Analisis y diseño de algoritmos

Juan Gutiérrez

September 2019

1 El método de división y conquista

- **Dividir** el problema en subproblemas que son instancias del mismo

  En MergeSort: dividir el arreglo de tamaño n en dos subsecuencias de tamaño $n/2$

- **Conquistar**: Resolver los subproblemas recursivamente. Si el tamaño es pequeño, resolverlos directamente.

  En MergeSort: ordenar las dos subsecuencias usando MergeSort

- **Combinar** las soluciones de los subproblemas en una solución para el porblema original.

  En MergeSort: Mezclar las dos subsecuencias ordenadas

Analizaremos el algoritmo MERGE-SORT, que esta basado en la subrutina MERGE. Esta subrutina recibe un vector$A[1..n]$ y tres indices p, q, r tales que $A[p..q]$ y $A[q+1..r]$ están ordenados, y ordena el vector $A[p..r]$.

```
MERGE-SORT(A, p, r)
if p < r
    q = ⌊(p + r)/2⌋
    MERGE-SORT(A, p, q)
    MERGE-SORT(A, q + 1, r)
    MERGE(A, p, q, r)
```

Figure 1: Tomada del libro Cormen, Introduction to Algorithms

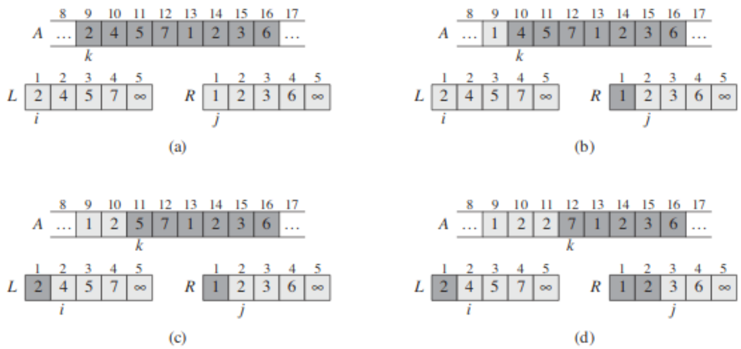

Figure 2: Tomada del libro Cormen, Introduction to Algorithms

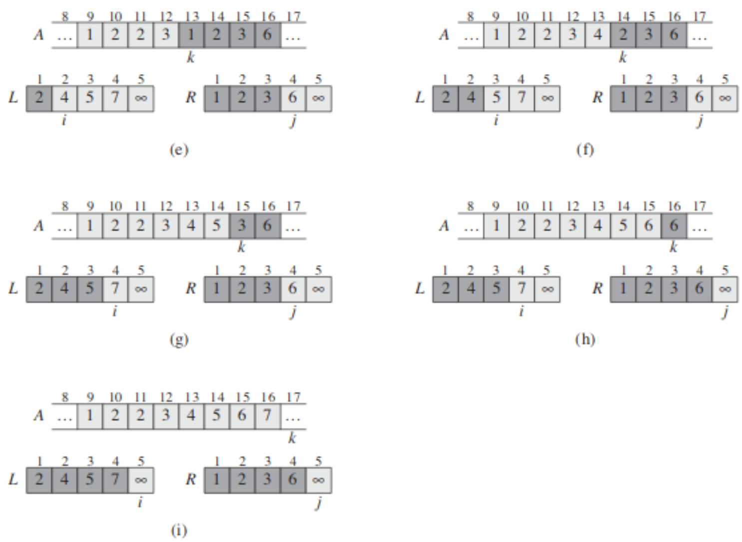

Figure 3: Tomada del libro Cormen, Introduction to Algorithms

```
MERGE(A, p, q, r)
1  n1 = q - p + 1
2  n2 = r - q
3  let L[1 .. n1 + 1] and R[1 .. n2 + 1] be new arrays
4  for i = 1 to n1
5      L[i] = A[p + i - 1]
6  for j = 1 to n2
7      R[j] = A[q + j]
8  L[n1 + 1] = ∞
9  R[n2 + 1] = ∞
10 i = 1
11 j = 1
12 for k = p to r
13     if L[i] ≤ R[j]
14         A[k] = L[i]
15         i = i + 1
16     else A[k] = R[j]
17         j = j + 1
```

Figure 4: Tomada del libro Cormen, Introduction to Algorithms

MERGE recibe un vector $A[1..n]$ y tres indices p, q, r tales que $A[p..q]$ y $A[q+1..r]$ están ordenados, y ordena el vector$A[p..r]$.

