Figure A1. List merge sort in memory.

```
MODULE List_merge
    SUB List_merge ( inp$(), links() )
        LOCAL lower_bound, upper_bound, zero, z_times_two, i, j, k, q
        LOCAL link_01, link_02, krun, kother, inp_j$, inp_k$, vi$
        ! Check for degenerate cases of zero or one key(s)
        ! Bounds of links() used to index inp$()
        LET lower_bound = Lbound( links)
            LET upper_bound = Ubound( links)
        IF upper_bound <= lower_bound THEN
           IF upper_bound < lower_bound THEN
              EXIT SUB
           END IF
           LET links( lower_bound) = lower_bound
           EXIT SUB
        END IF
        ! Use "z" to represent zero, < z for end-runs, and
        ! z_times_two - p to negate a pointer p,
        ! where z = Lbound( links() )- 1 and z_times_two = z*2.
        LET zero = lower_bound - 1
        LET z_times_two = 2 * zero
        ! link_01 as the second smallest; link_02 as the first smallest
        ! link_02 may be a null list
        ! link_01 is a null list if only two keys
        LET link_02 = lower_bound
        LET link_01 = link_02 + 1
        IF inp$( link_01) < inp$( link_02) THEN
           LET link_01 = link_02
           LET link_02 = link_01 + 1
        END IF
        LET krun = link_01
        LET kother = link_02
        LET inp_j$, inp_k$ = inp$( link_01)
        FOR i = lower_bound + 2 TO upper_bound
            LET vi$ = inp$( i) ! vi$ is current element
            IF vi$ < inp_j$ THEN  ! inp_j$ is inp$(link_01)
               ! Check for new or almost minimum
               IF vi$ >= inp$( link_02) THEN
                  LET links( i) = link_01
                  LET link_01 = i
               ELSE  ! New real minimum
                  LET links( i) = links( link_02)
                  LET links( link_02) = link_01
                  LET link_01 = link_02
                  IF kother = link_02 THEN
                     LET kother = i
                  END IF
                  LET link_02 = i
               END IF
               LET inp_j$ = inp$( link_01)
            ELSE
               IF vi$ >= inp_k$ THEN
                 ! inp_k$ is last value in run; this goes on the end
```

```
         LET links( krun) = i
       ELSE
          ! Can not start a new run
          LET links( kother) = z_times_two - i
          LET kother = krun
       END IF
       LET krun = i
       LET inp_k$ = vi$
    END IF
NEXT i
LET links( krun), links( kother) = zero
LET links( link_01) = z_times_two - links( link_01)
DO  ! Make multiple passes
   LET i = links( link_01)
   LET j = links( link_02)
   IF j = zero THEN
      EXIT DO
   END IF
   LET krun = link_01
   LET kother = link_02
   DO  ! Loop through runs on a single pass
      LET i = z_times_two - i
      LET j = z_times_two - j
      LET vi$ = inp$( i)
      LET inp_j$ = inp$( j)
      LET k = krun
      DO  ! Merge one run
         IF vi$ <= inp_j$ THEN
            LET links( k) = i
            LET k = i
            LET i = links( i)
            IF i <= zero THEN
               LET links( k) = j
               DO
                  LET k = j
                  LET j = links( j)
               LOOP WHILE j > zero
               EXIT DO
            END IF
            LET vi$ = inp$(i)
         ELSE
            LET links( k) = j
            LET k = j
            LET j = links( j)
            IF j <= zero THEN
               LET links( k) = i
               DO
                  LET k = i
                  LET i = links( i)
               LOOP WHILE i > zero
               EXIT DO
            END IF
            LET inp_j$ = inp$(j)
         END IF
      LOOP
      ! Mark start of run
```

```
            LET links( krun) = z_times_two - links( krun)
            LET krun = kother  ! Switch output list
            LET kother = k  ! Save the end of this run
         LOOP UNTIL j = zero
         LET links( krun) = i
         LET links( kother) = zero
      LOOP
      LET links( link_02) = link_01
      LET links( link_01) = z_times_two - links( link_01)
      ! Complete the sorted list
      LET i = link_02
      FOR j = ( z_times_two - lower_bound) TO
              ( z_times_two - upper_bound) STEP - 1
          LET k = links( link_02)
          LET links( link_02) = j
          LET link_02 = k
      NEXT j
      FOR i = lower_bound TO upper_bound
          LET j = links( i)
          IF j < zero THEN
             LET j = z_times_two - j
             DO
                LET q = z_times_two - links( j)
                LET links( j) = k
                LET k = j
                LET j = q
             LOOP WHILE j > zero
          END IF
      NEXT i
   END SUB
END MODULE
```

