

### Sesión 3 (3ª Semana)

David Morán ([david.moran@urjc.es](mailto:david.moran@urjc.es))

Juan Quintana ([juandavid.quintana@urjc.es](mailto:juandavid.quintana@urjc.es))

Sergio Pérez ([sergio.perez.pelo@urjc.es](mailto:sergio.perez.pelo@urjc.es))

Jesús Sánchez-Oro ([jesus.sanchezoro@urjc.es](mailto:jesus.sanchezoro@urjc.es))

Isaac Lozano ([isaac.lozano@urjc.es](mailto:isaac.lozano@urjc.es))

Raúl Martín ([raul.martin@urjc.es](mailto:raul.martin@urjc.es))

# Contenidos

- Algoritmos de Ordenamiento
  - Bubble Sort
  - Quick Sort
  - Merge Sort
  - Otros algoritmos
- Búsqueda binaria
- Algoritmos Voraces

# Algoritmos de Ordenamiento

Algunos problemas requieren tener ordenados una serie de elementos para dar una respuesta

- Algoritmos de ordenación
- Estructuras de datos ordenadas

... es importante su eficiencia

# Algoritmos de Ordenamiento

- BubbleSort y SelectionSort
  - Complejidad:  $O(n^2)$  (ineficientes)
  - No se incluyen en las bibliotecas estándar
    - Se implementan como ejercicio de aprendizaje

# Algoritmos de Ordenamiento

- Bubble Sort
  - Se itera desde 0 hasta N (i)
  - Se itera desde i+1 hasta N (j)
  - Si  $\text{Arr}(i) < \text{Arr}(j)$  - Intercambiamos  $\text{Arr}(i)$  y  $\text{Arr}(j)$

# Algoritmos de Ordenamiento

4	0	2	1	3	7	9	5
---	---	---	---	---	---	---	---

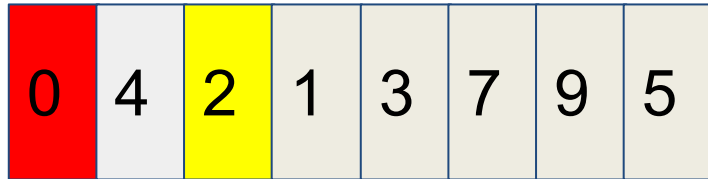
# Algoritmos de Ordenamiento



$i = 0$

$j = 1$

# Algoritmos de Ordenamiento

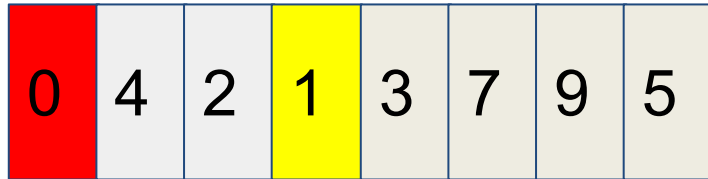


$i = 0$

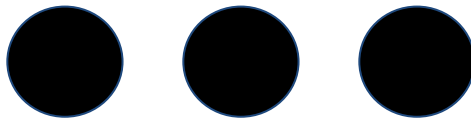
$j = 2$



# Algoritmos de Ordenamiento



# Algoritmos de Ordenamiento



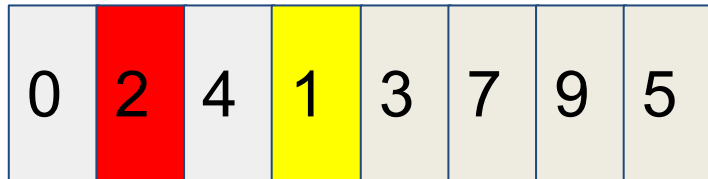
4 iteraciones más tarde...

# Algoritmos de Ordenamiento

0	4	2	1	3	7	9	5
---	---	---	---	---	---	---	---

$i = 1$   
 $j = 2$

# Algoritmos de Ordenamiento

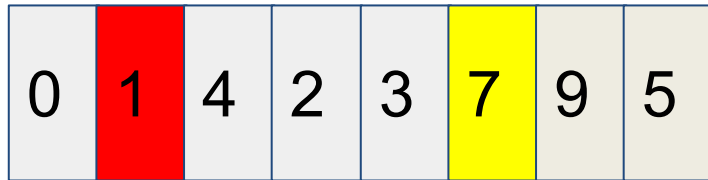


# Algoritmos de Ordenamiento



$i = 1$   
 $j = 4$

# Algoritmos de Ordenamiento



$i = 1$   
 $j = 5$

# Algoritmos de Ordenamiento



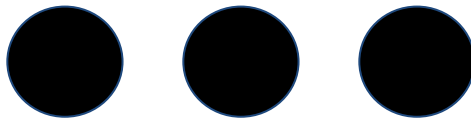
$i = 1$   
 $j = 6$

# Algoritmos de Ordenamiento





# Algoritmos de Ordenamiento



30 iteraciones más  
tarde...

# Algoritmos de Ordenamiento

0	1	2	3	4	5	7	9
---	---	---	---	---	---	---	---

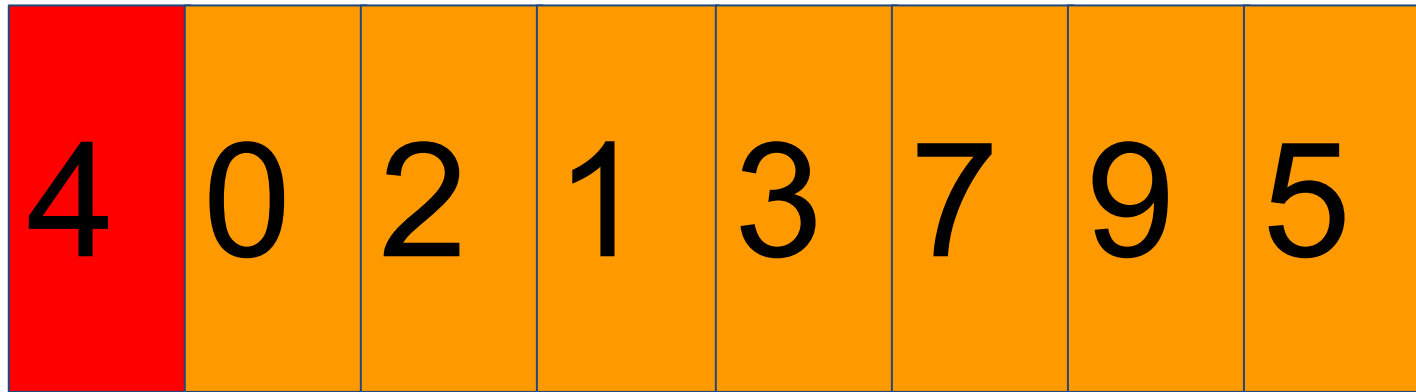
# Algoritmos de Ordenamiento

- Selection Sort
  - Se itera desde 0 hasta N (i)
    - Se declara una variable  $\text{min} = \text{Arr}(i)$ ,  $k = i$
  - Se itera desde  $i+1$  hasta N (j)
    - Si  $\text{Arr}(j) < \text{min}$ ,  $\text{min} = \text{Arr}(j)$ ,  $k = j$
  - Se intercambia la posición  $\text{Arr}(i)$  con  $\text{Arr}(k)$

