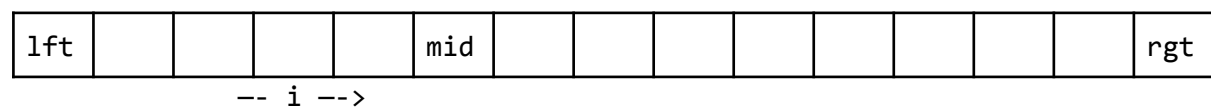
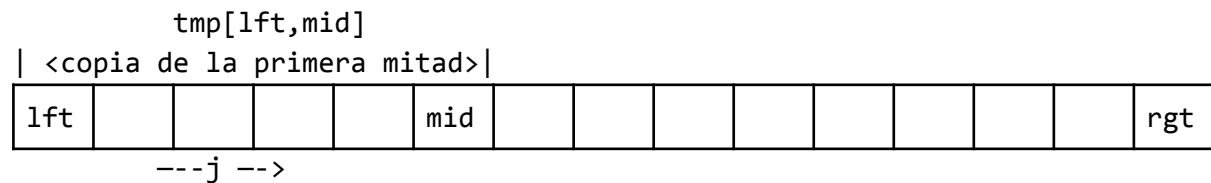
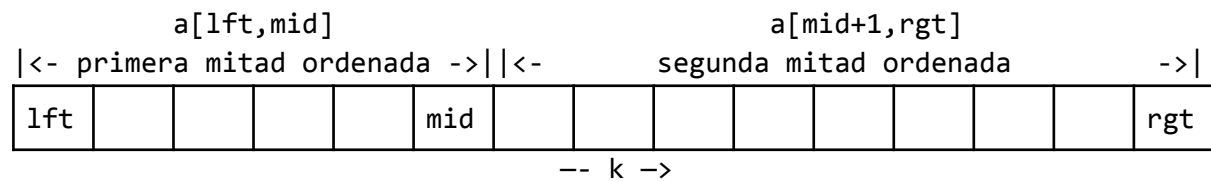


# Ordenación por intercalación

## Idea



Pre, post e invariante:

- {Pre:  $n \geq rgt \geq lft > 0 \wedge a = A$ }
- {Invariante:  
   $a[lft, mid]$  permutación ordenada de  $A[lft, mid]$   
   $\wedge a[mid+1, rgt]$  permutación ordenada de  $A[mid+1, rgt]$   
   $\wedge a[lft, rgt]$  permutación ordenada de  $A[lft, rgt]$ }
- {Post:  $a$  permutación de  $A$   
   $\wedge a[lft, rgt]$  permutación ordenada de  $A[lft, rgt]$ }

## Código

```
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 lft < rgt then
    mid := (lft + rgt) div 2 {- división entera -}
    merge_sort_rec(a, lft, mid)
    merge_sort_rec(a, mid+1, rgt)
    merge(a, lft, 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
  j, k: nat
  for i := lft to mid do
    tmp[i] := a[i]      {- copio la primera mitad en un temporal -}
  od

  j := lft
  k := mid + 1
  for i := lft to rgt do
```

```

    if j ≤ mid ∧ (rgt < k ∨ tmp[j] ≤ a[k]) then
        a[i] := tmp[j]
        j := j + 1
    else
        a[i] := a[k]
        k := k + 1
    fi
od
end proc

```

## Número de comparaciones

- El algoritmo merge\_sort(a) llama a merge\_sort\_rec(a,1,n).
- Por lo tanto, para contar las comparaciones de merge\_sort(a), debemos contar las de merge\_sort\_rec(a,1,n).
- Pero merge\_sort\_rec(a,1,n) llama a merge\_sort\_rec(a,1,[(n+1)/2]) y a merge\_sort\_rec(a,[(n+1)/2]+1, n).
- Por lo tanto, hay que contar las comparaciones de estas llamadas...

### Solución

- Sea  $t(m)$  = número de comparaciones que realiza merge\_sort\_rec(a,lft,rgt) cuando desde lft hasta rgt hay m celdas.
- O sea, cuando  $m = rgt + 1 - lft$ .
- Si  $m = 0$ ,  $lft = rgt + 1$ , la condición del if es falsa,  $t(m) = 0$ .
- Si  $m = 1$ ,  $lft = rgt$ , la condición del if es falsa también,  $t(m) = 0$ .
- Si  $m > 1$ ,  $lft > rgt$  y la condición del if es verdadera.
  - $t(m)$  en este caso, es el número de comparaciones de las dos llamadas recursivas, más el número de comparaciones que hace la intercalación.
  - $t(m) \leq t(\lceil m/2 \rceil) + t(\lfloor m/2 \rfloor) + m$

