

Algoritmos y Estructuras de Datos II

Ordenación avanzada

Contenidos

- 1 Ayudando al profe
- 2 Ordenación por intercalación
 - La idea
 - El algoritmo
 - Ejemplo
- 3 Ordenación rápida
 - El algoritmo
 - Ejemplo

Ayudando al profe

El profe de esta materia tarda media hora en ordenar alfabéticamente 100 exámenes. ¿Cuánto tardará en ordenar 200 exámenes?

Respuesta hasta ahora: 2 horas.

Ayuda Mauro y Santiago se ofrecen a colaborar para ordenar los 200 exámenes.

¿Cómo puede aprovechar el profe la colaboración?

Idea

Mauro Ordena 100 exámenes.

Santiago Ordena otros 100 exámenes.

Profe Al finalizar, los intercala.

¿Cuánto tarda todo esto?

Mauro + Santiago Media hora.

Profe ¿Cuánto tiempo lleva la intercalación?

Intercalando exámenes

¿Cómo intercalar dos secuencias con 100 exámenes ordenados?

- Comparar el primer examen de una, con el primer examen de la otra. Uno de esos tiene que ser el menor. Sacarlo de esa secuencia y colocarlo como primer examen.
1 comparación \rightarrow 1 examen en su lugar.
- Comparar el primer examen de una, con el primer examen de la otra. Uno de esos tiene que ser el segundo menor. Sacarlo de esa secuencia y colocarlo como segundo examen.
2 comparaciones \rightarrow 2 exámenes en su lugar.
- Etcétera.
- n comparaciones \rightarrow n exámenes en su lugar.

Terminando de intercalar exámenes

- Puede pasar que las dos secuencias ordenadas se van vaciando en forma pareja y la intercalación termina cuando se han colocado en su lugar los primeros 99 exámenes de cada secuencia, es decir, colocado en su lugar 198 con 198 comparaciones. Una última comparación sirve para determinar cuál de los dos exámenes que quedan es el penúltimo, y cuál es el último. 199 comparaciones \rightarrow 200 exámenes en su lugar.
- La otra posibilidad es que una secuencia ordenada se vacía cuando la otra todavía tiene, por ejemplo, 20 exámenes. Se han colocado 180 exámenes con 180 comparaciones. Los restantes 20 exámenes pueden colocarse en su lugar sin ninguna comparación adicional.

¿Cuánto tiempo llevó intercalar 200 exámenes?

- Peor caso: 199 comparaciones.
- Mejor caso: 100 comparaciones.

Entonces ¿cuánto lleva ordenar los 200 exámenes?

ordenar 100 exámenes	\longleftrightarrow	10.000 comparaciones
ordenar 100 exámenes	\longleftrightarrow	10.000 comparaciones
intercalar 200 exámenes	\longleftrightarrow	200 comparaciones

ordenar 100 exámenes	\longleftrightarrow	1/2 hora	} simultáneo
ordenar 100 exámenes	\longleftrightarrow	1/2 hora	
intercalar 200 exámenes	\longleftrightarrow	$\frac{1}{50}$ 1/2 hora	

$$\frac{1}{50} \text{ 1/2 hora} = 36 \text{ segundos.} \quad (1/2 \text{ hora} = 1800 \text{ segundos})$$

Ordenar 200 exámenes, con esta nueva idea

Tarea A Ordenar 100 exámenes como antes, 10.000 de comparaciones, 1/2 hora.

Tarea B Ordenar 100 exámenes como antes, 10.000 de comparaciones, 1/2 hora.

Tarea C Intercalar 200 exámenes, 200 comparaciones, 36 segundos.

Total 1 hora y 36 segundos (ignorando que las tareas A y B pueden realizarse en paralelo).

¡Ordenamos 200 exámenes en poco más de 1 hora! (sin la ayuda de Mauro y Santiago)

¿Podemos hacer algo mejor?

Ordenar 200 exámenes, aprovechando más la idea

Tarea A Ordenar 100 exámenes:

Tarea AA Ordenar 50, 2.500 comparaciones, $1/8$ hora.

