

## INSERTION-SORT( $A$ )

```
1  for  $j \leftarrow 2$  to  $\text{length}[A]$ 
2      do  $\text{key} \leftarrow A[j]$ 
3           $\triangleright$  Insert  $A[j]$  into the sorted sequence  $A[1 \dots j - 1]$ .
4           $i \leftarrow j - 1$ 
5          while  $i > 0$  and  $A[i] > \text{key}$ 
6              do  $A[i + 1] \leftarrow A[i]$ 
7                   $i \leftarrow i - 1$ 
8           $A[i + 1] \leftarrow \text{key}$ 
```

## Loop invariants and the correctness of insertion sort

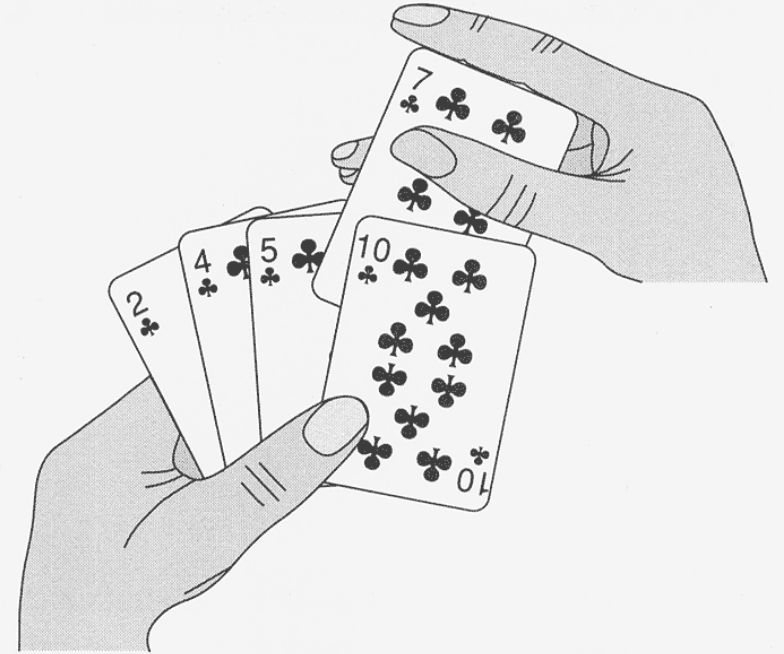


Figure 2.1 Sorting a hand of cards using insertion sort.

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

$t_j$  is the number of times the while loop test in line 5 is executed for that value of  $j$ .

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

**$T(n) O(n^2)$  , In Place sorting**

# Divide-and-Conquer: MERGE-SORT

```
MERGE-SORT( $A, p, r$ )
```

```
1  if  $p < r$ 
```

```
2      then  $q \leftarrow \lfloor (p + r) / 2 \rfloor$ 
```

```
3          MERGE-SORT( $A, p, q$ )
```

```
4          MERGE-SORT( $A, q + 1, r$ )
```

```
5          MERGE( $A, p, q, r$ )
```

