Quick Sort O(n^2)~O(nlogn)

- Based on divide & conquer
- “MergeSort in reverse”

Ideas:

1. Select a pivot element

   For us, choose the last element.
2. Partition: everything smaller than pivot goes left, larger goes right
3. Recursively sort both parts

```
QUICKSORT(A[1 ... n])
    k = PARTITION(A)        // location of the pivot
    QUICKSORT(A[1 ... k-1])
    QUICKSORT(A[k+1 ... n])
```

Obviously correct assuming PARTITION is correct.

```
PARTITION(A[1 ... n])
    pivot = A[n]
    i = 1
    for j = 1 to n-1:
        if A[i] <= pivot:
            swap A[i] with A[j]
            i = i + 1
    swap A{i} with A[n]
    return i
```

Loop invariant:

1. For all k ∈ {1 ... i-1}, A[k] ≤ pivot
2. For all k ∈ {i ... j-1}, A[k] ≥ pivot

Proof by induction

When the algorithm terminates, loop invariant guarantees that A[1] ... A[i-1] ≤ pivot, A[i+1] ... A[n] ≥ pivot, A[i] = pivot.

Runtime of PARTITION: O(n)

Runtime of QuickSort:

Runtime: $1 + ... + n = O(n^2)$

Average case: if the array we receive is randomly shuffled, we can expect the last element (pivot) not to be among the top or bottom 10 percentiles. This happens with probability 80%.

Moreover, assume that the pivot is exactly at the 10th percentile. Then,

$T(n) = T(\frac{9}{10}n) + T(\frac{1}{10}n) + n$

Depth of recursion tree: $log_{\frac{10}{9}} n$

Runtime in each layer: $\leq n$

Total runtime: $\leq nlog_{\frac{10}{9}}n = \Theta(nlogn)$ (with high probability)

Remark: usually, pivot is chosen differently

- One heuristic, is to take the median of {first, middle, last}. Works well in practice.
- Choose pivot at random. Then, expected runtime is $\Theta(nlogn)$ in the worst case. Analysis is similar to above.
- Choose the median of the whole array as the pivot