# CURSO DE PROGRAMACIÓN COMPETITIVA URJC - 2023

## Sesión 3 (3ª Semana)


Organizadores:

- Isaac Lozano (isaac.lozano@urjc.es)
- Raúl Martín(raul.martin@urjc.es)
- Sergio Salazar (s.salazarc.2018@alumnos.urjc.es)
- Francisco Tórtola (f.tortola.2018@alumnos.urjc.es)
- **Cristian Pérez(c.perezc.2018@alumnos.urjc.es)**
- Xuqiang Liu(x.liu1.2020@alumnos.urjc.es)
- Alicia Pina(a.pinaz.2020@alumnos.urjc.es)
- Sara García(s.garciarod.2020@alumnos.urjc.es)
- Raúl Fauste(r.fauste.2020@alumnos.urjc.es)

# Siguiente clase ONLINE

## Contest Entrenamiento AdaByron 2023

This contest is currently active.

| | |
|---|---|
| **CID** | c16 |
| **Short name** | ENTRENAMIENTO_ADABYRON |
| **Name** | Entrenamiento AdaByron 2023 |
| **Activate time** | 2023-03-02 10:20:00 Europe/Amsterdam ✓ |
| **Start time** | 2023-03-10 17:30:00 Europe/Amsterdam |
| **Scoreboard freeze** | 2023-03-10 18:30:00 Europe/Amsterdam |
| **End time** | 2023-03-10 19:00:00 Europe/Amsterdam |
| **Scoreboard unfreeze** | 2023-03-10 19:00:00 Europe/Amsterdam |
| **Deactivate time** | 2023-03-10 23:59:59 Europe/Amsterdam |
| **Process balloons** | No |
| **Process medals** | Yes |
| **Medals** | 3 different types of medals (Show/Hide details) < |

Mínimo 1 envío correcto para verificar la asistencia

# Contenidos

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

Divide y vencerás

# 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 Arr(i) > Arr(j) - Intercambiamos Arr(i) y Arr(j)

# Algoritmos de Ordenamiento

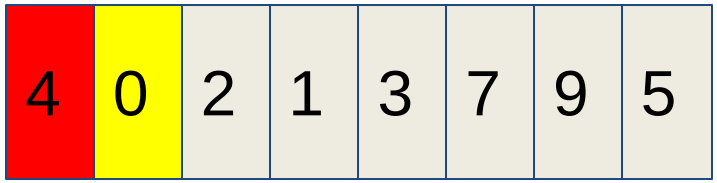
- Bubble Sort (pseudocódigo)

```
fun bubblesort(arr):
    for i in (0...arr.len):
        for j in (i+1...arr.len):
            if arr[i] > arr[j]: swap(arr[i], arr[j])
```

# Algoritmos de Ordenamiento

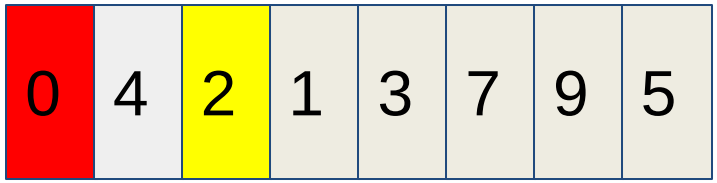
| 4 | 0 | 2 | 1 | 3 | 7 | 9 | 5 |
|---|---|---|---|---|---|---|---|

# Algoritmos de Ordenamiento



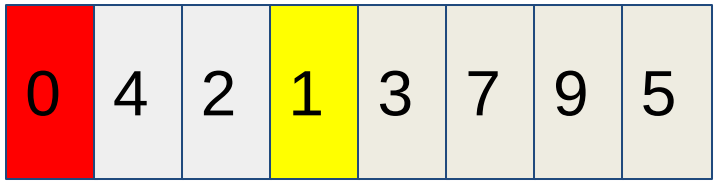

i = 0
j = 1

# Algoritmos de Ordenamiento



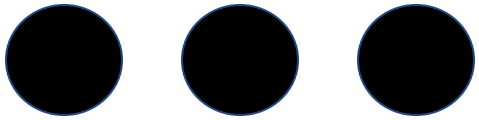

i = 0
j = 2

# Algoritmos de Ordenamiento



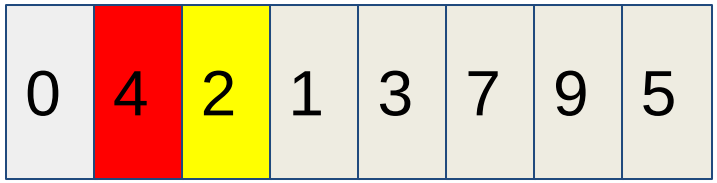

i = 0
j = 3

# Algoritmos de Ordenamiento



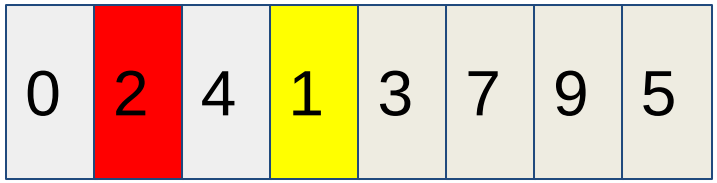
4 iteraciones más tarde...

# Algoritmos de Ordenamiento



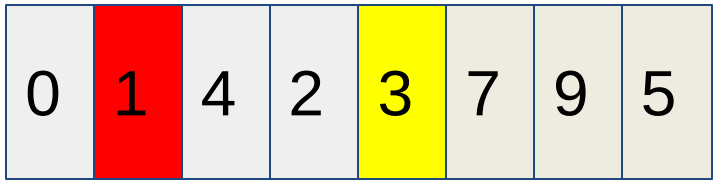

i = 1
j = 2

# Algoritmos de Ordenamiento



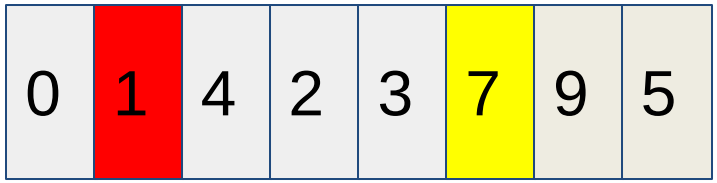

i = 1
j = 3

# Algoritmos de Ordenamiento



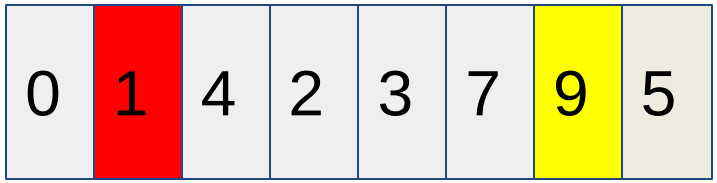

i = 1
j = 4

# Algoritmos de Ordenamiento



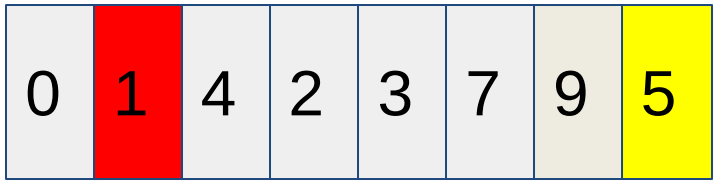

i = 1
j = 5

# Algoritmos de Ordenamiento



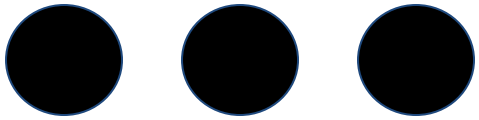

i = 1
j = 6

# Algoritmos de Ordenamiento



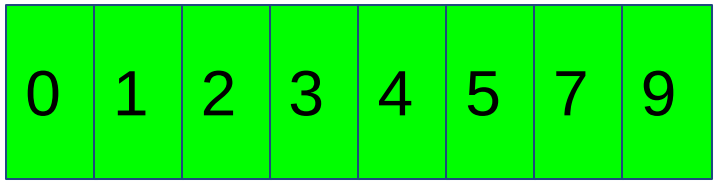

i = 1
j = 7

# Algoritmos de Ordenamiento



30 iteraciones más tarde...

# Algoritmos de Ordenamiento



# Algoritmos de Ordenamiento

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

# Algoritmos de Ordenamiento

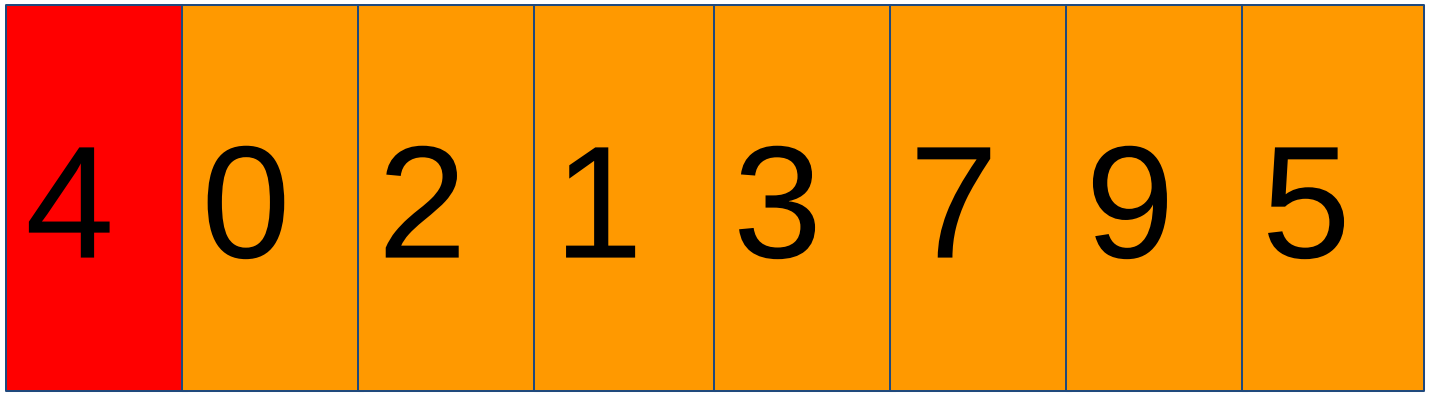
- Selection Sort (pseudocódigo)

```
fun selectionsort(arr):
    for i in (0...arr.len):
        min_index = i
        for j in (i+1...arr.len):
            if arr[min_index] > arr[j]:
                min_index = j
        temp = arr[i]
        arr[i] = arr[min_index]
        arr[min_index] = temp
```

