Departamento de Informática
Universidad de Valladolid
Campus de Segovia

TEMA 6: ALGORITMOS DE BÚSQUEDA Y ORDENACIÓN

Prof. José Vicente Álvarez Bravo

ALGORITMOS DE BÚSQUEDA Y ORDENACIÓN

- Algoritmos de búsqueda en arrays
  - Secuencial
  - Secuencial ordenada
  - Binaria
- Ordenación de vectores
  - Selección directa
  - Inserción directa
  - Intercambio directo
  - Ordenación rápida (Quick Sort)
  - Ordenación por mezcla (Merge Sort)

- Algoritmos de búsqueda y ordenación en archivos

ALGORITMOS DE BÚSQUEDA EN ARRAYS

- Surgen de la necesidad de conocer tanto si un dato se encuentra o no dentro de una colección como de la posición que ocupa.

- Búsqueda(vector,elemento):
  - $i \in \{1,....,n\}$ si existe tal elemento
  - 0 en otro caso

- Estructura de datos:

```
const
    N=100;
type
    tIntervalo=0..N;
    tvector=array[1..N] of tElem {tipo ordinal}
```

ALGORITMOS DE BÚSQUEDA EN ARRAYS

- Algoritmos de búsqueda:
  - Búsqueda secuencial
  - Búsqueda secuencial ordenada
  - Búsqueda binaria

BÚSQUEDA SECUENCIAL

- La búsqueda secuencial consiste en comparar secuencialmente el elemento deseado con los valores contenidos en las posiciones 1,....,n.

- El proceso termina cuando o bien encontramos el elemento o bien se alcanza el final del vector.

- Un primer nivel de diseño:

```
ind:=0
buscar elemento en vector
si vector[ind]=elemento entonces
    busquedasecuencial:=ind
sino
    busquedasecuencial:=0
```

BÚSQUEDA SECUENCIAL,
IMPLEMENTACIÓN EN PASCAL

```
FUNCTION Busquedasec(v:tvector ; elem:telem):tIntervalo;
{Dev. 0 si el elemento no está en ‘v’ o i si v[ì]=elem}

VAR
    i:tIntervalo;
BEGIN
    i:=0;
    repeat
        i:=i+1;
    until (v[i]=elem) or (i=N);
    if v[i]=elem then
                busquedasec:=i
        else
                busquedasec:=0
END; {busquedasec}
```

Este algoritmo en el peor de los casos es de orden O(n).

BÚSQUEDA SECUENCIAL ORDENADA

- El algoritmo anterior puede ser mejorado si el vector ‘v’ esta ordenado (i.e. Creciente).

- De esta forma si durante la búsqueda se alcanza una componente con mayor valor que ‘elem’, podremos asegurar que no se encuentra dentro de la colección.

BÚSQUEDA SECUENCIAL ORDENADA,
IMPLEMENTACIÓN EN PASCAL

```
FUNCTION Busquedasecord(v:tvector ; elem:telem):tIntervalo;
{Dev. 0 si el elemento no está en ‘v’ o i si v[ì]=elem}

VAR
    i:tIntervalo;
BEGIN
    i:=0;
    repeat
        i:=i+1;
    until (v[i]≥elem) or (i=N);
    if v[i]=elem then
                busquedasecord:=i
        else
                busquedasecord:=0
END; {Busquedasecord}
```

Este algoritmo en el peor de los casos es de orden O(n).

BÚSQUEDA BINARIA

- El hecho de que el vector este ordenado se puede, también, aprovechar para conseguir una mayor eficiencia planteando el siguiente algoritmo.
  - Comparar ‘elem’ con el elemento central del vector. Si este es el elemento buscado se finaliza. Si no se sigue buscando en la mitad del vector que determine la relación entre el valor del elemento central y el buscado.
  - Este algoritmo finaliza cuando se localiza el elemento o se termina el vector.
- Debido a que el vector es dividido sucesivamente en dos se denomina búsqueda binaria.

BÚSQUEDA BINARIA,
IMPLEMENTACIÓN EN PASCAL

```
FUNCTION Busquedabinaria(v:tvector ; elem:telem):tIntervalo;
{Prec. V esta ordenado crecientemente}
{Dev. 0 si el elemento no está en 'v' o i si v[ì]=elem}
VAR
    einf,esup,posmed:tIntervalo;
    encontrado:boolean;
BEGIN
    einf:=1; esup:=N; encontrado:=false;
    while not encontrado and (esup≥einf) do begin
            posmed:=(esup+einf) DIV 2;
            if  elem=v[posmed] then
                        encontrado:=true
                else if elem>v[posmed] then
                                    einf:=postmed+1
                        else
                                    esup:=postmed-1
    end {while}
    if encontrado then
            busquedabinaria:=posmed
      else
            busquedabinaria:=0
END; {Busquedabinaria}
```

BÚSQUEDA BINARIA

- La complejidad algorítmica de la búsqueda binaria es:

