**Quicksort**

**Quick sort** is an efficient [sorting](http://www.algorithmist.com/index.php/Sorting) algorithm invented by C.A.R. Hoare. Its average-case running time is *O*(*n*log*n*). Unfortunately, Quicksort's performance degrades as the input list becomes more ordered. The worst-case input, a sorted list, causes it to run in *O*(*n*2) time. An improvement upon this algorithm that detects this prevalent corner case and guarantees *O*(*n*log*n*) time is [Introsort](http://www.algorithmist.com/index.php/Introsort).

## Algorithm

1. Pick a "pivot point". Picking a good pivot point can greatly affect the running time.
2. Break the list into two lists: those elements less than the pivot element, and those elements greater than the pivot element.
3. Recursively sort each of the smaller lists.
4. Make one big list: the 'smallers' list, the pivot points, and the 'biggers' list.

Picking a random pivot point will not eliminate the *O*(*n*2) worst-case time, but it *will* usually transform the worst case into a less frequently occuring permutation. In practice, the sorted list usually comes up more often than any other permutation, so this improvement is often used.

## Pseudocode

Quicksort(A as array, low as int, high as int)

if (low < high)

pivot\_location = Partition(A,low,high)

Quicksort(A,low, pivot\_location - 1)

Quicksort(A, pivot\_location + 1, high)

Partition(A as array, low as int, high as int)

pivot = A[low]

leftwall = low

for i = low + 1 to high

if (A[i] < pivot) then

leftwall = leftwall + 1

swap(A[i], A[leftwall])

swap(A[low],A[leftwall])

return (leftwall)

## Alternative definition in Common Lisp

(defun q-sort (l &optional (f #'<))

(if (null (cdr l)) l

(append (q-sort (remove-if-not #'(lambda (x) (funcall f x (car l))) (cdr l)) f)

(list (car l))

(q-sort (remove-if #'(lambda (x) (funcall f x (car l))) (cdr l)) f))))