Analizamos la subrutina MERGE.

Invariante: Al inicio de cada iteraición del bucle for (líneas 12–17), el subvector $A[p..k-1]$contiene los $k-p$ elementos más pequeños entre $L[1..n_1+1]$ y $R[1..n_2+1]$ ordenados. También, $L[i]$ y $R[j]$ son los elementos más pequeños que no han sido copiados.

Prueba:

- **Inicialización** $k = p$, luego $A[p..k-1] = \emptyset$
- **Manuntención**

  Caso 1: $L[i] \leq R[j]$. Entonces se ejecuta la línea 14. Como $A[p..k-1]$ estaba ordenado con los menores elementos, entonces $A[p..k-1]$ tendrá los $k-p+1$ elementos menores. Caso 2: $L[i] > R[j]$: similar.
- **Terminación**

  $k = r+1$. Luego $A[p..k-1] = A[p..r]$ contiene los $k-p = r-p+1 = n_1+n_2-2$ elementos más pequeños de $L[1..n_1+1]$ y $R[1..n_2+1]$. Osea todos excepto los sentinelas.

Tiempo de ejecución de la subrutina MERGE:

- líneas 1–3, 8–11: tiempo constante

- líneas 4–7: tiempo $\Theta(n_1 + n_2) = \Theta(n)$
- líneas 12–17: tiempo $\Theta(n)$

Llamada inicial: MERGE-SORT(A,1,A.length).

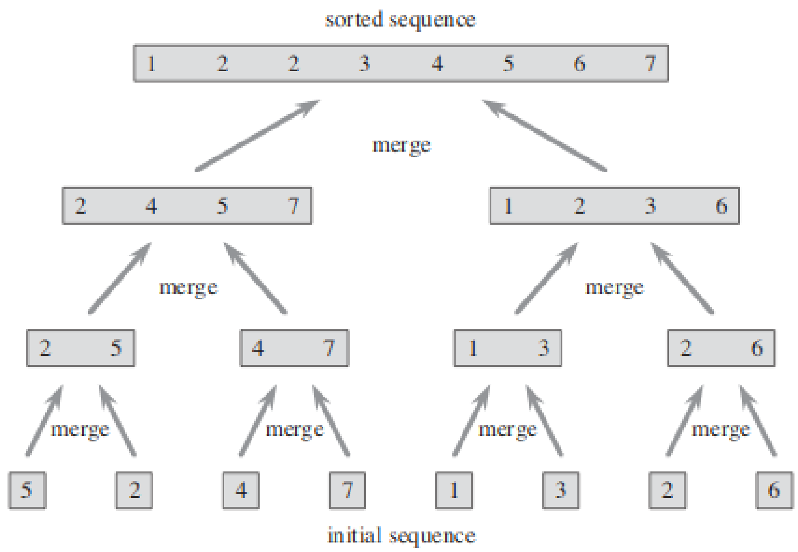

Figure 5: Tomada del libro Cormen, Introduction to Algorithms

2 Análisis de tiempo

En general, cuando un algoritmo tiene una llamada a sí mismo, describimos su tiempo de ejecución $T(n)$ mediante una recurrencia.

Si el tamaño es $n \leq n_0$, la solución toma tiempo constante: $T(n) = k$.

Si el tamaño es $n > n_0$ y tenemos a subproblemas, cada uno de tamaño n/b, la solución toma tiempo:

$$T(n) = aT(n/b) + D(n) + C(n),$$

donde $D(n)$ es el tiempo utilizado para subdividir en subproblemas y $C(n)$ es el tiempo utilizado para combinar las soluciones.

