Estructuras de Datos

Tema 3. Algoritmos de ordenación

Ordenación por Inserción

```
type TVector = array of TDato;

procedure OrdIns(var V: TVector);
var
  I,J : integer;
  Tmp : TDato;
begin
  for I := Low(V)+1 to High(V) do
  begin
    { Búsqueda de posición y desplazamiento }
    Tmp := V[I]; J := I-1;
    while (J >= Low(V)) and (V[J] > Tmp) do
    begin
      V[J+1] := V[J];
      J := J-1
    end;
    V[J+1] := Tmp; { Inserción }
  end
end;
```

Propiedades Ordenación por Inserción

- Eficiencia $O(n^2)$
  - Espacio $O(1)$
  - Peor caso (vector orden inverso): $n^2/2 + O(n)$
  - Mejor caso (vector ordenado): $\Theta(n)$
  - Promedio: $n^2/4 + O(n)$
  - Las fórmulas son iguales para movimientos y comparaciones
- Método universal
- Acceso secuencial: Sólo si el acceso es bidireccional (por ejemplo listas doblemente enlazadas)
- Estable
- Adaptativo
- Sobre el propio vector

Ordenación por Selección

```
type TVector = array of TDato;

procedure OrdSel(var V: TVector);
var
  I,J,Jmin : integer; { Tmp : TDato; }
begin
  for I := Low(V) to High(V)-1 do
  begin
    { Búsqueda del menor de zona no ordenada }
    Jmin := I;
    for J := I+1 to High(V) do
      if V[J] < V[Jmin] then Jmin := J;
    { Intercambio menor <-> primero zona no ord. }
    V[I] ⇔ V[Jmin]
  end
end;
```

Propiedades Ordenación por Selección

- Eficiencia $\Theta(n^2)$ [$\Theta(n^2)$ comparaciones, $\Theta(n)$ movimientos]
  - Espacio $O(1)$
  - Siempre hace el mismo número de operaciones
  - Comparaciones: $n^2/2 + O(n)$
  - Movimientos: $3n$
- Método universal
- Acceso secuencial: Sólo si el acceso permite marcas (marcar un punto y poder volver a él)
- No Estable (V[I] puede saltar por delante de elementos iguales)
- No adaptativo
- Sobre el propio vector

Ordenación por Intercambio (burbuja)

```
type TVector = array of TDato;

procedure OrdBur(var V: TVector);
var
  I,J : integer; { Tmp : TDato; }
begin
  for I := Low(V) to High(V)-1 do
    for J := High(V) downto I+1 do
      if V[J] > V[J+1] then
         V[J] ⇔ V[J+1]
end;
```

Propiedades Ordenación Burbuja

- Eficiencia $\Theta(n^2)$
  - Espacio $O(1)$
  - Comparaciones: $n^2/2 + O(n)$
  - Movimientos:
    - 0 mejor caso
    - $3n^2/4 + O(n)$ promedio
    - $3n^2/2 + O(n)$ peor caso
- Método universal
- Acceso secuencial: Sólo si el acceso es bidireccional
- Estable
- No adaptativo
- Sobre el propio vector

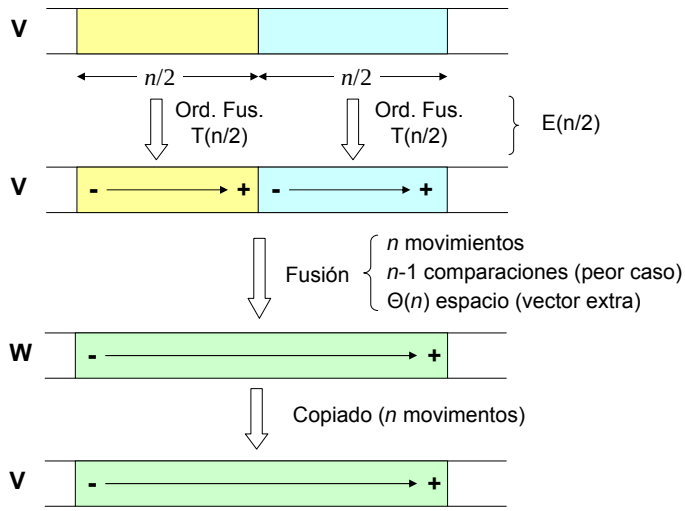
Ordenación por Fusión

```
type
  TDato = record Clave: TClave; ... end;
  TVector = array of TDato;

procedure OrdFus(var V: TVector);  { Se supone que Low(V) = 0 }
var W : TVector; { Vector temporal }
begin
  SetLength(W,Length(V));
  OrdFusRec(V,W,0,High(V))
end;

procedure OrdFusRec(var V,W: TVector; Ini,Fin: integer);
{ Ordena V[Ini..Fin] }
var Med,I : integer;
begin
  if Ini < Fin then
  begin
    Med := (Ini+Fin) div 2;
    OrdFusRec(V,W,Ini,Med);     { Ordenación primera mitad }
    OrdFusRec(V,W,Med+1,Fin);   { Ordenación segunda mitad }
    Fusion(V,W,Ini,Med,Fin);    { Fusionar mitades en W }
    for I := Ini to Fin do V[I] := W[I] { Copiar a V }
  end { else caso_base }
end;
```

