APUNTES

Estructura de Datos y Algoritmos

Curso 2003-2004

www.utomde.com

Pablo Perales Diaz

Algoritmos de clasificación en memoria principal

Estabilidad: decimos que un algoritmo es estable cuando la posición relativa de los elementos de igual valor no varían en el transcurso del algoritmo

Naturalidad: Se dice que el comportamiento de un algoritmo es natural si su mejor caso (en el que hacemos menos comparaciones y movimientos es cuando el array esta inicialmente ordenado

2.1 Inserción

La idea de este método de clasificación es la de insertar el elemento no ordenado en la posición que le corresponde dentro del conjunto de elementos ya ordenados.

2.1.1 Inserción directa

Se basa en coger el primer elemento de la parte desordenada y buscar la posición que le corresponda dentro de la parte ordenada.

Paso1: Partimos de un Array aleatoriamente ordenado y marcamos su primer elemento como parte ordenada. El resto será la parte desordenada.

Paso2: Cogemos el primer elemento de la parte no ordenada y lo almacenamos en una variable centinela (a[0])

Paso3: Comparamos empezando por el final de la parte ordenada hasta que encontramos un elemento menor.

Paso4: Movemos una posición a la Dcha todos los elementos que han resultado mayores que el que queremos insertar e instalamos el centinela en la posición encontrada.

CÓDIGO

```
TYPE tarray:Array[0..n] OF INTEGER

PROCEDURE Inserción_Directa(Var a:tarray)
Var i,j
        FOR i:= 2 TO n DO
                a[0]:= a[i];
                j:=I;
                WHILE a[0]<a[j-1] DO
                        a[j]:=a[j-1];
                        DEC(j);
                END
                a[j]:=a[0]
        END
END Inserción_directa;
```

ANÁLISIS DE COSTES

Comparaciones

* Caso mejor: El array está totalmente ordenado

Para cada valor de **i** solo hará la primera comparación y no entrara en el bucle así el número de comparaciones será:

Para i=2 ---------- 1
i=3 ---------- 1
.....
i=n ---------- 1

Luego

C_{mejor} = n-1

* Caso Peor: Array inicialmente totalmente desordenado.

Para cada valor de i tendrá que hacer todas las comparaciones hasta el final del array esto es:

Para i=2 ---------- 2
i=3 ---------- 3
.....
i=n ---------- n

Luego

[redacted]

* Caso promedio: El vector está aleatoriamente ordenado. Como la probabilidad de que un nº sea mayor que otro es de $\frac{1}{2}$ la probabilidad de que la comparación tenga como resultado true o False también será de $\frac{1}{2}$ luego

[redacted]

Movimientos

* Caso Mejor: En este caso el Array estará totalmente ordenado y no entrará nunca en el bucle con lo que solo se ejecutarán desde i=2 hasta n las 2 instrucciones de asignación externas al bucle luego

$$M_{mejor} = 2(n-1)$$

* Caso Peor: Además de las 2(n-1) asignaciones externas del bucle tendremos las internas que serán:

Para i=2 ---------- 1
i=3 ---------- 2
.....
i=n ----------[redacted]--------- n-1

Luego

[redacted]

Si le sumamos las 2(n-1) externas tendremos

$$M_{peor} = \frac{n^2 + 3n - 4}{2}$$

* Caso Promedio: Las llaves están aleatoriamente ordenadas.

Teniendo en cuenta que cada vez que se realice una comparación (excepto la del centinela) se realiza un movimiento en el *WHILE*, el número promedio de en el *WHILE* será:

y sumando los 2(n-1) externos:

$$M_{prom} = M_{prom(WHILE)} + 2(n-1) = \frac{n^2 + 9n - 10}{4}$$

2.1.2 Inserción binaria

Es una mejora del anterior.

Es un método de inserción que difiere de la inserción directa simplemente en la búsqueda de la posición de la inserción.

Paso 1 : Hallamos el elemnto central del area comprendida por la parte ordenada mas la posición del elemento a insertar.

Paso2: Comparamos el elemento central con el elemento que queremos insertar. Si dicho elemento central es meno o igual nos quedamos con la parte superior (SIN INCLUIR EL ELEMENTO CENTRAL).
En caso contrario nos quedamos con la mitad inferior, INCLUYNDO el elemento central.

Paso3: Repetimos el mismo proceso sobre el área con la que nos hemos quedado hasta que dicha área sea nula.

Paso4: Cuando el área sea nula tendremos la posición de inserción.

CÓDIGO

```
PROCEDURE Inserción_Binaria (VAR a:tarray)
VAR   i,j,L,R,central:Tipo Indice;
      Centinela:TipoDatos;
BEGIN
      FOR i:= 2 TO n DO
            L:=1;
            R:=i;
            Centinela:=a[i];
            WHILE L<R DO
                  Central:= (L+R) DIV 2;
                  IF a[Central]<= Centinela THEN
                        L:=Central+1;
                  ELSE
                        R:=Central;
                  END;
            END;
            FOR j:= I TO R+1 BY -1 DO
                  a[j]:=a[j-1];
            END;
            a[R]:=centinela;
      END;
END Inserción_Binaria;
```

ANÁLISIS DE COSTES

En este algoritmo los movimientos van a ser básicamente los mismos que en el directo ya que la variación tiene que ver con la búsqueda de la posición y por tanto o que variarán serán las comparaciones.

Asimismo el nº de comparaciones no dependerá del orden inicial

Para cada valor de i se realizarán tantas comparaciones como divisiones. Para calcular las veces que se divide el array sabemos que:

$n = 2^N$

Donde n= longitud del array

y

N= Numero de veces que se divide por 2

Luego N= log n

Tendremos entonces que:

Para resolver esta serie la aproximaremos a la integral

 (1)

Por análisis sabemos que

y también sabemos que si

 (3)

Sustituyendo (3) en (1)

 y aplicando (2)

$$C = \frac{1}{Ln2} x[Lnx - 1]_1^n$$

estaremos por tanto ante un orden n*logn

NOTA: El nº de comparaciones ha sido calculado considerando que no depende del orden inicial de los datos. Esto no es del todo cierto debido al carácter truncador de la división entera que desprecia el resto. Realmente el algoritmo tiene un comportamiento no natural.

