

ADA

Divide & Conquer

Angel Napa

March 25, 2024

1 Divide & Conquer Method

Method that uses recursion to solve problems. The recursion consists in 3 steps

- **Divide** the problem in one or more subproblems
(In MERGESORT: divide the array of size n in two subsequences of size $n/2$.)
- **Conquistar**: Solving the subproblems recursively.
If the size is small enough, solve it directly
(In MERGESORT: sort the two subsequences using MERGESORT.)
- **Combine** the subproblem solutions to form a solution to the original problem.
(In MERGESORT: Sorting the array with both halves already sorted)

Let's analyze the MERGE-SORT algorithm. This algorithm uses MERGE as a subtask, that receives $A[1 \dots n]$ and three indices p, q, r such that $A[p \dots q]$ and $A[q + 1 \dots r]$ are already sorted, and sorts the subarray $A[p \dots r]$.

```
MERGE-SORT( $A, p, r$ )
1  if  $p \geq r$                                 // zero or one element?
2      return
3   $q = \lfloor (p + r)/2 \rfloor$                         // midpoint of  $A[p : r]$ 
4  MERGE-SORT( $A, p, q$ )                          // recursively sort  $A[p : q]$ 
5  MERGE-SORT( $A, q + 1, r$ )                      // recursively sort  $A[q + 1 : r]$ 
6  // Merge  $A[p : q]$  and  $A[q + 1 : r]$  into  $A[p : r]$ .
7  MERGE( $A, p, q, r$ )
```

Figure 1: Cormen, Introduction to Algorithms

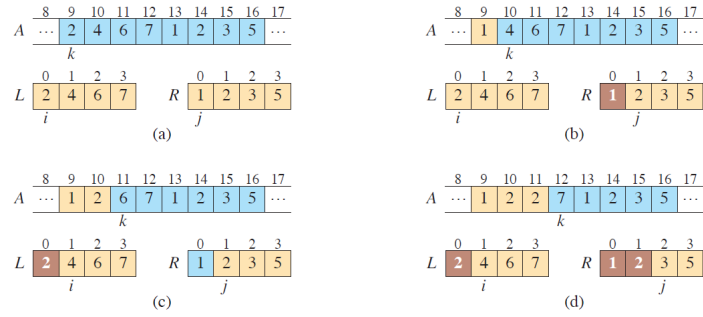


Figure 2: Cormen, Introduction to Algorithms

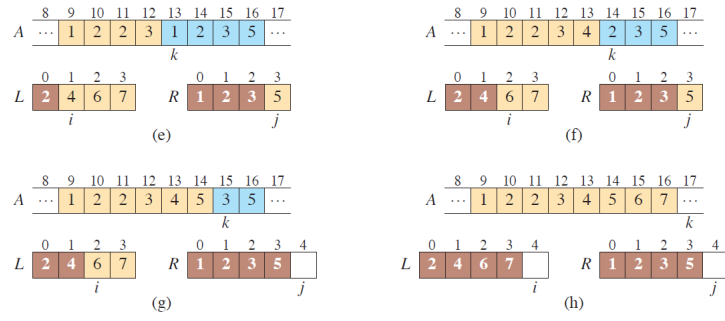


Figure 3: Cormen, Introduction to Algorithms

```

MERGE( $A, p, q, r$ )
1   $n_L = q - p + 1$       // length of  $A[p : q]$ 
2   $n_R = r - q$           // length of  $A[q + 1 : r]$ 
3  let  $L[0 : n_L - 1]$  and  $R[0 : n_R - 1]$  be new arrays
4  for  $i = 0$  to  $n_L - 1$  // copy  $A[p : q]$  into  $L[0 : n_L - 1]$ 
5       $L[i] = A[p + i]$ 
6  for  $j = 0$  to  $n_R - 1$  // copy  $A[q + 1 : r]$  into  $R[0 : n_R - 1]$ 
7       $R[j] = A[q + j + 1]$ 
8   $i = 0$                 //  $i$  indexes the smallest remaining element in  $L$ 
9   $j = 0$                 //  $j$  indexes the smallest remaining element in  $R$ 
10  $k = p$                 //  $k$  indexes the location in  $A$  to fill
11 // As long as each of the arrays  $L$  and  $R$  contains an unmerged element,
    // copy the smallest unmerged element back into  $A[p : r]$ .
12 while  $i < n_L$  and  $j < n_R$ 
13     if  $L[i] \leq R[j]$ 
14          $A[k] = L[i]$ 
15          $i = i + 1$ 
16     else  $A[k] = R[j]$ 
17          $j = j + 1$ 
18      $k = k + 1$ 
19 // Having gone through one of  $L$  and  $R$  entirely, copy the
    // remainder of the other to the end of  $A[p : r]$ .
20 while  $i < n_L$ 
21      $A[k] = L[i]$ 
22      $i = i + 1$ 
23      $k = k + 1$ 
24 while  $j < n_R$ 
25      $A[k] = R[j]$ 
26      $j = j + 1$ 
27      $k = k + 1$ 

```

Figure 4: Cormen, Introduction to Algorithms

Let's analyze the MERGE subtask.

