Algoritmos y Estructuras de Datos II

Ordenación avanzada

20 y 22 de marzo de 2017

Contenidos

Ayudando al bibliotecario

Un bibliotecario tarda un día en ordenar alfabéticamente una biblioteca con 1000 expedientes. ¿Cuánto tardará en ordenar una con 2000 expedientes?

Respuesta hasta ahora: 4 días.

Ayuda Al bibliotecario le asignan un colaborador para ordenar los 2000 expedientes.

¿Cómo puede aprovechar el bibliotecario la ayuda de su colaborador?

Idea del bibliotecario

Bibliotecario Ordena 1000 expedientes.

Colaborador Ordena otros 1000 expedientes.

Juntos Al día siguiente se intercala.

Ordenando 2000 expedientes

primer día El bibliotecario se pasa todo el día ordenando sus 1000 expedientes.

segundo día El bibliotecario se da cuenta de que su colaborador no hizo su parte . . . entonces se pasa todo el segundo día ordenando los otros 1000.

tercer día El bibliotecario se pone a intercalar los dos grupos de 1000 expedientes ordenados.

¿cuánto tiempo lleva la intercalación?

Intercalando expedientes

¿Cómo intercalar dos bibliotecas con 1000 expedientes ordenados?

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

Programa intercalar en Haskell

```
merge :: [T] → [T] → [T]
merge [] sts2 = sts2
merge sts1 [] = sts1
merge (t1:sts1) (t2:sts2) = if t1 ≤ t2
                            then t1:merge sts1 (t2:sts2)
                            else t2:merge (t1:sts1) sts2
```

Terminando de intercalar expedientes

- Puede pasar que las dos bibliotecas ordenadas se van vaciando en forma pareja y la intercalación termina cuando se han colocado en su lugar los primeros 999 expedientes de cada biblioteca, es decir, colocado en su lugar 1998 con 1998 comparaciones. Una última comparación sirve para determinar cuál de los dos expedientes que quedan es el penúltimo, y cuál es el último.
  1999 comparaciones $\rightarrow$ 2000 expedientes en su lugar.
- La otra posibilidad es que una biblioteca ordenada se vacía cuando la otra todavía tiene, por ejemplo, 200 expedientes. Se han colocado 1800 expedientes con 1800 comparaciones. Los restantes 200 expedientes pueden colocarse en su lugar sin ninguna comparación adicional.

¿Cuánto tiempo llevó intercalar 2000 expedientes?

- Peor caso: 1999 comparaciones.
- Mejor caso: 1000 comparaciones.

Entonces ¿cuánto lleva ordernar los 2000 expedientes?

$$\begin{array}{lcr} \text{ordenar 1000 expedientes} & \longleftrightarrow & 1.000.000 \text{ comparaciones} \\ \text{ordenar 1000 expedientes} & \longleftrightarrow & 1.000.000 \text{ comparaciones} \\ \text{intercalar 2000 expedientes} & \longleftrightarrow & 2.000 \text{ comparaciones} \end{array}$$

$$\begin{array}{lcr} \text{ordenar 1000 expedientes} & \longleftrightarrow & 1 \text{ día} \\ \text{ordenar 1000 expedientes} & \longleftrightarrow & 1 \text{ día} \\ \text{intercalar 2000 expedientes} & \longleftrightarrow & \frac{1}{500} \text{ día} \end{array}$$

$\frac{1}{500}$ día = 1 minuto. (jornada laboral de 8 horas = 480 minutos)

Ordenar 2000 expedientes, con esta nueva idea

Tarea A	Ordenar 1000 expedientes como antes, 1.000.000 de comparaciones, 1 día.
Tarea B	Ordenar 1000 expedientes como antes, 1.000.000 de comparaciones, 1 día.
Tarea C	Intercalar 2000 expedientes, 2000 comparaciones, 1 minuto.
Total	2 días y un minuto.

¡Ordenamos 2000 libros en poco más de 2 días!
¿Podemos hacer algo mejor?

Ordenar 2000 expedientes, aprovechando más la idea

Tarea A Ordenar 1000 expedientes:

- **Tarea AA** Ordenar 500, 250.000 comparaciones, $\frac{1}{4}$ día.
- **Tarea AB** Ordenar 500, 250.000 comparaciones, $\frac{1}{4}$ día.
- **Tarea AC** Intercalar 1000, 1000 comparaciones, $\frac{1}{2}$ minuto.

