Quick Sort
Quick Sort

- This was designed by Tony Hoare, a Turing-award winning, British Computer Scientist in 1960.

- Like merge sort, *quick sort* is based on the divide-and-conquer paradigm.
Main Idea

- **Divide step:**
  - Rearrange the elements in the list into two sublists so that all elements in the first sublist are less than (or equal to) all elements in the second sublist.

- **Conquer step:**
  - Call the quick sort function on each of the sublists.

- **Combine step:**
  - Not a whole lot left to do!

Compare this with merge sort, where all the work happens in the Combine step.
def generalQuickSort(L, left, right):
    # Base case: if left == right, then we have a list of length one
    # and there is nothing we need to do.

    # Recursive case
    if left < right:
        # Implementation of the divide step. This function call
        # returns an index p, left <= p <= right, such that every
        # element in L[left:p] <= L[p] and every element in L[p+1:right+1] >= L[p].
        p = partition(L, left, right)

        # Conquer step
        generalQuickSort(L, left, p-1)
        generalQuickSort(L, p+1, right)

    # There is no explicit combine step.
Quick Sort: Notes

- As in merge sort, no comparisons or movement of elements is happening in this recursive function.

- In merge sort, all of this was happening in the combine step, i.e., in the call to the function `merge`.

- In quick sort, all of this is happening in the divide step, i.e., in the call to the function `partition`. 
The partition function

- Function call:
  \[ p = \text{partition}(L, \text{left}, \text{right}) \]

- Rearranges elements and returns an index \( p \) such that
  - \( \text{left} \leq p \leq \text{right} \)
  - \( L[i] \leq L[p] \) for all \( i, \text{left} \leq i < p \).
  - \( L[i] \geq L[p] \) for all \( i, p < i \leq \text{right} \).
How to partition efficiently?

Typical situation:

Two possibilities:
- If \( L[\text{current}] \geq L[p] \)?
- If \( L[\text{current}] < L[p] \)?
How to partition efficiently?

Typical situation:

- If $L[current] \geq L[p]$?
  Do nothing and increment current.

- If $L[current] < L[p]$?
  - swap $L[p+1]$ and $L[current]$
  - Then swap $L[p+1]$ and $L[p]$
def partition(L, left, right):
    p = left
    for current in range(p+1, right+1):
        if L[current] < L[p]:
            swap(L, current, p+1)
            swap(L, p, p+1)
    p = p+1

    return p