Si

$$2^k \leq N \leq 2^{k+1}$$

entonces:

$$T(n) \leq T(2^{k+1})$$

En el peor de los casos:

$$T(n) \approx O[\log N]$$

$N \leq 2^{k+1} = 22^k$

$\log N \leq \log 2^{k+1} = k+1$

ORDENACIÓN DE VECTORES

- Como se ha presentado con anterioridad, disponer de una colección de datos ordenados facilita la operación de búsqueda. A continuación se presentan algunos de los algoritmos de ordenación más frecuentes:

  - Algoritmos cuadráticos:

    - Selección directa
    - Inserción directa
    - Intercambio directo

  - Algoritmos avanzados de búsqueda:

    - Algoritmo rápido (Quick Sort)
    - Algoritmo de mezcla (Merge Sort)

ORDENACIÓN DE VECTORES

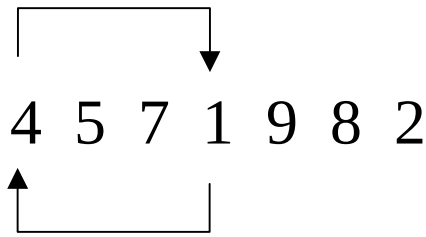
- Algoritmos cuadráticos:
  - Selección directa
  - Inserción directa
  - Intercambio directo
- Algoritmos avanzados de búsqueda:
  - Algoritmo rápido (Quick Sort)
  - Algoritmo de mezcla (Merge Sort)

SELECCIÓN DIRECTA

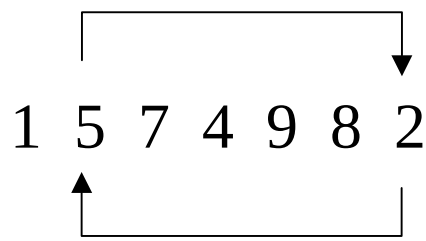
- Procedimiento: Recorre el vector y selecciona en cada uno de los recorridos el menor elemento para situarlo en su lugar correspondiente.

SELECCIÓN DIRECTA

- 1. Se sitúa en v[1] el menor elemento de entre v[1],...,v[n]. Para ello se intercambian los valores de v[1] y v[i] {v[i]=min(v)}



- 2. Se sitúa en v[2] el menor elemento de entre v[2],...,v[n]. Para ello se intercambian los valores de v[2] y v[i] {v[i]=min(v)}



- (n-1). Se sitúa en v[n-1] el menor elemento de entre v[n-1] y v[n]. Para ello se intercambian los valores de v[n-1] y v[n].

1 2 4 5 7 8 9

SELECCIÓN DIRECTA,
IMPLEMENTACIÓN EN PASCAL

```
PROCEDURE  Selecciondirecta( var v:tvector);
{Efecto. Se ordena 'v' ascendentemente}
VAR
    i, j,posmenor:tIntervalo;
    valmenor,aux:telem;
BEGIN
    for i:=1 to N-1 do begin
        valmenor:=v[i]
        posmenor:=i
        for j:=i+1 to N do
            if  v[j]<valmenor then begin
                valmenor:=v[j];
                posmenor:=j
            end; {if }
        if posmenor<>i then begin
            aux:=v[i];
            v[i]:=v[posmenor];
            v[posmenor]:=aux
         end {if}
    end{for i}
END; {Selecciondirecta}
```

Busca el menor elemento de entre i+1,....,N

Intercambio si posmenor es diferente de i

INSERCIÓN DIRECTA

- Procedimiento: Recorre el vector ‘v’ insertando el elemento v[i] en su lugar correcto entre los ya ordenados.

INSERCIÓN DIRECTA

- 1. Se considera v[1] como primer elemento.
- 2. Se inserta v[2] en su posición correspondiente en relación a v[1] y v[2].
- 3. Se inserta v[3] en su posición correspondiente en relación a v[1], v[2] y v[3].
- i. Se inserta v[i] en su posición correspondiente en relación con v[1],...,v[i].
- n. Se repite la operación con el último elemento del vector.

INSERCIÓN DIRECTA,
IMPLEMENTACIÓN EN PASCAL

```
PROCEDURE  Inserciondirecta( var v:tvector);
{Efecto. Se ordena 'v' ascendentemente}
VAR
    i, j:tIntervalo;
    aux:telem;
BEGIN
    for i:=2 to N do begin
        aux:=v[i]
        j:=i-1
        while (j≥1) and (v[j]>aux) do begin
            v[j+1]:=v[j];
            j:=j-1
        end; {while}
        v[j+1]:=aux
    end {for}
END; {inserciondirecta}
```

Desplazamiento del menor valor

Colocación del valor en su lugar correspondiente

INTERCAMBIO DIRECTO

- Procedimiento: Recorre el vector ‘v’ buscando el menor elemento desde la última posición hasta la actual y lo inserta en dicha posición.

