Dividir i vèncer

Algorísmica

Grau d'Enginyeria Informàtica

Mireia Ribera, ribera@ub.edu
Daniel Ortiz, daniel.ortiz@ub.edu

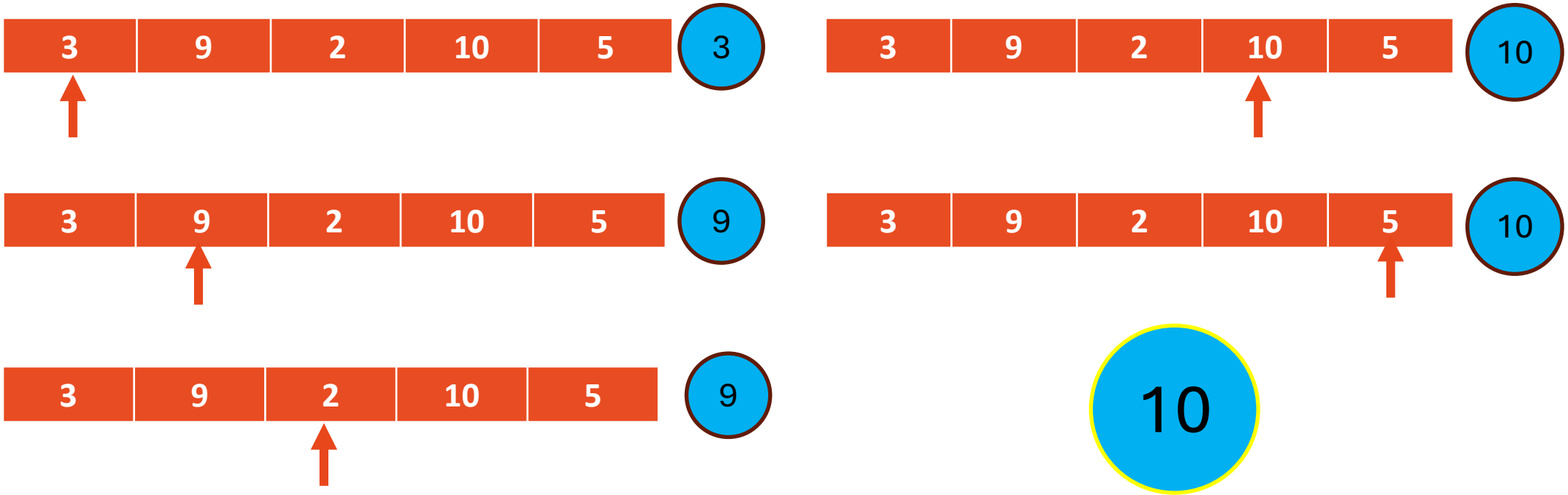
Dividir i vèncer: Conceptes clau

1. L'estratègia: dividir, vèncer i combinar
2. Complexitat: El *Teorema Mestre*
3. Algorismes d'ordenació: mergesort i quicksort