Análisis del mergesort

Supondremos por un momento que n es potencia de 2. Tenemos

- $D(n)$ (dividir): línea 2: tiempo constante k_1
- $C(n)$ (combinar): procedimiento MERGE (línea 5): $\Theta(n) = k_2 n$

Luego
$T(n) = c$ si $n = 1$ y, si $n > 1$

$$T(n) = 2T(n/2) + k_2 n + k_1 = 2T(n/2) + cn$$

Por facilidad (solo por esta vez), supondremos que $k_2 n + k_1 = cn$. Tenemos el siguiente árbol:

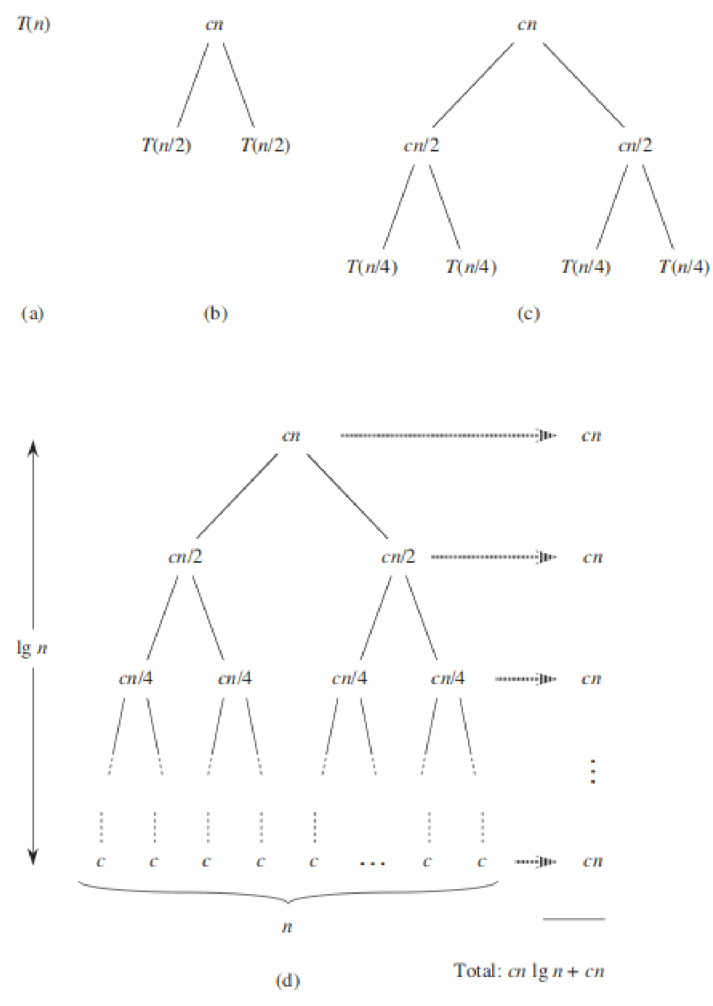

Figure 6: Tomada del libro Cormen, Introduction to Algorithms

Luego $T(n) = cn \lg n + cn = \Theta(n \lg n)$.

El método nos da una intuición, pero aun no es tan formal, en la siguiente sección veremos como resolver recurrencias.

3 Recurrencias

Ejemplo 3.1. *Sea* $F : \mathbb{N} \to \mathbb{R}^{>}$ *definido por*

$$F(n) = \begin{cases} 1 & n = 1 \\ 2F(n-1) + 1 & \textit{caso contrário} \end{cases}$$

Por ejemplo $F(2) = 2F(1) + 1 = 3, F(3) = 2F(2) + 1 = 7, F(4) = 15$. *Cuanto vale* $F(n)$*?*

Solución.

$$\begin{aligned} F(n) &= 2F(n-1) + 1 \\ &= 2(2F(n-2) + 1) + 1 \\ &= 4F(n-2) + 3 \\ &= 4(2F(n-3) + 1) + 3 \\ &= 8F(n-3) + 7 \\ &= \dots \\ &= 2^j F(n-j) + 2^j - 1 \\ &= 2^{n-1} F(1) + 2^{n-1} - 1 \\ &= 2 \cdot 2^{n-1} - 1 \\ &= 2^n - 1 \end{aligned}$$

Luego $F(n) = 2^n - 1 = \Theta(2^n)$.

