```
File created:   16-May-90 14:41:22  {DSK}<usr>local>lde>lispcore>sources>CMLSORT.;2

  changes to:   (VARS CMLSORTCOMS)

previous date:  25-Nov-86 14:28:27  {DSK}<usr>local>lde>lispcore>sources>CMLSORT.;1

  Read Table:   XCL

     Package:   INTERLISP

      Format:   XCCS
```

```
(RPAQQ CMLSORTCOMS
       (;; CLtL Section 14.5 Merging and Sorting

        (DECLARE\: DONTCOPY DOEVAL@COMPILE (FILES CMLSEQCOMMON))

        ;; vector sort functions

        (FUNCTIONS %VECTOR-QUICK-SORT-STEP %VECTOR-INSERTION-SORT-STEP %VECTOR-QSFENCE %SORT-VECTOR
               %STABLE-SORT-VECTOR %SIMPLE-VECTOR-QUICK-SORT %VECTOR-QUICK-SORT %SIMPLE-VECTOR-INSERTION-SORT
               %VECTOR-INSERTION-SORT)

        ;; list sort functions

        (FUNCTIONS %SORT-SUBLIST %MERGE-SUBLISTS-MACRO %SIMPLE-MERGE-SUBLISTS %MERGE-SUBLISTS)

        ;; vector merge functions

        (FUNCTIONS %MERGE-MACRO %SIMPLE-MERGE %MERGE %SIMPLE-MERGE-VECTORS %MERGE-VECTORS)

        ;; list merge functions

        (FUNCTIONS %SIMPLE-MERGE-LISTS %MERGE-LISTS)

        ;; user entry points

        (FUNCTIONS CL:SORT CL:STABLE-SORT CL:MERGE)
        (PROP FILETYPE CMLSORT)
        (DECLARE\: DONTCOPY DOEVAL@COMPILE DONTEVAL@LOAD (LOCALVARS . T))))



;; CLtL Section 14.5 Merging and Sorting

(DECLARE\: DONTCOPY DOEVAL@COMPILE

(FILESLOAD CMLSEQCOMMON)
)



;; vector sort functions


(DEFMACRO %VECTOR-QUICK-SORT-STEP (VECTOR PRED LOWER UPPER ACCESSOR-FORM &REST FORMS)
   `(LET ((I (CL:1+ ,LOWER))
          (J (CL:1- ,UPPER))
          (%X-LOWER ,(CL:SUBST LOWER 'INDEX ACCESSOR-FORM)))
         (CL:LOOP (CL:LOOP (CL:INCF I)
                         (CL:IF (NOT (CL:FUNCALL ,PRED ,(CL:SUBST 'I 'INDEX ACCESSOR-FORM)
                                                 %X-LOWER))
                                (RETURN NIL)))
                (CL:LOOP (CL:DECF J)
                       (CL:IF (NOT (CL:FUNCALL ,PRED %X-LOWER ,(CL:SUBST 'J 'INDEX ACCESSOR-FORM)))
                              (RETURN NIL)))
                (COND
                   ((> J I)
                    (CL:ROTATEF (CL:AREF ,VECTOR I)
                           (CL:AREF ,VECTOR J)))
                   (T (RETURN NIL))))
         (CL:ROTATEF (CL:AREF ,VECTOR ,LOWER)
                (CL:AREF ,VECTOR J))
         ,@FORMS))

(DEFMACRO %VECTOR-INSERTION-SORT-STEP (VECTOR COMPAREFN LOWER UPPER ACCESSOR-FORM)

   ;; Sort elements (LOWER .. UPPER) of the 1-dimensional CMLArray ARRAY using the ordering given by COMPAREFN.  ARRAY is sorted in place,
   ;; i.e.  destructively.  NO argument checking! Returns ARRAY.  INTENDED FOR FEWER THAN 20 ELEMENTS, USE QuickSort FOR LARGER
   ;; PROBLEMS!

   `(CL:DO ((%I (CL:1+ ,LOWER)
                (CL:1+ %I))
            %ITH-ELEMENT %ITH-COMPARATOR TEMP)
           ((EQL %I ,UPPER)
            ,VECTOR)
        (SETQ %ITH-ELEMENT (CL:AREF ,VECTOR %I))
        (SETQ %ITH-COMPARATOR ,(CL:SUBST '%ITH-ELEMENT 'INDEXED-ELEMENT ACCESSOR-FORM))
        (CL:DO ((%J %I (CL:1- %J)))
               ((OR (EQL %J ,LOWER)
                    (NOT (CL:FUNCALL ,COMPAREFN %ITH-COMPARATOR (PROGN (SETQ TEMP (CL:AREF ,VECTOR (CL:1- %J)))
                                                                        ,(CL:SUBST 'TEMP 'INDEXED-ELEMENT
                                                                                ACCESSOR-FORM)))))
                (CL:SETF (CL:AREF ,VECTOR %J)
                     %ITH-ELEMENT))
            (CL:SETF (CL:AREF ,VECTOR %J)
                   TEMP))))
```