Una altra manera d'aproximar un problema
L'exemple de trobar el màxim

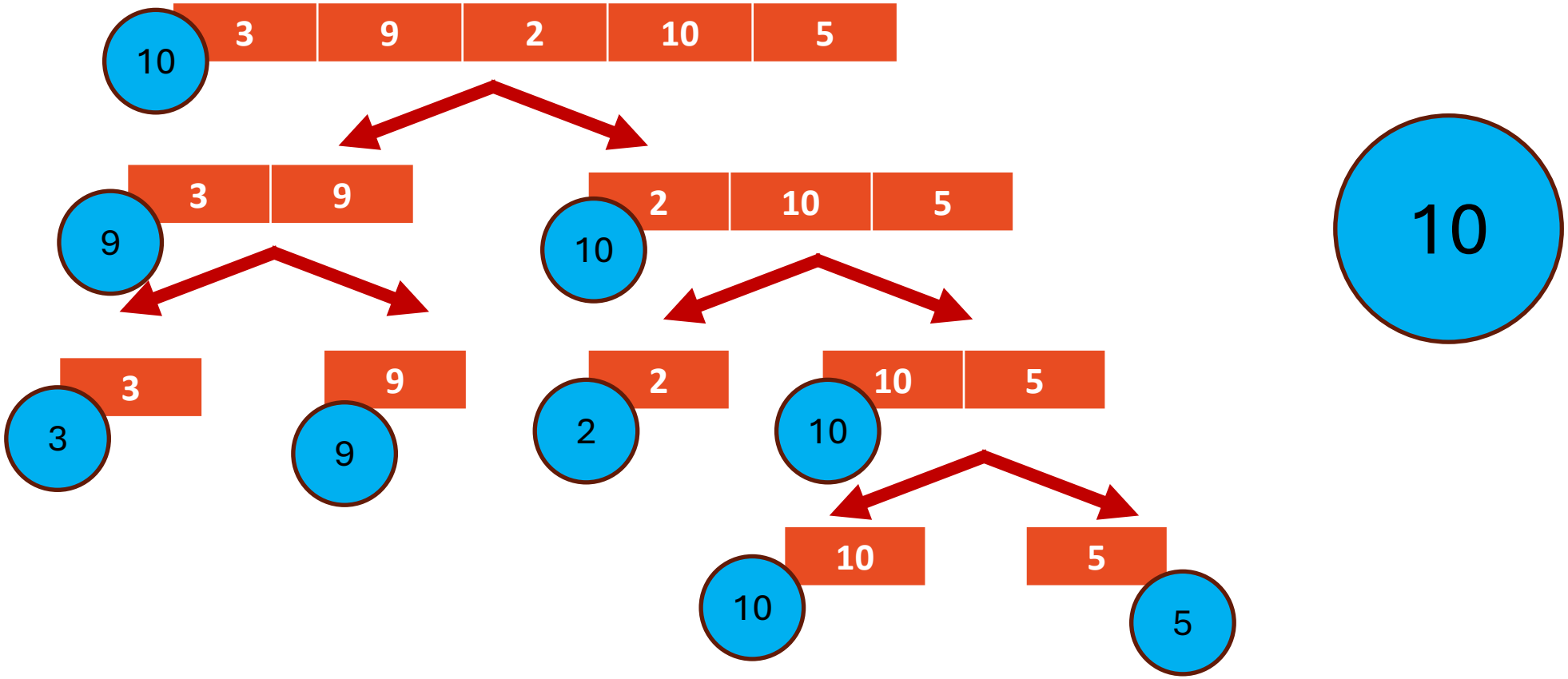
El problema

Donada una llista de n nombres, troba el màxim element. Per exemple dins [3,9,2,7,5] el màxim és 9. La manera intuïtiva de resoldre el problema seria de manera seqüencial:



Trobar el màxim. Resolució recursiva

Podem reformular el problema i anar-lo dividint en problemes molt petits fins que arribem a un cas molt simple, que resolem, per després reconstruir el resultat del problema global.



Trobar el màxim. Comparant les dues estratègies

En aquest cas l'aproximació seqüencial i l'estratègia de dividir tenen un cost equivalent, però sovint l'estratègia de dividir pot ser molt eficient!

Ambues fan n comparacions!

L'estratègia: dividir, vèncer i combinar

Dividir i vèncer (*divide and conquer*) és una estratègia de resolució de problemes que consisteix en:

- Dividir un problema en un o més subproblemes que són instàncies del mateix problema amb una entrada més petita.
- Vèncer: Amb les divisions arribem a un problema que sabem resoldre. Hem vençut el problema!
- Combinar adequadament les solucions dels subproblemes per trobar la solució del problema original.