Total Tarea A 501.000 comparaciones, $\frac{1}{2}$ día y $\frac{1}{2}$ minuto.

Tarea B Como Tarea A, 501.000 comparaciones, $\frac{1}{2}$ día y $\frac{1}{2}$ minuto.

Tarea C Intercalar 2000, 2000 comparaciones, 1 minuto.

Total 1.004.000 comparaciones, 1 día y 2 minutos.

Ordenar 2000 expedientes, aprovechando más la idea

Tarea A Ordenar 1000 expedientes:

- Tarea AA Ordenar 500 expedientes:
  - Tarea AAA Ordenar 250 expedientes, 62.500 comparaciones, $\frac{1}{16}$ día.
  - Tarea AAB Ordenar 250 expedientes, 62.500 comparaciones, $\frac{1}{16}$ día.
  - Tarea AAC Intercalar 500 expedientes, 500 comparaciones, $\frac{1}{4}$ minuto.
  - Total Tarea AA 125.500 comparaciones, $\frac{1}{8}$ día y $\frac{1}{4}$ minuto.
- Tarea AB 125.500 comparaciones, $\frac{1}{8}$ día y $\frac{1}{4}$ minuto.
- Tarea AC Intercalar 1000, 1000 comparaciones, $\frac{1}{2}$ minuto.
- Total Tarea A 252.000 comparaciones, $\frac{1}{4}$ día y 1 minuto.

Tarea B 252.000 comparaciones, $\frac{1}{4}$ día y 1 minuto.

Tarea C Intercalar 2000, 2000 comparaciones, 1 minuto.

Total 506.000 comparaciones, $\frac{1}{2}$ día y 3 minutos.

Reflexionando sobre lo que acabamos de hacer

ordenar[1] bloques de	tardanza
2000 expedientes	4 días
1000 expedientes	2 días y 1 min
500 expedientes	1 día y 2 min
250 expedientes	1/2 día y 3 min
125 expedientes	1/4 día y 4 min
63 expedientes	1/8 día (1 hora) y 5 min
32 expedientes	1/2 hora y 6 min
16 expedientes	1/4 hora y 7 min
8 expedientes	1/8 hora y 8 min
4 expedientes	1/16 hora (4min) y 9 min
2 expedientes	2 min y 10 min
1 expedientes	1 min y 11 min

[1] usando ordenación por selección o por inserción

Conclusión

- ¿Por qué no “ordenar” (con ordenación por selección o inserción) bloques de 1, y luego intercalar reiteradamente?
- 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).
- Ahora lo escribimos en Haskell.

En Haskell

```
merge_sort :: [T] → [T]
merge_sort [] = []
merge_sort [t] = [t]
merge_sort ts = merge sts1 sts2
                    where
                        sts1 = merge_sort ts1
                        sts2 = merge_sort ts2
                        (ts1,ts2) = split ts

split :: [T] → ([T],[T])
split ts = (take n ts, drop n ts)
           where n = length ts ÷ 2
```

En pseudocódigo

{Pre: n $\geq$ der $\geq$ izq $>$ 0 $\wedge$ a = A}
proc merge_sort_rec (**in**/**out** a: **array**[1..n] **of T**, **in** izq,der: **nat**)
    var med: **nat**
    if der $>$ izq $\rightarrow$ med:= (der+izq) $\div$ 2
                merge_sort_rec(a,izq,med)
            {a[izq,med] permutación ordenada de A[izq,med]}
                merge_sort_rec(a,med+1,der)
        {a[med+1,der] permutación ordenada de A[med+1,der]}
                merge(a,izq,med,der)
                {a[izq,der] permutación ordenada de A[izq,der]}
    fi
end proc
{Post: a permutación de A $\wedge$ a[izq,der] permutación ordenada de A[iz

