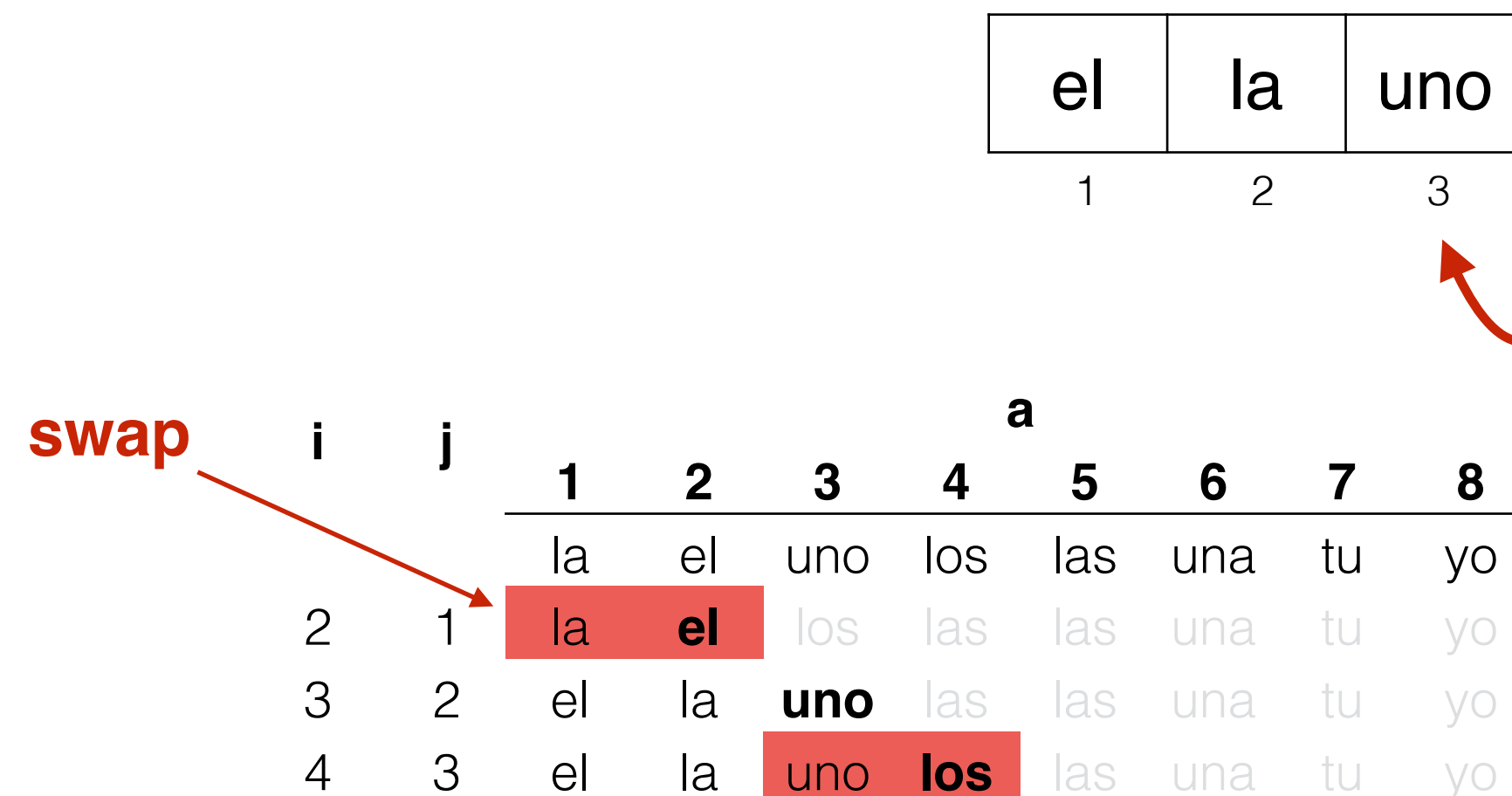
```
            Aj = k
```

```
            j = j - 1
```

