cápsula viernes 29 marzo : Quick sort.

4. Quick-sort.

" tomar los 'n' pivotes de un arreglo".

¿ que es un pivote?

Si suponemos que no hay elementos iguales al pivote, elegir un pivote p particiona la secuencia en

- Elementos mayores al pivote, $M = \{a_i \mid a_i > p\}$
- Elementos menores al pivote, $m = \{a_i \mid a_i < p\}$

} el pivote queda en su posición ordenada

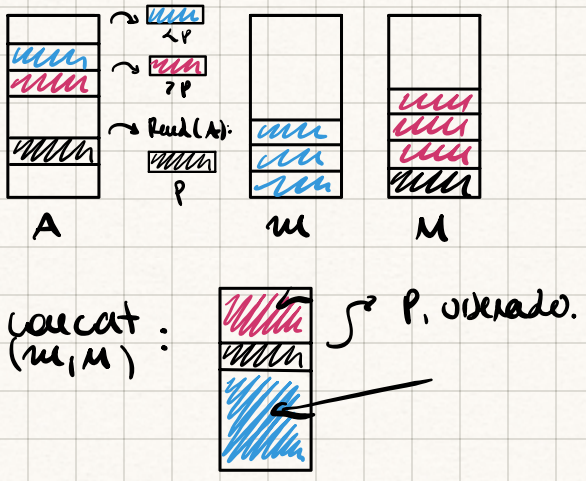
Operación Partition (obtener un pivote).

suponiendo elementos no-repetidos:

input : Secuencia $A[0, \ldots, n-1]$, índices i, f
output: Índice del pivote aleatorio en la secuencia ordenada

`Partition` (A, i, f):

```
1  p ← pivote aleatorio en A[i, f]
2  m, M ← secuencias vacías
3  Insertar p en M
4  for x ∈ A[i, f] :
5      if x < p : Insertar x en m
6      elif x > p : Insertar x en M
7  A[i, f] ← Concat(m, M)
8  return i + |m|
```



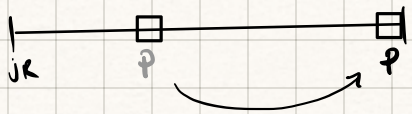
Versión in-place:

input : Arreglo $A[0, \ldots, n-1]$, índices i, f
output: Índice del pivote aleatorio en la secuencia ordenada

`Partition` (A, i, f):

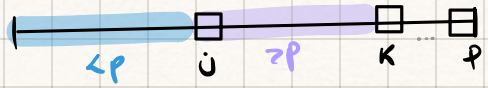
```
1  x ← índice aleatorio en {i, ..., f}
2  p ← A[x]
3  A[x] ⇄ A[f]
4  j ← i
5  for k = i ... f − 1 :
6      if A[k] < p :
7          A[j] ⇄ A[k]
8          j ← j + 1
9  A[j] ⇄ A[f]
10 return j
```

visualización ↓



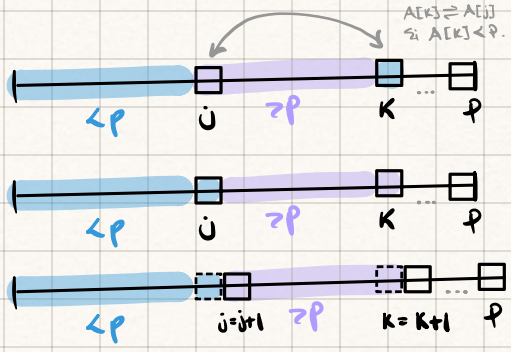
primero reservamos 'p' a un lado

j,K son barreras ; 'K' delimita los valores revisados , 'j' separa los elementos mayores de los menores a 'p'.



caso 1: A[K] < p

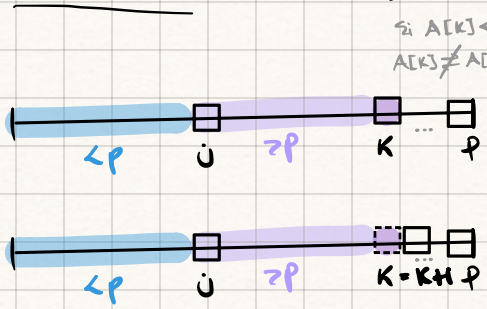
" El valor que estamos revisando es menor que 'p' ".



