

ORDENACIÓN

Cosas a tener en cuenta

- ¿Cuántos elementos vamos a ordenar?
- ¿Hay claves duplicadas?
- ¿Conocemos las distribuciones de las claves?
- ¿Qué sabemos de los datos?
- ¿Son las claves muy largas o difícil de comparar?
- ¿Es el rango de claves posibles muy pequeño?

- La clasificación puede dividirse en INTERNA o EXTERNA
- La estabilidad de los métodos es la capacidad de modificar lo menos posible el orden de los elementos.
- Los algoritmos más simples requieren tiempos $O(n^2)$, otros $O(n \cdot \log n)$, y algunos para datos especiales $O(n)$.

- Inserción Directa

- Preferido para tamaños pequeños

- Shellsort

- Preferido para tamaños medianos

CLASIFICACIÓN INTERNA

- Los valores de sus claves forman una secuencia no decreciente.
 - Usa con cuidado el almacenamiento.
 - C = números de comparaciones.
M = números de movimientos.
 - Los buenos algoritmos requieren $O(n^* \log n)$ comparaciones, los más sencillos $O(n^2)$
 - Métodos DIRECTOS → Mas fáciles
mas cortos
mejor para conjuntos chicos
- INDIRECTOS → Mas complejos

Tipos:

- Por inserción - Enumeración a cuenta
- Por intercambio - Por selección - Esp.

INSERCIÓN

COMIENZO

(1) Desde $i=2$ hasta N
hacer

(2) $Aux \leftarrow V[i]$

(3) $j = i - 1$

(4) Mientras $j > 0$ y
 $Aux.clave < V[j].clave$

hacer

(5) $V[j+1] \leftarrow V[j]$

(6) $j \leftarrow j - 1$

(7) fin mientras

(8) $V[j+1] \leftarrow Aux$

(9) Fin desde

FIN

COMIENZO

(1) Desde exactamente $N-1$ veces

(2) $O(1)$

(3) $O(1)$

(4) mientras: peor caso $N-i$
veces, mejor caso 1 vez

(5) $O(1)$

(6) $O(1)$

(7) fin mientras

(8) $O(1)$

(9) fin desde

FIN

Mejor caso $O(N)$ - ya ordenados

Peor caso promedio $O(N^2)$

EJEMPLO:

41	55	12	42	94	18	6	67	i
41	55	12	42	94	18	6	67	2
12	94	55	41	94	18	6	67	3
12	42	44	55	94	18	6	67	4
12	42	44	55	94				5
12	18	42	44	55	94			6
6	12	18	42	44	55	94		7
6	12	18	42	44	55	67	94	8

- ¿Cuál es la estructura más adecuada para la inserción directa?

LISTA ENCADENADA SIMPLE

COM

- (1) Mientras no (`ListaDeEntrada.Vacia`) hacer $O(N)$
 - (2) `ElementoAMover` \leftarrow `ListaDeEntrada.Primero` $O(1)$
 - (3) `ListaDeEntrada.EliminarPrimero` $O(1)$
 - (4) `ListaDeSalida.InsertarOrdenado(ElementoAMover)` $O(N)$
- (5) Fin mientras
- (6) Devolver `ListaDeSalida` $O(1)$

FIN

$O(N^2)$: peor y caso promedio

$O(N)$: mejor caso, la lista estaba ordenada al revés.

Ejemplo de LISTAS MULTIPLES

- M = 4
- Rangos: 0-249, 250-499, 500-749, 750-999

Conjunto original

503-087-512-061-908-170-897-275-653-426-
154-509-612-677-765-703

Resultados Final:

Lista 1: 061, 087, 154, 170

Lista 2: 275, 426

Lista 3: 503, 509, 512, 612, 653, 677, 703

Lista 4: 765, 897, 908

Comparaciones a N^2/M

SHELL SORT

COM

Desde $k=t$ hasta 1 hacer

$$h \leftarrow h_c[k]$$

Desde $i=1+h$ hasta N hacer

$$Aux \leftarrow V[i]$$

$$j = i - h$$

mientras $j > 0$ y $Aux < V[j]$. clare hacer

$$V[j+h] \leftarrow V[j]$$

$$j \leftarrow j - h$$

fin mientras

$$V[j+h] \leftarrow Aux$$

fin desde

fin desde

FIN

EJEMPLO

Primeros con incremento 5

81 99 11 96 12 35 17 95 28 58 41 75 15

35 17 11 28 12 41 75 15 96 58 81 94 95

Ultimos con incremento 3

35 17 11 28 12 41 75 15 96 58 81 94 95

28 12 11 35 15 41 58 17 94 75 81 96 95

Por ultimo con incremento 1

11 12 15 17 28 35 41 58 71 81 94 95 96



Array ordenado

METODOS DE INTERCAMBIO BURBUJA

- (1) Desde $i=1$ hasta $N-1$ hacer
 - (2) Desde $j=N$ hasta $i+1$ hacer
 - (3) si $V[j].clave < V[j-1].clave$ entonces
 (A) Intercambia ($V[j]$, $V[j-1]$)
 Fin Si
 - FIN Desde
 - FIN Desde
- $O(N^2)$

