Algoritmos de ordenamiento y búsqueda (parte 2)

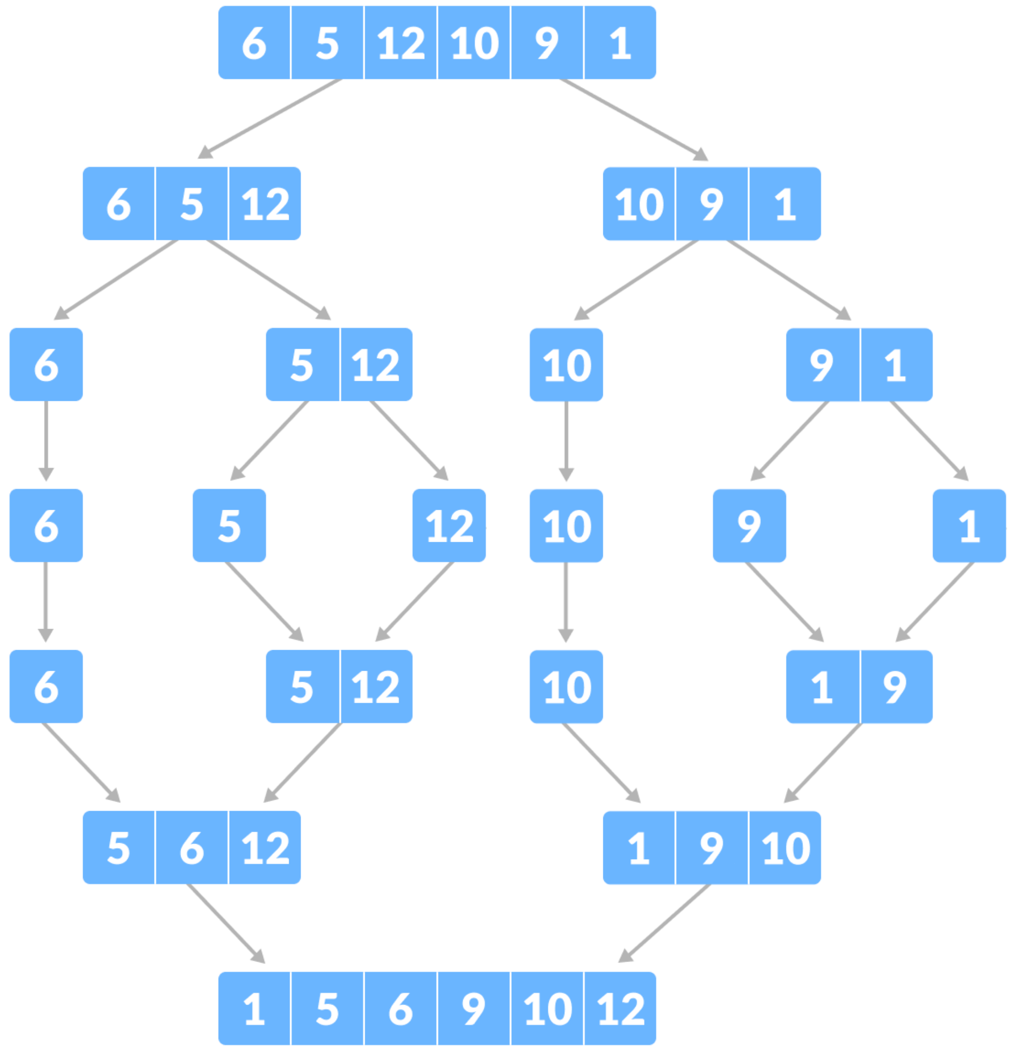
Cátedra de Fundamentos de Programación

Clases:

Elaborado por José Colbes (jcolbes@fiuna.edu.py)

¿Qué veremos?

- Mergesort
- Quicksort



Ordenamiento de datos

Consiste en reorganizar un conjunto de datos u objetos en una secuencia específica, la cual puede ser de dos formas:

Ascendente $i < j \rightarrow A[i] \leq A[j]$

Descendente $i < j \rightarrow A[i] \geq A[j]$

A	4	7	1	5	3
A_{Asc}	1	3	4	5	7
A_{Des}	7	5	4	3	1

Alumnos:

- Carlos Sauer
- Diego Stalder
- José Colbes
- Viviana Ortellado
- Juan Ovelar
- David Fretes

Por apellidos (Asc):

- José Colbes
- David Fretes
- Viviana Ortellado
- Juan Ovelar
- Carlos Sauer
- Diego Stalder

Ordenamiento de datos

El proceso de ordenamiento es uno de los ejemplos más interesantes para mostrar que existen múltiples soluciones para un mismo problema, y que cada algoritmo tiene sus propias ventajas y desventajas.

Según sus enfoques, los algoritmos de ordenamiento se dividen en grupos:

Básicos

- Selección
- Burbuja
- Inserción

Avanzados

- Merge-sort (Mezcla)
- Quicksort (Rápida)
- Shell
- Radix

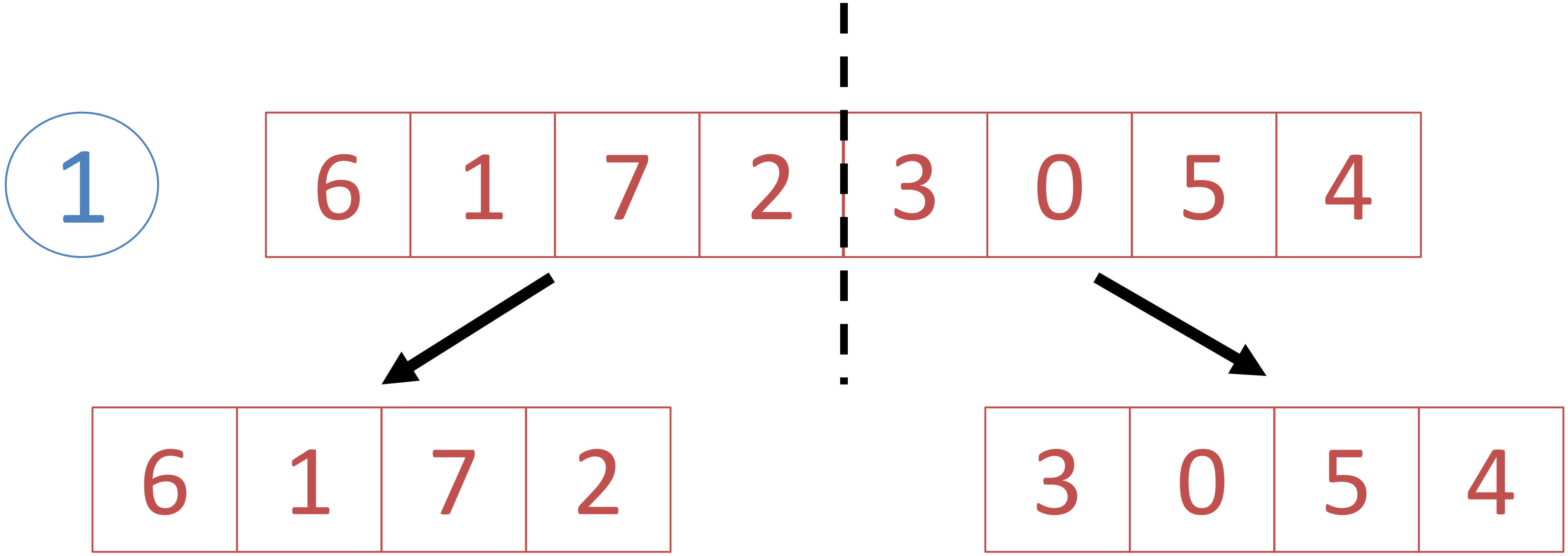
Merge-sort

El Merge-sort es un algoritmo de ordenamiento recursivo, y es más eficiente que otros algoritmos de ordenamiento (inserción, selección, burbuja). Básicamente realiza lo siguiente:

