Insertion sort

```
(defun insertion-sort (sequence)
  (let ((end (length sequence)))
    (labels ((insert (x index)
               (if (minusp index)
                   (setf (elt sequence (1+ index)) x)
                   (let ((y (elt sequence index)))
                     (if (< x y)
                         (progn
                           (setf (elt sequence (1+ index)) y)
                           (insert x (1- index)))
                         (setf (elt sequence (1+ index)) x)))))
             (repeat-insertion (start)
               (if (= start end)
                   (values)
                   (progn
                     (insert (elt sequence start) (1- start))
                     (repeat-insertion (1+ start))))))
      (unless (< end 2)
        (repeat-insertion 1))
      sequence)))
```

output

```
(insertion-sort (8 7 4 3 2 0 9 1 5 6) '<)
(0 1 2 3 4 5 6 7 8 9)
```

Merge sort

```
(defun merge-sort (result-type sequence predicate)
  (let ((split (floor (length sequence) 2)))
    (if (zerop split)
      (copy-seq sequence)
      (merge result-type (merge-sort result-type (subseq sequence 0 split) predicate)
                  (merge-sort result-type (subseq sequence split)   predicate)
                  predicate))))
```

merge is a standard Common Lisp function.

output

```
> (merge-sort 'list (list 1 3 5 7 9 8 6 4 2) #'<)
(1 2 3 4 5 6 7 8 9)
```

Selection sort

```
(defun selection-sort-list (list predicate)
 (flet ((min-first (list)
         (do ((before-min nil)
              (min (first list))
              (prev list (rest prev))
              (curr (rest list) (rest curr)))
             ((endp curr)
              (if (null before-min) list
                (let ((min (cdr before-min)))
                  (rplacd before-min (cdr min))
                  (rplacd min list)
                  min)))
           (when (funcall predicate (first curr) min)
```

```
            (setf before-min prev
                  min (first curr))))))
    (let ((result (min-first list)))
      (do ((head result (rest head)))
          ((endp (rest head)) result)
        (rplacd head (min-first (rest head)))))))
```

output

```
> (selection-sort (list 8 7 4 3 2 0 9 1 5 6) '<)
(0 1 2 3 4 5 6 7 8 9)
```

Quick sort

```
(defun quicksort (sequence)
  (labels ((swap (a b) (rotatef (elt sequence a) (elt sequence b)))
           (sub-sort (left right)
             (when (< left right)
               (let ((pivot (elt sequence right))
                     (index left))
                 (loop for i from left below right
                       when (<= (elt sequence i) pivot)
                         do (swap i (prog1 index (incf index))))
                 (swap right index)
                 (sub-sort left (1- index))
                 (sub-sort (1+ index) right)))))
    (sub-sort 0 (1- (length sequence)))
    sequence))
```

output

(quicksort (8 7 4 3 2 0 9 1 5 6) '<)

(0 1 2 3 4 5 6 7 8 9)