```
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] ≤ 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
```

```
MERGE-SORT(A, p, r)
1  if p ≥ r                          // zero or one element?
2      return
3  q = ⌊(p + r)/2⌋                   // 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)
```