L'esquema general d'aquests algorismes és:

Tenim un problema de mida n, que reformulem mitjançant la solució d'a problemes de mida n/b i llavors combinem les respostes en un temps $O(n^d)$

Dividir, conquerir i combinar a l'exemple del màxim

- Dividim la llista per 2 a cada crida.
- Vencem el problema, perquè quan la llista té 1 element, la solució és trivial.
- Combinem les solucions esquerra i dreta mirant quina és més gran.

En aquest cas:

- a, el nombre de subproblemes, és 2,
- b, el divisor de l'entrada, és 2 i
- d, l'exponent del cost de combinar les solucions, és 0.

El Teorema Mestre (Master Theorem)

Tenim un problema de mida n, que reformulem mitjançant la solució d'a problemes de mida n/b i llavors combinem les respostes en un temps $O(n^d)$.

La complexitat d'aquest tipus de problema es resol pel *Teorema Mestre*:

$$T(n) = aT(n/b) + O(n^d)$$

Si $T(n) = aT(n/b) + O(n^d)$ per algunes constants $a > 0$, $b > 1$, i $d >= 0$, llavors:

- Cas 1: $T(n) = O(n^d)$ si $a < b^d$, domina el pes de la combinació,
- Cas 2: $T(n) = O(n^d log \quad \mathrm{n})$ si $a = b^d$, ambdós pesos estan equilibrats,
- Cas 3: $T(n) = O(n^{log_b \, \mathrm{a}})$ si $a > b^d$, domina el pes dels subproblemes.

Aplicació del Teorema Mestre a l'exemple del màxim

En aquest cas: n és la mida de la llista,

- a, el nombre de subproblemes, és 2,
- b, el divisor de l'entrada és 2 i
- d, l'exponent del cost de combinar les solucions és 0.

$a > b^d,\ 2 > 2^0$, per tant estem en el cas 3, i $T(n) = O\left(n^{log_b\, a}\right)$, és a dir $T(n) = O\left(n^{log_2\, 2}\right),\ T(n) = O(n)$.

Altres exemples del Teorema Mestre

a	b	d	Cas Teorema Mestre	T(n)	O()
3	4	2	$a < b^d \rightarrow$ Cas 1 (combinació)	$O(n^d)$	$O(n^2)$
2	2	1	$a = b^d \rightarrow$ Cas 2 (iguals)	$O(n^d log \quad n)$	$O(n \; log \quad n)$
8	2	2	$a > b^d \rightarrow$ Cas 3 (subproblemes)	$O(n^{log_b \, a})$	$O(n^{log_2 \, 8}) = O(n^3)$

Algorismes d'ordenació eficients: mergesort i quicksort

Ordenar vol dir posar els elements de menor a major en una seqüència segons un determinat criteri.
En el tema 5 de força bruta havíem vist l'ordenació per selecció, d'$O(n^2)$.

Aquí veurem dos mètodes d'ordenació eficient, Mergesort i Quicksort, que tenen una complexitat d'$O(n\ log\, n)$, perquè ambdós usen una estratègia de dividir i vèncer amb 2 subproblemes de mida $n/2$ i amb un cost de combinar-los igual a n^1

a	b	d	Cas	T(n)	O()
2	2	1	$a = b^d \rightarrow$ Cas 2 (iguals)	$O(n^d log \quad \mathrm{n})$	$O(n\ log\, \mathrm{n})$

Python usa Quicksort per al mètode *.sort()*.