- Si en K hay un elemento menor ; se lo intercambia con 'j' y luego avanza la 'barrera' para acomodar al elemento nuevo.

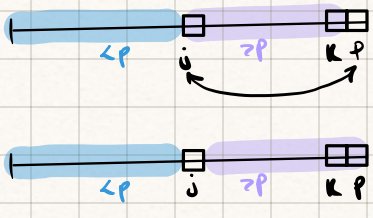
caso 2: A[K] ≥ p

" El valor que estamos revisando es mayor o igual a 'p' ".



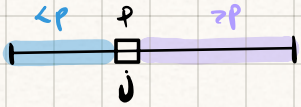
- Avanza solo K

al final: (K = N-2)



se intercambia 'p' con 'j'

Resultado:



un pivote ordenado. j

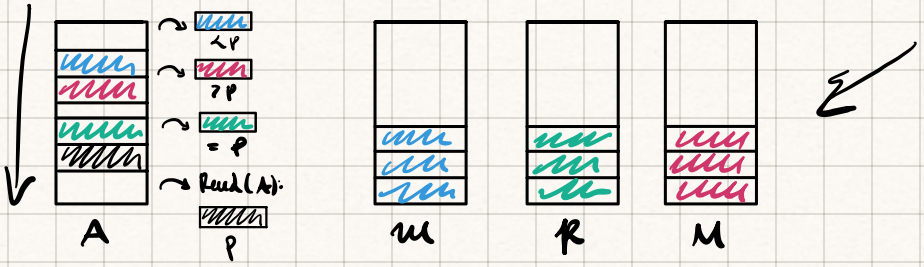
Ejercicio 1.

1) Escribe el algoritmo *quicksort3*, que, en lugar de particionar el arreglo A en dos, como lo hace *quicksort*, lo particiona en tres: datos menores que el pivote, datos iguales al pivote, y datos mayores que el pivote. Puedes suponer que las particiones van a parar a listas diferentes o bien al mismo arreglo —especifica. Usa una notación similar a la usada en las diapositivas.

Solución: Pueden haber muchos algoritmos correctos. Las distribución de puntaje se hizo de la siguiente forma:

- [1 pto.] Que se haya hecho en pseudocódigo y sin funciones específicas de ningún lenguaje, es decir, usando la notación que se usa en las diapositivas.
- [3 ptos.] Haya separación correcta entre los menores, iguales y mayores y que se vaya reordenando la lista según corresponda.
- [1 pto.] La nueva implementación de partition retorne dos pivotes.
- [1 pto.] Que el algoritmo termine de forma correcta.

Forma 2: insertar 3 secuencias



concat (m, M): → P, ordenado.



input : Secuencia $A[0,\ldots,n-1]$, índices i, f
output: Índice del pivote aleatorio en la secuencia ordenada

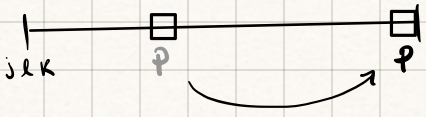
`Partition` (A, i, f):

```
1  p ← pivote aleatorio en A[i, f]
2  m, M ← secuencias vacías        ← R
3  Insertar p en M
4  for x ∈ A[i, f] :
5      if x < p : Insertar x en m
6      elif x > p : Insertar x en M
7  A[i, f] ← Concat(m, M)
8  return i + |m|
```

x = p : insertar en R

concat (m, P, R, M)

Forma 2: Algoritmo in-place



primero reservamos 'p' a un lado

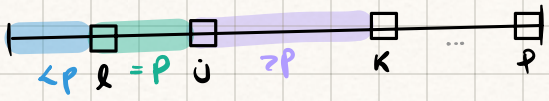
"Nos basamos en el esquema anterior"

j, l, K son barreras ;