EJ EMPLO

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

QUICKSORT

• Su orden es $n^* \log n$

256 - 458 - 365 - 298 - 043 - 648
L R

648 - 458 - 365 - 298 - 043 - 256

256 - 458 - 365 - 298 - 043 - 648
L R

256 - 043 - 365 - 298 - 458 - 648
L R

256 - 043 - 298 - 365 - 458 - 648
R L

El pivote era 365

L=3



Pivot = 256

$$256 - 043 - 298$$

L R

$$298 - 043 - 256$$

L R

$$043 - \underline{298} - 256$$

L R

$$\underline{043} - \underline{298} - 256$$

R L

Pivot = 298

043

$$298 - \underline{256}$$

L R

$$\underline{256} - \underline{298}$$

R L

↓ Pivot = 458

$$- 365 - 458 - \frac{648}{R}$$

$$\underline{648} - 458 - \frac{365}{R}$$

$$365 - \frac{458}{L} - \frac{648}{R}$$

$$365 - \frac{458}{LR} - 648$$

$$\underline{365} - \frac{458}{L} - 648$$

Pivot = 648

365

$$458 - 648$$

L R

$$648 - 458$$

$$458 - 648$$

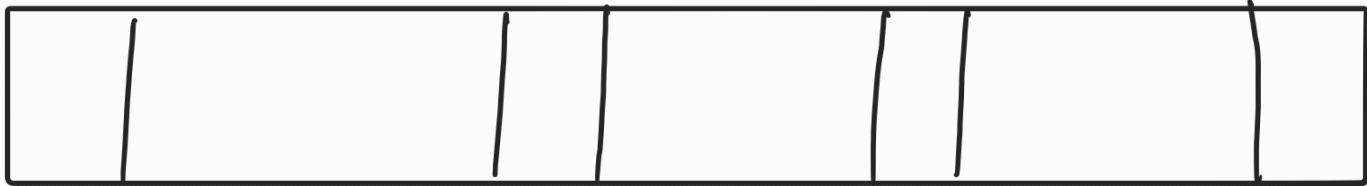
L R

$$\underline{458} - \frac{648}{L}$$

Final

043-256-298-365-458-648

QUICK SORT



i L R J

| claves < pivot | claves mezcladas | claves >= piv. |

Pasos:

- Rastrear
- Mover L a la derecha en los registros cuyas claves sean menores que el pivote.
Mover R a la izquierda en los claves \geq que el pivote.
- Probar
- Si $L > R$ ($L=R+1$) entonces el conjunto ya se ha dividido, salir.
- Desviar
- Si $L < R$, intercambiar $V[L]$ con $V[R]$.
Ahora $V[L]$. clave es $<$ que el pivote
y $V[R]$. clave es \geq

quicksort(i, j)

(1) Pivote \leftarrow ObtenerClavePivote(i, j)

(2) Si existe un Pivote entonces

(3) permutar $V[i] \dots V[j]$ de forma que para alguna k tal que $i+1 \leq k \leq j$, $V[i].clave \dots V[k-1].clave < Pivote$ y $V[k].clave \dots V[j].clave \geq Pivote$

(5) quicksort(i, k-1)

(6) quicksort(k, j)

FIN SI

function particion(i,j:integer; pnote:Tipotlare
:integer;

var

L,R : de tipo enteros;

com

L := i;

R := j;

Repetir

Intercambia (V[L], V[R]);

mientras V[L].clave < pivote hacer

L := L + 1;

Fin mientras

mientras V[R].clave >= pivote hacer

R := R - 1

fin mientras

Hasta que L > R

Devolver L;

end

Métodos Principal

quicksort (i, j : tipo entero);

pivote : TipoClave;

IndicePivote : tipo entero;

K : tipo entero;

COM

 IndicePivote \leftarrow EncuentraPivote (i, j)

 Si IndicePivote $\neq 0$ entonces

 pivote \leftarrow V[IndicePivote]. clave;

 K \leftarrow partición (i, j, pivote);

 quicksort (i, K-1);

 quicksort (K, j);

 FIN SI

FIN

Analisis del tiempo de ejecución

Mejor caso y caso prom = $O(n^* \log n)$

Peor caso $O(n^2)$

→ Cuando es un extremo

→ Cuando el pivote es la mediana

- Para partición el orden de cada llamada será $O(j-i)$
- Se ejecutarán $2N-1$ llamadas al algoritmo.

