QuickSort

- Yet another sorting algorithm!
- Usually faster than other algorithms on average, although worst-case is $O(n^2)$
- Divide-and-conquer:
  - Divide: Choose an element of the array for pivot. Divide the elements into two groups: those smaller than the pivot, and those larger or equal to the pivot.
  - Conquer: Recursively sort each group.
  - Combine: Concatenate the two sorted groups.

Example

$A = [6\ 3\ 5\ 9\ 2\ 5\ 7\ 8\ 4\ 5]$  

In-place quickSort

Algorithm quickSort(A, start, stop)  
Input: An array A to sort, indices start and stop  
Output: A[start...stop] is sorted  
if (start < stop) then  
pivot ← partition(A, start, stop)  
quickSort(A, start, pivot-1)  
quickSort(A, pivot+1, stop)  

Example of execution of partition

$A = [6\ 3\ 7\ 3\ 2\ 5\ 7\ 5]$  
pivot = 5  
$A = [6\ 3\ 7\ 3\ 2\ 5\ 7\ 5]$  
swap 6, 2  
$A = [2\ 3\ 7\ 3\ 6\ 5\ 7\ 5]$  
$A = [2\ 3\ 7\ 3\ 6\ 5\ 7\ 5]$  
swap 7,3  
$A = [2\ 3\ 3\ 7\ 6\ 5\ 7\ 5]$  
$A = [2\ 3\ 3\ 7\ 6\ 5\ 7\ 5]$  
swap 7, pivot  
$A = [2\ 3\ 3\ 5\ 6\ 5\ 7\ 7]$  

QuickSort running time

- Worse case:
  - Already sorted array (either increasing or decreasing)  
  - $T(n) = T(n-1) + cn + d$  
  - $T(n)$ is $O(n^2)$  
- Average case: If the array is in random order, the pivot splits the array in roughly equal parts, so the average running time is $O(n \log n)$
- Advantage over mergeSort:
  - constant hidden in $O(n \log n)$ are smaller for quickSort. Thus it is faster by a constant factor
  - QuickSort is easy to do "in-place"
Algorithm partition(A, start, stop)

Input: An array A, indices start and stop.

pivot ← A[stop]
left ← start
right ← stop - 1

while left ≤ right do
    while left ≤ right and A[left] < pivot do left ← left + 1
    while (left ≤ right) and A[right] ≥ pivot do right ← right - 1
    if (left < right) then exchange A[left] ↔ A[right]

return left

In-place algorithms

• An algorithm is in-place if it uses only a constant amount of memory in addition of that used to store the input.

• Importance of in-place sorting algorithms:
  – If the data set to sort barely fits into memory, we don’t want an algorithm that uses twice that amount to sort the numbers.
  – SelectionSort and InsertionSort are in-place: all we are doing is moving elements around the array.
  – MergeSort is not in-place, because of the merge procedure, which requires a temporary array.
  – QuickSort can easily be made in-place...