

CLASIFICACIÓN EN MEMORIA PRINCIPAL Tema 2

Tutor: Enrique Carmona
Asignatura: Estructuras de Datos y Algoritmos

TEMA II

CLASIFICACION EN MEMORIA PRINCIPAL

2.1. Introducción.

2.2. Métodos de clasificación directos:

2.2.1. Clasificación por inserción:

- **Directa**
- **Binaria.**

2.2.2. Clasificación por selección directa.

2.2.3. Clasificación por intercambio:

- **Directo (método de la burbuja)**
- **Método por sacudida**

2.3. Métodos de clasificación avanzados:

2.3.1. Inserción por incremento decreciente (Shell).

2.3.2. Clasificación por montón (Floyd)

2.3.3. Clasificación por partición (clasif. rápida)

- **Obtención del k-ésimo menor elemento.**

El Problema de la Clasificación

Dado el conjunto de elementos:

$$a_1, a_2, \dots, a_n$$

t.q. cada uno tiene un campo llave asociado

$$a_{k1}, a_{k2}, \dots, a_{kn}$$

resolver el *problema de la clasificación* consiste en:

ordenar esos elementos t. q.

$$f(a_{kj}) \leq f(a_{kp}) \leq \dots \leq f(a_{kr})$$

donde $j, p, r \in \{1, 2, \dots, n\}$ y f = función de ordenación

Premisa: En el conjunto de llaves es imprescindible que se pueda establecer una relación de orden, es decir, para cualquiera tres llaves a_{ki}, a_{km}, a_{kr} se cumple que:

$$1. a_{ki} < a_{km} \text{ o } a_{ki} = a_{km} \text{ o } a_{ki} > a_{km}$$

$$2. \text{ Si } a_{ki} < a_{km} \text{ y } a_{km} < a_{kr} \text{ entonces } a_{ki} < a_{kr}$$

Tipos de Clasificación:

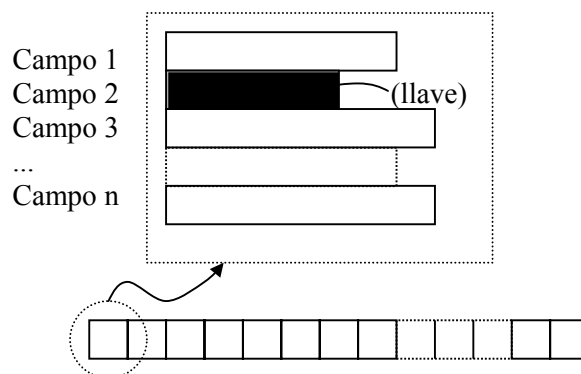
- Clasificación interna (Estructuras en Mem. Principal)
- Clasificación externa (Estructuras en Mem. Secundaria)

Interna	Externa
En Memoria Primaria: rápida	En Memoria Secundaria: lenta
Dispone todos los datos simult.	Dispone unos pocos a la vez

Definición de tipos (para Modula2):

```
TYPE Tipo_Datos = RECORD
    llave : Tipo_llave;
    (*otros componentes*)
END
```

```
VAR  a: ARRAY [1..n] OF Tipo_Datos;
```



- **Existe una gran diversidad de algoritmos.**
- **Cosas a tener en cuenta en la elección del algoritmo:**
 1. Tamaño del conjunto de datos.
 2. Eficacia. Depende de (1).
 3. Estabilidad: el orden relativo de los elementos con iguales claves permanece inalterado en el proceso de clasificación, es decir, no intercambia elementos con claves iguales.
 4. Comportamiento natural: Si el mejor caso es que el arreglo esté inicialmente ordenado.
 5. Tipo de clasificación: interna o externa.

- **Medida de la eficacia:**

Depende de:

- Número C de comparaciones.
- Número M de movimientos.

Si ORDEN: n^2 \Rightarrow Métodos Directos
 Si ORDEN: $n * \log(n)$ \Rightarrow Métodos Avanzados

(Siendo "n" el número de elementos a clasificar)

Algoritmos de clasificación interna (memoria principal)

Métodos directos:

- Clasificación por inserción:
 - Directa
 - Binaria
- Clasificación por selección directa.
- Clasificación por intercambio:
 - Directo (método de la burbuja)
 - Método por sacudida

Clasificación por inserción directa.

```
BEGIN
  FOR i := 2 TO n DO
    a[0] := a[i]; (*garantiza la salida del bucle while*)
    j := i;
    WHILE a[0] < a[j-1] DO
      a[j] := a[j-1];
      j := j-1
    END;
    a[j] := a[0] (*inserción*)
  END
END
```

ANÁLISIS:

Número de Comparaciones

$$C_{i, \min} = 1$$

$$C_{i, \max} = i \quad (\text{teniendo en cuenta el centinela})$$

$$C_{i, \text{medio}} = i/2$$

- Mejor caso (ya están ordenados):

$$C_{\min} = 1_2 + \dots + 1_n = n-1$$

- Peor caso (ordenados en orden inverso):

$$C_{\max} = 2+3+\dots+n = (n^2+n-2)/2$$

- Caso Promedio:

$$C_{\text{prom}} = 1/2 (2+3+\dots+n) = (n^2+n-2)/4$$

Número de Movimientos

- Mejor caso (*nunca se entra en el cuerpo del while*):

$$M_{\min} = 2(n-1)$$

- Peor caso ($i=2 \Rightarrow 1 \text{ mov}, i=3 \Rightarrow 2 \text{ mov}, \dots, i=n \Rightarrow n-1 \text{ mov}$):

$$M_{\max} = 2(n-1) + [1+2+\dots+(n-1)]_{\text{while}} = (n^2+3n-4)/2$$

- Caso Promedio:

$$M_{\text{prom}} = M_{\text{prom}_{\text{while}}} + 2(n-1) =$$

$$= C_{\text{prom}_{\text{while}}} + 2(n-1) = (n^2+9n-10)/4$$

Ejemplo de clasificación por inserción directa.