1) Divide la secuencia de n elementos en dos subsecuencias de $n/2$ elementos.

2) Ordena ambas subsecuencias utilizando merge-sort (**recursión**).

3) Fusiona las dos subsecuencias para producir una secuencia ordenada.



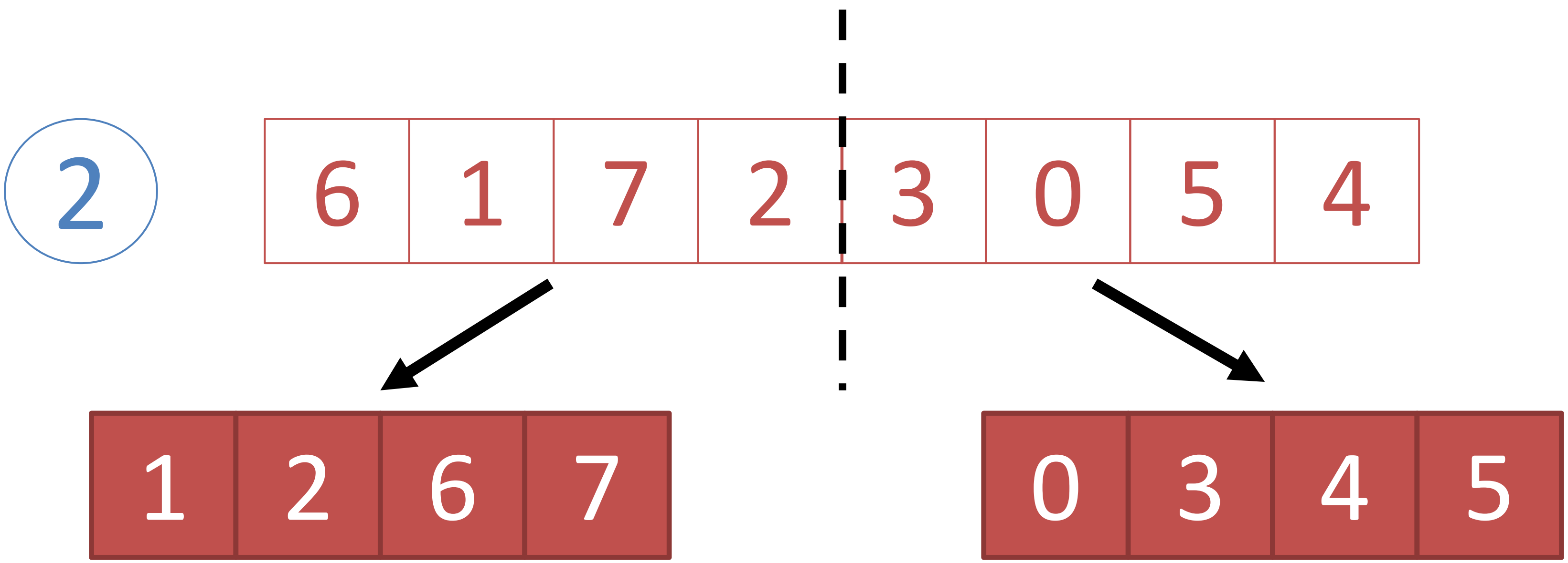
Merge-sort

El Merge-sort es un algoritmo de ordenamiento recursivo, y es más eficiente que otros algoritmos de ordenamiento (inserción, selección, burbuja). Básicamente realiza lo siguiente:

1) Divide la secuencia de n elementos en dos subsecuencias de $n/2$ elementos.

2) Ordena ambas subsecuencias utilizando merge-sort (**recursión**).

3) Fusiona las dos subsecuencias para producir una secuencia ordenada.



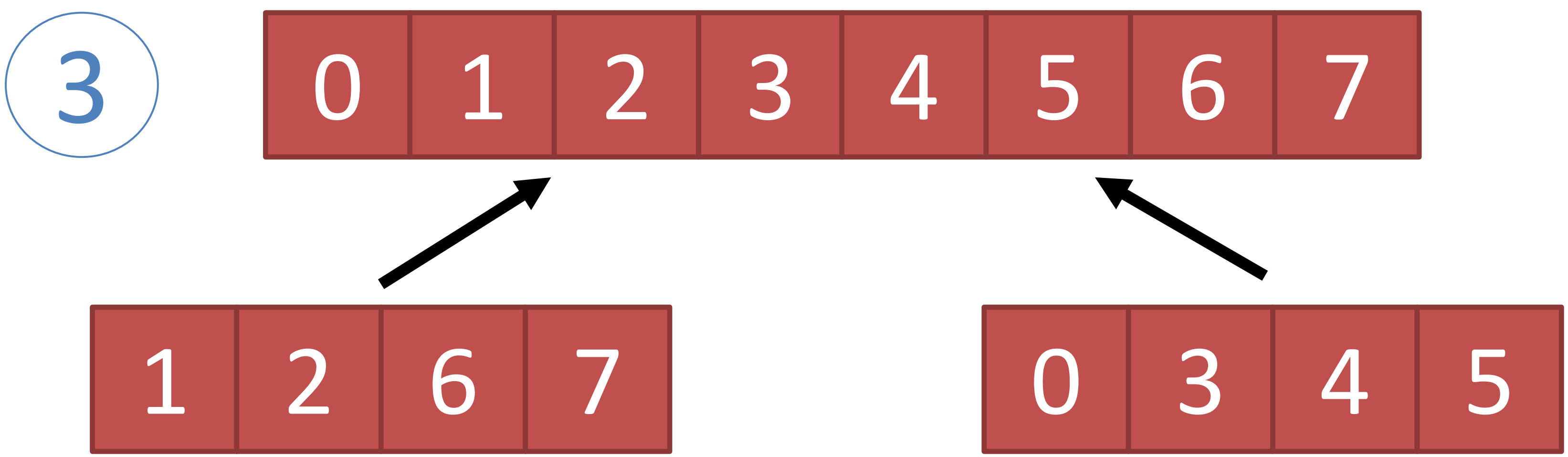
Merge-sort

El Merge-sort es un algoritmo de ordenamiento recursivo, y es más eficiente que otros algoritmos de ordenamiento (inserción, selección, burbuja). Básicamente realiza lo siguiente:

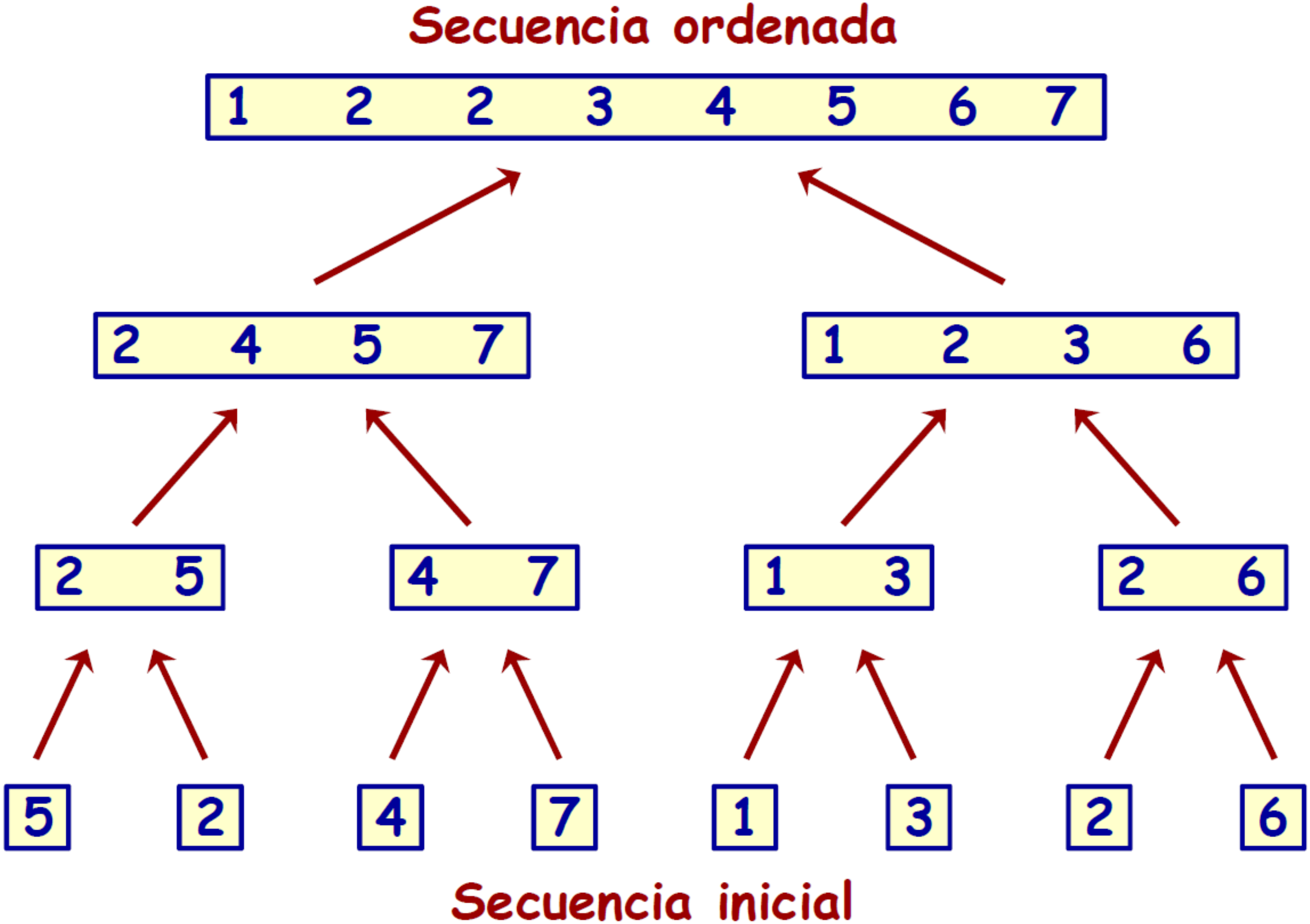
1) Divide la secuencia de n elementos en dos subsecuencias de $n/2$ elementos.

2) Ordena ambas subsecuencias utilizando merge-sort (**recursión**).

3) Fusiona las dos subsecuencias para producir una secuencia ordenada.



Merge-sort



Secuencia ordenada
1 2 2 3 4 5 6 7
2 4 5 7
1 2 3 6
2 5
4 7
1 3
2 6
5
2
4
7
1
3
2
6
Secuencia inicial

Merge-sort

El procedimiento clave del algoritmo merge-sort es la ***fusión de datos*** (*merging*) que usa para combinar soluciones parciales.

La fusión de datos genera una secuencia ordenada a partir de un par de secuencias ordenadas.

Entrada: Dos secuencias ordenadas de tamaño n.

Salida: Una secuencia ordenada de tamaño $2n$

$$A = \{1,3,7,8\}$$

$$B = \{2,4,5,9\}$$

$$C = \{1,2,3,4,5,7,8,9\}$$

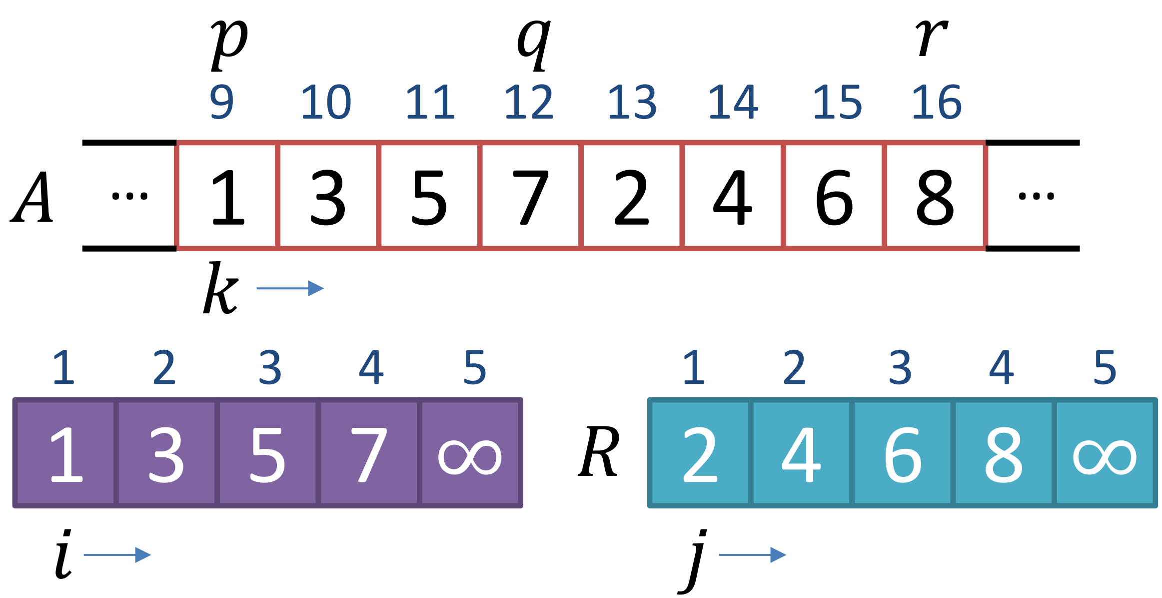
Merge-sort

MERGE(A,p,q,r)