Check for base case

Divide

Conquer

Conquer

combine

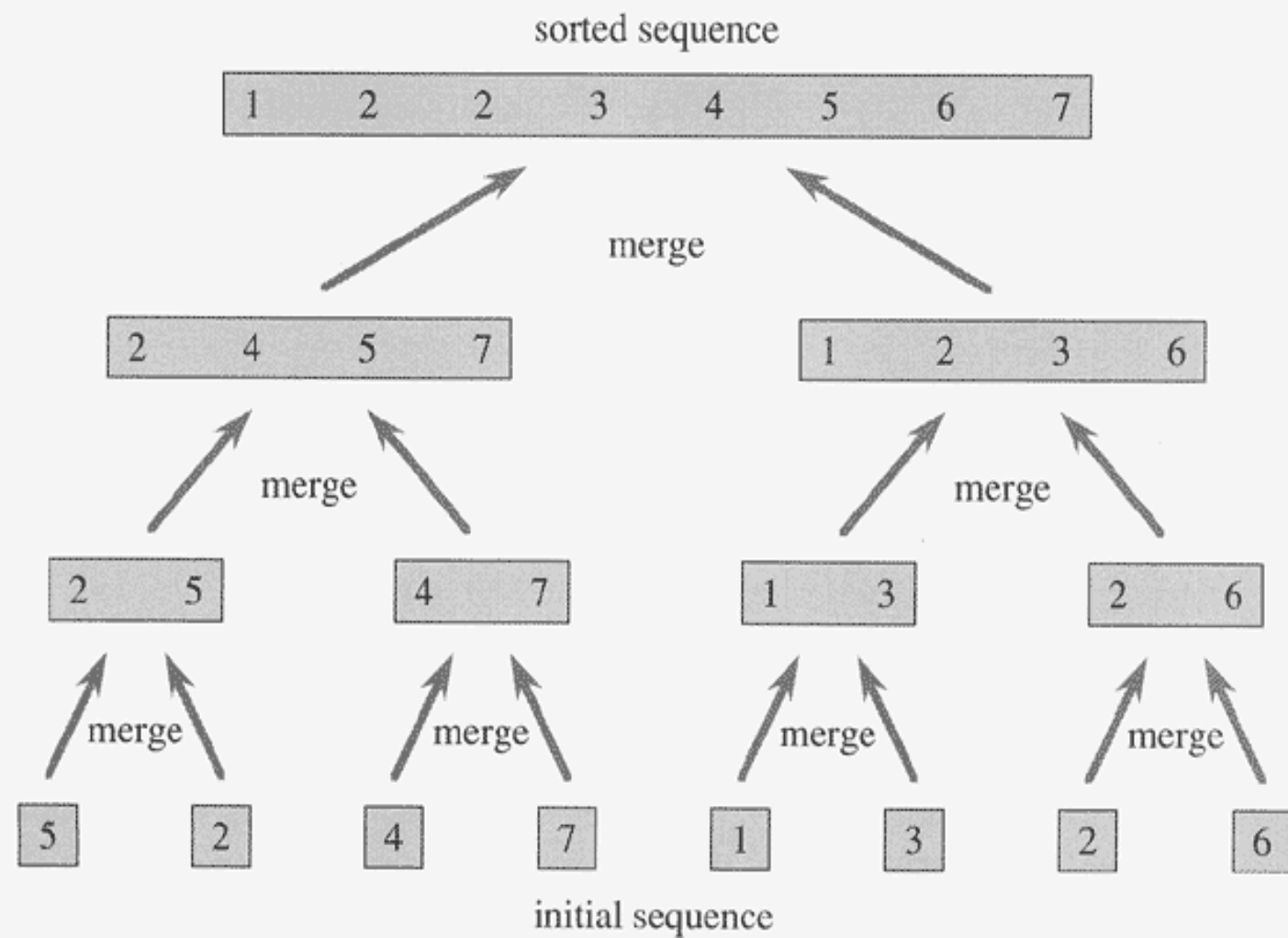
MERGE( $A, p, q, r$ )

```
1   $n_1 \leftarrow q - p + 1$ 
2   $n_2 \leftarrow r - q$ 
3  create arrays  $L[1..n_1 + 1]$  and  $R[1..n_2 + 1]$ 
4  for  $i \leftarrow 1$  to  $n_1$ 
5      do  $L[i] \leftarrow A[p + i - 1]$ 
6  for  $j \leftarrow 1$  to  $n_2$ 
7      do  $R[j] \leftarrow A[q + j]$ 
8   $L[n_1 + 1] \leftarrow \infty$ 
9   $R[n_2 + 1] \leftarrow \infty$ 
10  $i \leftarrow 1$ 
11  $j \leftarrow 1$ 
12 for  $k \leftarrow p$  to  $r$ 
13     do if  $L[i] \leq R[j]$ 
14         then  $A[k] \leftarrow L[i]$ 
15              $i \leftarrow i + 1$ 
16         else  $A[k] \leftarrow R[j]$ 
17              $j \leftarrow j + 1$ 
```

MERGE() requires *extra* space – arrays L and R – of the size of the input + 2.

What is the time complexity of MERGE?

**Ques:** Could the merging be done *in-place*?



**Figure 2.4** The operation of merge sort on the array  $A = \langle 5, 2, 4, 7, 1, 3, 2, 6 \rangle$ . The lengths of the sorted sequences being merged increase as the algorithm progresses from bottom to top.

## Analyzing divide-and-conquer algorithms

Use a *recurrence equation* (more commonly, a *recurrence*) to describe the running time of a divide-and-conquer algorithm.

Let  $T(n)$  = running time on a problem of size  $n$ .

- If the problem size is small enough (say,  $n \leq c$  for some constant  $c$ ), we have a base case. The brute-force solution takes constant time:  $\Theta(1)$ .
- Otherwise, suppose that we divide into  $a$  subproblems, each  $1/b$  the size of the original. (In merge sort,  $a = b = 2$ .)
- Let the time to divide a size- $n$  problem be  $D(n)$ .
- Have  $a$  subproblems to solve, each of size  $n/b \Rightarrow$  each subproblem takes  $T(n/b)$  time to solve  $\Rightarrow$  we spend  $aT(n/b)$  time solving subproblems.
- Let the time to combine solutions be  $C(n)$ .
- We get the recurrence

$$T(n) = \begin{cases} \Theta(1) & \text{if } n \leq c, \\ aT(n/b) + D(n) + C(n) & \text{otherwise.} \end{cases}$$

## Analyzing merge sort

For simplicity, assume that  $n$  is a power of 2  $\Rightarrow$  each divide step yields two sub-problems, both of size exactly  $n/2$ .