(DEFMACRO **%VECTOR-QSFENCE** (VECTOR PRED LOWER UPPER ACCESSOR-FORM)

;; Identify the partitioning element as the median-of-three estimate of the median.  Reference: Sedgewick, R.  'Implementing Quicksort Programs'
;; CACM vol.  21 no.  10 pp.  847--857.

```
   `(LET ((%UP-IDX (CL:1- ,UPPER))
          (%MD-IDX (CL:ASH (+ ,LOWER ,UPPER)
                         -1))
          (%LW-IDX+1 (CL:1+ ,LOWER)))
         (CL:ROTATEF (CL:AREF ,VECTOR %MD-IDX)
                (CL:AREF ,VECTOR %LW-IDX+1))
         (CL:IF (CL:FUNCALL ,PRED ,(CL:SUBST '%UP-IDX 'INDEX ACCESSOR-FORM)
                     ,(CL:SUBST '%LW-IDX+1 'INDEX ACCESSOR-FORM))
             (CL:ROTATEF (CL:AREF ,VECTOR %LW-IDX+1)
                    (CL:AREF ,VECTOR %UP-IDX)))
         (CL:IF (CL:FUNCALL ,PRED ,(CL:SUBST '%UP-IDX 'INDEX ACCESSOR-FORM)
                     ,(CL:SUBST 'LOWER 'INDEX ACCESSOR-FORM))
             (CL:ROTATEF (CL:AREF ,VECTOR ,LOWER)
                    (CL:AREF ,VECTOR %UP-IDX)))
         (CL:IF (CL:FUNCALL ,PRED ,(CL:SUBST 'LOWER 'INDEX ACCESSOR-FORM)
                     ,(CL:SUBST '%LW-IDX+1 'INDEX ACCESSOR-FORM))
             (CL:ROTATEF (CL:AREF ,VECTOR %LW-IDX+1)
                    (CL:AREF ,VECTOR ,LOWER)))))
```

(CL:DEFUN **%SORT-VECTOR** (VECTOR PRED KEY)

;; Sort the 1-dimensional CMLArray ARRAY using the ordering given by COMPAREFN.  ARRAY is sorted in place, i.e.  destructively.  Returns
;; ARRAY.  Reference: Sedgewick, R.  'Implementing Quicksort Programs' CACM vol.  21 no.  10 pp.  847--857.

```
   (LET ((LOWER 0)
         (UPPER (VECTOR-LENGTH VECTOR)))
        (COND
           (KEY (%VECTOR-QUICK-SORT VECTOR PRED KEY LOWER UPPER)
                (%VECTOR-INSERTION-SORT VECTOR PRED KEY LOWER UPPER))
           (T (%SIMPLE-VECTOR-QUICK-SORT VECTOR PRED LOWER UPPER)
              (%SIMPLE-VECTOR-INSERTION-SORT VECTOR PRED LOWER UPPER)))
        VECTOR))
```

(CL:DEFUN **%STABLE-SORT-VECTOR** (VECTOR PRED KEY)

;; Uses Insertion sort which, although slower than quick sort, is stable

```
   (LET ((LENGTH (VECTOR-LENGTH VECTOR)))
        (CL:IF KEY
            (%VECTOR-INSERTION-SORT VECTOR PRED KEY 0 LENGTH)
            (%SIMPLE-VECTOR-INSERTION-SORT VECTOR PRED 0 LENGTH)))
   VECTOR)
```