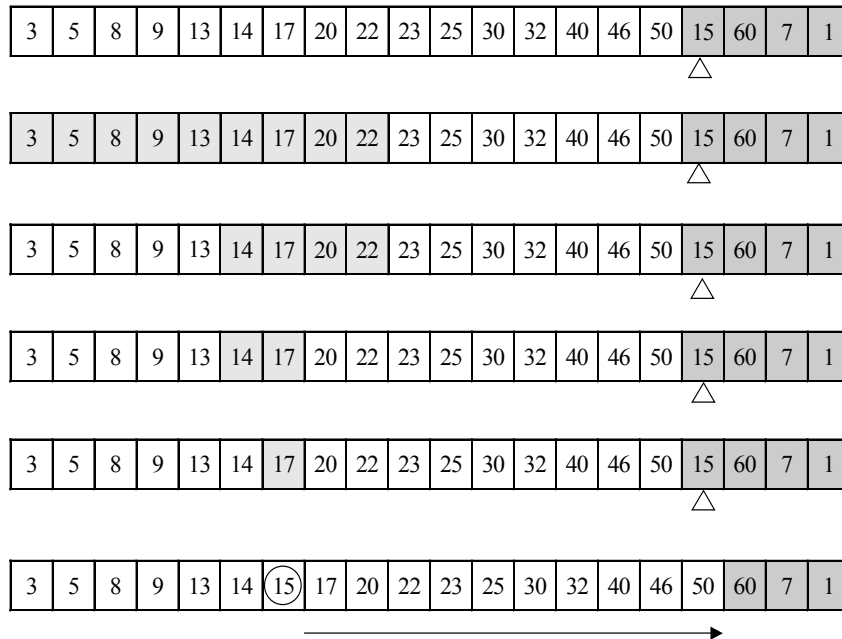
	a[0]	a[1]	a[2]	...	
	↓	↓	↓		
Llaves iniciales		44	55	12	42 94 18 06 67
i=2	55	44	55	12	42 94 18 06 67
i=3	12	12	44	55	42 94 18 06 67
i=4	42	12	42	44	55 94 18 06 67
i=5	94	12	42	44	55 94 18 06 67
i=6	18	12	18	42	44 55 94 06 67
i=7	06	06	12	18	42 44 55 94 67
i=8	67	06	12	18	42 44 55 67 94

Mejora \Rightarrow **Inserción binaria:**

Darse cuenta de que la secuencia destino donde debe insertarse el nuevo elemento ya está ordenada.

1. Búsqueda binaria para localizar el lugar de inserción.
2. Desplazar elementos.
3. Insertar.
4. Repetir para todos los elementos.

Algoritmo de clasificación por inserción binaria



```

BEGIN
  FOR i:=2 to N DO
    X:= a[i];
    L:= 1;
    R:=i;
    WHILE L<R DO
      m:=(L+R) DIV 2;
      IF a[m] <= x THEN
        L:= m+1
      ELSE
        R:= m
      END
    END;
    FOR J:= i TO R+1 BY -1 DO
      a[j]:= a[j-1] (*desplazamiento*)
    END;
    a[R]:=x (*inserción*)
  END
END

```

Analisis:

La posición de inserción se encuentra si $L=R$ en R , se habrá dividido por 2 tantas veces como permita la longitud del segmento de arreglo que se está manejando.

Para $i=m$ se realizan $(\log_2 m)$ divisiones ya que la longitud “m” del arreglo se puede poner como $m=2^{n^\circ \text{ de divisiones}}$

Sumando para todas las iteraciones:

$$C = \sum_{i=1}^n \lceil \log_2 i \rceil$$

Aproximando por la integral:

$$C = \int_1^n \log_2 x dx = n * (\log_2 n - c) + c$$

$$\text{donde } c = \log_2 e = 1/\ln 2 = 1.44269$$

- Mejora el número de comparaciones pero no el de movimientos que sigue siendo, en el peor de los casos (ordenados en orden inverso) o en el caso promedio, del orden de n^2 :

$$M_{\min} = 2_2 + 2_3 + \dots + 2_n = (n-1)$$

$$M_{\max} = [1+2+\dots+(n-1)]_{\text{for}} + 2(n-1) = (n^2+3n-4)/2$$

$$M_{\text{prom}} = \frac{1}{2} [1+2+\dots+(n-1)]_{\text{for}} + 2(n-1) = (n^2+7n-8)/2$$

- Si los elementos ya están ordenados es peor que el método de inserción directa ($C_{\min} = n-1$).

Clasificación por selección directa.

Al hacer un movimiento colocamos el elemento en su sitio definitivo.

1. Seleccionar el elemento con clave menor.
2. Intercambiarlo con el primer elemento.
3. Repetir estas operaciones con los $n-1$ elementos restantes, luego con los $n-2$ hasta que no quede más que un elemento (el más grande).

Ejemplo de clasificación por selección directa.

Llaves iniciales

44	55	12	42	94	18	06	67
06	55	12	42	94	18	44	67
06	12	55	42	94	18	44	67
06	12	18	42	94	55	44	67
06	12	18	42	94	55	44	67
06	12	18	42	44	55	94	67
06	12	18	42	44	55	94	67
06	12	18	42	44	55	67	94

◆ Es contrario al de inserción directa:

Inserción directa: en cada paso *un* elemento siguiente de la secuencia fuente y *todos* los del arreglo destino.

Selección directa: *Todos* los elementos del arreglo fuente para encontrar *el que* es menor y depositarlo como el siguiente elemento de la secuencia destino.

Clasificación por Selección Directa.

FOR i:=1 TO n-1 DO

Asignar el índice del elemento más pequeño de $a_i...a_n$ a k;

Intercambiar a_i con a_k ;

END

BEGIN

FOR i := 1 TO n-1 DO

k := i;

x:= a[i]; *(*elemento actual a analizar*)*

FOR j := i+1 TO n DO *(*asigna el índice del*

IF a[j] < x THEN *elemento más pequeño*
k := j; *de $a_i...a_n$ a k*)*

x:= a[k] *(*en x el elemento menor*)*

END

END;

a[k] := a[i]; *(*intercambio*)*

a[i] := x

END

END

Análisis:

- ♦ **Comparaciones** (su número es independiente del orden de las llaves):

$$C = (n^2 - n) / 2$$

- ♦ **Movimientos:**

$$M_{\min} = 3(n-1)$$

$$M_{\max} = [1+3+5+\dots+(n-1)]_{\text{if}} + 3(n-1) = n^2/4 + 3(n-1)$$

$$M_{\text{prom}} \cong n * [\ln(n) + \gamma] \quad \text{donde } \gamma = 0,577216$$

Clasificación por intercambio directo.

- Característica principal: Intercambio de dos elementos.
- Método: Comparar e intercambiar pares de elementos contiguos hasta clasificar todos los elementos.

Ejemplo de clasificación por el método de la burbuja.

(Ver el arreglo a clasificar por columnas).

i =	1	2	3	4	5	6	7
44	44	12	12	12	12	06	06
55	12	42	42	18	06	12	12
12	42	44	18	06	18	18	18
42	55	18	06	42	42	42	42
94	18	06	44	44	44	44	44
18	06	55	55	55	55	55	55
06	67	67	67	67	67	67	67
67	94	94	94	94	94	94	94

- En cada pase sobre el arreglo produce el descenso de una burbuja => clasificación por burbuja.

¿Cómo paso, por ejemplo, de $i=1$ a $i=2$ en el ejemplo anterior de clasificación por el método de la burbuja?

Principio de $i=1$

Final de $i=1$

44	44	44	44	44	44	44	44
55	55	12	12	12	12	12	12
12	12	55	42	42	42	42	42
42	42	42	55	55	55	55	55
94	94	94	94	94	18	18	18
18	18	18	18	18	94	06	06
06	06	06	06	06	06	94	67
67	67	67	67	67	67	67	94

Clasificación por intercambio directo.

