Analisis y diseño de algoritmos

Juan Gutiérrez

September 2019

1 InsertionSort: correctitud y análisis

Resuelve el problema de ordenamiento.
Entrada: Secuencia $< a_1, a_2, \ldots a_n >$
Salida: Permutacion $< a'_1, a'_2, \ldots a'_n >$ tal que $a'_1 \leq a'_2 \leq \ldots \leq a'_n$

```
INSERTION-SORT(A)
for j = 2 to A.length
    key = A[j]
    // Insert A[j] into the sorted sequence A[1..j − 1].
    i = j − 1
    while i > 0 and A[i] > key
        A[i + 1] = A[i]
        i = i − 1
    A[i + 1] = key
```

Figure 1: Tomada del libro Cormen, Introduction to Algorithms

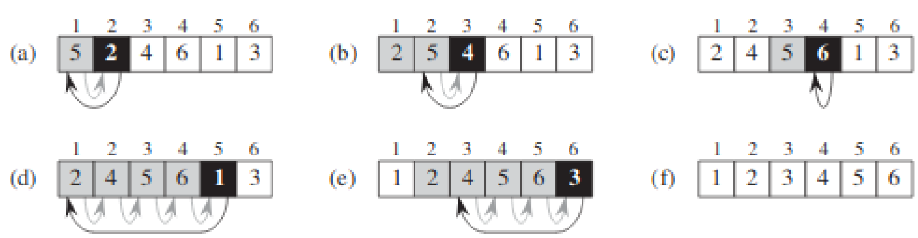

Figure 2: Tomada del libro Cormen, Introduction to Algorithms

1.1 Correctitud

Invariante: Al inicio de cada iteracion del for de las lineas 1–8, el subarreglo A[1..j-1] consiste en los elementos de A[1..j-1] pero ordenados

Probaremos que la invariante es cierta.

Inicialización: La invariante es cierta antes de la primera iteración. Ya que, cuando $j = 2$, tenemos que A[1..j-1]=A[1] consiste en el mismo A[1] ordenado.

Manuntención Asumimos que la invariante se cumple al inicio de la j-ésima iteración. Sabemos que $A[1..j]$ está ordenado. Debido al bucle while (lineas 5–7), el elemento $A[j + 1]$ es insertado en la posición $i + 1$ de manera tal que $A[1..j + 1]$ queda ordenado. (Obs: una demostracion completa deberia probar que el elemento es insertado adecuadamente, con una invariante nueva en el bucle while)

Terminación Tenemos que $j = n + 1$. Luego $A[1..j - 1] = A[1..n]$ está ordenado.

1.2 Análisis

Suposiciones: Modelo RAM: instrucciones una tras otra sin paralelismo. Cada instruccion toma tiempo constante (sumar, restar, multiplicar, copiar, etc).

Tamaño de la entrada depende del problema. Sort: numero de elementos. Multiplicar dos numeros: numero de bits. Grafos: numero de vertices y numero de aristas.

	INSERTION-SORT(A)	*cost*	*times*
1	**for** $j = 2$ **to** $A.length$	c_1	n
2	$key = A[j]$	c_2	$n - 1$
3	// Insert $A[j]$ into the sorted sequence $A[1..j-1]$.	0	$n - 1$
4	$i = j - 1$	c_4	$n - 1$
5	**while** $i > 0$ and $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
6	$A[i + 1] = A[i]$	c_6	$\sum_{j=2}^{n} (t_j - 1)$
7	$i = i - 1$	c_7	$\sum_{j=2}^{n} (t_j - 1)$
8	$A[i + 1] = key$	c_8	$n - 1$

Figure 3: Tomada del libro Cormen, Introduction to Algorithms

Para cada $j = 2 \ldots n$, sea t_j el número de veces que el while de la linea 5 es ejecutado.

Sea $T(n)$ el tiempo de ejecución con n valores de entrada.

$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2} t_j + c_6 \sum_{j=2} (t_j - 1) + c_7 \sum_{j=2} (t_j - 1) + c_8(n - 1)$

En el mejor caso, si el vector está ordenado, tenemos $t_j = 1$ para todo j.