INTERCAMBIO DIRECTO

- 1. Situar el elemento menor en la primera posición. Para ello se compara el último elemento con el penúltimo intercambiando los valores si están en orden decreciente. Se repite esta operación hasta llegar a la primera posición. (el elemento menor se encuentra en la primera posición).

- 2. Situar el segundo menor elemento en la segunda posición. Para ello se procede como antes finalizando al alcanzar al segunada posición.

- 3. Repetir este proceso con la tercera, cuarta,...,y con la enésima posición intercambiando si procede en cada caso.

INTERCAMBIO DIRECTO,
IMPLEMENTACIÓN EN PASCAL

```
PROCEDURE  Intercambiodirecto( var v:tvector);
{Efecto. Se ordena 'v' ascendentemente}
VAR
    i, j:tIntervalo;
    aux:telem;
BEGIN
    for i:=1 to N-1 do
        for j:=N downto i+1
        {se busca el menor desde atrás y se sitúa en v_i}
            if v[j-1]>v[j] then begin
                aux:=v[j];
                v[j]:=v[j-1];   <-------  Intercambio
                v[j-1]:=aux;
            end; {if}
END; {intercambiodirecto}
```

ALGORITMOS DE BÚSQUEDA AVANZADA

- Ordenación rápida (Quick sort)
- Ordenación por mezcla (Merge sort)

ORDENACIÓN RÁPIDA

- El algoritmo consiste en dividir el vector que se desea ordenar en dos bloques. En el primero se sitúan todos los elementos que son menores que un cierto valor que se toma como referencia (pivote), mientras que en segundo irían el resto.

- Este procedimiento se repite dividiendo a su vez cada uno de estos bloques y repitiendo la operación anteriormente descrita.

- La condición de parada se da cuando el bloque que se desea ordenar está formado por un único elemento (bloque ordenado).

- El esquema seguido por este algoritmo es el de ‘divide y venceras’.

ORDENACIÓN RÁPIDA, PSEUDOCÓDIGO

Si v es de tamaño 1 entonces
    el vector v ya está ordenado
sino
    dividir v en dos bloques A y B
    con todos los elementos de A menores que los de B
fin {si}

Ordenar A y B usando Quick Sort
Devolver v ya ordenado.

- Donde dividir v en dos bloques se puede refinar:

Elegir un elemento como pivote de v
para cada elemento de v hacer:
    si el elemento es < pivote colocarlo en A
    en otro caso situarlo en B.

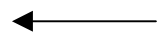
ORDENACIÓN RÁPIDA,
IMPLEMENTACIÓN EN PASCAL

```
PROCEDURE  Quicksort( var v:tvector);
{Efecto. Se ordena ‘v’ ascendentemente}
    PROCEDURE Sortdesdehasta(var v:tvector;izq,der:tintervalo);
    {Efecto. Se ordena ‘v[izq..der]’ ascendentemente}
     {siguiente página}

BEGIN {Quicksort}
    Sortdesdehasta(v,1,n);
END; {Quicksort}
```

ORDENACIÓN RÁPIDA,
IMPLEMENTACIÓN EN PASCAL

```
PROCEDURE Sortdesdehasta(var v:tvector;izq,der:tintervalo);
{Efecto. Se ordena 'v[izq..der]' ascendentemente}
VAR
        i, j:tIntervalo;
        p,aux:telem;
BEGIN
        i:=izq; j:=der; p:=v[(izq+der) DIV 2];
        while i<j do begin      {se reorganizan los vectores}
                while v[i]<p do
                        i:=i+1;
                while p<v[j] do
                        j:=j-1;
                if  i≤ j then begin {intercambio de elementos}
                        aux:=v[i];
                        v[i]:=v[j];
                        v[j]:=aux;
                        i:=i+1; j:=j-1; {ajuste posiciones}
                end; {if}
        end; {while}
if izq<j then sortdesdehasta(v, izq, j);
if i<der then sortdesdehasta(v, i, der);
END; {Sortdesdehasta}
```



ORDENACIÓN RÁPIDA,
TRAZA PARA UN EJEMPLO

V=[0,5,15,9,11]

1. Sortdesdehasta(v,1,5) p=v[3]=15

		i=1 [0,5,15,9,11]	0<15
		i=2 [0,5,15,9,11]	5<15
		i=3 [0,5,15,9,11]	15 not <15
	i=3	j=5 [0,5,15,9,11]	11 not >15
	i=4	j=4 [0,5,11,9,15]	intercambio. Salida bucle

1.1.sortdesdehasta(v,1,4) p=v[2]=5

		i=1 [0,5,11,9]	0<5
		i=2 [0,5,11,9]	5 not< 5
	i=2	j=4 [0,5,11,9]	9>5
	i=2	j=3 [0,5,11,9]	11>5
	i=2	j=3 [0,5,11,9]	5 not>5
	i=3	j=1 [0,5,11,9]	intercambio. Salida bucle

1.1.1.sortdesdehasta(v,3,4) p=v[3]=11

		i=3 [11,9]	11 not <11
		i=4 [11,9]	9 not> 11
	i=4	j=3 [9,11]	intercambio. Salida bucle