The base case occurs when  $n = 1$ .

When  $n \geq 2$ , time for merge sort steps:

**Divide:** Just compute  $q$  as the average of  $p$  and  $r \Rightarrow D(n) = \Theta(1)$ .

**Conquer:** Recursively solve 2 subproblems, each of size  $n/2 \Rightarrow 2T(n/2)$ .

**Combine:** MERGE on an  $n$ -element subarray takes  $\Theta(n)$  time  $\Rightarrow C(n) = \Theta(n)$ .

Since  $D(n) = \Theta(1)$  and  $C(n) = \Theta(n)$ , summed together they give a function that is linear in  $n$ :  $\Theta(n) \Rightarrow$  recurrence for merge sort running time is

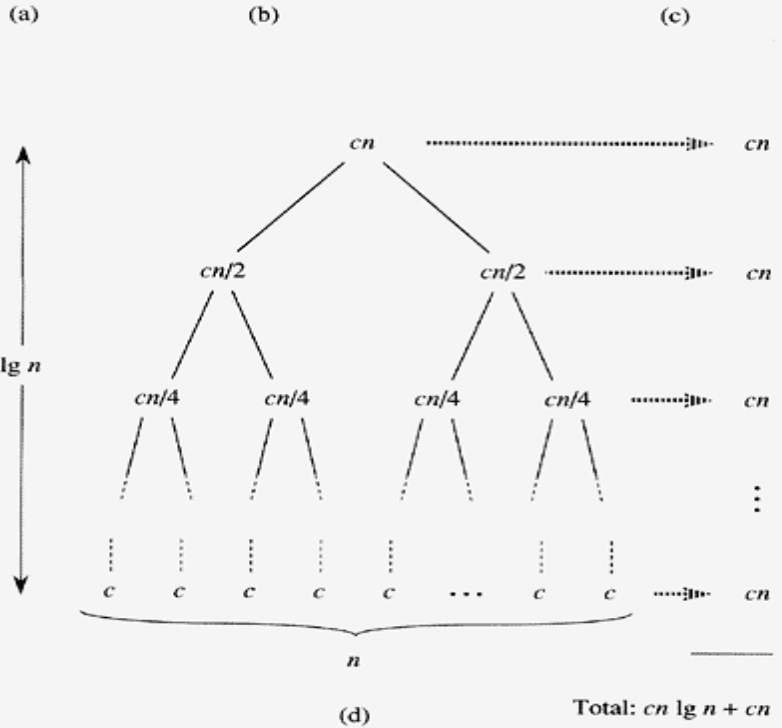
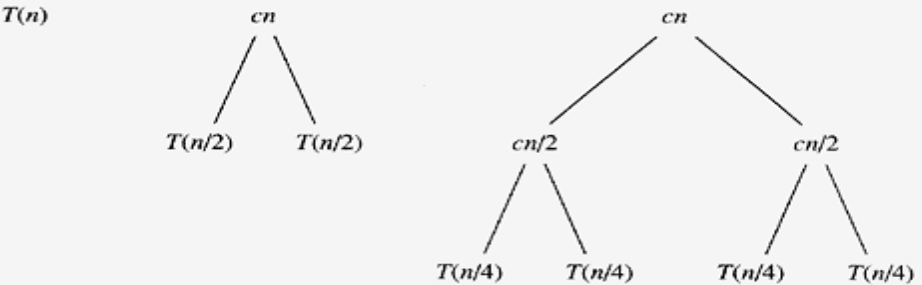
$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1, \\ 2T(n/2) + \Theta(n) & \text{if } n > 1. \end{cases}$$

The recurrence for the worst-case running time  $T(n)$  of MERGE-SORT:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 2T(n/2) + \Theta(n) & \text{if } n > 1 \end{cases}$$

equivalently

$$T(n) = \begin{cases} c_1 & \text{if } n = 1 \\ 2T(n/2) + c_2n & \text{if } n > 1 \end{cases}$$



**Figure 2.5** The construction of a recursion tree for the recurrence  $T(n) = 2T(n/2) + cn$ . Part (a) shows  $T(n)$ , which is progressively expanded in (b)–(d) to form the recursion tree. The fully expanded tree in part (d) has  $\lg n + 1$  levels (i.e., it has height  $\lg n$ , as indicated), and each level contributes a total cost of  $cn$ . The total cost, therefore, is  $cn \lg n + cn$ , which is  $\Theta(n \lg n)$ .