Luego, $T(n) = c_1n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1) = (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_3 + c_5 + c_8) = an + b$

En el peor caso, tendriamos que el vector está ordenado de manera decreciente, luego $t_j = j$ para todo j. Entonces $T(n) = c_1n + c_2(n-1) + c_4(n-1) + c_5\sum_{j=2}^{n} j + c_6\sum_{j=2}^{n}(j-1) + c_7\sum_{j=2}^{n}(j-1) + c_8(n-1) = c_1n + c_2(n-1) + c_4(n-1) + c_5(\frac{n(n+1)}{2} - 1) + c_6(\frac{n(n+1)}{2}) + c_7(\frac{n(n+1)}{2} - 1) + c_8(n-1) = (c_5/2 + c_6/2 + c_7/2)n^2 + (...)n - (...) = an^2 + bn + c$

1.3 Análisis de peor caso y caso medio

Nos interesa el peor caso: el mayor tiempo de ejecución para cualquier entrada de tamaño n. Con eso garantizamos que el algoritmo no va a tomar más tiempo.

Muchas veces el caso medio es igual de malo (en el insertion sort tenemos que $t_j = j/2$ en el caso medio).

Veremos mas adelante el concepto de tiempo de ejecución esperado en el análisis del quicksort.

2 Crecimiento de funciones

No es necesario tener una precision exacta del tiempo de ejecución. Para entradas grandes, ya no importan las constantes multiplicativas ni los términos de menor orden. Nos interesa la eficiencia asintótica de los algoritmos, osea como se comporta la función en el límite.

2.1 Notación Θ

Dada una función $g(n)$,

$$\Theta(g(n)) = \{f(n) : \text{existen constantes positivas } c_1, c_2, n_0$$
$$\text{tales que } 0 \leq c_1g(n) \leq f(n) \leq c_2g(n) \text{ para todo } n \geq n_0\}$$

Como $\Theta(g(n))$ es un conjunto, podemos decir $f(n) \in \Theta(g(n))$. También diremos que $f(n) = \Theta(g(n))$. Tambien se dice $f(n)$ *es* $\Theta(g(n))$.

Ejemplo 2.1. $\frac{1}{2}n^2 - 3n = \Theta(n^2)$

Borrador: $c_1n^2 \leq \frac{1}{2}n^2 - 3n \leq c_2n^2$. Entonces $c_1 \leq \frac{1}{2} - 3/n \leq c_2$. Entonces tomo $c_2 = 1/2$. Cuando $n = 7$, tengo $c_1 \leq 1/2 - 3/7 = 1/14$.

Prueba. Note que para $n \geq 7$, se cumple que $\frac{1}{2}n^2 - 3n = n^2(1/2 - 3/n) \geq \frac{n^2}{14}$ y también se cumple que $\frac{1}{2}n^2 - 3n \leq \frac{1}{2}n^2$. Luego, cuando $c_1 = 1/14, c_2 = 1/2, n_0 = 7$, tenemos que $0 \leq c_1n^2 \leq \frac{1}{2}n^2 - 3n \leq c_2n^2$ para todo $n \geq n_0$. Luego $\frac{1}{2}n^2 - 3n = \Theta(n^2)$

Ejemplo 2.2. $6n^3 \neq \Theta(n^2)$

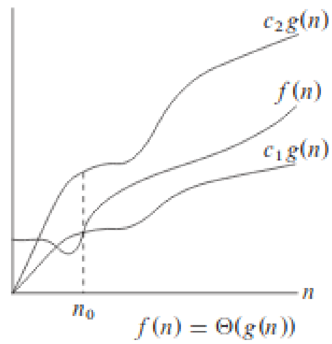

Figure 4: Tomada del libro Cormen, Introduction to Algorithms

Suponga por contradicción que $6n^3 = \Theta(n^2)$. Entonces existen $c_1, c_2, n_0 > 0$ tales que $0 \leq c_1 n^2 \leq 6n^3 \leq c_2 n^2$. Tendriamos $6n \leq c_2$ para todo $n \geq n_0$. Contradicción pues c_2 es constante.