También llamado método de la burbuja.

```
BEGIN
  FOR i:=1 TO n-1 DO
    FOR j:=1 TO n-i DO
      (*comparar elementos contiguos*)
      IF a[j]>a[j+1] THEN
        aux:=a[j+1];    (*intercambiar*)
        a[j+1]:=a[j];
        a[j]:=aux;
      END
    END;
  END;
END;
```

Mejoras:

Recordar si durante un pase ha tenido algún intercambio. Lo contrario indica el final de la clasificación.

Recordar la posición del último intercambio. Antes de esa posición ya todo esta ordenado.

Una clave pequeña en una posición pesada requiere un gran número de pasadas para ordenarse. Esto sugiere alternar la dirección de los pases consecutivos.

Análisis:

Comparaciones (es independiente del orden de llaves):

$$C = (n^2 - n)/2$$

Movimientos:

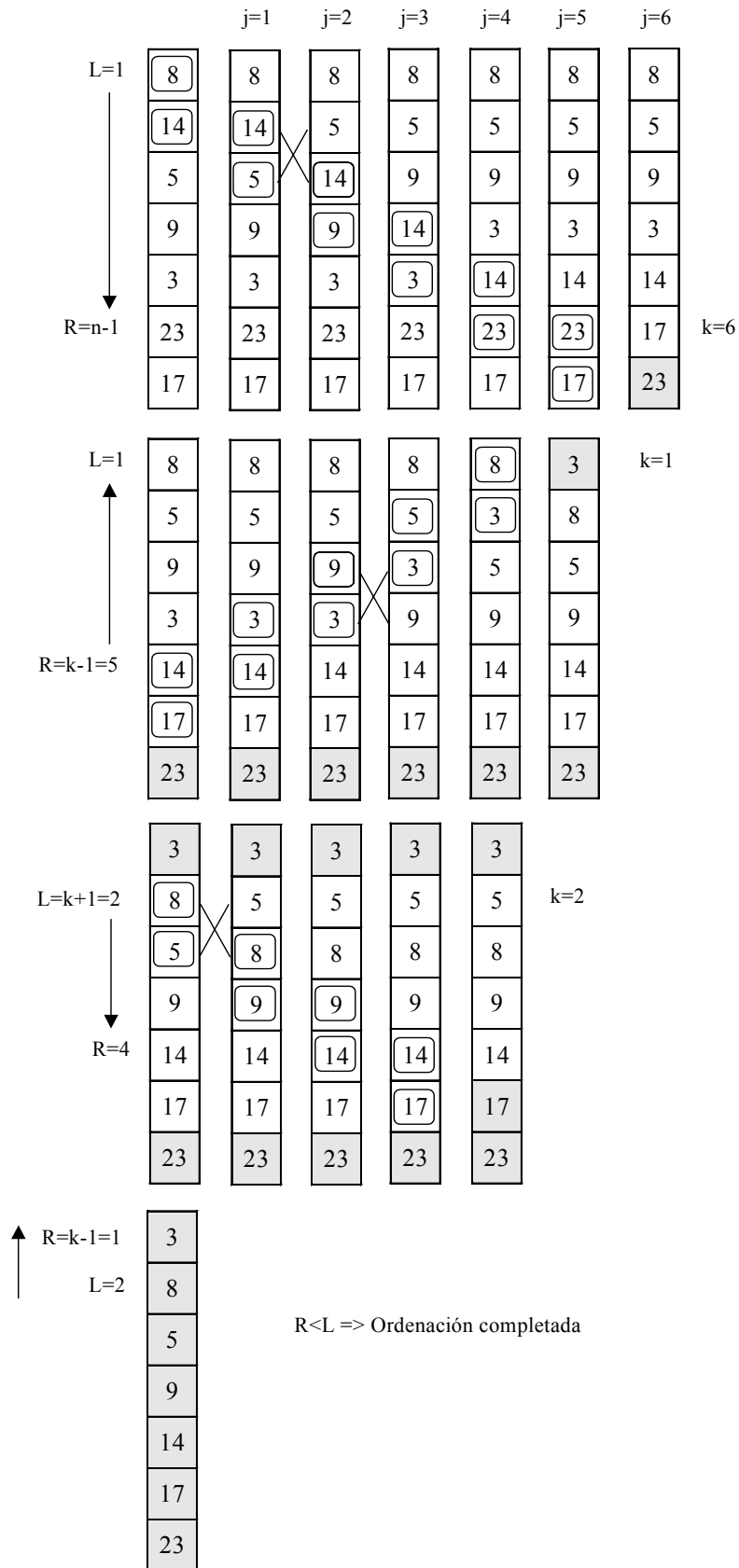
$$M_{\min} = 0$$

$$M_{\max} = 3 * C = 3(n^2 - n)/2$$

$$M_{\text{prom}} = \frac{1}{2} M_{\max} = 3(n^2 - n)/4$$

- Las mejoras no afectan al número de intercambios, sólo al de comparaciones.
- Los intercambios son mucho más costosos que las comparaciones.
- Se usa cuando se sabe que los elementos ya están casi en orden.

Clasificación por sacudida o por vibración



Algoritmo de clasificación por sacudida o por vibración

```
BEGIN
  L:=1; R:=n-1; k:=1;
  REPEAT
    FOR j:=L TO R DO
      IF a[j]>a[j+1] THEN
        aux:=a[j+1];
        a[j+1]:=a[j];
        a[j]:=aux;
        k:=j
      END
    END;
    R:=k-1;
    FOR j:=R TO L BY -1 DO
      IF a[j]>a[j+1] THEN
        aux:= a[j+1];
        a[j+1]:=a[j];
        a[j]:=aux;
        k:=j;
      END;
    END;
    L:=k+1;
  UNTIL L>R;
END;
```

Comparación de los métodos directos de clasificación de arreglos.

n: n° de elementos a clasificar.

C: comparaciones requeridas entre claves.

M: movimientos de elementos.

Fórmulas analíticas para los métodos de clasificación directa:

		Mín	Prom	Máx
Inserción Directa	C	$(n-1)$	$(n^2+n-2)/4$	$(n^2+n-2)/2$
	M	$2(n-1)$	$(n^2+9n-10)/4$	$(n^2+3n-4)/2$
Inserción Binaria	C	$n \lg n$	$n \lg n$	$n \lg n$
	M	$n-1$	$(n^2+7n-8)/4$	$(n^2+3n-4)/2$
Selección Directa	C	$(n^2-n)/2$	$(n^2-n)/2$	$(n^2-n)/2$
	M	$3(n-1)$	$n[\ln(n)+0.57]$	$n^2/4+3(n-1)$
Intercambio Directo	C	$(n^2-n)/2$	$(n^2-n)/2$	$(n^2-n)/2$
	M	0	$3(n^2-n)/4$	$3*(n^2-n)/2$
Por Vibración	C	$(n-1)$	$\frac{1}{2}[n^2-n(k_2+\ln n)]$	$O(n^2)$
	M	0	$O(n^2)$	$O(n^2)$

- La complejidad de los métodos directos de clasificación de arreglos es del $O(n^2)$.

