Algoritmos sobre secuencias y conjuntos de datos

Algoritmos de ordenación

Alberto Valderruten

Dept. de Computación, Universidade da Coruña

alberto.valderruten@udc.es

Ordenación por Inserción (1)

```
procedimiento Ordenación por Inserción (var T[1..n])
    para i:=2 hasta n hacer
        x:=T[i];
        j:=i-1;
        mientras j>0 y T[j]>x hacer
            T[j+1]:=T[j];
            j:=j-1
        fin mientras;
        T[j+1]:=x
    fin para
fin procedimiento
```

$$\begin{cases} \text{peor caso: max } i \text{ comparaciones para cada } i \Rightarrow \sum_{i=2}^{n} i = \Theta(n^2) \\ \text{mejor caso: min 1 comparación para cada } i \text{ (entrada ordenada)} \Rightarrow \Theta(n) \\ \text{¿caso medio?} \end{cases}$$

$\Rightarrow$ **Cota inferior** (Ω) **para los algoritmos de ordenación que intercambian elementos adyacentes**: inserción, selección, burbuja...

Ordenación por Inserción (2)

Observación: ¿Inserción intercambia elementos adyacentes?

$\rightarrow$ *abstracción*

Sea $T[1..n]$ la entrada del algoritmo:
Definición: *inversión* $\equiv$ cualquier $(i,j) : i < j \wedge T[i] > T[j]$

Ejemplo:

3	1	4	1	5	9	2	6	5	3

$\rightarrow$ inversiones: $(3,1),(3,1),(3,2),\ldots,(5,3)$

Sea I el número de inversiones: *"medida del desorden"*

En el ejemplo, $I = 15$

Intercambiar 2 elementos adyacentes elimina una inversión

En el ejemplo, $I = 15 \Rightarrow$ 15 intercambios para ordenar

hasta $I = 0 \equiv$ vector ordenado

$\Rightarrow$ Inserción = $O(I+n)$

Ordenación por Inserción (3)

- Inserción = $\begin{cases} O(n) \text{ si } I = 0 \text{ (mejor caso)} \ \vee \ I = O(n) \ (*) \\ O(n^2) \text{ si } I = O(n^2) \text{ (peor caso)} \end{cases}$

  (*) Nuevo resultado

- Caso medio $\Rightarrow$ ¿I_{medio} en una entrada?

  Hipótesis: $\begin{cases} \text{sin duplicados} \\ \text{permutaciones equiprobables} \end{cases}$

  $\Rightarrow$ ¿I_{medio} en una permutación?

  Teorema: $I_{medio} = n(n-1)/4$

  Demostración: sean $T[1..n]$ el vector, $T_i[1..n]$ el vector *inverso*:

  cualquier (x, y) es inversión en T o en T_i

  Nº total de (x, y) con $y > x$

  $= (n-1) + (n-2) + \ldots + 1 = \sum_{i=1}^{n-1} i = n(n-1)/2$

  equiprobabilidad $\Rightarrow$ T_{medio} tiene la mitad de esas inversiones

  □

  $\Rightarrow$ Caso medio de Inserción: $I = O(n^2) \Rightarrow \boxed{T(n) = O(n^2)}$

Ordenación por Inserción (4)

Teorema: cualquier algoritmo que ordene intercambiando elementos adyacentes requiere un tiempo $\Omega(n^2)$ en el caso medio.

Demostración:

$I_{medio} = n(n-1)/4 = \Omega(n^2)$

cada intercambio elimina sólo una inversión

$\Rightarrow \Omega(n^2)$ intercambios

□

¿Cómo conseguir un trabajo $o(n^2) \equiv \neg\Omega(n^2) \equiv$ "bajar de n^2"?

