Quicksort Pseudocode

5	1	3	2	4

arr: 1D array

low: lowest index, high: highest index

```
function quicksort(arr, low, high)
    if (low < high) then
        p <-- partition(arr, low, high) // returns an index
        quicksort(arr, low, p-1)
        quicksort(arr, p+1, high)
    end if
end function
```

```
function partition(arr, low, high)
    pivot <-- arr[high]
    i <-- low
    for low <= j < high
        if (arr[j] < pivot) then
            swap(arr, i, j)
            i = i + 1
        end if
    end for
    swap(arr, i, high)
    return i
end function
```

```
function swap(arr, i, j)
    temp <-- arr[i]
    arr[i]  <-- arr[j]
    arr[j]  <-- temp
end function
```

function swap c++ version

```
a ^= b
b ^= a
a ^= b
```

Quicksort Analysis: Best Case

arr: 1D array
low: lowest index, high: highest index

```
function quicksort(arr, low, high)                                  T(n)
    if (low < high) then                                            C_0
        p <-- partition(arr, low, high) // returns an index         C_1(n) + c_2
        quicksort(arr, low, p-1)                                    T(n/2)
        quicksort(arr, p+1, high)                                   T(n/2)
    end if
end function
```

```
function partition(arr, low, high)          T(N)
    pivot <-- arr[high]                     C_0
    i <-- low                               C_1
    for low <= j < high                     C_2(n) + C_3
        if (arr[j] < pivot) then            C_3(n-1)
            swap(arr, i, j)                 C_4(n-1)
            i = i + 1                       C_5(n-1)
        end if
    end for
    swap(arr, i, high)                      C_6
    return i                                C_7
end function
```

$$2T(n/2) + C_1 N + C_2 + C_0$$

$$2T(n/2) + \underline{C_1 N + C_3} = 2T(n/2) + \Theta(n)$$

Master theorem in the form T(n) = aT(n/b) + f(n)

Quicksort Analysis: Worst Case

arr: 1D array

low: lowest index, high: highest index

```
function quicksort(arr, low, high)                          T(n)
    if (low < high) then                                    C_0
        p <-- partition(arr, low, high) // returns an index C_1(n) + C_2
        quicksort(arr, low, p-1)                            T(n-1)
        quicksort(arr, p+1, high)                           T(0)
    end if
end function
```

$$T(n) = c_0 + c_1(n) + c_2 + T(n-1) + T(0)$$
$$= c_0 + c_1(n) + c_2 + T(n-1) + c_3$$
$$= c_1(n) + c_4 + T(n-1)$$

$$T(n) = c(n) + c + (c(n-1) + c + t(n-2))$$
$$= 3c + 2cn + t(n-2)$$
$$= 3c + 2cn +-(c(n-2) + c + t(n-3))$$
$$= 5c + 3cn + t(n-3)$$