2.2.1 Selección Directa

Se trata de elegir el menor elemento de entre todos los de la parte no ordenada. Este elemento se intercambiara por el primero de dicha parte.

Paso1: Partimos de un Array en el que todo será parte no ordenada.

Paso2: Cogemos el primer elemento de la parte no ordenada y lo guardamos en una variable llamada menor. También almacenaremos su posición en posMenor.

Paso3: Recorremos toda la parte no ordenada comparando el menor con cada elemento. Si encontramos uno menor almacenaremos dicho elemento y su posición.

Paso4: Intercambiamos el elemento almacenado como menor con el 1º de la parte no ordenada y repetimos desde el paso 2 con la parte no ordenada una posición más a la derecha.

CÓDIGO

```
PROCEDURE Selección_Directa(VAR a:tarray)
VAR   i,j:TipoIndice;
      Menor:TipoDatos;
BEGIN
      FOR i:=1 TO n-1 DO
            Menor:=a[i];
            posMenor:=i;
            FOR j:= i+1 TO n DO
                  IF menor>a[j] THEN
                        Menor:=a[j];
                        Posmenor:=j;
                  END
            END;
            a[posmenor]:=a[i];
            a[i]:=menor;
      END;
END Selección_directa
```

ANÁLISIS DE COSTES

Comparaciones

Al estar dentro de bucles FOR no van a depender del orden inicial.

Caso General

Para i=1 ---------- n-1
i=2 ---------- n-2
.....
i=n-1 ---------- 1

$$C = \sum_{i=2}^{n}(n-i) = \frac{1^{o}\ \text{elemento de la serie} + \text{último element o}}{2} * \text{nº de iteraciones} = \frac{1+(n-1)}{2}(n-1) = \frac{n^2-n}{2}$$

Movimientos

* Caso Mejor

Nunca se cumple la condición del IF por lo que tendremos 3 instrucciones de asignación que se repiten n-1 veces por tanto

* Caso Peor

En este caso además de los 3(n-1) movimientos internos al bucle FOR mas externo se realizara siempre el interno a la condición IF.

6	5	4	3	2	1

Si seguimos el algoritmo veríamos que se producen 5 movimientos

1	5	4	3	2	6

La siguiente vuelta 3 movimientos

1	2	4	3	5	6

Y por ultimo un movimiento

Así podemos generalizar que los movimientos serán

(n-1) + (n-3) + ...+ 1 por tanto

$$M_{peor} = 3(n-1) + \frac{1^{o}\ \text{elemento de la serie} + \text{último element o}}{2} * \text{nº de iteraciones} = \frac{\text{n-1+1}}{2}\frac{n}{2} = \frac{n^2}{4} + 3(n-1) = \frac{n^2+12n-12}{4}$$

* Caso Promedio

Al estar los datos ordenados aleatoriamente habrá que utilizar cálculos de probabilidad

La probabilidad de que un elemento sea menor que otro es de $\frac{1}{2}$, que sea el menor de 3 1/3 y la de que sea el menor de i elementos será 1/i

Por tanto

[redacted] (1)

Siendo Hi el i-esimo nº Armónico

Y teniendo en cuenta que despreciando los términos del orden [redacted]

[redacted]

Y sustituyendo en (1)

Realmente estos serían los movimientos de un pase. Para n Pases

Finalmente aproximando a integrales

$$M_{prom} = 3(n-1) + \sum_{i=1}^{n}(\ln i + \gamma - 1) = 3(n-1) + n(\gamma - 1) + \int_{i}^{n} \ln x dx = 3(n-1) + n(\gamma - 1) + \left[x \ln x - 1\right]_{i}^{n}$$

Simplificando y debido a que es una aproximación expresándole en orden asintótico será O(nlogn)

Conclusiones: Este resultado permite concluir que el método de selección directa ya que su numero de movimientos promedio es del orden nlogn pero si se compara con el de inserción binaria tenemos que mientras para este el nº de comparaciones es logarítmico mientras que en la selección directa es cuadrático , dándose la situación contraria para los movimientos. En general es mayor el coste de un movimiento que el de una comparación. Por tanto como norma general la selección será la forma a elegir, aunque la inserción es algo mas rápida cuando la clasificación es predominante al principio

2.3 ALGORITMOS DE INTERCAMBIO

2.3.1 Algoritmo de la Burbuja o Intercambio directo

Este algoritmo consistirá en recorrer el array comparando los elementos que están en posiciones contiguas. Si estos están desordenados se intercambian.

Paso1: Partimos de un array inicialmente desordenado en el que la parte desordenada ocupara su totalidad.

Paso2: Empezando SIEMPRE desde la primera posición del Array vamos subiendo hacia posiciones posteriores comparando los elementos consecutivos dos a dos

Paso 3: Si dichos elementos están desordenados se intercambian consiguiendo que al final de un recorrido nos quede el elemento de mayor valor del parte desordenada en la ultima posición.

Paso4: Incrementamos la parte ordenada una posición a la izquierda desde el final y repetimos el mismo proceso desde el paso 2 hasta que la parte ordenada alcance la primera posición.

