Instituto Politécnico Nacional

Escuela Superior de Cómputo

Análisis de Algoritmos

Profesor: Edgardo Adrián Martínez Franco

Alumno: Ortega Victoriano Ivan

Fecha: 24 de Marzo 2017

Análisis de algoritmos no recursivos.
Ejercicio 1: Encuentre el orden O de complejidad temporal y espacial del algoritmo de ordenamiento por Burbuja Simple.

```
Procedimiento BurbujaSimple(A,n)
  para i=1 hasta (i<n) hacer
    para j=0 hasta (j<n-1) hacer
      si (A[j]>A[j+1]) hacer
        temp = A[j]
        A[j] = A[j+1]
        A[j+1] = temp
      fin si
    fin para
  fin para
fin Procedimiento
```

Sol: Para el análisis temporal se considerarán como operaciones básicas las 3 asignaciones dentro del *if*, la comparación del *if* y los incrementos en los loops. Además, se tomará el peor caso para encontrar la cota O.
Para este algoritmo, el peor caso se da cuando el arreglo esta ordenado de forma descendente, es decir, de mayor a menor, ya que así, siempre estará entrando a la condición del if, ejecutando las 3 asignaciones. Del código, vemos lo siguiente:

```
para i=1 hasta (i<n) hacer          //Compara n veces
  para j=0 hasta (j<n-1) hacer      //Compara n veces
    si (A[j]>A[j+1]) hacer          //1 comparacion
      temp = A[j]                   //3 asignaciones
      A[j] = A[j+1]
      A[j+1] = temp
    fin si
  fin para
fin para
```

De tal forma que la función de complejidad temporal del algoritmo, estará dada por:

$$f_t(n) = (n)(n)(3) = 3n^2$$

Ahora para encontrar el orden de complejidad temporal, de acuerdo con la notación de Landau:

Notación de Landau. *Se dice que* $f(n)$ *es de orden* $O(g(n))$ *si*

$$\exists c \geq 0 \ y \ n_0 \geq 0 \mid |f(n)| \leq c|g(n)| \ , \forall n \geq n_0.$$

Ahora, sea $g(n) = n^2$ y $c = 4$, vemos que:
Si n=0,

$$f(0) = 3(0)^2 = 0 \leq 0 = 4(0)^2 = g(0)$$

Si n=1,

$$f(1) = 3(1)^2 = 3 \leq 4 = 4(1)^2 = g(1)$$

Si n=2,

$$f(2) = 3(2)^2 = 12 \leq 16 = 4(2)^2 = g(2)$$

En general, se cumple que

$$|f(n)| \leq c|g(n)| \ , \forall n \geq n_0.$$

De lo que podemos decir que $f_t(n)$ es de orden $O(n^2)$.

Por otro lado, para el análisis espacial, tenemos

```
para i=1 hasta (i<n) hacer      // 1 variable i
  para j=0 hasta (j<n-1) hacer // 1 variable j
    si (A[j]>A[j+1]) hacer
      temp = A[j]   // 1 variable temp
      A[j] = A[j+1] // n variables del arreglo "A"
      A[j+1] = temp
    fin si
  fin para
fin para
```

De tal forma que la función de complejidad espacial del algoritmo estará dada por:

$$f_e(n) = 3 + n$$

Para encontrar la cota O de $f_e(n)$, se hace al igual que en el caso anterior.
Sea $g(n) = n$, $c = 4$ y $n_0 = 1$, tenemos que
Si n=1,

$$f(1) = 3 + 1 = 4 \leq 4 = 4(1) = g(1)$$

Si n=2,

$$f(2) = 3 + 2 = 5 \leq 8 = 4(2) = g(2)$$

Si n=3,

$$f(0) = 3 + 3 = 6 \leq 12 = 4(3) = g(3)$$

En general, se cumple que

$$|f(n)| \leq c|g(n)| \text{ , } \forall n \geq n_0.$$

De lo que podemos decir que $f_e(n)$ es de orden $O(n)$.