ORDENACIÓN RÁPIDA,
TRAZA PARA UN EJEMPLO

V=[0,5,15,9,11]

1.2 Sortdesdehasta(v,4,5) p=v[4]=11

	i=4 [11,15]		11 not <11
i=4	j=5 [11,15]		15>11
i=4	j=4 [11,15]		11 not >11
i=5	j=3 [11,15]		intercambio. Salida bucle

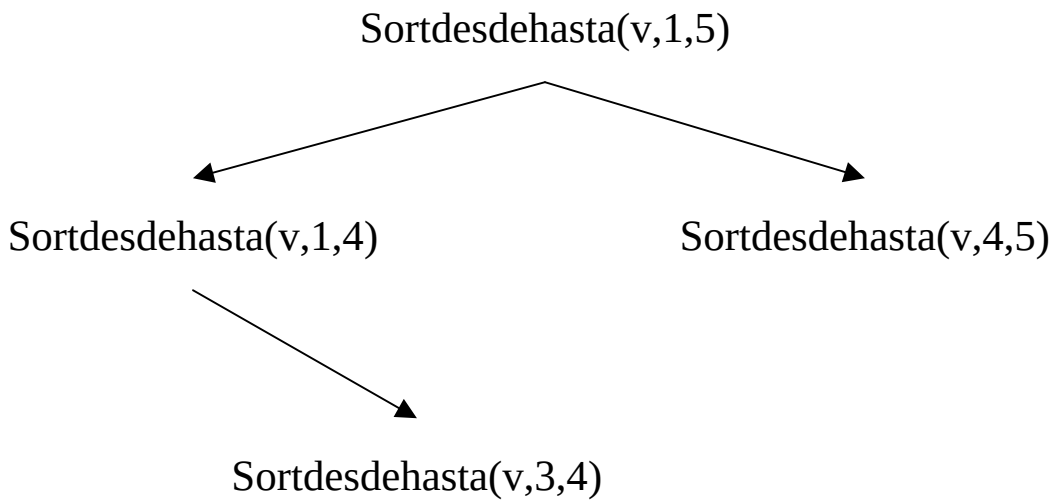
izq=4 not <j=3

i=5 not <der=5

termina recursión

vector ordenado: [0,5,9,11,15]

ORDENACIÓN RÁPIDA,
TRAZA PARA UN EJEMPLO



ORDENACIÓN RÁPIDA,
CÁLCULO DE SU COMPLEJIDAD ALGORÍTMICA

- El peor caso: Si se elige como pivote el primer elemento del vector y además se considera que el vector esta ordenado decrecientemente entonces, el bucle '... Para cada elemento...' se ejecutará en total:

  (n-1)+(n-2)+(n-3)+....+1

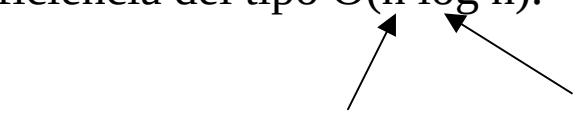
  Cada miembro de este sumando proviene de cada una de las sucesivas ordenaciones recursivas.

  Este sumatorio da lugar a la siguiente expresión:

$$\sum_{i=1}^{n}(n-i)=\frac{[(n-1)+1](n-1)}{2}=\frac{n(n-1)}{2}$$

  Que es de orden cuadrático $T(n)\in O(n^2)$

- Como se puede apreciar si la elección del pivote es tal que los subvectores son de igual tamaño y además los vectores están ordenados de forma aleatoria entonces el algoritmo posee una eficiencia del tipo O(n log n).



Recorrido del vector Numero de divisiones

ORDENACIÓN POR MEZCLA

- En este caso se sigue un esquema similar a la ordenación rápida, el esquema de divide y venceras. Sin embargo el mayor esfuerzo no se realiza dividiendo el vector y reorganizando cada división, sino que tiene lugar cuando finalmente se construye el vector ordenado a partir de la mezcla de los subvectores que se generan.

- La idea consiste en dividir el vector v en dos subvectores para posteriormente mezclar ordenadamente las soluciones obtenidas al ordenar A y B, aplicando nuevamente el algoritmo ‘merge sort’ a ambos subvectores.

ORDENACIÓN POR MEZCLA, PSEUDOCÓDIGO

Si v es de tamaño 1 entonces
    el vector v ya está ordenado
sino
    dividir v en dos subvectores A y B
fin {si}

Ordenar A y B usando Mergesort
Mezclar las ordenaciones de A y B para generar el vector ordenado.

- Donde dividir v en dos subvectores se puede refinar:

  Asignar a A el subvector $[v_1,.....,v_{nDIV2}]$
  Asignar a B el subvector $[v_{nDIV2+1},.....v_n]$

- Mientras que mezclar las ordenaciones de A y B consiste en ir entremezclando adecuadamente las componentes ya ordenadas de A y B.