- Veremos que para los métodos de clasificación avanzada la complejidad es del $O(n*\lg(n))$.

Clasificación por inserción por incremento decreciente (Shell)

- Constituye un refinamiento del método de clasificación por inserción directa.

- Dada la secuencia

$$h_1, h_2, \dots, h_t$$

tal que

$$h_t = 1 \text{ y } h_i > h_{i+1}$$

se ordenan⁽¹⁾ los elementos que difieren en h_1 posiciones, después los que difieren en h_2 , etc., hasta los que difieren en $h_t=1$ posición.

Ejemplo:

44 55 12 42 94 18 06 67

Clasificación-4 ($h_1=4$)

44 18 06 42 94 55 12 67

Clasificación-2 ($h_2=2$)

06 18 12 42 44 55 94 67

Clasificación-1 ($h_3=1$)

06 12 18 42 44 55 67 94

⁽¹⁾ Siguiendo el método de inserción directa

Extensión del arreglo de datos para alojar los *centinelas* en el peor de los casos:

a: ARRAY $[(-h_1+1) .. n]$ OF item

Algoritmo Clasificación de Shell:

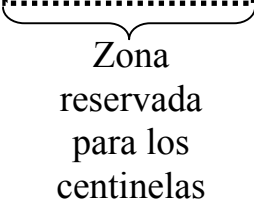
```
CONST  t=3; (*Utiliza una serie de 3 incrementos*)
VAR    i, j, k, s: tipo_indice;
        m: [1..t];
        h : ARRAY [1..t] OF INTEGER

BEGIN
    h[1]:=4; h[2]:=2; h[3]:=1;
    FOR m:=1 TO t DO
        k:= h[m]; s:=-k;
        FOR i:= k+1 TO n DO
            j:=i-k;
            IF s=0 THEN s:=-k END;
            s:=s+1; a[s]:= a[i];
            WHILE a[s]< a[j] DO
                a[j+k]:= a[j]; j:=j-k
            END
            a[j+k]:=a[s]
        END;
    END
END;
```

Ejemplo de aplicación del método de clasificación Shell

- Serie: $h[3], h[2], h[1]$
- m: FOR m:=1 TO $N^{\circ}_{total_elementos_de_la_serie}$
- k: $k:=h[m]$ (Offset a izquierdas y a derechas)
- s: (Inicio $s:=-k$), $s:=s+1$ (apunta a la posición centinela)
- i: (Inicio $i:=k+1$), $i:=i+1$ (recorre todo el array)
- j: (inicio $j:=i-k$), $j:=j-k$ (apunta a los elementos que se comparan con el centinela)

Secuencia: $h[1]=3, h[2]=2, h[3]=1$													
			a[-2]	a[-1]	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]
						(4)	7	(15)	(1)	14	(3)	(2)	10
m=1, $k=h[m]=h[1]=3, s=-k=-3$													
i=k+1=4	s=-2	j=1,-2	1			1			4				
i=5	s=-1	j=2		14			7			14			
i=6	s=0	j=3,0			3			3			15		
i=7	s=-2	j=4,1	2			1			2			4	
i=8	s=-1	j=5,2		10			7			10			14
						(1)	7	(3)	2	(10)	15	(4)	14
m=2, $k=h[m]=h[2]=2, s=-k=-2$													
i=3	s=-1	j=1		3		1		3					
i=4	s=0	j=2,0			2		2		7				
i=5	s=-1	j=3		10		1		3		10			
i=6	s=0	j=4			15		2		7		15		
i=7	s=-1	j=5,3		4		1		3		4		10	
i=8	s=0	j=6,4			14		2		7		14		15
						1	2	3	7	4	14	10	15
m=3, $k=h[m]=h[3]=1, s=-k=-1$													
i=2	s=0	j=1			2	1	2						
i=3	s=0	j=2			3	1	2	3					
i=4	s=0	j=3			7	1	2	3	7				
i=5	s=0	j=4,3			4	1	2	3	4	7			
i=6	s=0	j=5			14	1	2	3	4	7	14		
i=7	s=0	j=6,5			10	1	2	3	4	7	10	14	
i=8	s=0	j=7			15	1	2	3	4	7	10	14	15



Zona reservada para los centinelas

Análisis de movimientos del peor caso:

- n es potencia de 2 y $n/2$ llaves se encuentran con valores grandes en las posiciones pares y $n/2$ con valores pequeños en las impares, ambas ordenadas de forma inversa.
- Los elementos de la secuencia se toman pares (excepto el último).

Al llegar al último pase ($h_1=1$), los $n/2$ valores grandes están todavía en posiciones pares y los $n/2$ pequeños en posiciones impares

9	90	7	80	5	70	3	60	1	50
1	50	3	60	5	70	7	80	9	90

El i -ésimo menor elemento está en la posición $2i-1 \Rightarrow$ Para colocarlo en su lugar, habrá que moverlo:

$$(2i-1) - i = i - 1 \text{ posiciones}$$

Para todos los $n/2$ elementos menores:

$$M_{h1} = \text{SUM}_{(i=1:n/2)} i-1 = (n^2-2n)/8$$

En cada pase anterior, de incremento h_k , realiza h_k inserciones directas de aproximadamente n/h_k elementos cada una. Se sabe que la inserción directa es del orden $N^2 \Rightarrow$ El coste total de un pase será

$$M_{hk} = h_k * N^2 = h_k (n/h_k)^2 = n^2/h_k$$

$+ (O[n^2] \text{ para } h_1)$

Y sumando para todos los pases:

$$M = O[\text{SUM}(i=2:t) N^2/h_i] = O[N^2 \text{ SUM}(i=2:t) 1/h_i] = O[n^2]$$

puesto que $0 < \text{SUM}(i=2:t) 1/h_i < 1$

Consideraciones finales:

- En la elección de incrementos con múltiplos de 2, se observa que, en cada pase, la Clasificación- i clasifica dos grupos ya ordenados por la anterior Clasificación- $2i$ (redundancia) \Rightarrow Deben evitarse conjuntos de incrementos en los que unos sean múltiplos de otros ya que no aprovechan la clasificación realizada anteriormente

- Ejemplo de secuencia eficiente:

Secuencia de incrementos de Hibbard:

$$1, 3, 7, 15, 31, 2^k-1, \dots$$

$$\begin{aligned} \text{donde } h_{k-1} &= 2 h_k + 1 \\ h_t &= 1 \\ t &= \lfloor \log_2 n \rfloor - 1 \end{aligned}$$