# Algoritmos de Ordenamiento

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

# Algoritmos de Ordenamiento

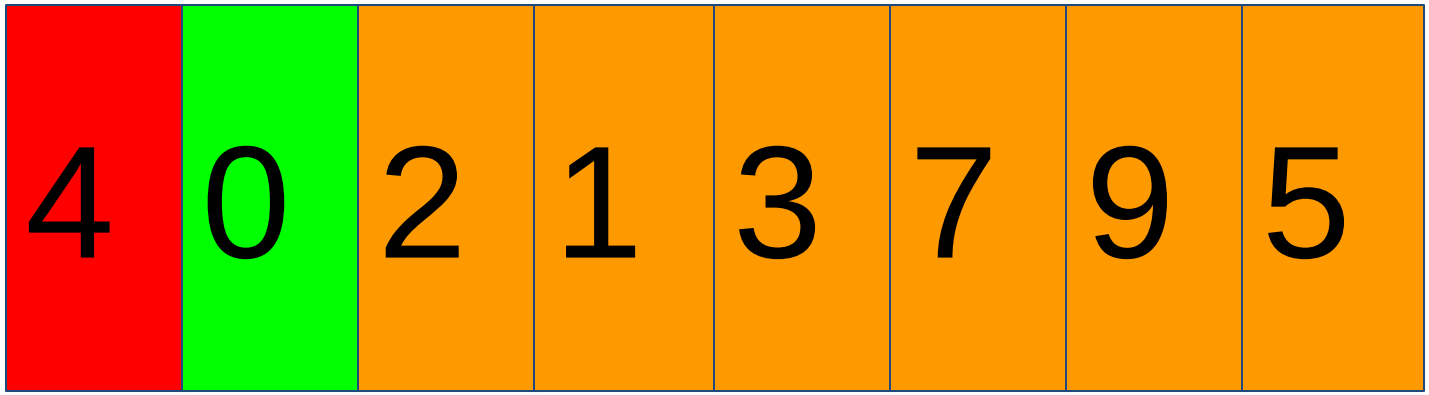



i = 0

k = 0

min = 4

# Algoritmos de Ordenamiento

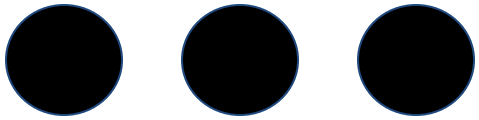



i = 0

k = 1

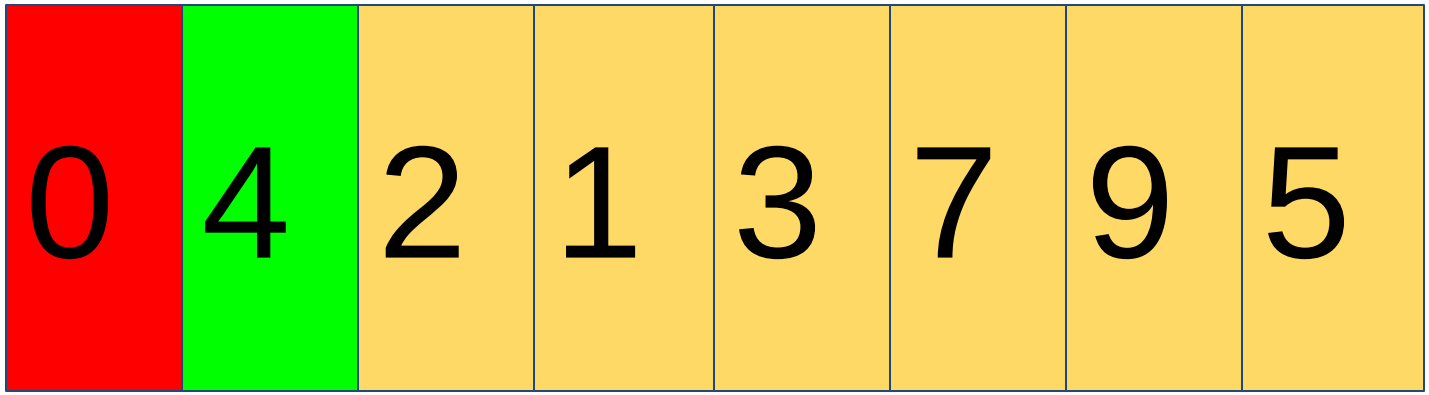
min = 0

# Algoritmos de Ordenamiento



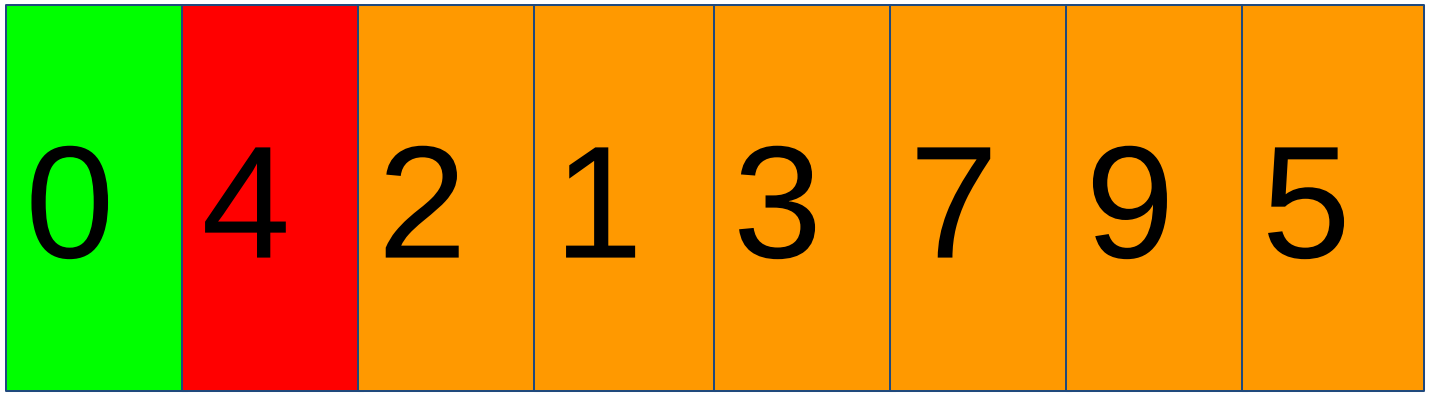
6 iteraciones más tarde...

# Algoritmos de Ordenamiento




Intercambiamos

# Algoritmos de Ordenamiento

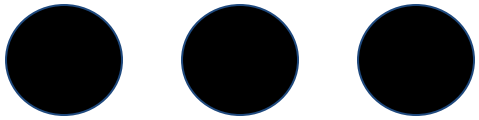



i = 1

k = 1

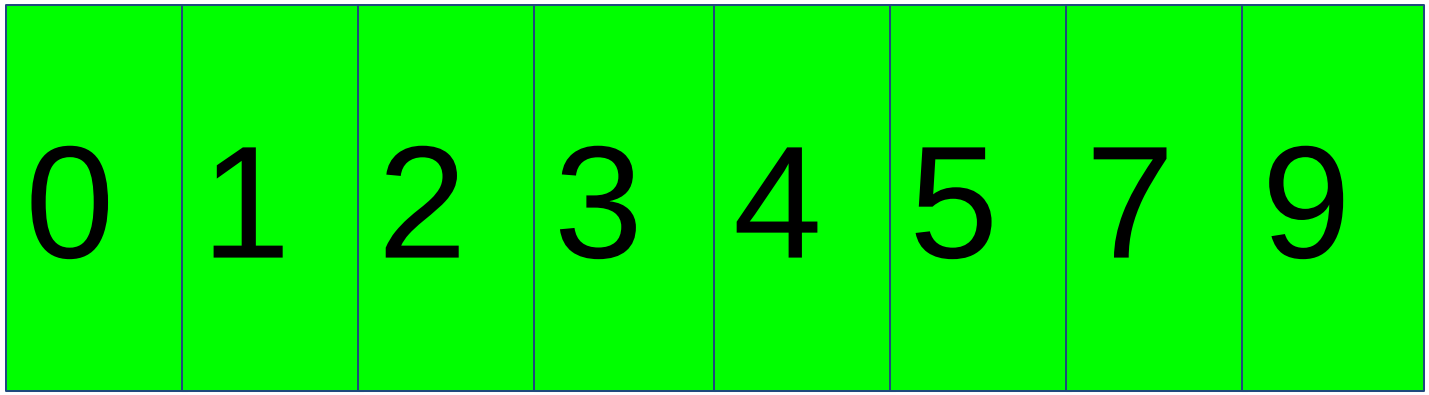
min = 4

# Algoritmos de Ordenamiento



42 iteraciones más tarde...

# Algoritmos de Ordenamiento



# Divide Y Vencerás

- Quicksort
- Mergesort
- Búsqueda binaria

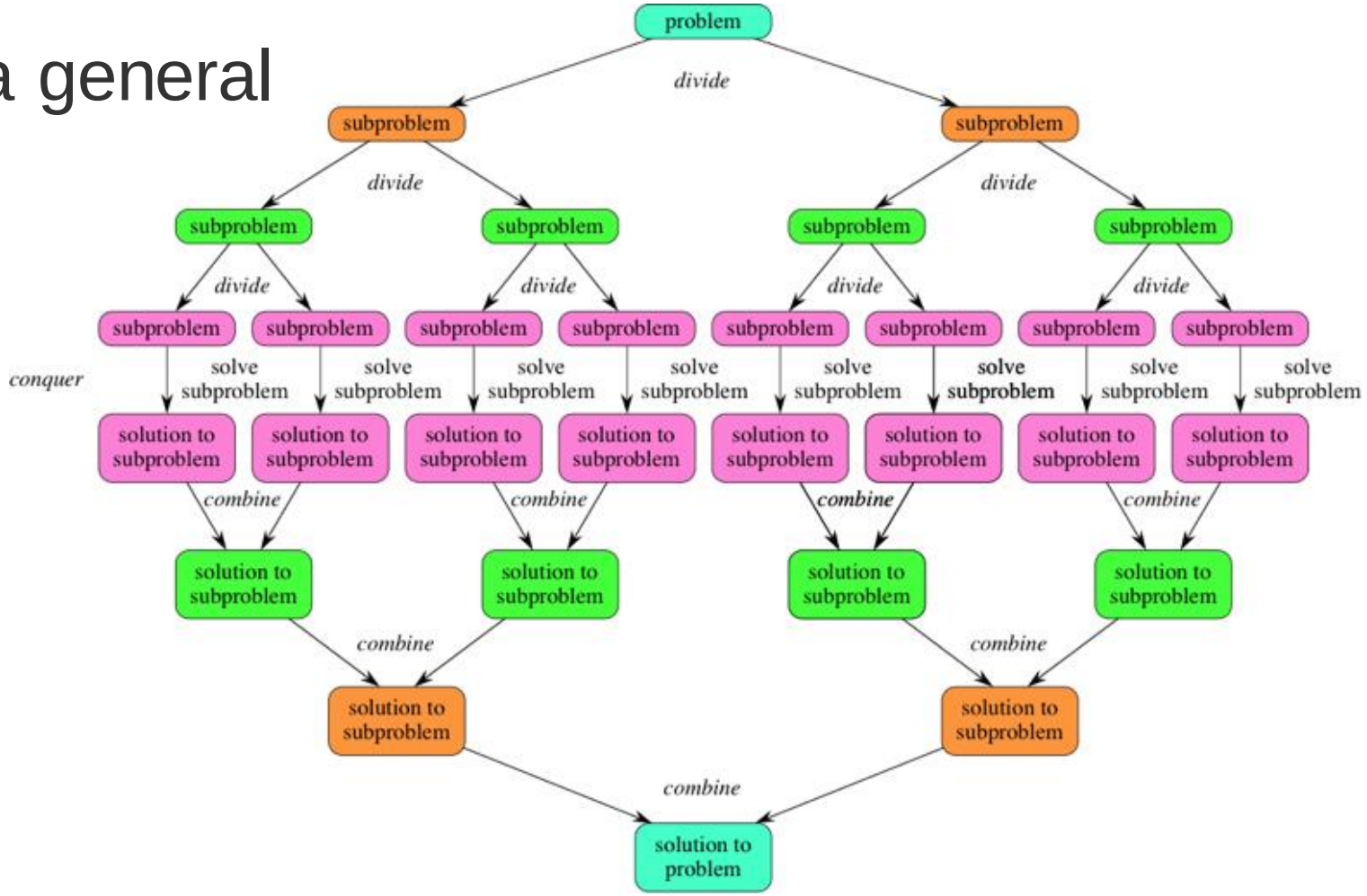
Los tres son ejemplos de esta aproximación...