$$(2k-1)c + kcn + t(n-k)$$

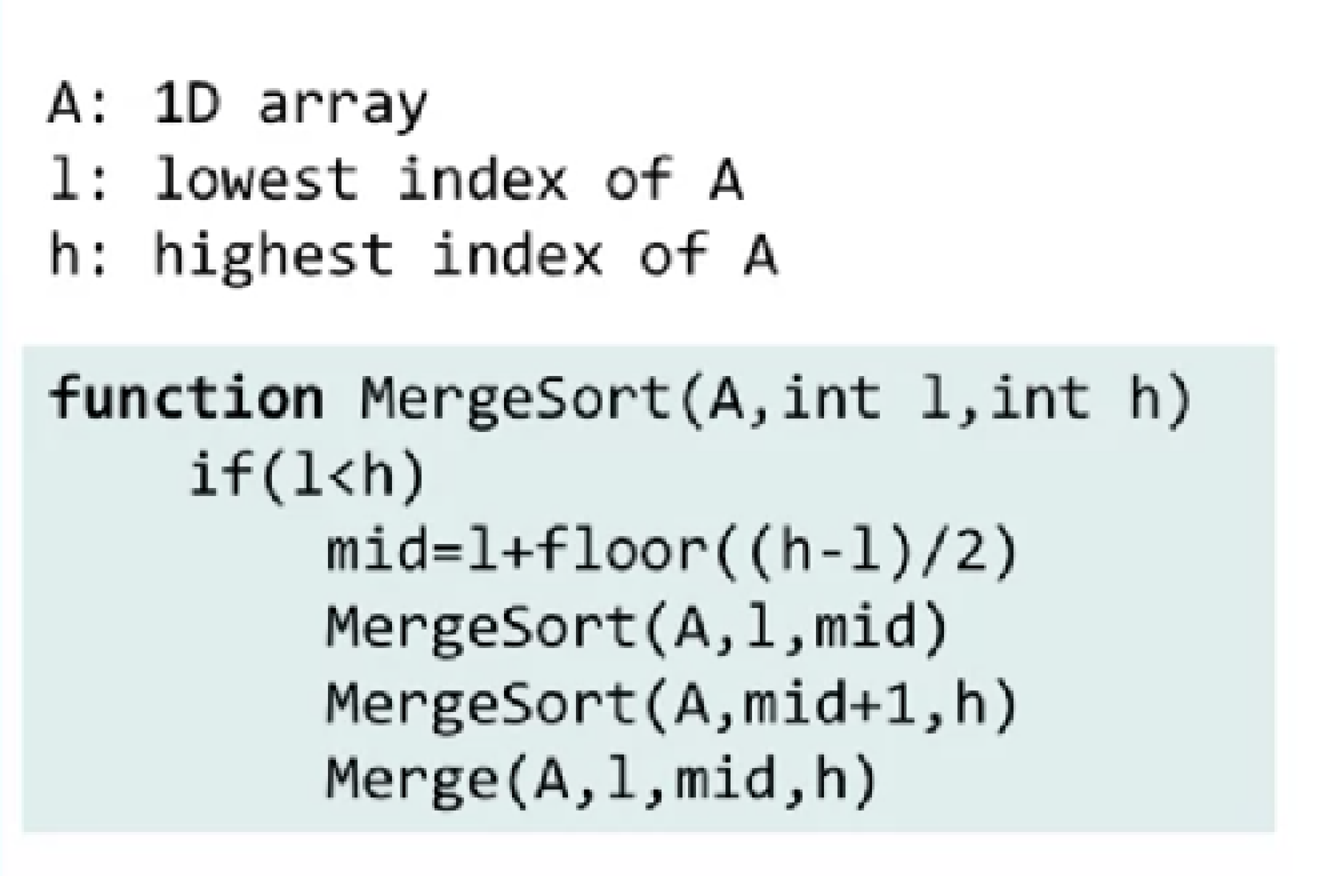
$$(2n-1)c + cn^2 + c$$

$$2cn + c + cn^2 + c$$

$$cn^2 + 2cn + c$$

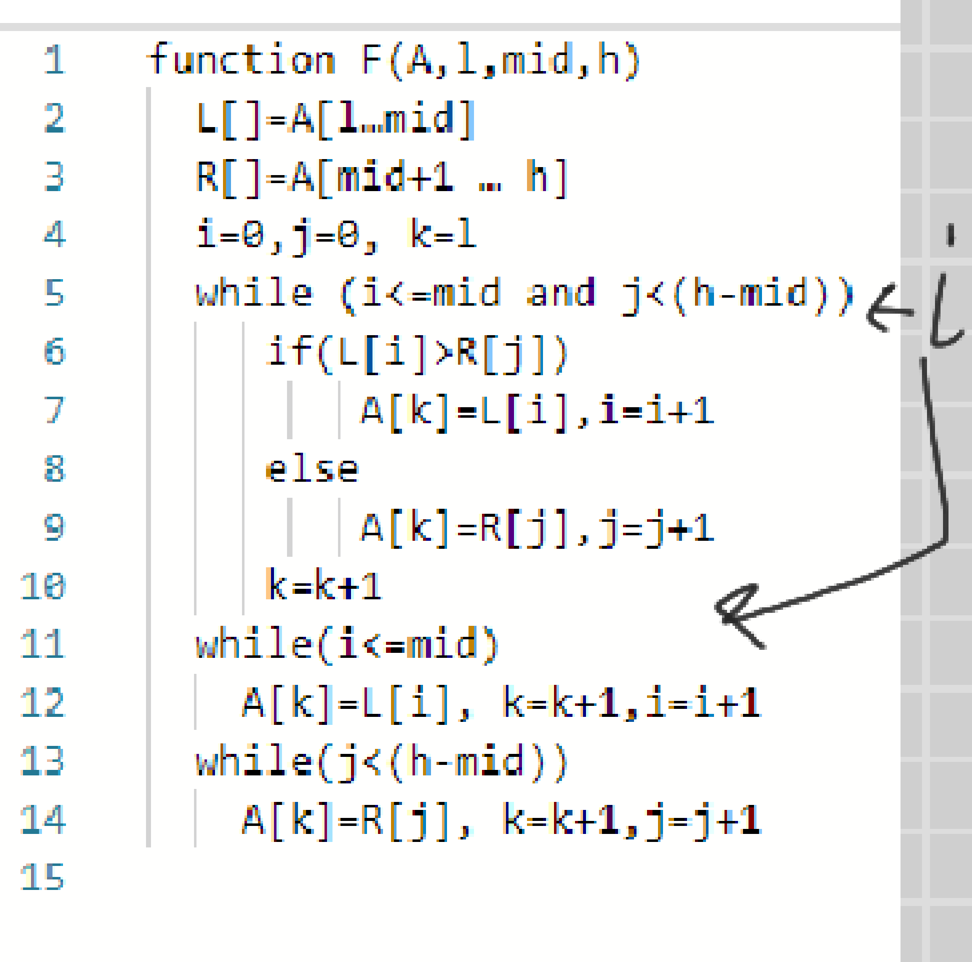
```
function partition(arr, low, high)          T(N)
    pivot <-- arr[high]                     C_0
    i <-- low                               C_1
    for low <= j < high                     C_2(n) + C_3
        if (arr[j] < pivot) then            C_3(n-1)
            swap(arr, i, j)                 C_4(n-1)
            i = i + 1                       C_5(n-1)
        end if
    end for
    swap(arr, i, high)                      C_6
    return i                                C_7
end function
```