(CL:DEFUN **%SIMPLE-VECTOR-QUICK-SORT** (VECTOR PRED LOWER UPPER)

```
   (CL:WHEN (> (- UPPER LOWER)
               10)
       (%VECTOR-QSFENCE VECTOR PRED LOWER UPPER (CL:AREF VECTOR INDEX))
```

;; Perform the partitioning. At this point array[(1+ LOWER)] <= array[LOWER] <= array[(1- UPPER)]

```
       (%VECTOR-QUICK-SORT-STEP VECTOR PRED LOWER UPPER (CL:AREF VECTOR INDEX)
              (COND
                 ((> (- J LOWER)
                     (- UPPER I))
                  (%SIMPLE-VECTOR-QUICK-SORT VECTOR PRED LOWER J)
                  (%SIMPLE-VECTOR-QUICK-SORT VECTOR PRED I UPPER))
                 (T (%SIMPLE-VECTOR-QUICK-SORT VECTOR PRED I UPPER)
                    (%SIMPLE-VECTOR-QUICK-SORT VECTOR PRED LOWER J))))))
```

(CL:DEFUN **%VECTOR-QUICK-SORT** (VECTOR PRED KEY LOWER UPPER)

```
   (CL:WHEN (> (- UPPER LOWER)
               10)
       (%VECTOR-QSFENCE VECTOR PRED LOWER UPPER (CL:FUNCALL KEY (CL:AREF VECTOR INDEX)))
```

;; Perform the partitioning. At this point array[(1+ LOWER)] <= array[LOWER] <= array[(1- UPPER)]

```
       (%VECTOR-QUICK-SORT-STEP VECTOR PRED LOWER UPPER (CL:FUNCALL KEY (CL:AREF VECTOR INDEX))
              (COND
                 ((> (- J LOWER)
                     (- UPPER I))
                  (%VECTOR-QUICK-SORT VECTOR PRED KEY LOWER J)
                  (%VECTOR-QUICK-SORT VECTOR PRED KEY I UPPER))
                 (T (%VECTOR-QUICK-SORT VECTOR PRED KEY I UPPER)
                    (%VECTOR-QUICK-SORT VECTOR PRED KEY LOWER J))))))
```

(CL:DEFUN **%SIMPLE-VECTOR-INSERTION-SORT** (VECTOR PRED LOWER UPPER)

```
   (%VECTOR-INSERTION-SORT-STEP VECTOR PRED LOWER UPPER INDEXED-ELEMENT))
```

```
(CL:DEFUN %VECTOR-INSERTION-SORT (VECTOR PRED KEY LOWER UPPER)
   (%VECTOR-INSERTION-SORT-STEP VECTOR PRED LOWER UPPER (CL:FUNCALL KEY INDEXED-ELEMENT)))
```