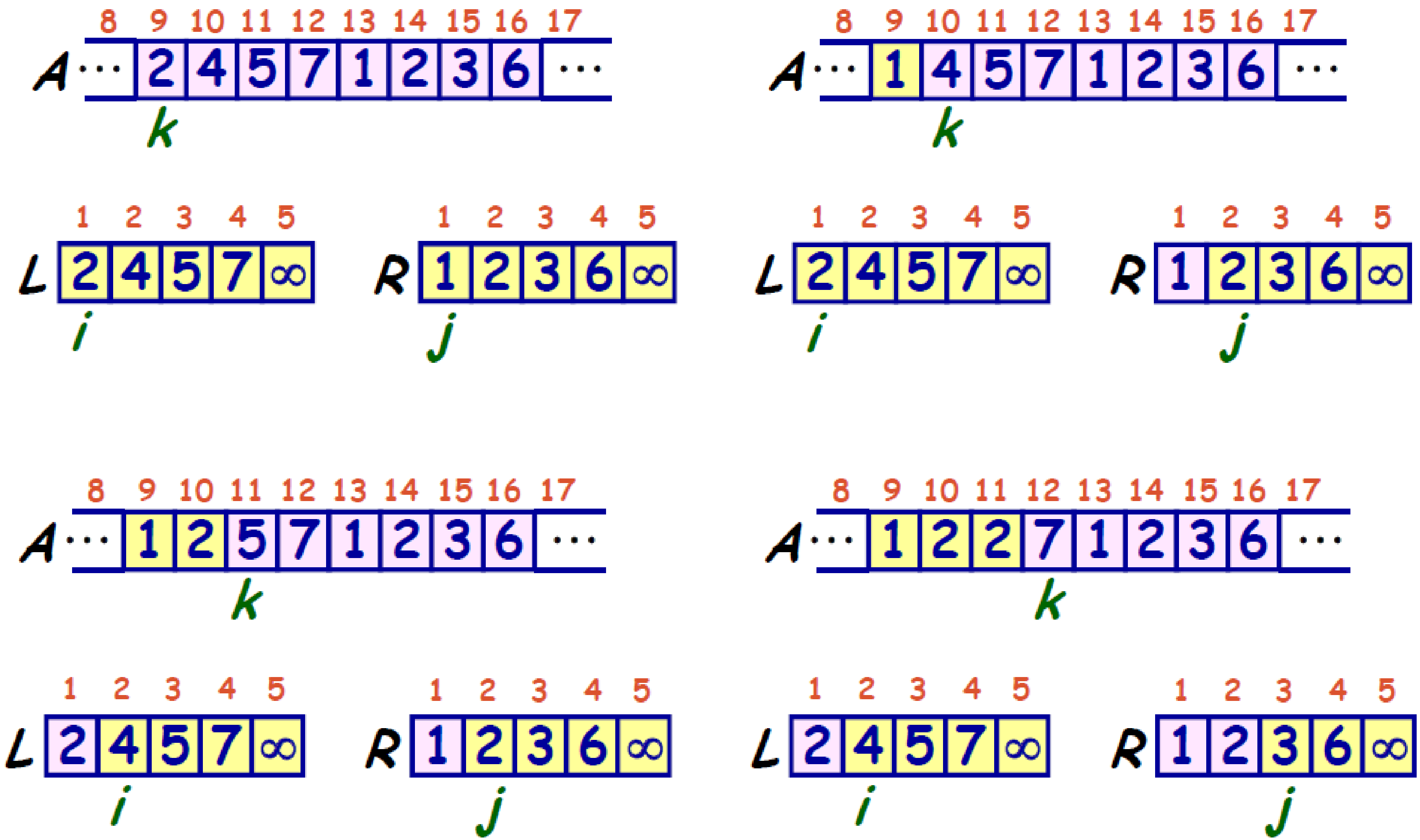
```
n1 = q-p+1
n2 = r-q
Crear arreglos L[n1+1] y R[n2+1]
desde i=0 hasta (n1-1)
    L[i]=A[p+i]
desde i=0 hasta (n2-1)
    R[i]=A[q+i+1]
L[n1] = ∞
/* ∞ representa un valor
muy alto! Mayor que
cualquiera de A */
R[n2] = ∞
i=0
j=0
desde k=p hasta r
    si(L[i]<=R[j])
        A[k]=L[i]
        i=i+1
    sino
        A[k]=R[j]
        j=j+1
```

MERGESORT(A,p,r)

```
si(p<r)
    q=(p+r)/2
    MERGESORT(A,p,q)
    MERGESORT(A,q+1,r)
    MERGE(A,p,q,r)
```