Mergesort Analysis: Pesudocode



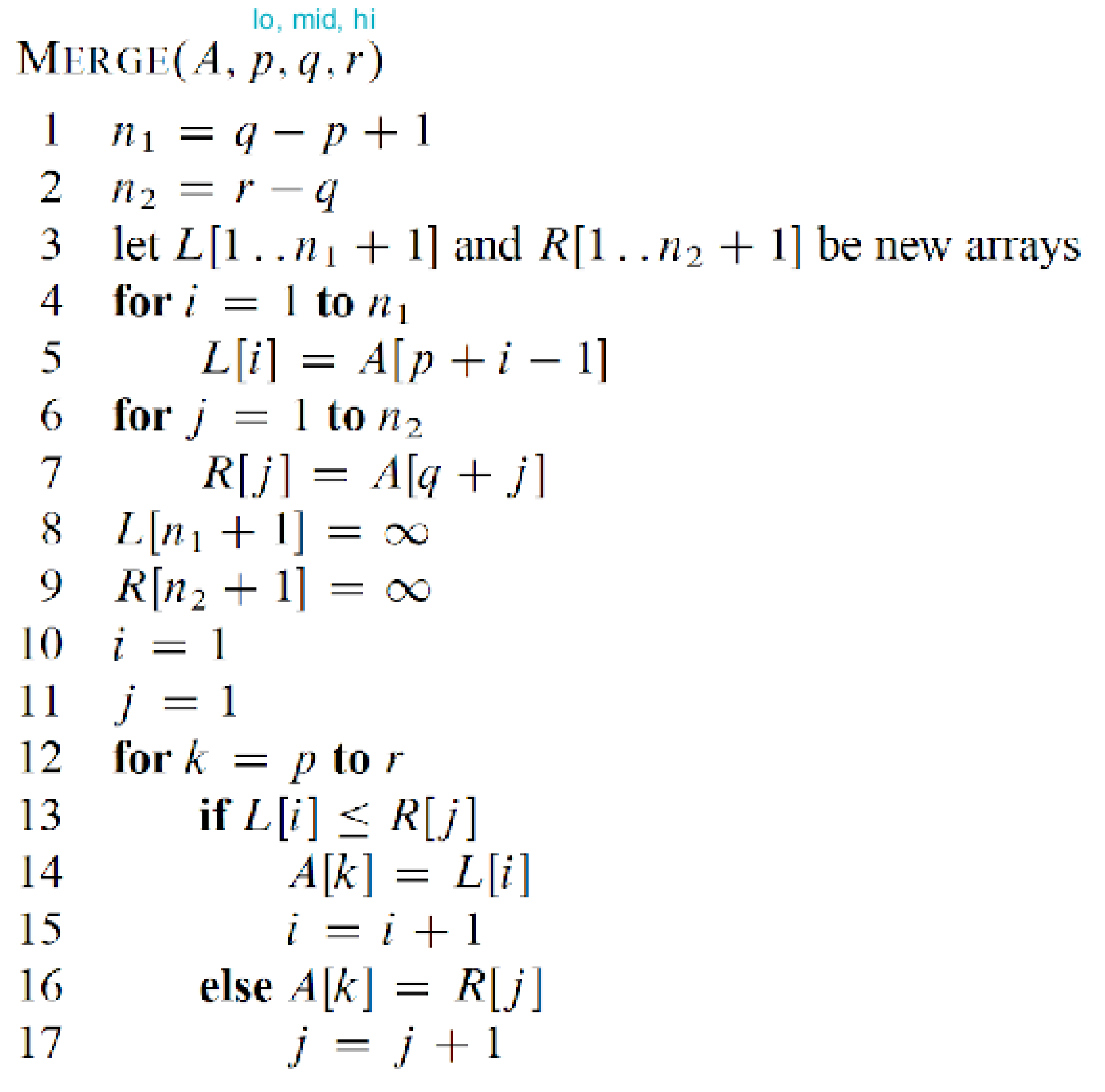
```
A: 1D array
l: lowest index of A
h: highest index of A
```

```
function MergeSort(A,int l,int h)
    if(l<h)
        mid=l+floor((h-l)/2)
        MergeSort(A,l,mid)
        MergeSort(A,mid+1,h)
        Merge(A,l,mid,h)
```



```
function F(A,l,mid,h)
  L[]=A[l…mid]
  R[]=A[mid+1 … h]
  i=0,j=0, k=l
  while (i<=mid and j<(h-mid))
    if(L[i]>R[j])
      A[k]=L[i],i=i+1
    else
      A[k]=R[j],j=j+1
    k=k+1
  while(i<=mid)
   A[k]=L[i], k=k+1,i=i+1
  while(j<(h-mid))
   A[k]=R[j], k=k+1,j=j+1
```