;; list sort functions

```
(CL:DEFUN %SORT-SUBLIST (START END PRED KEY)
   ;; Based on the old Interlisp list sorter due to Deutch and Masinter
   (CL:IF (OR (EQ START END)
              (EQ (CDR START)
                  END))                                          ; At bottom of recursion
       START
       (LET ((MID                                                ; Split sublist by setting MID to  its midpoint.
                  (CL:DO ((FAST-POINTER START)
                          (SLOW-POINTER START))
                         ((OR (EQ (SETQ FAST-POINTER (CDR FAST-POINTER))
                                  END)
                              (EQ (SETQ FAST-POINTER (CDR FAST-POINTER))
                                  END))
                          (CDR SLOW-POINTER))
                      (SETQ SLOW-POINTER (CDR SLOW-POINTER)))))  ; sort the two halves separately
            (%SORT-SUBLIST START MID PRED KEY)
            (%SORT-SUBLIST MID END PRED KEY)                     ; Now merge back
            (CL:IF KEY
                (%MERGE-SUBLISTS START MID END PRED KEY)
                (%SIMPLE-MERGE-SUBLISTS START MID END PRED)))))

(DEFMACRO %MERGE-SUBLISTS-MACRO (START1 START2 END PRED KEY)
   `(LET ((HANDLE ,START1)
          (END1                                                  ; always (eq (cdr end1) start2)
                (CL:DO ((L ,START1 (CDR L)))
                       ((EQ (CDR L)
                            ,START2)
                        L)))
          ,@(CL:IF KEY
                '(KEY-1 KEY-2)))
         (CL:LOOP (CL:IF (OR (EQ ,START1 ,START2)
                             (EQ ,START2 ,END))
                         (RETURN HANDLE))
                ,@(CL:IF KEY
                      `((CL:IF (NULL KEY-1)
                              (SETQ KEY-1 (CL:FUNCALL ,KEY (CAR ,START1))))
                        (CL:IF (NULL KEY-2)
                              (SETQ KEY-2 (CL:FUNCALL ,KEY (CAR ,START2))))))
                (COND
                   ((NOT ,(CL:IF KEY
                              `(CL:FUNCALL ,PRED KEY-2 KEY-1)
                              `(CL:FUNCALL ,PRED (CAR ,START2)
                                      (CAR ,START1))))
                    ,@(CL:IF KEY
                          `((SETQ KEY-1 NIL)))
                    (SETQ ,START1 (CDR ,START1)))
                   (T ,@(CL:IF KEY
                            `((SETQ KEY-2 NIL)))
                      (UNINTERRUPTABLY

                          ;; Move first element of second sublist to before first element of first  sublist .  This must be done by exchanging the
                          ;; CARs and then patching up the CDRs, so that handle always points to the start of the list.

                          (COND
                             ((EQ ,START1 END1)                  ; Special case.
                              (RPLACA ,START1 (PROG1 (CAR ,START2)
                                                  (RPLACA ,START2 (CAR ,START1))))
                              (SETQ ,START2 (CDR (SETQ ,START1 (SETQ END1 ,START2)))))
                             (T (RPLACD END1 (PROG1 (CDR ,START2)
                                                 (RPLACA ,START1 (PROG1 (CAR ,START2)
                                                                     (RPLACD (RPLACA ,START2 (CAR ,START1))
                                                                            (CDR ,START1))
                                                                     (RPLACD ,START1 ,START2)))))
                                (SETQ ,START1 ,START2)
                                (SETQ ,START2 (CDR END1))))))))))

(CL:DEFUN %SIMPLE-MERGE-SUBLISTS (START1 START2 END PRED)
   (%MERGE-SUBLISTS-MACRO START1 START2 END PRED))

(CL:DEFUN %MERGE-SUBLISTS (START1 START2 END PRED KEY)
   (%MERGE-SUBLISTS-MACRO START1 START2 END PRED KEY))
```