Invariant: At the start of each iteration of the first **while** bucle (lines 12–18), the subarray $A[p \dots k - 1]$ contains the $k - p$ smallest elements between $L[0 \dots n_L - 1]$ and $R[0 \dots n_R - 1]$, sorted. Also, $L[i]$ y $R[j]$ are the smallest elements of each array that are not in that subarray.

Proof:

- **Inicialization** $k = p$, luego $A[p \dots k - 1] = \emptyset$

- **Maintenance**

Case 1: $L[i] \leq R[j]$. Then, we execute line 14. Since $A[p \dots k - 1]$ was sorted with the smallest elements, then $A[p \dots k]$ will have the $k - p + 1$ smallest elements. Case 2: $L[i] > R[j]$: similar.

- **Termination**

Let $k = pos + 1$. Then $A[p \dots k - 1] = A[p \dots pos]$ contains the $k - p = pos - p + 1$ smallest elements of $L[0 \dots n_L - 1]$ y $R[0 \dots n_R - 1]$. If $pos = r$,

we already finished sorting all of the elements of the subarray $A[p \dots r]$. There is nothing to prove. Otherwise, we are just adding the remaining numbers, that are already sorted, in the back of the subarray. (for more formal proof we should prove another two invariants). Either way, we sort the entire subarray.

Execution time of the subtask MERGE:

- Líneas 1–2, 8–11, 19: constant time
- Líneas 3–7: time $\Theta(n_1 + n_2) = \Theta(n)$.
- Líneas 12–27: time $\Theta(n)$.

Initial call: MERGE-SORT($A, 1, A.length$).

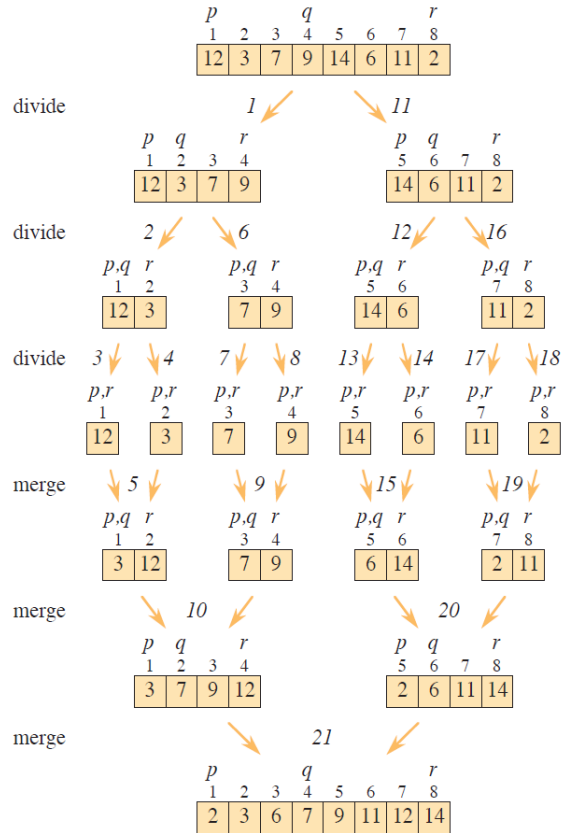


Figure 5: Cormen, Introduction to Algorithms.

2 Runtime Analysis

Generally speaking, when an algorithm call itself in its definition, we describe its execution time $T(n)$ by recurrence.

If the input size $n \leq n_0$, the solution takes a constant time: $T(n) = k$. If $n > n_0$ and a subproblems, each from size n/b , the solution takes:

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

Where $D(n)$ is the amount of time used to divide the problems into subproblems and $C(n)$ is the time used to combine the subproblems.

Mergesort Analysis

Suppose for one moment that n is a power of 2. In this case

- $D(n)$ (divide). line 3: constant time k_1 .
- $C(n)$ (combine): subtask MERGE (line 7): $\Theta(n) = k_2n$.