**Idea:** Insertar  $A_i$  en la sublista ordenada  $A_1$  hasta  $A_j$

# Ordenamiento por Inserción

- **Iteración i:** intercambiar repetidamente  $a_i$  si el elemento a la izquierda es menor
- **Propiedad:** después de  $i$ -ésima iteración, los elementos entre  $a_0$  y  $a_i$  están ordenados.



```
def insertionsort(A):
```

```
    for i=2 to N:
```

```
        k = Ai
```

```
        j = i-1
```

```
        while j >= 1 and Aj > k:
```

```
            Aj+1 = Aj
```

```
            Aj = k
```

```
            j = j - 1
```

**Idea:** Insertar  $A_i$  en la sublista ordenada  $A_1$  hasta  $A_j$

# Ordenamiento por Inserción

- **Iteración i:** intercambiar repetidamente  $a_i$  si el elemento a la izquierda es menor
- **Propiedad:** después de  $i$ -ésima iteración, los elementos entre  $a_0$  y  $a_i$  están ordenados.

el	la	los	uno	las	una	tu	yo
1	2	3	4	5	6	7	8

		a								
		1	2	3	4	5	6	7	8	
	i	j	la	el	uno	los	las	una	tu	yo
swap	2	1	la	el	los	las	las	una	tu	yo
	3	2	el	la	uno	las	las	una	tu	yo
	4	3	el	la	uno	los	las	una	tu	yo
	4	2	el	la	los	uno	las	una	tu	yo

```
def insertionsort(A):
```

```
    for i=2 to N:
```

```
        k = Ai
```

```
        j = i-1
```

```
        while j >= 1 and Aj > k:
```

```
            Aj+1 = Aj
```

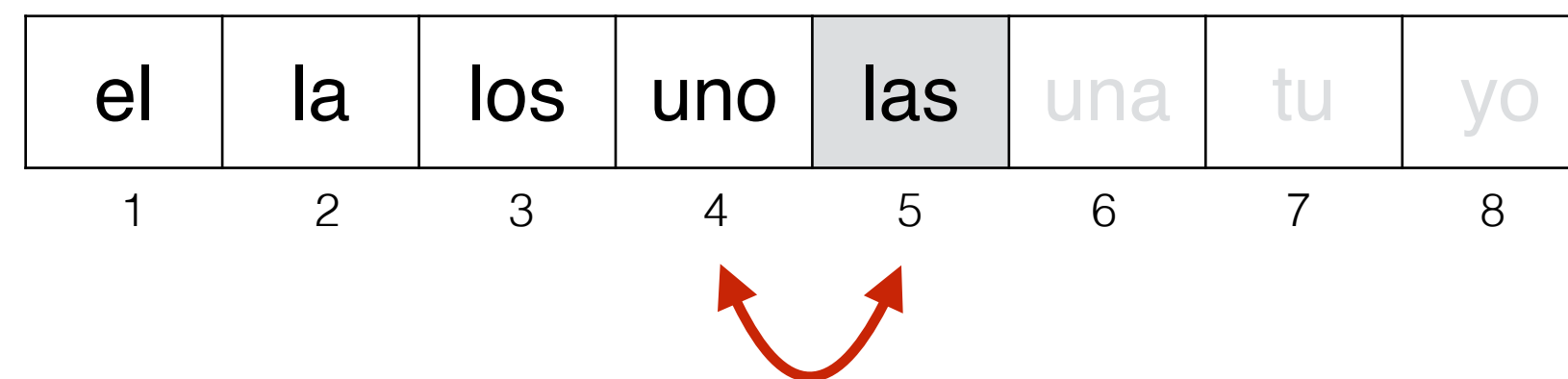
```
            Aj = k
```

```
            j = j - 1
```