# Algoritmos de Ordenamiento

4	0	2	1	3	7	9	5
---	---	---	---	---	---	---	---

# Algoritmos de Ordenamiento

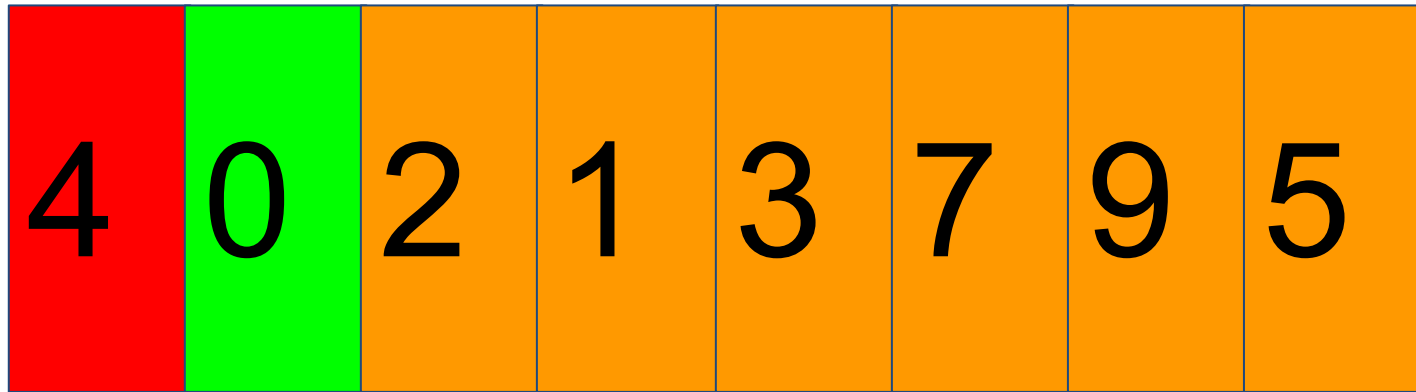


$i = 0$

$k = 0$

$\text{min} = 4$

# Algoritmos de Ordenamiento

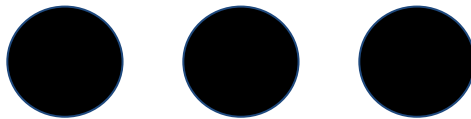


$i = 0$

$k = 1$

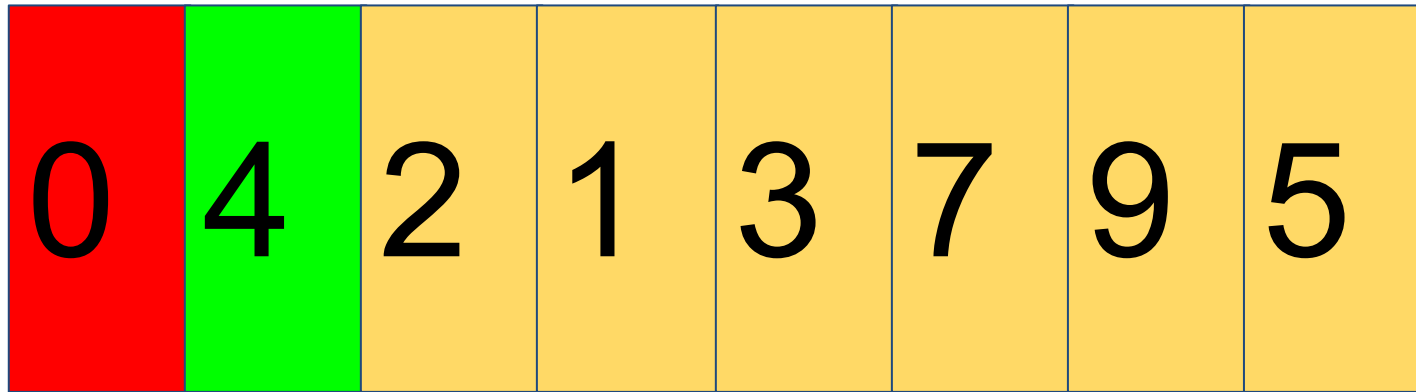
$\text{min} = 0$

# Algoritmos de Ordenamiento



6 iteraciones más tarde...

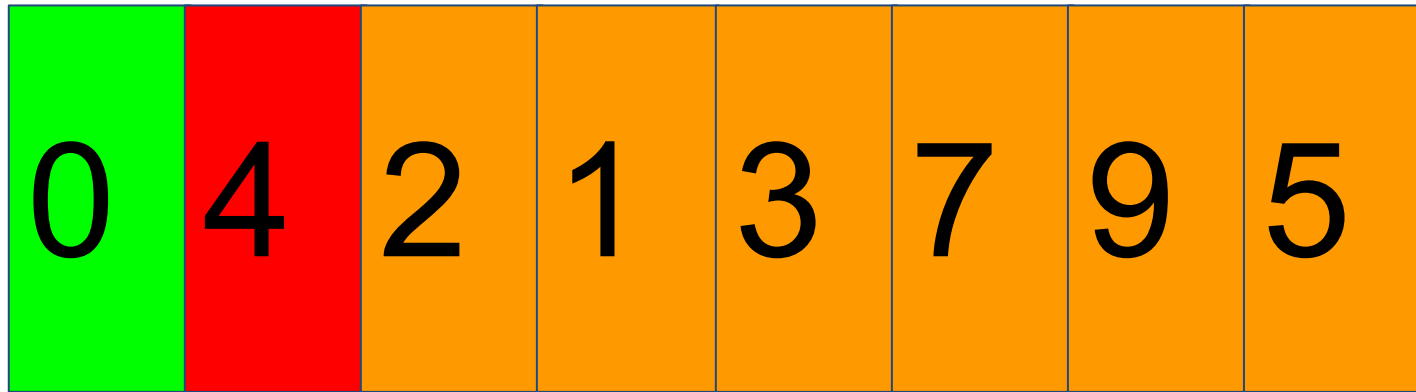
# Algoritmos de Ordenamiento



Intercambiamos



# Algoritmos de Ordenamiento

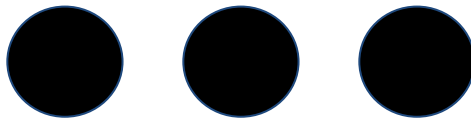


$i = 1$

$k = 1$

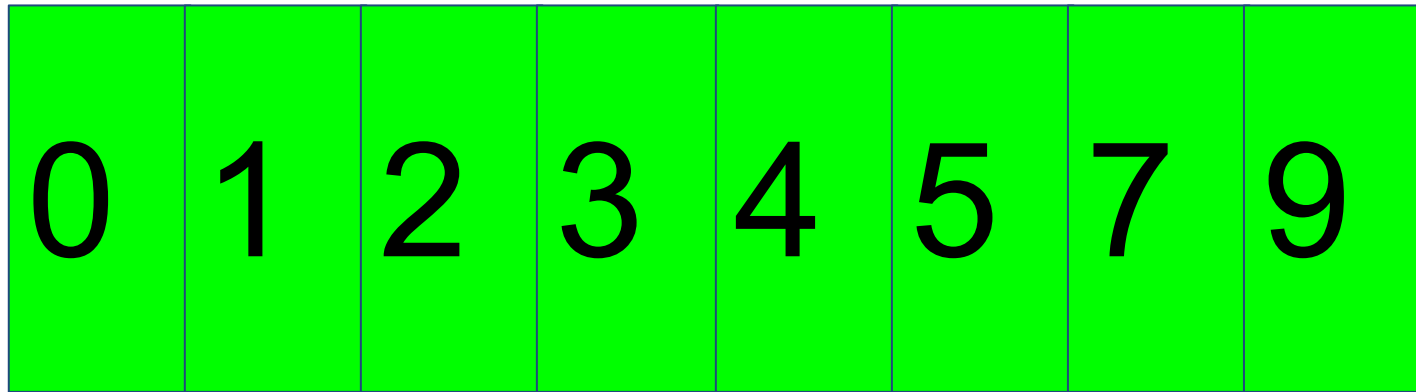
$\text{min} = 4$

# Algoritmos de Ordenamiento



42 iteraciones más tarde...

# Algoritmos de Ordenamiento



# Algoritmos de Ordenamiento

- Quicksort / Mergesort
  - Complejidad:  $O(n\log(n))$
  - Ya están implementadas en las bibliotecas básicas ¡No hay que programarlos!

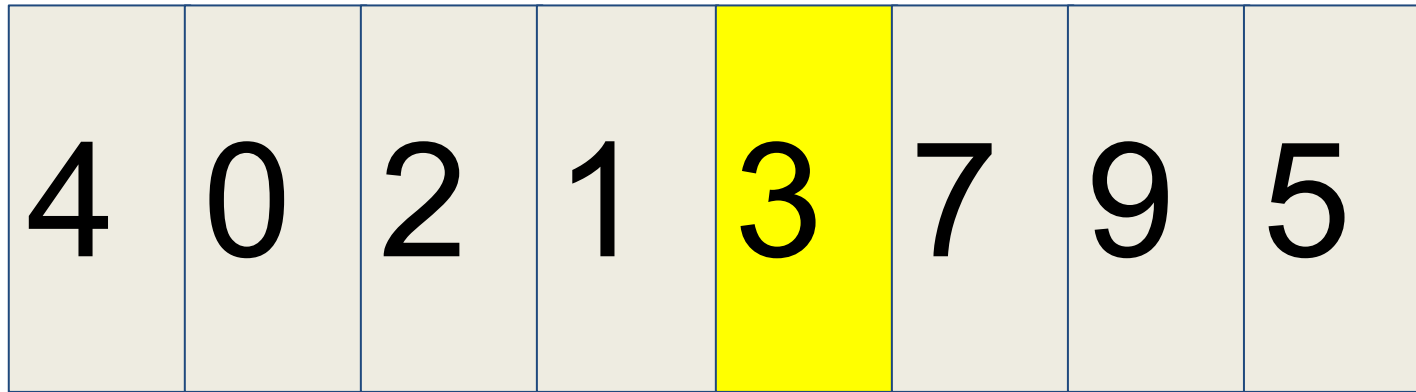
# Algoritmos de Ordenamiento

- Quicksort (idea básica)
  - Se toma un pivote “al azar” del array (un elemento cualquiera del array)
  - Dos arrays mantienen los elementos menores y mayores al pivote
  - Recursivamente se tratan estos dos arrays por separado y se concatena su resultado
  - Se repite el proceso hasta que quede 1

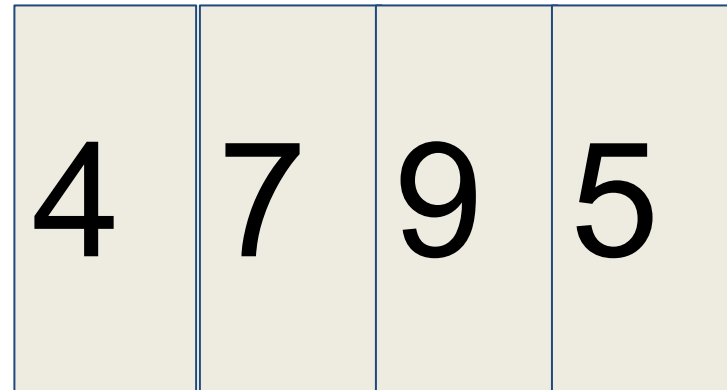
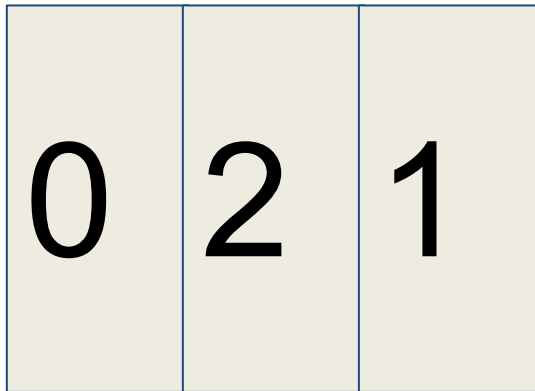
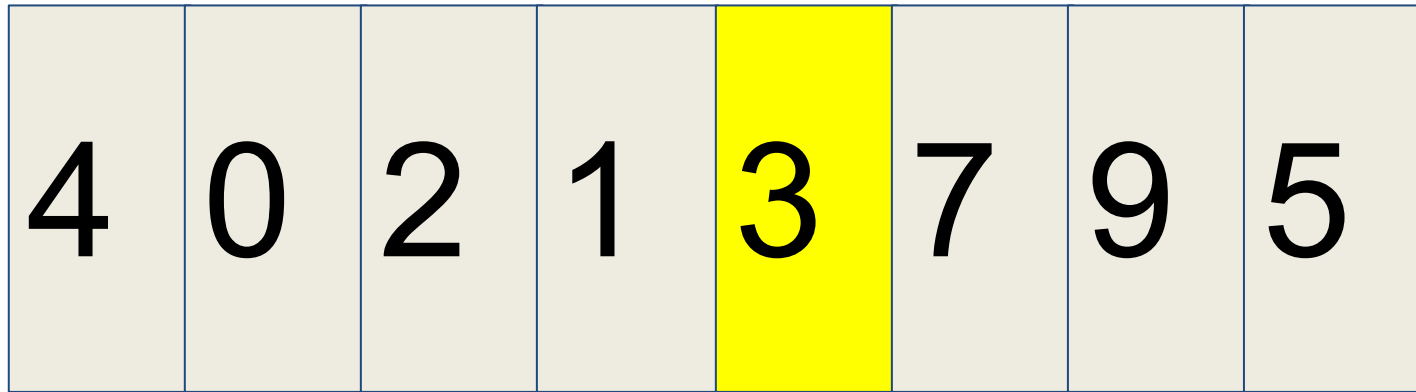
# Algoritmos de Ordenamiento