- *Intercambiar elementos alejados*

  $\Rightarrow$ deshacer varias inversiones a la vez:

Ordenación de Shell

Ordenación de Shell (1)

- 1er algoritmo de ordenación que baja de $O(n^2)$ en el peor caso
- Secuencia de ***incrementos*** $\equiv$ distancias para intercambios:
  Naturales, ordenados descendentemente: $h_t, \ldots h_k, h_{k-1}, \ldots h_1 = 1$
- t iteraciones: en la iteración k utiliza el incremento h_k
  Postcondición $= \{\forall i, T[i] \leq T[i + h_k]\}$
  $\equiv$ los elementos separados por h_k posiciones están ordenados
  $\rightarrow$ **vector** h_k***-ordenado***
  Trabajo de la iteración k: h_k ordenaciones por Inserción
- **Propiedad**:

  un vector h_k-ordenado que se h_{k-1}-ordena
  sigue estando h_k-ordenado

- Problema: ¿secuencia óptima de incrementos?
  incrementos de Shell: $h_t = \lfloor n/2 \rfloor$, $h_k = \lfloor h_{k+1}/2 \rfloor$

Ordenación de Shell con incrementos de Shell

```
procedimiento Ordenación de Shell (var T[1..n])
    incremento := n;
    repetir
        incremento := incremento div 2;
        para i := incremento+1 hasta n hacer
            tmp := T[i];
            j := i;
            seguir := cierto;
            mientras j-incremento > 0 y seguir hacer
                si tmp < T[j-incremento] entonces
                    T[j] := T[j-incremento];
                    j := j-incremento
                sino seguir := falso ;
            T[j] := tmp
    hasta incremento = 1
fin procedimiento
```

Ordenación de Shell (3)

- Otros incrementos también funcionan.

Ejemplo: $n = 13 \rightarrow 5, 3, 1$ en vez de Shell $(6, 3, 1)$

ini	81	94	11	96	12	35	17	95	28	58	41	75	15
(5)	35	17	**11**	28	12	41	75	**15**	96	58	81	94	**95**
(3)	28	12	11	35	15	41	58	17	94	75	81	96	95
(1)	11	12	15	17	28	35	41	58	75	81	94	95	96

- **Teorema**: *Shell con incrementos de Shell* es $\Theta(n^2)$ (peor caso).

Demostración:

1. ¿$\Omega(n^2)$?

   $n = 2^k \Rightarrow$ incrementos pares excepto el último $(= 1)$

   Peor situación: los $n/2$ mayores están en las posiciones pares

   Ejemplo (el más favorable dentro de ésta situación):

1	9	2	10	3	11	4	12	5	13	6	14	7	15	8	16

   ¿Más favorable? $\rightarrow$ 8, 4 y 2-ordenado; todo el trabajo: 1-ordenar

Ordenación de Shell (4)

Demostración (cont.):

El i-ésimo menor está en la posición $2i-1$, $i \leq n/2$

(ej: 8 en posición 15)

$\rightarrow$ hay que moverlo $i-1$ veces (ej: 8 $\rightarrow$ 7 desplazamientos)

$\Rightarrow$ colocar menores: $\sum_{i=1}^{n/2} i-1 = \Omega(n^2)$

2. ¿$O(n^2)$?

Trabajo realizado en iteración k con el incremento h_k:

h_k ordenaciones por Inserción sobre n/h_k elementos cada una

$= h_k O((n/h_k)^2) = O(h_k(n/h_k)^2) = O(n^2/h_k)$

En el conjunto de iteraciones del algoritmo:

$T(n) = O(\sum_{i=1}^{t} n^2/h_i) = O(n^2 \sum_{i=1}^{t} 1/h_i) = O(n^2)$

□

Observación 1: $\neq O(n^3)$ ($\leftarrow$ "3 bucles anidados")

Observación 2: Bajar de $O(n^2)$ en peor caso? $\rightarrow$ *otros incrementos*

Ordenación de Shell (5)

- **Otros incrementos**:

incrementos	peor caso	caso medio
Hibbard: $1,3,7..,2^k-1$	$\Theta(n^{3/2})$ (teo[b])	$O(n^{5/4})$ (sim[c])
Sedgewick[a]: $1,5,19,41,109...$	$O(n^{4/3})$ (sim[c])	$O(n^{7/6})$ (sim[c])

Tabla: Ordenación de Shell con distintos incrementos

[a] varias secuencias de incrementos
[b] demostración formal (teorema)
[c] comprobación empírica (simulaciones)