← $i \le$ mid - low



lo, mid, hi

```
MERGE(A, p, q, r)
  $n_1 = q - p + 1$
  $n_2 = r - q$
  let $L[1 \,..\, n_1 + 1]$ and $R[1 \,..\, n_2 + 1]$ be new arrays
  for $i = 1$ to $n_1$
      $L[i] = A[p + i - 1]$
  for $j = 1$ to $n_2$
      $R[j] = A[q + j]$
  $L[n_1 + 1] = \infty$
  $R[n_2 + 1] = \infty$
  $i = 1$
  $j = 1$
  for $k = p$ to $r$
      if $L[i] \le R[j]$
          $A[k] = L[i]$
          $i = i + 1$
      else $A[k] = R[j]$
          $j = j + 1$
```

```
function mergeSort(Vector)
  size ← LENGTH[Vector]
  if (size = 1)
    return Vector
  end if
  midpoint = $\lfloor \frac{size+1}{2} \rfloor$
  new Vector left(midpoint)
  new Vector right(size - midpoint)
  left ← vector[1 : midpoint]
  right ← vector[midpoint+1 : size]
  return merge(mergeSort(left), mergesort(right))
end function
```

newVector ← myVector[first:last]
Copies values from vector "myVector" between "first" and "last" indecies inclusive to the vector "newVector".

from ADS1

```
function merge(leftVec, rightVec)
  leftSize ← LENGTH[leftVec]
  rightSize ← LENGTH[rightVec]
  new Vector solution(leftSize + rightSize)
  i ← 1, j ← 1, k ← 1  k will be used to index the solution vector
  while (i ≤ leftSize ∧ j ≤ rightSize) do
    if (leftVec[i] < rightVec[j]) then
      solution[k] ← leftVec[i]
      i ← i+1
    else
      solution[k] ← rightVec[j]; j ← j+1
    end if
    k ← k+1
  end
  while (i ≤ leftSize) do
    solution[k] ← leftVec[i]
    i ← i+1
    k ← k+1
  end while
  while (j ≤ rightSize)
    solution[k] ← rightVec[j]
    j ← j+1
    k ← k+1
  end while
  return Solution
end function
```