Merge Sort Algorithm

```
MERGE-SORT(A, p, r)
if p ≥ r                              // zero or one element?
    return
q = ⌊(p + r)/2⌋                       // midpoint of A[p : r]
MERGE-SORT(A, p, q)                   // recursively sort A[p : q]
MERGE-SORT(A, q + 1, r)               // recursively sort A[q + 1 : r]
// Merge A[p : q] and A[q + 1 : r] into A[p : r].
MERGE(A, p, q, r)
```

```
MERGE(A, p, q, r)
n_L = q − p + 1          // length of A[p : q]
n_R = r − q              // length of A[q + 1 : r]
let L[0 : n_L − 1] and R[0 : n_R − 1] be new arrays
for i = 0 to n_L − 1     // copy A[p : q] into L[0 : n_L − 1]
    L[i] = A[p + i]
for j = 0 to n_R − 1     // copy A[q + 1 : r] into R[0 : n_R − 1]
    R[j] = A[q + j + 1]
i = 0                    // i indexes the smallest remaining element in L
j = 0                    // j indexes the smallest remaining element in R
k = p                    // k indexes the location in A to fill
// As long as each of the arrays L and R contains an unmerged element,
//      copy the smallest unmerged element back into A[p : r].
while i < n_L and j < n_R
    if L[i] ≤ R[j]
        A[k] = L[i]
        i = i + 1
    else A[k] = R[j]
        j = j + 1
    k = k + 1
// Having gone through one of L and R entirely, copy the
//      remainder of the other to the end of A[p : r].
while i < n_L
    A[k] = L[i]
    i = i + 1
    k = k + 1
while j < n_R
    A[k] = R[j]
    j = j + 1
    k = k + 1
```