Mejor Ejemplos

17, 12, 2, 77, 25, 44, 88

Utilizamos el pivote el mayor de los dos primeros = 17

, L, 12, 2, 77, 25, 44, 88

12 < 17 ✓ entonces L se mueve 1

12, L, 2, 77, 25, 44, 88

2 < 17 ✓

12, 2, L, 77, 25, 44, 88

17 > 77 ✗ salimos del mientras L vamos con R

12, 2, , 77, 25, 44, 88
R N R N R N R

88 > 17 ✓ R - 1 2 > 17 ✗

44 > 17 ✓ R - 1

25 > 17 ✓ R - 1

77 > 17 ✓ R - 1

12, 2, 17, 77, 25, 44, 88

↓
pivot = 12

↓
pivot = 77

L 25 < 77 ✓ 44 < 77 ✓

88 < 77 ✗

R 88 > 77 ✓ 44 > 77 ✗

2 < 12 ✓

2 ✓, 12 ✓, 17 ✓, 25, 44, 77 ✓, 88 ✓

↓
pivot = 44

, , 74

L 44 < 25 ✗

R 25 > 44 ✗

2, 12, 17, 25, 44, 77, 88

ORDENADO

Métodos	Orden	Info
Inserción Directa	Peor = N^2 Mejor = N	Preferidos para tamaños pequeños
Shell Sort	Peor = $N^{1.6} \sim N \log N$ Mejor = N	Tamaños medianos
Burbuja	N^2	Peor método de clasificación
Quick Sort	Mejor = $N \log N$ Peor = N^2	Menor tiempo de ej en el mejor caso
Selección Directa	N^2	N^2 comparaciones N movimientos
Heap Sort	$N \log N$	Idea es mejorar las comparaciones para obtener el menor de los elementos

SELECCION DIRECTA

Desde $i=1$ hasta $N-1$ hacer

IndiceDelMenor $\leftarrow i$

ClaveMenor $\leftarrow V[i].clave$

Desde $j=i+1$ hasta N hacer

Si $V[j].clave < ClaveMenor$ entonces

IndiceDelMenor $\leftarrow j$

ClaveDelMenor $\leftarrow V[j].clave$

FIN SI

Fin Desde

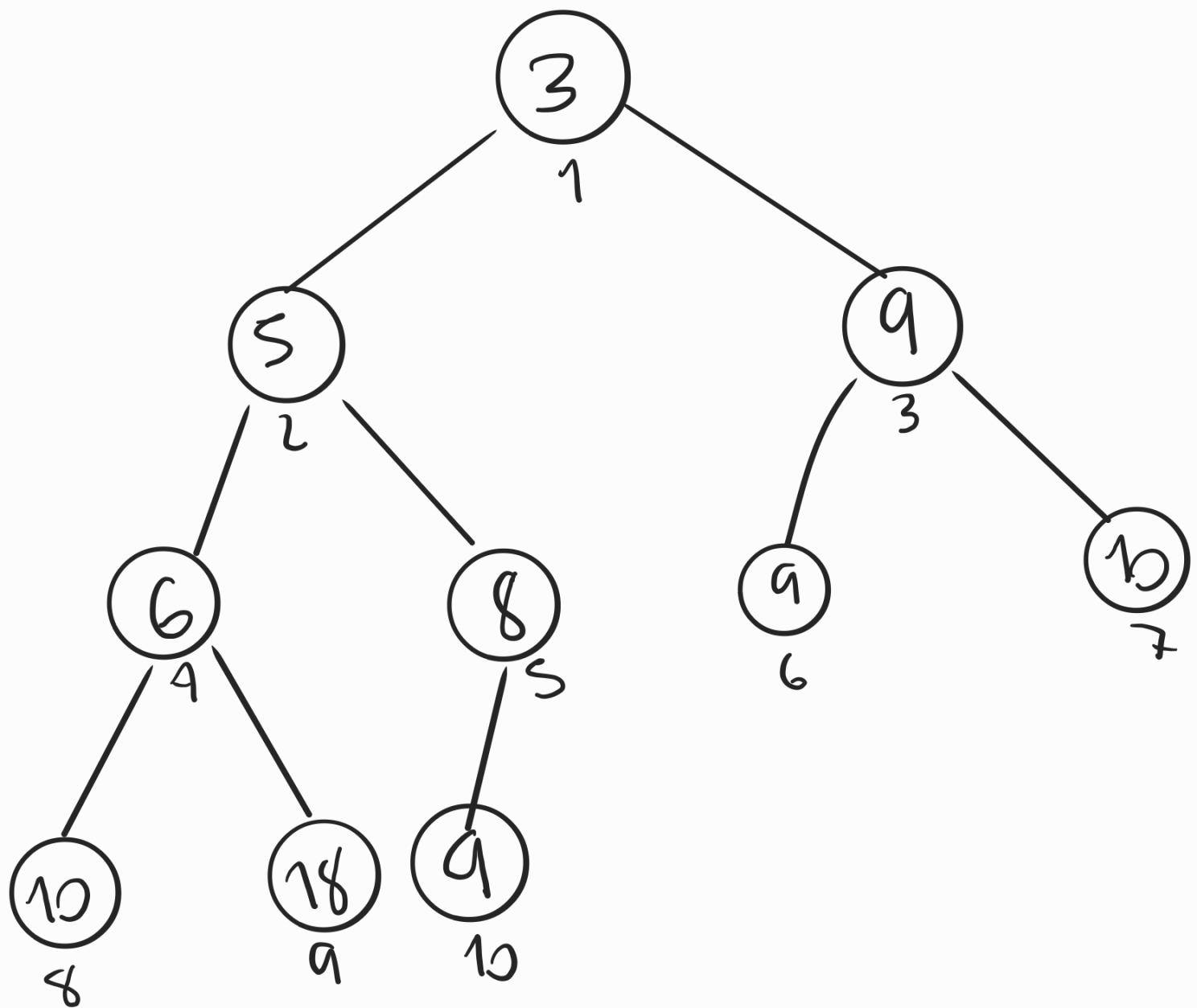
Intercambia ($V[i]$, $V[IndiceDelMenor]$)