4	0	2	1	3	7	9	5
---	---	---	---	---	---	---	---

# Algoritmos de Ordenamiento



# Algoritmos de Ordenamiento

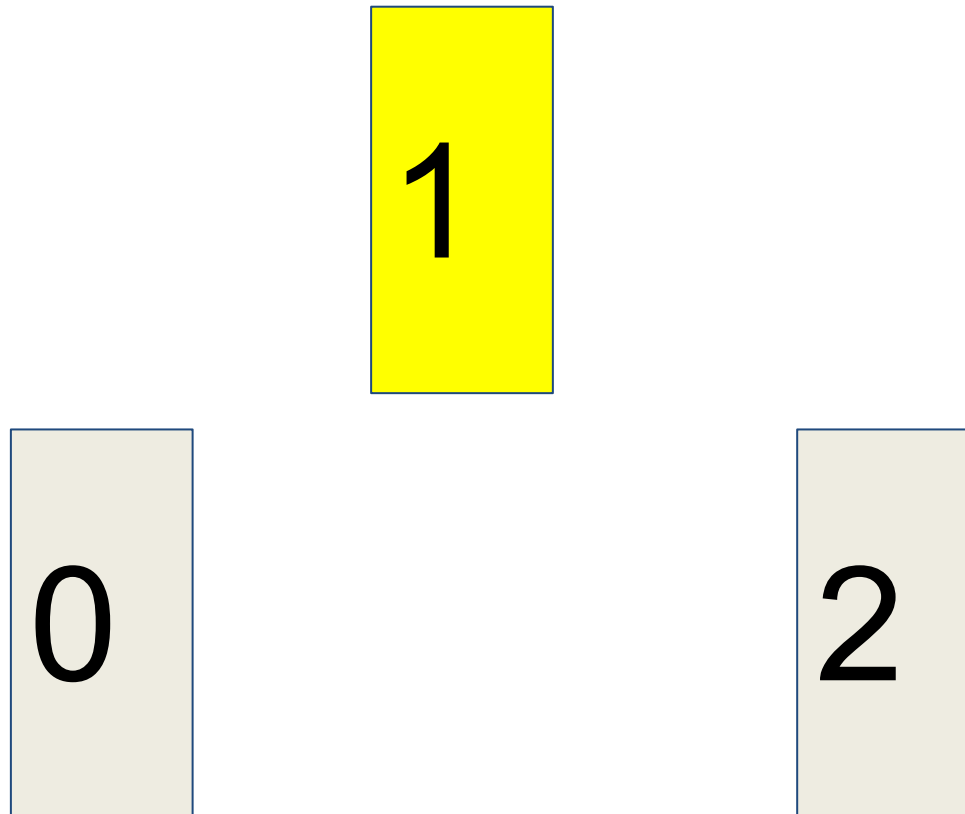




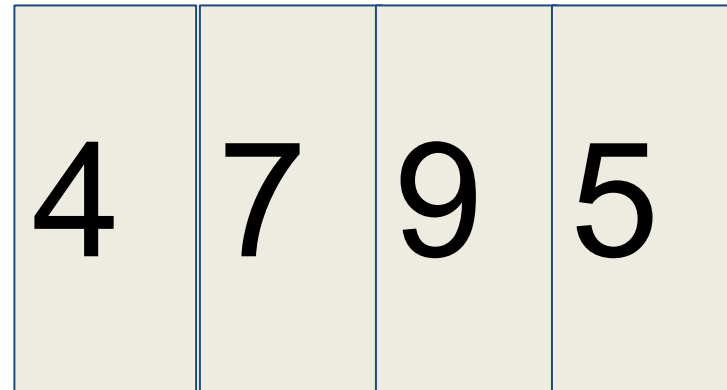
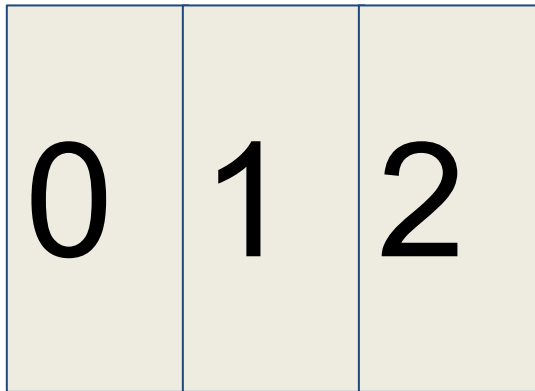
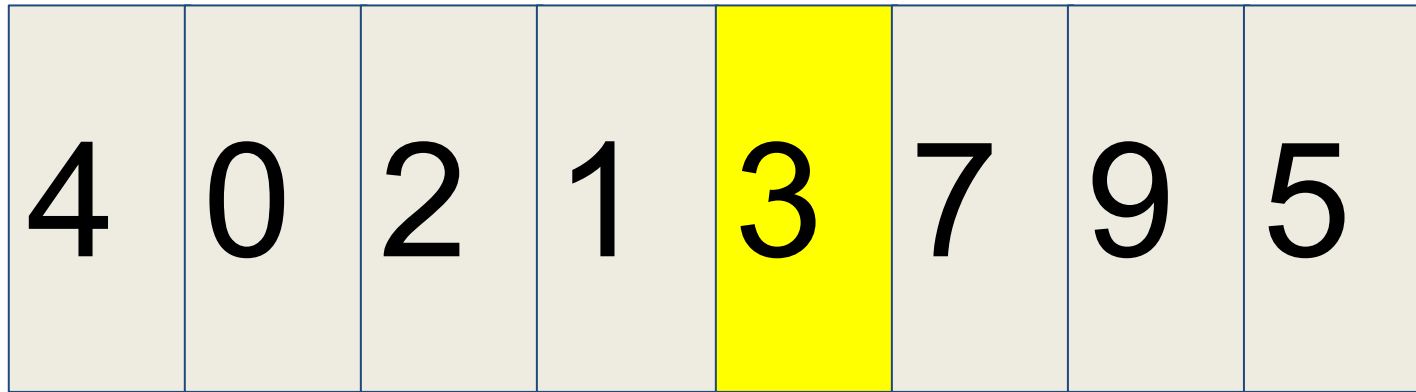
# Algoritmos de Ordenamiento