Merge-sort

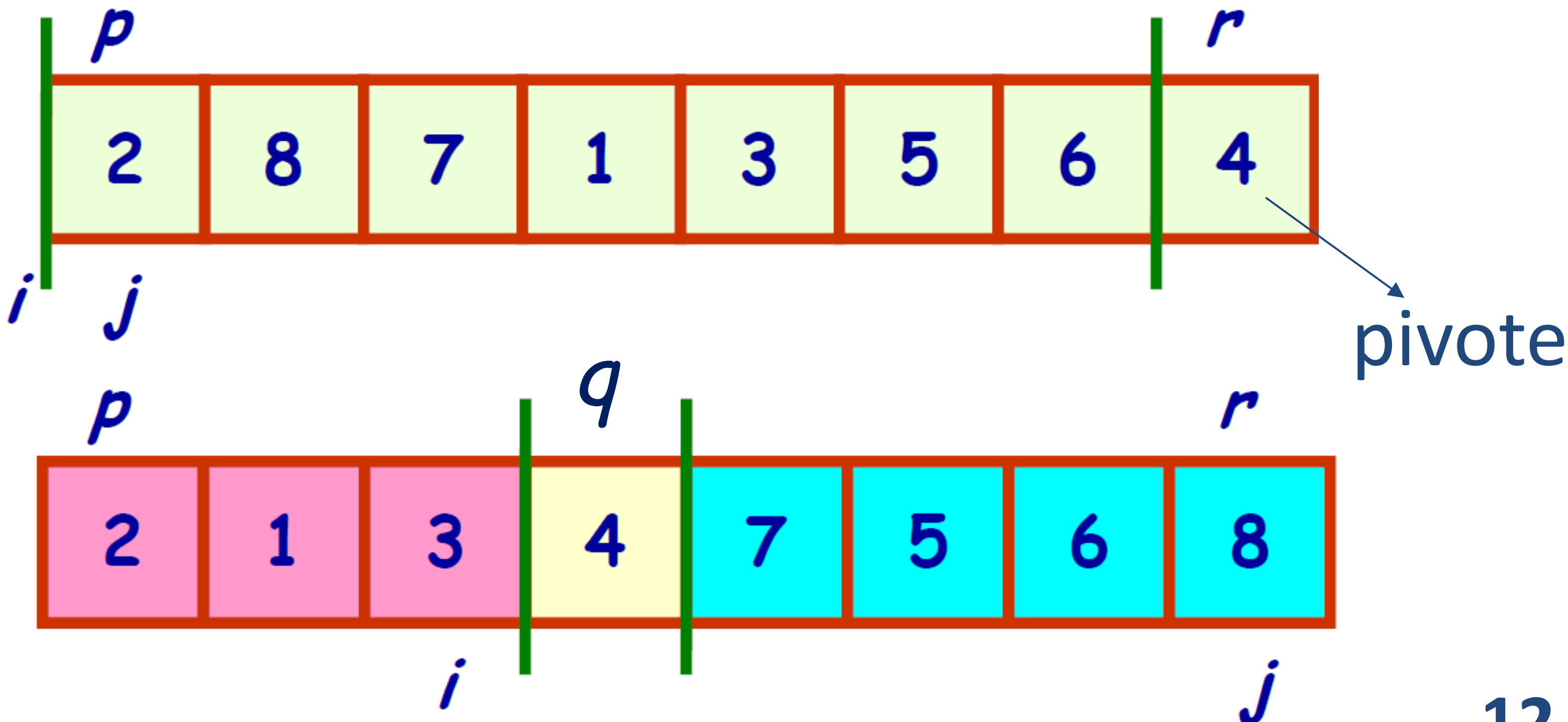


8 9 10 11 12 13 14 15 16 17
A ··· 2 4 5 7 1 2 3 6 ···
k
1 2 3 4 5
L 2 4 5 7 ∞
i
1 2 3 4 5
R 1 2 3 6 ∞
j
8 9 10 11 12 13 14 15 16 17
A ··· 1 4 5 7 1 2 3 6 ···
k
1 2 3 4 5
L 2 4 5 7 ∞
i
1 2 3 4 5
R 1 2 3 6 ∞
j
8 9 10 11 12 13 14 15 16 17
A ··· 1 2 5 7 1 2 3 6 ···
k
1 2 3 4 5
L 2 4 5 7 ∞
i
1 2 3 4 5
R 1 2 3 6 ∞
j
8 9 10 11 12 13 14 15 16 17
A ··· 1 2 2 7 1 2 3 6 ···
k
1 2 3 4 5
L 2 4 5 7 ∞
i
1 2 3 4 5
R 1 2 3 6 ∞
j

Quicksort

El Quicksort también es un algoritmo de ordenamiento recursivo, y es uno de los más empleados debido a rapidez. Sus pasos son:

1. Se divide el arreglo $A[p \dots r]$ en dos subarreglos $A[p \dots q-1]$ y $A[q+1 \dots r]$ con la siguiente propiedad: cada elemento del arreglo $A[p \dots q-1]$ es menor o igual que $A[q]$ y por consiguiente menor o igual que cualquier elemento del arreglo $A[q+1 \dots r]$. Se calcula el índice q en este paso.



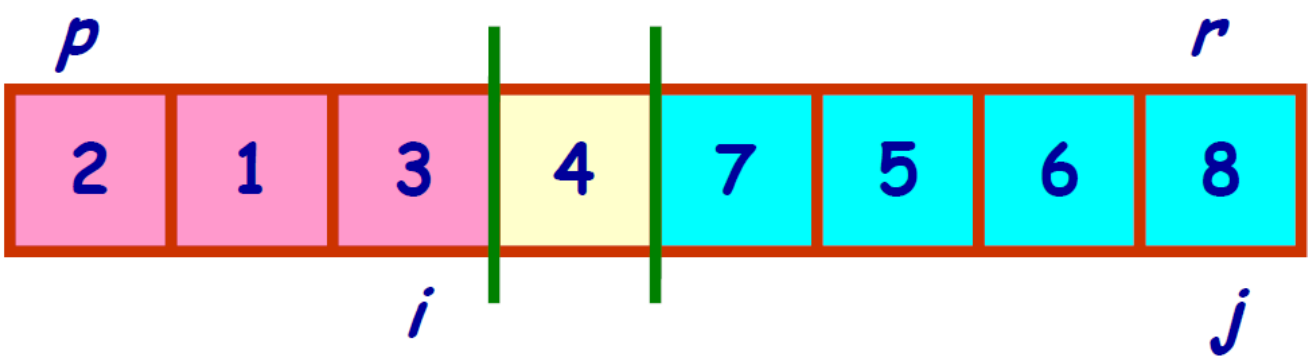
Quicksort

El Quicksort también es un algoritmo de ordenamiento recursivo, y es uno de los más empleados debido a rapidez. Sus pasos son:

1. Se divide el arreglo $A[p \ldots r]$ en dos subarreglos $A[p \ldots q-1]$ y $A[q+1 \ldots r]$ con la siguiente propiedad: cada elemento del arreglo $A[p \ldots q-1]$ es menor o igual que $A[q]$ y por consiguiente menor o igual que cualquier elemento del arreglo $A[q+1 \ldots r]$. Se calcula el índice q en este paso.

2. Los dos subarreglos $A[p \ldots q-1]$ y $A[q+1 \ldots r]$ se ordenan en forma recursiva utilizando el mismo algoritmo Quicksort.

3. Debido a que los subarreglos están ordenados en el lugar correcto, no es necesario realizar ninguna operación adicional. Por lo tanto el arreglo $A[p \ldots r]$ está ordenado.