**Idea:** Insertar  $A_i$  en la sublista ordenada  $A_1$  hasta  $A_j$

# Ordenamiento por Inserción

- **Iteración i:** intercambiar repetidamente  $a_i$  si el elemento a la izquierda es menor
- **Propiedad:** después de  $i$ -ésima iteración, los elementos entre  $a_0$  y  $a_i$  están ordenados.



		a							
		1	2	3	4	5	6	7	8
		la	el	uno	los	las	una	tu	yo
swap	2	1	la	el	los	las	una	tu	yo
	3	2	el	la	uno	las	una	tu	yo
	4	3	el	la	uno	los	las	una	yo
	4	2	el	la	los	uno	las	una	yo
	5	4	el	la	los	uno	las	una	yo

```
def insertionsort(A):
```

```
    for i=2 to N:
```

```
        k = Ai
```

```
        j = i-1
```

```
        while j >= 1 and Aj > k:
```

```
            Aj+1 = Aj
```

```
            Aj = k
```


```
            j = j - 1
```

**Idea:** Insertar  $A_i$  en la sublista ordenada  $A_1$  hasta  $A_j$

# Ordenamiento por Inserción

- **Iteración i:** intercambiar repetidamente  $a_i$  si el elemento a la izquierda es menor
- **Propiedad:** después de  $i$ -ésima iteración, los elementos entre  $a_0$  y  $a_i$  están ordenados.

el	la	los	las	uno	una	tu	yo
1	2	3	4	5	6	7	8



**swap** 

	i	j	a							
			1	2	3	4	5	6	7	8
			la	el	uno	los	las	una	tu	yo
	2	1	la	el	los	las	las	una	tu	yo
	3	2	el	la	uno	las	las	una	tu	yo
	4	3	el	la	uno	los	las	una	tu	yo
	4	2	el	la	los	uno	las	una	tu	yo
	5	4	el	la	los	uno	las	una	tu	yo
	5	3	la	la	los	las	uno	una	tu	yo

```
def insertionsort(A):
```

```
    for i=2 to N:
```

```
        k = Ai
```

```
        j = i-1
```

```
        while j >= 1 and Aj > k:
```

```
            Aj+1 = Aj
```

```
            Aj = k
```

```
            j = j - 1
```

**Idea:** Insertar  $A_i$  en la sublista ordenada  $A_1$  hasta  $A_j$

# Ordenamiento por Inserción

- **Iteración i:** intercambiar repetidamente  $a_i$  si el elemento a la izquierda es menor
- **Propiedad:** después de  $i$ -ésima iteración, los elementos entre  $a_0$  y  $a_i$  están ordenados.

swap

	i	j	a							
			1	2	3	4	5	6	7	8
			la	el	uno	los	las	una	tu	yo
	2	1	la	el	los	las	las	una	tu	yo
	3	2	el	la	uno	las	las	una	tu	yo
	4	3	el	la	uno	los	las	una	tu	yo
	4	2	el	la	los	uno	las	una	tu	yo
	5	4	el	la	los	uno	las	una	tu	yo
	5	3	la	la	los	las	uno	una	tu	yo
	5	2	el	la	las	los	uno	una	tu	yo

```
def insertionsort(A):  
    for i=2 to N:  
        k = Ai  
        j = i-1  
        while j >= 1 and Aj > k:  
            Aj+1 = Aj  
            Aj = k  
            j = j - 1
```


Idea: Insertar A<sub>i</sub> en la sublista ordenada A<sub>1</sub> hasta A<sub>j</sub>



# Ordenamiento por Inserción

- **Iteración i:** intercambiar repetidamente  $a_i$  si el elemento a la izquierda es menor
- **Propiedad:** después de  $i$ -ésima iteración, los elementos entre  $a_0$  y  $a_i$  están ordenados.