# Divide Y Vencerás

- La idea intuitiva es la separación del problema en problemas más pequeños.
- Así, la solución global está compuesta de la combinación de subsoluciones.

# Divide Y Vencerás

Esquema general



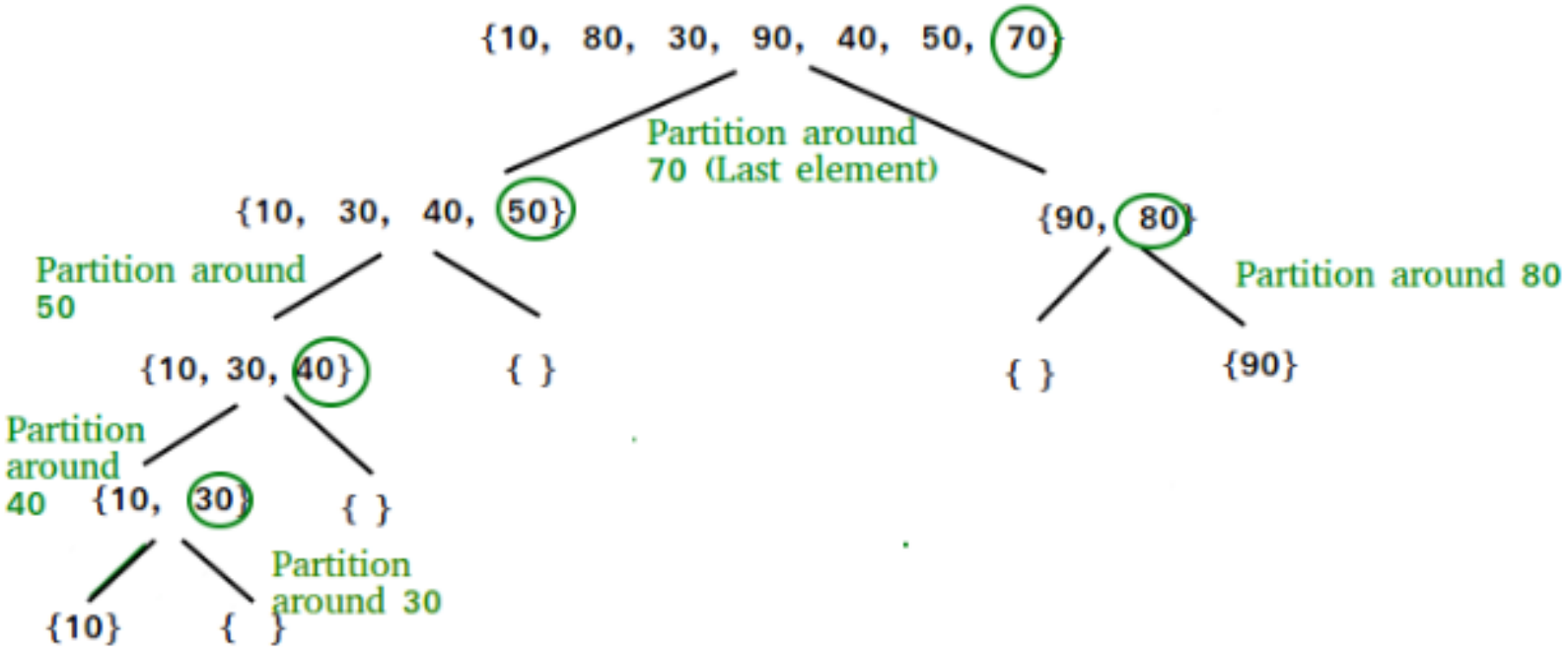
# Algoritmos de Ordenamiento

- Quicksort / Mergesort
  - Complejidad: O(nLog(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