Then, $T(n) = c$ if $n = 1$, otherwise, if $n > 1$,

$$T(n) = 2T(n/2) + k_2n + k_1 = 2T(n/2) + cn.$$

For simplicity (just this time), we are assuming that $k_2n + k_1 = cn$. We have the following tree:

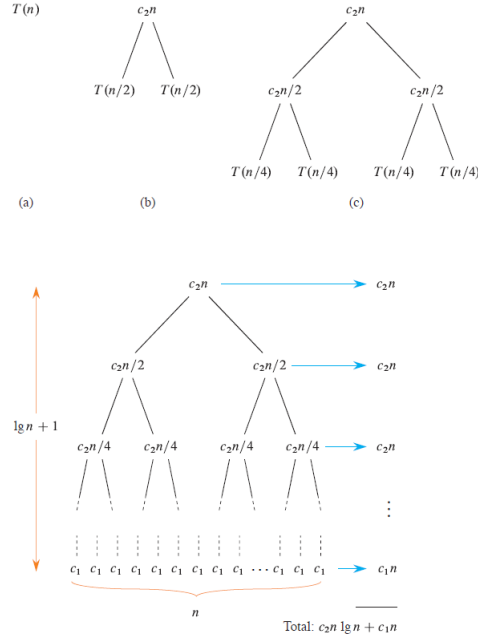


Figure 6: Tomada del libro Cormen, Introduction to Algorithms

Then, we can claim intuitively that $T(n) = cn \lg n + cn = \Theta(n \lg n)$.

This method gives us an intuition, but is not formal. In the next section we will see how to solve recurrences

3 Recurrences

A recurrence is a function that depends of itself in its definition. Stating and solving recurrences will help us find upper and lower bounds of the execution time of a divide & conquer algorithm

3.1 Explicit solution

We will see in this section a formal and more detailed way to solve recurrences.

Example 3.1. Let $F : \mathbb{N} \rightarrow \mathbb{R}^+$ defined as

$$F(n) = \begin{cases} 1 & n = 1 \\ 2F(n-1) + 1 & \text{otherwise} \end{cases}$$

For example: $F(2) = 2F(1) + 1 = 3, F(3) = 2F(2) + 1 = 7, F(4) = 15$. What is the value of $F(n)$?

First solution

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

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

Second solution

For arbitrary k , we have:

$$F(k) - 2F(k-1) = 1$$

Thus, multiplying both sides by 2^{n-k} we have

$$\begin{aligned} 2^{n-k}F(k) - 2^{n-k+1}F(k-1) &= 2^{n-k} \quad \forall 2 \leq k \leq n \\ a_k - a_{k-1} &= 2^{n-k} \end{aligned}$$

Where a_k is defined as $2^{n-k}F(k)$. Adding up all possible k we have

$$\begin{aligned} F(n) - 2^{n-1} &= F(n) - 2^{n-1}F(1) \\ &= a_n - a_1 \\ &= \sum_{k=2}^n a_k - a_{k-1} \\ &= \sum_{k=2}^n 2^{n-k} \\ &= \sum_{j=0}^{n-2} 2^j \\ &= 2^{n-1} - 1 \end{aligned}$$

Adding both sides 2^{n-1} we also obtain here that $F(n) = 2^n - 1$.

Third solution

Let $G(n)$ be the function defined as:

$$G(n) = F(n) + 1$$

Using the recurrence of F , we can also find a recurrence for G :

$$G(n) = \begin{cases} 2 & n = 1 \\ 2G(n-1) & \text{otherwise} \end{cases}$$

This automatically tell us that G is a exponential form. so

$$G(n) = 2^n$$

We conclude that $F(n) = G(n) - 1 = 2^n - 1$.

Fourth solution

Using induction. This is left as an exercise to the reader.

Example 3.2. Let $F : \mathbb{N} \rightarrow \mathbb{R}^+$ define as

$$F(n) = \begin{cases} 1 & : n = 1 \\ F(n-1) + n & : \text{otherwise} \end{cases}$$

For example $F(2) = F(1) + 2 = 3$, $F(3) = F(2) + 3 = 6$. What is the value of $F(n)$?

Solution

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

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

Example 3.3. Let $F : \mathbb{N} \rightarrow \mathbb{R}^+$ defined as

$$F(n) = \begin{cases} 1 & : n = 1 \\ 2F(\lfloor n/2 \rfloor) + n & : \text{otherwise} \end{cases}$$

For example $F(2) = 2F(1) + 2 = 4$, $F(3) = 2F(1) + 3 = 5$. How can we find bounds of $F(n)$?

Solution First, suppose that n is a power of 2, this means $n = 2^j$ for some j . We have,

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

Now, suppose that n is an arbitrary positive integer. Let j such that $2^j \leq n < 2^{j+1}$. Since F is increasing (see the end), we have that

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

(this last inequality holds because $j \leq \lg n$). furthermore,

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

(this last inequality holds because $j+1 > \lg n$). We conclude that $F(n) = \Theta(n \lg n)$.

Finally, we will show that $F(n)$ is increasing. We have to prove that $F(n) < F(n+1)$ for all $n \geq 1$. The proof will use induction on n . If $n = 1$, we have that $F(1) = 1 < 4 = F(2)$.