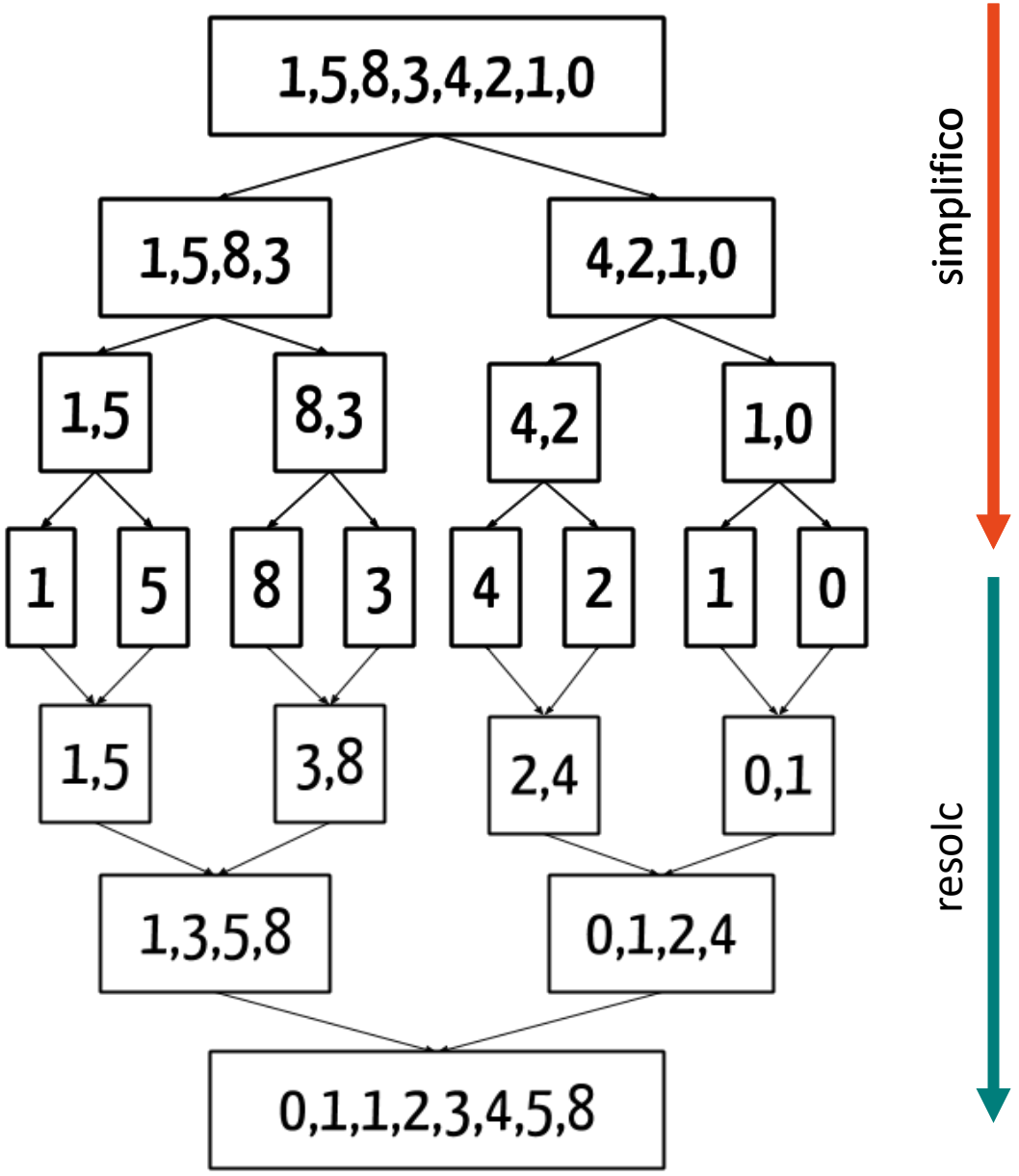
Mergesort: model recursiu

Si jo tinc dues piles de cartes i les vull barrejar ordenades, quin és el cas trivial?

- Que a cada pila hi hagi una o cap carta

I un cop resolt el cas trivial, com puc combinar les solucions parcials?

- Perquè ara sé que les piles em venen ordenades.