Algoritmo de Fusión

```
procedure Fusion(var V,W: TVector; Ini,Med,Fin: integer);
{ Fusiona V[Ini..Med] y V[Med+1..Fin] en W[Ini..Fin] }
var Ia,Ib,Ic : integer;
begin
  { "Extraer" mínimos y llevarlos a W }
  Ia := Ini; Ib := Med+1; Ic := Ini;
  while (Ia <= Med) and (Ib <= Fin) do
    if V[Ia].Clave < V[Ib].Clave then
    begin
      W[Ic] := V[Ia]; Inc(Ia); Inc(Ic)
    end else begin
      W[Ic] := V[Ib]; Inc(Ib); Inc(Ic)
    end;
  { Copiar zona no vacía a W }
  while Ia <= Med do
  begin
    W[Ic] := V[Ia]; Inc(Ia); Inc(Ic)
  end;
  while Ib <= Fin do
  begin
    W[Ic] := V[Ib]; Inc(Ib); Inc(Ic)
  end;
end;
```


Análisis Ordenación por Fusión



Propiedades Ordenación por Fusión

- Eficiencia $\Theta(n \log n)$
  - $2n \log_2 n$ movimientos
  - $n \log_2 n - n$ comparaciones (peor caso ≈ caso promedio)
  - n datos (vector extra) + $\Theta(\log n)$ espacio adicional
- Método universal
- Fácilmente adaptable a acceso secuencial (separando ambas mitades en estructuras distintas)
- Estable
- No adaptativo
- Sobre el propio vector (no se puede usar el vector extra para devolver el resultado)

Ordenación Rápida

```
procedure OrdRap(var V: TVector);
var I,J : integer;
begin
  { Desordenar vector (Knuth shuffle algorithm) }
  for I := Low(V) to High(V)-1 do
    V[I] ⇔ V[Low(V)+Random(High(V)-I+1)]
  { Ordenación recursiva }
  OrdRapRec(V,Low(V),High(V))
end;

procedure OrdRapRec(var V: TVector; Ini,Fin: integer);
{ Ordena V[Ini..Fin] }
var Fin_Men,Ini_May : integer;
begin
  if Ini < Fin then
  begin
    Particion(V,Ini,Fin,Fin_Men,Ini_May);
    OrdRapRec(V,W,Ini,Fin_Men);
    OrdRapRec(V,W,Ini_May,Fin);
  end { else caso_base }
end;
```