ORDENACIÓN POR MEZCLA,
IMPLEMENTACIÓN EN PASCAL

```
PROCEDURE  Mergesort( var v:tvector);
{Efecto. Se ordena ‘v’ ascendentemente}
    PROCEDURE Mergedesdehasta(var v:tvector; izq, der:
    tintervalo);
    {Efecto. Se ordena ‘v[izq..der]’ ascendentemente}
    VAR
    centro :tIntervalo
        PROCEDURE Merge(var v:tvector; iz, ce, de:tintervalo;
        var w:tvector);
        {siguiente página}
    BEGIN {Mergedesdehasta}
        centro:=(izq+der)DIV2;
        if izq<centro then
                Mergedesdehasta(v,izq,centro);
        if centro<der then
                Mergedesdehasta(v,centro+1,der);
        Merge(v,izq,centro,der,v)
    END; {Mergedesdehasta}

BEGIN {Mergesort}
    Mergedesdehasta(v,1,n)
END; {Mergesort}
```

ORDENACIÓN POR MEZCLA,
IMPLEMENTACIÓN EN PASCAL

```
PROCEDURE Merge(var v:tvector; iz, ce, de:tintervalo; var
    w:tvector);
{Efecto.w: mezcla ordenada de los subvec. v[iz..ce],v[ce+1..de] }
    VAR
        i, j,k:tIntervalo;
    BEGIN
        i:=iz; j:=ce+1; k:=iz;  {k recorre w}
        while( i ≤ ce) and (j ≤de) do begin
            if v[i]<v[j] then begin
                w[k]:=v[i];
                i:=i+1
            end; {if}
            else begin
                w[k]:=v[j];
                j:=j+1;
            end; {else}
            k:=k+1
        end; {while}
        for k:=j to de do
            w[k]:=v[k]
            for k:=i to ce do
                w[k+de-ce]:=v[k]
    END; {Merge}
```

ORDENACIÓN POR MEZCLA,
TRAZA PARA UN EJEMPLO

V=[8,5,7,3]

1. Mergedesdehasta(v,1,4) centro=2

1.1. Izq(1) <centro(2) Mergedesdehasta(v,1,2) centro=1

1.1.1. izq(1) not< centro(1)

1.1.2. centro(1)< der(2)

1.1.2.1 izq(2) not< centro(2)

1.1.2.2. centro(2) not< der(2)

1.1.2.3. Merge(v,2,2,2,v) trivial

Merge(v1,1,2,v)-------- w1[5,8]

1.2. Centro(2) < der(4) Mergedesdehasta(v,3,4) centro=3

1.2.1. izq(3) not< centro(3)

1.2.2. centro(3) < der(4)

1.2.2.1. izq(4) not< centro(4)

1.2.2.2. centro(4) not< der(4)