'K' delimita los valores revisados

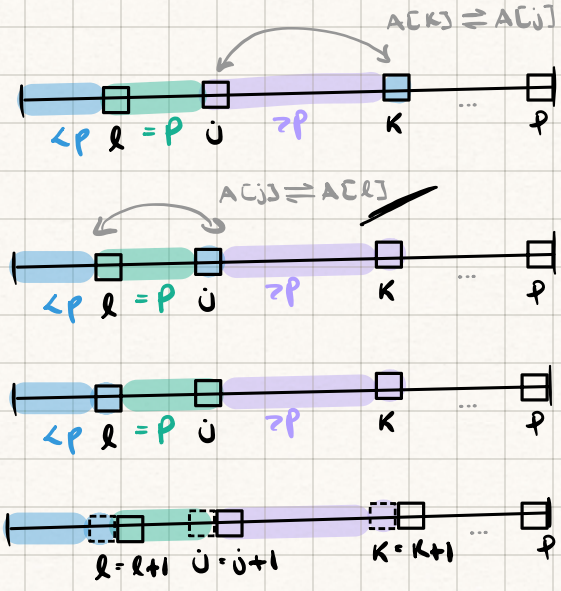
'j' separa los elementos mayores de los menores/iguales a 'p'.

'l' " iguales de los menores.



caso 1: A[K] < p

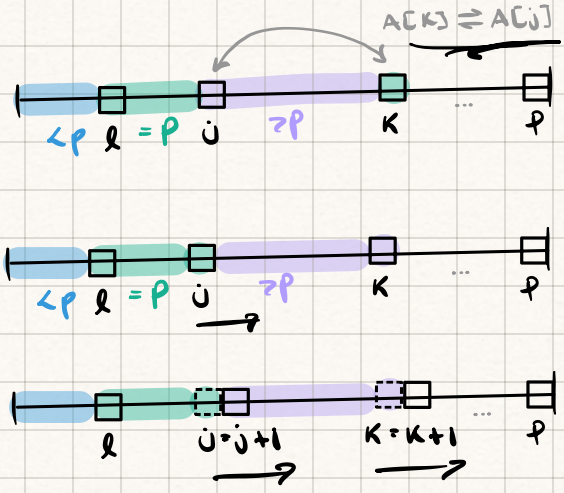
"El valor que estamos revisando es menor que 'p'".



- Si en K hay un elemento menor; se lo intercambia con 'j', luego con 'l' y avanza 'j' y 'K' para acomodar al nuevo elemento.

caso 2: A[K] = p

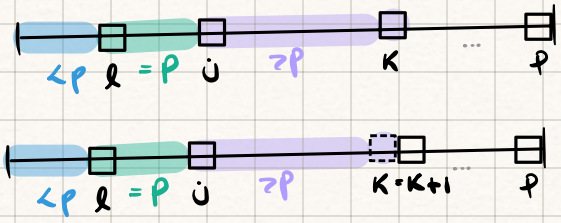
"El valor que estamos revisando es igual que 'p'".



↑ 'l' se queda donde mismo porque sigue delimitando bien

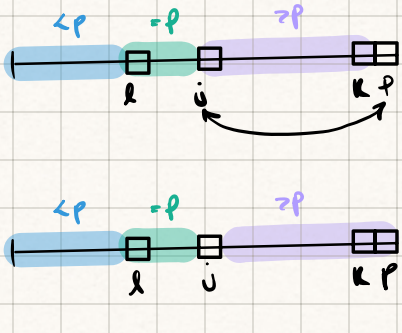
Caso 3: A[K] > P

"El valor que estamos revisando es mayor o igual a 'p'".

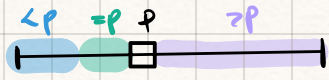


- Avanza solo K

al final: (K = N-2)



Resultado:



un segmento ordenado.