Mergesort: codi. Funció auxiliar merge

```python
def mergesort(l:list[any])->list[any]:
    if len(l) < 2:
        return l
    else:
        meitat = len(l) // 2
        esquerra = mergesort(l[:meitat])
        dreta = mergesort(l[meitat:])
        return merge(esquerra, dreta)
```

Simplifico, 2 subproblemes, de mida n/2

combino amb merge, O(n)

Resolc O(n)

```python
def merge(x:list[any],y:list[any])->list[any]:
    if len(x) < 1: return y
    if len(y) < 1: return x
    if x[0] <= y[0]:
        return [x[0]] + merge(x[1:],y)
    else:
        return [y[0]] + merge(x,y[1:])
```

(millor usar la versió no-recursiva del notebook de codi)

Mergesort: complexitat

La n és la mida de la llista,

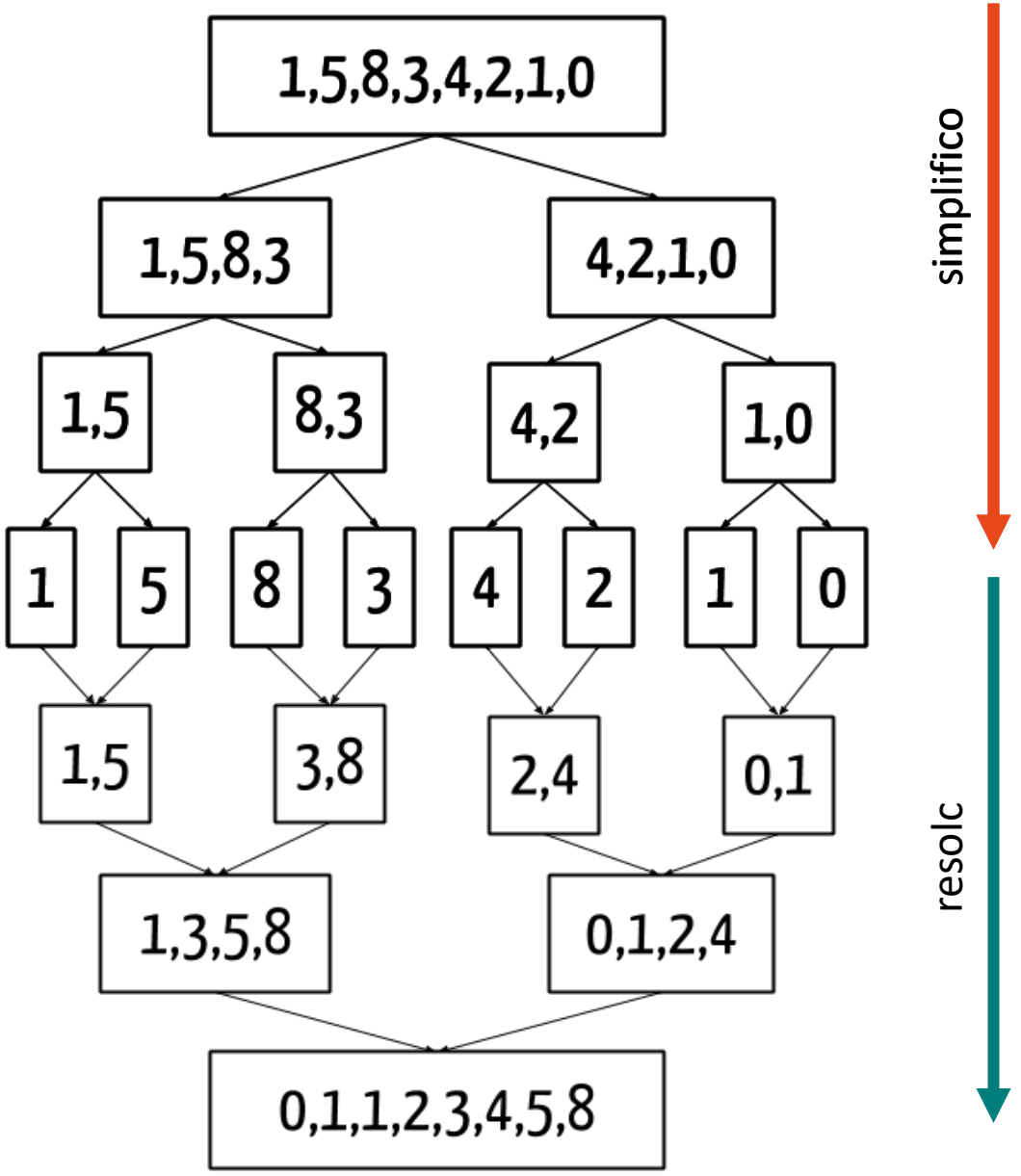
- a, és 2
- b, és 2 i
- d, és 1.

