Fundamentos de Análise e Deseño de Algoritmos e Estruturas de Dados

---

TEMA 3: ALGORITMOS DE ORDENACIÓN.

3.1.- Ordenación por inserción.

Executa-se nun tempo $O(n^2)$ por mor dos dous bucles aniñados. Primeiro bucle: 2 a n (for). Se nos fixamos no test que se fai para acceder ao segundo bucle while (j>0 e T[j]>x) vai-se realizar un máximo de i veces para cada i.

O test repite-se $2+3+...+n = \Sigma(i=2..n)i$

e como $1+2+...+n = \Sigma(i=1..n)i = n\cdot(n+1)/2 \approx n^2/2$

ten-se que $\Sigma(i=2..n)i = \theta(n^2)$ (O e Ω).

No caso particular no que a entrada está ordenada, o test do segundo bucle só se executa unha vez e falla, entón é O(n).

En canto ao caso promédio, é $\theta(n^2)$ (O e Ω), como veremos un pouco mais adiante [dem. O no Th.1; dem. Ω no Th.2]

Imos calcular unha cota inferior para algoritmos de ordenación simples sendo T un vector [1..n]

Def.- Inversión: Calquer par (i,j) que verifique que i é inferior a j e que T[i]>T[j] (hai unha violación de orden xá que consideramos a ordenación ascendente, de menor a maior)

Ejemplo: T={3,1,4,1,5,9,2,6,5,3}

inversións:

(3,1),(3,1),(3,2)
(4,1),(4,2),(4,3)
(5,2),(5,3)
(9,2),(9,6),(9,5),(9,3)
(6,5),(6,3)
(5,3)

Se chamamos I ao número total de inversións, neste exemplo I=15. Isto quer dicer que no algoritmo se producirán I=15 intercâmbios. Os intercâmbios, para este algoritmo, son sempre entre dous elementos adxacentes. Cada intercâmbio elimina unha inversión: se temos {4,1,1} hai I=2 inversións (4,1) e (4,1), ao realizar o primeiro intercâmbio {1,4,1} elimina-se tamén unha inversión, xá só queda a segunda (4,1). Obviamente cando o vector teña 0 inversións estará ordenado. De isto deduce-se que a ordenación por inserción é O(I+n). Entón se I=0 ou se I=O(n) ten-se que a inserción é O(n). Se $I=O(n^2)$ entón a inserción é $O(n^2)$

Cal é o número médio de inversións?

(Partimos das hipóteses de que non hai duplicados e de que todas as permutacións son equiprobábeis).

•TEOREMA 1

"O número médio de inversións nun vector con n números diferentes é $n\cdot(n-1)/4$"

/demostración/

Sexa T un vector e Ti o vector inverso. Dado un par (x,y) con x<y, será unha inversión, xá en T, xá en Ti. O número total de pares é $(n-1)+(n-2)+...+1 = \Sigma\ (i=1..n-1)\ i = n\cdot(n-1)/2$ e dado que o número médio de inversións é $n\cdot(n-1)/2$ entre 2, é $n\cdot(n-1)/4$ c.q.d.

Aplicación: na ordenación por inserción como $I = n\cdot(n-1)/4$ o caso promédio é $O(n^2)$

•TEOREMA 2

"Calquer algoritmo que ordena intercambiando elementos adxacentes requere un tempo $\Omega(n^2)$ en promédio"

/demostración/

I promédio = nº médio de inversións = $n\cdot(n-1)/4 = \Omega(n^2)$

e como cada intercâmbio elimina só unha inversión exactamente
$\rightarrow \Omega(n^2)$ intercâmbios $\rightarrow \Omega(n^2)$ tempo

A família de algoritmos de ordenación que intercâmbian elementos adxacentes inclui:

-inserción
-selección
-burbulla

3.2.- Ordenación de Shell ou por diminución de incrementos.

Shell é o primeiro algoritmo que rebaixa $O(n^2)$

Como conseguir rebaixar $O(n^2)$? A partir do intercâmbio de elementos o mais lonxanos posíbeis entre si: estaremos probàbelmente eliminando mais de unha inversión con cada intercâmbio.