CÓDIGO

```
PROCEDURE Burbuja (Var a:tarray)
        VAR i,j: TipoIndice
        Aux: Tipodatos
BEGIN
        FOR i:= 1 TO n-1 DO
                FOR j:= 1 TO n-I DO
                        If a[j]>a[j+1] THEN
                                Aux:= a[j+1]
                                A[j+1]:=a[j]
                                A[j]:= aux
                        END
                END
        END
END Burbuja;
```

ANÁLISIS DE COSTES

Comparaciones:

Al estar el If dentro de dos bucles For su nº va a ser independiente del orden inicial

* Caso general:

Para i=2 ---------- n-1
i=3 ---------- n-2
.....
i=n-1 ---------- 1

Luego

$$C_{comp.gral} = \sum_{i=1}^{n-1}(n-i) = \frac{1^{\circ}\text{ elemento de la serie} + \text{último elemento}}{2} * \text{ nº de iteraciones} = \frac{(n-1)+1}{2}(n-1) = \frac{n^2-n}{2}$$

Movimientos:

Caso Mejor: Al estar los movimientos dentro del array en caso de que no entre nunca no se va a producir ningún movimiento.

Caso Peor:

Para i=1 ---------- 3(n-1)
i=3 ---------- 3(n-2)
.....
i=n-1 ---------- 3

$$M_{peor} = \sum_{i=1}^{n-1}(n-i) = \frac{1^{o}\ \text{elemento de la serie} + \text{último element o}}{2} * \text{nº de iteraciones} = \frac{3(n-1)+3}{2}(n-1) = \frac{3n^2-3n}{2}$$

Caso promedio:

$$M_{prom} = \frac{3n^2-3n}{4}$$

Una posible mejora es comprobar si ya está ordenado para no seguir haciendo pasadas.

2.3.2 Sacudida o vibración (mejora de la burbuja)

Se basa en el mimo concepto de intercambio que el algoritmo de la burbuja salvo que alterna el sentido del recorrido.

Paso1: Partimos de un array inicialmente desordenado en el que la parte desordenada ocupara su totalidad.

Paso2: Recorremos el array en sentido ascendente desde la primera (L) hasta la ultima posición de la parte desordenada (R) comparando elementos consecutivos e intercambiándolos cuando sea necesario.

NOTA: En este algoritmo empezamos a recorrer desde L que no será siempre la 1ª Posición del Array

Paso 3: Una vez acabado el recorrido debemos marcar la ultima posición donde hubo intercambio para que en la pasada del sentido contrario dicha pasada nos sirva de punto de partida

Paso 4: Repetimos el mismo proceso en sentido contrario hasta que lleguemos a la posición L marcando donde se realizó el ultimo intercambio.

Paso 5: Repetimos el mismo proceso desde 2 una vez actualizada la L Hasta que l sea mayor que R.

CÓDIGO

```
PROCEDURE Vibracion (var: a tarray)
        VAR i,j,l,r,K:TipoIndice
        Aux:TipoDatos
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 Vibración
```

CONCLUSIONES

* De todos los métodos presentados el de selección directa será el mas recomendable a elegir cuando el array está aleatoriamente ordenado.

* El método de inserción binaria será el más rápido cuando el desorden se encuentre el las primeras posiciones del array

* El método de vibración será el mas eficiente cuando el array esta inicialmente ordenado.

METODOS AVANZADOS

Algoritmo de Shell o incrementos decrecientes

Se trata de ir realizando la inserción directa con todos los elementos que difieren en h_i posiciones dentro del array. Posteriormente haremos la inserción directa con elementos que difieran [redacted] posiciones y así sucesivamente hasta obtener la ordenación total

Nota: Para que este algoritmo realice una clasificación satisfactoria el ultimo nº de elementos del conjunto de incrementos decrecientes debe ser igual a 1, caso en el que se realizara la inserción directa normal.

Paso 1: Cogemos el primer elemento del conjunto de incrmentos decrecientes y vamos haciendo grupo con los elementos que difieren en tantas posiciones como nos indique dicho número.

Nota: Siempre nos quedaran tantos grupos como el elemento [redacted]

[4,2,1]

6	1	8	7	2	4	3	10	0	22
Grp1	Grp2	Grp3	Grp4	grp1	Grp2	Grp3	Grp4	grp1	Grp2

Paso 2: Realizamos la inserción directa con cada uno de los grupos. Para la implementación lo que haremos pasar el elemento que queremos insertar al centinela y compararlo con el ultimo de la parte ordenada h_i posiciones anteriores. El momento en el que encontremos un elemento menor o el principio del array pararemos y movemos todos los elementos del mismo grupo [redacted] posiciones hacia la derecha.

0	1	3	7	2	4	8	10	6	22
Grp1	Grp2	Grp3	Grp4	grp1	Grp2	Grp3	Grp4	grp1	Grp2

Paso 3: Una vez hecha la inserción directa sobre los [redacted] grupos, cogemos el siguiente elemento del conjunto de incrementos decrecientes [redacted] y volvemos al paso 1 hasta que no queden elementos por coger.

0	1	3	7	2	4	8	10	6	22
Gr 1	Gr2	Gr1	Gr2	Gr1	Gr2	Gr1	Gr2	Gr1	Gr2

Ventaja de este Método: Con respecto a la inserción directa este método este método mejora el nº de movimientos ya que antes de llegar al incremento decreciente 1 los elementos del array se han ido posicionando cerca del lugar que ocuparan al final con desplazamientos " a gran escala ".