$a = b^d$, $2 = 2^1$, per tant estem en el cas 2, i

$T(n) = O(n^d log \quad n)$, és a dir

$T(n) = O(n^1 log \quad n)$, $T(n) = O(n \ log \quad n)$.

n	Ordenació per selecció $O(n^2)$	Mergesort $O(n \log n)$
8	64	24
32	1024	160
1024	1048576	10240



Quicksort: model recursiu i funció partition

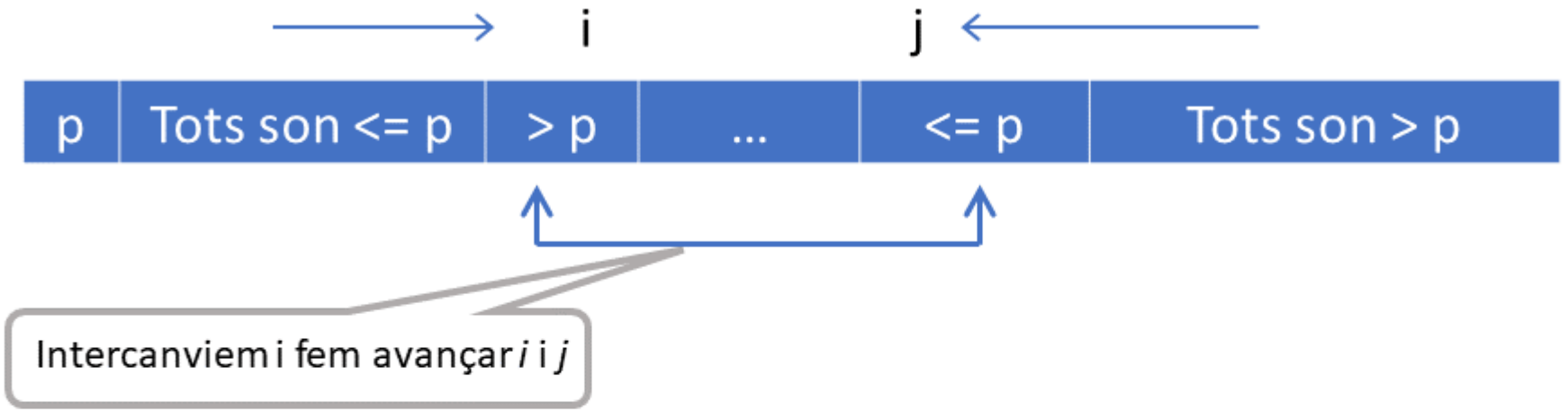
Es basa en anar dividint la llista en 2 i en deixar un element ben col·locat al mig d'ambdues llistes, fins a tenir tots els elements ben col·locats.

