Implementación del Quick Sort

De manera similar a la implementación del mergesort, definimos un procedimiento **recursivo**, que tomará el arreglo de elementos y dos índices correspondientes al pedazo de arreglo que se ordenará. El algoritmo principal llama a este procedimiento con los índices 1 y n, correspondiendo con la ordenación del arreglo completo.

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

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

Este procedimiento recursivo tiene su caso más simple cuando `lft` y `rgt` son iguales, lo que significa que estoy ordenando un arreglo de un solo elemento. En el caso interesante, llamaremos al procedimiento `partition` que será el encargado de acomodar los elementos del pedazo de arreglo utilizando el elemento de más a la izquierda como pivot.

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

El procedimiento `partition` modifica el arreglo desde `lft` hasta `rgt` dejando al comienzo todos los elementos que son menores o iguales al que se encontraba originalmente en la posición `lft`, y al final a todos los que son mayores o iguales. También modifica la variable `ppiv` asignándole el índice correspondiente al lugar donde queda ubicado definitivamente el elemento que se usó como pivot.

Luego lo único que queda por hacer es llamar recursivamente al procedimiento, una vez para los elementos que quedaron acomodados a la izquierda del pivot, y otra vez para los elementos que quedaron acomodados a la derecha.

```
proc quick_sort_rec(in/out a: array[1..n] of T, in lft,rgt: nat)
    var ppiv: nat
    if rgt > lft --> partition(a,lft,rgt,ppiv)
                     quick_sort_rec(a,lft,ppv-1)
                     quick_sort_rec(a,ppiv+1,rgt)
    fi
end proc
```

Queda por ver el procedimiento `partition`. Toma el arreglo, los dos índices que indican qué fragmento estamos ordenando, y una variable de solo escritura, en la cual indicaremos el índice en donde queda el pivot una vez que finalice el procedimiento.

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

Lo que tenemos que hacer en este procedimiento es ir mirando con un índice los elementos que están a la izquierda y con otro los que están a la derecha. El índice `i` indicará el elemento que estoy mirando desde la izquierda, y respectivamente `j` indicará el de la derecha. El elemento tomado como pivot será el que está más a la izquierda.

```
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
    ......
    ......
```

Luego debemos ir viendo si el elemento indicado con el índice `i` y el indicado con `j` están bien ubicados, es decir, si `a[i]` es menor o igual al pivot, y si `a[j]` es mayor o igual. En caso que sea así, "avanzaremos" el índice. Este avance corresponde a sumar uno para `i`, y a restar uno para `j`. En caso que el elemento de la izquierda esté mal ubicado, debemos encontrar un elemento de la derecha que también esté mal ubicado, y los intercambiamos.

```
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)
                  fi
    od
    ............
    ............
```

Esto se repite hasta que los índices i y j se hayan cruzado. En ese momento se termina el ciclo y solo queda ubicar correctamente al elemento pivot, y asignar la variable `ppiv` para que indique la posición final en donde queda el mismo.

```
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)
                  fi
    od
    swap(a,ppiv,j)
    ppiv:= j
end proc
```

Aquí el código completo:

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

proc quick_sort_rec(in/out a: array[1..n] of T, in lft,rgt: nat)
    var ppiv: nat
    if rgt > lft --> partition(a,lft,rgt,ppiv)
                     quick_sort_rec(a,lft,ppv-1)
                     quick_sort_rec(a,ppiv+1,rgt)
    fi
end proc

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)
                  fi
    od
    swap(a,ppiv,j)
    ppiv:= j
end proc
```