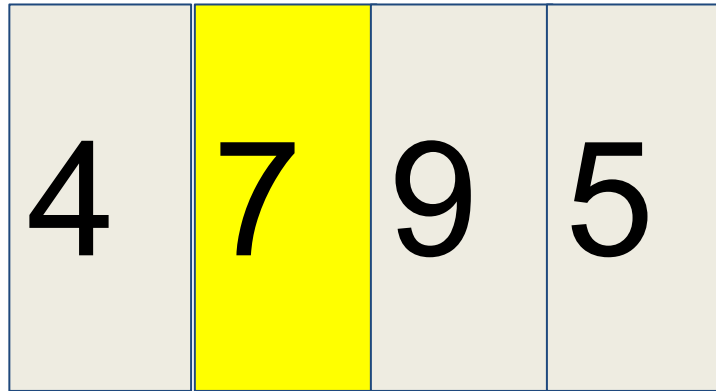
# Algoritmos de Ordenamiento



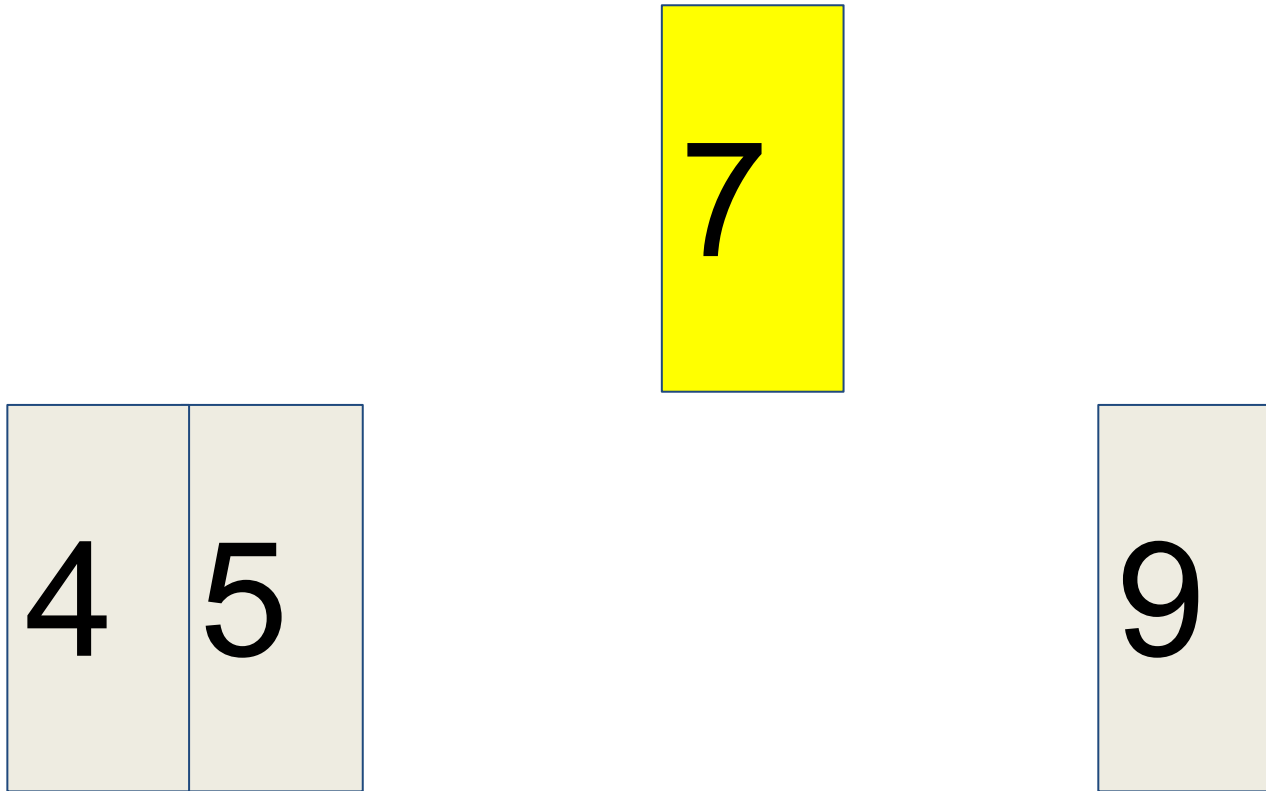
# Algoritmos de Ordenamiento



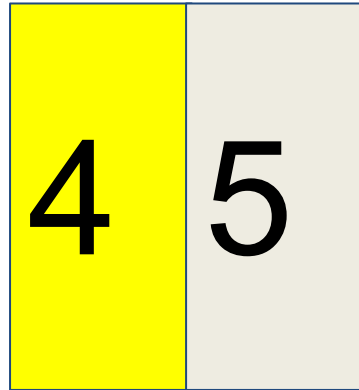
# Algoritmos de Ordenamiento



# Algoritmos de Ordenamiento



# Algoritmos de Ordenamiento

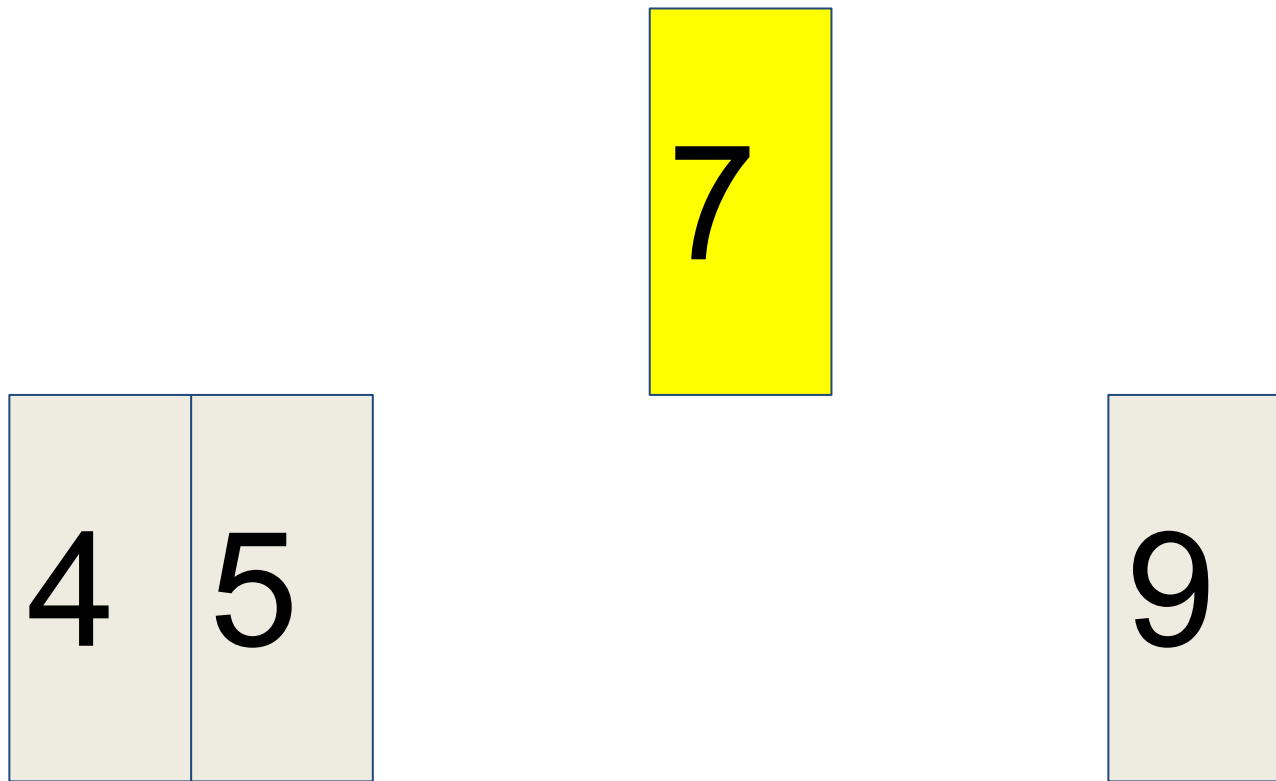


# Algoritmos de Ordenamiento

4

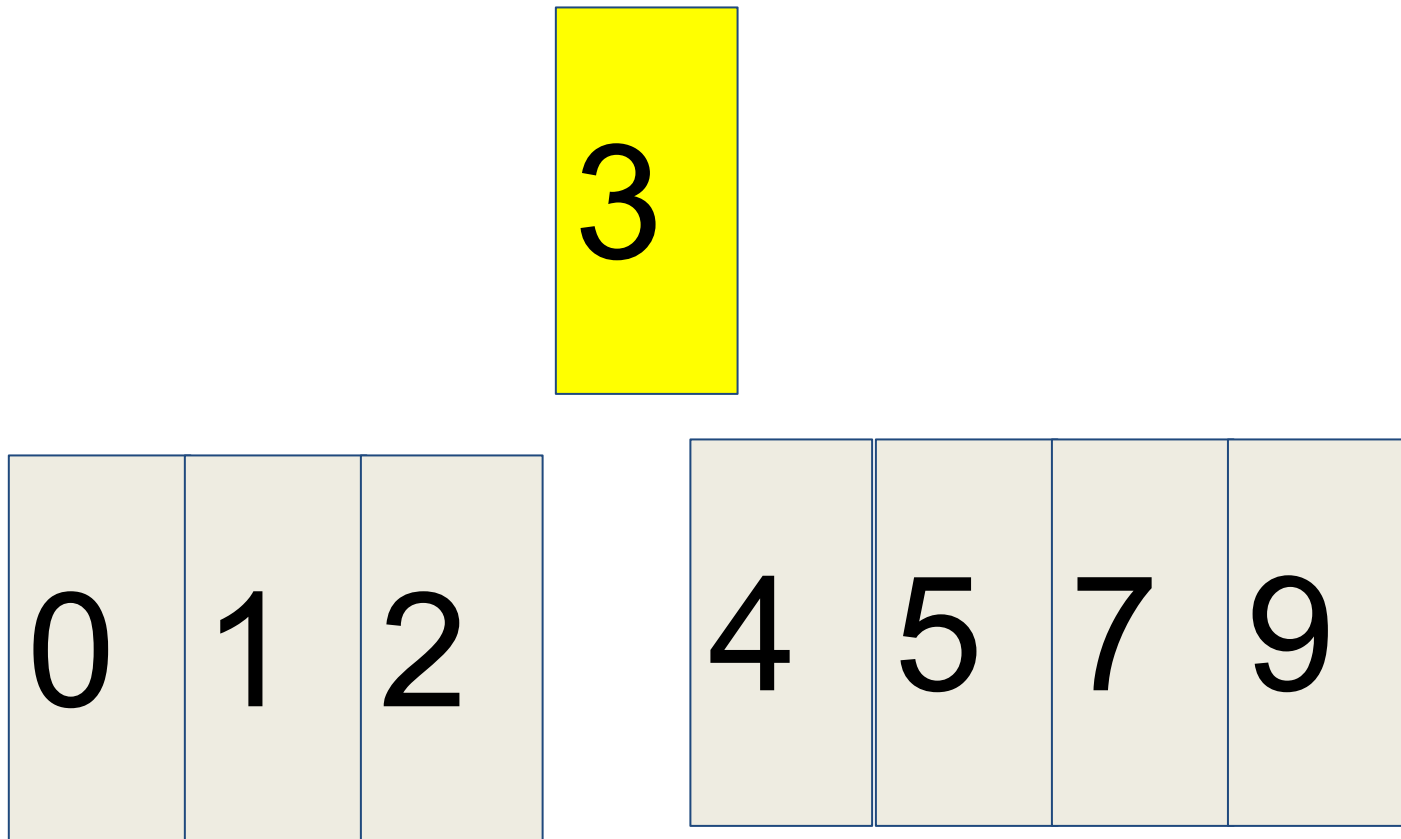
5

# Algoritmos de Ordenamiento

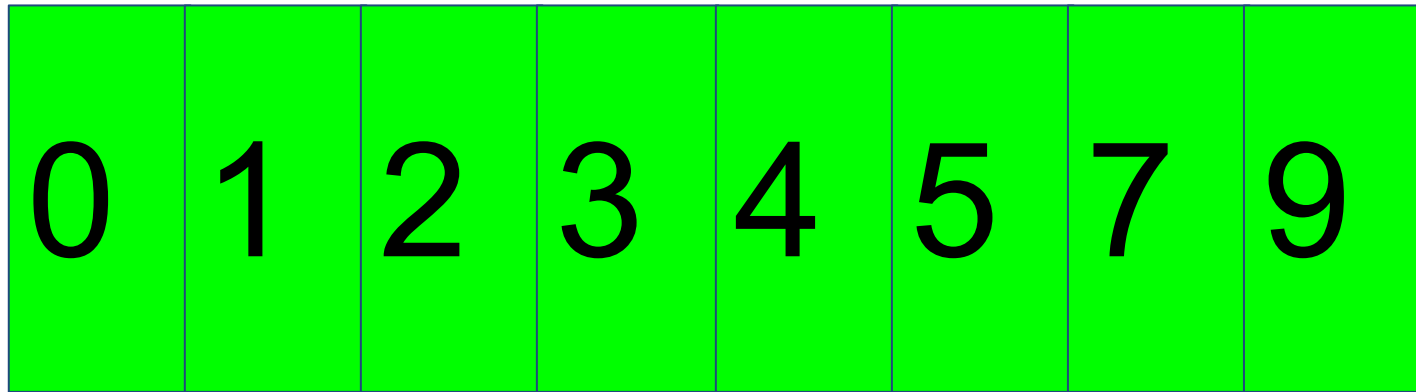




# Algoritmos de Ordenamiento



# Algoritmos de Ordenamiento



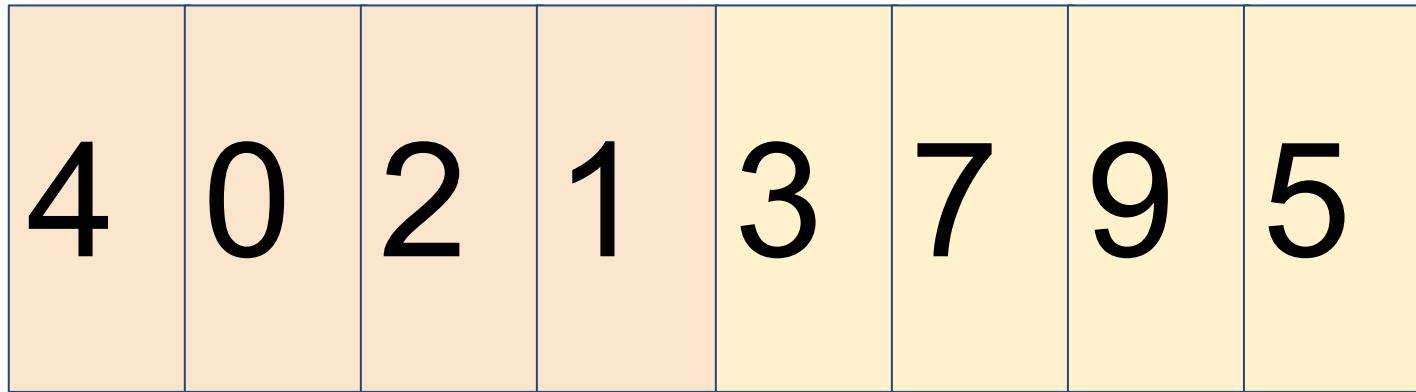