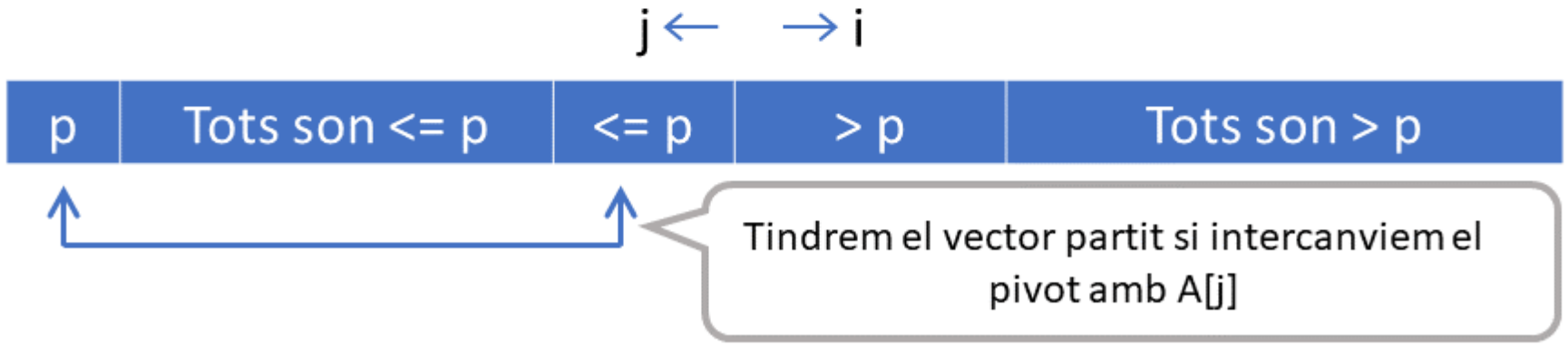
- L'algorisme que col·loca l'element i reordena els elements és un algorisme auxiliar, que s'anomena *partition*.
- L'algorisme que crida partition i divideix la llista és el propi Quicksort.

Quicksort: algorisme auxiliar *partition*

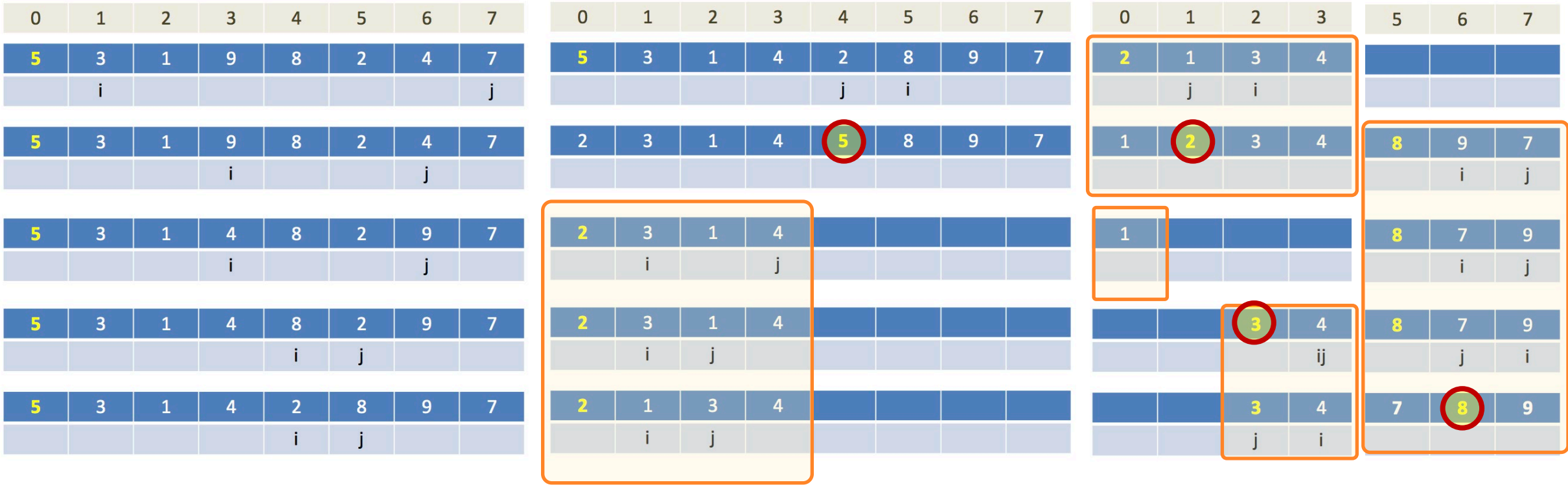
1. Primer seleccionem un element, respecte del qual dividirem la llista. L'anomenem pivot. Per exemple pivot podria ser A[0]
2. Reordenem per aconseguir una partició. Això ho podem fer amb dues passades (d'esquerra a dreta i de dreta a esquerra) de la llista.
    1. La passada d'esquerra a dreta (i) comença pel segon element i no s'atura fins trobar un element més gran que el pivot (p).
    2. La passada de dreta a esquerra (j) comença per l'últim element i s'atura quan troba un element més petit o igual que el pivot.