- **Conclusión**:
  código sencillo y resultados muy buenos en la práctica

Ordenación por Montículos (*heapsort*)

```
procedimiento Ordenación por Montículos (var T[1..n])
    Crear montículo (T, M);
    para i := 1 hasta n hacer
        T[n-i+1] := Obtener mayor valor (M);
        Eliminar mayor valor(M)
    fin para
fin procedimiento
```

Para crear un montículo a partir de un vector:

```
procedimiento Crear montículo (V[1..n], var M)
{ V[1..n]: entrada: vector con cuyos datos se construirá el montículo
 M: entrada/salida: montículo a crear }
    Copiar V[1..n] en M[1..n];
    para i := n div 2 hasta 1 paso -1 hacer
        hundir(M,i)
    fin para
fin procedimiento
```

Ordenación por Montículos (2)

- ¿Cómo mejorar la complejidad espacial (y algo $T(n)$)?
  $\rightarrow$ *utilizar la misma estructura*. Ejemplo:

entrada	4	3	7	9	6	5	8
Crear Mont.	9	6	8	3	4	5	7
Eliminar(9)	8	6	7	3	4	5	**9**
Eliminar(8)	7	6	5	3	4	**8**	9
Eliminar(7)	6	4	5	3	**7**	8	9
Eliminar(6)	5	4	3	**6**	7	8	9
Eliminar(5)	4	3	**5**	6	7	8	9
Eliminar(4)	3	**4**	5	6	7	8	9
Eliminar(3)	**3**	4	5	6	7	8	9

- **Teorema**: La ordenación por montículos es $O(nlogn)$
  Demostración:
  Crear Montículo es $O(n)$, y n Eliminar es $O(nlogn)$
  □
- **Observación**: Incluso en el peor caso es $O(nlogn)$, pero en la práctica es más lento que Shell con incrementos de Sedgewick.

Ordenación por Fusión: procedimiento Fusión

```
procedimiento Fusión ( var T[Izda..Dcha], Centro:Izda..Dcha )
{fusiona los subvectores ordenados T[Izda..Centro] y T[Centro+1..Dcha] en
 en T[Izda..Dcha], utilizando un vector auxiliar Aux[Izda..Dcha]}
    i := Izda ; j := Centro+1 ; k := Izda ;
    {i, j y k recorren T[Izda..Centro], T[Centro+1..Dcha]
     y Aux[Izda..Dcha] respectivamente}
    mientras i <= Centro y j <= Dcha hacer
        si T[i] <= T[j] entonces Aux[k] := T[i] ; i := i+1
        sino Aux[k] := T[j] ; j := j+1 ;
        k := k+1 ;
    {copia elementos restantes del subvector sin recorrer}
    mientras i <= Centro hacer
        Aux[k] := T[i] ; i := i+1 ; k := k+1 ;
    mientras j <= Dcha hacer
        Aux[k] := T[j] ; j := j+1 ; k := k+1 ;
    para k := Izda hasta Dcha hacer
        T[k] := Aux[k]
fin procedimiento
```

Ordenación por Fusión (2)

- *mergesort*
- O bien, ordenación *por intercalación*.
- Utiliza un algoritmo de **Fusión** de un vector cuyas *mitades* están ordenadas para obtener un vector ordenado.
- El procedimiento Fusión es lineal (n comparaciones).
- Ordenación: algoritmo Divide y Vencerás
  - Divide el problema en 2 *mitades*, que
  - se resuelven recursivamente;
  - *Fusiona* las mitades ordenadas en un vector ordenado.
- **Mejora**: Ordenación por Inserción para *vectores pequeños*:
  $n <$ *umbral*, que se determina empíricamente.