el	la	las	los	uno	una	tu	yo
1	2	3	4	5	6	7	8



		a								
	i	j	1	2	3	4	5	6	7	8
			la	el	uno	los	las	una	tu	yo
2	1		la	el	los	las	las	una	tu	yo
3	2		el	la	uno	las	las	una	tu	yo
4	3		el	la	uno	los	las	una	tu	yo
4	2		el	la	los	uno	las	una	tu	yo
5	4		el	la	los	uno	las	una	tu	yo
5	3		la	la	los	las	uno	una	tu	yo
5	2		el	la	las	los	uno	una	tu	yo
6	5		el	la	las	los	uno	una	tu	yo

```
def insertionsort(A):
```

```
    for i=2 to N:
```

```
        k = Ai
```

```
        j = i-1
```

```
        while j >= 1 and Aj > k:
```

```
            Aj+1 = Aj
```

```
            Aj = k
```

```
            j = j - 1
```

**Idea:** Insertar  $A_i$  en la sublista ordenada  $A_1$  hasta  $A_j$

# Ordenamiento por Inserción

- **Iteración i:** intercambiar repetidamente  $a_i$  si el elemento a la izquierda es menor
- **Propiedad:** después de  $i$ -ésima iteración, los elementos entre  $a_0$  y  $a_i$  están ordenados.

el	la	las	los	una	uno	tu	yo
1	2	3	4	5	6	7	8

swap			a							
	i	j	1	2	3	4	5	6	7	8
			la	el	uno	los	las	una	tu	yo
	2	1	la	el	los	las	las	una	tu	yo
	3	2	el	la	uno	las	las	una	tu	yo
	4	3	el	la	uno	los	las	una	tu	yo
	4	2	el	la	los	uno	las	una	tu	yo
	5	4	el	la	los	uno	las	una	tu	yo
	5	3	la	la	los	las	uno	una	tu	yo
	5	2	el	la	las	los	uno	una	tu	yo
	6	5	el	la	las	los	uno	una	tu	yo
	6	4	el	la	las	los	una	uno	tu	yo

```
def insertionsort(A):
```

```
    for i=2 to N:
```

```
        k = Ai
```

```
        j = i-1
```

```
        while j >= 1 and Aj > k:
```

```
            Aj+1 = Aj
```

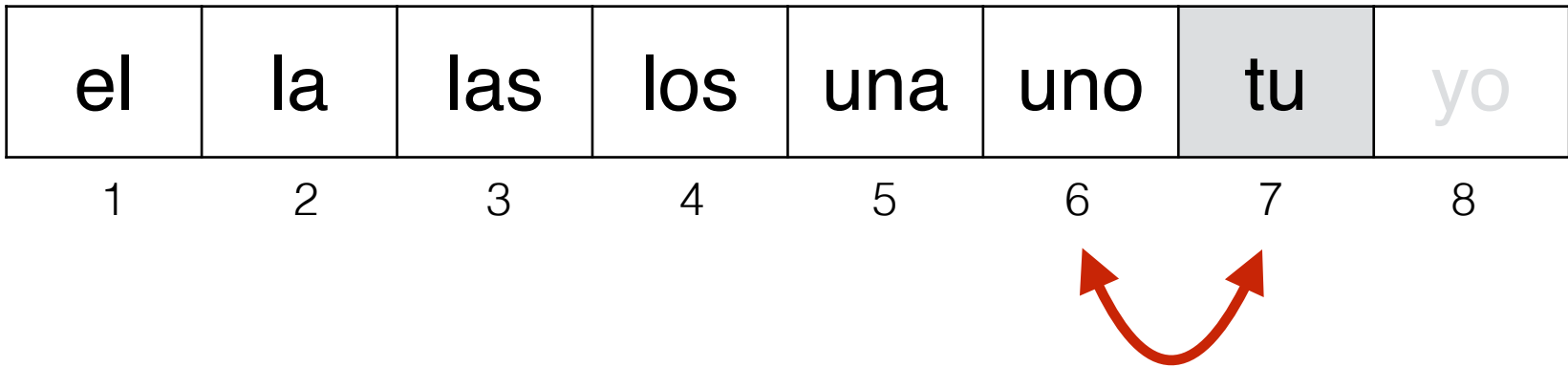
```
            Aj = k
```

```
            j = j - 1
```

