Práctico 1.2: Ejercicio 3

...

Algoritmos y Estructuras de Datos II - 2020
FaMAF - UNC

Práctico 1.2: Ejercicio 3: El k-ésimo más chico

3. Escribí un algoritmo que dado un arreglo a : **array**$[1..n]$ **of int** y un número natural $k \leq n$ devuelve el elemento de a que quedaría en la celda $a[k]$ si a estuviera ordenado. Está permitido realizar intercambios en a, pero no ordenarlo totalmente. La idea es explotar el hecho de que el procedimiento partition del quick_sort deja al pivot en su lugar correcto.

- Encabezado:

```
proc k-esimo(in/out a: array[1..n] of T, in k: nat, out elem: T)
```

- Entrada: El arreglo **a**, el número **k**.
- Salida: El elemento **elem**, el arreglo **a** (o sea, se puede modificar el arreglo).

Ejemplo: El k-ésimo más chico

3. Escribí un algoritmo que dado un arreglo a : **array**$[1..n]$ **of int** y un número natural $k \leq n$ devuelve el elemento de a que quedaría en la celda $a[k]$ si a estuviera ordenado. Está permitido realizar intercambios en a, pero no ordenarlo totalmente. La idea es explotar el hecho de que el procedimiento partition del quick_sort deja al pivot en su lugar correcto.

- Entrada: `a = [8, -2, 9, 0, 13], k = 4`.
- Salida: `elem = 9, a = [0, -2, 8, 9, 13]`.
- Observación: se hicieron algunos swap pero no se ordenó el arreglo por completo.

Ayuda: Usar partition

- Encabezado:

```
proc partition (in/out a: array[1..n] of T, in lft, rgt: nat, out ppiv: nat)
```

- Entrada: Un arreglo y un intervalo (lft, rgt).

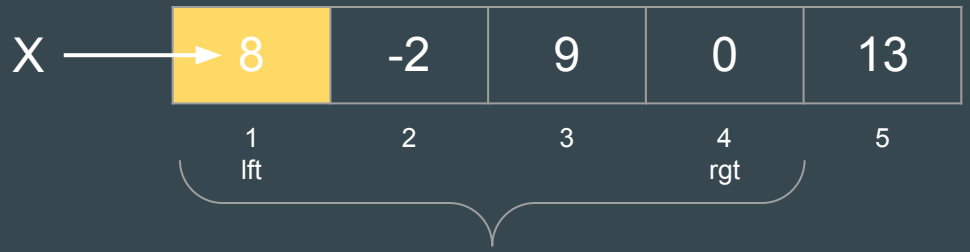
- Resultado:
    - Agarra el primer elemento X del segmento (llamado pivot).
    - Dentro del segmento, reubica todos los elementos menores a X a la izquierda de X.
    - Y reubica todos los elementos mayores a X a la derecha de X.
    - Devuelve en ppiv la nueva posición de X dentro del arreglo (el pivot).

Partition: Ejemplo

- `a = [8, -2, 9, 0, 13], lft = 1, rgt = 4.`

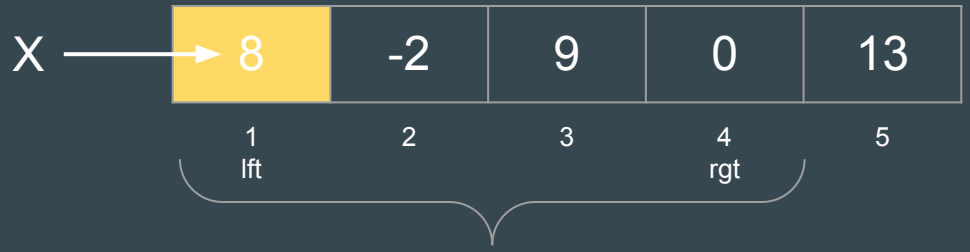
Partition: Ejemplo

- `a = [8, -2, 9, 0, 13], lft = 1, rgt = 4.`



Partition: Ejemplo

- `a = [8, -2, 9, 0, 13], lft = 1, rgt = 4.`



- Resultado: `a = [0, -2, 8, 9, 13], ppiv = 3.`

Partition: Ejemplo

- Observación: Partition deja el elemento pivot “en el lugar correcto”.
  - O sea, en el lugar que le tocaría dentro del segmento, si éste estuviera ordenado.

- En el ejemplo, el 8 ya ocupa el lugar que le toca en el segmento ordenado:

¡Usemos Partition!

- Recordemos: queremos encontrar el k-ésimo si el arreglo estuviera ordenado:

```
proc k-esimo(in/out a: array[1..n] of T, in k: nat, out elem: T)
```

- Llamemos a partition sobre todo el arreglo:

```
partition(a, 1, n, ppiv)
```

- Si `ppiv = k` ya está, porque `ppiv` está en la posición correcta si el arreglo estuviera ordenado.

