Rekursion

Divide and Conquer

Divide the problem into a number of subproblems that are smaller instances of the same problem.

Conquer the subproblems by solving them recursively. If the subproblem sizes are small enough, however, just solve the subproblems in a straightforward manner.

Combine the solutions to the subproblems into the solution for the original problem.

Merge Sort

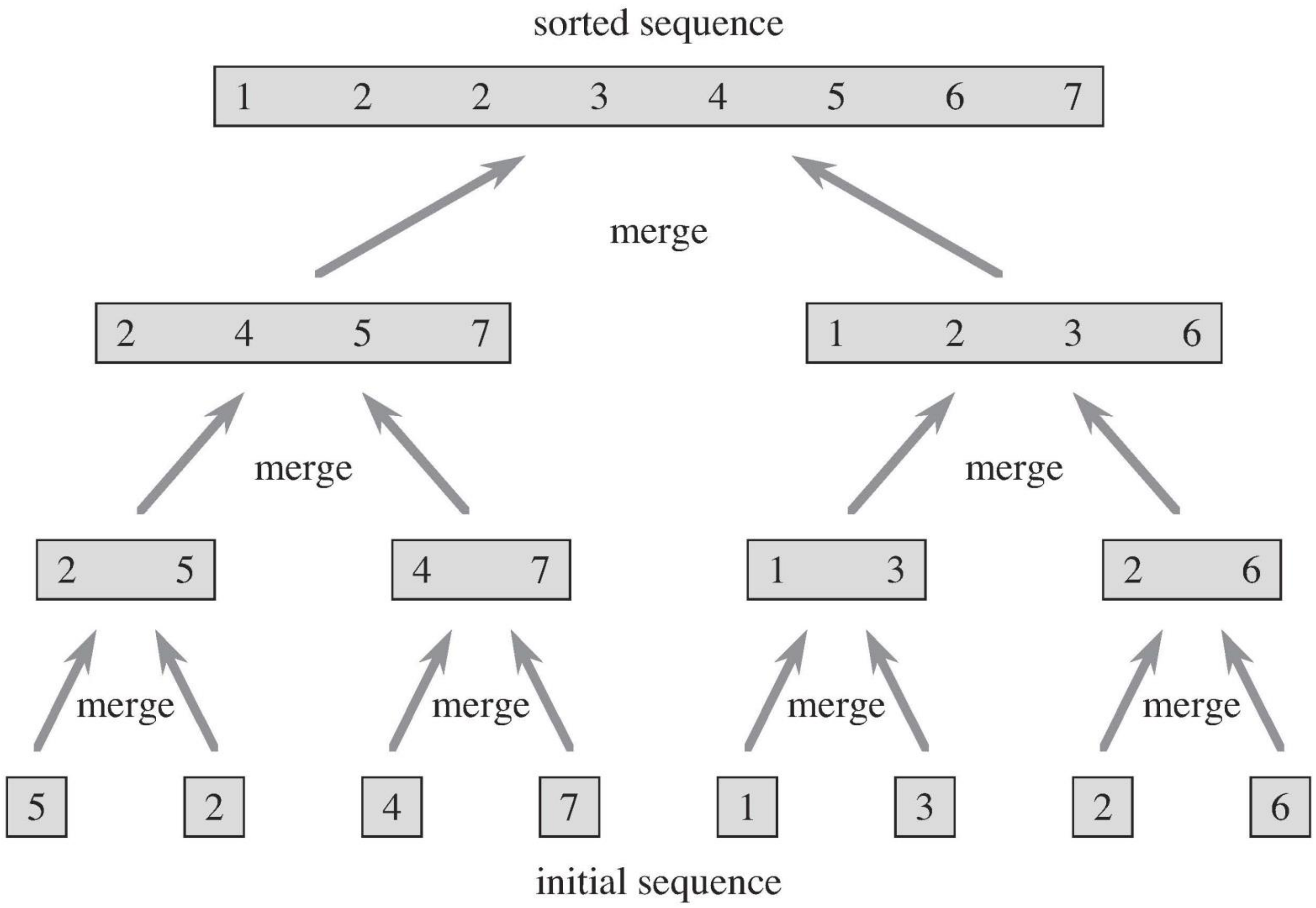
Divide: Divide the n-element sequence to be sorted into two subsequences of $n/2$ elements each.