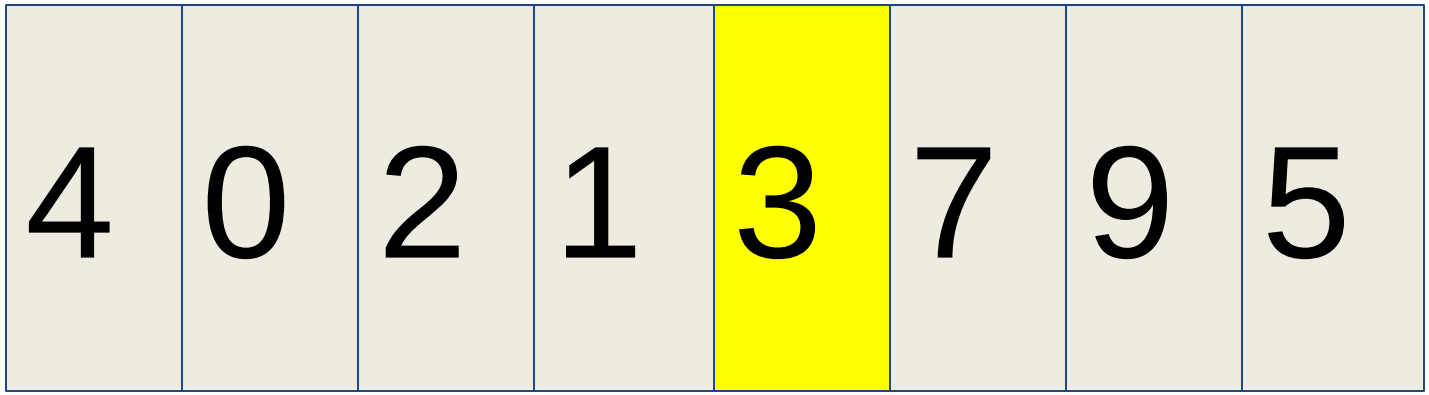
# Algoritmos de Ordenamiento



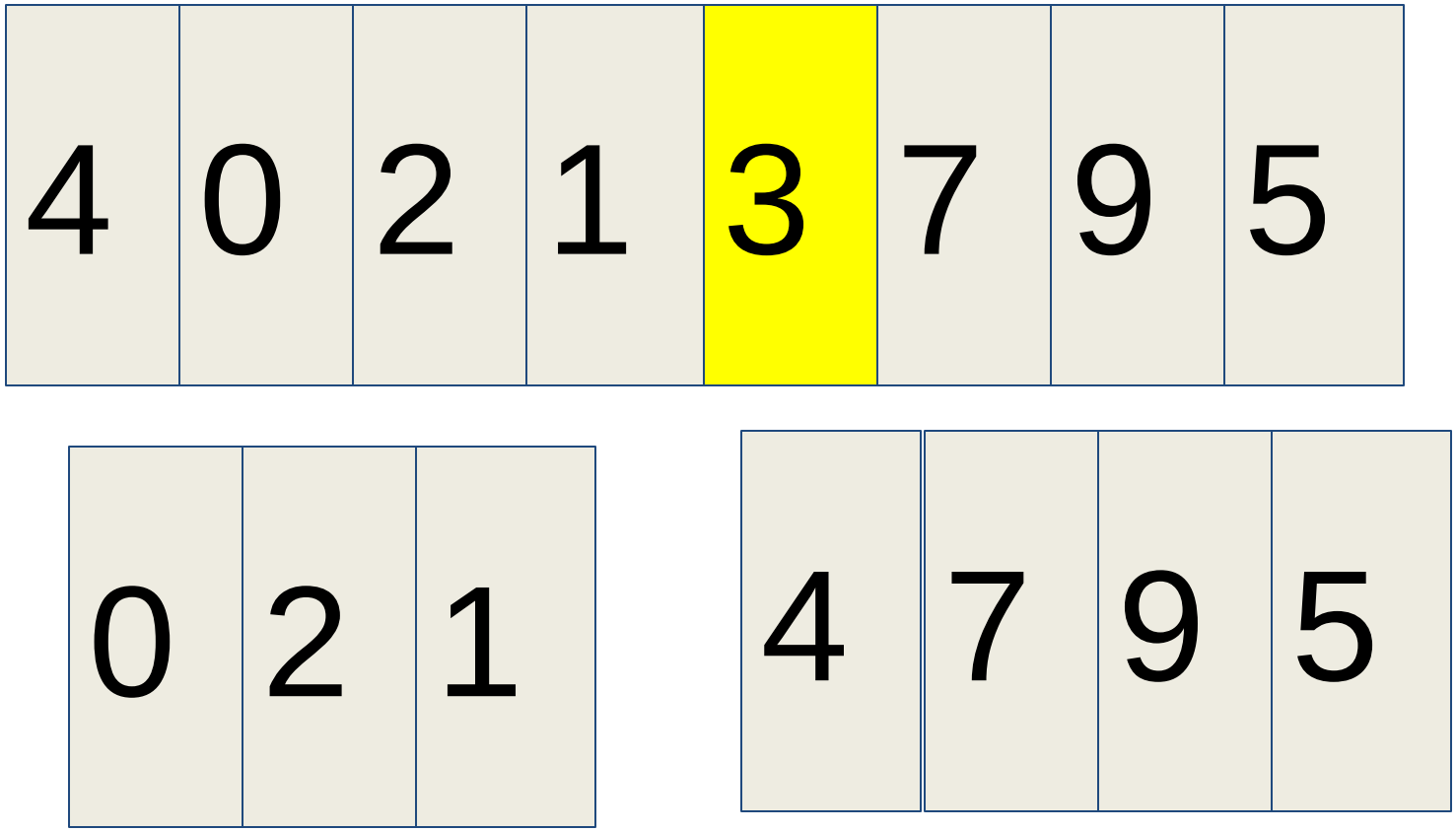
# Algoritmos de Ordenamiento

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

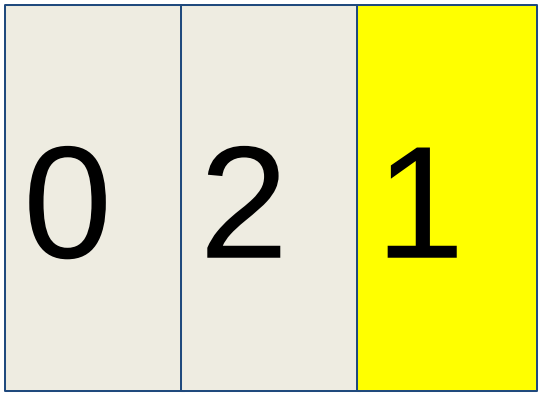
# Algoritmos de Ordenamiento



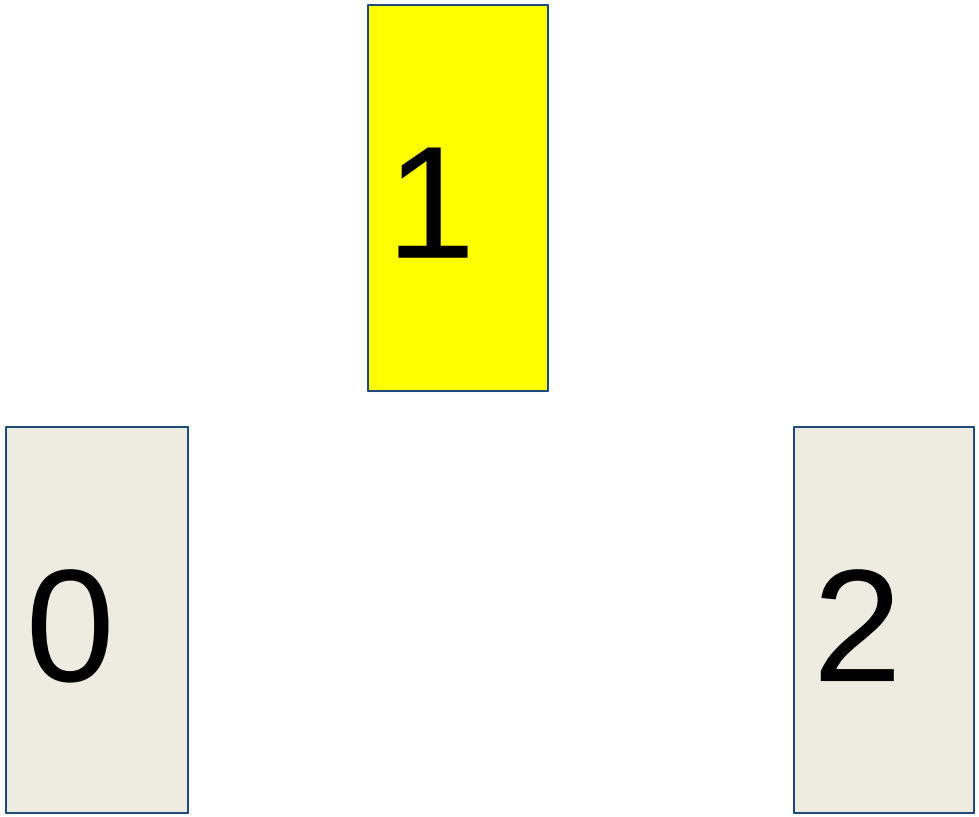
# Algoritmos de Ordenamiento



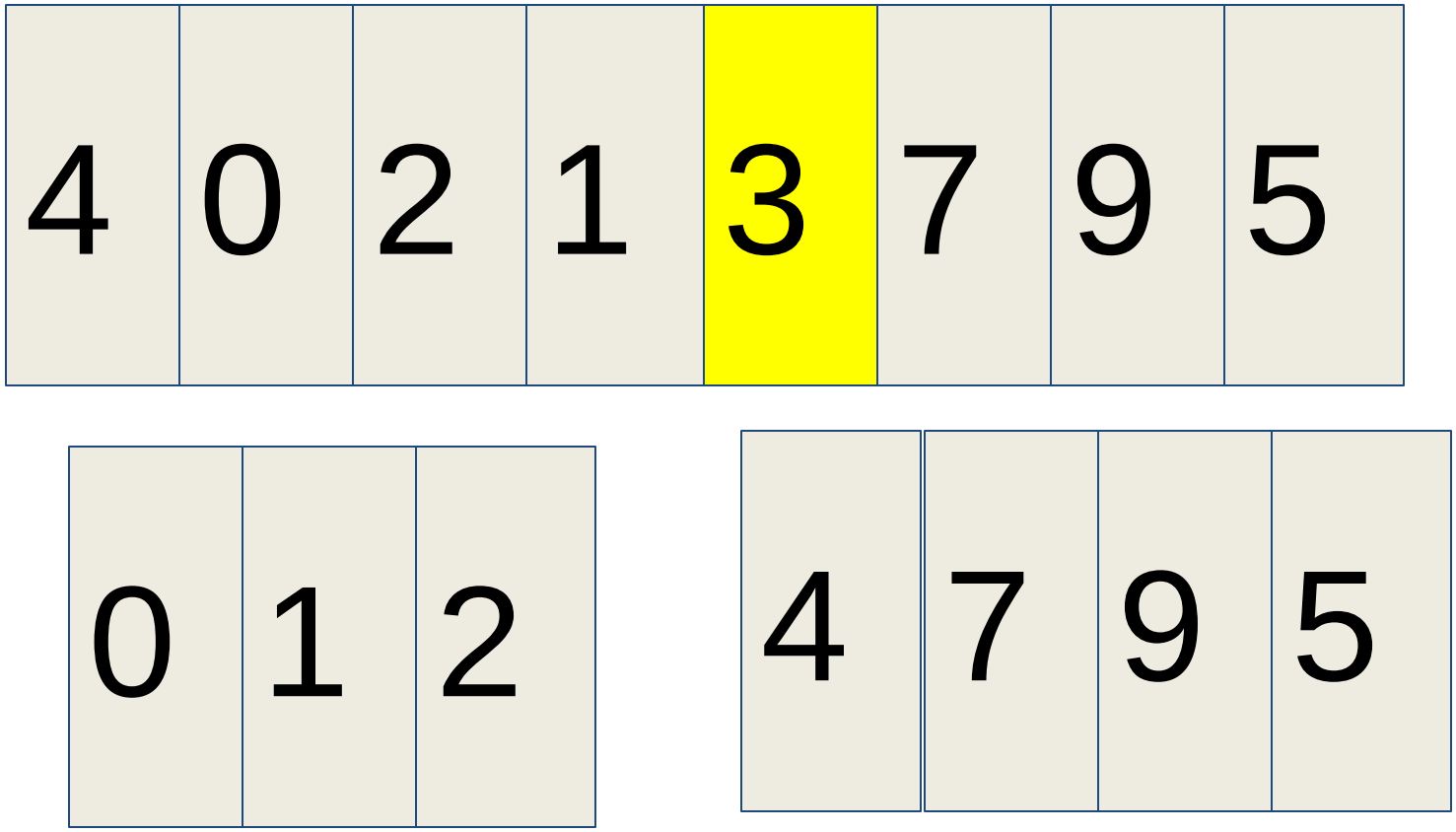