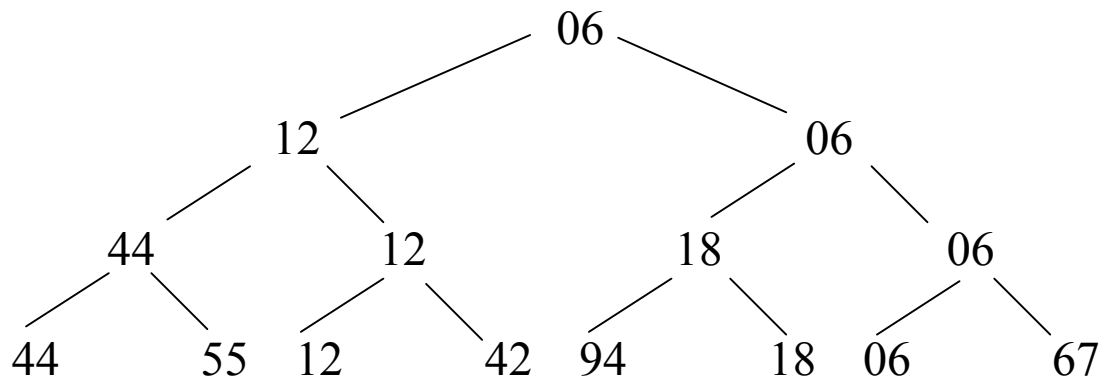
Coste de peor caso: $O[n^{3/2}]$ (mejora los métodos directos)

- Ciertos problemas para la elección de la mejor secuencia de incrementos debido a la dificultad matemática en demostrar el orden de eficacia de los mismos.

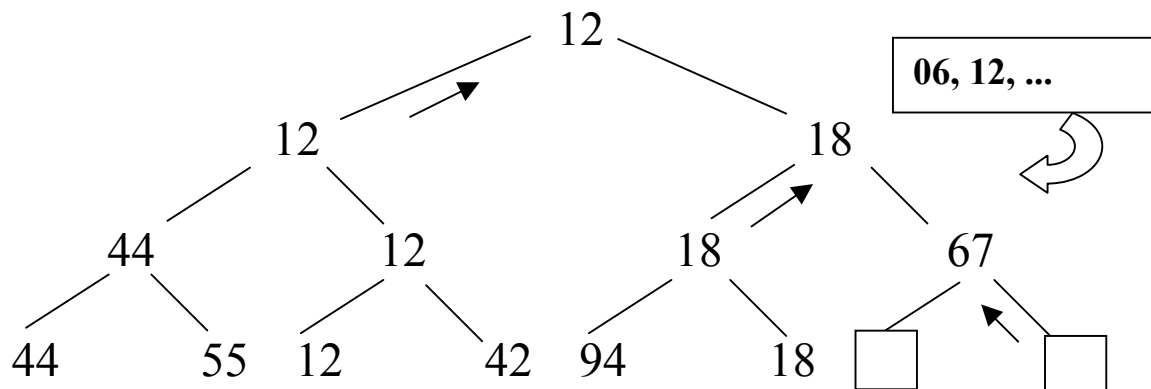
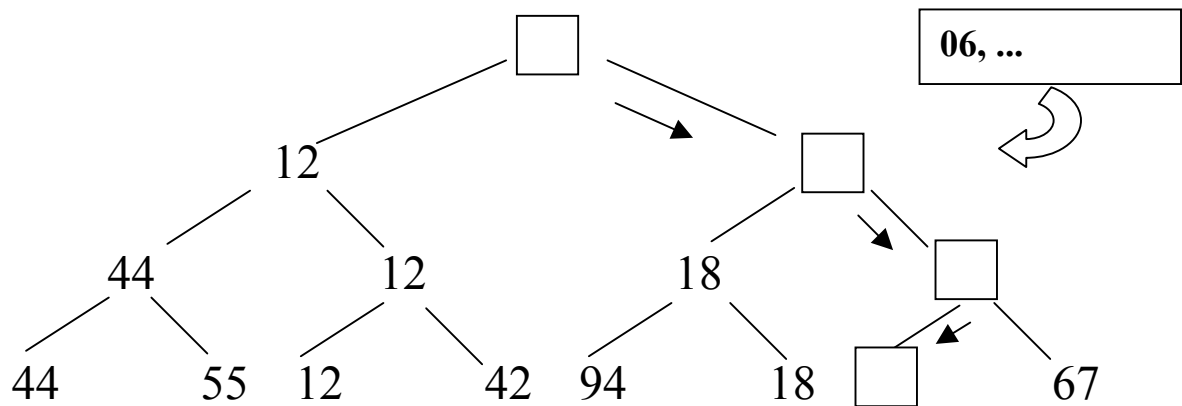
Clasificación por Montón

- Mejora el método de clasificación por selección directa.

1º) Construcción del árbol



2º) Proceso de clasificación (" $n \cdot \log n$ " Comparaciones)



Problema: Buscar una estructura de datos que represente el árbol y que además permita el proceso de clasificación *in situ*.

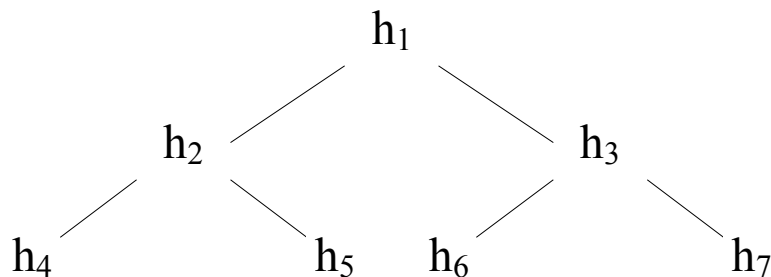
Solución: Montón en array (Floyd) a partir de la definición de montón de Williams y algoritmo de clasificación por montón.

Definición de montón de Williams:

Secuencia de llaves h_L, h_{L+1}, \dots, h_R tales que

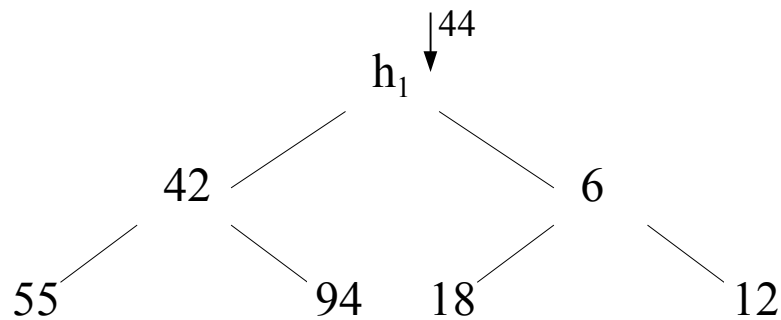
$$\left. \begin{array}{l} h_i \leq h_{2i} \\ h_i \leq h_{2i+1} \end{array} \right\} \text{para } i = L \dots R/2$$

Para todo montón $h_1 = \min(h_1, \dots, h_n)$

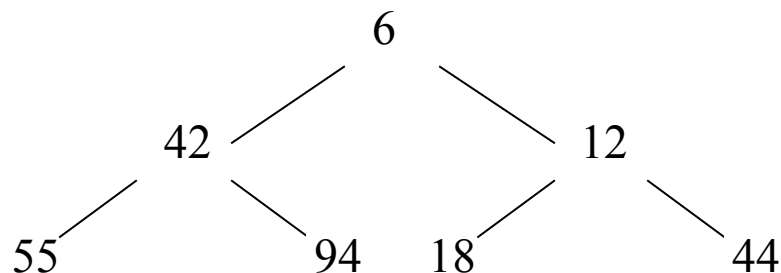


¿Cómo añadir un elemento x a un montón?