FIN Desde

SELECCION DIRECTA

	1	2	3	4	5	6	7	8
1er	223	784	376	285	015	440	666	007
1	007	784	376	285	015	440	666	223
2	007	015	376	285	784	440	666	223
3	007	015	223	285	784	440	666	376
4	007	015	223	285	784	440	666	376
5	007	015	223	285	376	440	666	784
6	007	015	223	285	376	440	666	784
7	007	015	223	285	376	440	666	784
8	007	015	223	285	376	440	666	784

HEAPSORT



Armar el heap a partir de arbol

1	2	3	4	5	6	7	8	9	10
3	5	9	6	8	9	10	10	18	9

Armar el heap a partir de vector

1	2	3	4	5	6	7	8	9	10	11	12
5	6	10	18	8	9	9	10	3	9	5	
5	6	10	3	8	9	9	10	18	9	1	
5	6	9	3	8	10	9	10	18	9	3	
5	3	9	6	8	10	9	10	18	9	2	
3	5	9	6	8	10	9	10	18	9	1	



heap armado

ahora hay que ordenarlo

5	6	9	9	8	10	9	16	18	3
6	8	9	9	18	10	9	10	5	3
8	9	9	10	18	10	9	6	5	3
9	9	9	10	18	10	8	6	5	3
9	10	9	10	18	9	8	6	5	3
9	10	18	10	9	9	8	6	5	3
10	10	18	9	9	9	8	6	5	3
10	18	10	9	9	9	8	6	5	3
18	10	10	9	9	9	8	6	5	3
18	10	10	9	9	9	8	6	5	3

Desplazamiento Elemento (Primer, Ultimo: tipo entero)

Comenzar

Actual \leftarrow Primer

mientras Actual \leftarrow (ultimo/2) hacer

Si ultimo = 2 * Actual entonces

Si V[Actual].clave > V[2 * Actual].clave entonces

Intercambia(V[actual], V[2 * actual])

fin Si

Actual \leftarrow Ultimo

Sino

Menor \leftarrow MenorHip(2 * Actual, 2 * Actual + 1)

Si V[actual].clave > V[menor].clave entonces

Intercambia(V[Actual], V[Menor])

Actual \leftarrow Menor

Sino

Actual \leftarrow Ultimo

FIN SI

FIN SI

FIN MIENTRAS

Algoritmos de HeapSort

Contenido

Desde $i=N$ dW 2 hasta 1 hacer

DesplazarElementos(i, N);

fin Desde

Desde $i=N$ hasta 2 hacer

Intercambia ($V[1], V[i]$)

DesplazarElementos(1, $i-1$);

Fin Desde

FIN

$$O(N^* \log N)$$