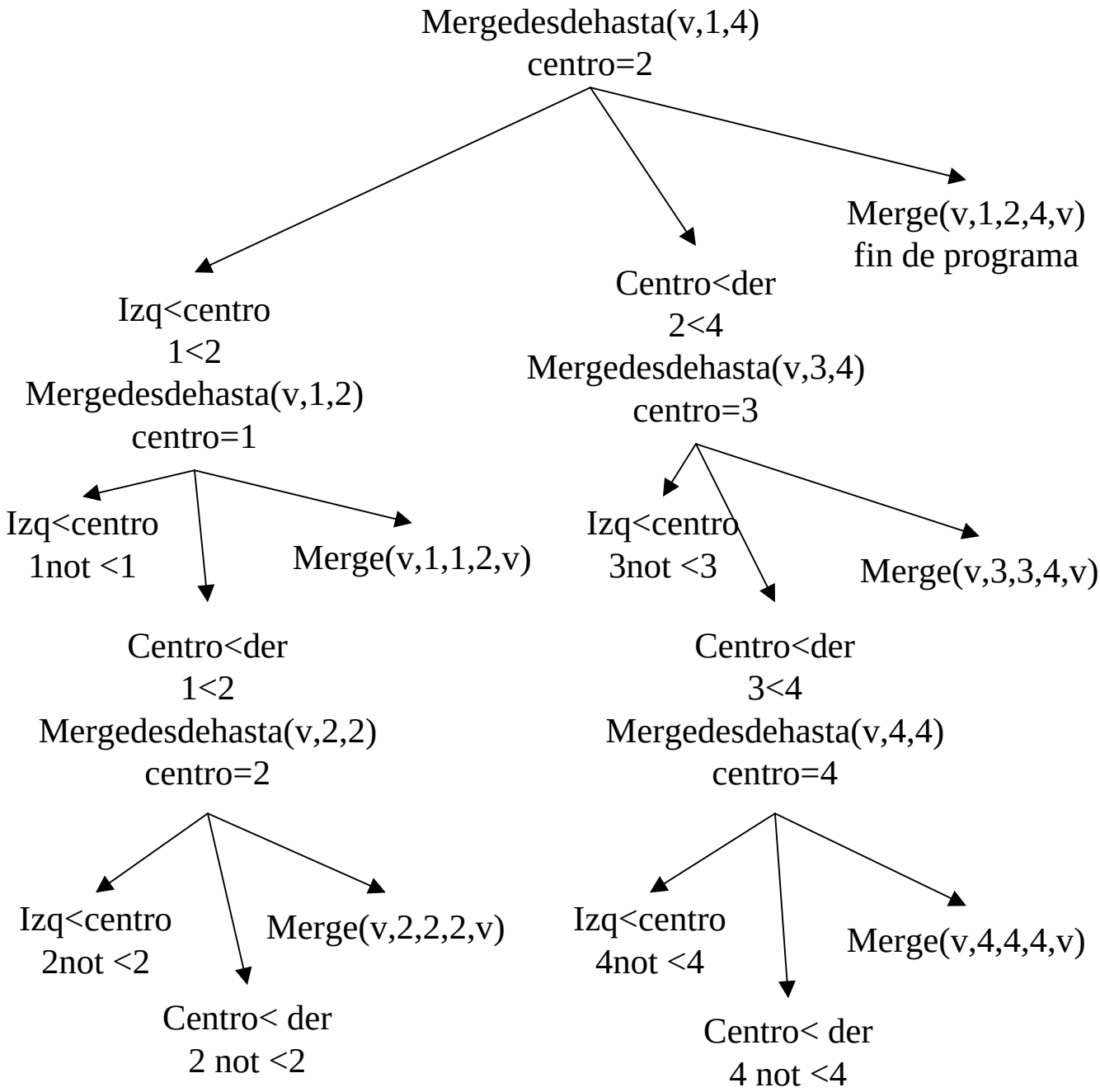
1.1.2.3. Merge(v,4,4,4,v) trivial

Merge(v3,3,4,v)-------- w2[3,7]

1.3. Merge(v,1,2,4,v)----------w[3,5,7,8]

Mezcla ordenada de w1 y w2

ORDENACIÓN POR MEZCLA,
TRAZA PARA UN EJEMPLO



ORDENACIÓN POR MEZCLA,
CÁLCULO DE SU COMPLEJIDAD ALGORÍTMICA

- El peor caso: Puesto que la mezcla se ejecuta en un tiempo proporcional a n (la longitud del vector a ordenar) el coste en tiempo vendrá dado por la siguiente recurrencia:

$$T(n) = \begin{cases} k_1 & si\ n = 1 \\ 2T\left(\frac{n}{2}\right) + k_2 n + k_3 & si\ n > 1 \end{cases}$$

- Esta ecuación se resuelve mediante sustituciones sucesivas cuando n es una potencia de 2 (j tal que $n=2^j$).

$$\begin{aligned} T(n) &= 2\left(2T\left(n/4\right) + k_2\left(n/2\right) + k_3\right) + k_2 n + k_3 \\ &= 4T\left(n/4\right) + 2k_2 n + k_4 = \\ &= 4\left(2T\left(n/8\right) + k_2\left(n/4\right) + k_3\right) + 2k_2 n + k_4 = \\ &= 8T\left(n/8\right) + 3k_2 n + k_5 = \\ &= \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots = \\ &= 2^j T(1) + jk_2 n + k_j \end{aligned}$$

- De donde sustituyendo j=log n lo que da una función de orden O(n logn) (una vez aplicadas la regla de la suma y de la escalabilidad).

ALGORITMOS DE BÚSQUEDA EN ARCHIVOS SECUENCIALES

- La variedad de algoritmos de búsqueda en archivos se ve reducida por el tipo de acceso secuencial, que se utiliza en el Pascal estándar.

- La estrategia a emplear será la misma que la empleada en la búsqueda secuencial en vectores.

ALGORITMOS DE BÚSQUEDA EN ARCHIVOS SECUENCIALES

```
TYPE
tElem=record
    clave: tclave;
    resto de la información
    end;
tarchivoelem=file of tElem.

PROCEDURE buscar(var f:tarchivoelem;c:tclave;var
    elem:tElem; var encontrado:boolean);
{Efecto.  Si c está en f entonces encontrado=true y elem es el
    elemento buscado, en  otro caso encontrado=false}
Begin
    encontrado:=false;
    while not Eof(f) and not encontrado do begin
            read(f,elem);
            if c:=elem.clave then
                    encontrado:=true
    end; {while}
end; {buscar}
```

BÚSQUEDA EN ARCHIVOS SECUENCIALES ORDENADOS

El algoritmo, en este caso puede ser mejorado:

```
PROCEDURE buscarord(var f:tarchivoelem;c:tclave;var
    elem:tElem; var encontrado:boolean);
{Efecto.  Si c está en f entonces encontrado=true y elem es el
    elemento buscado, en  otro caso encontrado=false}
Var
    ultimaclave:clave;
Begin
    encontrado:=false;
    ultimaclave:=cota superior de las claves;
    while not Eof(f) and ultimaclave>c do begin
        read(f,elem);
        if c:=elem.clave then
            encontrado:=true;
            ultimaclave:=elem.clave
    end; {while}
end; {buscarord}
```

ORDENACIÓN POR MEZCLA DE ARCHIVOS SECUENCIALES

- Cuando los archivos son pequeños una forma de ordenarlos consiste en almacenar su contenido en una array y emplear uno de los algoritmos ya presentados.