Dado el montón



al añadir, por ejemplo, el elemento $h_1=44$, se obtiene un nuevo *montón ampliado* poniendo éste sobre la cima y luego dejando que se desplace hacia abajo.



Según la idea de **Floyd** para construir un montón en un arreglo y hacer la clasificación in situ: dado h_1, \dots, h_n , sin duda, $h_{(n \div 2)+1}, \dots, h_n$ ya forman un montón.

El montón se amplía cada vez a la izquierda y cada elemento se sitúa correctamente.

44	55	12	42		94	18	06	67	
44	55	12	42		94	18	06	67	
44	55		06	42	94	18	12	67	
44	55	06	42		94	18	12	67	← Paso intermedio
44		42	06	55	94	18	12	67	
06	42	44	55		94	18	12	67	← Paso intermedio
06	42	12	55		94	18	44	67	

Algoritmo:

```

PROCEDURE amontonar(L,R:Tipo_indice);
  (*R:posición del extremo derecho del montón*)
  (*L:posición del elemento que se esta incorporando al
  montón*)
  VAR i,j:Tipo_indice; x:Tipo_datos;
  BEGIN
    i:=L;
    j:=2*L;
    x:=a[L];  (*elemento que se incorpora al montón*)
    IF (j<R) & (a[j+1]<a[j]) THEN
      j:=j+1
    END;
    (*j: indice del elemento que intercambiar de entre el 2i y el
    2i+1*)
    WHILE (j<=R) & (a[j]<x) DO
      a[i]:= a[j];
      a[j]:=x;      (*intercambio*)
      i:=j;
      j:=2*j;
      IF(j<R) & (a[j+1]<a[j]) THEN
        j:=j+1
      END
    END      (*verificar montón hasta el final*)
  END;
END amontonar;

```

Algoritmo (continuación):

Así, el proceso de *generar un montón* de n elementos *in situ* se describe:

```
L:=(n DIV 2) +1;  
R:=n;  
WHILE L>1 DO  
    L:=L-1;  
    amontonar(L, R)  
END;
```

Una vez creado, *para ordenarlo*, en cada paso se intercambia la posición del R-ésimo componente “x” con la cima, y este “x” se desplaza hacia abajo en el montón.

```
R:=n;  
WHILE R>1 DO  
    x:= a[1];  
    a[1]:= a[R]; a[R]:=x; (*Intercambio*)  
    R:=R-1;  
    (*el R-ésimo elemento se desplaza hacia abajo*)  
    amontonar(1,R);  
END;
```

06	42	12	55	94	18	44	67
67	42	12	55	94	18	44	06
12	42	67	55	94	18	44	06
12	42	18	55	94	67	44	06
18	42	44	55	94	67	12	06
42	55	44	67	94	18	12	06
44	55	94	67	42	18	12	06
55	67	94	44	42	18	12	06
67	94	55	44	42	18	12	06
94	67	55	44	42	18	12	06

```

        PROCEDURE amontonar(L, R : Tipo_índice);
VAR i, j : Tipo_índice; x : Tipo_datos;
BEGIN
    i := L;      (*índice del elemento actual que inserto*)
    j := 2*L;    (*índice del posible hijo izquierdo
                  del actual que inserto*)
    x := a[L];   (*elemento actual*)
    IF (j < R)    (*asegurarnos de que tiene hijo izq., si
                  no tiene izq. no tendrá derch., pues el
                  montón es árbol completo*)
        & (a[j+1] < a[j]) THEN j := j+1 END;
        (*j es el índice del hijo menor*)
    WHILE (j <= R) & (a[j] < x) DO
        a[i] := a[j]; a[j] := x; i := j; j := 2*j;
        (*Subo elemento menor e intercambio*)
        IF (j < R) & (a[j+1] < a[j]) THEN j := j+1 END
        (* j <= índice del menor del siguiente nivel* )
    END
END amontonar;

VAR L,R: Tipo_indice; x: Tipo_datos;
BEGIN
    (*Construcción del montón*)
    L := (n DIV 2) + 1;
    R := n;
    WHILE L > 1 DO
        L := L-1;
        amontonar (L,R)
    END;

    (*Intercambio de la cima del montón con el elemento R-ésimo*)
    WHILE R > 1 DO
        x := a[1];
        a[1] := a[R]; a[R] := x; (*intercambio*)
        R := R-1;
        amontonar (L,R) (* es lo mismo que “amontonar(1,R)” *)
    END

```

Análisis:

- * La eficiencia crece al crecer n (mejor que la clasificación de Shell).
- * Para n pequeños es poco eficiente.
- * Puede demostrarse que el n° de comparaciones es:

$$\text{Peor de los casos: } C_{\max} = 2n \log n - O[n]$$

$$\text{Caso Promedio: } C_{\text{med}} = 2n \log n - O[n \log(\log n)]$$

- * Como los movimientos dependen de que las comparaciones sean cierta:

$$M_{\max} \leq C_{\max}$$

$$M_{\text{med}} \leq C_{\text{med}}$$

Clasificación por partición (Clasificación rápida)

Se basa en el método de clasificación por intercambio.

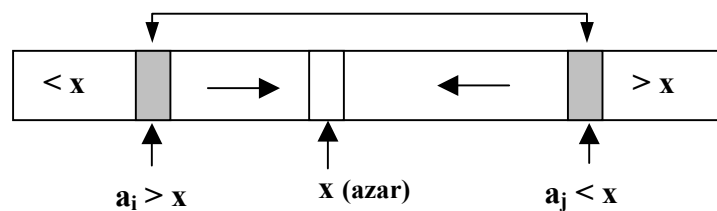
Método:

1º) Se selecciona un elemento al azar x .

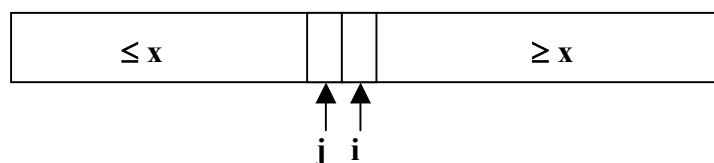
2º) Se rastrea desde la izquierda hasta un a_i tal que $a_i > x$.

3º) Se rastrea desde la derecha hasta un a_j tal que $a_j < x$.

4º) Se intercambian ($i=i+1, j=j-1$).



5º) Se repiten estos pasos hasta que $i > j$.