Ejemplo 3.2. *Sea* $F : \mathbb{N} \to \mathbb{R}^{>}$ *definido por*

$$F(n) = \begin{cases} 1 & n = 1 \\ F(n-1) + n & \textit{caso contrário} \end{cases}$$

Por ejemplo $F(2) = F(1) + 2 = 3, F(3) = F(2) + 3 = 6$. *Cuanto vale* $F(n)$*?*

Solución.

$$\begin{aligned} F(n) &= F(n-1) + n \\ &= F(n-2) + (n-1) + n \\ &= F(n-3) + (n-2) + (n-1) + n \\ &= \dots \\ &= F(n-j) + (n-j+1) + \cdots + n \\ &= F(1) + 2 + 3 + \cdots + n \\ &= 1 + 2 + 3 + \cdots + n \\ &= \frac{n(n+1)}{2} \end{aligned}$$

Luego $F(n) = \Theta(n^2)$.

Ejemplo 3.3. *Sea* $F : \mathbb{N} \to \mathbb{R}^{>}$ *definido por*

$$F(n) = \begin{cases} 1 & n = 1 \\ 2F(\lfloor n/2 \rfloor) + n & \textit{caso contrário} \end{cases}$$

Por ejemplo $F(2) = 2F(1) + 2 = 4, F(3) = 2F(1) + 3 = 5$. *Como podemos acotar* $F(n)$*?*

Solución. Primero, supongamos que n es potencia de 2, osea $n = 2^j$ para algún j. Tenemos,

$$\begin{aligned} F(2^j) &= 2F(2^{j-1}) + 2^j \\ &= 2(2F(2^{j-2}) + 2^{j-1}) + 2^j \\ &= 2^2F(2^{j-2}) + 2^j + 2^j \\ &= 2^3F(2^{j-3}) + 2^j + 2^j + 2^j \\ &= 2^3F(2^{j-3}) + 3 \cdot 2^j \\ &= 2^iF(2^{j-i}) + i \cdot 2^j \\ &= 2^jF(1) + j \cdot 2^j \\ &= (j+1)2^j \\ &= (\lg n + 1)n. \end{aligned}$$

Ahora, suponga que n un número natural cualesquiera. Sea j tal que $2^j \leq n < 2^{j+1}$. Como F es creciente (ver final), tenemos que

$$F(n) < F(2^{j+1}) = (j+2) \cdot 2^{j+1} \leq 3j \cdot 2^{j+1} = 6j \cdot 2^j \leq 6n \lg n$$

(la última desigualdad vale porque $j \leq \lg n$). y también,

$$F(n) \geq F(2^j) = (j+1)2^j = \frac{1}{2}(j+1)2^{j+1} > \frac{1}{2}n \lg n$$

(la última desigualdad vale porque $j + 1 > \lg n$). Concluimos que $F(n) = \Theta(n \lg n)$.

Finalmente, probaremos que $F(n)$ es creciente. Debemos probar que $F(n) < F(n+1)$ para todo $n \geq 1$. Probaremos por inducción en n. Si $n = 1$, tenemos que $F(1) = 1 < 4 = F(n+1)$.

Si $n > 1$, tenemos dos casos. Si n es par, tenemos que $F(n) < F(n) + 1 = 2F(n/2) + n + 1 = 2F(\left\lfloor \frac{n+1}{2} \right\rfloor) + n + 1 = F(n+1)$. Si n es impar, tenemos que $F(n) = 2F(\frac{n-1}{2}) + n < 2F(\frac{n+1}{2}) + n < 2F(\frac{n+1}{2}) + n + 1 = F(n+1)$.

3.1 Teorema Maestro

Sean $a \geq 1$, $b \geq 2$, $k \geq 0$, $n_0 \geq 1 \in \mathbb{N}$. Sea $c \in \mathbb{R}^{>}$. Sea $F : \mathbb{N} \to \mathbb{R}^{>}$ una función no decreciente tal que

$$F(n) = aF(n/b) + cn^k$$

para $n = n_0 b^1, n_0 b^2, n_0 b^3, \ldots$. Se cumple que

- Si $\lg a / \lg b > k$ entonces $F(n) = \Theta(n^{\lg a / \lg b})$
- Si $\lg a / \lg b = k$ entonces $F(n) = \Theta(n^k \lg n)$
- Si $\lg a / \lg b < k$ entonces $F(n) = \Theta(n^k)$

Cuando $b = 2$ tenemos que

- Si $\lg a > k$ entonces $F(n) = \Theta(n^{\lg a})$
- Si $\lg a = k$ entonces $F(n) = \Theta(n^k \lg n)$
- Si $\lg a < k$ entonces $F(n) = \Theta(n^k)$