Tarea AB Ordenar 50, 2.500 comparaciones, $1/8$ hora.

Tarea AC Intercalar 100, 100 comparaciones, 18 segundos.

Total Tarea A 5.100 comparaciones, $1/4$ hora y 18 segundos.

Tarea B Como Tarea A, 5.100 comparaciones, $1/4$ hora y 18 segundos.

Tarea C Intercalar 200, 200 comparaciones, 36 segundos.

Total 10.400 comparaciones, $1/2$ hora y 72 segundos.

Ordenar 200 exámenes, aprovechando más la idea

Tarea A Ordenar 100 exámenes:

Tarea AA Ordenar 50 exámenes:

Tarea AAA Ordenar 25, 625 comparaciones, 1/32 hora.

Tarea AAB Ordenar 25, 625 comparaciones, 1/32 hora.

Tarea AAC Intercalar 50, 50 comparaciones, 9 segundos.

Total Tarea AA 1.300 comparaciones, 1/16 hora y 9 segundos.

Tarea AB 1.300 comparaciones, 1/16 hora y 9 segundos.

Tarea AC Intercalar 100, 100 comparaciones, 18 segundos.

Total Tarea A 2.700 comparaciones, 1/8 hora y 36 segundos.

Tarea B 2.700 comparaciones, 1/8 hora y 36 segundos.

Tarea C Intercalar 200, 200 comparaciones, 36 segundos.

Total 5.600 comparaciones, 1/4 hora y 108 segundos.

Reflexionando sobre lo que acabamos de hacer

ordenar ¹ bloques de	tardanza	segundos
200 exámenes	2 horas	7200
100 exámenes	1 hora y 36 segundos	3636
50 exámenes	1/2 hora y 72 segundos	1872
25 exámenes	1/4 hora y 108 segundos	1008
13 exámenes	1/8 hora y 144 segundos	594
7 exámenes	1/16 hora y 180 segundos	405
4 exámenes	1/32 hora y 216 segundos	328
2 exámenes	1/64 hora y 252 segundos	308
1 examen	1/128 hora y 288 segundos	316

¹ usando ordenación por selección o por inserción

Conclusión

- ¿Por qué no “ordenar” bloques de 1, y luego intercalar reiteradamente?
- ¡Podríamos ordenar 200 exámenes en 5 minutos!
- Pero ordenar bloques de 1 es trivial, ¡cada bloque de 1 está ordenado!
- ¡Entonces esta manera de ordenar **solamente intercala!**
- Esto se llama **ordenación por intercalación** o **merge sort** en inglés.
- No es tan sencillo de escribir en lenguajes imperativos (porque la operación de intercalación requiere espacio auxiliar).

En el lenguaje imperativo

{Pre: $n \geq rgt \geq lft > 0 \wedge a = A$ }

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

var mid: **nat**

if rgt > lft \rightarrow mid:= (rgt+lft) \div 2

 merge_sort_rec(a,lft,mid)

{a[lft,mid] permutación ordenada de A[lft,mid]}

 merge_sort_rec(a,mid+1,rgt)

{a[mid+1,rgt] permutación ordenada de A[mid+1,rgt]}

 merge(a,lft,mid,rgt)

