Algoritmos y Estructuras de Datos II

Ordenación rápida

16 de marzo de 2015

Contenidos

Ayuda de Juan

- La idea original de Juan fue
  - Que cada uno ordenara la mitad.
  - Que luego se intercalen las dos mitades ya ordenadas.
  - Este proceso, iterado, dio lugar a la ordenación por intercalación.
- otra idea parecida puede ser:
  - Separar en dos mitades: por un lado los que irían al principio y por el otro los que irían al final.
  - Que cada uno ordene su mitad.
  - Que finalmente se junten las dos mitades ordenadas.
  - Esta idea da lugar al algoritmo conocido por quicksort u ordenación rápida.

Ordenación rápida en Haskell

```
qsort :: [T] → [T]
qsort [] = []
qsort [a] = [a]
qsort (a:as) = qsort xs ++ [a] ++ qsort ys
                    where (xs,ys) = (filter (<=a) as, filter (>a) as)
```

Ordenación rápida en pseudocódigo

```
{Pre: 0 ≤ der ≤ n ∧ 1 ≤ izq ≤ n+1 ∧ izq-1 ≤ der ∧ a = A}
proc quick_sort_rec (in/out a: array[1..n] of T, in izq,der: nat)
        var piv: nat
        if der > izq → pivot(a,izq,der,piv)
                        izq ≤ piv ≤ der
                        elementos en a[izq,piv-1] < ó = que a[piv]
                        elementos en a[piv+1,der] > a[piv]}
                       quick_sort_rec(a,izq,piv-1)
                       quick_sort_rec(a,piv+1,der)
        fi
end proc
{Post: a permutación de A ∧ a[izq,der] permutación ordenada de A[iz
```

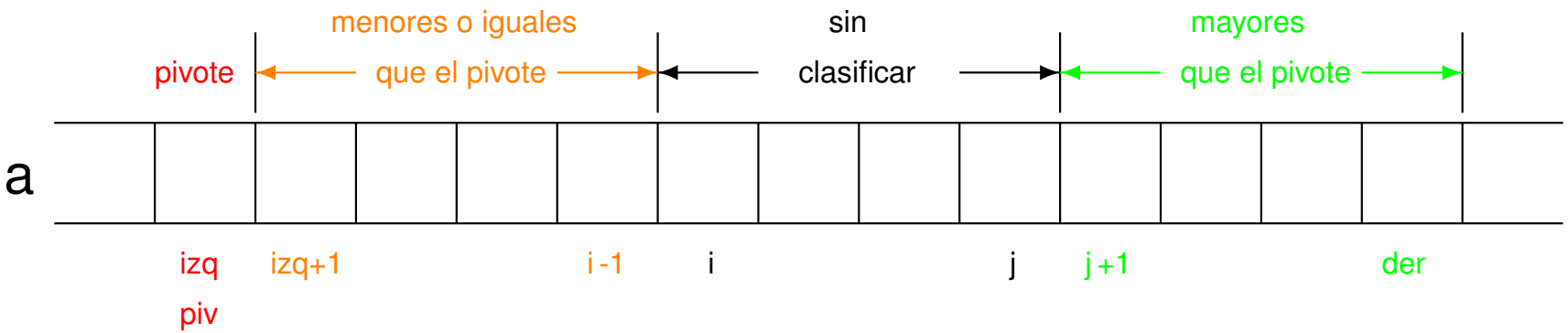
Algoritmo principal

```
proc quick_sort (in/out a: array[1..n] of T)
        quick_sort_rec(a,1,n)
end proc
```

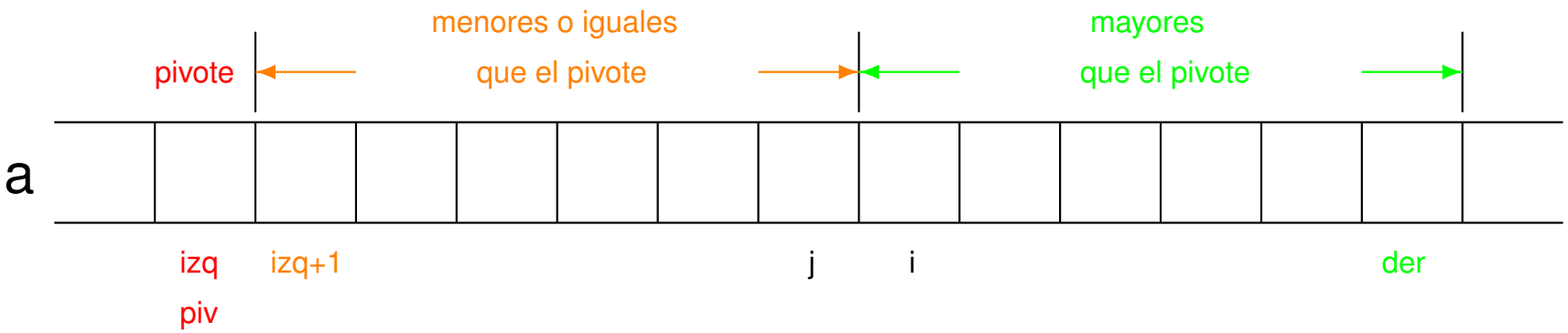
Procedimiento pivot

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

Invariante del procedimiento pivot



al finalizar queda así:



y se hace un swap entre las posiciones izq y j.

Pre, post e invariante

- $\{$Pre: $1 \leq$ izq $<$ der $\leq$ n $\wedge$ a = A$\}$
- $\{$Post: a[1,izq) = A[1,izq) $\wedge$ a(der,n] = A(der,n]
  $\wedge$ a[izq,der] permutación de A[izq,der]
  $\wedge$ izq $\leq$ piv $\leq$ der
  $\wedge$ los elementos de a[izq,piv] son $\leq$ que a[piv]
  $\wedge$ los elementos de a(piv,der] son $>$ que a[piv]$\}$
- $\{$Inv: izq = piv $<$ i $\leq$ j+1 $\leq$ der+1
  $\wedge$ todos los elementos en a[izq,i) son $\leq$ que a[piv]
  $\wedge$ todos los elementos en a(j,der] son $>$ que a[piv]$\}$

Ejemplo

3	9	1	2	7	3	5
3	9	1	2	7	3	5
3	3	1	2	7	9	5
2	3	1	3	7	9	5
2	3	1	3	7	9	5
2	1	3	3	7	9	5
1	2	3	3	7	9	5
1	2	3	3	7	9	5
1	2	3	3	7	9	5
1	2	3	3	7	9	5

1	2	3	3	7	5	9
1	2	3	3	5	7	9
1	2	3	3	5	7	9
1	2	3	3	5	7	9

Ejemplo de pivot

3	9	1	2	7	3	5
3	9	1	2	7	3	5
3	3	1	2	7	9	5
3	3	1	2	7	9	5
3	3	1	2	7	9	5
3	3	1	2	7	9	5
2	3	1	3	7	9	5

Análisis del algoritmo

Queda para la clase que viene.