Implementación del Merge Sort

Vamos a ver un poquito en detalle la implementación del Merge Sort en el lenguaje de la materia. La parte de este texto que corresponde con código estará en la fuente `monospace` y encerrada entre líneas. Cuando haya código incompleto, pondremos puntos suspensivos.

`código`

La manera más simple de implementar este algoritmo es utilizando **recursión**. La idea es definir un procedimiento al que le pasamos en qué parte del arreglo queremos hacer lo fundamental del mergesort: dividirlo en dos, ordenar cada mitad y luego intercalar las dos mitades.
Este procedimiento es **`merge_sort_rec`**, que toma el arreglo a y las posiciones inicial y final del pedazo de arreglo que vamos a ordenar. El procedimiento principal llama al recursivo con los índices 1 y n (el arreglo completo).

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

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

El procedimiento **`merge_sort_rec`** toma el arreglo `a`, y los índices `lft` y `rgt`, que corresponden con el comienzo y el final del pedazo de arreglo que queremos ordenar. Recordando la idea del algoritmo, el caso más simple es cuando el pedazo de arreglo tiene **un solo elemento**. En nuestra implementación eso corresponde a que `lft` sea igual a `rgt`. En ese caso el procedimiento no debe hacer nada, ya que el pedazo está trivialmente ordenado.
En caso que no se dé esa situación, debemos:

* Dividir el pedazo de arreglo en dos
* Ordenar cada una de esas mitades utilizando el mismo algoritmo, y
* Intercalar cada mitad ordenada.

Para dividir el pedazo de arreglo, definimos una variable `mid` de tipo `nat` a la que le asignaremos el índice correspondiente a la posición del medio.

```
proc merg_sort_rec(in/out a: array[1..n] of T, in lft,rgt: nat)
    var mid: nat
    if rgt > lft --> mid := (rgt+lft) `div` 2
    .........
    .........
```

Ahora entonces debemos llamar recursivamente al procedmiento dos veces: una para la primera mitad que irá desde la posición `lft` hasta `mid`, y otra para la segunda mitad, que irá desde la posición `mid+1` hasta `rgt`.

```
proc merg_sort_rec(in/out a: array[1..n] of T, in lft,rgt: nat)
    var mid: nat
    if rgt > lft --> mid := (rgt+lft) `div` 2
                     merge_sort_rec(a,lft,mid)
                     merge_sort_rec(a,mid+1,rgt)
    .......
    .......
```

---

y por último, tenemos que **intercalar**. Esta tarea la implementaremos con un procedimiento llamado **`merge`,** que definiremos luego.

```
proc merge_sort_rec(in/out a: array[1..n] of T, in lft,rgt: nat)
    var mid: nat
    if rgt > lft --> mid := (rgt+lft) `div` 2
                     merge_sort_rec(a,lft,mid)
                     merge_sort_rec(a,mid+1,rgt)
                     merge(a,lft,mid,rgt)
    fi
end proc
```

Nos queda la tarea de implementar el procedimiento de intercalación. Este procedimiento recibe el arreglo y tres posiciones: la primera posición de la primera mitad, la última posición de la primera mitad, y la última posición de la segunda mitad. La primera mitad va desde `lft` hasta `mid`, y la segunda desde `mid+1` hasta `rgt`.

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

Recordando la idea del algoritmo, para intercalar necesitamos un arreglo auxiliar, en donde guardaremos los valores de la primera mitad a intercalar.
Definimos entonces una variable de tipo array, dos variables en las que luego almacenaremos índices `j` y `k`, y compiamos la primera mitad del arreglo en el arreglo auxiliar:

```
proc merge(in/out a: array[1..n] of T, in lft,mid,rgt: nat)
    var tmp: array[1..n] of T
    var j,k: nat
    for i:=lft to mid do tmp[i] := a[i] od
    .........
```

Los índices `j` y `k` indicarán respectivamente el elemento de la primera mitad que

estoy analizando para insertar en el pedazo de arreglo que quedará ordenado, y el índice de la segunda mitad que estoy analizando. Inicialmente observo el primero de cada mitad, es decir `lft` y `mid+1`.

```
proc merge(in/out a: array[1..n] of T, in lft,mid,rgt: nat)
    var tmp: array[1..n] of T
    var j,k: nat
    for i:=lft to mid do tmp[i] := a[i] od
    j := lft
    k := mid+1
    .........
```

Ahora debemos rellenar el pedazo completo de arreglo que contendrá las dos mitades intercaladas ordenadamente. Lo recorreremos con un **for** desde `lft` hasta `rgt`. Y observaremos en cada paso si el elemento que estoy observando de la primera mitad es menor o igual que el de la segunda mitad, de acuerdo a esa comparación sabremos qué elemento va a ubicarse en el arreglo ordenado.
Recordemos que los elementos de la primera mitad los observaremos desde el arreglo auxiliar en donde copiamos todos sus elementos.

```
proc merge(in/out a: array[1..n] of T, in lft,mid,rgt: nat)
    var tmp: array[1..n] of T
    var j,k: nat
    for i:=lft to mid do tmp[i] := a[i] od
    j := lft
    k := mid+1
    for i := lft to rgt do
        if j <= mid ∧ (k > rgt ∨ tmp[j] <= a[k])
           then a[i] := tmp[j]
                j:= j+1
           else a[i] := a[k]
                k := k+1
        fi
    od
end proc
```

En la guarda del if tenemos también que considerar el caso en que ya haya agotado todos los elementos de la segunda mitad, lo que sucederá cuando `k > rgt`, y entonces en ese caso también completaremos con los elementos de la primera mitad (es decir los que están en el arreglo auxiliar).

Aquí el código completo:

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

```
proc merge_sort_rec(in/out a: array[1..n] of T, in lft,rgt: nat)
    var mid: nat
    if rgt > lft --> mid := (rgt+lft) `div` 2
                     merge_sort_rec(a,lft,mid)
                     merge_sort_rec(a,mid+1,rgt)
                     merge(a,lft,mid,rgt)
    fi
end proc

proc merge(in/out a: array[1..n] of T, in lft,mid,rgt: nat)
    var tmp: array[1..n] of T
    var j,k: nat
    for i:=lft to mid do tmp[i] := a[i] od
    j := lft
    k := mid+1
    for i := lft to rgt do
        if j <= mid ∧ (k > rgt ∨ tmp[j] <= a[k])
           then a[i] := tmp[j]
                j:= j+1
           else a[i] := a[k]
                k := k+1
        fi
    od
end proc

---------------------------
```