Nota: Cuando el conjunto de incrementos decrecientes esta compuesto por elementos que son múltiplos entre si la clasificación no será tan efectiva porque ya habremos incluido en el mismo grupo a varios elementos que ya se compararon en iteraciones anteriores. Existen dos formas "mejores" de elegir los elementos:

- Hibbard se escogen $(2^n)-1$: P.EJ: 15 7 3 1. En este caso en el caso peor el coste es de [redacted] y en el promedio[redacted]
- Sedgewick propuso otras secuencias en las cuales el coste en el peor caso es [redacted]y en el promedio [redacted]

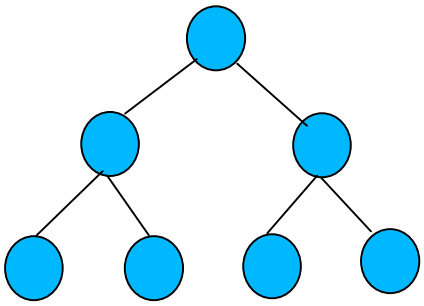
En ambos casos el caso promedio no está demostrado analíticamente

2.3.2 CLASIFICACIÓN POR MONTON

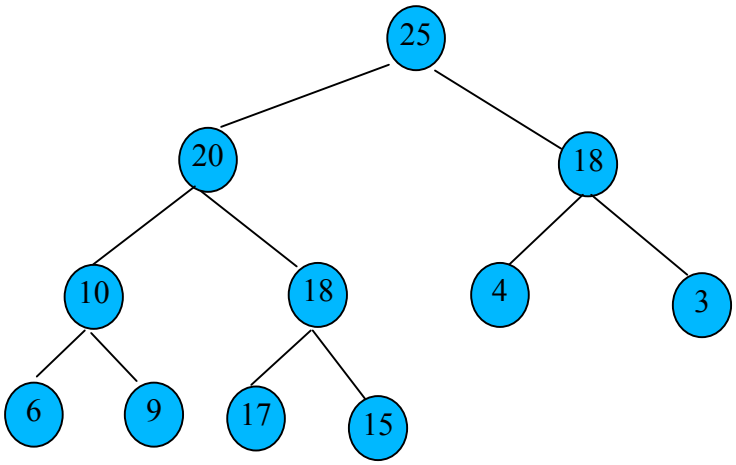
Para comprender mejor un montón veremos primero su implementación sobre un árbol, ya que sobre un arreglo es una consecuencia de este.

En un árbol, a partir de cada elemento (nodo) tenemos acceso a más de de un nodo. Dependiendo del nº de nodos tendremos árboles binarios, ternarios, etc. Los nodos que no tienen descendientes se llaman hojas.

Llamaremos un árbol completo a aquel en que todos los niveles tienen el máximo nº de nodos que pueden tener.



La propiedad de ser completo es muy exigente, para que pueda serlo tendría que tener [redacted] nodos. Hay otro árbol denominado esencialmente completo el cual es completo hasta el penúltimo nivel estando el último ordenado. P. Ej



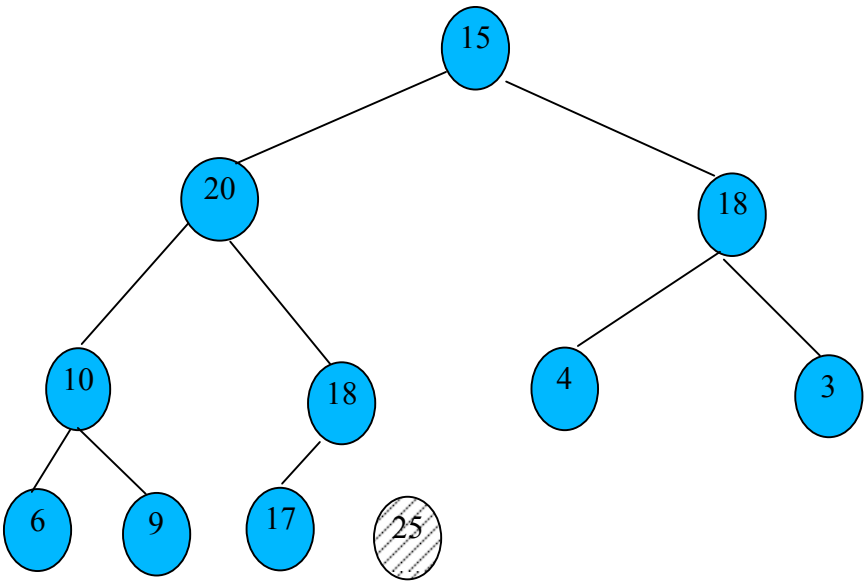
Montículo de máximos: Es un árbol binario esencialmente completo con la propiedad de que el valor de cada nodo es mayor que el de todos sus descendientes. Entre los hermanos no hay relación. Para organizarlo en un arreglo como viene en el libro, copiaremos todos los elementos del arbol por niveles y de derecha a izquierda.

L										R
25	20	18	10	18	4	3	6	9	17	15

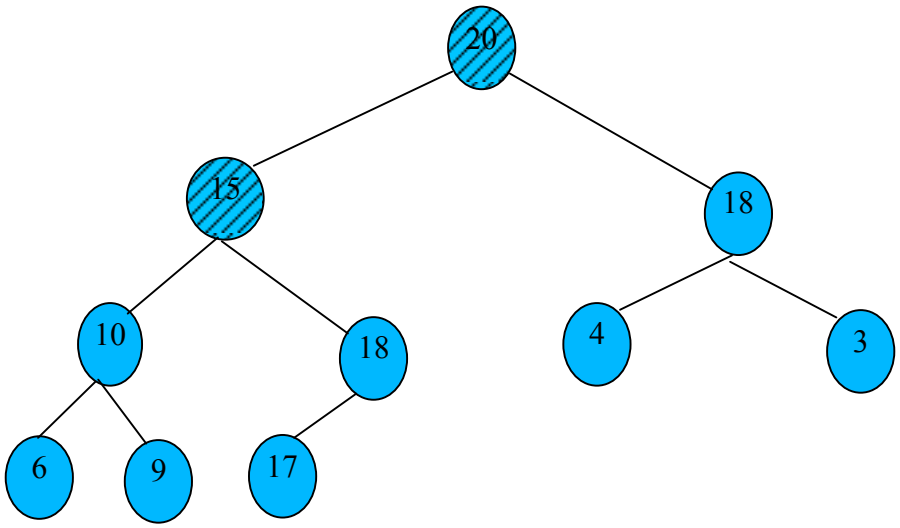
Como podemos comprobar si el padre esta en la posición p los hijos estarán en la posición [redacted] y [redacted]. Si quiero ordenar este montón tendré que llevar el 25 al final del arreglo para lo cual habré de intercambiarlo por el 15. Dejamos el 25 fuera como elemento ya ordenado.