No Shell supomos a secuência de incrementos ht, ..., h1 onde h1=1

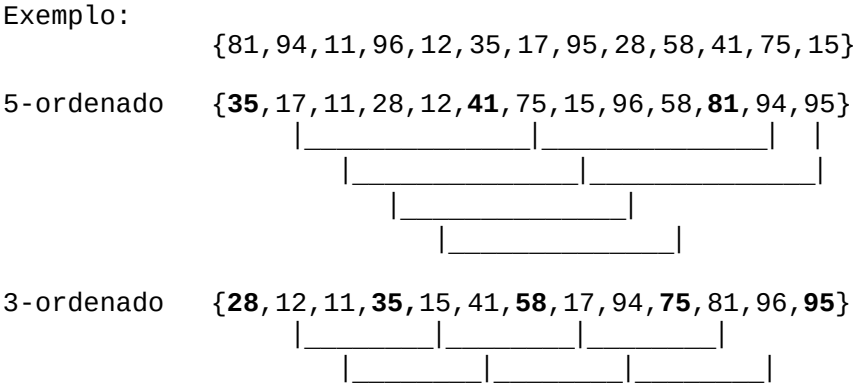
Na fase k da execución do algoritmo temos o incremento hk que unha vez aplicado ao vector debe asegurar que $\forall$ i $T[i] \leq T[i+hk]$ é dicer que os elementos espaciados por hk posicións están ordenados entre si.

Por definición dicimos que o vector T está hk-ordenado (despois da fase k)

•PROPRIEDADE

"Un vector hk-ordenado que despois é h_{k-1}-ordenado segue estando hk-ordenado"

Exemplo:



Vimos de empregar unha ordenación de Shell con secuência de incrementos 5,3,1

A secuência de incrementos proposta por Shell (incrementos de Shell) para un vector de lonxitude n é

$ht = \lfloor n/2 \rfloor$ [equival a n div 2]

$hk = \lfloor h_{k+1}/2 \rfloor$

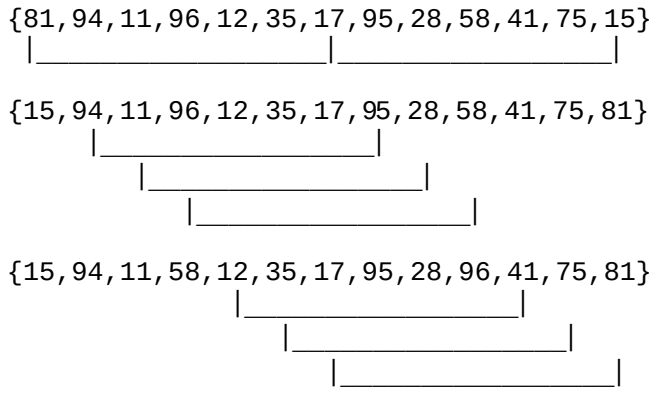
Existen outras secuências de incrementos que veremos mais adiante (Hibbard, Sedgewick).

```
{-------------------------------------------------------------------------------}
procedimento ordenar_Shell ( T [ 1 .. n ] )
   incremento ← n
   repetir
        incremento ← incremento div 2
        para i ← incremento + 1 até n facer
             tmp ← T [ i ]
             j ← i
             seguir ← certo
             mentres j - incremento > 0
             e seguir
             facer
                  se tmp < T [ j - incremento ]
                  entón
                       T [ j ] ← T [ j - incremento ]
                       j ← j - incremento
                  senón
                       seguir ← falso
                  finse
             finmentres
             T [ j ] ← tmp
        finpara
   até incremento = 1
finprocedimento
{-------------------------------------------------------------------------------}
```

Exemplo: Ordenación Shell con incrementos de Shell (metades enteiras)

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

n = 13 → incremento = 13 div 2 = 6



está 6-ordenado

ANÁLISE DO PIOR CASO DA ORDENACIÓN SHELL

•TEOREMA

"O tempo de execución do pior caso usando os incrementos de Shell é $\Theta(n^2)$"

/demostración/

[**1**] $\Omega(n^2)$ é cota inferior?

Supoñamos que n é potência de 2 $\rightarrow$ os incrementos son pares agás o derradeiro.

Exemplo: {1,9,2,10,3,11,4,12,5,13,6,14,7,15,8,16}

n=16, incr. Shell $\rightarrow h_t = \lfloor 16/2 \rfloor = 8 \rightarrow 4 \rightarrow 2 \rightarrow 1$

este é exemplo do pior caso porque xá está 2-ordenado inicialmente
números maiores en posicións pares
números menores en posicións impares

Despois da secuência de incrementos 8,4,2 o vector segue igual debido a que en princípio xá estaba 2-ordenado. Todo o traballo está en 1-ordenar. Debemos notar que o i-ésimo menor está na posición 2i-1 $\forall$ i $\leq$ n/2