# Algoritmos de Ordenamiento



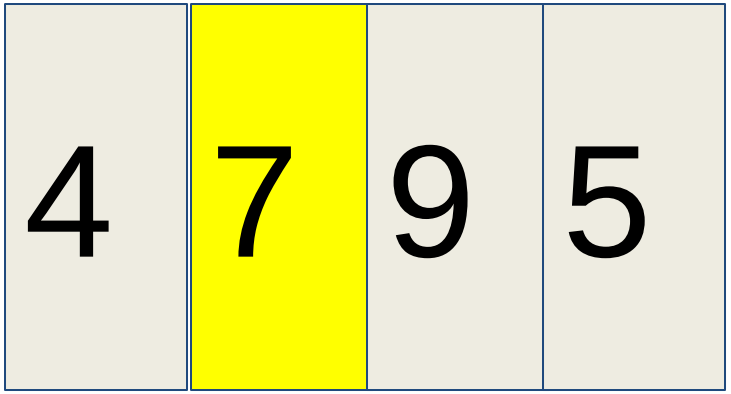
# Algoritmos de Ordenamiento



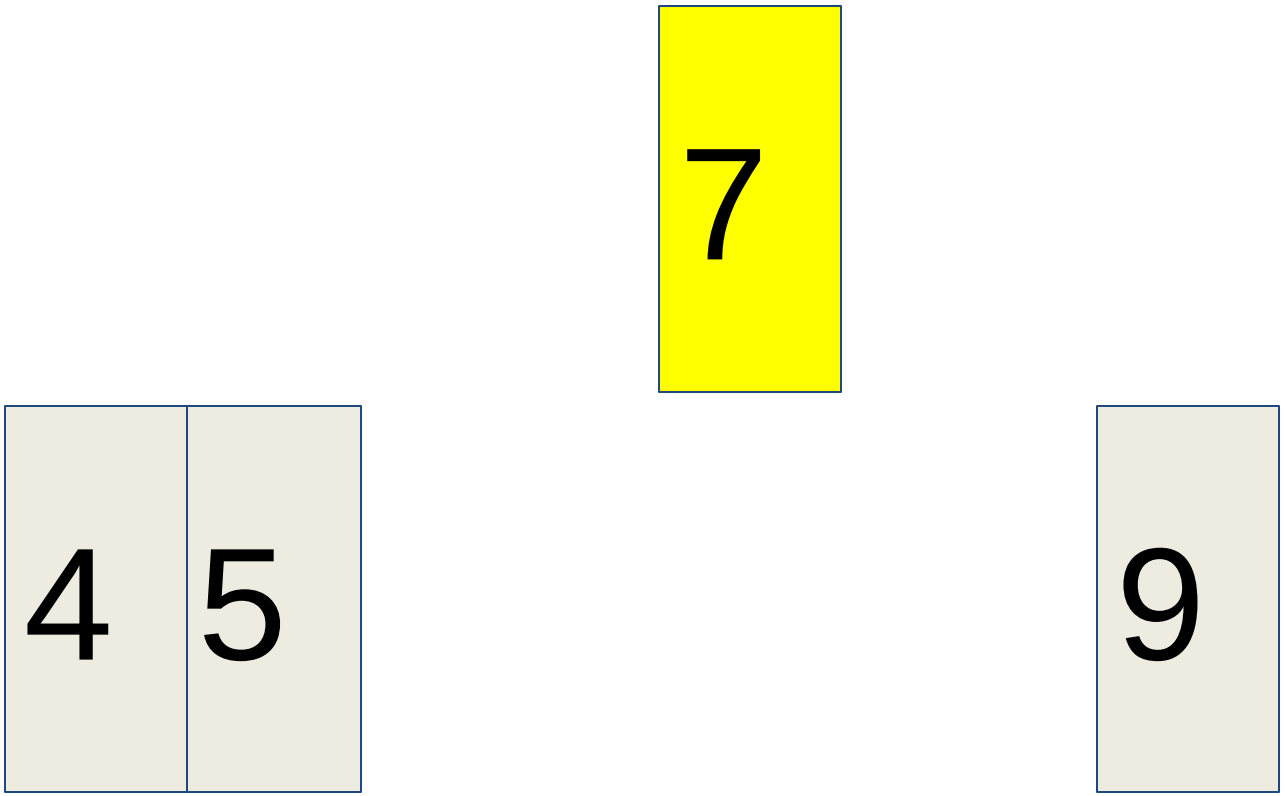
# Algoritmos de Ordenamiento



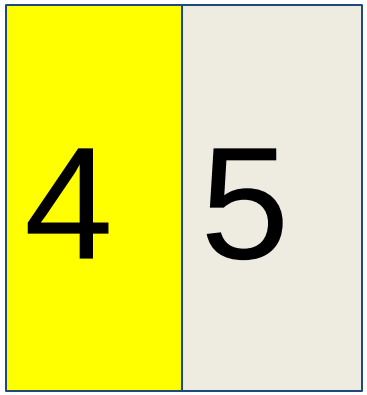
# Algoritmos de Ordenamiento



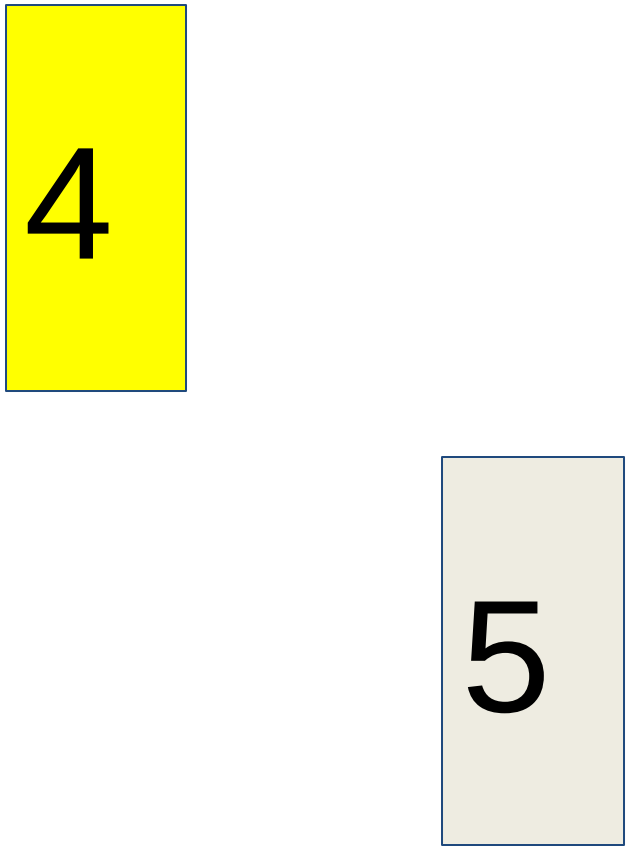
# Algoritmos de Ordenamiento



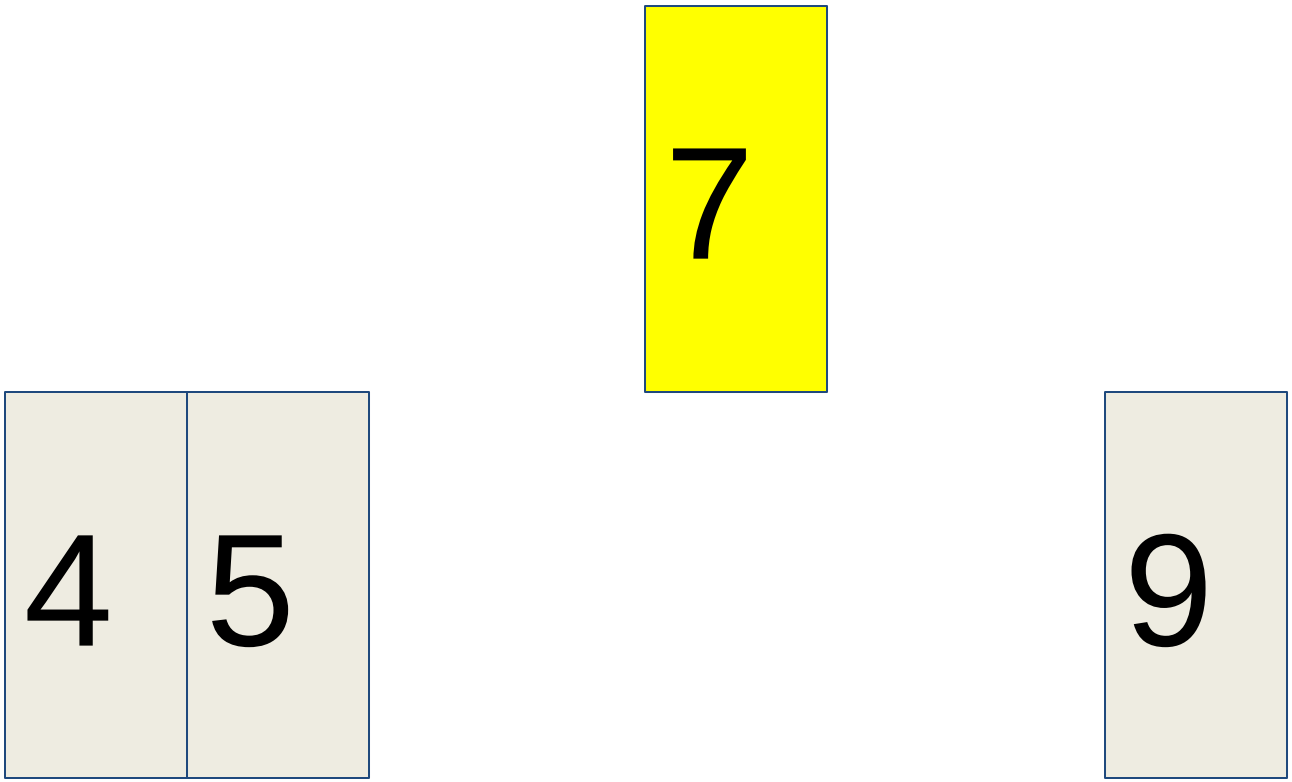
# Algoritmos de Ordenamiento



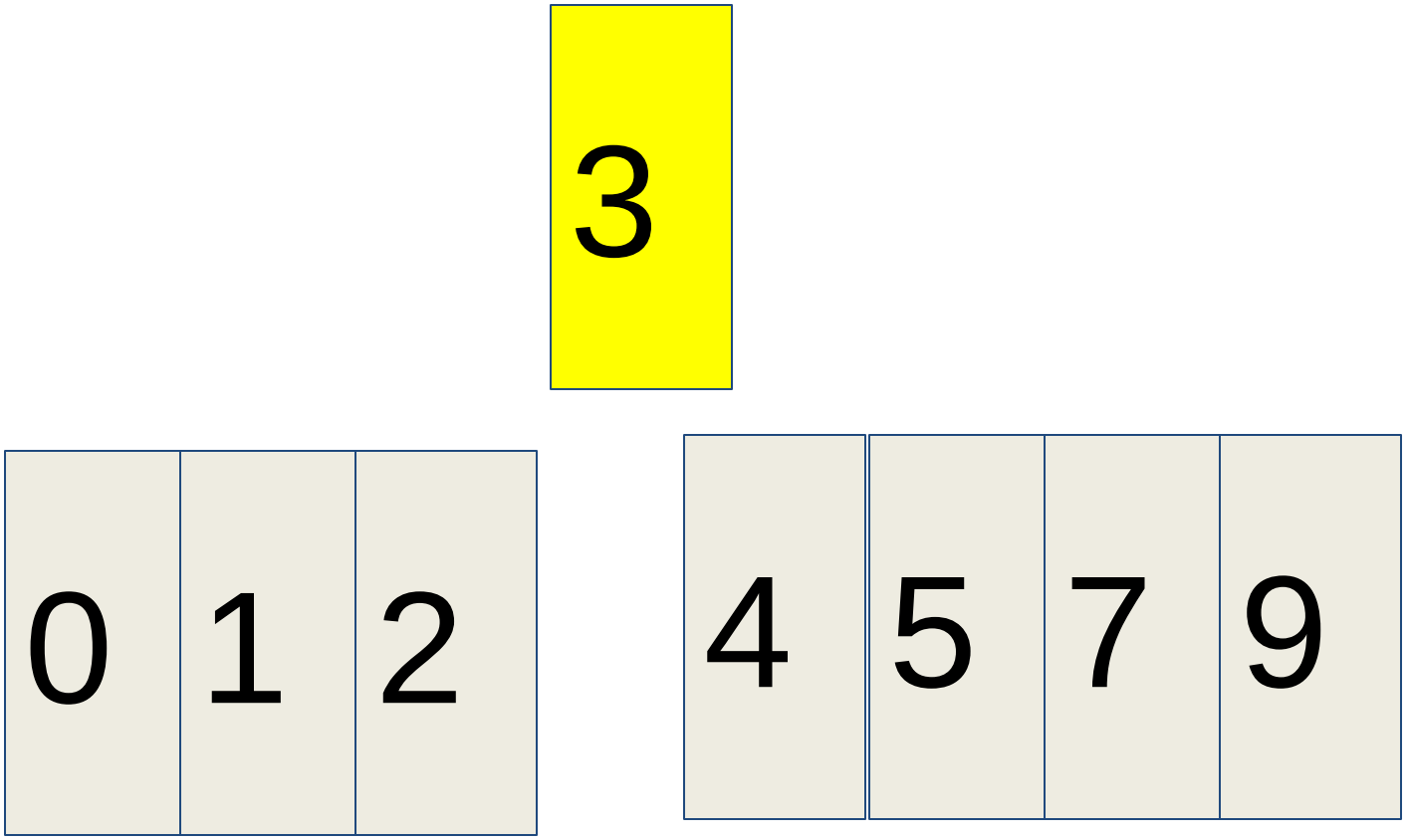