L									R	
15	20	18	10	18	4	3	6	9	17	25

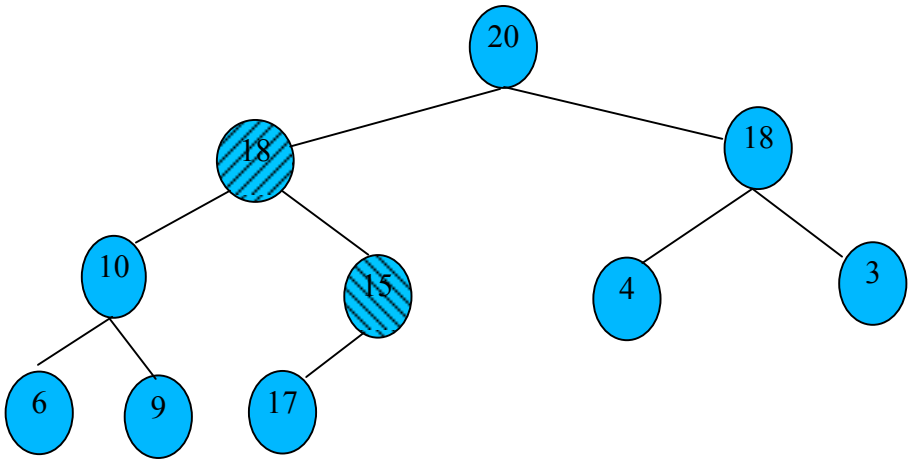
Ahora no tenemos un montón ya que el 15 lo "estropea". Para volver a formar el montón habrá que "hundir" el 15 hasta su posición (lo que en el libro llaman amontonar.



De sus dos hijos lo intercambiaremos por el mayor y asi hasta que de nuevo sea un montón



Luego tendríamos que intercambiarlo por el 18



y finalmente por el 17. Al final nos quedaría el siguiente vector "asociado"

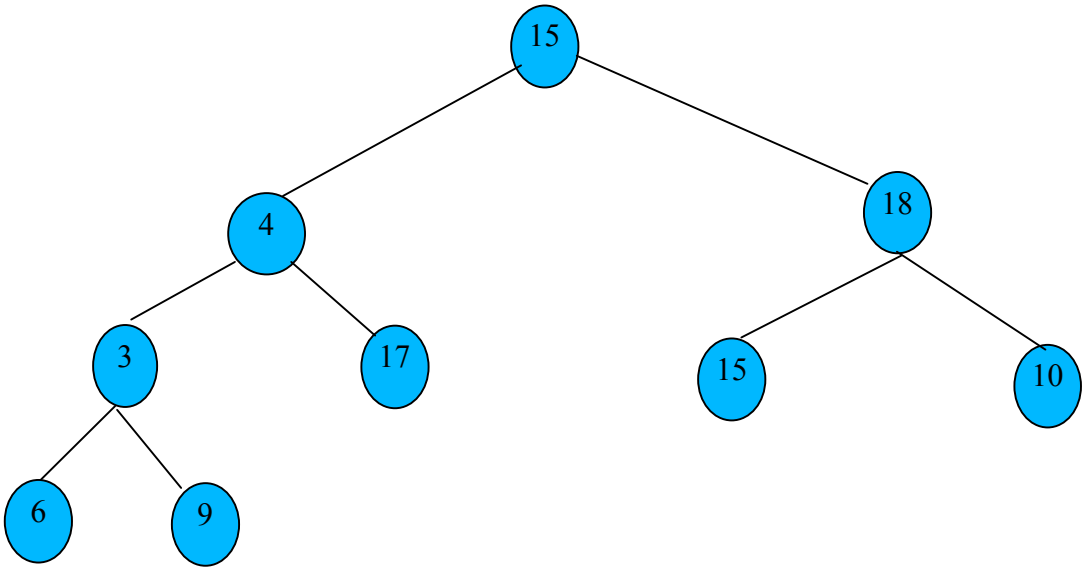
L									R	
20	18	18	10	17	4	3	6	9	15	25

Con lo que tendríamos un nuevo montón y procederíamos a pasar el primer elemento que es el mayor al último lugar y repetiríamos.

Por tanto el algoritmo de para ordenar un montón sería

```
R:=n
WHILE R>1  DO
        aux:= a[1];
        a[1]:=aux;
        a[R]:= aux;
        R:=R-1;
        amontonar(L,R); (* lo que hemos visto como hundir el elemento*)
END;
```

En general nunca partiremos de un montón sino de un arreglo que no es de un montón p.ej.



Todas las hojas son montones y además si nos fijamos son 5 esto es:

nº de hojas= nº de elementos DIV 2

Si pasamos a arreglo el árbol

				L				R
15	4	18	3	17	15	10	6	9

Tendré que ir hundiendo todos los demás empezando desde los nodos inferiores para ir construyendo un montón. A esto se le lama formar *el montón in situ.*

El código del algoritmo sería

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

A continuación entraremos en el procedimiento amontonar que hemos usado hasta ahora.

```
PROCEDURE amontonar (L,R:INTEGER);
VAR
        i,,j,x:INTEGER;
BEGIN
1       i:=L; j:=2*L; x:=a[L];
2       IF (j<R) AND  a[j+1]>a[j] THEN j:=j+1 END; (*Comparamos los dos hijos*)
3       WHILE (j<=R) AND a[j]>x DO
4               a[i]:=a[j]; i:=j;j:=2*I;
5               (j<R) AND  a[j+1]>a[j] THEN j:=j+1 END; (*Comparamos los dos hijos*)
6       END;
7       a[i];=x;
END amontonar;
```

En la línea 1 identificaremos el índice i con la posición del padre. El j con la del hijo derecho y almacenamos en la variable auxiliar x el valor del padre que es el elemento que vamos a hundir.