Opcional

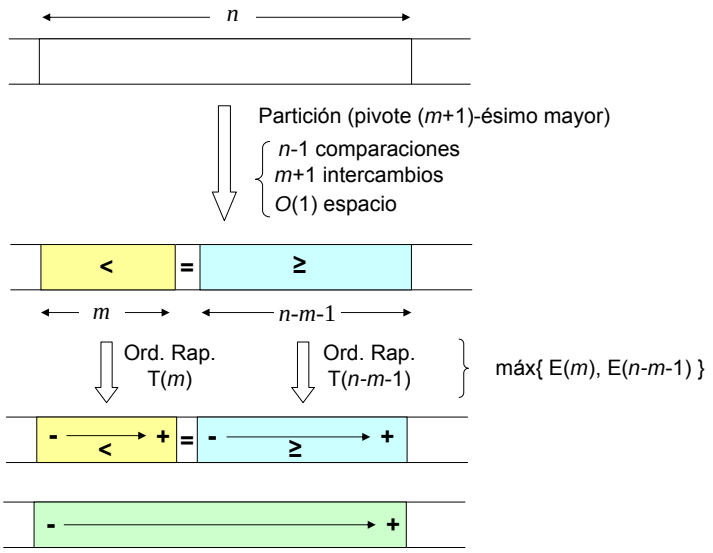
Primer método de partición

```
procedure Partición(var V: TVector; Ini,Fin: integer;
                    var Fin_Men, Ini_May: integer);
{ Reorganiza V[Ini..Fin] de manera que termine organizado en tres zonas:
  · V[Ini..Fin_Men] contiene elementos menores o iguales al pivote.
  · V[Fin_Men+1..Ini_May-1] contiene elementos iguales al pivote.
  · V[Ini_May..Fin] contiene elementos mayores o iguales al pivote.
  Ninguna zona se extiende en todo V[Ini..Fin] }
var
  Izda,Dcha : integer; Pivote : TClave;
begin
  Pivote := V[Ini].Clave; { Hay otras alternativas a elección de pivote }
  Izda := Ini; Dcha := Fin;
  while Izda <= Dcha do
  begin { Invariante: V[Ini..Izda-1] <= Pivote, V[Dcha+1..Fin] >= Pivote }
    while V[Izda].Clave < Pivote do Inc(Izda);
    while V[Dcha].Clave > Pivote do Dec(Dcha);
    if Izda <= Dcha then
    begin
      V[Izda] ⇔ V[Dcha];
      Inc(Izda); Dec(Dcha)
    end
  end;
  Fin_Men := Dcha; Ini_May := Izda
end;
```


Segundo método de partición

```
procedure Partición(var V: TVector; Ini,Fin: integer;
                    var Fin_Men, Ini_May: integer);
{ Reorganiza V[Ini..Fin] de manera que termine organizado en tres zonas:
  · V[Ini..Fin_Men] contiene elementos menores que el pivote.
  · V[Fin_Men+1] contiene el pivote.
  · V[Ini_May..Fin] contiene elementos mayores o iguales al pivote.
  IniMay = Fin_Men+2 }
var
  I,Lim : integer;
begin
  { Se toma como pivote el 1er elemento, V[Ini] (hay otras alternativas) }
  Lim := Ini+1;
  for I := Ini+1 to Fin do
    { Invariante: V[Ini+1..Lim-1] < Pivote, V[Lim..I-1] >= Pivote }
    if V[I].Clave < V[Ini].Clave then
    begin
      V[I] ⇔ V[Lim];
      Inc(Lim)
    end; { if-for }
  V[I] ⇔ V[Lim-1]; { Pivote entre menores y mayores/iguales }
  Fin_Men := Lim-2; Ini_May := Lim
end;
```

Análisis Ordenación Rápida (2º método part.)



Mejor caso ordenación rápida

- Se da cuando las particiones son equilibradas ($m \approx n/2$)
- $T(n) = 2T(n/2) + \Theta(n)$, $E(n) = E(n/2) + O(1)$
- Comparaciones: $n \log_2 n + O(n)$ [igual a ord. fusión]
- Movimientos: $1.5\, n \log_2 n + O(n)$ [mejor que ord. fusión]
- Espacio: $\Theta(\log n)$ [mucho mejor que ord. fusión]

Peor caso ordenación rápida

- Se da cuando las particiones son desequilibradas: Una de las zonas está vacía ($m = 0$ ó $m = n-1$)
- $T(n) = T(n-1) + \Theta(n)$, $E(n) = E(n-1) + O(1)$
- Comparaciones: $n^2/2 + O(n)$
- Movimientos: $\Theta(n)$ si $m = 0$, $1.5\ n^2 + O(n)$ si $m = n-1$
- Espacio: $\Theta(n)$
- La ordenación rápida es un algoritmo $O(n^2)$
- Tipos de vectores que provocan el peor caso:
  - Pivote primero o último: Vector ordenado o en orden inverso
  - Pivote elem. medio: Vector con elementos en orden creciente hasta la mitad y decreciente a partir de entonces, o al revés.
  - Pivote al azar: La probabilidad de caer en el peor caso decrece exponencialmente.