Ejemplo:

- `k-esimo(a, k, elem)` con `a = [8, 0, 9, -2, 13]`, `k = 3`.
- Primero llamamos: `partition(a, 1, n, ppiv)`

- Resultado: `ppiv = 3 = k`. ¡Ya está! `elem = a[ppiv] = 8`.

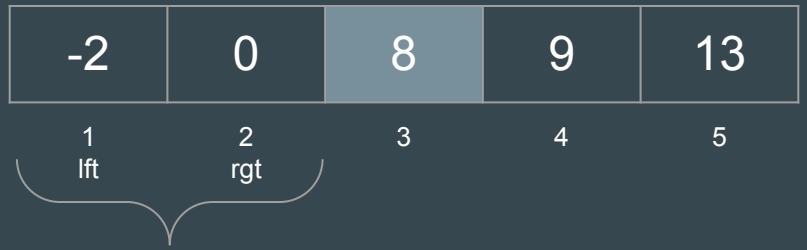
¿Qué pasa si ppiv != k?

- `k-esimo(a, k, elem)` con `a = [8, -2, 9, 0, 13]`, `k = 2`.
- Primero llamamos: `partition(a, 1, n, ppiv)`

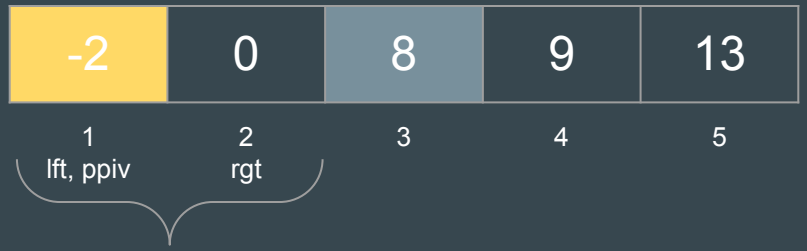
- Resultado: `ppiv = 3`, pero `k = 2`. ¡El elemento buscado está a la izquierda!

¿Qué pasa si ppiv != k?

- Llamamos a partition a la izquierda: `partition(a, 1, 2, ppiv)`

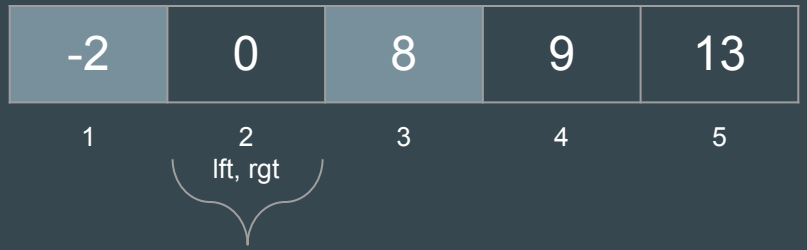


- Resultado: `ppiv = 1`, pero `k = 2`. Hay que buscar a la derecha de este ppiv.

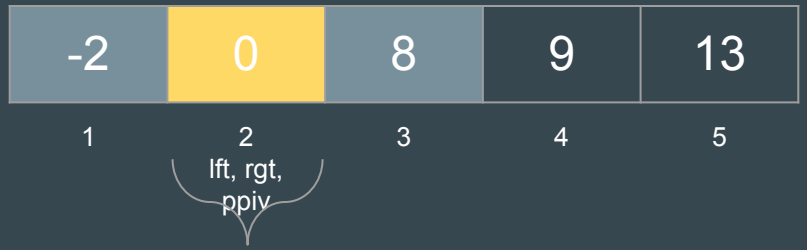


¿Qué pasa si ppiv != k?

- Llamamos de nuevo a partition: `partition(a, 2, 2, ppiv)`



- Resultado: Ahora `ppiv = 2 = k`. ¡Ya está! `elem = a[ppiv] = 0`.



Resumen: k-esimo(a, k, elem)

- Primero llamar a partition sobre el arreglo entero:

```
lft := 1
rgt := n
partition(a, lft, rgt, ppiv)
```

- Luego, mientras suceda que `ppiv != k`, repetir lo siguiente:
  - Si k > ppiv, hay que buscar hacia la derecha de este ppiv:

    ```
    lft := ppiv + 1
    ```

  - Si k < ppiv, hay que buscar hacia la izquierda de este ppiv:

    ```
    rgt := ppiv - 1
    ```

  - Llamar a partition:

    ```
    partition(a, lft, rgt, ppiv)
    ```

Resultado Final

```
proc k-esimo(in/out a: array[1..n] of T, in k: nat, out elem: T)
  var ppiv, lft, rgt: nat
  lft := 1
  rgt := n
  partition(a, lft, rgt, ppiv)
  do ppiv != k ->
    if ppiv < k ->
      lft := ppiv + 1
    else
      rgt := ppiv - 1
    fi
    partition(a, lft, rgt, ppiv)
  od
  elem := a[k]
end proc
```

¡Gracias!