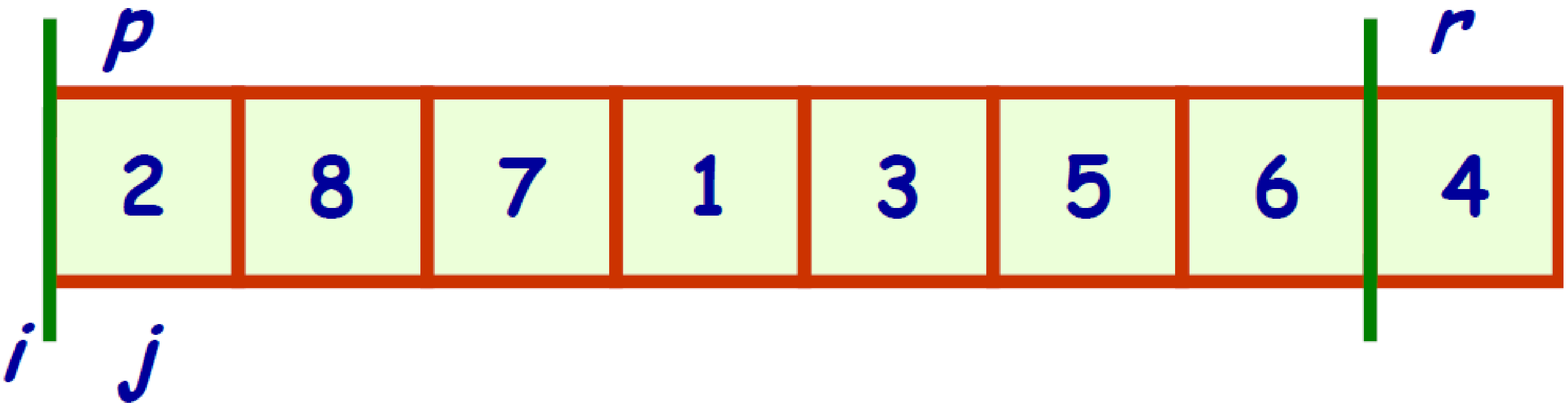


Quicksort - Seudocódigo

```
QUICKSORT(A, p, r)
if p < r
    q = PARTITION(A, p, r)
    QUICKSORT(A, p, q − 1)
    QUICKSORT(A, q + 1, r)
```

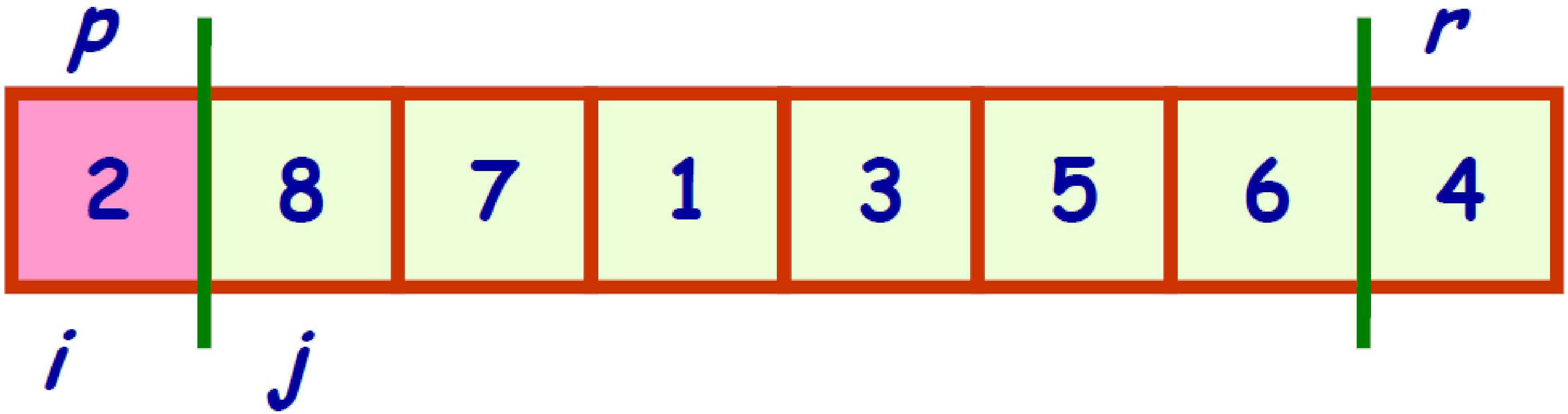
```
PARTITION(A, p, r)
x = A[r]
i = p − 1
for j = p to r − 1
    if A[j] ≤ x
        i = i + 1
        exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```

Quicksort - Funcionamiento



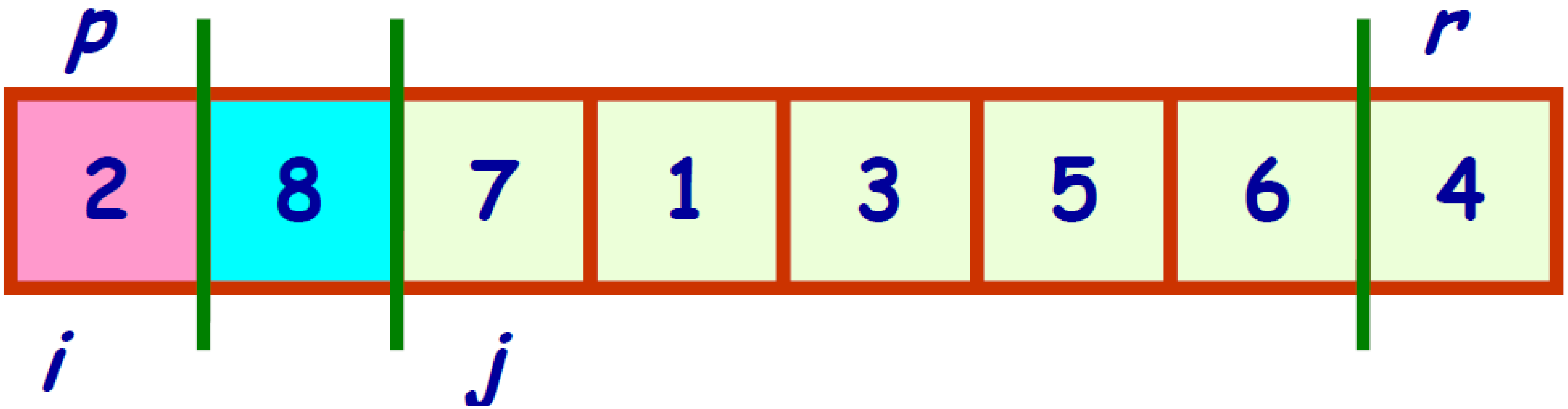

```
PARTITION(A, p, r)
x = A[r]
i = p - 1
for j = p to r - 1
    if A[j] ≤ x
        i = i + 1
        exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```