**Idea:** Insertar  $A_i$  en la sublista ordenada  $A_1$  hasta  $A_j$

# Ordenamiento por Inserción

- **Iteración i:** intercambiar repetidamente  $a_i$  si el elemento a la izquierda es menor
- **Propiedad:** después de  $i$ -ésima iteración, los elementos entre  $a_0$  y  $a_i$  están ordenados.



**swap**

		a							
i	j	1	2	3	4	5	6	7	8
		la	el	uno	los	las	una	tu	yo
2	1	la	el	los	las	las	una	tu	yo
3	2	el	la	uno	las	las	una	tu	yo
4	3	el	la	uno	los	las	una	tu	yo
4	2	el	la	los	uno	las	una	tu	yo
5	4	el	la	los	uno	las	una	tu	yo
5	3	la	la	los	las	uno	una	tu	yo
5	2	el	la	las	los	uno	una	tu	yo
6	5	el	la	las	los	uno	una	tu	yo
6	4	el	la	las	los	una	uno	tu	yo
7	6	el	la	las	los	una	uno	tu	yo

```
def insertionsort(A):
```

```
    for i=2 to N:
```

```
        k = Ai
```

```
        j = i-1
```

```
        while j >= 1 and Aj > k:
```

```
            Aj+1 = Aj
```

```
            Aj = k
```


```
            j = j - 1
```

**Idea:** Insertar  $A_i$  en la sublista ordenada  $A_1$  hasta  $A_j$

# Ordenamiento por Inserción

- **Iteración i:** intercambiar repetidamente  $a_i$  si el elemento a la izquierda es menor
- **Propiedad:** después de  $i$ -ésima iteración, los elementos entre  $a_0$  y  $a_i$  están ordenados.

el	la	las	los	una	tu	uno	yo
1	2	3	4	5	6	7	8



**swap** →

i	j	1	2	3	4	5	6	7	8
		la	el	uno	los	las	una	tu	yo
2	1	la	el	los	las	las	una	tu	yo
3	2	el	la	uno	las	las	una	tu	yo
4	3	el	la	uno	los	las	una	tu	yo
4	2	el	la	los	uno	las	una	tu	yo
5	4	el	la	los	uno	las	una	tu	yo
5	3	la	la	los	las	uno	una	tu	yo
5	2	el	la	las	los	uno	una	tu	yo
6	5	el	la	las	los	uno	una	tu	yo
6	4	el	la	las	los	una	uno	tu	yo
7	6	el	la	las	los	una	uno	tu	yo
7	5	el	la	las	los	una	tu	uno	yo

```
def insertionsort(A):
```

```
    for i=2 to N:
```

```
        k = Ai
```

```
        j = i-1
```

```
        while j >= 1 and Aj > k:
```

```
            Aj+1 = Aj
```

```
            Aj = k
```


```
            j = j - 1
```

**Idea:** Insertar  $A_i$  en la sublista ordenada  $A_1$  hasta  $A_j$

# Ordenamiento por Inserción

- **Iteración i:** intercambiar repetidamente  $a_i$  si el elemento a la izquierda es menor
- **Propiedad:** después de  $i$ -ésima iteración, los elementos entre  $a_0$  y  $a_i$  están ordenados.