Algoritmo principal

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

Intercalación en pseudocódigo

```
proc merge (in/out a: array[1..n] of T, in izq,med,der: nat)
        var tmp: array[1..n] of T
              j,k: nat
        for i:= izq to med do tmp[i]:=a[i] od
        j:= izq
        k:= med+1
        for i:= izq to der do
              if j ≤ med ∧ (k > der ∨ tmp[j] ≤ 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

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,$\lfloor$(n+1)/2$\rfloor$) y a merge_sort_rec(a,$\lfloor$(n+1)/2$\rfloor$+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,izq,der) cuando desde izq hasta der hay m celdas.
- O sea, cuando m = der + 1 - izq.
- Si m = 0, izq = der + 1, la condición del **if** es falsa, t(m) = 0.
- Si m = 1, izq = der, la condición del **if** es falsa también, t(m) = 0.
- Si m > 1, izq > der 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 $k > 1$
- $$\begin{aligned} t(m) &= t(2^k) \\ &\leq t(\lceil 2^k/2 \rceil) + t(\lfloor 2^k/2 \rfloor) + 2^k \\ &= t(2^{k-1}) + t(2^{k-1}) + 2^k \\ &= 2 * t(2^{k-1}) + 2^k \end{aligned}$$
- $$\begin{aligned} \frac{t(2^k)}{2^k} &\leq \frac{2*t(2^{k-1})+2^k}{2^k} \\ &= \frac{2*t(2^{k-1})}{2^k} + \frac{2^k}{2^k} \\ &= \frac{t(2^{k-1})}{2^{k-1}} + 1 \end{aligned}$$

Solución (potencias de 2)

- $$\begin{aligned}
\frac{t(2^k)}{2^k} &\leq \frac{t(2^{k-1})}{2^{k-1}} + 1 \\
&\leq \frac{t(2^{k-2})}{2^{k-2}} + 1 + 1 \\
&= \frac{t(2^{k-2})}{2^{k-2}} + 2 \\
&\leq \frac{t(2^{k-3})}{2^{k-3}} + 3 \\
&\dots \\
&\leq \frac{t(2^0)}{2^0} + k \\
&= t(1) + k \\
&= k
\end{aligned}$$

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

Cota inferior y superior

- Partimos de $t(m) \leq t(\lceil m/2 \rceil) + t(\lfloor m/2 \rfloor) + m$,
- llegamos a $t(m) \leq m * \log_2 m$ para m potencia de 2.
- También vale $t(m) \geq t(\lceil m/2 \rceil) + t(\lfloor m/2 \rfloor) + \frac{m}{2}$,
- que nos permite mostrar que $t(m) \geq \frac{m * \log_2 m}{2}$ para m potencia de 2.
- Conclusión: ordenación por intercalación es del orden de $n * \log_2 n$ para n potencia de 2.

Cuando n no es potencia de 2

Si n no es potencia de 2, sea k tal que $2^k \leq n < 2^{k+1}$ y por lo tanto $k \leq \log_2 n \leq k+1$.

$$
\begin{array}{rcll}
t(n) & \leq & t(2^{k+1}) & \text{por ser } t \text{ creciente} \\
 & \leq & 2^{k+1} * (k+1) & \text{por ser } 2^{k+1} \text{ potencia de 2} \\
 & = & 2^{k+1} * k + 2^{k+1} & \text{por distributividad} \\
 & \leq & 2^{k+1} * k + 2^{k+1} * k & \text{por } k \geq 1 \\
 & = & 2 * 2^{k+1} * k & \text{por suma} \\
 & = & 4 * 2^k * k & \text{por multiplicación} \\