Pseudo código

input : Arreglo $A[0,\ldots,n-1]$, índices i, f
output: Índice del pivote aleatorio en la secuencia ordenada

`Partition` (A, i, f):

```
1   x ← índice aleatorio en {i, ..., f}
2   p ← A[x]
3   A[x] ⇄ A[f]
4   j ← i      l ← i
5   for k = i ... f − 1 :
6       if A[k] < p :
7           A[j] ⇄ A[k]
8           j ← j + 1
9   A[j] ⇄ A[f]
10  return j
```

modificar

```
for K = i ... f-1 :
    if A[K] < p :
        ...
    if A[K] = p :
        ...
    else :
        ...
```

ver siguiente paso.

```
Partition3 (A, i, f)
    x ← random(A)
    p ← A[x]
    A[x] ⇌ A[f]
    j ← i
    l ← j
    for k: i ... f-1 :
        if A[k] < p :
            A[k] ⇌ A[j]
            A[j] ⇌ A[l]
            j ← j+1
            l ← l+1
        if A[k] = p :
            A[k] ⇌ A[j]
            j ← j+1
    A[j] ⇌ A[f]
    return l, j
```

```
QuickSort3 (A, i, f)        (con partition3 in-place o normal)
    if i ≤ f :
        p1, p2 = Partition3 (A, i, f)
        QuickSort3 (A, i, p1-1)
        QuickSort3 (A, p2+1, f)
```

Ejercicio 2.

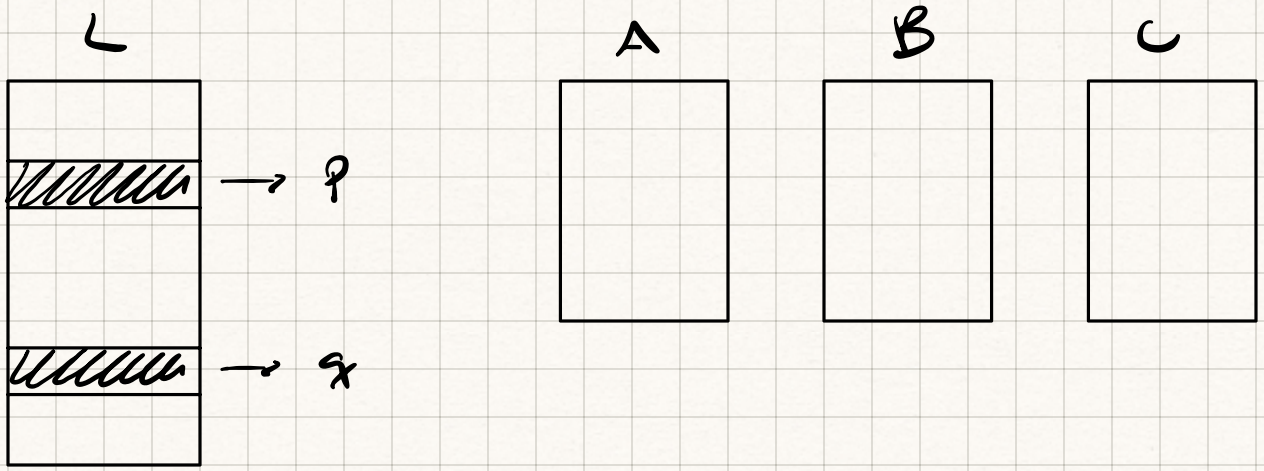
b) El algoritmo quicker-sort llama a la subrutina pq-partition, que utiliza dos pivotes p y q $(p < q)$ para particionar el arreglo en 5 partes: los elementos menores que p, el pivote p, los elementos entre p y q, el pivote q, y los elementos mayores que q. Escribe el pseudocódigo de pq-partition. ¿Es quicker-sort más eficiente que quick-sort? Justifica.