# Algoritmos de Ordenamiento



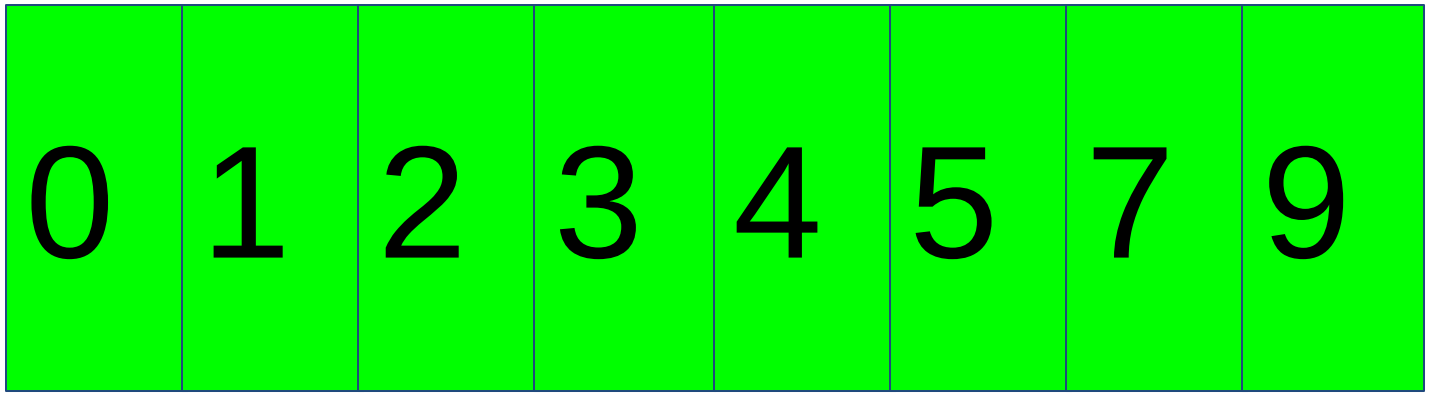
# Algoritmos de Ordenamiento



# Algoritmos de Ordenamiento



# Algoritmos de Ordenamiento



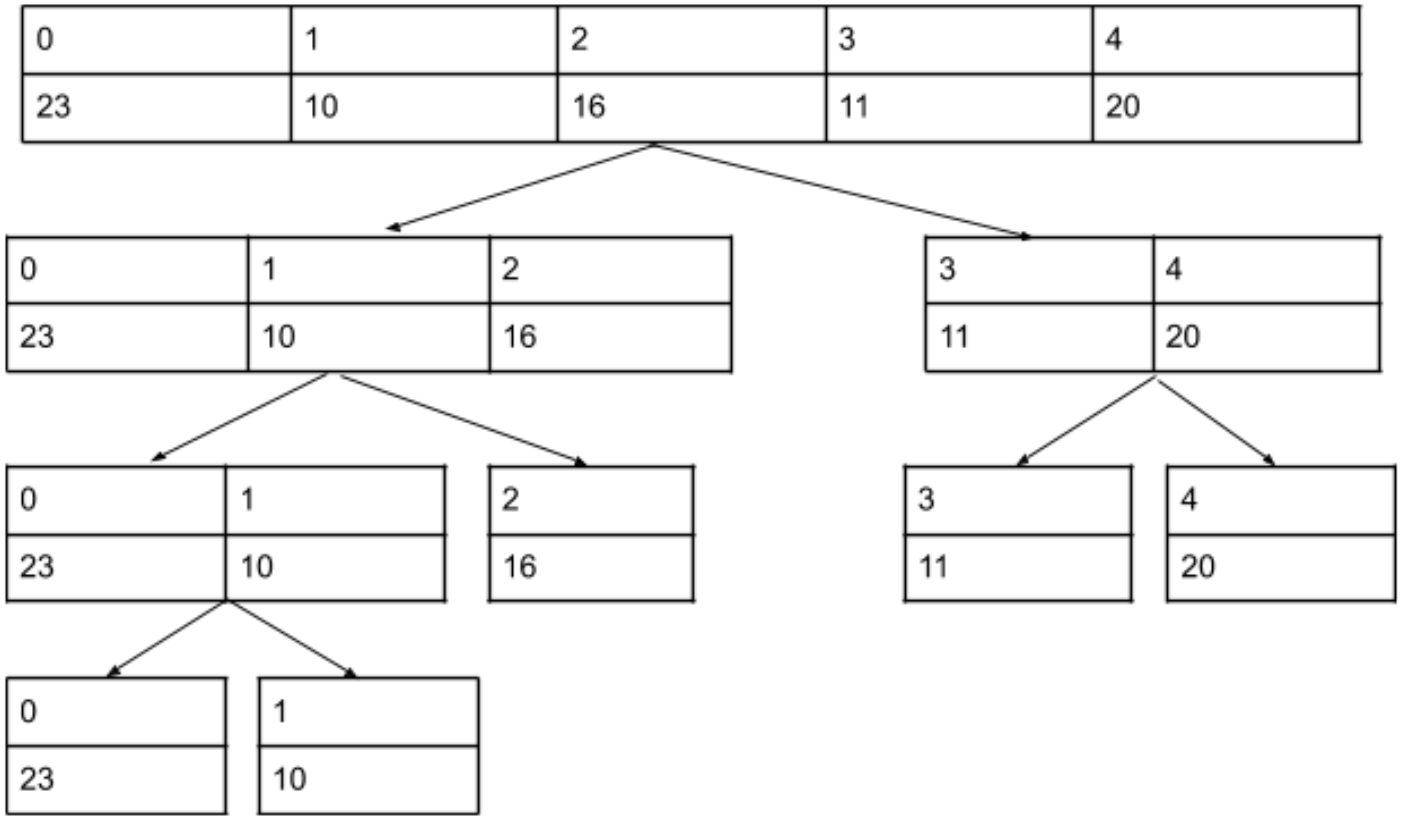
# Algoritmos de Ordenamiento

- Quicksort (idea básica)
  - Si se hace de manera ideal y perfecta, su complejidad es O(NlgN)
  - 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

