Quick Sort
def generalQuickSort(L, first, last):
    # Base case: if first == last, then there is only one element in the
    # slice that needs sorting. So there is nothing to do.

    # Recursive case: if there are 2 or more elements in the slice L[first:last+1]
    if first < last:
        # Divide step: partition returns an index p such that
        # first <= p <= last and everthing in L[first:p] is <= L[p]
        # and everything in L[p+1:last+1] is >= L[p]
        p = partition(L, first, last)

        # Conquer step
        generalQuickSort(L, first, p-1)
        generalQuickSort(L, p+1, last)

    # Combine step: there is nothing left to do!
The partition function

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

- Rearranges elements and returns an index \( p \) such that:
  - \( \text{left} \leq p \leq \text{right} \)
  - \( L[i] < L[p] \) for all \( i, \text{left} \leq i < p \)
    (Elements to the left of \( L[p] \) are strictly smaller)
  - \( L[j] \geq L[p] \) for all \( j, p < j \leq \text{right} \)
    (Elements to the right of \( L[p] \) are at least as large.)
Example: partition function

- L is \([14, 1, 3 19, 4, 8, 22, 22, 14, 2, 34, 40]\)

- Suppose we call
  \[p = \text{partition}(L, 0, 11)\]

- The function could rearrange elements in L as:
  \([1, 2, 3, 8, 4, 14, 22, 22, 14, 19, 40, 34]\)
  and return index 5.
The partition algorithm

Main idea:

- Use the first element $L[left]$ as a special element called the *pivot*.
- Compare all other elements against the pivot:
  - those that are smaller should end up to the left of the pivot.
  - those that are at least as large should end up to the right of the pivot.

Goal is to ensure that the movement of elements after comparison with pivot is not too costly.
How to move elements efficiently in partition?

Typical situation:

```
[11, 12, 9, 14, 20, 17, x, unprocessed]
```

Two possibilities:

- L[current] >= L[p]: element stays in its location
- L[current] < L[p]: element moves to the left of p
  - swap L[current] and L[p+1]
  - swap L[p] and L[p+1]
Example: partition

- Suppose the current element is 2
  \[p\]
  \[11, 12, 9, 14, 20, 17, 2, 15, \ldots\]

- After \texttt{swap(L, p+1, current)} we get
  \[11, 12, 9, 14, 2, 17, 20, 15, \ldots\]

- After \texttt{swap(L, p, p+1)} we get
  \[11, 12, 9, 2, 14, 17, 20, 15, \ldots\]
def partition(L, first, last):
    # We pick the element L[first] as the "pivot" around which we partition the list
    p = first

    # We process the rest of the elements, one-by-one, in left-to-right order
    for current in range(p+1, last+1):
        # If L[current] is smaller than the pivot, it needs to move into the first block,
        # to the left of the pivot.
        if L[current] < L[p]:
            swap(L, current, p+1)
            swap(L, p, p+1)
        p = p + 1

    return p
The partition function in action

- Suppose \( L = [8, 3, 4, 9, 3, 2, 11, 10, 3, 1] \).
- We call

  \[
  \text{partition}(L, 0, 9)
  \]

<table>
<thead>
<tr>
<th></th>
<th>8</th>
<th>[3, 4, 9, 3, 2, 11, 10, 3, 1]</th>
</tr>
</thead>
<tbody>
<tr>
<td>[]</td>
<td>8</td>
<td>[3, 4, 9, 3, 2, 11, 10, 3, 1]</td>
</tr>
<tr>
<td>[3]</td>
<td>8</td>
<td>[4, 9, 3, 2, 11, 10, 3, 1]</td>
</tr>
<tr>
<td>[3, 4]</td>
<td>8</td>
<td>[9, 3, 2, 11, 10, 3, 1]</td>
</tr>
<tr>
<td>[3, 4]</td>
<td>8</td>
<td>[9, 3, 2, 11, 10, 3, 1]</td>
</tr>
<tr>
<td>[3, 4, 3]</td>
<td>8</td>
<td>[9, 2, 11, 10, 3, 1]</td>
</tr>
<tr>
<td>[3, 4, 3, 2]</td>
<td>8</td>
<td>[9, 11, 10, 3, 1]</td>
</tr>
<tr>
<td>[3, 4, 3, 2]</td>
<td>8</td>
<td>[9, 11, 10, 3, 1]</td>
</tr>
<tr>
<td>[3, 4, 3, 2]</td>
<td>8</td>
<td>[9, 11, 10, 3, 1]</td>
</tr>
<tr>
<td>[3, 4, 3, 2, 3]</td>
<td>8</td>
<td>[11, 10, 9, 1]</td>
</tr>
</tbody>
</table>