### Solución (potencias de 2)

- Sea  $m = 2^k$ , con  $k > 1$ 
  - $t(m) = t(2^k)$ 

$$\leq t(\lceil 2^k/2 \rceil) + t(\lfloor 2^k/2 \rfloor) + 2^k$$

$$= t(2^{k-1}) + t(2^{k-1}) + 2^k$$

$$= 2 * t(2^{k-1}) + 2^k$$
  - $t(2^k)/2^k \leq (2 * t(2^{k-1}) + 2^k)/2^k$ 

$$= (2 * t(2^{k-1}))/2^k + 2^k/2^k$$

$$= t(2^{k-1})/2^{k-1} + 1$$

$$\leq t(2^{k-1})/2^{k-1} + 1$$

$$\leq t(2^{k-2})/2^{k-2} + 1 + 1$$

$$= t(2^{k-2})/2^{k-2} + 2$$

$$\leq t(2^{k-3})/2^{k-3} + 3$$

$$\dots$$

$$\leq t(2^0)/2^0 + k$$

$$= t(1) + k$$

$$= k$$
- Entonces  $t(2^k) \leq 2^k * k$ .

- Entonces  $t(m) \leq m * \log_2 m$  para  $m$  potencia de 2.

## Conclusión

La ordenación por intercalación es del orden de  $n * \log_2(n)$ .

# Ordenación rápida

## Código

```

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 then
        partition(a,lft,rgt,ppiv)
        quick_sort_rec(a,lft,ppiv-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
    while i ≤ j do
        if a[i] ≤ a[ppiv] then
            i:= i+1
        else if a[j] ≥ a[ppiv] then
            j:= j-1
        else if a[i] > a[ppiv] ∧ a[j] < a[ppiv] then
            swap(a,i,j)
            i:= i+1
            j:= j-1
        fi
    od
    ppiv:= i
end proc

```

## Invariante del procedimiento partition

pivot | <- ≤ que el pivot -> || <- sin clasificar -> || <- ≥ que el pivot -> |  

lft	lft+1				i-1	i			j	j+1				rgt
-----	-------	--	--	--	-----	---	--	--	---	-----	--	--	--	-----

al final queda así:

pivot	<-	≤	que el pivot				->		<-	≥ que el pivot				->	
lft	lft+1						i	j							rgt

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

Pre, post e invariante:

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

## Análisis de la ordenación rápida

- La estructura del algoritmo es muy similar a la de la ordenación por intercalación:
  - ambos tienen un procedimiento principal que llama al recursivo con idénticos parámetros,
  - en ambos el procedimiento recursivo es `if rgt > lft then`,
  - en ambos después del `then` hay dos llamadas recursivas
- pero **difieren** en que
  - en el primer caso están primero las llamadas y luego intercalar (que es del orden de  $n$ )
  - en el otro, primero se llama a `partition` (que se verá que es orden de  $n$ ) y luego las llamadas recursivas
  - en el primero el fragmento de arreglo se parte al medio, en el segundo puede ocurrir particiones menos equilibradas
- es interesante observar que los procedimientos intercalar y `partition` son del orden de  $n$ .

## El procedimiento `partition` es del orden de $n$

- Sea  $n$  el número de celdas en la llamada a `partition` (es decir,  $\text{rgt}+1-\text{lft}$ ),
- el ciclo **do** se repite a lo sumo  $n - 1$  veces, ya que en cada caso la brecha entre  $i$  y  $j$  se acorta en uno o dos
- en cada ejecución del ciclo se realiza un número constante de comparaciones,
- por lo tanto su orden es  $n$ .

## Orden de la ordenación rápida

- Se parece a la ordenación por intercalación incluso después del **then**:

- ambos realizan dos llamadas recursivas y una operación, diferente, pero en ambos casos del orden de  $n$
- Por ello, esencialmente el mismo análisis se aplica,
- siempre y cuando el procedimiento `partition` parta el arreglo al medio.

### Conclusión

En ese caso la ordenación rápida es entonces del orden de  $n * \log_2(n)$ .

### Casos

- **caso medio:** el algoritmo en la práctica es del orden de  $n * \log_2(n)$
- **peor caso:** cuando el arreglo ya está ordenado, o se encuentra en el orden inverso, es del orden de  $n^2$
- **mejor caso:** es del orden de  $n * \log_2(n)$ , cuando el procedimiento parte exactamente al medio.