# Algoritmos de Ordenamiento

- Quicksort (idea básica)
  - Si se hace de manera ideal y perfecta, su complejidad es  $O(N \lg N)$
  - Es relativamente fácil de implementar

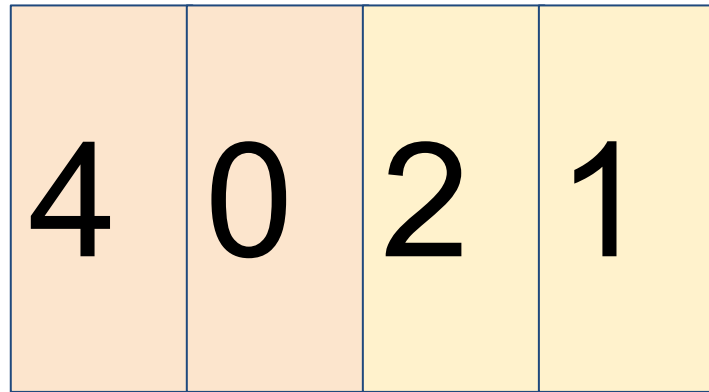
# Algoritmos de Ordenamiento

- MergeSort (idea básica)
  - Se llama recursivamente combinando los arrays desde 0 hasta  $N/2$  y de  $N/2$  hasta  $N$
  - Si el elemento tiene 1 elemento, se asume que está ordenado
  - Formar un nuevo array tomando en cuenta los dos arrays formados

