MERGE SORT

Merge sort is based on divide-and-conquer technique. It sorts a given array $A[0..n-1]$ by dividing it into two halves $A[0..\lfloor n/2\rfloor-1]$ and $A[\lfloor n/2\rfloor..n-1]$, sorting each of them recursively, and then merging the two smaller sorted arrays into a single sorted one.

ALGORITHM *Merge sort (A[0..n − 1])*

//Sorts array $A[0..n-1]$ by recursive merge sort

//Input: An array $A[0..n-1]$ of orderable elements

//Output: Array $A[0..n-1]$ sorted in non-decreasing order

if $n > 1$

copy $A[0..\lfloor n/2\rfloor-1]$ to $B[0..\lfloor n/2\rfloor-1]$

copy $A[\lfloor n/2\rfloor..n-1]$ to $C[0..\lceil n/2\rceil-1]$

Merge sort$(B[0..\lfloor n/2\rfloor-1])$

Merge sort$(C[0..\lceil n/2\rceil-1])$

Merge (B, C, A) //see below

The ***merging*** of two sorted arrays can be done as follows. Two pointers (array indices) are initialized to point to the first elements of the arrays being merged.

The elements pointed to are compared, and the smaller of them is added to a new array being constructed; after that, the index of the smaller element is incremented to point to its immediate successor in the array it was copied from.

This operation is repeated until one of the two given arrays is exhausted, and then the remaining elements of the other array are copied to the end of the new array.

ALGORITHM *Merge*($B[0..p-1]$, $C[0..q-1]$, $A[0..p+q-1]$)
//Merges two sorted arrays into one sorted array
//Input: Arrays $B[0..p-1]$ and $C[0..q-1]$ both sorted
//Output: Sorted array $A[0..p+q-1]$ of the elements of B and C
$i \leftarrow 0$; $j \leftarrow 0$; $k \leftarrow 0$
while $i < p$ **and** $j < q$ **do**
    if $B[i] \le C[j]$
        $A[k] \leftarrow B[i]$; $i \leftarrow i + 1$
    else $A[k] \leftarrow C[j]$; $j \leftarrow j + 1$
    $k \leftarrow k + 1$
if $i = p$
    copy $C[j..q-1]$ to $A[k..p+q-1]$
else copy $B[i..p-1]$ to $A[k..p+q-1]$

The operation of the algorithm on the list 8, 3, 2, 9, 7, 1, 5, 4 is illustrated in Figure 2.10

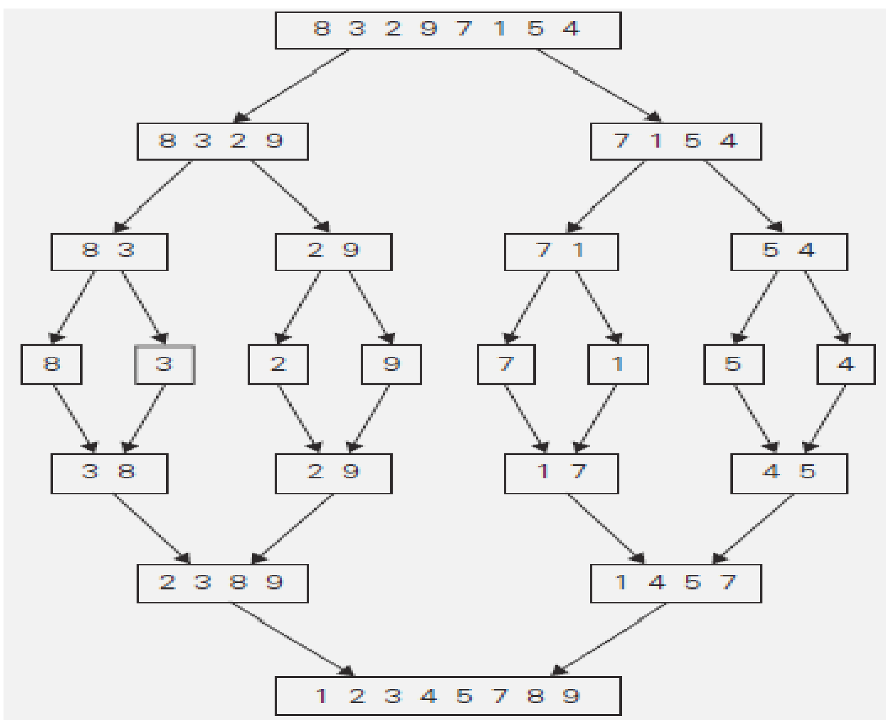

FIGURE 2.10 Example of merge sort operation

The recurrence relation for the number of key comparisons $C(n)$ is

$$C(n) = 2C(n/2) + C_{merge}(n) \text{ for } n > 1, C(1) = 0.$$

In the worst case, $C_{merge}(n) = n - 1$, and we have the recurrence

By Master Theorem, $C_{worst}(n) \in \Theta(n \log n)$

the exact solution to the worst-case recurrence for $n = 2^k$

$$C_{worst}(n) = n \log_2 n - n + 1.$$

For large n, the number of comparisons made by this algorithm in the average case turns out to be about $0.25n$ less and hence is also in **Θ *(n* log *n)*.**

First, the algorithm can be implemented bottom up by merging pairs of the array's elements, then merging the sorted pairs, and so on. This avoids the time and space overhead of using a stack to handle recursive calls. Second, we can divide a list to be sorted in more than two parts, sort each recursively, and then merge them together. This scheme, which is particularly useful for sorting files residing on secondary memory devices, is called ***multiway merge sort***.