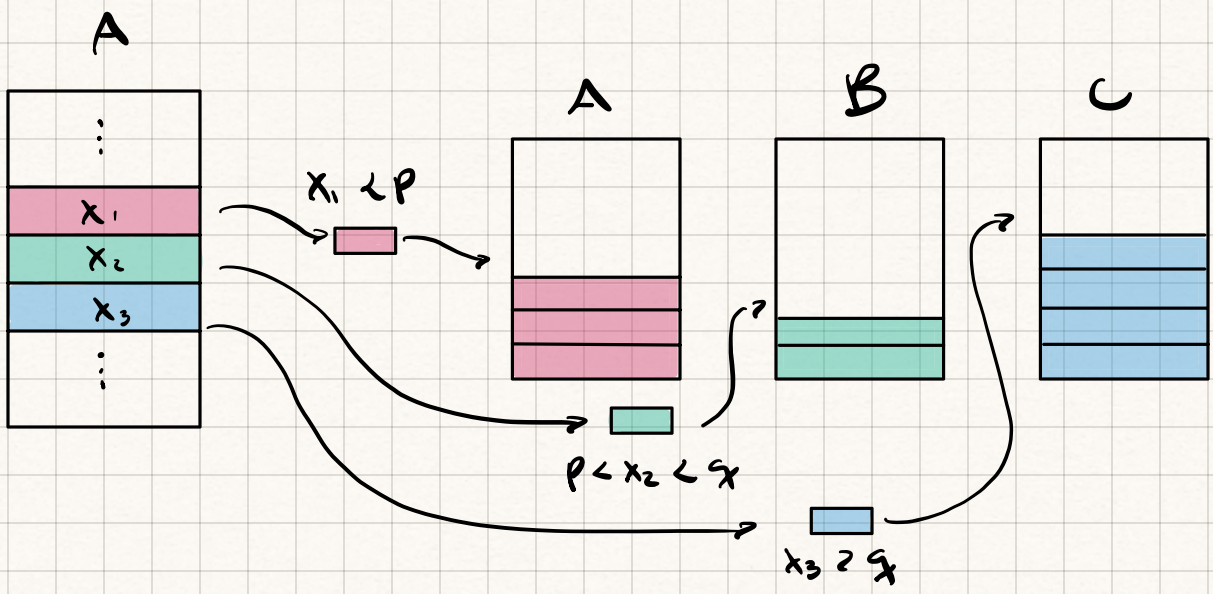
Solución Pregunta 1b)

Ya que la definición dice que $p < q$, entonces es posible asumir que no hay datos repetidos. De todos modos, esto no afecta el análisis.

Lo más simple es implementar partition con listas ligadas como vimos en clases:

```
1:  procedure PQ-PARTITION(lista ligada L)
2:      p, q ← dos nodos de L tal que p < q. Quitar estos nodos de L.
3:      A, B, C ← listas vacias   ▷ Los elementos menores a p, entre p y q y mayores a q respectivamente
4:      for nodo x ∈ L do
5:          if x < p then
6:              agregar x al final de A
7:          else if x < q then
8:              agregar x al final de B
9:          else
10:             agregar x al final de C
11:         end if
12:     end for
13:     return A, p, B, q, C
14: end procedure
```





return →



" p y q en posiciones ordenadas ".

p,q QuickSort (A, i, f)

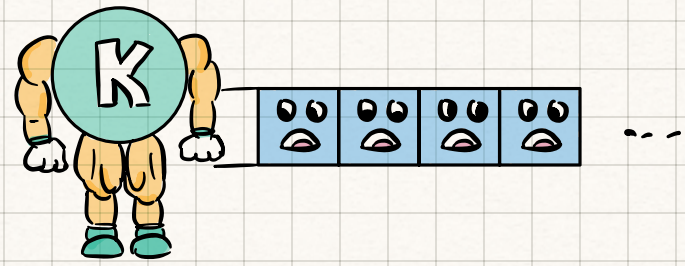
if i ≤ f :

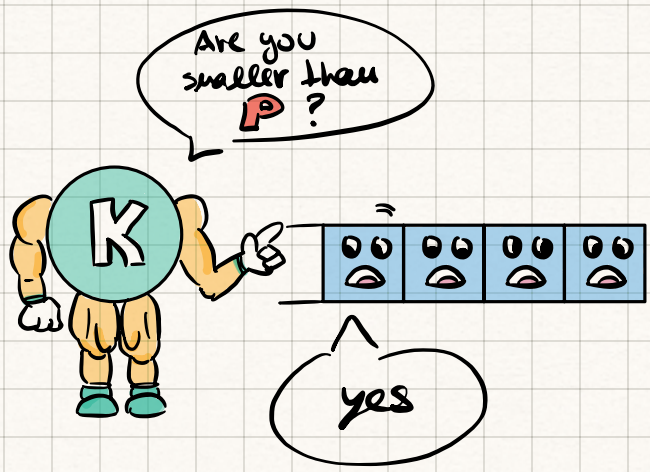
p,q = p-q Partition (A, i, f)