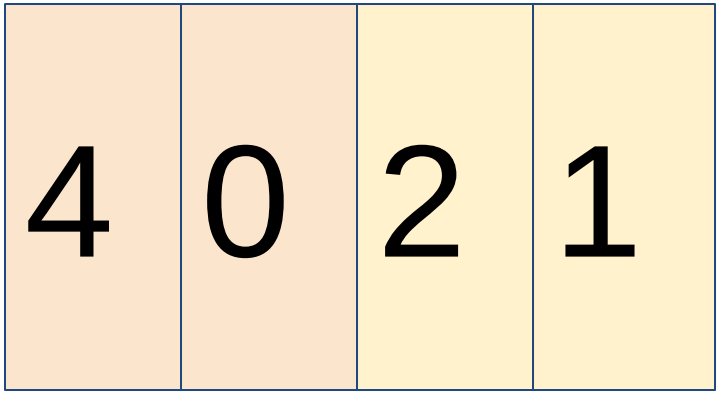
# Algoritmos de Ordenamiento



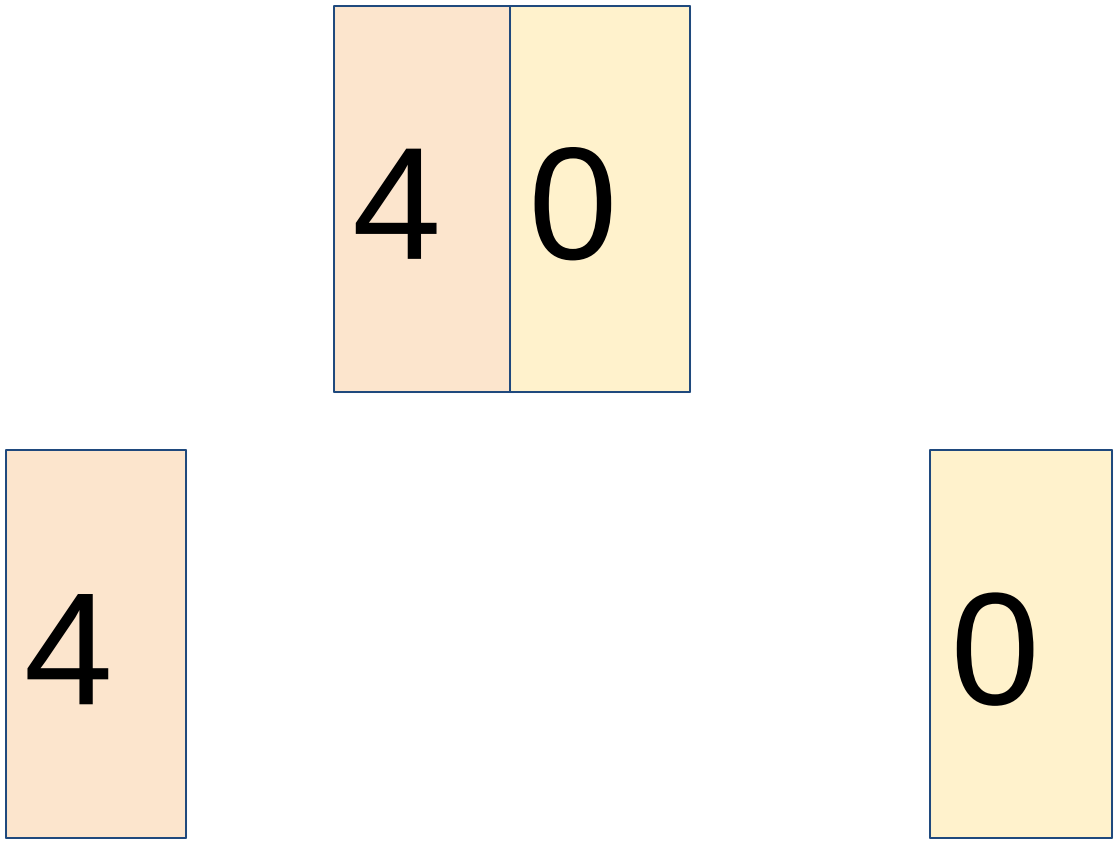
# Algoritmos de Ordenamiento

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

# Algoritmos de Ordenamiento



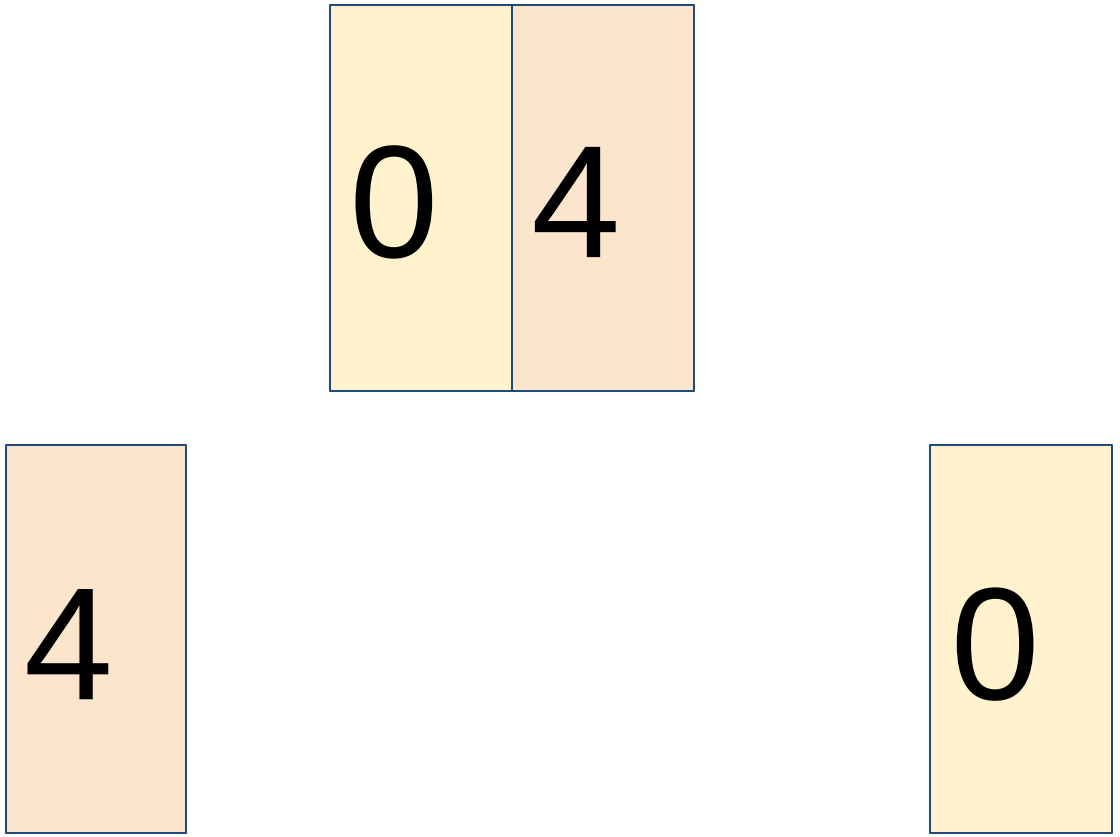
# Algoritmos de Ordenamiento



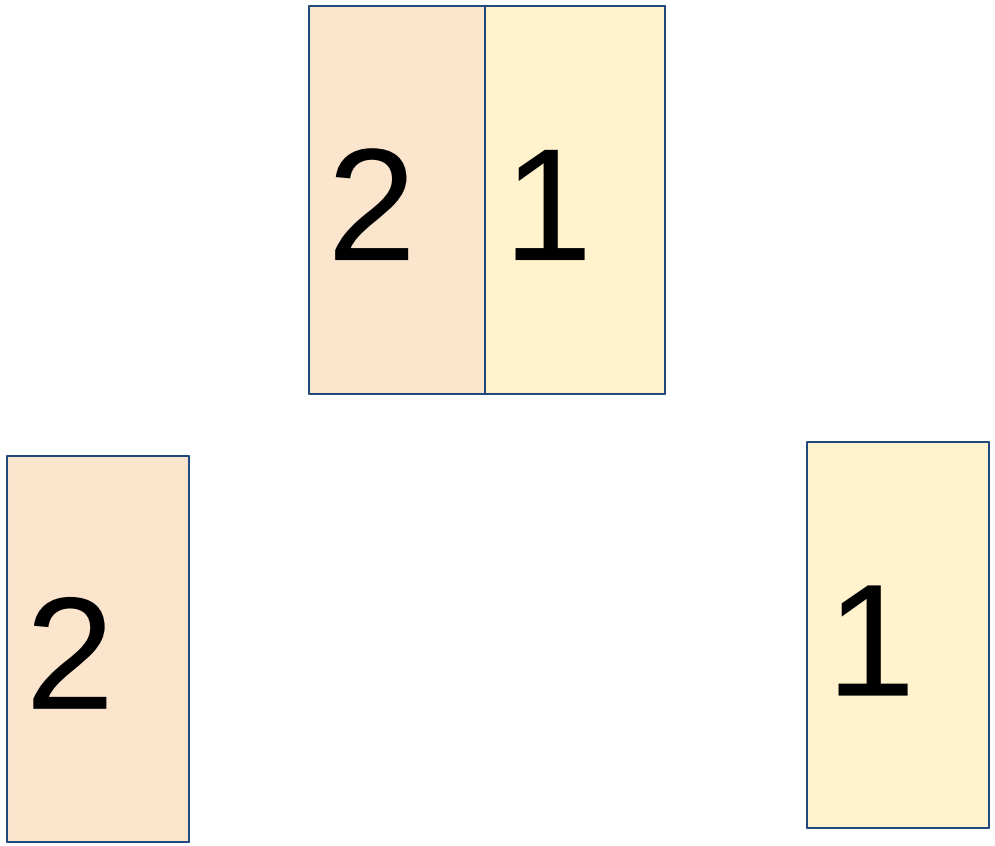
# Algoritmos de Ordenamiento

- 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

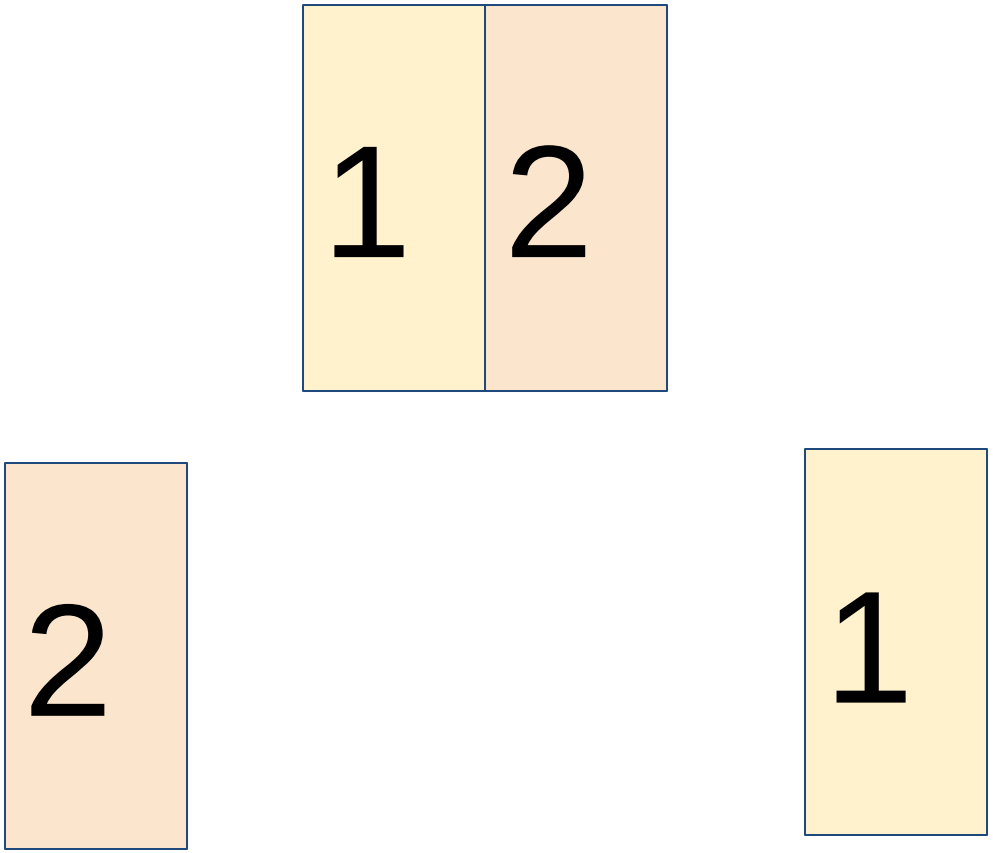
# Algoritmos de Ordenamiento



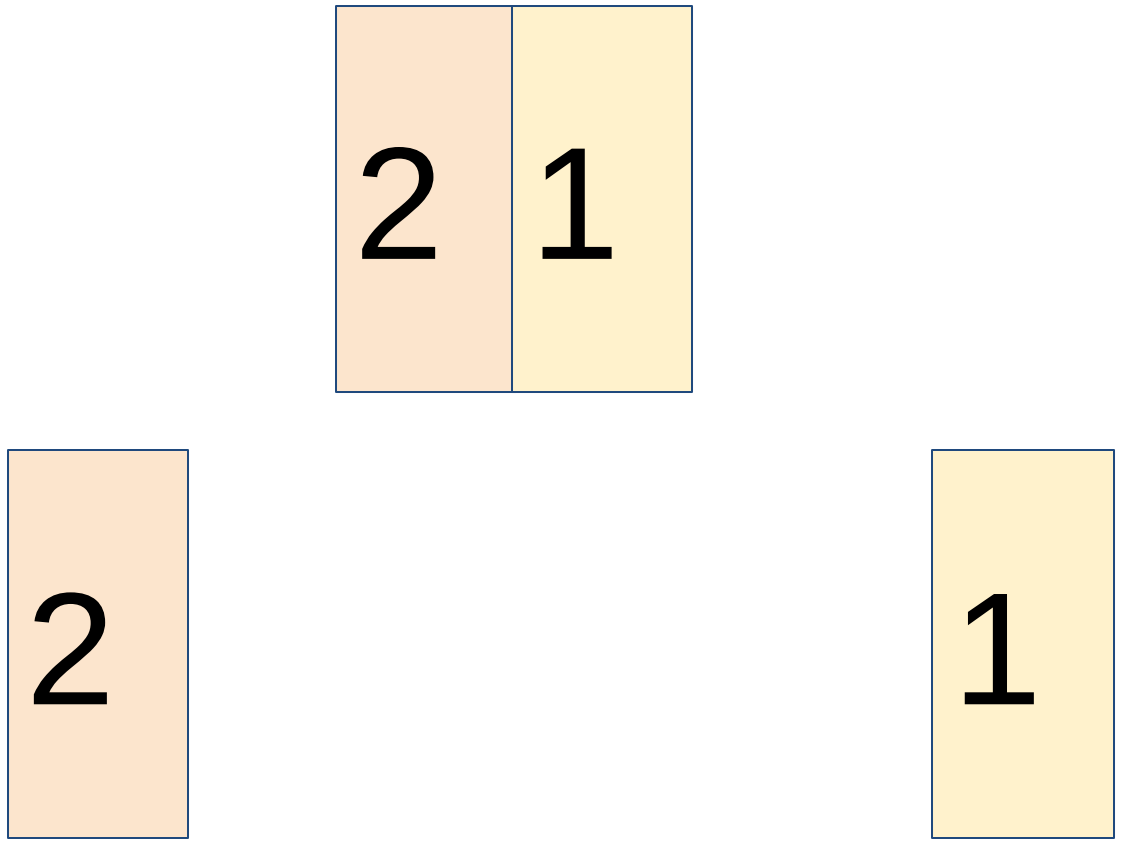
# Algoritmos de Ordenamiento



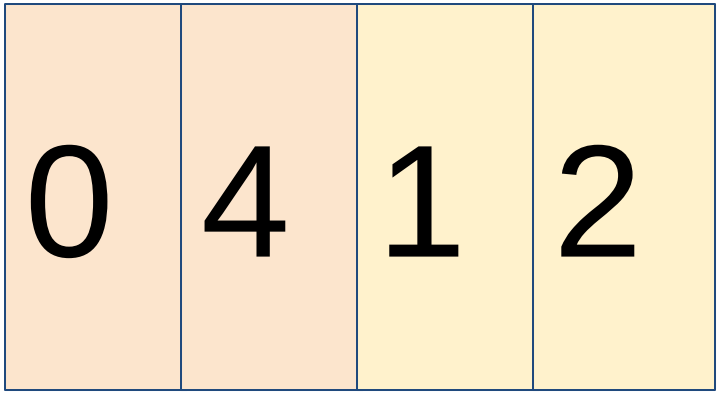
# Algoritmos de Ordenamiento



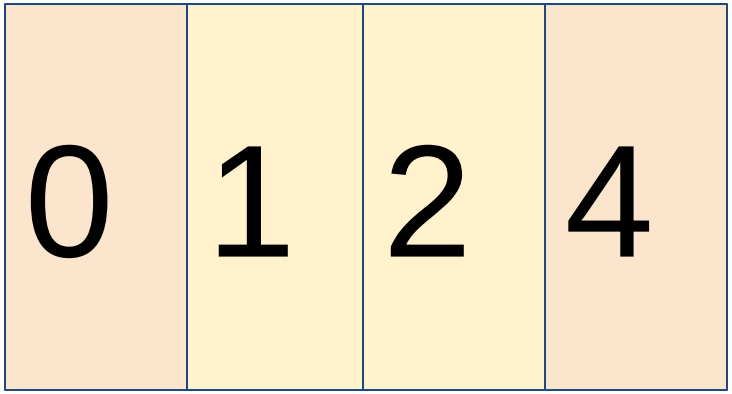
# Algoritmos de Ordenamiento



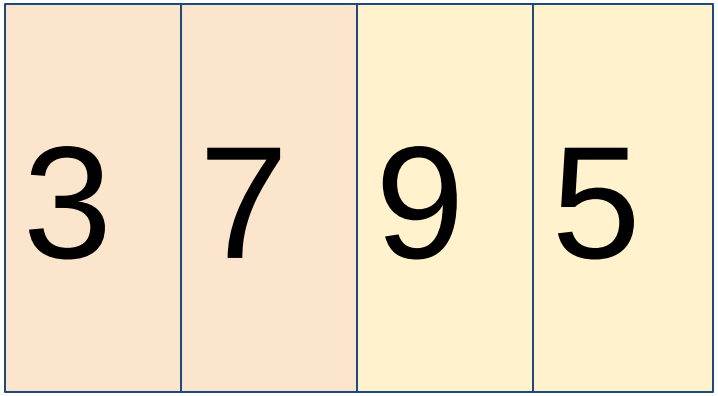
# Algoritmos de Ordenamiento



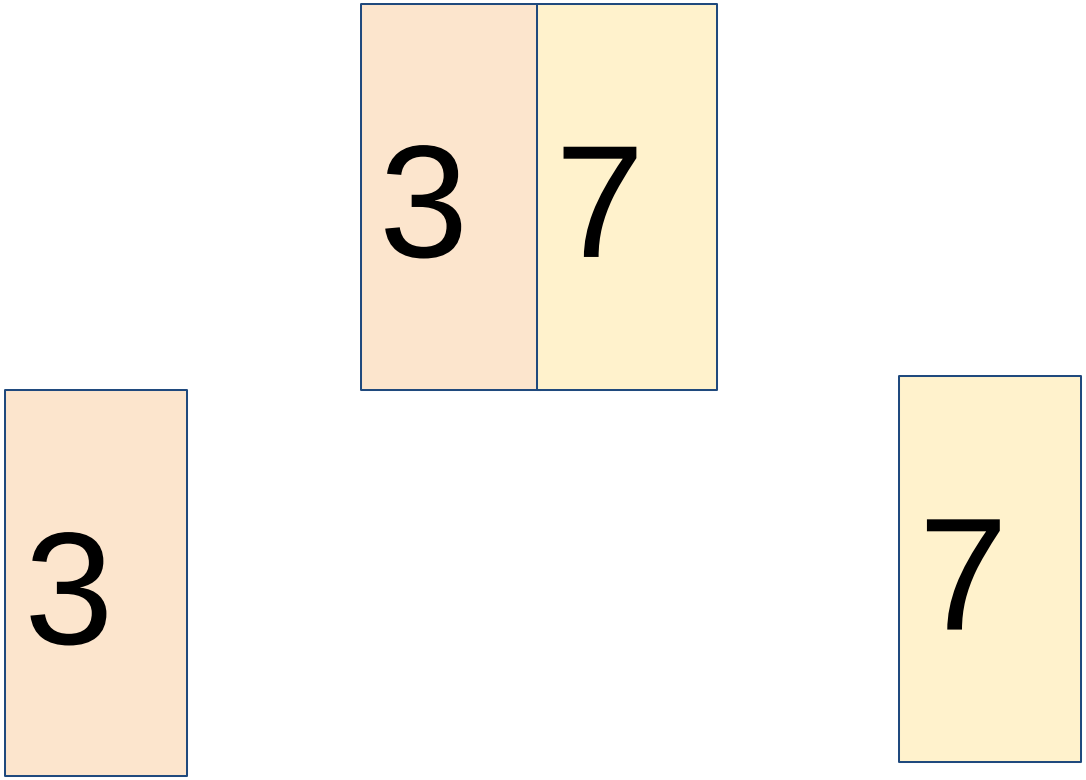
# Algoritmos de Ordenamiento



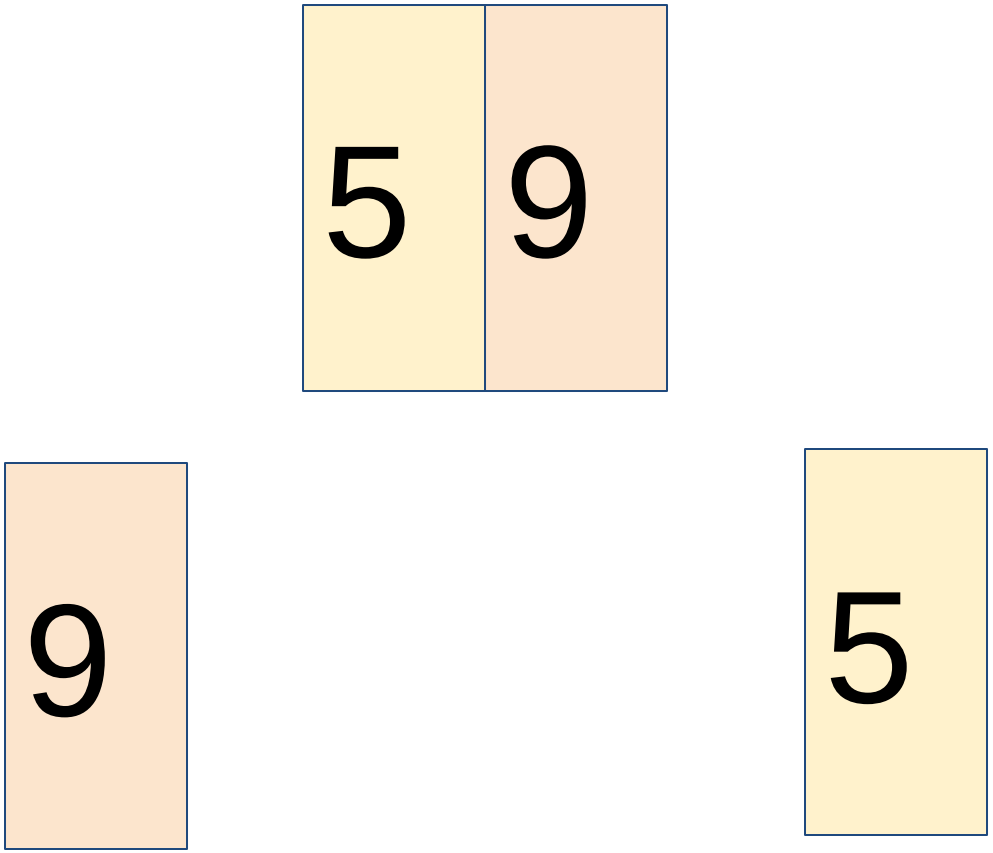
# Algoritmos de Ordenamiento



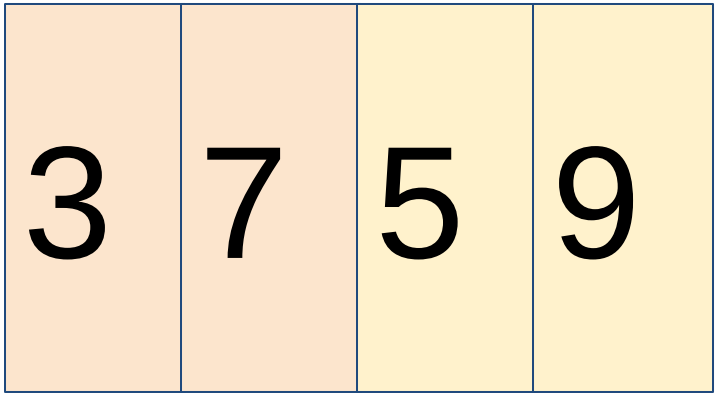
# Algoritmos de Ordenamiento



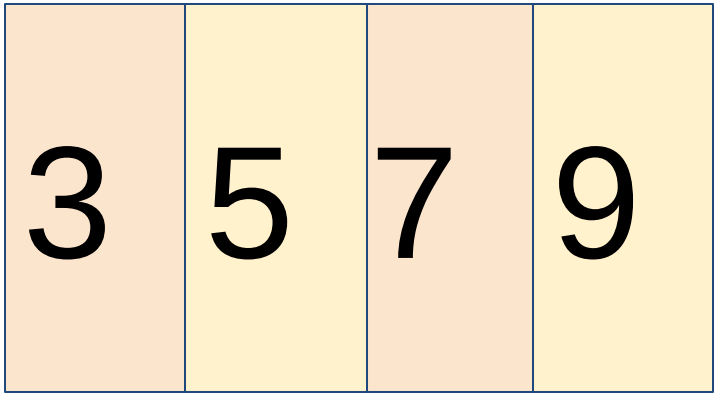
# Algoritmos de Ordenamiento



# Algoritmos de Ordenamiento



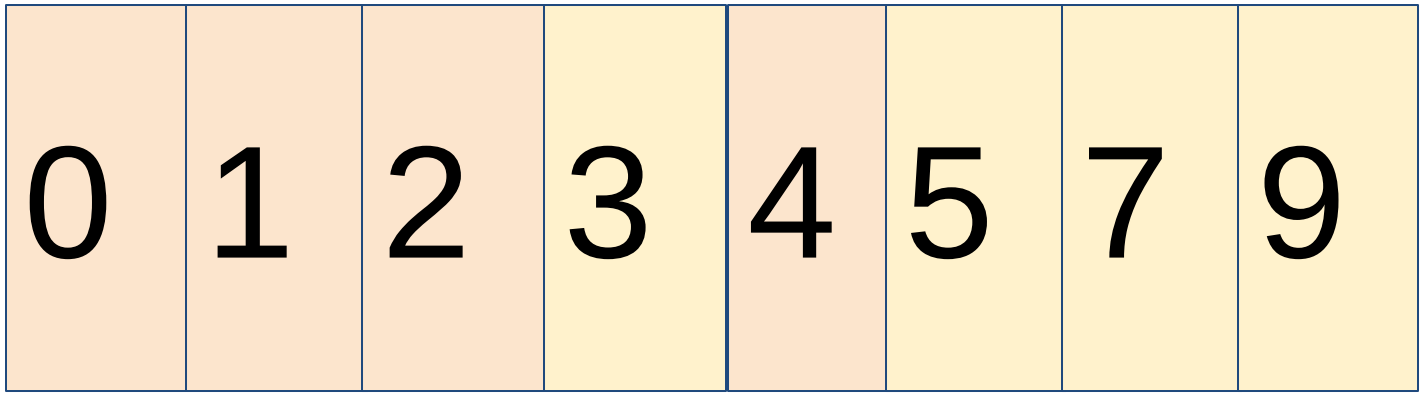
# Algoritmos de Ordenamiento



# Algoritmos de Ordenamiento

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