 & \leq & 4 * n * \log_2 n & \text{por } 2^k \leq n \text{ y } k \leq \log_2 n
\end{array}
$$

Cuando n no es potencia de 2

- Obtuvimos $t(n) \leq 4 * n * \log_2 n$.
- También podemos obtener $t(n) \geq \frac{1}{8} * n * \log_2 n$.
- Por lo tanto, ordenación por intercalación es del orden de $n * \log_2 n$ incluso cuando n no es potencia de.

Problema del bibliotecario

Un bibliotecario tarda un día en ordenar alfabéticamente una biblioteca con 1000 expedientes. ¿Cuánto tardará en ordenar una con 2000 expedientes?

Si el algoritmo que usa el bibliotecario es el de ordenación por intercalación:

expedientes cantidad	comparaciones $n * \log_2 n$	tiempo días
1000	10.000	1
2000	22.000	2,2

Para alumnos decepcionados

- Algunos alumnos se decepcionan cuando ven esos números, ya que hasta hace un rato se trataba sólo de minutos.
- Notar que ahora hemos asumido que el bibliotecario es capaz de hacer sólo 10.000 comparaciones por día, contra 1.000.000 que asumíamos cuando usaba ordenación por selección.
- Un bibliotecario tarda un día en ordenar alfabéticamente una biblioteca con 1000 expedientes usando ordenación por **selección**. ¿Cuánto tardará en ordenar una con 2000 expedientes usando ordenación por **intercalación**?
- 1.000.000 de comparaciones = 1 día.
- 22.000 comparaciones = 11 minutos.

Ayuda del “colaborador”

- La idea original fue
  - Que cada uno (bibliotecario + colaborador) 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: $0 \leq$ der $\leq$ n $\wedge$ 1 $\leq$ izq $\leq$ n+1 $\wedge$ izq-1 $\leq$ der $\wedge$ a = A}
proc quick_sort_rec (**in**/**out** a: **array**[1..n] **of T**, **in** izq,der: **nat**)
    var pivot: **nat**
    if der $>$ izq $\rightarrow$ partition(a,izq,der,pivot)
                izq $\leq$ pivot $\leq$ der
                elementos en a[izq,pivot-1] $\leq$ a[pivot]
                elementos en a[pivot+1,der] $>$ a[pivot]}
            quick_sort_rec(a,izq,pivot-1)
            quick_sort_rec(a,pivot+1,der)
    fi
end proc
{Post: a permut. de A $\wedge$ a[izq,der] permut. ordenada de A[izq,der]}

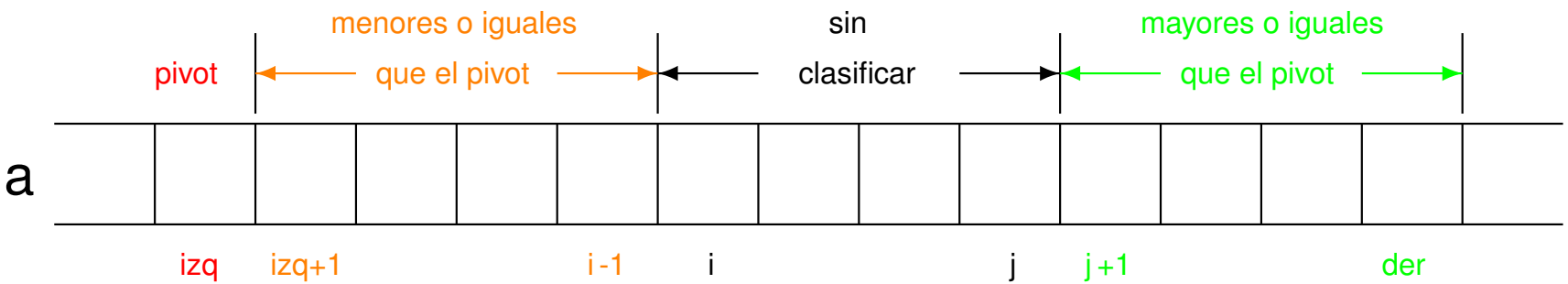
Algoritmo principal

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

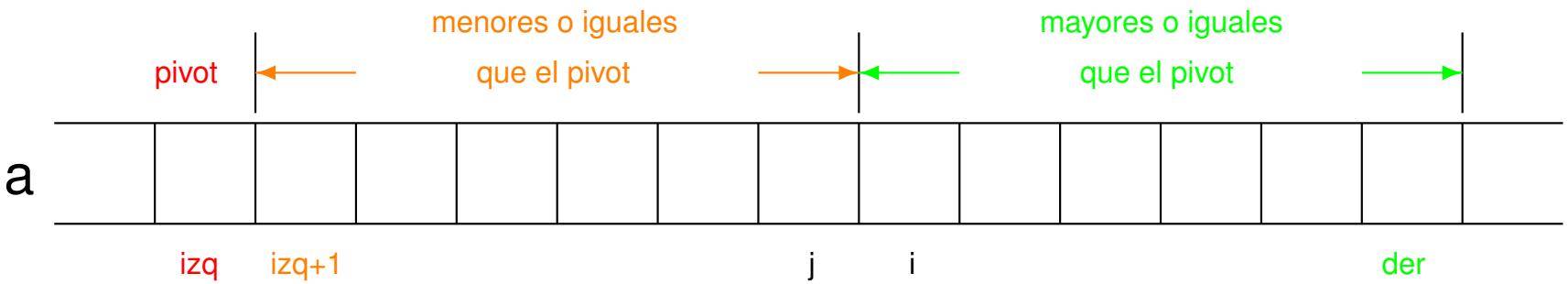
Procedimiento partition

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

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** der > izq **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, der+1-izq),
- 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 oden 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 oden inverso, es del orden de n^2
- mejor caso: es del orden de $n \log_2 n$, cuando el procedimiento parte exactamente al medio.