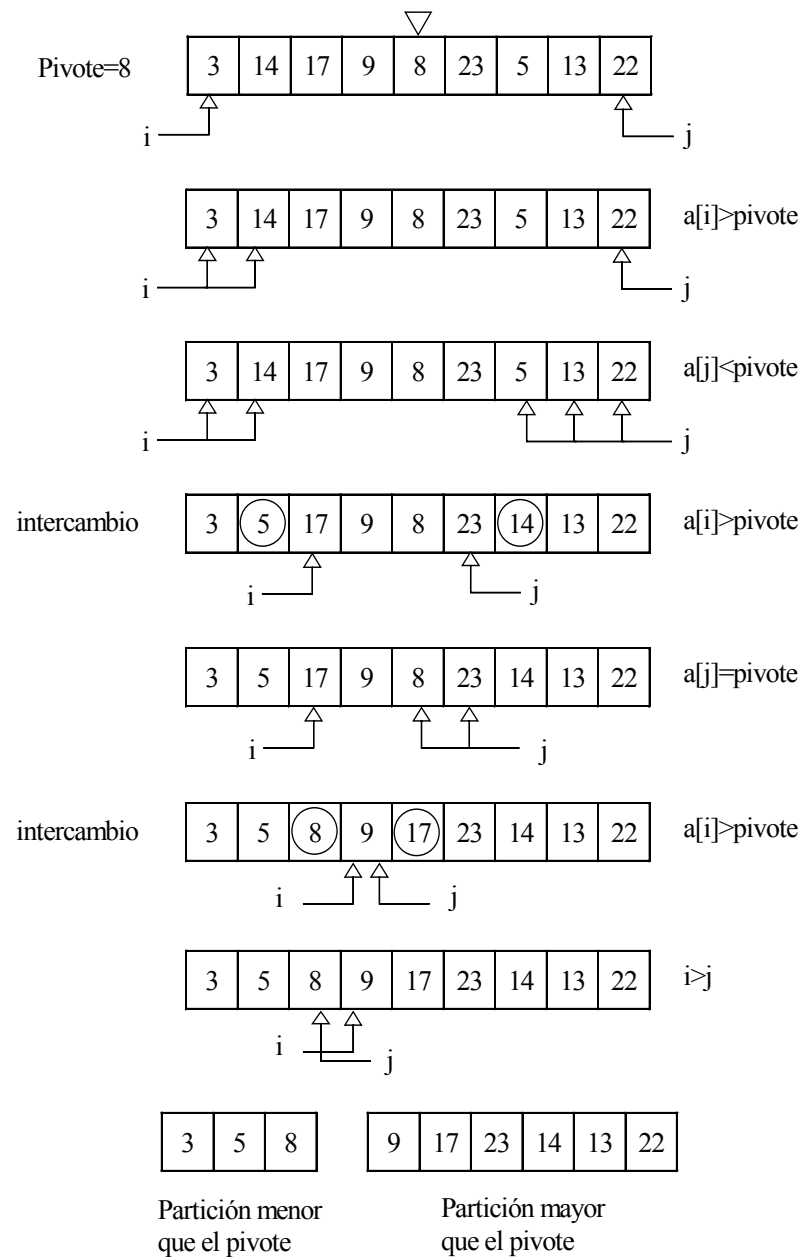


Tras esto, se cumple que:

$$A_k: 1 \leq k < i : a_k \leq x$$

$$A_k: j < k \leq n : x \leq a_k$$

Ejemplo que muestra la obtención de la primera partición para un conjunto de datos dados



Clasificación por Partición (Clasificación Rápida).

Recurativo:

PROCEDURE Partición;

 PROCEDURE ordena (L, R :Tipo_índice);

 VAR

 i, j : Tipo_índice; (*índices a inicio y final de arreglo*)

 aux, (*auxiliar para intercambio*)

 piv : Tipo_datos; (*pivote*)

 BEGIN

 i := L; j:= R;

 piv := a[(L+R) DIV 2]; (*pivote*)

 REPEAT

 WHILE a[i] < piv DO i := i+1 END;

 WHILE piv < a[j] DO j := j-1 END;

 IF i <= j THEN

 aux:= a[i]; a[i]:= a[j]; a[j]: = aux;

 (*intercambio*)

 i := i+1;

 j := j-1;

 END

 UNTIL i > j;

 (*recursividad para las 2 particiones formadas*)

 IF L < j THEN ordena(L,j) END;

 IF i < R THEN ordena(i,R) END;

 END ordena;

BEGIN ordena (1,n);

END Partición;

Iterativo:

```
PROCEDURE NonRecursiveQuickSort;
  CONST M=12;
  VAR i,j,L,R: index; w,x:item; s: [0..M];
  stack: ARRAY[1..M] OF RECORD L,R:index END;

BEGIN
  s:=1; stack[1].L:=1;stack[s].R:=n;
  REPEAT (*tomar petición de la cima de la pila*)
    L:= stack[s].L; R:= stack[s].R; s:=s-1;
    REPEAT (*partición a[L].. a[R]*)
      i:= L; j= R; x:= a[(L+R) DIV 2];
      REPEAT
        WHILE a[i]<x DO i:=i+1 END;
        WHILE x<a[j] DO j:=j-1 END;
        IF i<=j THEN
          w:=a[i];a[i]:= a[j];[j]:=w; i:=i+1;j:=j-1
        END
      UNTIL i>j
    IF i <R THEN (*apilar partición de la derecha*)
      s:=s+1; stack[s].L:=i; stack[s].R:=R
    END
    R:=j (*Ahora L y R delimitan la partición de la izquierda*)
  UNTIL L>=R
UNTIL s=0
END NonRecursiveQuickSort
```

Análisis:

- El coste vendrá dado por dos llamadas recursivas más el tiempo que transcurre en la partición (proporcional a n):

$$T(n) = T(i) + T(n - i) + cn \quad T(1) = 0$$

donde i es el nº de elementos de la partición de los menores y “c” una constante de proporcionalidad.