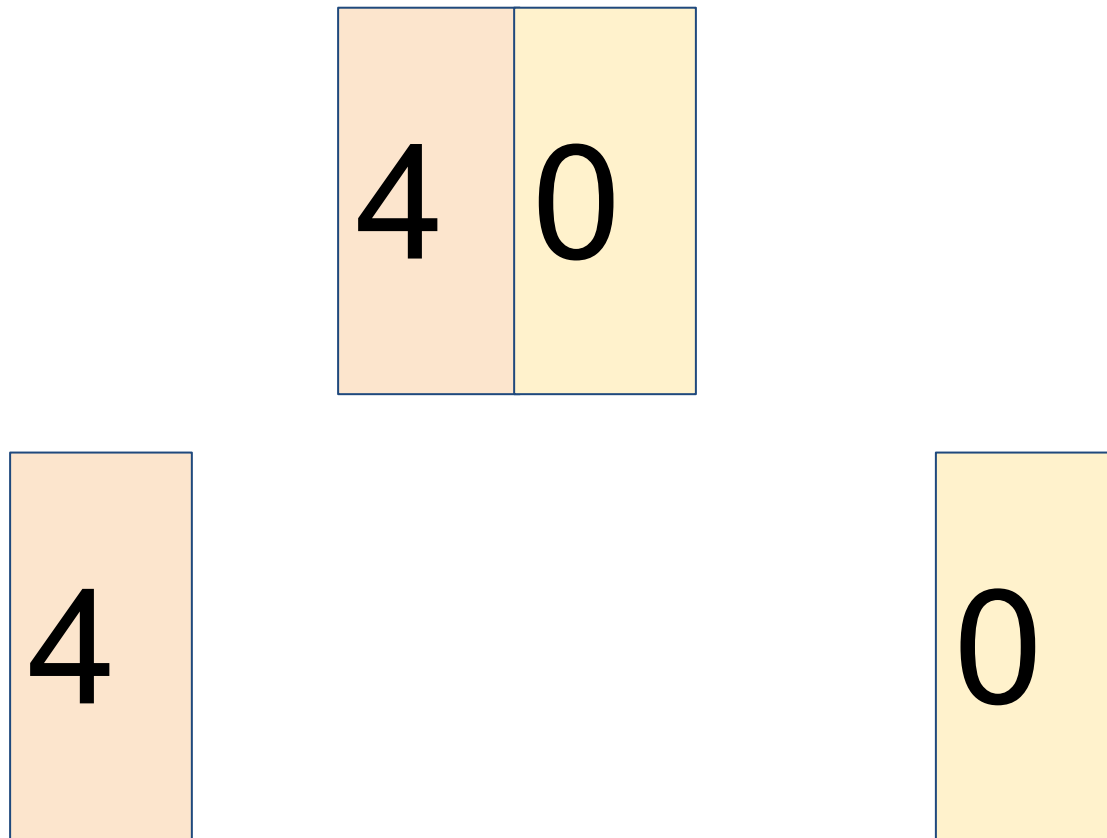
# Algoritmos de Ordenamiento



# Algoritmos de Ordenamiento



# Algoritmos de Ordenamiento

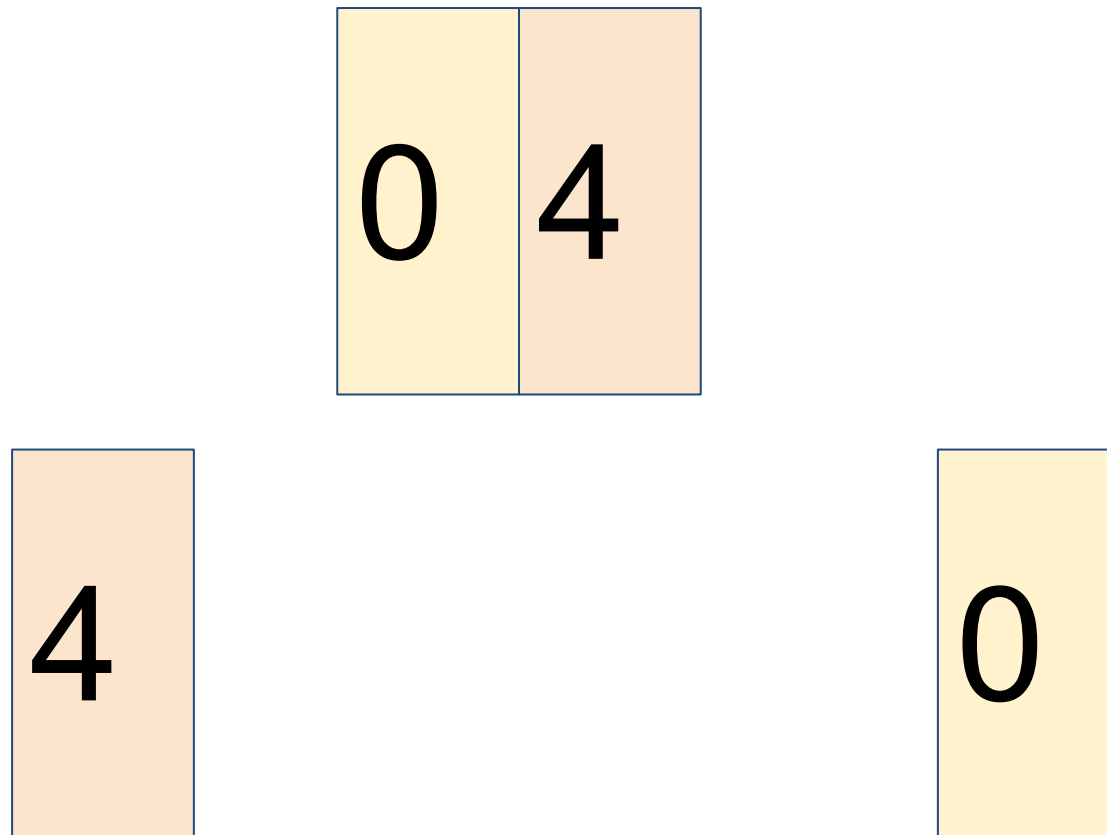


# Algoritmos de Ordenamiento

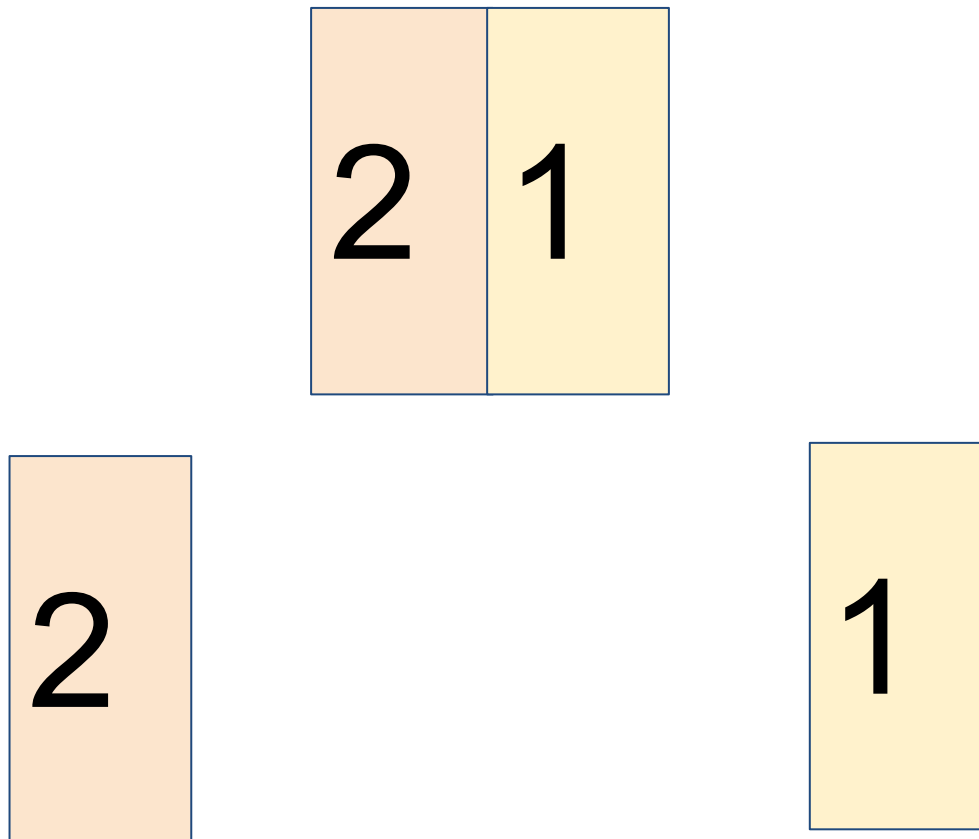
- MergeSort (idea básica)
  - Se tendrán tres arrays,  $A = [4]$ ,  $B = [0]$  y  $C$  que será el producto de la mezcla entre los dos
  - Si  $A[i] < B[j]$ , se inserta en  $C[k]$  y se suma 1 a  $k$  y a  $i$
  - Se inserta en  $C[k]$  y se suma 1 a  $k$  y a  $j$  en caso contrario