- Si no es así el método más frecuente es una variante no recursiva del algoritmo de mezcla.

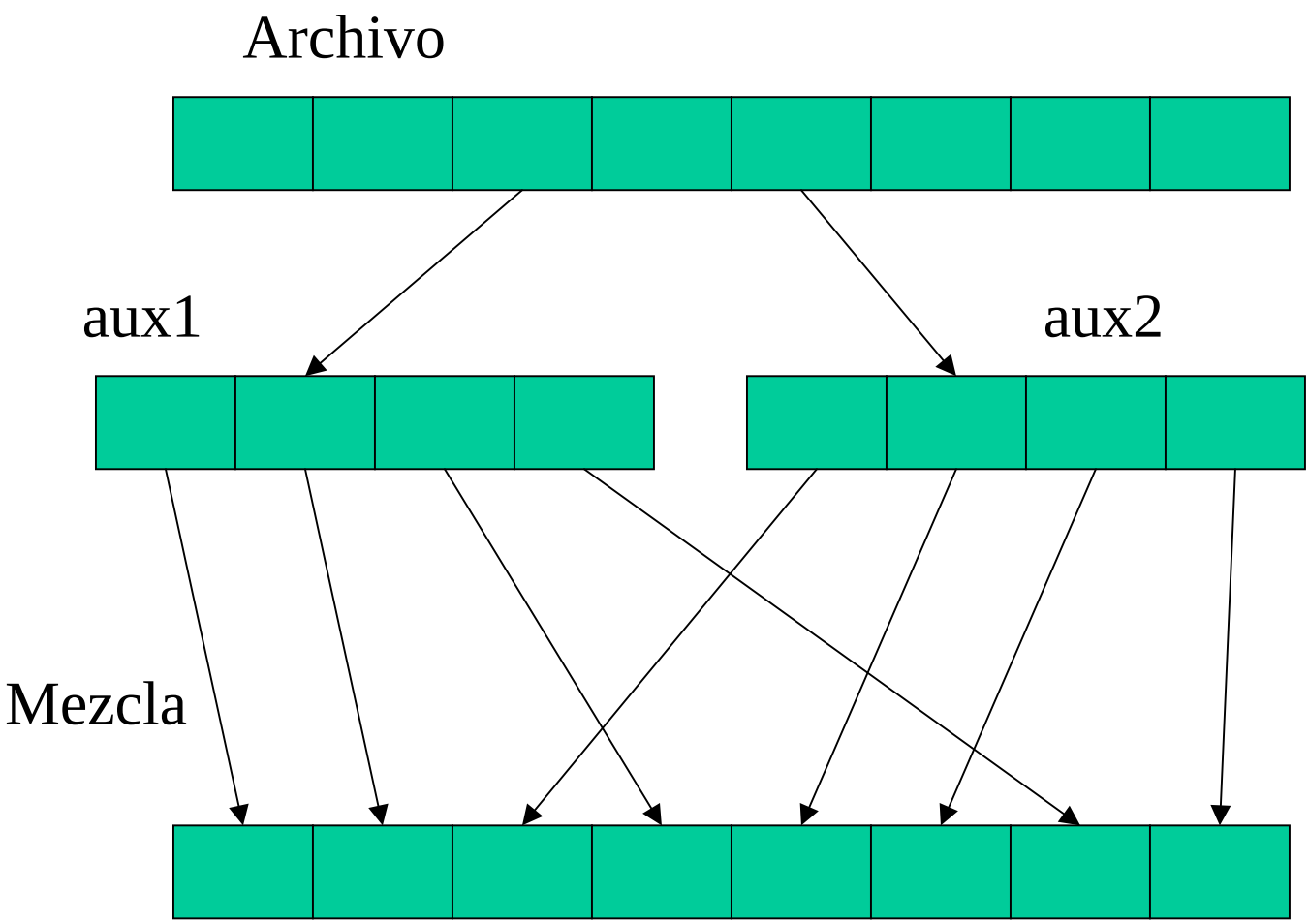
- Este consiste en dividir el archivo origen en dos archivos auxiliares para después mezclarlos ordenadamente sobre el propio archivo de origen, obteniendo de esta forma al menos pares de valores ordenados.

- Este proceso se debe repetir hasta que el archivo esté ordenado.

- Cuando en este proceso se emplean sólo dos archivos auxiliares la mezcla se denomina simple. Si se emplean más se denomina múltiple.