En la línea 2 comparamos el valor de los hijos derecho e izquierdo para ver por que rama le vamos a hundir

Entramos en el bucle y mientras que no lleguemos al final y el valor de los nodos hijos sea mayor que el del padre original iremos subiendo el nodo hijo a su nueva posición, en la instrucción 5 se repite la 2 con la misma intención.

Finalmente en la línea 7, una vez que hayamos encontrado su posición insertaremos el elemento que hemos "hundido".

Análisis: El peor caso se produce cuando se realiza el máximo nº de intercambios en la construcción de los sucesivos montones. En cada pase i la reconstrucción del montón requerirá 2[logi] comparaciones como máximo. Sumando todos los pases, el número de comparaciones en el peor caso es 2nlog n – O(n). En el caso promedio resulta del orden 2nlogn- O(nloglogn) .

- **Clasificación por partición (QuickSort)**

Es la mejora del algoritmo de intercambio, en este se intercambiaban elementos adyacentes. Se basará que en este caso cuanto mayor sea la distancia entre los elementos que se intercambian más eficaz será la clasificación.

Se elige un valor de *pivote* al azar, x. Recorreremos el arreglo desde la izquierda hasta encontrar una llave mayor que el pivote, y, y desde la derecha hasta encontrar una menor, cuando esto suceda intercambiaremos ambos elementos y avanzaremos los índices, siguiendo así hasta que se crucen. El objetivo es poner a la derecha todos los mayores o iguales que el pivote y a la izquierda todos lo menores. Veamos un ejemplo.

			x				
46	57	14	44	96	20	8	70
i							j

Elegimos como pivote el de en medio (por ejemplo, podríamos haber elegido otro cualquiera).

Avanzaremos la i hasta que encontremos algún elemento mayor que el 44 y retrocederemos la j hasta que encontremos alguno menor.

a[i]=46>44; se para
j=j-1;
a[j]=8<44 se para
Intercambiamos ambos elementos
Avanzamos i y retrocedemos j

8	57	14	44	96	20	46	70
	i				j		

a[i]=57>44; se para
a[j]=20<44 se para
Intercambiamos ambos elementos
Avanzamos i y retrocedemos j

8	20	14	44	96	57	46	70
		i		j			

avanzamos i y descendemos j

8	20	14	44	96	57	46	70
			ij				

como a[i] y a[j] son iguales que el pivote intercambiamos y avanzamos (el objetivo de este intercambio es simplemente el avanzar y que así se crucen).

8	20	14	44	96	57	46	70
		j		i			

Una vez se han cruzado los índices pararíamos y tendríamos dos subvectores el de la derecha con los elementos mayores que el pivote y el de la izquierda con los menores, estando el pivote por tanto en su posición.

A continuación haríamos llamadas recursivas a cada sub-vector hasta que sean de tamaño 1 momento en el cual tendríamos ordenado el arreglo inicial.

Cálculo del coste del quicksort

El método de clasificación por partición es como ya hemos comentado de naturaleza recursiva, por lo que utilizaremos fórmulas de recurrencia.

Para calcular tendremos

i elementos	x	n-i-1 elementos

La recurrencia a resolver será:

T(0)=T(1)= 1;
T(n)= T(i)+T(n-i-1) + cn.

Dependiendo de que sea el peor caso, el mejor o el promedio sólo habrá una parte, con el pivote en un extremo, dos iguales, o cada parte aleatoria.

En el caso peor una de las partes no existirá, estando el pivote en el extremo y la otra parte tendrá n-1 elementos (todos excepto el pivote) asi que la ecuación general quedará

T(n)= T(0)+T(n-1) + cn ≡ T(n-1) + cn (1)

T(n-1)= T(n-2)+c(n-1); sustituyendo en (1) tendremos:

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

T(n-2)= T(n-3)+c(n-2); sustituyendo en (1) tendremos:

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

.....

$$T(n)= T(n-n) +c(\sum_{i=0}^{n-1}(n-i)) \Rightarrow O(n^2)$$

En el mejor de los casos, y aproximando, tendremos dos sub vectores de tamaño n/2 con lo que la ecuación general

T(0)=T(1)= 1;
T(n)= T(i)+T(n-i-1) + cn.

nos quedará en este caso:

$$T(n) = 2 \times T\left(n/2\right) + cn$$

Podemos resolver por dos formas:

1. Sabiendo que en la recurrencia mediante división

$T(n) = aT\left(n/b\right) + cn^k$

si $a = b^k$ entonces

$T(n) \equiv \mathrm{O}(n^k \log n)$; en el caso que nos ocupa k=1 luego

$T(n) \equiv \mathrm{O}(n \log n)$

2.. Resolviendo la recurrencia por expansión

$T(n) = 2 \times T\left(n/2\right) + cn$; dividimos por n

$\frac{T(n)}{n} = 2 \times \frac{T\left(n/2\right)}{n} + c$; introducimos el 2 en el divisor;

$\frac{T(n)}{n} = \frac{T\left(n/2\right)}{n/2} + c$ (1)

Expandimos

$\frac{T\left(n/2\right)}{n/2} = \frac{T\left(\frac{n/2}{2}\right)}{\frac{n/2}{2}} + c = \frac{T\left(n/4\right)}{n/4} + c$; Sustituyendo en (1);

$\frac{T(n)}{n} = \frac{T\left(n/4\right)}{n/4} + 2c$; si volvemos a expandir

$\frac{T\left(n/4\right)}{n/4} = \frac{T\left(\frac{n/4}{2}\right)}{\frac{n/4}{2}} + c = \frac{T\left(n/8\right)}{n/8} + c$; sustituyendo

$\frac{T(n)}{n} = \frac{T\left(n/8\right)}{n/8} + 3c$; El término genérico será

$\frac{T(n)}{n} = \frac{T\left(n/2^k\right)}{n/2^k} + kc$; si $\frac{n}{2^k} = 1 \Rightarrow n = 2^k \Rightarrow k = \log n$; Por tanto

$\frac{T(n)}{n} = \frac{T(1)}{1} + c \log n \Rightarrow T(n) \equiv \mathrm{O}(n \log n)$

CLASIFICACION EN MEMORIA EXTERNA O SECUNDARIA

El TDA a considerar será la secuencia, implementada mediante una cinta, por lo que tendremos que tener en cuenta que el acceso será secuencial y no aleatorio, es decir no podremos acceder al elemento n si haber accedido previamente al n-1.