Debemos mover o i-ésimo menor i-1 veces cara à esquerda. Colocar todos os elementos menores no seu sítio requere

$$\sum_{i=1}^{n/2} i-1 \text{ movimentos} = \Omega(n^2) \text{ como mínimo}$$

(Vimos de atopar unha cota inferior)

[**2**] $O(n^2)$

"O traballo que se fai nunha fase con incremento hk é equivalente a hk ordenacións por inserción nun conxunto limitado de n/hk elementos"

Entón o tempo de execución para unha fase é $O(hk \cdot (n/hk)^2) = O(n^2/hk)$

Entón para facer o conxunto de todas as fases o tempo de execución é

$$O\left(\sum_{i=1}^{t} n^2/hi\right) = O\left(n^2 \cdot \sum_{i=1}^{t} 1/hi\right)$$

onde o sumatório

$$\sum_{i=1}^{t} 1/hi = 1/1+1/2+1/4+...+1/(n/2) < 2 \rightarrow \text{tempo execución } O(n^2)$$

(incremento de cada fase)

↑

série xeométrica de factor común 2 e termo maior $h_1=1$

/!\ non é $O(n^3)$ só porque haxa três bucles aniñados, hai que fixar-se en cantas veces se executa cada un dos bucles.

O problema dos incrementos Shell é que non son primos entre si, entón o menor incremento pode ter pouco efeito.

Incrementos distintos dos de Shell, que melloran este tempo de execución $O(n^2)$:

Incrementos de Hibbard: $1,3,7,...,2^k-1$ (incrementos consecutivos non tên factores comúns).

Teorema: O tempo de execución do pior caso con incrementos de Hibbard é $\Theta(n^{3/2})$

Incrementos de Sedgewick: 1,5,19,41,109
Esta família de secuências é $O(n^{4/3})$ no pior caso

3.3.- Ordenación por montículos (Heapsort).

$O(n \cdot \log n)$

O mellor tempo visto até aqui, ainda que na prática o mais rápido é o Shell con incrementos de Sedgewick, porque o algoritmo é mais sinxelo.

Estratéxia: construir montículo de n elementos, logo n eliminacións armacenando os valores nun segundo vector.
Copiar o segundo vector no orixinal.

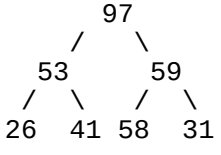
construir:	$O(n)$
n eliminacións	$n \cdot O(\log n)=O(n \cdot \log n)$
copiar:	$O(n)$
	$2 \cdot O(n)+O(n \cdot \log n)=O(n \cdot \log n)$

$O(n \cdot \log n)$ é complexidade temporal; podemos ter un problema coa memória, na cópia do vector; unha complexidade espacial.

Mellora: uso de celdas do vector que non están asignados ao montículo.

Ex. {31,41,59,26,53,58,97}

Construir:



```
     97   53   59   26   41   58   31
0    1    2    3    4    5    6    7

Eliminar_maior: 97 (raiz)

                    59
                  /   \
                53     58
               /  \   /
              26  41 31

     59   53   58   26   41   31   97
0    1    2    3    4    5    6    7
                      montículo ←|→ extraídos

                                59   97
0    1    2    3    4    5    6    7
                  montículo ←|→ extraídos

...

...  ...  ...  ...  ...  ...  59   97
0    1    2    3    4    5    6    7
|→ extraídos = vector ordenado
```

Na prática, o Shell é o mais usado.

3.4.- Ordenación por intercalación (Mergesort).

Pior caso: $O(n \cdot \log n) \Rightarrow$ exemplo do bo uso da recursividade.

Baseado en procedimento de intercalación de duas listas ordenadas nunha terceira lista ordenada.

u $[1|2|4|...]_m$

v $[3|7|\ |...]_{ñ}$

T $[1|2|3|4|7|...]_{n=m+ñ}$

```
procedimento intercala(u[1...m+1],v[1...ñ+1],t[1...n])
{construi un vector ordenado t[1...n] a partir de u[1...m] e v[1...ñ]}
   i←1
   j←1
   {centinelas:}
   u[m+1]←∞
   v[ñ+1]←∞
   para u[i] < v[j]
   entón t[k]←u[i]
        i←i+1
   senón t[k]←v[j]
        j←j+1
   finse
finprocedimento
```

Agora construiremos o algoritmo de ordenación a partir de intercala:

* se n=1 ⇒ vector xá ordenado.
* senón ⇒ intercala(1ª metade, 2ª metade) ——duas chamadas recursivas con metades ordenadas——.

Segue a estratéxia de divide e vencerás.

