Input: Array $A = [a_1, a_2, ..., a_n]$.
Result: Sorted array A

procedure MERGE_SORT(A)
  if $|A| \leq 1$ **then return**
  $mid \leftarrow \lfloor |A|/2 \rfloor$
  $A_l \leftarrow [A_1, A_2, ..., A_{mid}]$
  $A_r \leftarrow [A_{mid+1}, A_{mid+2}, ..., A_{|A|}]$

  $Merge\_Sort(A_l)$
  $Merge\_Sort(A_r)$

  $Combine(A, A_l, A_r)$
  return

Input: Original array A, sorted arrays L and R of A corresponding to left and right subarrays of A
Result: L and R combined into A to form a sorted array A

procedure COMBINE(A, L, R)
  $l \leftarrow 0$ ▷ Here, we use 0-based indexing
  $r \leftarrow 0$
  $i \leftarrow 0$ ▷ Index for array A

  while $l < |L|$ OR $r < |R|$ **do**
    if $(l < |L|)$ AND $(r \geq |R|$ OR $L[l] \leq R[r])$ **then**
      $A[i] \leftarrow L[l]$
      $l \leftarrow l + 1$
    else
      $A[i] \leftarrow R[r]$
      $r \leftarrow r + 1$
    $i \leftarrow i + 1$
  return

Input: Array A, left and right indices l and r corresponding to the index bounds of the array that should be sorted
Output: Sorted array A (in-place) [only sorts between l and r inclusive]

```
procedure QUICK_SORT(A, l, r)
    if l < r then
        p ← Partition(A, l, r)
        Quick_Sort(A, l, p − 1)
        Quick_Sort(A, p + 1, r)
```

Input: Array A, left and right indices l and r corresponding to bounds of the array to partition
Output: Partitioned array A (in-place); The function itself returns the index p corresponding to the element e that was chosen for partitioning. Everything to the left of e will be smaller than e, and everything to the right of e will be greater or equal to e.

Lomuto partition scheme

```
procedure PARTITION(A, l, r)
    pivot ← A[r]
    i ← l

    for j ← l to r − 1 do
        if A[j] < pivot then
            Swap A[i] with A[j]
            i ← i + 1

    Swap A[i] with A[r]
    return i
```

Hoare partition scheme

```
procedure PARTITION(A, l, r)
    mid ← l + (r − l)/2
    pivot ← A[mid]
    i ← l − 1
    j ← r + 1
    while true do
        i ← i + 1
        while A[i] < pivot do
            i ← i + 1

        j ← j − 1
        while A[j] > pivot do
            j ← j − 1

        if i < j then
            Swap A[i] with A[j]
        else
            return j
```