Conquer: Sort the two subsequences recursively using merge sort.

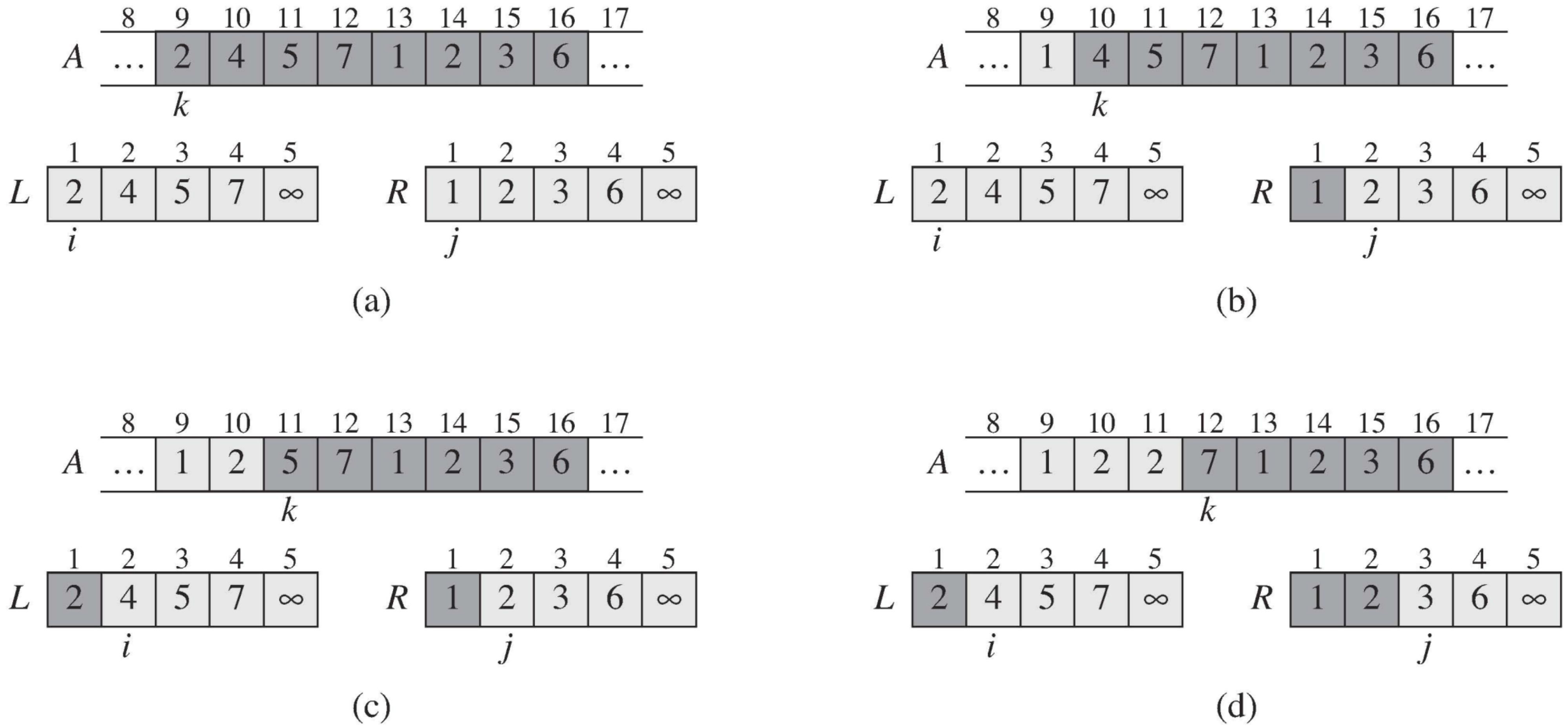
Combine: Merge the two sorted subsequences to produce the sorted answer.