Ordenación por Fusión (3)

```
procedimiento Ordenación por Fusión Recursivo ( var T[Izda..Dcha] )
    si Izda+UMBRAL < Dcha entonces
        Centro := ( Izda+Dcha ) div 2 ;
        Ordenación por Fusión Recursivo ( T[Izda..Centro] ) ;
        Ordenación por Fusión Recursivo ( T[Centro+1..Dcha] ) ;
        Fusión ( T[Izda..Dcha], Centro )
    sino Ordenación por Inserción ( T[Izda..Dcha] )
fin procedimiento

procedimiento Ordenación por Fusión ( var T[1..n] )
    Ordenación por Fusión Recursivo ( T[1..n] );
fin procedimiento
```

Ordenación por Fusión (4)

- **Análisis** de la versión puramente recursiva (UMBRAL $= 0$):
  $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + O(n)$ (Fusión)
  $$n = 2^k \Rightarrow \begin{cases} T(1) & = O(1) & = 1 \\ T(n) & = 2T(n/2) + O(n) & = 2T(n/2) + n, n > 1 \end{cases}$$
  Teorema Divide y Vencerás: $l = 2, b = 2, c = 1, k = 1, n_0 = 1$
  caso $l = b^k \Rightarrow \boxed{T(n) = \Theta(n \log n)}$
- Podría mejorarse la complejidad espacial ($= 2n$: vector auxiliar)
  $\rightarrow$ El algoritmo adecuado es *quicksort*
- **Observación**:
  Importancia de *balancear* los subcasos en Divide y Vencerás:
  Si llamadas recursivas con vectores de tamaño $n-1$ y 1
  $\Rightarrow T(n) = T(n-1) + T(1) + n = O(n^2)$
  ... pero ya no sería Ordenación por Fusión.

Ordenación Rápida (*quicksort*) [Hoare]

- Paradigma de *Divide y Vencerás*.
  Con respecto a Fusión:
  - más trabajo para construir las subinstancias (pivote...),
  - pero trabajo nulo para combinar las soluciones.
- **Selección del pivote** en $T[i..j]$:
  - Objetivo: obtener una partición lo más balanceada posible
    $\Rightarrow$ ¿**Mediana**? Inviable
  - Usar el **primer valor** del vector ($T[i]$):
    Ok si la entrada es aleatoria.
    Pero elección muy desafortunada con entradas ordenadas o parcialmente ordenadas (caso bastante frecuente)
    $\rightarrow O(n^2)$ para no hacer nada...
  - Usar un valor elegido al azar (**pivote aleatorio**):
    Más seguro, evita el peor caso detectado antes, pero depende del generador de números aleatorios (eficiencia vs. coste).
  - Usar la **mediana de 3 valores**: $T[i], T[j], T[(i+j)div2]$

Ordenación Rápida (2)

- **Selección del pivote** (cont.): Mediana de $T[i], T[j], T[(i+j)div2]$

```
procedimiento Mediana 3 ( var T[i..j] )
    centro := ( i+j ) div 2 ;
    si T[i] > T[centro] entonces intercambiar ( T[i], T[centro] ) ;
    si T[i] > T[j] entonces intercambiar ( T[i], T[j] ) ;
    si T[centro] > T[j] entonces intercambiar ( T[centro], T[j] ) ;
    intercambiar ( T[centro], T[j-1] )
fin procedimiento
```

- **Estrategia de partición**:
  Ejemplo: $\rightarrow$ Hipótesis: sin duplicados

entrada		8	1	4	9	6	3	5	2	7	0
Mediana 3		0	1	4	9	**7**	3	5	2	6	8

Ordenación Rápida (3)

```
procedimiento Qsort ( var T[i..j] )
    si i+UMBRAL <= j entonces
        Mediana 3 ( T[i..j] ) ;
        pivote := T[j-1] ; k := i ; m := j-1 ;    {sólo con Mediana 3}
        repetir
            repetir k := k+1 hasta T[k] >= pivote ;
            repetir m := m-1 hasta T[m] <= pivote ;
            intercambiar ( T[k], T[m] )
        hasta m <= k ;
        intercambiar ( T[k], T[m] ) ;  {deshace el último intercambio}
        intercambiar ( T[k], T[j-1] ) ;          {pivote en posición k}
        Qsort ( T[i..k-1] ) ;
        Qsort ( T[k+1..j] )
fin procedimiento

procedimiento Quicksort ( var T[1..n] )
    Qsort ( T[1..n] ) ;
    Ordenación por Inserción ( T[1..n] )
fin procedimiento
```

Ordenación Rápida (4)