Caso Promedio ordenación rápida

- Relación de recurrencia general (2º método partición)

$$T(n,m) = T(m,\cdot) + T(n-m-1,\cdot) + f(n,m)$$

- Donde m es el número de elementos menores que el pivote
- $f(n,m)$ son las operaciones no recursivas (partición):
  - $n-1$ comparaciones
  - $m+2$ intercambios
- Si el pivote es un elemento cualquiera de la zona, entonces cualquier valor de $m \in [0..n-1]$ es equiprobable.
- Número de operaciones promedio:

$$\widehat{T}(n) = \frac{1}{n}\sum_{m=0}^{n-1} T(n,m)$$

Caso Promedio ordenación rápida

- Aplicandolo a la relación de recurrencia (comparaciones):

$$\hat{T}(n)=\frac{1}{n}\sum_{m=0}^{n-1}\left(\hat{T}(m)+\hat{T}(n-m-1)+n-1\right)$$

- Operando:

$$\hat{T}(n)=\frac{2}{n}\sum_{m=0}^{n-1}\hat{T}(m)+n-1$$

$$n\cdot\hat{T}(n)=2\sum_{m=0}^{n-1}\hat{T}(m)+n\cdot(n-1) \qquad \text{(eq. A)}$$

$$(n-1)\cdot\hat{T}(n-1)=2\sum_{m=0}^{n-2}\hat{T}(m)+(n-1)\cdot(n-2) \qquad \text{(eq. B)}$$

Caso Promedio ordenación rápida

- Restando las ecuaciones A y B:

$$n\cdot\hat{T}(n)-(n-1)\cdot\hat{T}(n-1)=2\cdot\hat{T}(n-1)+2(n-1)$$

$$n\cdot\hat{T}(n)=(n+1)\cdot\hat{T}(n-1)+2(n-1)$$

$$\frac{\hat{T}(n)}{n+1}=\frac{\hat{T}(n-1)}{n}+\frac{2(n-1)}{n\cdot(n+1)}$$

$$\frac{\hat{T}(n)}{n+1}=\frac{\hat{T}(n-1)}{n}+\frac{4}{n}-\frac{2}{n+1}$$

$$\hat{T}(n)=n\cdot\ln n+O(n)=1.44\cdot n\cdot\log_2 n$$

Propiedades ordenación rápida

- Eficiencia $O(n^2)$
  - Peor caso: $\Theta(n^2)$ tiempo, $\Theta(n)$ espacio.
  - Mejor caso: $\Theta(n \log n)$ tiempo, $\Theta(\log n)$ espacio
  - Promedio: $\Theta(n \log n)$ tiempo, $\Theta(\log n)$ espacio
  - El tiempo promedio sólo es un 40% mayor que el mejor
- Método universal
- Acceso secuencial: No.
- No Estable
- No adaptativo → Antiadaptativo
- Sobre el propio vector

Ordenación por Montículos

```
procedure Partición(var V: TVector); { Se supone que Low(V) = 0 }
var Lim,I,P,H : integer;
begin
  Lim := High(V);
  for I := (Lim-1) div 2 downto 0 do
  begin
    P := I; H := 2*I+1;
    if (H < Lim) and (V[H] < V[H+1]) then Inc(H);
    while (H <= Lim) and (V[P] < V[H]) do
    begin
      V[P] ⇔ V[H]; P := H; H := 2*P+1;
      if (H < Lim) and (V[H] < V[H+1]) then Inc(H)
    end
  end;
  for I := Lim-1 downto 0 do
  begin
    V[0] ⇔ V[I+1]; P := 0; H := 1;
    if (H < I) and (V[H] < V[H+1]) then Inc(H);
    while (H <= I) and (V[P] < V[H]) do
    begin
      V[P] ⇔ V[H]; P := H; H := 2*P+1;
      if (H < I) and (V[H] < V[H+1]) then Inc(H)
    end
  end
end;
```

Propiedades ordenación por monticulos

- Eficiencia Θ(*n* log *n*)
  - Espacio *O*(1)
  - No recursiva
  - Entre 2 y 5 veces más lenta que la ordenación rápida
- Método universal
- Acceso secuencial: No.
- No Estable
- No adaptativo
- Sobre el propio vector

Vector de índices

- Hay ocasiones en que no deseamos modificar el vector original, sino obtener un vector de índices que permita un recorrido ordenado del vector.
- Entrada: Vector de datos V[1..n]. Salida: Vector de índices W[1..n], de manera que W[i] sea el índice del *i*-ésimo menor elemento de V.
- Modificación de los algoritmos:
  - En comparaciones, sustituir V[i] por V[W[i]]
  - En intercambios o movimientos, sustituir V por W.

```
type
  TVector = array of TDato;
  TVecInd = array of integer;

procedure OrdBur(const V: TVector; var W: TVecInd);
var I,J : integer; { Tmp : TDato; }
begin
  for I := Low(V) to High(V) do W[I] := I;
  for I := Low(V) to High(V)-1 do
    for J := High(V) downto I+1 do
      if V[W[J]] > V[W[J+1]] then W[J] ⇔ W[J+1]
end;
```