Algoritmos y Estructuras de Datos II

Ordenación

9 de marzo de 2016

Contenidos

Ordenación por inserción

- **No siempre** es posible contar el **número exacto** de operaciones.
- Un ejemplo de ello lo brinda otro algoritmo de ordenación: la ordenación por inserción.
- Es un algoritmo que se utiliza por ejemplo en juegos de cartas, cuando es necesario mantener un gran número de cartas en las manos, en forma ordenada.
- Cada carta que se levanta de la mesa, se inserta en el lugar correspondiente entre las que ya están en las manos, manteniendolas ordenadas.

Ordenación por inserción

Ejemplo

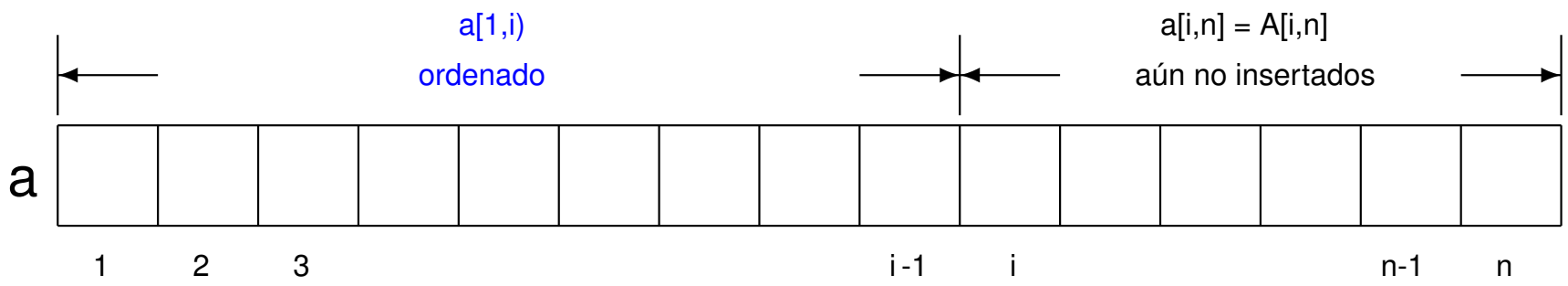
9	3	1	3	5	2	7
9	3	1	3	5	2	7
3	9	1	3	5	2	7
3	1	9	3	5	2	7
1	3	9	3	5	2	7
1	3	3	9	5	2	7
1	3	3	5	9	2	7

1	3	3	5	2	9	7
1	3	3	2	5	9	7
1	3	2	3	5	9	7
1	2	3	3	5	9	7
1	2	3	3	5	7	9

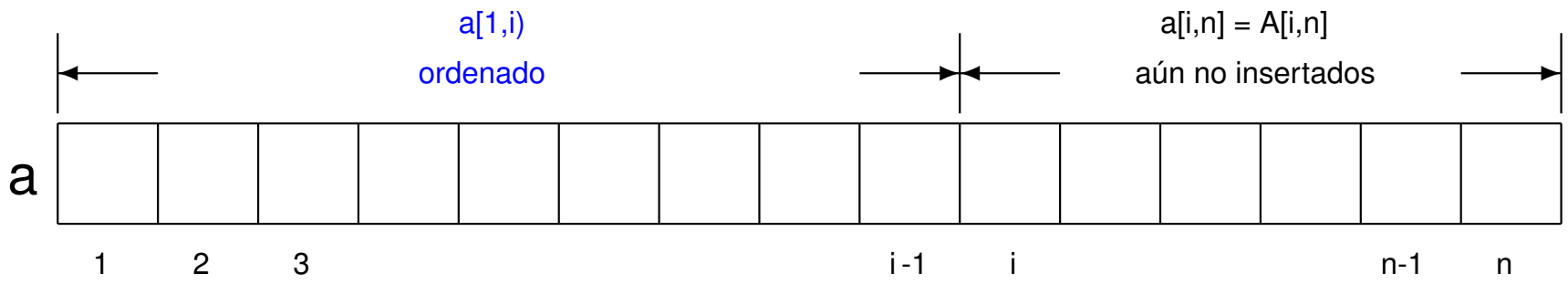
Demo (www.sorting-algorithms.com)