Ejercicio 2: Encuentre el orden O de complejidad temporal y espacial del algoritmo de ordenamiento por Inserción.

```
Procedimiento Insercion(A,n)
  para i=1 hasta i<n hacer
    temp=A[i]
    j=i-1
    mientras((A[j]>temp)&&(j>=0)) hacer
      A[j+1]=A[j]
      j--
    fin mientras
    A[j+1]=temp
  fin para
fin Procedimiento
```

Sol: Para este algoritmo, al igual que en el caso del ordenamiento burbuja, nos basaremos en el análisis del peor caso para encontrar la cota O. De tal forma que para el ordenamiento por inserción el peor caso se da de igual forma cuando el arreglo está ordenado de forma descendente, ya que así, siempre estará ejecuntando las operaciones del while, de no serlo así, habría ocasiones en las que simplemente no entraría al ciclo, ya que no se cumpliría que $A[j] > temp$.
Del código, vemos lo siguiente:

```
Procedimiento Insercion(A,n)
  para i=1 hasta i<n hacer    // compara n veces, sin embargo,
    temp=A[i]         // las operaciones  realizadas
    j=i-1           // dependen del while
    mientras((A[j]>temp)&&(j>=0)) hacer // su analisis se hace
      A[j+1]=A[j]           // a continuacion
      j--
    fin mientras
    A[j+1]=temp
  fin para
fin Procedimiento
```

Analizando el caso para el ciclo *while*, tenemos que para el contador $i = 1$ el ciclo se ejecutará una sola vez, ya que $j = 0$, al decrementarla para la siguiente iteración tendríamos $j = -1$, lo cual ya no cumpliría con la condición del *while* que pide que $j <= 0$. Así para $i = 2$, tendríamos para las iteraciones del *while* que para la primera $j = 1$, se decrementa, luego $j = 0$, se decrementa, y así sucesivamente.

De tal forma que para la función de complejidad temporal tenemos lo siguiente:

$$f_t(n) = 1 + 2 + 3 + ... + (n-1) + n = \sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

Si observamos la función, nos damos cuenta que es de orden cuadrático, ahora obtengamos la cota O de $f_t(n)$.
Sea $g(n) = n^2$, $c = 1$ y $n_0 = 1$, se tiene que
Si n=1,

$$f(1) = \frac{1^2+1}{2} = 1 \leq 1 = 1^2 = g(1)$$

Si n=2,

$$f(2) = \frac{2^2+2}{2} = 3 \leq 4 = 2^2 = g(2)$$

Si n=3,

$$f(1) = \frac{3^2+3}{2} = 6 \leq 9 = 3^2 = g(3)$$

En general, se cumple que

$$|f(n)| \leq c|g(n)| \ , \forall n \geq n_0.$$

De lo que podemos decir que $f_t(n)$ es de orden $O(n^2)$.
Ahora para el análisis de complejidad espacial tenemos que:

```
Procedimiento Insercion(A,n)
  para i=1 hasta i<n hacer    // 1 variable i y 1 variable temp
    temp=A[i]          // n variables del arreglo "A"
    j=i-1            // 1 variable j
    mientras((A[j]>temp)&&(j>=0)) hacer
      A[j+1]=A[j]
      j--
    fin mientras
    A[j+1]=temp
  fin para
fin Procedimiento
```

De tal forma que la función de complejidad espacial queda de la siguiente manera:

$$f_e(n)) = n + 3$$

Para encontrar su cota O, propongamos $g(n) = n$, $c = 4$ y $n_0 = 1$
Si n=1,

$$f(1) = 1 + 3 = 4 \leq 4 = 4(1) = g(1)$$

Si n=2,

$$f(2) = 2 + 3 = 5 \leq 8 = 4(2) = g(2)$$

Si n=3,

$$f(3) = 3 + 3 = 6 \leq 12 = 4(3) = g(3)$$

En general, se cumple que

$$|f(n)| \leq c|g(n)| \text{ , } \forall n \geq n_0.$$

De lo que podemos decir que $f_e(n)$ es de orden $O(n)$.