QUICKSORT PSEUDOCODE

```
// Quicksort algorithm: This code sorts A[low..high].
quicksort(A[], int low, int high)
  if (low < high)                  // lists of size=0 or 1 don't need to be sorted
    pos = partition(A, low, high)
    quicksort(A, low, pos - 1)
    quicksort(A, pos + 1, high)

// The partition function chooses the [low] element in the array A as the pivot.
// It rearranges A so that all the elements less than the pivot are on the left
// side, and all the elements greater than the pivot are on the right side.
partition(A[], int low, int high)
  pivot = A[low]  // choose pivot as first element in sub-array of A
  i = low         // i=index for search starting on the left
  j = high        // j=index for search starting on the right

  while (i < j)
    // search from right for an element <= pivot
    while (A[j] > pivot)
      j = j - 1

    // search from left for an element > pivot
    while (i < j && A[i] <= pivot)
      i = i + 1

    if (i < j)
      swap A[i] and A[j]

  // End of searches; place pivot in correct spot (index j).
  pos = j
  A[low] = A[pos]
  A[pos] = pivot
  return pos
```

Notes:

- First call to quicksort should be quicksort(A, 0, A.size()-1).
- Often programmers will make an overloaded quicksort(A[]) function that calls the three-argument version.
- Choice of the pivot matters a lot. In the real world, choosing the pivot as the left-most element (at index low) is a bad idea, e.g., results in poor performance for arrays that are already sorted.