Ordenación por inserción

En un arreglo



Ordenación por inserción
Invariante



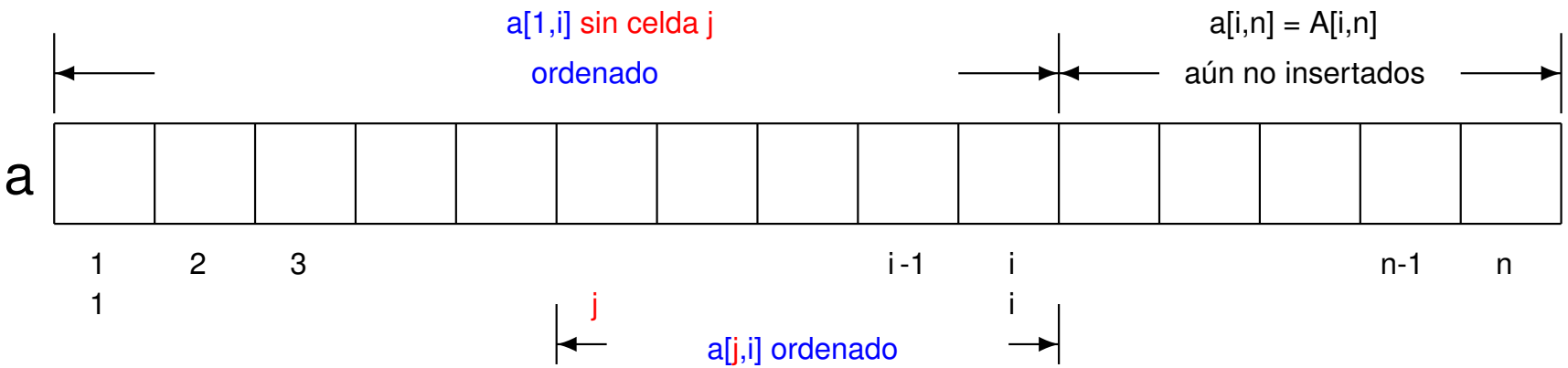
- Invariante:
  - el arreglo a es una permutación del original y
  - un segmento inicial a[1,i) del arreglo está ordenado.
  - (pero en general a[1,i) **no** contiene los mínimos del arreglo)

Ordenación por inserción
Pseudocódigo

{Pre: n $\geq$ 0 $\wedge$ a = A}
proc insertion_sort (**in**/**out** a: **array**[1..n] **of T**)
    for i:= 2 **to** n **do** {Inv: Invariante de recién}
        insert(a,i)
    od
end proc
{Post: a está ordenado y es permutación de A}

Ordenación por inserción

Invariante del procedimiento de inserción



- Invariante:
  - el arreglo a es una permutación del original
  - a[1,i] sin celda j está ordenado, y
  - a[j,i] también está ordenado.

Ordenación por inserción

Procedimiento de inserción

{Pre: 0 < i $\leq$ n $\wedge$ a = A}
proc insert (**in**/**out** a: **array**[1..n] **of T**, **in** i: **nat**)
    var j: **nat**
    j:= i   {Inv: Invariante de recién}
    do $j > 1 \wedge a[j] < a[j-1] \rightarrow$ swap(a,j-1,j)
                                         j:= j-1
    od
end proc
{Post: a[1,i] está ordenado $\wedge$ a es permutación de A}

Ordenación por inserción

Todo junto

```
proc insertion_sort (in/out a: array[1..n] of T)
        for i:= 2 to n do
            insert(a,i)
        od
end proc

proc insert (in/out a: array[1..n] of T, in i: nat)
        var j: nat
        j:= i
        do j > 1 ∧ a[j] < a[j − 1] → swap(a,j-1,j)
                                     j:= j-1
        od
end proc
```