Se entiende por mezcla la tarea de combinar varias secuencias en una sola mediante selección de los elementos accesibles en cada momento.

Se entiende por fase cada operación que trata de un conjunto completo de datos.

Se entiende por pase al proceso más corto que por repetición constituye un proceso de clasificación.

En el caso de la memoria secundaria la desproporción entre movimientos y comparaciones va a ser mucho mayor por lo que serán los primeros los que nos determinen la bondad de un algoritmo.

Clasificación basada en mezclas

Mezcla directa

Los pasos a seguir serán:

- ✓ Tomar como fuente la secuencia original
- ✓ División o distribución: Dividir la fuente en dos mitades en las cintas destino c1 y c2
- ✓ Mezcla: Mezclar c2 y c3 combinando cada elemento accesible en pares ordenados
- ✓ Repetir el pase compuesto de las dos etapas hasta el ordenamiento de la cinta.

Ejemplo:

c1

46	57	14	44	96	20	8	70

PRIMER PASE

Fase división

c2

46	14	96	8

c3

57	44	20	70

Fase Mezcla

Solo tenemos acceso al primer elemento de cada cinta y los vamos mezclando en c1

46	57	14	44	20	96	8	70

tenemos "pares" que sabemos ordenados asi que en el segundo pase la división la haremos de 2 en 2

SEGUNDO PASE

Fase división

c2

46	57	20	96

c3

14	44	8	70

Fase Mezcla

c1

14	44	46	57	8	20	70	96

TERCER PASE

Fase división

c2

14	44	46	57

c3

8	20	70	96

Fase Mezcla

c1

8	17	20	44	46	57	70	96

Cuando una de las dos cintas se agota se procede a copiar el resto de los elementos de la otra. Esto se conoce como *copiado de cabos*

Si hubiéramos ordenado 2 elementos hubiéramos necesitado 1 pase

para 4 ---------- 2 pases
para 8 ---------- 3pases

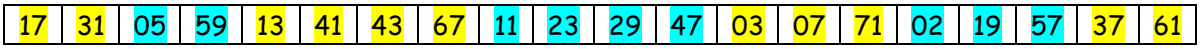
para n ---------- [log n] (parte entera superior de…)

En cada pase el elemento se copia dos veces, como hay n elementos, el nº de movimientos será 2n[log n]

MEZCLA NATURAL

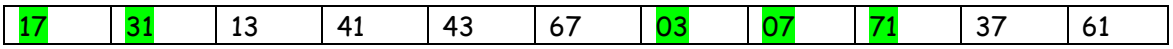
La mezcla natural va a sacar partido de que tanto la secuencia original como las que se van a ir formando tendrán elementos consecutivos que ya estén ordenados. La mezcla natural aprovechará estas ordenaciones parciales consiguiendo un mayor rendimiento.

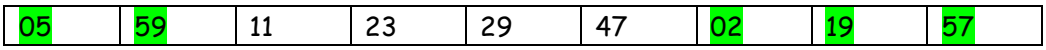
Ejemplo:



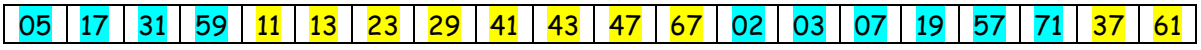
PRIMER PASE

Distribución



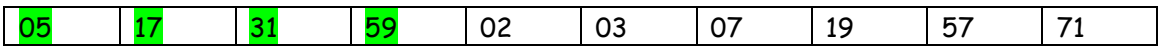


Mezcla



SEGUNDO PASE

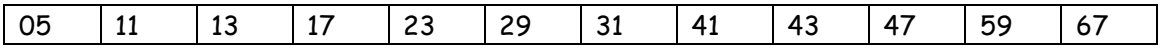
Distribución

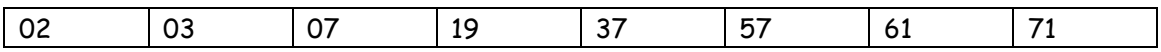


Mezcla

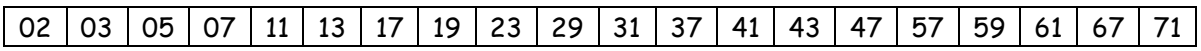
TERCER PASE

Distribución





Mezcla



Como vemos hemos realizado la ordenación de 20 elementos en 3 pases cuando en la directa hubiéramos necesitado 5. En este caso las comparaciones son muchas más pero como dijimos al principio del témale coste de una comparación es muy inferior al de un movimiento por lo que este incremento no resulta significativo

MEZCLA BALANCEADA MÚLTIPLE

En este caso eliminaremos la fase de distribución copiando las subsecuencias ordenadas en el proceso de mezcla.

