\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_BUBBLE\_SORT\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

BUBBLE\_SORT ( )

1. pass <- 1

2. sorted <- false

3. while (not sorted) and (pass < n)

4.       sorted <- true

5.       for i <- 0 to n-pass-1

6.             if A[i] > A[i+1]

7.                   troque A[i] <-> A[i+1]

8.                   sorted <- false

9.       pass <- pass + 1

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_QUICKSoRT\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

QUICKSORT (p, r)

1. if p < r

2.    then q <- PARTITION (p, r)

3.         QUICKSORT (p, q)

4.         QUICKSORT (q+1, r)

PARTITION (p, r)

 1. piv <- A[p]

 2. i <- p - 1

 3. j <- r + 1

 4. while (true)

 5.       do j <- j - 1

 6.       while A[j] > piv

 7.       do i <- i + 1

 8.       while A[i] < piv

 9.       if i < j

10.          then troque A[i] <-> A[j]

11.          else return j

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_INSERTION\_SORT\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

INSERTION\_SORT ( )

1. for k <- 1 to n-1

2.       x <- A[k]

3.       i <- k - 1

4.       found <- false

5.       while (not found) and (i >= 0)

6.             if A[i] > x

7.                then troque A[i+1] <-> A[i]

8.                     i <- i - 1

9.                else found <- true

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_MERGESORT\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

MERGESORT (lo, hi)

1. if lo < hi then

2.    mid <- (lo + hi) / 2

3.    MERGESORT (lo, mid)

4.    MERGESORT (mid+1, hi)

5.    L <- lo

6.    H <- mid + 1

7.    for k <- lo to hi

8.          if L <= mid and (H > hi or A[L] < A[H])

9.              then scratch[k] <- A[L]

10.                  L <- L + 1

11.             else scratch[k] <- A[H]

12.                  H <- H + 1

13.    for k <- lo to hi

14.          A[k] <- scratch[k]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_SELECTION\_SORT\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

SELECTION\_SORT ( )

1. for i <- 0 to n-2

2.       current <- i

3.       for k <- i+1 to n-1

4.             if A[current] > A[k]

5.                then current <- k

6.       troque A[i] <-> A[current]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_HEAPSORT \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

HEAPSORT ( )

1. BUILD-HEAP ( )

2. for k <- n - 1 downto 1

3.       troque A[0] <-> A[k]

4.       tam-heap <- tam-heap - 1

5.       HEAPIFY (0)

BUILD-HEAP ( )

1. tam-heap <- n

2. for j <- (n div 2) - 1 downto 0

3.       HEAPIFY (j)

HEAPIFY (i)

 1. l = 2 \* i + 1

 2. r = 2 \* i + 2

 3. if l < tam-heap and A[l] > A[i]

 4.    then max <- l

 5.    else max <- i

 6. if r < tam-heap and A[r] > A[max]

 7.    then max <- r

 8. if max != i

 9.    then troque A[i] <-> A[max]

10.         HEAPIFY (max)