;; vector merge functions

```
(DEFMACRO %MERGE-MACRO (RESULT SEQUENCE1 SEQUENCE2 PRED ACCESSOR-FORM KEY)
   `(LET ((LENGTH1 (CL:LENGTH ,SEQUENCE1))
          (LENGTH2 (CL:LENGTH ,SEQUENCE2))
          (RESULTLENGTH (CL:LENGTH ,RESULT))
          (INDEX-1 0)
          (INDEX-2 0)
          (RESULT-INDEX 0)
          OBJECT-1 OBJECT-2 ,@(CL:IF KEY
                                  '(KEY-1 KEY-2)))
         (CL:LOOP (COND
                     ((EQL RESULT-INDEX RESULTLENGTH)
                      (RETURN ,RESULT))
                     ((EQL INDEX-1 LENGTH1)
                      (RETURN (CL:REPLACE ,RESULT ,SEQUENCE2 :START1 RESULT-INDEX :START2 INDEX-2)))
                     ((EQL INDEX-2 LENGTH2)
                      (RETURN (CL:REPLACE ,RESULT ,SEQUENCE1 :START1 RESULT-INDEX :START2 INDEX-1))))
                (CL:WHEN (NULL OBJECT-1)
                    (SETQ OBJECT-1 ,(CL:SUBST SEQUENCE1 'OBJECT (CL:SUBST 'INDEX-1 'INDEX ACCESSOR-FORM)))
                    ,@(CL:IF KEY
                          `((SETQ KEY-1 (CL:FUNCALL ,KEY OBJECT-1)))))
                (CL:WHEN (NULL OBJECT-2)
                    (SETQ OBJECT-2 ,(CL:SUBST SEQUENCE2 'OBJECT (CL:SUBST 'INDEX-2 'INDEX ACCESSOR-FORM)))
                    ,@(CL:IF KEY
                          `((SETQ KEY-2 (CL:FUNCALL ,KEY OBJECT-2)))))
                (COND
                   ((CL:FUNCALL ,PRED ,(CL:IF KEY
                                              'KEY-2
                                              'OBJECT-2)
                           ,(CL:IF KEY
                                 'KEY-1
                                 'OBJECT-1))
                    (CL:SETF ,(CL:SUBST RESULT 'OBJECT (CL:SUBST 'RESULT-INDEX 'INDEX ACCESSOR-FORM))
                           OBJECT-2)
                    (CL:INCF INDEX-2)
                    (SETQ OBJECT-2 NIL))
                   (T (CL:SETF ,(CL:SUBST RESULT 'OBJECT (CL:SUBST 'RESULT-INDEX 'INDEX ACCESSOR-FORM))
                             OBJECT-1)
                      (CL:INCF INDEX-1)
                      (SETQ OBJECT-1 NIL)))
                (CL:INCF RESULT-INDEX))))


(CL:DEFUN %SIMPLE-MERGE (RESULT SEQUENCE1 SEQUENCE2 PRED)
   (%MERGE-MACRO RESULT SEQUENCE1 SEQUENCE2 PRED (CL:ELT OBJECT INDEX)))


(CL:DEFUN %MERGE (RESULT SEQUENCE1 SEQUENCE2 PRED KEY)
   (%MERGE-MACRO RESULT SEQUENCE1 SEQUENCE2 PRED (CL:ELT OBJECT INDEX)
          KEY))


(CL:DEFUN %SIMPLE-MERGE-VECTORS (RESULT VECTOR1 VECTOR2 PRED)
   (%MERGE-MACRO RESULT VECTOR1 VECTOR2 PRED (CL:AREF OBJECT INDEX)))


(CL:DEFUN %MERGE-VECTORS (RESULT VECTOR1 VECTOR2 PRED KEY)
   (%MERGE-MACRO RESULT VECTOR1 VECTOR2 PRED (CL:AREF OBJECT INDEX)
          KEY))

;; list merge functions


(CL:DEFUN %SIMPLE-MERGE-LISTS (LIST1 LIST2 PRED)

   ;; %SIMPLE-MERGE-LISTS destructively merges LIST1 with LIST2 In the resulting list, elements of LIST2 are guaranteed to come after equal
   ;; elements of LIST1

   (CL:DO* ((HANDLE (LIST NIL))
            (LAST-CONS HANDLE))

           ;; LAST-CONS = pointer to last cell of result. Done when either list used up in which case, append the other list. Returns the result sans
           ;; header.

           ((OR (NULL LIST1)
                (NULL LIST2))
            (CL:IF (NULL LIST1)
                (RPLACD LAST-CONS LIST2)
                (RPLACD LAST-CONS LIST1))
            (CDR HANDLE))
       (COND
          ((CL:FUNCALL PRED (CAR LIST2)
                  (CAR LIST1))

           ;; Append the lesser list to last cell of result. Note: test must be done for LIST2 < LIST1 so merge will be stable for LIST1

           (RPLACD LAST-CONS LIST2)
           (SETQ LIST2 (CDR LIST2)))
          (T (RPLACD LAST-CONS LIST1)
             (SETQ LIST1 (CDR LIST1))))
       (SETQ LAST-CONS (CDR LAST-CONS))))
```

```