Ejemplo con dos cintas fuente y dos destino

46	57	14	44	96	20	8	70	50	75	36	78

sólo realizamos distribución en el primer pase

46	57	20	50	75

14	44	96	8	70	36	78

mezclaremos y haremos distribución a la vez en las dos cintas destino

14	44	46	57	96	36	50	75	78

8	20	70

Ahora son origen y mezclamos y distribuimos a las destino

8	14	20	44	46	57	70	96

36	50	75	78

y queda ordenado en el último pase

8	14	20	36	44	46	50	57	70	75	78	96

Análisis

Generalizando a N cintas destino y N cintas origen

m= nº de subsecuencias ordenadas

n=nº de elementos

Por tanto tendré m/N subsecuencias en cada cinta, al siguiente pase irán a N cintas otra vez luego tendré m/N^2, en el k-esimo tendré m/N^k siendo k el nº de pases en los que he conseguido ordenar y por tanto sólo tendría una subsecuencia

$1 = \frac{m}{N^k}; m = N^k; k = \log_N m$ siendo k el nº de pases como en cada pase copio los n números $O(n) \equiv n \log_N m;$ en el peor de los casos cada subsecuencia será de un solo elemento y por tanto m=n luego el coste en el peor de los casos será:

$$O(n) \equiv n \log_N n$$

cuanto mayor sea el nº de cintas (N) mas eficiente será el algoritmo

CLASIFICACIÓN POLIFÁSICA

Mejora el rendimiento de la mezcla balanceada. Las cintas fuente y destino no son establecidas a priori sino como consecuencia de la mezcla realizada.

Habrá N-1 cintas origen y 1 destino que irá rotando.

Pongamos un pequeño con 3 cintas con las subsecuencias ordenas de un solo elemento (peor de los casos) para facilitar la comprensión

15	13	9	8	5	4	3	1

En este caso tendremos que repartirlo en dos cintas a la que cada una "mandaremos" un número determinado de subsecuencias, luego veremos por que. En este caso 5 y 3.

c1

15	13	9	8	5

c2

4	3	1

Comenzamos a mezclar

c3

4	15	3	13	1	9

Ahora la cinta c2 está vacía con lo que en vez de proceder al copiado de cabos como antes con c1 lo que hacemos es convertir c2 en cinta destino y seguir mezclando. (Observar que nos quedan 3 subsecuencias y partíamos de 5).

c3

4	15	3	13	1	9

c1

			8	5

c2

4	8	15	3	5	13

c1 queda vacía con lo que pasa a ser destino

c2

4	8	15	3	5	13

c3

				1	9

c1

1	4	8	9	15

c3 queda vacía con lo que pasa a ser destino

c1

1	4	8	9	15			

c2

			3	5	13

1	3	4	5	8	9	13	15

Si nos fijamos en el nº de subsecuencias ordenadas que ha habido en cada pase tendremos

c1	c2	c3
5	3	0
2	0	3
0	2	1
1	1	0
0	0	1

Si "vamos al revés"

C1	C2	Suma
1	0	1
1	1	2
2	1	3
3	2	5
5	3	8
8	5	13
13	8	21

El nº de subsecuencias ordenadas tiene que ser un nº de Fibonacci. Además el reparto en las dos cintas origen tiene que ser de números de Fibonacci consecutivos. Esto es condición suficiente aunque no necesaria, puede haber otros repartos que funcionen. Si por ejemplo quisiéramos ordenar 12 secuencias deberíamos añadir una corrida ficticia para poder hacerlo

Si en vez de 3 tuviéramos 4 cintas el nº de subsecuencias sería el siguiente

C1	C2	C3	Suma
1	1	1	3
2	2	1	5
4	3	2	9
7	6	4	17
13	11	7	31

Vemos que la suma son números de Fibonacci de orden 2 es decir suma de los 3 anteriores. Y el nº de subsecuencias en cada cinta se reparte de la siguiente forma

C1	C2	C3
x	y	z
x+y	x+z	x
4	3	2
7	6	4
13	11	7

en general el nº de subsecuencias necesarias tendrá que ser Fibonacci del orden N-2 donde N es el nº de cintas de las que disponemos.

Numero de cintas (N)

Números de Fibonacci N-2

3	4	5	6	7	8
3	5	7	9	11	13
5	9	13	17	21	25
8	17	25	33	41	49
13	31	49	65	81	97
21	57	94	129	161	193
34	105	181	253	321	385
55	193	355	497	636	769
89	355	673	977	1261	1531

En azul vemos los primeros números de cada serie, los cuales se pueden calcular rápido ya que si nos fijamos son el doble del anterior menos 1.

TABLAS HASH

La idea de estas tablas no es ordenar sino saber donde guardo la información y por tanto donde puedo buscarla rápidamente.

La idea es que aunque el acceso los sectores de un disco de la memoria secundaria es directo. Con lo cual cada vez que tenga un dato tendré que tener una *función de dispersión* que lo mande a un sector.

Esta función tendrá que ser eficiente (es decir sencilla) y uniforme (esto es, que reparta más o menos igual los datos entre todos los sectores). Un ejemplo de una función de dispersión podría ser el si tenemos alumnos que llegan con un nº de alumno determinado hacer una función por ejemplo

nº alumno mod 15; si tuviéramos el alumno 715 iría al sector 15

Supongamos por simplificar que en cada sector sólo cabe un "dato"

Si nuestra función mandara a una posición ocupada un dato, este no cabría. Se produce lo que se denomina una colisión. Ante este problema hay varias soluciones.

1. Estrategia de encadenamiento: Tener un área de desbordamiento mediante punteros.
2. Hashing lineal: Consiste en guardar el dato el el siguiente sector que encuentre libre. El problema de este método es que se pueden construir bloques de sectores consecutivos muy grandes, con lo que tendré que recorrer muchos sectores para encontrar el dato buscado o para asegurar que no está.
3. Hashing cuadrático: Cojo el índice (15 en el caso del ejemplo) y lo elevo al cuadrado y el resultado mod 15 es donde va el dato, en este caso el 25. El problema es que se obtengan *ciclos*, si tuviéramos el 615 no le encontraríamos ubicación libre. Si el tamaño de la tabla es 4j+3 no se producen ciclos, por ejemplo en lugar de una tabla del 0 al 100 posiciones, hacer una del 0 al 102.

Para que la tabla no sea demasiado "densa" y halla "huecos" conviene que supere en un 20% el tamaño necesario.