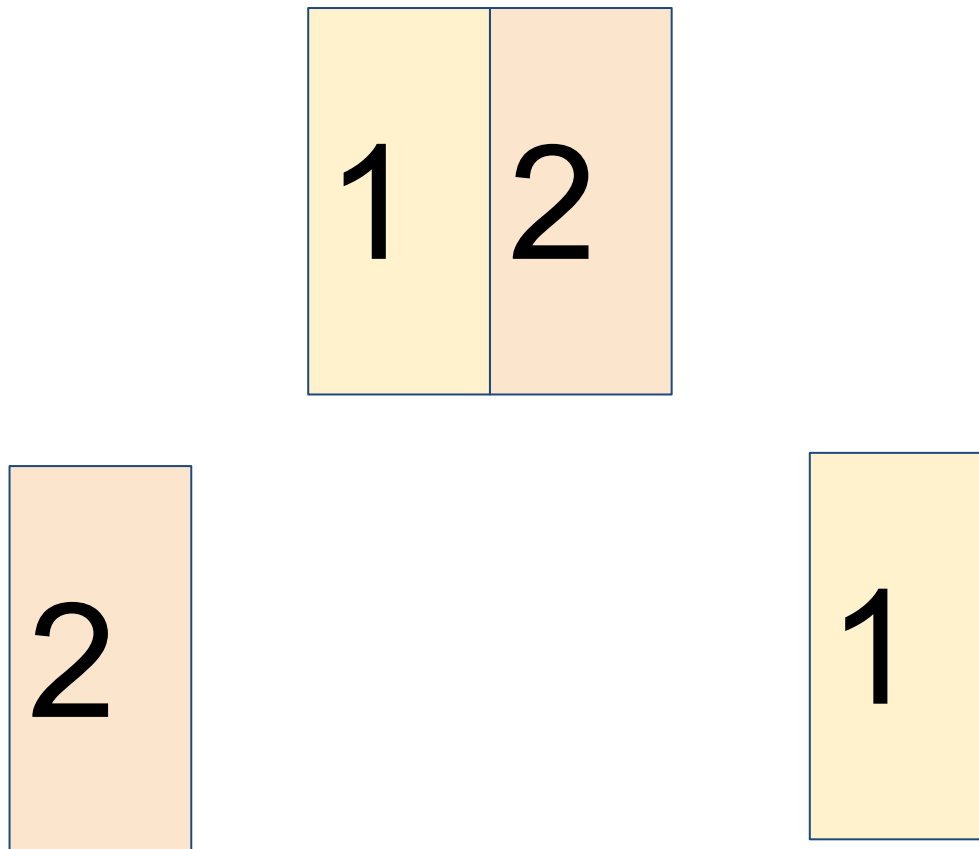
# Algoritmos de Ordenamiento



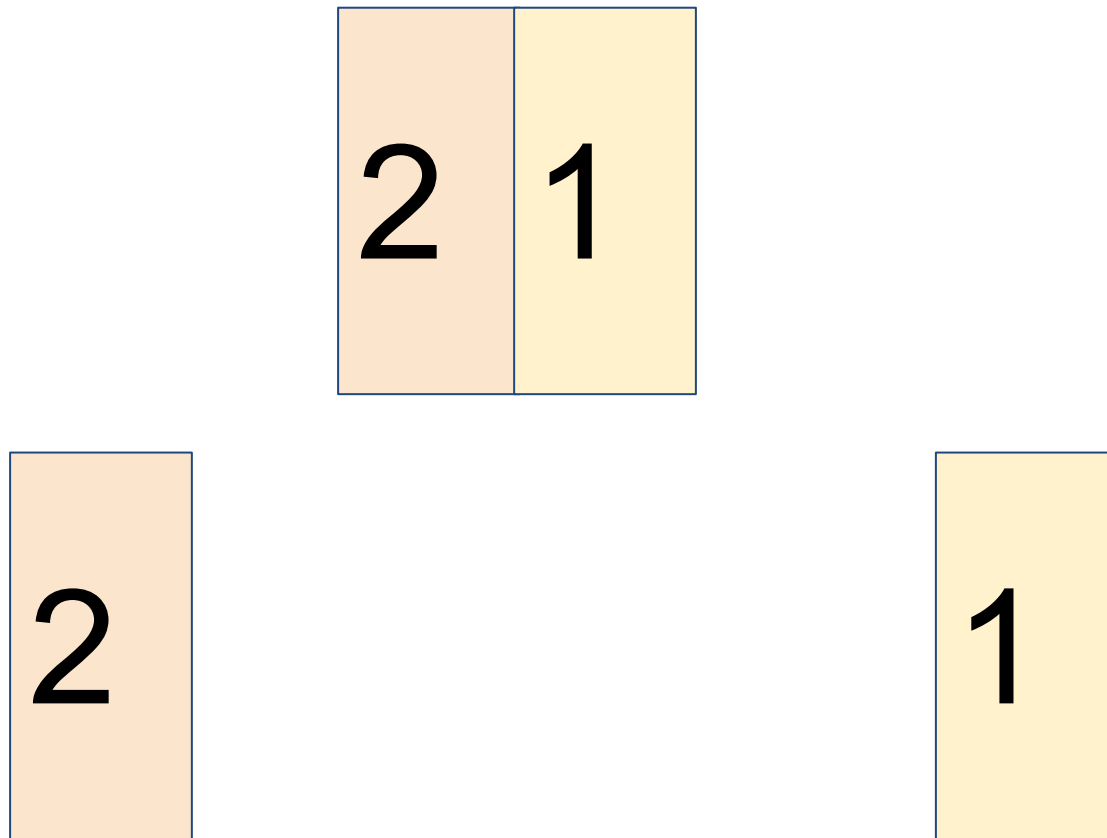
# Algoritmos de Ordenamiento



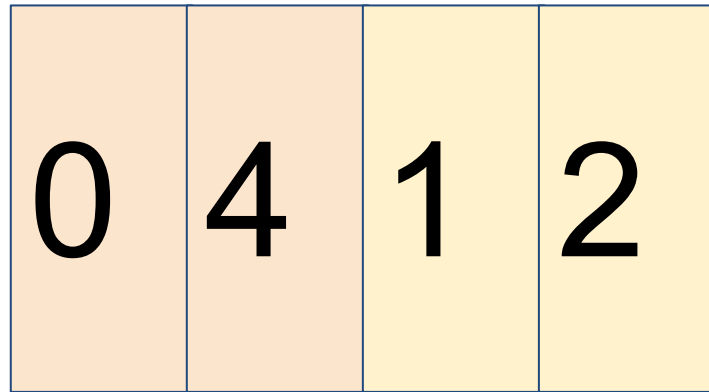
# Algoritmos de Ordenamiento



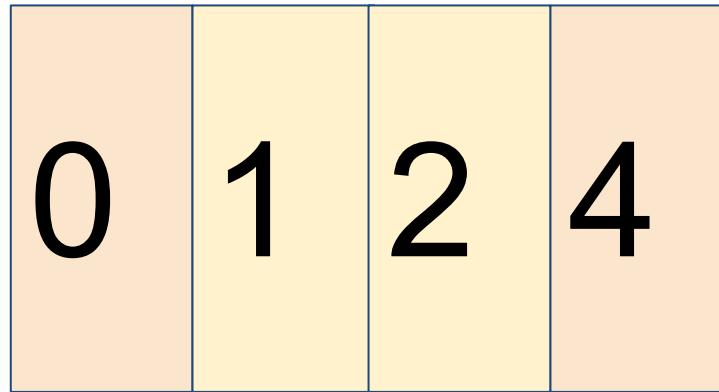
# Algoritmos de Ordenamiento



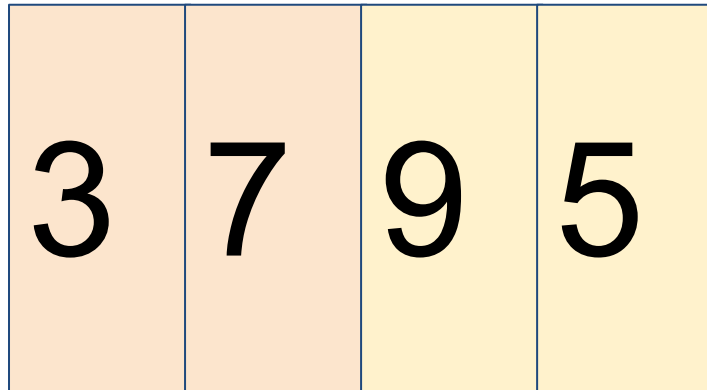
# Algoritmos de Ordenamiento



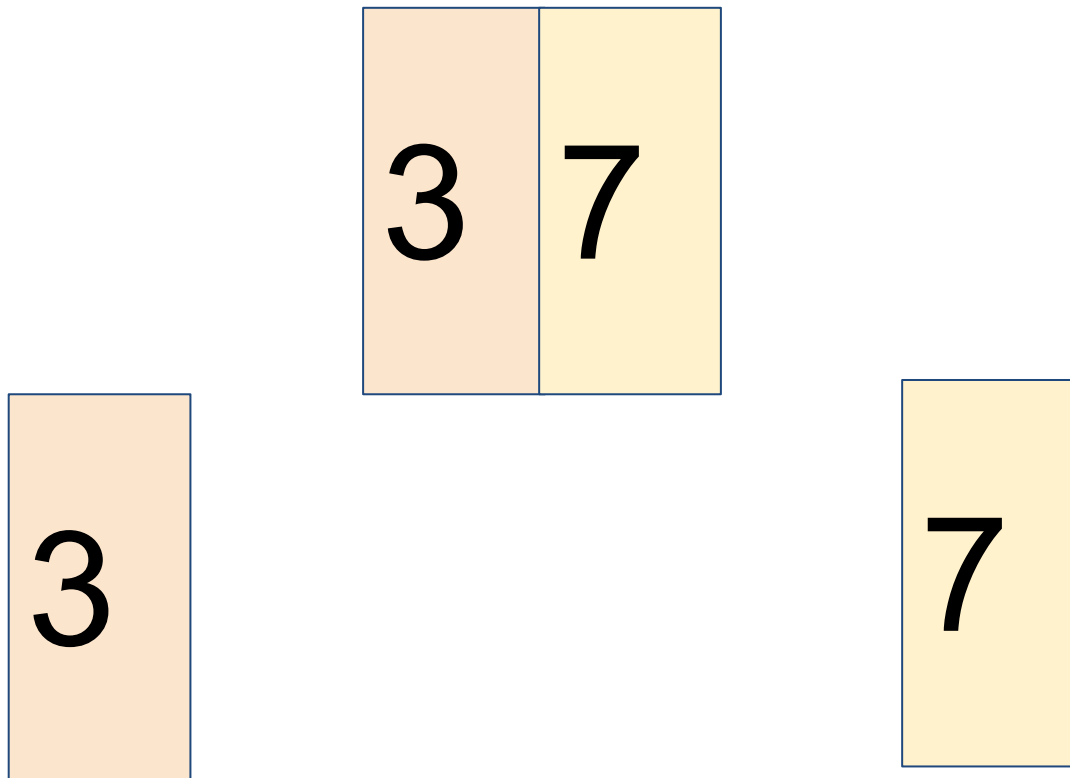
# Algoritmos de Ordenamiento



# Algoritmos de Ordenamiento

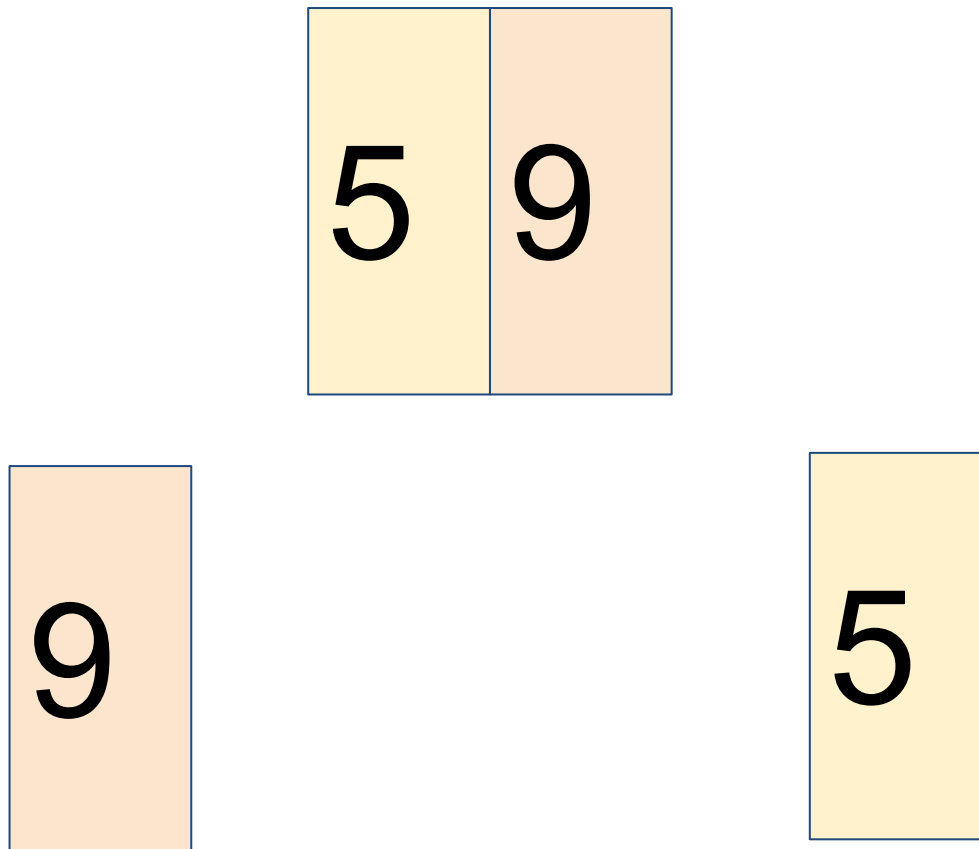


# Algoritmos de Ordenamiento

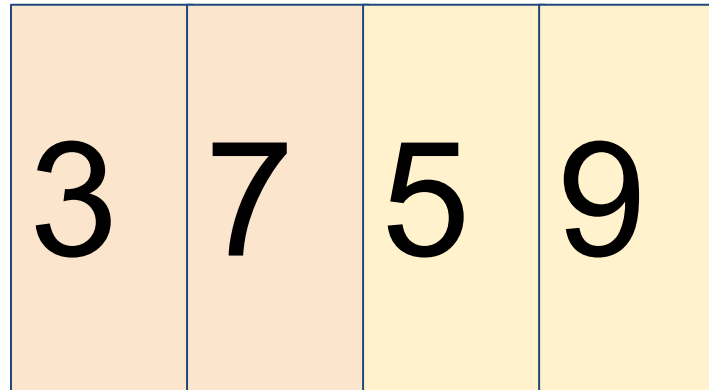




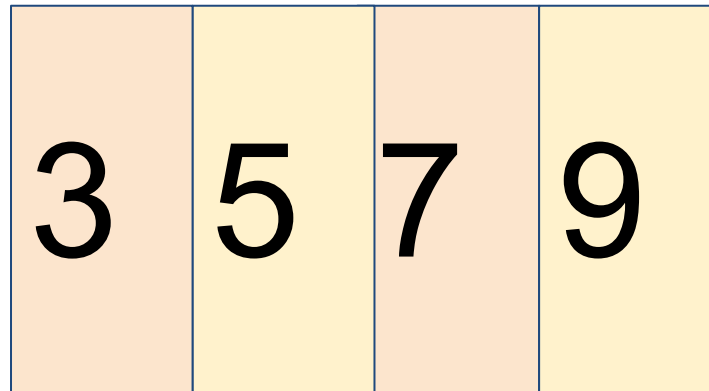
# Algoritmos de Ordenamiento



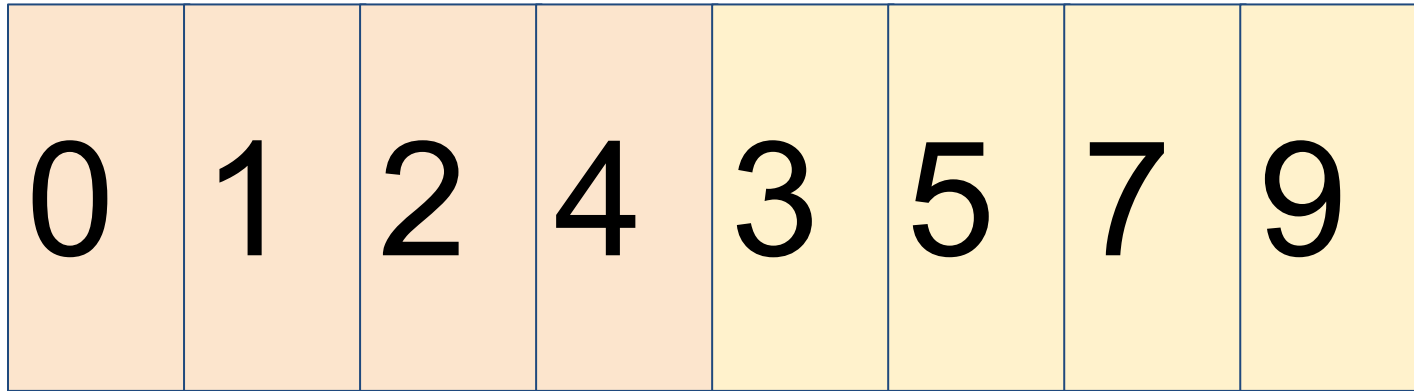
# Algoritmos de Ordenamiento



# Algoritmos de Ordenamiento



# Algoritmos de Ordenamiento



# Algoritmos de Ordenamiento

0	1	2	3	4	5	7	9
---	---	---	---	---	---	---	---

# Algoritmos de Ordenamiento

- C++
  - `std::sort()`
- Java
  - `Collections.sort()` -> MergeSort\*
  - `Arrays.sort()` -> Quicksort\* (para tipos primitivos)

\*En realidad son variantes más eficientes

# Algoritmos de Ordenamiento

- Estructuras ordenadas nos permiten hacer búsquedas “inteligentes” sobre ellas (búsqueda binaria, ternaria, etc)
  - ¡Como los árboles binarios!
- Otros algoritmos de ordenamiento (RadixSort, BucketSort)

# Búsqueda Binaria

- Definición
- Donde aplicar
- Ejemplo



# Búsqueda Binaria

- Se utiliza cuando un problema contiene una función  $f$  monótona creciente o decreciente
- Una función es monótona creciente si para cualquier  $x, y$  con  $x < y$  tal que  $f(x) \leq f(y)$ . Es monótona decreciente en el caso contrario

# Búsqueda Binaria

- La idea detrás del algoritmo es ir descartando mitades donde sabemos que no podemos encontrar la respuesta
- Ej. Si queremos encontrar un mínimo

# Búsqueda Binaria

- Acotamos la función para un  $x$  mínimo y máximo dentro de  $f(x)$
- Por cada iteración, sea  $\text{mid} = (\text{Cotamin} + \text{Cotasup}) / 2$
- Si  $f(\text{mid}) > \text{Obj}$ , significa que nos hemos pasado, por lo tanto,  $\text{Cotasup} = \text{mid}$
- Si  $f(\text{mid}) \leq \text{Obj}$ , significa que hemos subestimado, por lo tanto,  $\text{Cotainf} = \text{mid}$

# Búsqueda Binaria

- La respuesta final estará en Cotainf

```
function binary_search(f, inf, sup, t):  
    while sup-inf > 1  
        mid = (sup+inf) / 2  
        if f(mid) <= t:  
            inf = mid  
        else  
            sup = mid  
    return inf
```

# Algoritmos Voraces

- Definición
- Partes del algoritmo
- Funcionamiento
- Problemas frecuentes

# Algoritmos Voraces

## Definición

- Búsqueda eligiendo la opción más prometedora en cada paso local con la esperanza de llegar a una solución general óptima