Ejercicio 2.1. $an^2 + bn + c = \Theta(n^2)$

2.2 Notación O

Dada una función $g(n)$,

$$O(g(n)) = \{f(n) : \text{existen constantes positivas } c, n_0$$
$$\text{tales que } 0 \leq f(n) \leq cg(n) \text{ para todo } n \geq n_0\}$$

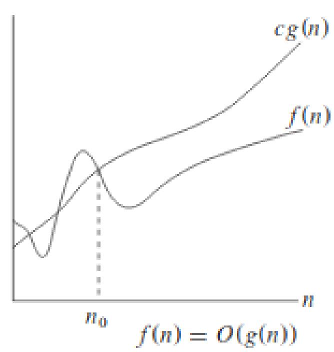

Figure 5: Tomada del libro Cormen, Introduction to Algorithms

Ejemplo 2.3. $an + b = O(n^2)$

Ejercicio: tomar $c = a + |b|$ y $n_0 = \max\{1, -b/a\}$.

Obs. Notacion sirve para acotar el peor caso del tiempo de ejecución de un algoritmo, y portanto también cada caso del algoritmo.

2.3 Notación Ω

Dada una función $g(n)$,

$$\Omega(g(n)) = \{f(n) : \text{existen constantes positivas } c, n_0$$
$$\text{tales que } 0 \leq cg(n) \leq f(n) \text{ para todo } n \geq n_0\}$$

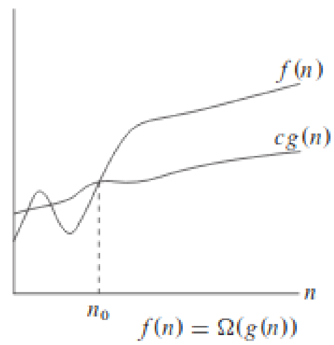

Figure 6: Tomada del libro Cormen, Introduction to Algorithms

Obs. Sirve para acotar el mejor caso inferiormente, y por lo tanto cada caso inferiormente.

En el insertion sort $T(n) = O(n^2)$ $T(n) = \Omega(n)$

Teorema 2.1. *$f = \Theta(g(n)$ si y solo si $f = O(g(n))$ y $f = \Omega(g(n))$*

2.4 Notación o

Dada una función $g(n)$,

$$o(g(n)) = \{f(n) : \text{para cada constante } c > 0$$
$$\text{existe una constante } n_0 \text{ tal que } 0 \leq f(n) < cg(n) \text{ para todo } n \geq n_0\}$$

Ejemplo 2.4. $2n = o(n^2)$

Borrador. Quiero $2n < cn^2$. Entonces $n > 2/c$. Tomamos $n_0 = 1 + \frac{2}{c}$
Prueba: ejercicio...

Ejemplo 2.5. $2n^2 \neq o(n^2)$

Suponga por contradicción que $2n^2 \neq o(n^2)$. Tome $c = 1$, tendriamos $2n^2 < n^2$, contradicción.

Obs.

$f(n) = o(g(n))$ si y solo si $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$

2.5 Notación ω

Dada una función $g(n)$,

$$\omega(g(n)) = \{f(n) : \text{para cada constante } c > 0$$
$$\text{existe una constante } n_0 \text{ tal que } 0 < cg(n) \leq f(n) \text{ para todo } n \geq n_0\}$$

Obs.

$f(n) = \omega(g(n))$ si y solo si $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$.

2.6 Comparaciones

- $f(n) = \Theta(g(n))$, $g(n) = \Theta(h(n))$, entonces $f(n) = \Theta(h(n))$
- $f(n) = O(g(n))$, $g(n) = O(h(n))$, entonces $f(n) = O(h(n))$

  ...
- $f(n) = O(f(n))$
- $f(n) = \Theta(g(n))$ entonces $g(n) = \Theta(f(n))$
- $f(n) = O(g(n))$ entonces $g(n) = \Omega(f(n))$
- $f(n) = o(g(n))$ entonces $g(n) = \omega(f(n))$

Obs hay funciones no comparables: n y $n^{1+\sin n}$