Merge-Sort

Let us try divide and conquer:

1. Split the problem into smaller instances.
2. Recursively solve the subproblems.
3. Combine the solutions to solve the original problem.

```
def merge_sort(a):
  if len(a) <= 1:
    return a
  mid = len(a) // 2
  return merge(merge_sort(a[:mid]),
               merge_sort(a[mid:]))
```


Merging

We are given two sorted lists a and b, and we wish to combine them into one sorted list.

```
def merge(a, b):
  i = 0;  j = 0
  res = []
  while i < len(a) and j < len(b):
    va = a[i]
    vb = b[j]
    if va <= vb:
      res.append(va)
      i += 1
    else:
      res.append(vb)
      j += 1
  res.extend(a[i:])
  res.extend(b[j:])
  return res
```


Merge-Sort — Time Analysis

Merging takes $O(n)$ time.

Let $T(n)$ be the time taken by Merge-Sort for n elements. Then $T(1) = O(1)$ and

$$T(n) = 2T(n/2) + O(n)$$

The solution is $O(n \log n)$.

Quick-sort

Divide and conquer:

1. Split the problem into smaller instances.
2. Recursively solve the subproblems.
3. Combine the solutions to solve the original problem.

In Merge-Sort, the divide step is trivial, and the combine step is where all the work is done.

In Quick-Sort, the combine step is trivial, and all the work is done in the divide step:

1. If L has less than two elements, return. Otherwise, select a pivot p from L. Split L into three lists S, E, and G, where
   - S stores the elements of L smaller than x,
   - E stores the elements of L equal to x, and
   - G stores the elements of L greater than x.
2. Recursively sort S and G.
3. Form result by concatenating S, E, and G in this order.

Quick-Sort

```
def quick_sort(a):
  if len(a) <= 1:
    return a
  pivot = a[len(a) // 2]
  small = []
  equal = []
  large = []
  for x in a:
    if x < pivot:
      small.append(x)
    elif x == pivot:
      equal.append(x)
    else:
      large.append(x)
  return (quick_sort(small) + equal +
          quick_sort(large))
```


Quick-Sort Analysis

The running time depends strongly on the choice of the pivot.

In the worst case, it is $O(n^2)$.

In the best case, it is $O(n \log n)$.

If the pivot is selected randomly, the expected running time is $O(n \log n)$.

Quick-Sort can be implemented in-place (using one array only).