Quicksort - Funcionamiento



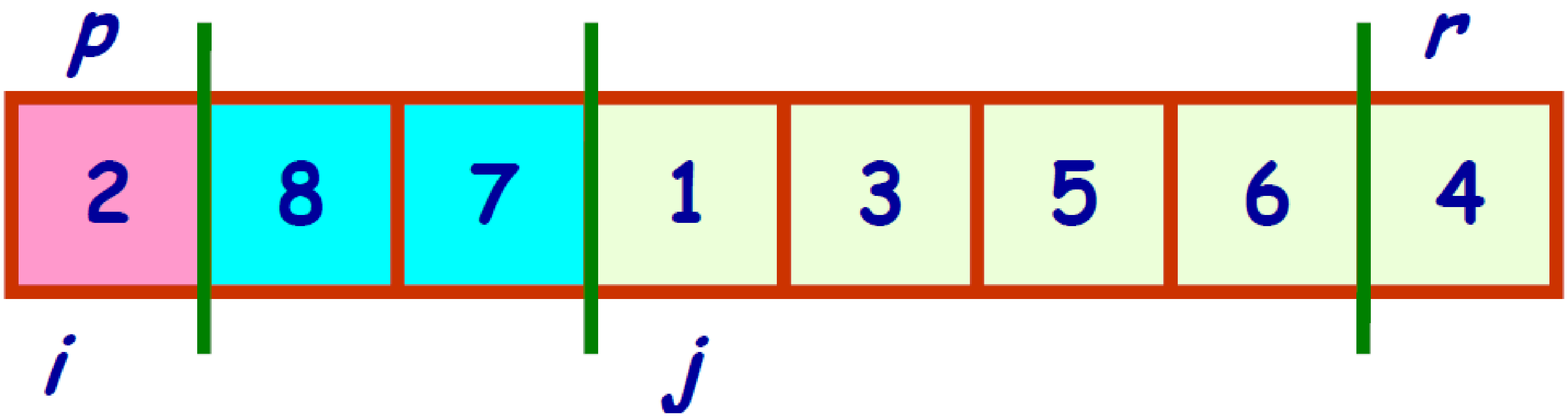

```
PARTITION(A, p, r)
x = A[r]
i = p − 1
for j = p to r − 1
    if A[j] ≤ x
        i = i + 1
        exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```

Quicksort - Funcionamiento



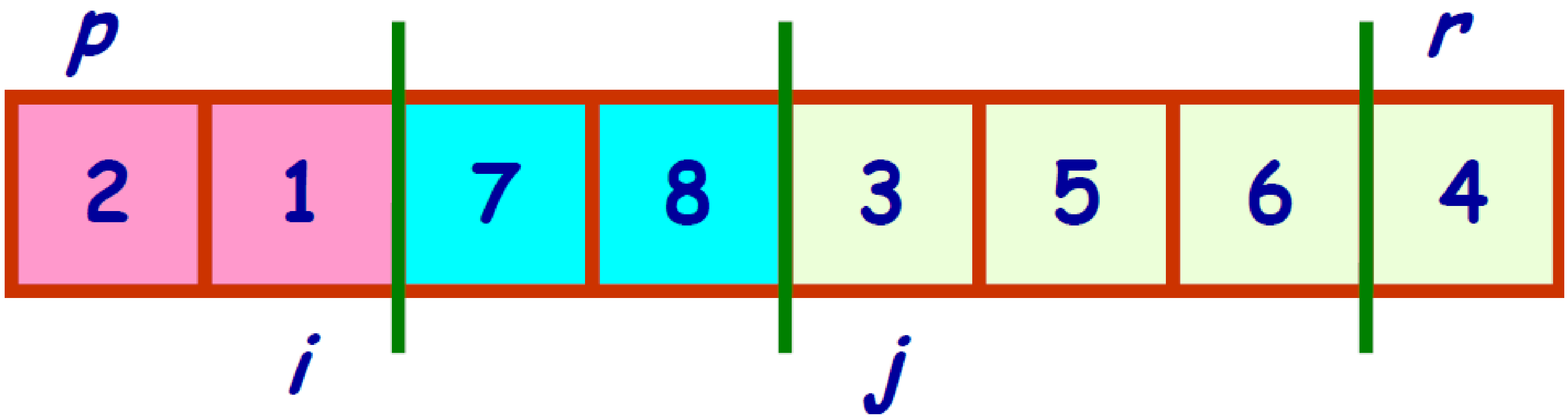

```
PARTITION(A, p, r)
x = A[r]
i = p − 1
for j = p to r − 1
        if A[j] ≤ x
                i = i + 1
                exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```

Quicksort - Funcionamiento



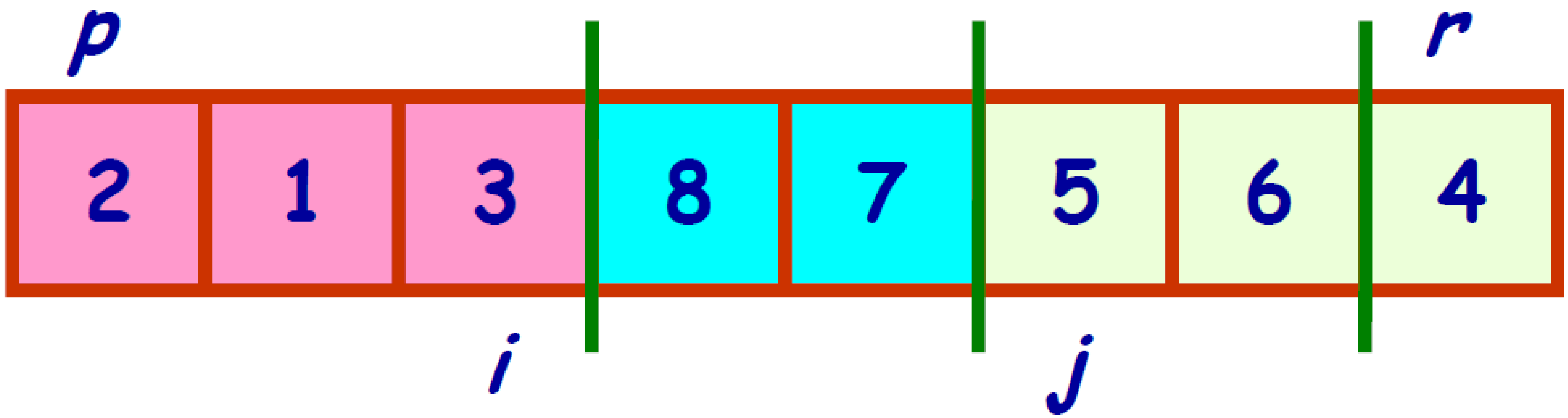

```
PARTITION(A, p, r)
    x = A[r]
    i = p − 1
    for j = p to r − 1
        if A[j] ≤ x
            i = i + 1
            exchange A[i] with A[j]
    exchange A[i + 1] with A[r]
    return i + 1
```