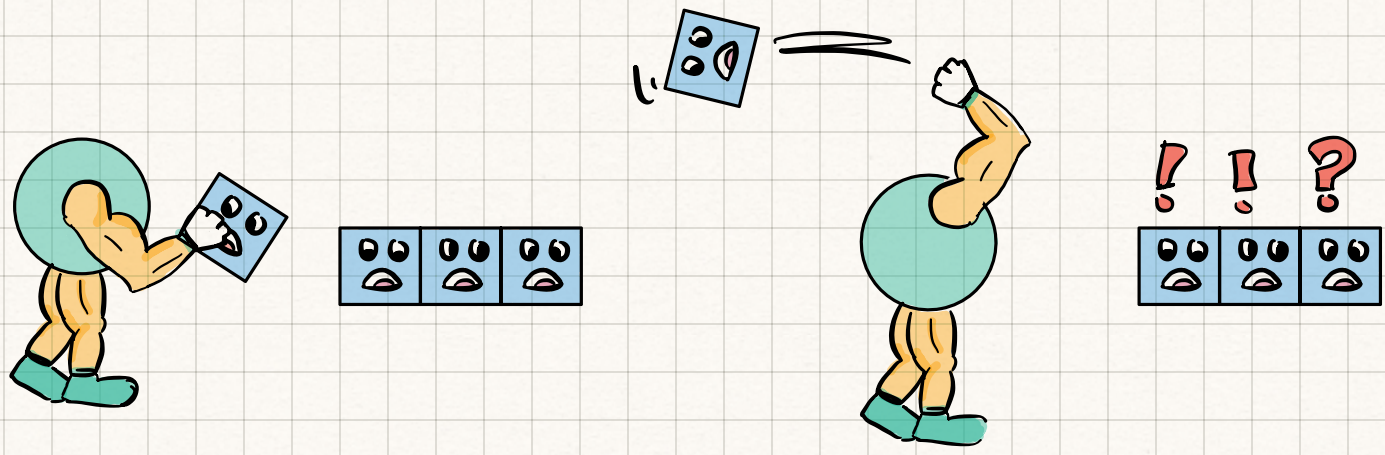
p-q QuickSort (A, i, p-1)

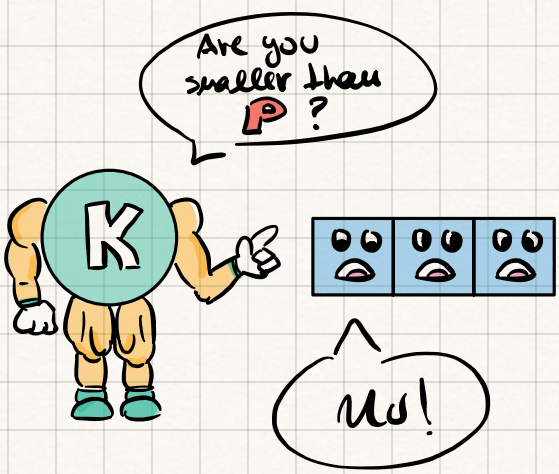
p-q QuickSort (A, p+1, q-1)

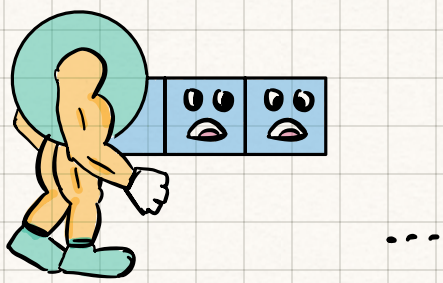
p-q QuickSort (A, q+1, f)

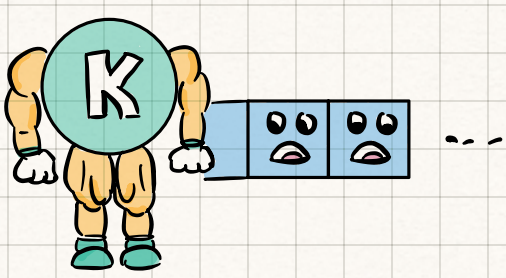

K


Are you smaller than P ?
K
yes


! ! ?


Are you smaller than P ?
K
Nu!




K