Quicksort Algorithm For Linked-Lists, Reading

Applying quicksort`to linked lists differs from its application to arrays in that it uses pointers instead of indexes. And it's easier and more straight-forward if it's a **doubly-linked** list.

Here's how the algorithm changes for a list whose start and end pointers are named START and END. Note that lowercase "start" and "end" are used for "sub lists" within the full list.

```
define a struct with two Node*'s to store start and end pointers
create a stack of Node*-Node* objects using the struct
create a Node*-Node* object with START and END, and push onto stack
while the stack is not empty
  create and set start pointer = stack top's start
  create and set end pointer = stack top's end
  pop the top of the stack
  if start == end, "continue;"
  if start's prev == end, "continue;"
  calculate the pivot pointer = start

  create and set a left pointer = start
  create and set a right pointer = end
  start a loop

    while left != pivot && value at left <= value at pivot
      left = left's next

    while pivot != right && value at pivot <= value at right
      right = right's prev

    if left == right
      create an object with start=start, end=pivot's prev; push
      create an object with start=pivot's next, end=end; push
      break out of the loop
    else if left == pivot
      swap value at right with value at pivot
      set pivot to right
      left = left's next
    else if right == pivot
      swap value at left with value at pivot
      set pivot to left
      right = right's prev
    else
      swap value at left with value at right
      left = left's next
      right = right's prev
```

Also note that the pivot is the *first* value in the list instead of the middle, simply because the middle is not easy to find.