el	la	las	los	tu	una	uno	yo
1	2	3	4	5	6	7	8



**swap** →

i	j	1	2	3	4	5	6	7	8
		la	el	uno	los	las	una	tu	yo
2	1	la	el	los	las	las	una	tu	yo
3	2	el	la	uno	las	las	una	tu	yo
4	3	el	la	uno	los	las	una	tu	yo
4	2	el	la	los	uno	las	una	tu	yo
5	4	el	la	los	uno	las	una	tu	yo
5	3	la	la	los	las	uno	una	tu	yo
5	2	el	la	las	los	uno	una	tu	yo
6	5	el	la	las	los	uno	una	tu	yo
6	4	el	la	las	los	una	uno	tu	yo
7	6	el	la	las	los	una	uno	tu	yo
7	5	el	la	las	los	una	tu	uno	yo
7	4	el	la	las	los	tu	una	uno	yo

```
def insertionsort(A):
```

```
    for i=2 to N:
```

```
        k = Ai
```

```
        j = i-1
```

```
        while j >= 1 and Aj > k:
```

```
            Aj+1 = Aj
```

```
            Aj = k
```

```
            j = j - 1
```

**Idea:** Insertar  $A_i$  en la sublista ordenada  $A_1$  hasta  $A_j$

# Ordenamiento por Inserción

- **Iteración i:** intercambiar repetidamente  $a_i$  si el elemento a la izquierda es menor
- **Propiedad:** después de  $i$ -ésima iteración, los elementos entre  $a_0$  y  $a_i$  están ordenados.

el	la	las	los	tu	una	uno	yo
1	2	3	4	5	6	7	8

**swap** →

i	j	1	2	3	4	5	6	7	8
		la	el	uno	los	las	una	tu	yo
2	1	la	el	los	las	las	una	tu	yo
3	2	el	la	uno	las	las	una	tu	yo
4	3	el	la	uno	los	las	una	tu	yo
4	2	el	la	los	uno	las	una	tu	yo
5	4	el	la	los	uno	las	una	tu	yo
5	3	la	la	los	las	uno	una	tu	yo
5	2	el	la	las	los	uno	una	tu	yo
6	5	el	la	las	los	uno	una	tu	yo
6	4	el	la	las	los	una	uno	tu	yo
7	6	el	la	las	los	una	uno	tu	yo
7	5	el	la	las	los	una	tu	uno	yo
7	4	el	la	las	los	tu	una	uno	yo
8	7	el	la	las	los	tu	una	uno	yo

```
def insertionsort(A):
```

```
    for i=2 to N:
```

```
        k = Ai
```

```
        j = i-1
```

```
        while j >= 1 and Aj > k:
```

```
            Aj+1 = Aj
```

```
            Aj = k
```