- **Peor caso.** El pivote es un extremo (el menor o el mayor elemento) en todas las particiones. Si se supone que en todas es el menor (i=1):

$$T(n) = T(n - 1) + cn$$

Aplicando la recursividad:

$$T(n-1) = T(n - 2) + c(n-1)$$

$$T(n-2) = T(n - 3) + c(n-2)$$

...

$$T(2) = T(1) + c(2)$$

$$T(n) = \cancel{T(1)} + c \text{ SUM}(i=2:n) i = O[n^2]$$

0

- **Mejor caso.** El pivote siempre queda en el centro:

$$T(n) = 2T(n/2) + cn$$

Se divide ambos miembros por n

$$T(n)/n = T(n/2)/(n/2) + c$$

Aplicando la recursividad

$$T(n/2)/(n/2) = T(n/4)/(n/4) + c$$

$$T(n/4)/(n/4) = T(n/8)/(n/8) + c$$

...

$$T(2)/2 = T(1)/1 + c$$

$$T(n)/n = \underset{0}{\cancel{T(1)/1}} + [\text{SUM}(i=1:\log_2 n) c] = c \log_2 n$$

Es decir:

$$T(n) = c n \log_2 n = O[n \log_2 n]$$

Caso promedio. Todos los tamaños de la partición con elementos menores al pivote son equiprobables ($p = 1/n$):

El promedio de $T(i)$, y por tanto de $T(n-i)$ es

$$\begin{aligned} T(i)_{\text{med}} &= T(0)/n + T(1)/n + \dots + T(n-1)/n = \\ &= (1/n) [\text{SUM}(j=0:n-1) T(j)] \end{aligned}$$

Aplicando la ecuación general:

$$T(n) = (2/n) [\text{SUM}(j=0:n-1) T(j)] + c n$$

Multiplicando por n:

$$n T(n) = 2 [\text{SUM}(j=0:n-1)T(j)] + c n^2$$

Aplicando para n-1:

$$(n-1) T(n-1) = 2 [\text{SUM}(j=0:n-2)T(j)] + c (n-1)^2$$

Restando:

$$n T(n) - (n-1) T(n-1) = 2 T(n-1) + 2 c n - \cancel{c}$$

$$n T(n) = (n+1) T(n-1) + 2 c n$$

Dividiendo por n(n+1) y aplicando recursividad:

$$T(n)/(n+1) = T(n-1)/n + 2c/(n+1)$$

$$T(n-1)/n = T(n-2)/(n-1) + 2c/n$$

$$T(n-2)/(n-1) = T(n-3)/(n-1) + 2c/(n-1)$$

$$\dots$$
$$T(2)/3 = T(1)/2 + 2c/3$$

$$T(n)/(n+1) = T(1)/2 + 2c [\text{SUM}(i=3:n+1) 1/i]$$

Como $[\text{SUM}(i=3:n+1) 1/i] \approx \ln(n+1) + \gamma - 3/2$

$$T(n) = O [n \ln n]$$

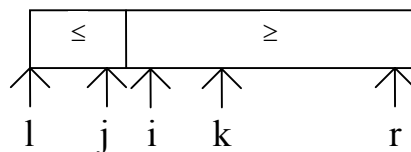
Cálculo del k-ésimo mayor elemento

Algoritmo: Realizar una operación de partición con $L=1$ y $R=n$ y a_k seleccionado como el valor divisor de cada partición. Los valores de i y j resultantes son tales que:

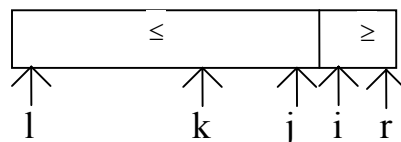
1. $a_h < x$ para toda $h < i$
2. $a_h > x$ para toda $h > j$
3. $i > j$

Hay 3 casos posibles:

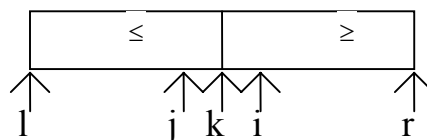
1. Límite de divisor x demasiado pequeño (Repetir proceso de partición con $a_i \dots a_R$).



2. Límite demasiado grande (Repetir proceso de partición con $a_L \dots a_j$).



3. $j < k < i$: Límite correcto .



Se repite el proceso de partición hasta llegar al caso 3

Algoritmo:(Obtención del k-ésimo elemento más grande)

```
PROCEDURE Encontrar(k:INTEGER);
    VAR L, R, i, j: integer; w, x: Tipo_datos;
BEGIN
    L := 1; R := n;
    WHILE L < R DO
        x := a[k]; i := L; j := R;
        REPEAT
            WHILE a[i] < x DO i := i+1 END;
            WHILE x < a[j] DO j := j-1 END;
            IF i <= j THEN
                w := a[i]; a[i] := a[j]; a[j] := w;
                i := i+1; j := j-1
            END
        UNTIL i > j;
        IF j < k THEN L := i END;
        IF k < i THEN R := j END
    END
    x:=a[k];
END
```

Obsérvese que el cálculo de la mediana consiste en calcular el k-ésimo elemento mayor, con $k=n/2$.

Análisis:

Caso promedio⁽¹⁾: $C = n + n/2 + n/4 + \dots + 1 = 2n$

Caso peor⁽²⁾: $C = n + (n-1) + (n-2) + \dots + 2 = O[n^2]$

⁽¹⁾ Cada división divide a la mitad el tamaño de la partición.

⁽²⁾ Cada división deja un elemento en una partición y el resto en la otra.

CONCLUSIONES

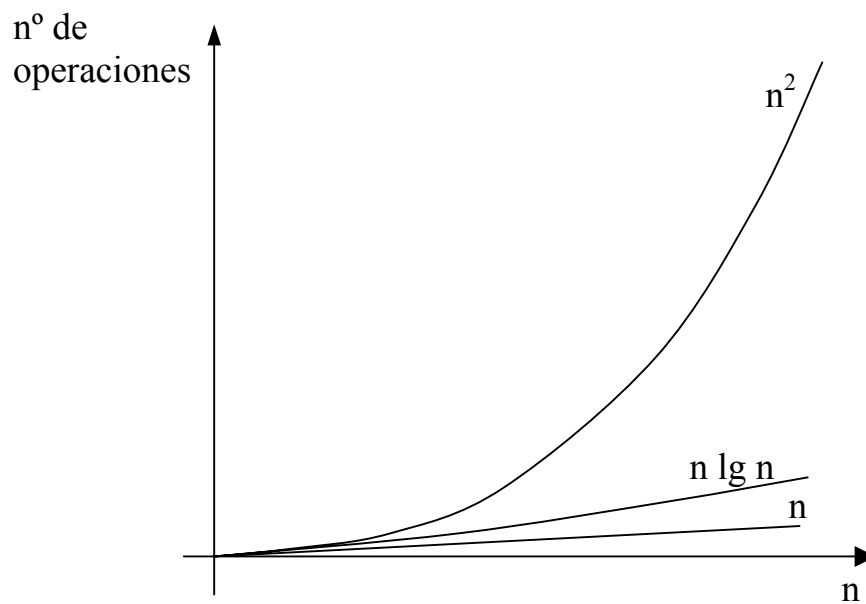
Clasificación rápida (Quicksort): prestaciones altas en el caso promedio y mejor de los casos

Clasificación por montón: Supera las prestaciones de la clasificación rápida cuando abundan las llaves iguales. Este caso constituye el peor caso de la clasificación rápida.

Clasificación Shell: es la que ofrece menos ventajas y, además, depende fuertemente de la sucesión de incrementos elegida.

Comparación entre métodos directos y avanzados

Los métodos avanzados requieren más operaciones que los directos: los primeros sólo recomendables para n grandes y los segundos para n pequeños.



n	$n \lg_2 n$	n^2
10	34	100
50	283	2.500
1.000	≈ 10.000	1 millón
10^6	≈ 20 millones	1 billón