Introducimos unha mellora na ordenación por intercalación:

```
procedimento ordenación_intercalación(t[1...n])
   se n é suficientemente pequeno {n < umbral}
   entón ordenación_inserción(t[1...n])
   senón u[1...⌊n/2⌋]←t[1...⌊n/2⌋] {⌊n/2⌋=n div 2}
         v[1...⌈n/2⌉]←t[1+⌊n/2⌋...n]
         ordenación_intercalación(u[1...⌊n/2⌋)
         ordenación_intercalación(v[1...⌈n/2⌉)
         intercala(u,v,t)
   finse
finprocedimento
```

Se u e v non fosen variáveis locais teriamos:
* existirian log n vectores
* complexidade elevada

Entón colleremos u e v como variábeis globais para incrementar eficácia espacial.

Unha mellora plantexaria-se mediante o non uso de u e v. Separar t en u e v $\rightarrow$ O(n). Pode traballar-se sobre o próprio vector (usar un vector) pero o algoritmo é mais complicado; xá compensaria usar o rápido (Quicksort).

T(n) é o tempo de execución de ordenación por intercalación para un vector de n elementos.

Entón $T(n)=T(\lceil n/2 \rceil)+T(\lfloor n/2 \rfloor)+O(n)$

$O(n) \rightarrow$ separar+intercala=$O(n)+O(n)$

Supoñamos para simplificar que n é potência de 2, $n=2^k$.

$T(1)=O(1)$
$T(n)=2 \cdot T(n/2)+O(n)$

$\downarrow$

$T(1)=1$
$T(n)=2 \cdot T(n/2)+n$

Dividir T(n)/n

$\forall$ n $\rightarrow$ log n ecuacións que se suman $\Rightarrow$ $T(n)/n=T(n)/1+\log n$

$\Rightarrow$ $T(n)=n \cdot \log n+n=O(n \cdot \log n)$

Tamén se pode facer para $n \neq 2^k$, chegamos ao mesmo resultado.

Porblema: Complexidade espacial(u,v) $\rightarrow$ Quicksort. QS elimina a complexidade espacial. Importância de balancear (unha das bases de divide e vencerás).

```
procedimento ordenación_intercalación(t[1...n])
   .
   .
   .

   u[1...n-1] ← t[1...n-1]
```

```
   v[1] ← t[n]
   {chamadas recursivas:}
        ordenación_intercalación(u[1...n-1])
        ordenación_intercalación(v[1...1])
        intercala(u,v,t)
   .
   .
   .
finprocedimento
```

Entón $T(n) = T(n-1)+T(1)+O(n) = O(n^2)$

≡ ordenación por inserción (ademais mal implementada por uso inxustificado de recursividade).

3.5.- Ordenación rápida (Quicksort).

Método usado: Divide e vencerás.
Parte non recursiva: →construción de subinstâncias (partición).
→combinar as solucións.
Con respeito à ordenación por intercalación:
→construción ↑
→combinar ↓
Elección do pivote: o mais balanceado posíbel: mediana (cara de mais!).

Implementación: primeiro elemento como pivote.

Subvector t[i...j], p=t[i]

```
t [p|                    | ]
   i                      j
   ↑                      ↑
   k → >p           ≤p ← L
         ↑          ↑
        intercâmbio
```

Traballo até o cruzamento dos índices. Ao final intercâmbio entre p e t[L].

```
procedimento pivote(t[i...j];var L)
{postcondición: (i≤L≤j)∧(t[k]≤p ∀ i≤k≤L)∧(t[L]=p)∧(t[k]>p ∀ L<k≤j)}
{precondición: (p=t[i])}
   p←t[i]
   k←i
   L←j+1
   repetir k←k+1 até (t[k]>p) ou (k≥j)
   repetir L←L-1 até (t[L]≤p)
   mentres k<L facer
        intercambiar(t[k],t[L])
        repetir k←k+1 até t[k]>p
        repetir L←L+1 até t[L]≤p
   finmentres
intercambiar(t[i],t[L])
finprocedimento

procedimento quicksort(t[i...j])
   se j-i é suficientemente pequeno
   entón ordenación_inserción(t)
   senón pivote(t[i...j],L)
        quicksort(t[i...L-1])
```

```
        quicksort(t[L+1...j])
   finse
finprocedimento
```

Chamada: quicksort(t[1...n])

Selección do pivote:

* ideal: mediana → inviábel.
* usar o primeiro elemento: aceitábel se entrada é aleatória. Se entrada ordenada, mui deficiente: $O(n^2)$. Se entrada parcialmente ordenada, ineficiente.
* ao azar: pode ser bo. Problema: xeración custosa.
* mediana de três valores esquerda, direita, médio: t[i]+t[j]+t[(i+j)div 2]

Implementación: variación de estratéxia de partición.