- Rutinas muy eficientes  $O(n)$ ,  $O(n^2)$
- **NO** suelen proporcionar la solución óptima

# Algoritmos Voraces

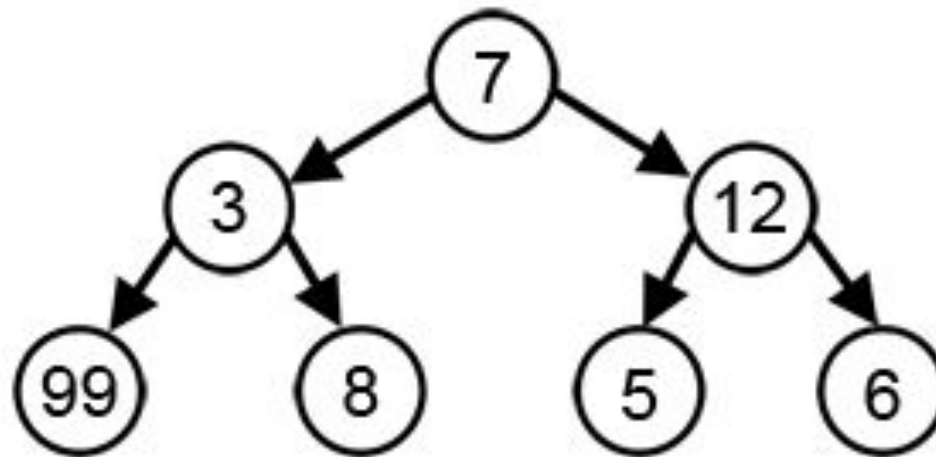
## Partes del algoritmo

- **Conjunto de candidatos (C).** Entradas del problema
- **Función solución.** Comprueba, en cada paso, si el subconjunto actual de candidatos elegidos forma una solución
- **Función de selección.** Informa cuál es el elemento más prometededor para completar la solución
- **Función de factibilidad.** Informa si a partir de un conjunto se puede llegar a una solución.
- **Función objetivo.** Es aquella que queremos maximizar o minimizar, el núcleo del problema

# Algoritmos Voraces

## Funcionamiento

Algoritmo que busca el camino de mayor peso





# Algoritmos Voraces

## SPOJ\_STAMPS

- <http://www.spoj.com/problems/STAMPS/>
- Lucy quiere superar en número la colección de sellos de Raymond
- Para ello, pide sellos a sus amigos
- Entrada:  $n^\circ$  de escenarios (casos de prueba),  $n^\circ$  de sellos necesarios para alcanzar a Raymond,  $n^\circ$  de amigos que nos dejarán sellos, lista de los sellos que nos dejará cada uno

# Algoritmos Voraces

## SPOJ\_STAMPS

- **Conjunto de candidatos:** lista de sellos que nos dejará cada amigo
- **Función solución:** Comprueba si hemos superado o no los sellos de Raymond
- **Función de selección:** Entre todos los amigos, escogeremos primero los que más sellos nos presten
- **Función de factibilidad:** ¿Tenemos suficientes sellos entre todos los amigos para superar a Raymond?
- **Función objetivo:** Minimizar el número de amigos necesarios para alcanzar a Raymond

# Algoritmos Voraces

## SPOJ\_STAMPS: Ejemplo

- **Objetivo:** Llegar a 100 sellos con 6 amigos (100 6)
- **Lista de candidatos:** 13 17 42 9 23 57
- **Función solución:** suma $\geq$ needed
- **Función de selección:** Escoger uno a uno los elementos del Array ordenado de mayor a menor
- **Función de factibilidad:** ¿suma $\leq$ needed?
- **Función objetivo:** Minimizar el número de amigos necesarios para alcanzar a Raymond
- **Solución (Java):** <https://pastebin.com/2LDxFwR9>

# ¡Hasta la próxima semana!

Ante cualquier duda sobre el curso o sobre los problemas podéis escribirnos (copia a los cuatro)

David Morán ([david.moran@urjc.es](mailto:david.moran@urjc.es))

Juan Quintana ([juandavid.quintana@urjc.es](mailto:juandavid.quintana@urjc.es))

Sergio Pérez ([sergio.perez.pelo@urjc.es](mailto:sergio.perez.pelo@urjc.es))

Jesús Sánchez-Oro ([jesus.sanchezoro@urjc.es](mailto:jesus.sanchezoro@urjc.es))

Isaac Lozano ([isaac.lozano@urjc.es](mailto:isaac.lozano@urjc.es))

Raúl Martín ([raul.martin@urjc.es](mailto:raul.martin@urjc.es))