Quicksort: estratègia de partition





Quicksort: funcionament i funció partition



Partition col·loca un element (el pivot) de la llista a la seva posició definitiva i deixa a l'esquerra elements més petits que el pivot, i a la dreta elements més grans que el pivot. Quicksort, va fent crides recursives amb les parts esquerra i dreta resultants.

Quicksort: codi. Funció auxiliar partitition

```python
def quick_sort(A:list[any]):
    quick_sort_r(A, 0, len(A) - 1)

def quick_sort_r(A:list[any], p:int, d:int):
    if d > p:
        piv:int = partition(A, p, d)
        quick_sort_r(A, p, piv - 1)
        quick_sort_r(A, piv + 1, d)
```

Quicksort no crea una nova llista, modifica la que li passem.

```python
def partition(A:list[any], p:int, d:int):
    piv:int = p
    i = p + 1
    j = d
    indexs_creuats:bool = False
    while not(indexs_creuats):
        while i <= d and A[i] <= A[piv]: i += 1
        while j >= p and A[j] > A[piv]: j -= 1
        if i >= j:
            indexs_creuats = True
        else:
            A[i], A[j] = A[j], A[i]
    A[j], A[piv] = A[piv], A[j]
    return j
```

Quicksort: complexitat

En el millor cas estem en el cas 2 perquè la llista es divideix en 2 subproblemes de mida n/2, i unir-les costa O(n). Cost $O(n * log_2 n)$

En el pitjor cas la llista no es parteix per la meitat, el pivot és als extrems i el cost serà $O(n^2)$. En el cas promig serà $O(1.38n * log_2 n)$

La complexitat però l'expressarem com a $O(n * log_2 n)$

Notebook de suport

Podeu consultar tot el codi de les transparències a:
Tema6DividirVencer-Codi.ipynb

Ara et toca...

Defineix valors d'a, b i d tals que 2 d'ells corresponguin al cas 1 del Teorema Mestre, 2 al cas 2, i 2 al cas 3. En tots ells, calcula la complexitat.

Ara et toca...

Pistes:
1. Tria una posició aleatòria de la llista i mira'n el valor.
2. Parteix la llista en dos (menors i majors que el valor).
3. Cerca només en una de les subllistes.

Escriu una funció que retorni el k-èssim element més petit d'una llista.

Quina complexitat té?

Gràcies i recapitulació

MIREIA RIBERA | ACCESSIBILITAT DIGITAL
| EXPERIÈNCIA D'USUARI
| VISUALITZACIÓ DE DADES

UNIVERSITAT DE BARCELONA

Recapitulació

➗🏆 Per abordar problemes complexos usem l'estratègia de dividir i vèncer

- 🐣 Hem de pensar un problema trivial a resoldre (vencer)
- 🐓 Hem de saber com dividir el problema actual per fer-lo més petit i arribar al problema trivial
- 🪺 Hem de pensar com combinar les solucions senzilles fins arribar als casos més complexos.

👩🏿‍🏫 El Teorema Mestre ens diu com calcular la complexitat dels problemes de dividir i vèncer.

- 🧮 El càlcul té en compte el que costa resoldre els subproblemes, ajuntar-los o ambdós.

🔀 Quicksort i Mergesort són dos algorismes d'ordenació de complexitat $O(n * log\,\mathrm{n})$

🐍 Mergesort és millor, però Python usa Quicksort, perquè es va descobrir primer.