ACCIONES A SEGUIR

- Dividir el archivo original en otros dos
- Mezclar ordenadamente los dos archivos auxiliares sobre el archivo original.



MEZCLA DE LOS ARCHIVOS

- Si la primera componente de aux1 es menor que la de aux2, esta se escribe en el archivo final y se accede al siguiente valor del archivo aux1.

- En caso contrario se escribe en el archivo final la primera componente de aux2 avanzando en este archivo.

- En caso de llegar al final , i.e., al final del archivo aux1, se deberá proceder a copiar el resto de las componentes de aux2 sobre el archivo final.

ORDENACIÓN POR MEZCLA,
IMPLEMENTACIÓN EN PASCAL

```
PROCEDURE  Mezcla( var aux1,aux2,arch:tArchivo);
{Efecto. Arch=mezcla ordenada de aux1 y aux2}
VAR
    c1,c2:tcomponente;
    finarch1,finarch2:boolean;
    PROCEDURE leerelemdetectandofin(var arch:tArchivo; var
    comp:tcomponente; var finarch: boolean);
    begin
        finarch:=Eof(arch);
        if not finarch then
            read(arch,comp)
    end; {Leerelemdetectandofin}

    PROCEDURE Pasarelemdetectandofin (var archorigen,
    archdestino:tarchivo; var comp:tcomponente; var
    finarchorigen:boolean)
    begin
        write(archdestino,comp);
        Leerelemdetectandofin(archorigen,comp,finarchorigen);
    end; {Pasarelemdetectandofin}
```

ORDENACIÓN POR MEZCLA,
IMPLEMENTACIÓN EN PASCAL

```
begin {mezcla}
        reset(aux1);
        reset(aux2);
        rewrite(arch);
        leerelemdetectandofin(aux1,c1,finarch1);
        leerelemdetectandofin(aux2,c2,finarch2);
        while not finarch1 and not finarch2 do
            if c1<c2 then
                pasarelemdetectandofin(aux1,arch,c1,finarch1)
            else
                pasarelemdetectandofin(aux2,arch,c2,finarch2)
        while not finarch1 do
                pasarelemdetectandofin(aux1,arch,c1,finarch1)
        while not finarch2 do
                pasarelemdetectandofin(aux2,arch,c2,finarch2)
        close(arch);
        close(aux1);
        close(aux2)
end; {Mezcla}
```

DIVISIÓN DE LOS ARCHIVOS

- Para optimizar la tarea de dividir un archivo en dos auxiliares es conveniente tener en cuenta los tramos que ya estén ordenados.

- De esta forma conforme se va leyendo los sucesivos tramos de ‘arch’ que ya están ordenados así se van escribiendo en aux1 y aux2.

- En el momento que una componente en ‘arch’ esté desordenada se pasa a escribir al otro fichero auxiliar.

- Este método, que consigue que los dos ficheros generados posean el mismo números de tramos o difieran en uno, se denomina equilibrado.

- Además de esta forma se detecta si arch está ordenado ya que si es así sólo existirá un único tramo que se escribiría en aux1.

ALGORITMO DE DIVISIÓN,
IMPLEMENTACIÓN EN PASCAL

```
PROCEDURE  Division( var  arch,aux1,aux2:tArchivo; var
    esvacio2:boolean);
{Efecto. Arch=se divide en dos archivos: aux1 y aux2}
VAR
    valoractual,valoranterior:tcomponente;
    cambio:boolean; {conmuta la escritura entre aux1 y aux2}
    begin
        reset(arch);
        rewritet(aux1);
        rewrite(aux2);
        cambio:=true;
        esvacio2:=true;
        if not Eof(arch) then begin
                read(arch,valoractual);
                write(aux1,valoractual);
                valoranterior:=valoractual
        end; {if}
        while not Eof(arch) do begin
                read(arch,valoractual);
                if valoranterior>valoractual then
                        cambio:=not cambio;
```

ALGORITMO DE DIVISIÓN,
IMPLEMENTACIÓN EN PASCAL

```
                    if cambio then
                            write(aux1,valoractual)
                    else begin
                            write(aux2,valoractual);
                            esvacio2:=false
                    end; {else}
                    valoranterior:=valoractual;
        end; {while}
close(arch);
close(aux1);
close(aux2)
end; {Division}
```

ALGORITMO DE ORDENACIÓN,
IMPLEMENTACIÓN EN PASCAL

```
Program ordenacion_archivo;
TYPE
    tcomponente:char;
    tclave:integer;
    tElem=record
            clave:tclave;
            contenido:tcomponente
            end; {record}
    tArchivo=file of tElem;
VAR
            fichaux1,fichaux2,arch:tArchivo;
            vacioarch2:boolean;
PROCEDURE.....
.............................
Begin
    Division(arch,fichaux1,fichaux2,vacioarch2);
    while not vacioarch2 do begin
            Mezcla(fichaux1,fichaux2,arch);
            Division(arch,fichaux1,fichaux2,vacioarch2)
    end {while}
end; {Division}
```