If $n > 1$, we have 2 cases:

If n is even, we have that

$$F(n) < F(n) + 1 = 2F\left(\frac{n}{2}\right) + n + 1 = 2F\left(\left\lfloor \frac{n+1}{2} \right\rfloor\right) + n + 1 = F(n+1)$$

Else, if n is odd, we have:

$$F(n) = 2F\left(\frac{n-1}{2}\right) + n < 2F\left(\frac{n+1}{2}\right) + n < 2F\left(\frac{n+1}{2}\right) + n + 1 = F(n+1)$$

Practice more exercises in the sheet of exercises.

3.2 Proofs by induction

Example 3.4. Let $T : \mathbb{N} \rightarrow \mathbb{R}^+$ defined as

$$T(n) = \begin{cases} 1 & n = 1 \\ 2T(\lfloor \frac{n}{2} \rfloor) + n & \text{otherwise} \end{cases}$$

Prove by induction that $T(n) = O(n^2)$. Prove by induction that $T(n) = \Omega(n \lg n)$.

First we will prove by induction on n that $T(n) \leq 2n^2$ for $n \geq 1$. If $n = 1$, we have $T(1) = 1 \leq 2 \leq 2n^2$. If $n > 1$, we have that

$$\begin{aligned} T(n) &= 2T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + n \\ &\leq 4\left(\left\lfloor \frac{n}{2} \right\rfloor\right)^2 + n \\ &\leq 4\left(\frac{n}{2}\right)^2 + n^2 \\ &= 2n^2. \end{aligned}$$

Now we will prove by induction on n that $T(n) \geq \frac{1}{4}n \lg n$ for $n \geq 4$. If $n = 4$,

we have that $T(4) = 12 \geq \frac{1}{4}4 \lg 4 = \frac{1}{4}n \lg n$. If $n > 4$, we have that

$$\begin{aligned}
T(n) &= 2T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + n \\
&\geq 2 \cdot \frac{1}{4} \left\lfloor \frac{n}{2} \right\rfloor \lg \left\lfloor \frac{n}{2} \right\rfloor + n \\
&\geq \frac{1}{2} \left(\frac{n}{2} - 1\right) \lg \left(\frac{n}{2} - 1\right) + n \\
&\geq \frac{1}{2} \left(\frac{n}{2} - 1\right) \lg \left(\frac{n}{4}\right) + n \\
&= \frac{1}{2} \left(\frac{n}{2} - 1\right) (\lg n - 2) + n \\
&= \frac{1}{4}(n \lg n) + n/2 + 2 - \lg n/2 \\
&\geq \frac{1}{4}(n \lg n).
\end{aligned}$$

Example 3.5. Let $T : \mathbb{N} \rightarrow \mathbb{R}^+$ defined as

$$T(n) = \begin{cases} 1 & n = 1 \\ 2T(\lfloor \frac{n}{2} \rfloor) + 1 & \text{otherwise} \end{cases}$$

Prove by induction that $T(n) = O(n)$.

We will prove by induction on n that $T(n) \leq 2n - 1$ for each $n \geq 1$. If $n = 1$, we have $T(1) = 1 = 2 - 1 \leq 2n - 1$. If $n > 1$, we have

$$\begin{aligned}
T(n) &= 2T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + 1 \\
&\leq 2 \left(2 \left\lfloor \frac{n}{2} \right\rfloor - 1\right) + 1 \\
&= 4 \left\lfloor \frac{n}{2} \right\rfloor - 1 \\
&\leq 4 \cdot \frac{n}{2} - 1 \\
&= 2n - 1.
\end{aligned}$$

Thus, $T(n) \leq 2n - 1 \leq 2n$ para $n \geq 1$, therefore $T(n) = O(n)$.

3.3 Master Theorem

Let $a \geq 1$, $b \geq 2$, $k \geq 0$, $n_0 \geq 1 \in \mathbb{N}$; $c \in \mathbb{R}^+$. Let $F : \mathbb{N} \rightarrow \mathbb{R}^+$ a nondecreasing function such that

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

for $n = n_0b^1, n_0b^2, n_0b^3, \dots$

It holds that

- If $\lg a / \lg b > k$ then $F(n) = \Theta(n^{\lg a / \lg b})$.

- If $\lg a / \lg b = k$ then $F(n) = \Theta(n^k \lg n)$.
- If $\lg a / \lg b < k$ then $F(n) = \Theta(n^k)$.

In particular, when $b = 2$ we have

- If $\lg a > k$ then $F(n) = \Theta(n^{\lg a})$.
- If $\lg a = k$ then $F(n) = \Theta(n^k \lg n)$.
- If $\lg a < k$ then $F(n) = \Theta(n^k)$.