Figure A2. Radix hash sort in memory.

```
! Number of elements to sort
LET n = 2 ^ 9
! Record length is 26 bytes
LET rec_len = 26
! Possible bit patterns in one byte = 256
LET kd = 256
! Maximum bit pattern number from zero index
LET theta = 256 - 1
!
OPTION BASE 0
DIM inp$( 0)
MAT REDIM inp$( n)
DIM inp( 0)
MAT REDIM inp( n)
MAT inp = 0
DIM hash$( 0, 0)
! Array already has element subscript 0
MAT REDIM hash$( theta, theta)
MAT hash$ = Nul$
! Initialize values in input array of first sort pass
FOR i = n TO 1 STEP - 1
    LET inp( i) = i
NEXT i
! Radix sort with hash sort for each key chunk
! Process 2-byte chunks of each record to sort
FOR h = rec_len TO 2 STEP - 2
    MAT hash$ = Nul$  ! Clear hash$ array
    ! Hash sort chunks
    FOR j = 1 TO n
        ! Must be 1..n, not n..1, to make it stable
        ! Byte order is h - 1 ... h
        !   with row as h - 1 and column as h
        LET col = ORD( inp$( inp( j))[ h    : h    ])
        LET row = ORD( inp$( inp( j))[ h - 1: h - 1])
        LET hash$( row, col) =
            hash$( row, col) & NUM$( inp( j))
    NEXT j
    ! One greater due to subtraction operation below
    LET count = n + 1
    ! Read IEEE 8-byte record numbers into inp index
    FOR row = theta TO 0 STEP - 1
        FOR col = theta TO 0 STEP - 1
            ! Sum number of sub keys for each hash pattern
            FOR rec_num =
              LEN( hash$( row, col)) / 8 TO 1 STEP - 1
                LET count = count - 1
                LET char_num = ( rec_num - 1) * 8
                LET inp( count) =  NUM( hash$( row, col)
                           [ char_num + 1: char_num + 8])
            NEXT rec_num
        NEXT col
    NEXT row
NEXT h
END
```

Figure A3. Performance for list merge sort and radix hash sort.

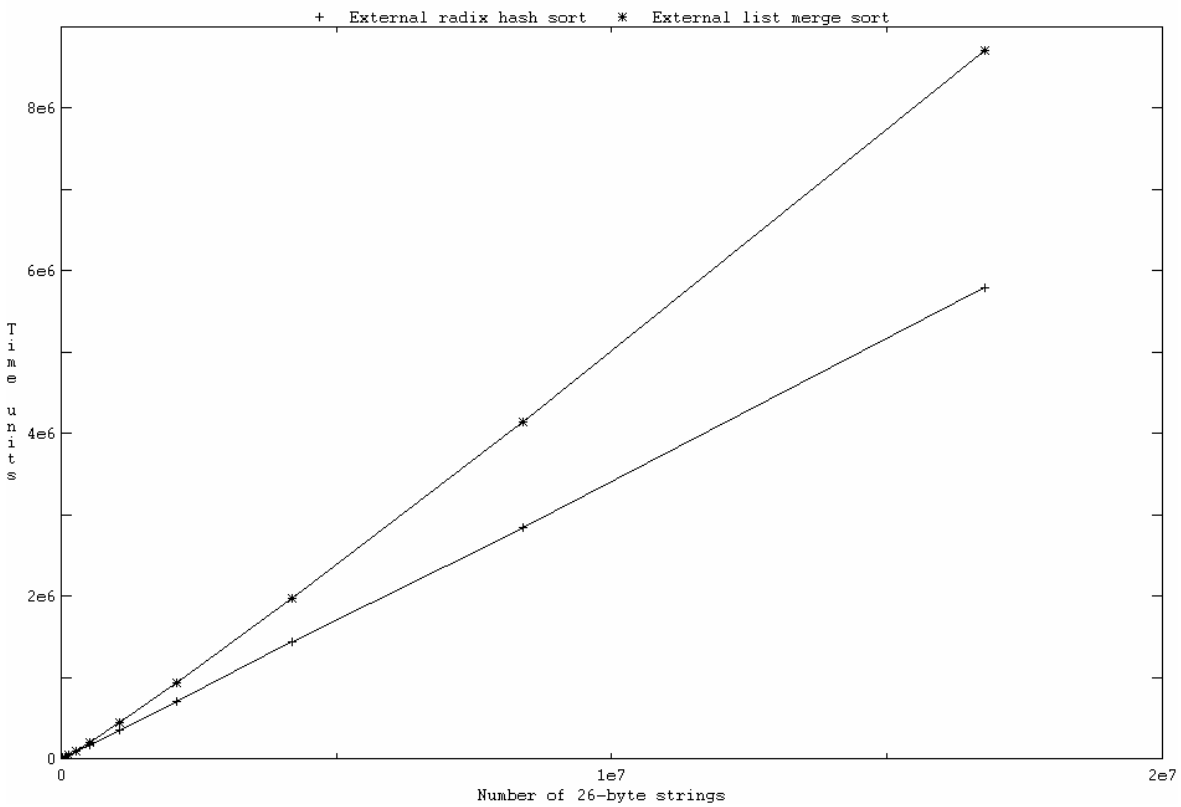

Figure A4. Logarithmic performance of list merge and radix hash sorts.