Quicksort - Funcionamiento



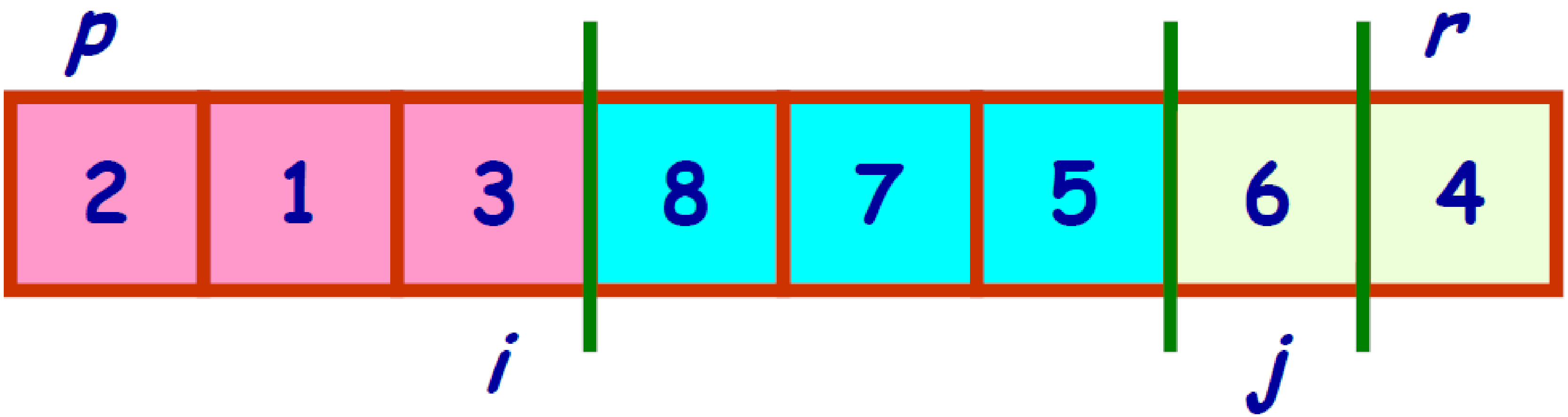

```
PARTITION(A, p, r)
x = A[r]
i = p − 1
for j = p to r − 1
      if A[j] ≤ x
            i = i + 1
            exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```

Quicksort - Funcionamiento



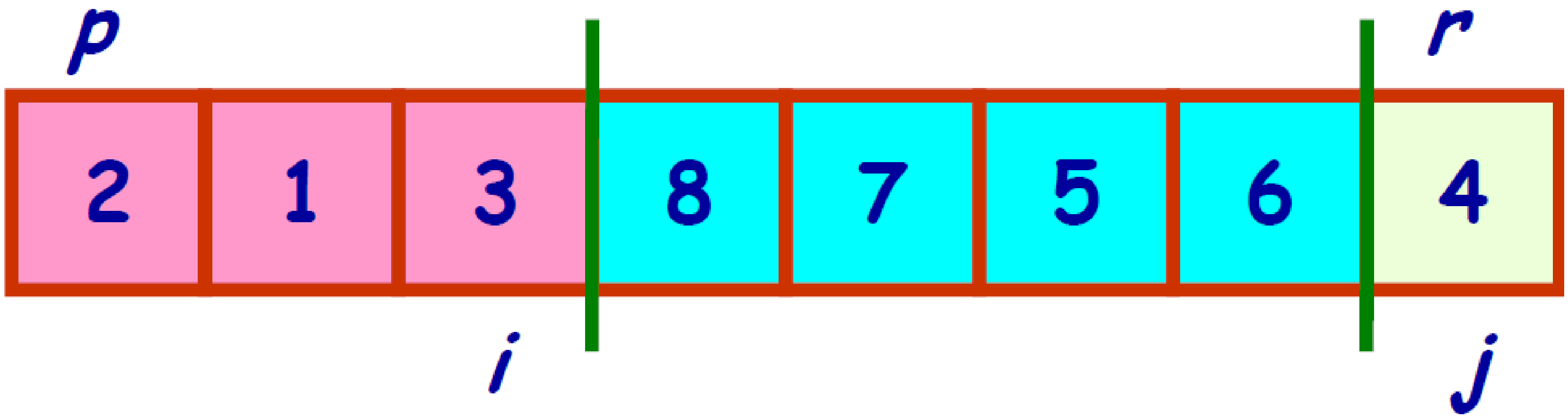

```
PARTITION(A, p, r)
x = A[r]
i = p − 1
for j = p to r − 1
    if A[j] ≤ x
        i = i + 1
        exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```

Quicksort - Funcionamiento




```
PARTITION(A, p, r)
x = A[r]
i = p − 1
for j = p to r − 1
    if A[j] ≤ x
        i = i + 1
        exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```

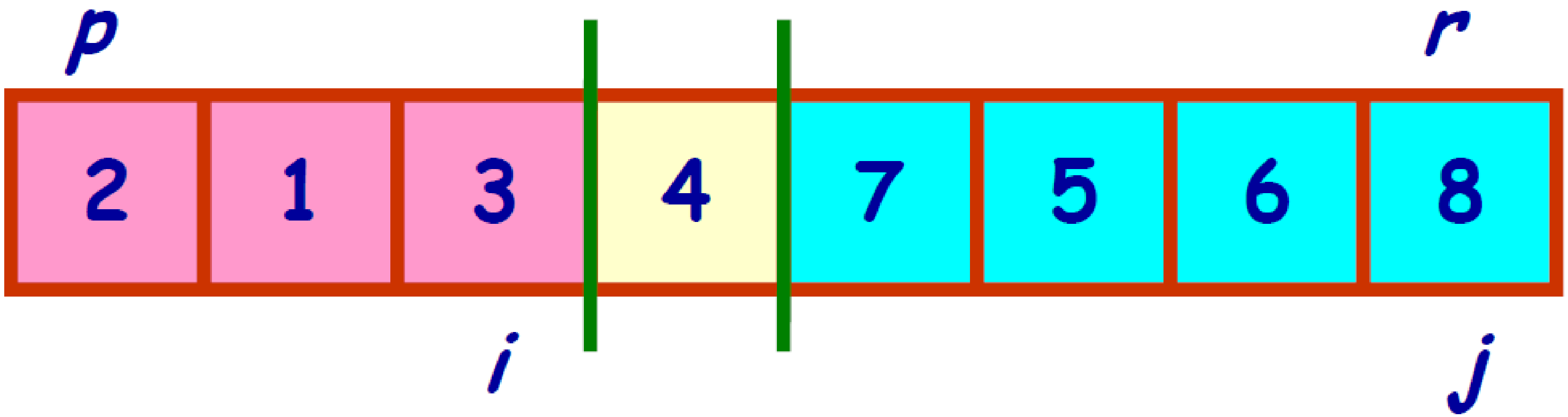
Quicksort - Funcionamiento



PARTITION(A, p, r)

```
x = A[r]
i = p - 1
for j = p to r - 1
    if A[j] ≤ x
        i = i + 1
        exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```

Quicksort - Funcionamiento




```
PARTITION(A, p, r)
x = A[r]
i = p − 1
for j = p to r − 1
    if A[j] ≤ x
        i = i + 1
        exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```

Timsort

El algoritmo Timsort es el método empleado por defecto para el ordenamiento en Python (`sorted()` y `.sort()`).

Este método es híbrido y consiste en la combinación del método de **inserción** y el **Mergesort**.

La característica principal de Timsort es que aprovecha la ventaja de tener elementos *casi ordenados* en un conjunto de datos, lo cual aparece en muchas situaciones en la práctica.

El algoritmo realiza lo siguiente:

- Separa el conjunto de datos en bloques de tamaño fijo (entre 32 y 64 elementos).
- Ordena cada bloque mediante el método de inserción.
- Ordena el conjunto de datos mediante Mergesort.

https://realpython.com/sorting-algorithms-python/#the-timsort-algorithm-in-python
https://en.wikipedia.org/wiki/Timsort

Gracias por la atención