Merge Sort

```
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)
```


sorted sequence
1 2 2 3 4 5 6 7
merge
2 4 5 7
1 2 3 6
merge
merge
2 5
4 7
1 3
2 6
merge
merge
merge
merge
5
2
4
7
1
3
2
6
initial sequence

```
MERGE(A, p, q, r)
n1 = q - p + 1
n2 = r - q
let L[1..n1 + 1] and R[1..n2 + 1] be new arrays
for i = 1 to n1
    L[i] = A[p + i - 1]
for j = 1 to n2
    R[j] = A[q + j]
L[n1 + 1] = ∞
R[n2 + 1] = ∞
i = 1
j = 1
for k = p to r
    if L[i] ≤ R[j]
        A[k] = L[i]
        i = i + 1
    else A[k] = R[j]
        j = j + 1
```


8 9 10 11 12 13 14 15 16 17
A … 2 4 5 7 1 2 3 6 …
k
1 2 3 4 5
L 2 4 5 7 ∞
i
1 2 3 4 5
R 1 2 3 6 ∞
j
(a)
8 9 10 11 12 13 14 15 16 17
A … 1 4 5 7 1 2 3 6 …
k
1 2 3 4 5
L 2 4 5 7 ∞
i
1 2 3 4 5
R 1 2 3 6 ∞
j
(b)
8 9 10 11 12 13 14 15 16 17
A … 1 2 5 7 1 2 3 6 …
k
1 2 3 4 5
L 2 4 5 7 ∞
i
1 2 3 4 5
R 1 2 3 6 ∞
j
(c)
8 9 10 11 12 13 14 15 16 17
A … 1 2 2 7 1 2 3 6 …
k
1 2 3 4 5
L 2 4 5 7 ∞
i
1 2 3 4 5
R 1 2 3 6 ∞
j
(d)

Invariante

At the start of each iteration of the **for** loop of lines 12–17, the subarray $A[p \mathinner{..} k-1]$ contains the $k-p$ smallest elements of $L[1 \mathinner{..} n_1+1]$ and $R[1 \mathinner{..} n_2+1]$, in sorted order. Moreover, $L[i]$ and $R[j]$ are the smallest elements of their arrays that have not been copied back into A.

Initialisierung

Initialization: Prior to the first iteration of the loop, we have $k = p$, so that the subarray $A[p \mathinner{.\,.} k-1]$ is empty. This empty subarray contains the $k - p = 0$ smallest elements of L and R, and since $i = j = 1$, both $L[i]$ and $R[j]$ are the smallest elements of their arrays that have not been copied back into A.

Erhaltung

Maintenance: To see that each iteration maintains the loop invariant, let us first suppose that $L[i] \leq R[j]$. Then $L[i]$ is the smallest element not yet copied back into A. Because $A[p \mathinner{..} k-1]$ contains the $k-p$ smallest elements, after line 14 copies $L[i]$ into $A[k]$, the subarray $A[p \mathinner{..} k]$ will contain the $k-p+1$ smallest elements. Incrementing k (in the **for** loop update) and i (in line 15) reestablishes the loop invariant for the next iteration. If instead $L[i] > R[j]$, then lines 16–17 perform the appropriate action to maintain the loop invariant.

Terminierung

Termination: At termination, $k = r + 1$. By the loop invariant, the subarray $A[p \mathinner{.\,.} k - 1]$, which is $A[p \mathinner{.\,.} r]$, contains the $k - p = r - p + 1$ smallest elements of $L[1 \mathinner{.\,.} n_1 + 1]$ and $R[1 \mathinner{.\,.} n_2 + 1]$, in sorted order. The arrays L and R together contain $n_1 + n_2 + 2 = r - p + 3$ elements. All but the two largest have been copied back into A, and these two largest elements are the sentinels.

Analyse I

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

Analyse II