Daily Coding Problem #169

Problem

This problem was asked by Google.

Given a linked list, sort it in O(n log n) time and constant space.

For example, the linked list `4 -> 1 -> -3 -> 99` should become `-3 -> 1 -> 4 -> 99`.

Solution

We can sort a linked list in O(n log n) by doing something like merge sort:

- Split the list in half by using `fast` and `slow` pointers.
- Recursively sort each half list (base case: when size of list is 1).
- Merge the sorted halves together by using the standard `merge` algorithm.

However, since we divide the list in half and recursively sort it, the function call stack can grow and use up to `log n` space. We want to do this in constant space.

Since the problem here comes from the call stack, we can transform the algorithm into an iterative one and keep track of the array indices ourselves to use only constant space. We can do this by merging blocks at a time from the bottom-up. Let `k` be equal to 1. Then we'll merge lists of size `k` into lists of size `2k`. Then double `k` and repeat, until there are no more merges left to be done.

As an example, consider the linked list `8 -> 6 -> 3 -> 21 -> 12 -> 20 -> 23 -> 5`.

After the first pass, we'll combine all pairs so that they're sorted:

```
6 -> 8 -> 3 -> 21 -> 12 -> 20 -> 5 -> 23
```

And then all groups of 4:

```
3 -> 6 -> 8 -> 21 -> 5 -> 12 -> 20 -> 23
```

And then finally the entire list:

```
3 -> 5 -> 6 -> 8 -> 12 -> 20 -> 21 -> 23
```

```
class Node:
    def __init__(self, val, nxt=None):
        self.val = val
        self.next = nxt

def sort(head):
    if not head:
        return head

    k = 1

    while True:
        first = head
        head = None
        tail = None

        merges = 0
        while first:
            merges += 1

            # Move second `k' steps forward.
            second = first
            first_size = 0
            for i in range(k):
                first_size += 1
                second = second.next
                if second is None:
                    break

            # Merge lists first and second.
            second_size = k
            while first_size > 0 or (second_size > 0 and second is not None):
                e = None
                if first_size == 0:
                    e = second
                    second = second.next
                    second_size -= 1
                elif second_size == 0 or second is None:
                    e = first
                    first = first.next
                    first_size -= 1
                elif first.val <= second.val:
                    e = first
```

```
                first = first.next
                first_size -= 1
            else:
                e = second
                second = second.next
                second_size -=1

            if tail is not None:
                tail.next = e
            else:
                head = e
            tail = e

        first = second


    tail.next = None
    if merges <= 1:
        return head

    k = k * 2
```

This takes O(n log n) time and constant space.