{a[lft,rgt] permutación ordenada de A[lft,rgt]}

fi

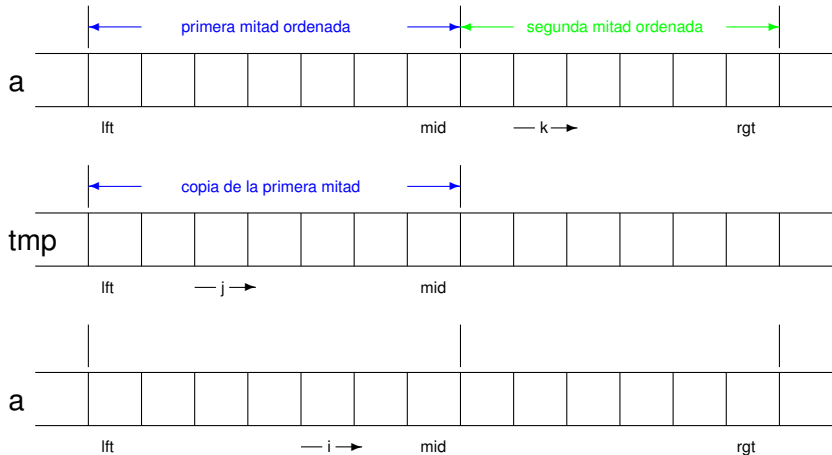
end proc

{Post: a permutación de A \wedge a[lft,rgt] permutación ordenada de A[lft,rgt]}

Algoritmo principal

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

Invariante del procedimiento intercalación



Intercalación en el lenguaje de la materia

```
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] od  
  j:= lft  
  k:= mid+1  
  for i:= lft to rgt do  
    if  $j \leq \text{mid} \wedge (k > \text{rgt} \vee \text{tmp}[j] \leq a[k])$  then a[i]:= tmp[j]  
                                          j:=j+1  
    else a[i]:= a[k]  
          k:=k+1  
    fi  
  od  
end proc
```


Ejemplo

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

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

Ejemplo de intercalación

1	3	3	9
1	3	3	9
	3	3	9
	3	3	9
		3	9
			9
			9
			9

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

Una idea similar

- La idea original fue
 - Que se ordenen las dos mitades separadamente.
 - 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 (por ejemplo: de la A a la L y de la M a la Z).
 - Que se ordenen las dos mitades separadamente.
 - Que finalmente se junten las dos mitades ordenadas.
 - Esta idea da lugar al algoritmo conocido por quicksort u ordenación rápida.

Implementación de Ordenación rápida

```
{Pre:  $0 \leq rgt \leq n \wedge 1 \leq lft \leq n+1 \wedge lft-1 \leq rgt \wedge a = A$ }  
proc quick_sort_rec (in/out a: array[1..n] of T, in lft,rgt: nat)  
  var ppiv: nat  
  if rgt > lft  $\rightarrow$  partition(a,lft,rgt,ppiv)  
    lft  $\leq$  ppiv  $\leq$  rgt  
    elementos en  $a[lft,ppiv-1] \leq a[ppiv]$   
    elementos en  $a[ppiv+1,rgt] \geq a[ppiv]$   
    quick_sort_rec(a,lft,ppiv-1)  
    quick_sort_rec(a,ppiv+1,rgt)  
  fi  
end proc  
{Post: a permut. de A  $\wedge$   $a[lft,rgt]$  permut. ordenada de  $A[lft,rgt]$ }
```

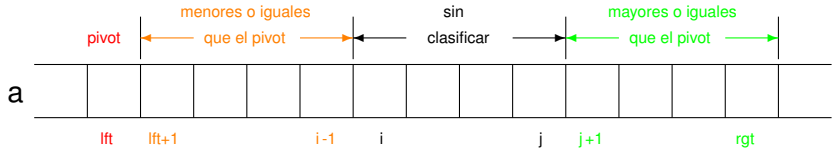
Algoritmo principal

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

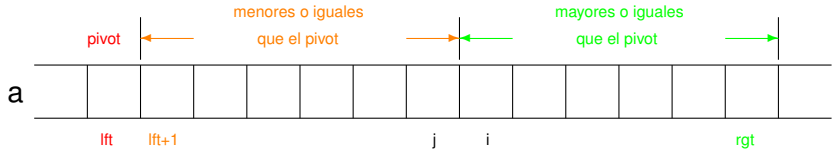
Procedimiento partition

```
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  $i \leq j \rightarrow$  if  $a[i] \leq a[ppiv] \rightarrow i:= i+1$   
     $a[j] \geq a[ppiv] \rightarrow j:= j-1$   
     $a[i] > a[ppiv] \wedge a[j] < a[ppiv] \rightarrow$  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
```

Invariante del procedimiento partition



al finalizar queda así:



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 $a[\text{piv}]$ }

Ejemplo

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

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

Ejemplo de partition

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