Ex. hipótese: todos valores diferentes.

8, 1, 4, 9, 6, 3, 5, 2, 7, 0
8, 1, 4, 9, 0, 3, 5, 2, 7, 6
2, 1, 4, 9, 0, 3, 5, 8, 7, 6
2, 1, 4, 5, 0, 3, 9, 8, 7, 6
2, 1, 4, 5, 0, 3, 6, 8, 7, 9

Se pudese haber repetecións:

* parar k cando t[k] = pivote?
* parar L cando t[L] = pivote?

Facer o mesmo: se non facemos o mesmo, todos os que son iguais ao pivote caen ao mesmo lado, todos iguais → partición mui desequilibrada: $O(n^2)$.

As duas se detên: todos iguais → muitos intercâmbios entre elementos idênticos ⇒ inútil, pero cruzan-se na metade → equilibrada → $O(n \cdot \log n)$.

Nengún para: todos iguais, evitar que sobrepasen i e j nengún intercâmbio → $O(n^2)$.

Vectores pequenos:

Vector suficientemente pequeno é j-i ≤ 20, entón o método mais axeitado é a inserción.

t[1...n] lixeiramente desordenado → inserción todo vector de custe suficientemente pequeno está entre 5 e 20. Evitamos dexeneracións.

Implementación: parcialmente desordenado.

```
procedimento quicksort(t[1...n])
   qsort(t[1...n]) {parcialmente desordenado}
   ordenación_inserción(t[1...n])
finprocedimento

procedimento mediana3(t[1...n])
   centro ← (i+j) div 2
   se t[i] > t[centro]
   entón intercambiar(t[i],t[centro])
   finse
   se t[i] > t[j]
   entón intercambiar(t[i],t[j])
   finse
   se t[centro] > t[j]
   entón intercambiar(t[centro],t[j])
   finse
   intercambiar(t[centro],t[j-1])
finprocedimento
```

Ex.

8, 1, 4, 9, 6, 3, 5, 2, 7, 0

calculamos mediana e colocamos na sua posición

0, **1, 4, 9, 6, 3, 5, 2, 7**, 8

traballo sobre os que están en negriña

Ex.

8, 1, 4, 9, 6, 3, 5, 2, 7, 0

cálculo de mediana e ordenar

0, 1, 4, 9, 7, 3, 5, 2, 6, 8

0 centinela
partición nos elementos 1, 4, 9, 7, 3, 5, 2
6 pivote
8 centinela

```
procedimento qsort(t[i...j])
   se i+umbral ≤ j
   entón
        mediana3(t[i...j])
        pivote ← t[j-1]
        k ← j
        L ← j-1
        repetir
             repetir k ← k+1 até t[k] ≥ pivote
             repetir L ← L-1 até t[L] ≥ pivote
             intercambiar(t[k],t[L])
        até L ≤ k
        intercambiar(t[k],t[L])
        intercambiar(t[k],t[j-1])
        qsort(t[i...k-1])
        qsort(t[k+1...j])
   finse
finprocedimento
```

Para a análise, supomos:
* pivote é un valor aleatório.
* non faremos corte para ordenación por inserción.

$T(0)=T(1)=1$
$T(n)=T(i)+T(n-i-1)+cn$ (cte. n lineal O(n) para partición)

pior caso: cando o pivote é o menor elemnto do vector, sempre
$T(n)=T(n-1)+cn$, $n>1$
proxeccións:
$T(n-1)=T(n-2)+c(n-1)$
...
$T(2)=T(1)+c2$
_____
$T(n)=T(1)+c\Sigma_{i=2}^{n} i = O(n^2)$

mellor caso: pivote sempre na metade. Pivote sempre a mediana.
$T(n)=2T(n/2)+cn$
$=cn\cdot\log n+n$
$=O(n\cdot\log n)$

caso médio: cada tamaño da primeira partición é equiprobábel. Probabilidade de ter ese tamaño é 1/n.

$T(i)=T(n-i-1)=1/n \cdot \Sigma_{j=0}^{n-1} T(j)$

$T(n)=2/n \cdot [\Sigma_{j=0}^{n-1} T(j)]+cn$

$T(n)/(n+1)=T(1)/2+2c \cdot [\Sigma_{j=3}^{n+1} 1/i]$

$T(n)/(n+1)=O(\log n)$

$T(n)=O(n \cdot \log n)$

<fin do tema 3>