Número de Comparaciones e intercambios

Procedimiento insert(a,i)

	comparaciones		intercambios	
si el valor de i es ...	mín	máx	mín	máx
2	1	1	0	1
3	1	2	0	2
4	1	3	0	3
⋮	⋮	⋮	⋮	⋮
n	1	n-1	0	n-1
total insertion_sort	n - 1	$\frac{n^2}{2} - \frac{n}{2}$	0	$\frac{n^2}{2} - \frac{n}{2}$

Ordenación por inserción, casos

- mejor caso: arreglo ordenado, n comparaciones y 0 intercambios.
- peor caso: arreglo ordenado al revés, $\frac{n^2}{2} - \frac{n}{2}$ comparaciones e intercambios, es decir, del orden de n^2.
- caso promedio: del orden de n^2.

Resumen

- Hemos analizado dos algoritmos de ordenación
  - ordenación por selección
  - ordenación por inserción
- la ordenación por selección hace siempre el mismo número de comparaciones, del orden de n^2.
- la ordenación por inserción también es del orden de n^2 en el peor caso (arreglo ordenado al revés) y en el caso medio,
- la ordenación por inserción es del orden de n en el mejor caso (arreglo ordenado),
- la ordenación por inserción realiza del orden de n^2 swaps (contra n de la ordenación por selección) en el peor caso.

Problema del bibliotecario

- Con cualquiera de los dos algoritmos la respuesta es 4 días,
- salvo que se trate de una biblioteca ya ordenada o casi ordenada, en cuyo caso:
  - ordenación por inserción es del orden de n,
  - y por ello la respuesta sería: 2 días.

Repaso de la ordenación por selección

```
proc selection_sort (in/out a: array[1..n] of T)
        var minp: nat
        for i:= 1 to n-1 do
            minp:= min_pos_from(a,i)
            swap(a,i,minp)
        od
end proc

fun min_pos_from (a: array[1..n] of T, i: nat) ret minp: nat
      minp:= i
      for j:= i+1 to n do if a[j] < a[minp] then minp:= j fi
      od
end fun
```

Se lo puede abreviar omitiendo la función auxiliar.

Forma abreviada de la ordenación por selección

```
proc selection_sort (in/out a: array[1..n] of T)
    var minp: nat
    for i:= 1 to n-1 do
        minp:= i
        for j:= i+1 to n do
            if a[j] < a[minp] then minp:= j fi
        od
        swap(a,i,minp)
    od
end proc
```

Repaso de la ordenación por inserción

```
proc insertion_sort (in/out a: array[1..n] of T)
        for i:= 2 to n do
            insert(a,i)
        od
end proc

proc insert (in/out a: array[1..n] of T, in i: nat)
        j:= i
        do j > 1 ∧ a[j] < a[j − 1] → swap(a,j-1,j)
                                     j:= j-1
        od
end proc
```

También puede abreviarse omitiendo el procedimiento auxiliar.

Forma abreviada de la ordenación por inserción

```
proc insertion_sort (in/out a: array[1..n] of T)
        for i:= 2 to n do
            j:= i
            do j > 1 ∧ a[j] < a[j − 1] → swap(a,j-1,j)
                                         j:= j-1
            od
        od
end proc
```

Demo (www.sorting-algorithms.com)

- Ejecución de ordenación por selección
  - entrada aleatoria
  - casi ordenada
  - invertida
  - con repeticiones
- Ejecución de ordenación por inserción
  - entrada aleatoria
  - casi ordenada
  - invertida
  - con repeticiones
- Comparación y conclusiones.

Reflexión sobre paralelismo

¿Qué provecho podríamos sacar a los algoritmos que hemos visto si contáramos con varios o muchos procesadores capaces de cooperar entre ellos?