# Algoritmos de Ordenamiento



# 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) <= 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 mid = (Cotamin + Cotasup) / 2
- Si f(mid) > Obj, significa que nos hemos pasado, por lo tanto, Cotasup = mid
- Si f(mid) <= Obj, signifiica que hemos subestimado, por lo tanto, Cotainf = 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

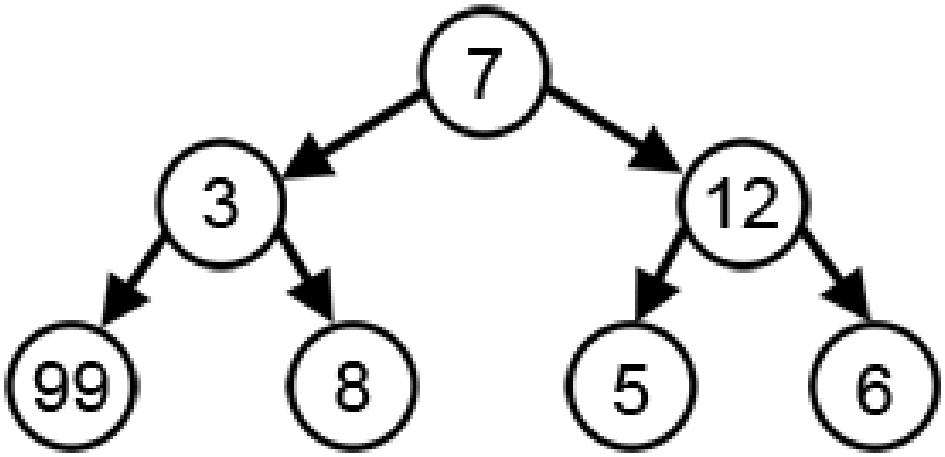
## Partes del algoritmo

- **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 prometedor 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



# Problemas

https://www.aceptaelreto.com/problem/statement.php?id=650&cat=30

https://www.aceptaelreto.com/problem/statement.php?id=194&cat=30

# ¡Hasta la próxima semana!

Ante cualquier duda sobre el curso o sobre los problemas podéis escribirnos (preferiblemente con copia a algunos / todos los docentes)

- Isaac Lozano (isaac.lozano@urjc.es)
- Raúl Martín(raul.martin@urjc.es)
- Sergio Salazar (s.salazarc.2018@alumnos.urjc.es)
- Francisco Tórtola (f.tortola.2018@alumnos.urjc.es)
- **Cristian Pérez(c.perezc.2018@alumnos.urjc.es)**
- Xuqiang Liu(x.liu1.2020@alumnos.urjc.es)
- Alicia Pina(a.pinaz.2020@alumnos.urjc.es)
- Sara García(s.garciarod.2020@alumnos.urjc.es)
- Raúl Fauste(r.fauste.2020@alumnos.urjc.es)