(CL:DEFUN %MERGE-LISTS (LIST1 LIST2 PRED KEY)

   ;; %MERGE-LISTS* destructively merges LIST1 with LIST2 In the resulting list, elements of LIST2 are guaranteed to come after equal elements of
   ;; LIST1

   (CL:DO* ((HANDLE (LIST NIL))
            (LAST-CONS HANDLE)
           KEY-1 KEY-2)

           ;; LAST-CONS = pointer to last cell of result. Done when either list used up in which case, append the other list. Returns the result sans
           ;; header.

           ((OR (NULL LIST1)
                (NULL LIST2))
            (CL:IF (NULL LIST1)
                (RPLACD LAST-CONS LIST2)
                (RPLACD LAST-CONS LIST1))
            (CDR HANDLE))
       (CL:IF (NULL KEY-1)
           (SETQ KEY-1 (CL:FUNCALL KEY (CAR LIST1))))
       (CL:IF (NULL KEY-2)
           (SETQ KEY-2 (CL:FUNCALL KEY (CAR LIST2))))
       (COND
          ((CL:FUNCALL PRED KEY-2 KEY-1)

           ;; Append the lesser list to last cell of result. Note: test must be done for LIST2 < LIST1 so merge will be stable for LIST1

           (RPLACD LAST-CONS LIST2)
           (SETQ LIST2 (CDR LIST2))
           (SETQ KEY-2 NIL))
          (T (RPLACD LAST-CONS LIST1)
             (SETQ LIST1 (CDR LIST1))
             (SETQ KEY-1 NIL)))
       (SETQ LAST-CONS (CDR LAST-CONS))))

;; user entry points


(CL:DEFUN CL:SORT (SEQUENCE PREDICATE &KEY KEY)
   "Destructively sorts sequence. Predicate should returns non-NIL if Arg1 is to precede Arg2."

;;; Sort dispatches to type specific sorting routines.

   (SEQ-DISPATCH SEQUENCE (%SORT-SUBLIST SEQUENCE NIL PREDICATE KEY)
          (%SORT-VECTOR SEQUENCE PREDICATE KEY)))


(CL:DEFUN CL:STABLE-SORT (SEQUENCE PREDICATE &KEY KEY)
   "Destructively sorts Sequence. Predicate should return non-Nil if Arg1 is to precede Arg2. Stable sort is the
   same as sort, but it guarantees that equal elements will not change places. For lists, use the normal
   sort-list function, but vectors must use a less efficient algorithm."
   (SEQ-DISPATCH SEQUENCE (%SORT-SUBLIST SEQUENCE NIL PREDICATE KEY)
          (%STABLE-SORT-VECTOR SEQUENCE PREDICATE KEY)))


(CL:DEFUN CL:MERGE (RESULT-TYPE SEQUENCE1 SEQUENCE2 PREDICATE &KEY (KEY NIL KEY-P))

   ;; The sequences SEQUENCE1 and SEQUENCE2 are destructively merged into a sequence of type RESULT-TYPE using the PREDICATE to order
   ;; the elements.

   (CL:IF (AND (EQ RESULT-TYPE 'LIST)
               (CL:LISTP SEQUENCE1)
               (CL:LISTP SEQUENCE2))
       (CL:IF KEY-P
           (%MERGE-LISTS SEQUENCE1 SEQUENCE2 PREDICATE KEY)
           (%SIMPLE-MERGE-LISTS SEQUENCE1 SEQUENCE2 PREDICATE))
       (LET ((RESULT (MAKE-SEQUENCE-OF-TYPE RESULT-TYPE (+ (CL:LENGTH SEQUENCE1)
                                                           (CL:LENGTH SEQUENCE2)))))
            (CL:IF KEY-P
                (CL:IF (AND (CL:VECTORP RESULT)
                            (CL:VECTORP SEQUENCE1)
                            (CL:VECTORP SEQUENCE2))
                    (%MERGE-VECTORS RESULT SEQUENCE1 SEQUENCE2 PREDICATE KEY)
                    (%MERGE RESULT SEQUENCE1 SEQUENCE2 PREDICATE KEY))
                (CL:IF (AND (CL:VECTORP RESULT)
                            (CL:VECTORP SEQUENCE1)
                            (CL:VECTORP SEQUENCE2))
                    (%SIMPLE-MERGE-VECTORS RESULT SEQUENCE1 SEQUENCE2 PREDICATE)
                    (%SIMPLE-MERGE RESULT SEQUENCE1 SEQUENCE2 PREDICATE))))))

(PUTPROPS CMLSORT FILETYPE CL:COMPILE-FILE)

(DECLARE\: DONTCOPY DOEVAL@COMPILE DONTEVAL@LOAD

(DECLARE\: DOEVAL@COMPILE DONTCOPY

(LOCALVARS . T)
)
)
```

```
(PUTPROPS CMLSORT COPYRIGHT ("Venue & Xerox Corporation" 1986 1990))
```

**{MEDLEY}<sources>CMLSORT.;1      28-Jun-2024 18:34:03**
-- Listed on 30-Jun-2024 13:15:35 --

## FUNCTION INDEX

## MACRO INDEX

## PROPERTY INDEX