Recursive Divide-and-Conquer
Quick Sort
The *Divide-and-Conquer* Paradigm

- This is an important algorithmic technique to efficiently solving computational problems.

- It is commonly used for
  - Efficient sorting
  - Multiplying large numbers
  - Multiplying matrices
  - Finding a closest pair of points in Euclidean space

- It is usually implemented using recursion.
In `binarySearch(L, k)`, we make one comparison: \( k \) compared to \( L[mid] \).

Based on the outcome of this comparison, we either stop, search the left half, or search the right half.

Thus the problem of searching for \( k \) in \( L \) is reduced to search for \( k \) in \( L[:mid] \) or \( L[mid+1:] \).
Divide-and-Conquer Paradigm

- **Divide step:** Partition the problem into sub-problems.

- **Conquer step:** Solve each sub-problem separately.

- **Combine step:** Combine the solutions of the sub-problems into a solution of the original problem.
Algorithms such as selection sort, insertion sort, bubble sort, etc. are all extremely slow for large lists.

This is because they take about $N^2$ time on a list of size $N$.

Algorithms that are based on “divide-and-conquer” such as merge sort or quick sort are much faster.

These algorithms run in about $N \log N$ time on a list of size $N$. 
Quick Sort: Main Idea

- **Divide Step:**
  - Rearrange the elements in the lists into two sublists so that all elements in the first sublist are smaller than all elements in the second sublist.

- **Conquer Step:**
  - Sort each of the halves separately.

- **Combine Step:**
  - Combine the two sorted halves into a sorted whole – actually there is not a whole lot to do in this step.
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!
Quick Sort: Notes

- No comparisons or swapping of elements is happening in this function. (So, where is all this happening?)

- All the work of comparing elements and moving elements is happening in the Divide step, namely the function call to `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 \( i, \text{left} \leq i < p \)
  - \( L[j] \geq L[p] \) for all \( j, p < j \leq \text{right} \)