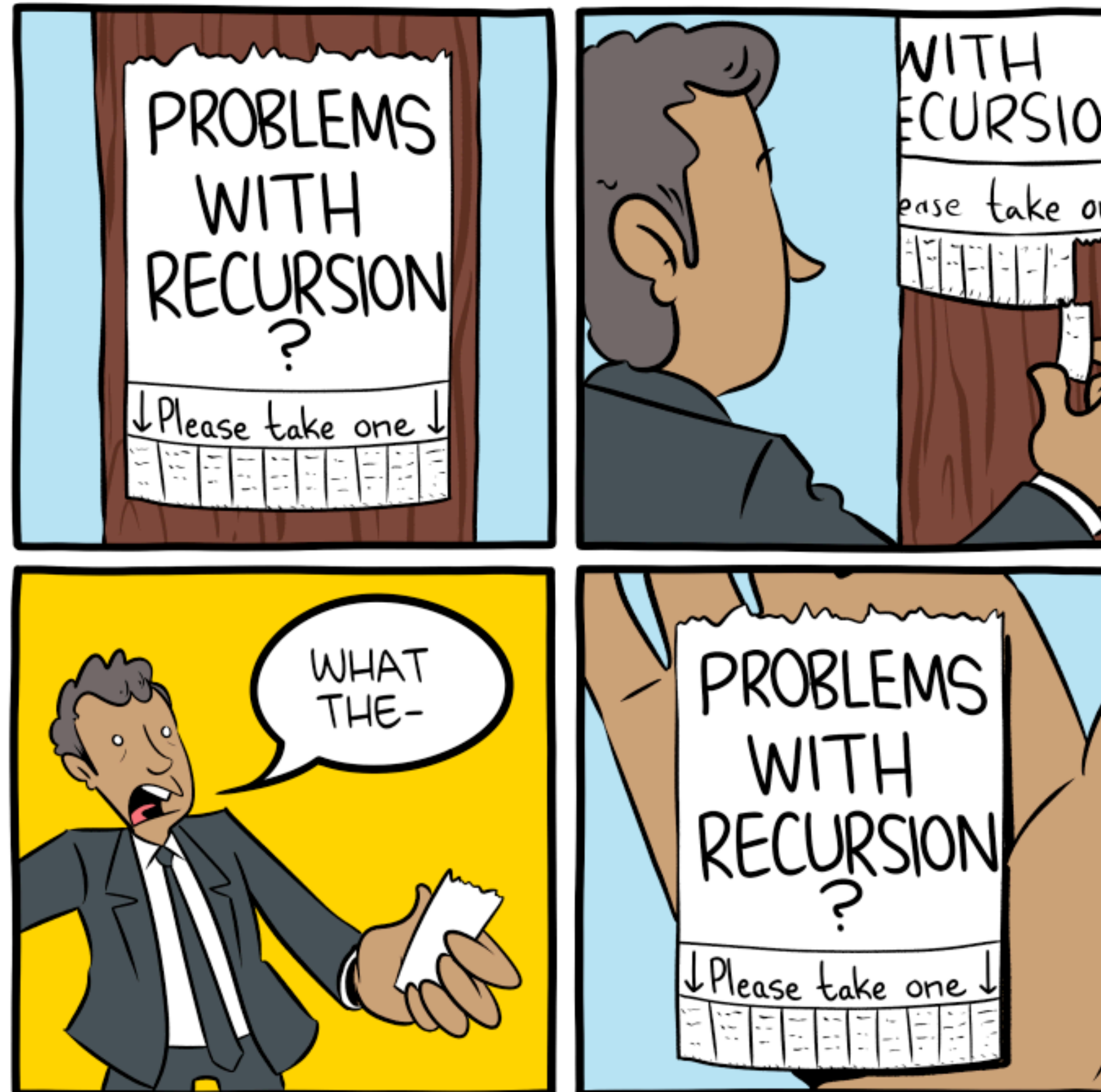
```
            j = j - 1
```

**Idea:** Insertar  $A_i$  en la sublista ordenada  $A_1$  hasta  $A_j$

# Actividad

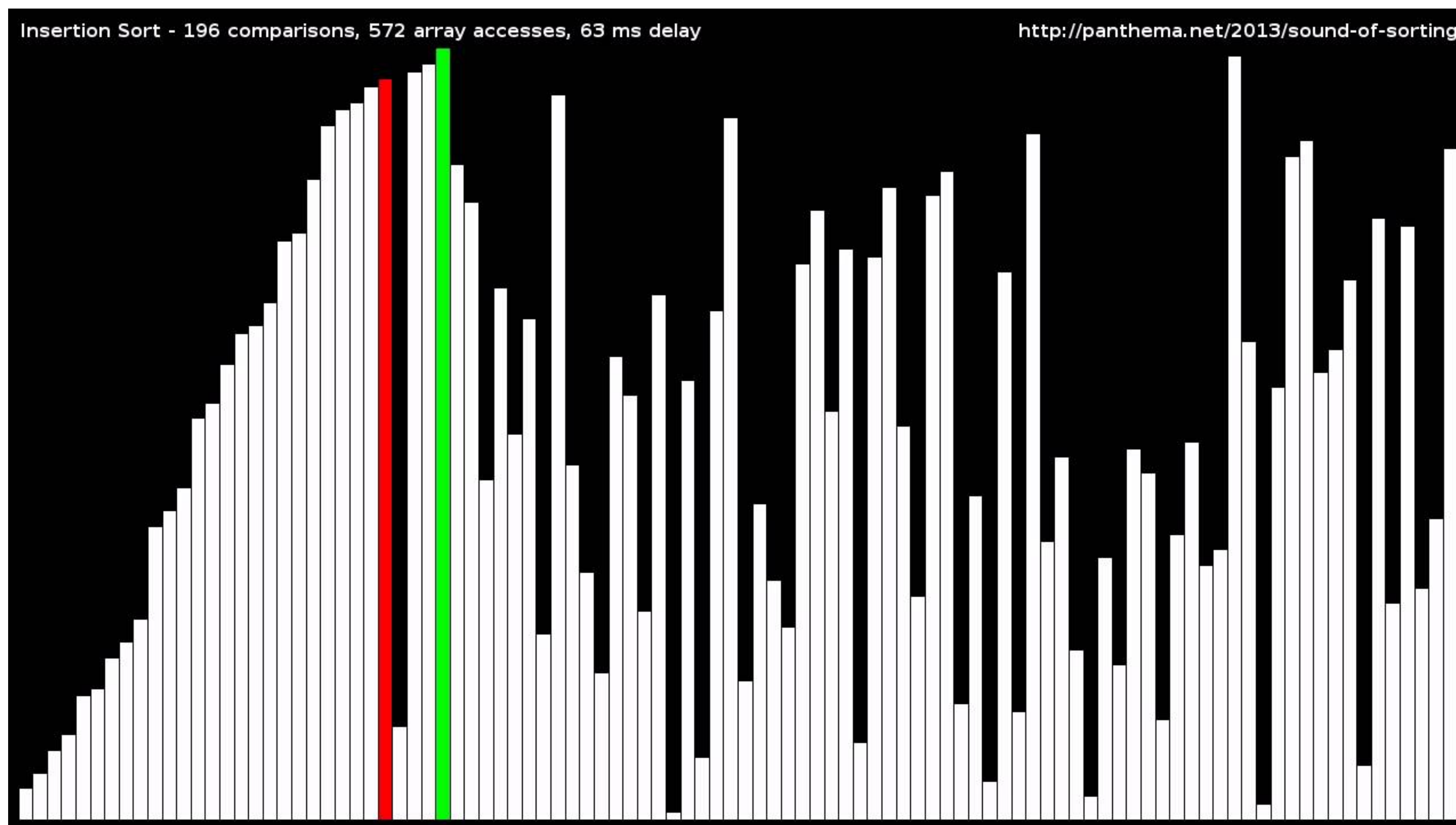
- Haga el diagrama de flujo del algoritmo insertionsort.
- Cuántas comparaciones  $A_j > k$  (en el ciclo while) se ejecutan para estas 3 entradas:
  1.  $A = 5,4,3,2,1$
  2.  $A = 1,2,3,4,5$
  3.  $A = 1,2,5,4,3$
- Con respecto a la pregunta anterior, ¿cuál diría que es el “mejor caso” y “peor caso” del algoritmo? (entendiendo que entre más comparaciones se realizan, más tiempo tomará ejecutar el algoritmo). Utilice la estrategia de resolución de problemas de George Polya.
- Si el tamaño de la lista  $A$  es  $N$ , cuántas comparaciones se realizan en el mejor caso? y en el peor caso? Utilice la estrategia de resolución de problemas de George Polya.





[smbc-comics.com](http://www.smbc-comics.com)





<https://www.youtube.com/watch?v=8oJS1BMKE64>

```
def insertionsort(A):  
    for i=2 to N:  
        k = Ai  
        j = i-1  
        while j >= 1 and Aj > k:  
            Aj+1 = Aj  
            j = j - 1  
        Aj+1 = k
```

$$\gcd(a, b) = \begin{cases} a & \text{si } b = 0 \\ \gcd(b, a \bmod b) & \text{en otro caso} \end{cases}$$