Ordenación por selección

Idea

a[1,i) a[i,n]

|<- mínimos ordenados ->||<- aún no ordenados ->|

Invariante:

- el arreglo a es una permutación del original
- un segmento inicial a[i,i) del arreglo está ordenado
- dicho segmento contiene los elementos mínimos del arreglo

Código

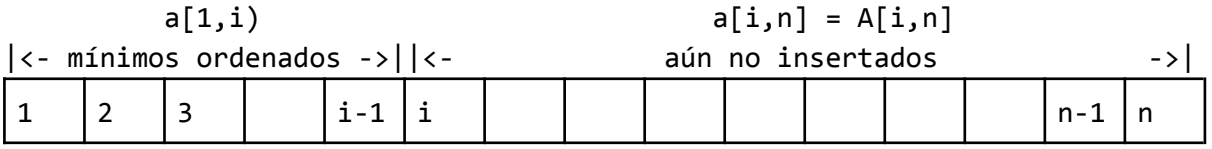
```
proc selection_sort (in/out a: array[1..n] of T)
    var i, minp: nat
    i := 1
    while i < n do { inv: el invariante de recién }
        minp := min_pos_from(a,i)
        swap (a,i,minp)
        i := i + 1
    od
end proc


proc swap (in/out a: array[1..n] of T, in i, j: nat)
    var tmp: T
    tmp := a[i]
    a[i] := a[j]
    a[j] := tmp
end proc


proc 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
```

Número de comparaciones

- Al llamarse a min_pos_from(a,i) se realizan **n-i** comparaciones
- selection_sort llama a min_pos_from(a,i) para i ∈ {1,2,...,n-1}
- por lo tanto, en total son (n-1) + (n-2) + … + (n-(n-1)) comparaciones
- es decir, (n-1) + (n-2) + … + 1) = (n*n(n-1))/2 comparaciones.

Ordenación por inserción

Idea



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

Código

```
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
    while 1 < j ∧ a[j] < a[j-1] do
        swap(a,j-1,j)
        j := j - 1
    od
end proc
```

Número de comparaciones e intercambios

	comparaciones		**intercambios**	
si el valor de i es…	**min**	**max**	**min**	**max**
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**	**$(n^2/2)-(n/2)$**	**0**	**$(n^2/2)-(n/2)$**

Casos

- mejor caso: arreglo ordenado, n comparaciones y 0 intercambios

- peor caso: arreglo ordenado al revés, del orden de n^2
- caso promedio: del orden de n^2

Número de operaciones de un programa

- Una vez que uno sabe qué **operación** quiere contar, debe imaginar una ejecución arbitraria, genérica del programa intentando contar el número de veces que esa ejecución arbitraria realizará **dicha operación**.
- Es imprescindible comprender **cómo** se ejecuta el programa.
- Una secuencia de comandos se ejecuta de manera secuencial, del primero al último. $(C_1;C_2;\dots;C_n)$
- Para contar cuántas veces se ejecuta **la operación**, entonces, se cuenta cuántas veces se ejecuta en el primero, cuántas en el segundo, etc. y luego se suman los números obtenidos:
- $ops(C_1;C_2;\dots;C_n) = ops(C_1) + ops(C_2) + \dots + ops(C_n)$

Skip

- El comando **skip** equivale a una secuencia vacía:
- ops(**skip**) = 0

For

- El comando **for** k:= n **to** m **do** C(k) **od** “equivale” también a una secuencia:
- **for** k:= n **to** m **do** C(k) **od** “equivale” a: (C(n),C(n+1)…C(m))
- De esta equivalencia resulta:
- ops(**for** k:= n **to** m **do** C(k) **od**) = ops(C(n)) + ops(C(n+1)) + … + ops(C(m))
- que también se puede escribir:
- ops(**for** k:= n **to** m **do** C(k) **od**) = Σ(k=n **to** m) ops(C(k))
- La ecuación solamente vale cuando no hay interés en contar las operaciones que involucran el índice k implícitas en el for: inicialización, comparación con la cota m, incremento; ni el cómputo de los límites n y m. Por eso escribimos “equivale” entre comillas.

If

- El comando **if** b **then** C **else** D **fi** se ejecuta evaluando la condición b y luego, en función del valor de verdad que se obtenga, ejecutando C (caso verdadero) o D (caso falso).
- Para contar cuántas veces se ejecuta la **operación**, entonces, se cuenta cuántas veces se la ejecuta durante la evaluación de b y luego cuántas en la ejecución de C o D
- ops(**if** b **then** C **else** D **fi**) = |ops(b)+ops(C) caso b V
  |ops(b)+ops(D) caso b F

Asignación

- El comando **x:=e** se ejecuta evaluando la expresión e y modificando la posición de memoria donde se aloja la variable x con el valor de e.

- ops(**x:=e**) = |ops(e)+1 si se desea contar la asignación o las modificaciones de memoria
  |ops(e) en caso contrario
- Tener en cuenta que la evaluación de e puede implicar la llamada a funciones auxiliares cuyas operaciones deben ser también contadas.

Número de operaciones de una expresión

- Similares ecuaciones se pueden obtener para la evaluación de expresiones.
- Por ejemplo, para evaluar la expresión e < f , primero se evalúa la expresión e, luego se evalúa la expresión f y luego se comparan dichos valores.
- ops(e<f) = |ops(e)+ops(f)+1 si se cuentan comparaciones
  |ops(e)+ops(f) caso contrario

Ejemplo

Comparaciones de la ordenación por selección

```
proc selection_sort (in/out a: array[1..n] of T)
    var minp: nat
    for i := 1 to n 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
```

```
ops(selection_sort(a))
= ops(for i:=1 to n do minp:= min_pos_fr…;swap… od)
= Σ(i=1 to n) ops(minp:=min_pos_from(a,i);swap(a,i,minp))
= Σ(i=1 to n) (ops(minp:= min_pos_from(a,i)) + ops(swap(a,i,minp)))
= Σ(i=1 to n) ops(minp:= min_pos_from(a,i))
= Σ(i=1 to n) ops(min_pos_from(a,i))
= Σ(i=1 to n) ops(minp:= i;for j:= i+1 to n do if … fi od)
= Σ(i=1 to n) (ops(minp:= i) + ops(for j:= i+1 to n do if…fi od))
= Σ(i=1 to n) ops(for j:= i+1 to n do if…fi od)
= Σ(i=1 to n) Σ(j=i+1 to n) ops(if a[j] < a[minp] then minp:= j fi)
= Σ(i=1 to n) Σ(j=i+1 to n) (ops(a[j] < a[minp]) + ops(minp:= j)) o
ops(skip)
= Σ(i=1 to n) Σ(j=i+1 to n) ops(a[j] < a[minp])
= Σ(i=1 to n) Σ(j=i+1 to n) 1
= Σ(i=1 to n) (n-i)
= Σ(i=0 to n-1) i
= n*(n-1) / 2
= (n²/2) − (n/2)
```

Intercambios de la ordenación por selección

```
ops(selection_sort(a))
= ops(for i:= 1 to n do minp:= min_pos_fr. . . ;swap. . . od)
= Σ(i=1 to n) ops(minp:= min_pos_from(a,i);swap(a,i,minp))
= Σ(i=1 to n) (ops(minp:= min_pos_from(a,i)) + ops(swap(a,i,minp)))
= . . .
= Σ(i=1 to n) (0 + ops(swap(a,i,minp)))
= Σ(i=1 to n) ops(swap(a,i,minp))
= Σ(i=1 to n) 1
= n
```