

Práctico 2021-04-12

Práctico 1.2 - Ejercicio 6

```
proc partition (in/out a: array[1..n] of T, in lft, rgt: nat, out ppiv: nat)
  var i,j: nat
  ppiv:= lft
  i:= lft+1
  j:= rgt
  do i ≤ j →
    if a[i] ≤ a[ppiv] → i:= i+1
    a[j] ≥ a[ppiv] → j:= j-1
    a[i] > a[ppiv] ∧ a[j] < a[ppiv] →
      swap(a,i,j)
      i:= i+1
      j:= j-1
    fi
  od
  swap(a,ppiv,j) {- dejando el pivot en una posición más central -}
  ppiv:= j      {- señalando la nueva posición del pivot -}
end proc
```

El procedimiento `partition` que se dio en clase separa un fragmento de arreglo principalmente en dos segmentos: menores o iguales al pivot por un lado y mayores o iguales al pivot por el otro. Modificá ese algoritmo para que separe en tres segmentos: los menores al pivot, los iguales al pivot y los mayores al pivot. En vez de devolver solamente la variable pivot, deberá devolver `ppiv_izq` y `ppiv_der` que informan al algoritmo quick sort rec las posiciones inicial y final del segmento de repeticiones del pivot. Modificá el algoritmo quick sort rec para adecuarlo al nuevo procedimiento `partition`.

Encabezado del procedimiento:

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

Ejemplo de entrada: `a = [3, 5, 1, 3, 3, 7, 10, -1]`. `lft=1`, `rgt=8`.

Ejemplo de salida:

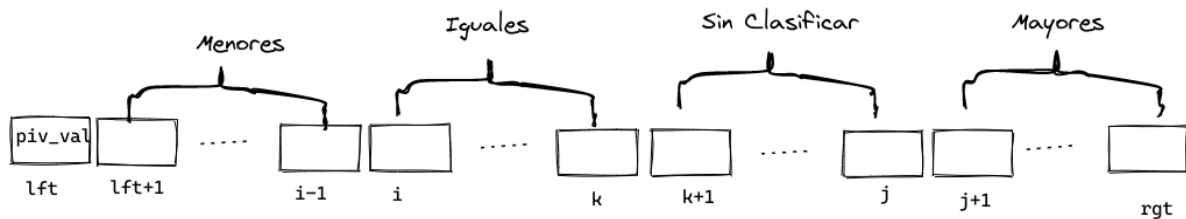
```
a = [1, -1, 3, 3, 3, 7, 10, 5]
    1 2 3 4 5 6 7 8
ppiv_lft = 3
ppiv_rgt = 5
```

```
proc partition (in/out a: array[1..n] of T, in lft, rgt: nat, out ppiv_lft,
ppiv_rgt: nat)
  var piv_val: T
  var i,j,k: nat
  piv_val := a[lft]
  i := lft+1      {- acá también anda i := lft y nos gusta más -}
  j := rgt
```

```

k := lft
{- al inicio, el intervalo sin clasificar es todo desde lft+1 hasta rgt -}
do k+1 <= j → {- termino cuando no queda nada sin clasificar -}
  {- INVARIANTE:   lft+1 <= i <= k <= j <= rgt
    - entre lft+1 e i-1 : menores al pivot   (un para todo)
    - entre j+1 y rgt: mayores al pivot      (un para todo)
    - entre i y k: iguales al pivot          (un para todo)
    - entre k+1 y j: sin clasificar          (todavía no procesada)
  -}

```



```

{- quiero achicar la parte sin clasificar (va de k+1 a j), puedo mirar el
elemento a[k+1]. -}
if a[k+1] = ppiv_val →
  {- no hace falta mover el elemento de lugar -}
  k := k + 1
else if a[j] > ppiv_val →
  {- no hace falta mover el elemento de lugar -}
  j := j - 1
else if a[k+1] > ppiv_val →
  swap(a, k+1, j) {- lo pongo en la posición j -}
  j := j - 1
else if a[k+1] < ppiv_val → {- o directamente else -}
  {- hay que mover el elemento a[k+1] al sector de los mayores -}
  {- [ ... menores ... [-7] iguales ... [3] sin clasificar ... mayores ... ]
     lft+1 i-1 i k k+1 j j+1 rgt
  {- acá tenemos que mandar el a[k+1] al intervalo de los menores -}
  {- tendría que ir a la posición i -}
  swap(a, i, k+1) {- QUÉ HACER PARA PRESERVAR INVARIANTE? -}
  i := i+1 {- así queda el -7 dentro de los menores -}
  k := k+1 {- así vuelvo a meter el 3 dentro de los iguales -}
  {- [ ... menores [-7] ... iguales [3] ... sin clasificar ... mayores ... ]
     lft+1 i-1 i k k+1 j j+1 rgt
  fi fi fi fi
od
{- terminamos, la parte sin clasificar quedó vacía. qué falta? -}
{- primero: acomodar el pivot que está en la posición lft -}
{- hay que meter a[lft] dentro del intervalo entre i y k- }
swap(a, lft, i-1)
i := i - 1
{- segundo: devolver valores ppiv_lft y ppiv_rgt -}
ppiv_lft := i
ppiv_rgt := k
end proc

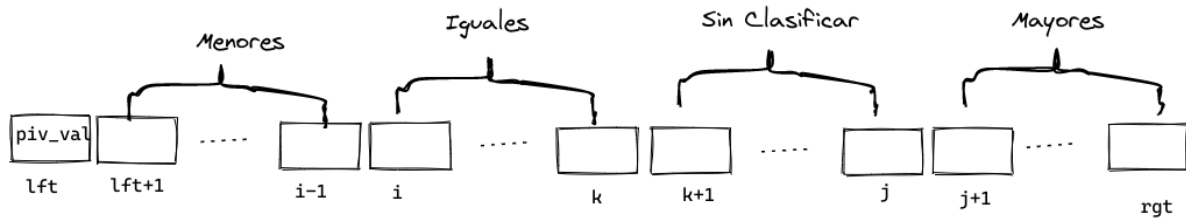
```

En limpio y nuestra versión preferida:

```

proc partition (in/out a: array[1..n] of T, in lft, rgt: nat, out ppiv_lft,
ppiv_rgt: nat)
  var piv_val: T
  var i,j,k: nat
  piv_val := a[lft]
  i := lft
  j := rgt
  k := lft
  do k+1 <= j →

```



```

    if a[k+1] = ppiv_val →
      k := k + 1
    else if a[k+1] > ppiv_val →
      swap(a, k+1, j)
      j := j - 1
    else
      swap(a, i, k+1)
      i := i+1
      k := k+1
    fi fi
  od
  ppiv_lft := i
  ppiv_rgt := k
end proc

```