- **Estrategia de partición**:
  Ejemplo (Cont.):

entrada	8	1	4	9	6	3	5	2	7		0
Mediana 3	0	1	4	9	**7**	3	5	2	6		8
iteración 1	0	1	4	**2**	7	3	5	**9**	6		8
iteración 2	0	1	4	2	**5**	3	**7**	9	6		8
iteración 3	0	1	4	2	5	**7**	**3**	9	6		8
corrección	0	1	4	2	5	**3**	**7**	9	6		8
final	0	1	4	2	5	3	6	9	**7**		8

- **Observaciones** sobre intercambiar:
  - Mejor deshacer un intercambio que incluir un test en el bucle
  - Evitar llamadas a funciones
- La estrategia de partición depende de la selección del pivote
  [Brassard & Bratley] $\rightarrow$ Mediana 3 sin sentido si UMBRAL < 3
  (de hecho, el algoritmo propuesto falla; ejercicio: corregirlo)
- **Ejercicio**: escribir Quicksort con pivote aleatorio

Ordenación Rápida (5)

- **Considerar valores repetidos**:
  $\leftrightarrow$ ¿Parar o no parar cuando $T[k] = pivote$ o $T[m] = pivote$?
  - Uno de los índices se detiene y el otro no:
    $\Rightarrow$ los valores idénticos al pivote van al mismo lado
    $\equiv$ desbalanceo
    Caso extremo (*): todos los valores son idénticos $\Rightarrow O(n^2)$
    $\Rightarrow$ *Hacer lo mismo*
  - Parar los 2 índices: (*) $\rightarrow$ muchos intercambios inútiles,
    pero los índices se cruzan en la mitad
    $\equiv$ partición balanceada, $O(nlogn)$
  - No parar ninguno: (*) $\rightarrow$ evitar que sobrepasen $[i..j]$,
    pero sobretodo no se produce ningún intercambio
    $\equiv$ desbalanceo, $O(n^2)$

Ordenación Rápida (6)

- **Vectores pequeños**:
  - Recursividad $\leftrightarrow$ muchas llamadas (en las hojas) con vectores pequeños, que serán mejor tratados por Inserción, si nos aseguramos que *I* es $O(n)$.
  - $\Rightarrow$ Utilizar un **umbral** para determinar los casos base.
    Su valor óptimo se encuentra empíricamente:
    entre $n = 12$ y $n = 15$
  - Otra mejora: hacer una única llamada a Inserción con todo el vector:
    El *I* total es igual a la suma de los *I* locales.

Ordenación Rápida (7)

- **Análisis**: pivote aleatorio y sin umbral

$$\Rightarrow \begin{cases} T(0) = T(1) = 1 \\ T(n) = T(z) + T(n-z-1) + cn, n > 1 \end{cases}$$

  - **Peor caso**: p *siempre* es el menor o el mayor elemento

$$\Rightarrow T(n) = T(n-1) + cn, n > 1$$

$$\Rightarrow \boxed{T(n) = O(n^2)}$$

  - **Mejor caso**: p *siempre* coincide con la mediana

$$\Rightarrow T(n) = 2T(n/2) + cn, n > 1$$

$$\Rightarrow \boxed{T(n) = O(n\log n)}$$

Ordenación Rápida (8)

- **Análisis** (Cont.):
  - **Caso medio**:

    Sea z: tamaño de la parte izquierda;

    Cada valor posible para z $(0, 1, 2, \ldots, n-1)$

    es *equiprobable*: $p = 1/n$

    $\leftrightarrow T(z) = T(n-z-1) = 1/n \sum_{x=0}^{n-1} T(x)$

    $\Rightarrow T(n) = 2/n[\sum_{x=0}^{n-1} T(x)] + cn, n > 1$

    $\Rightarrow \boxed{T(n) = O(nlogn)}$

    (cálculo similar al de la profundidad media de un ABB = $O(logn)$)

- **Algoritmos aleatorios**: El peor caso ya no es una entrada particular, sino que depende de la secuencia de números aleatorios obtenida durante la ejecución.

  ¿Mejor caso? ¿Caso medio?

  $\rightarrow$ Otros problemas: calidad de los números aleatorios...