Sort List

Sort a linked list in $O(n \log n)$ time using constant space complexity.

Solution 1

```
/**
 * Definition for singly-linked list.
 * class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) {
 *         val = x;
 *         next = null;
 *     }
 * }
 */
public class Solution {
    public ListNode sortList(ListNode head) {
        if (head == null || head.next == null)
            return head;
        ListNode f = head.next.next;
        ListNode p = head;
        while (f != null && f.next != null) {
            p = p.next;
            f =  f.next.next;
        }
        ListNode h2 = sortList(p.next);
        p.next = null;
        return merge(sortList(head), h2);
    }
    public ListNode merge(ListNode h1, ListNode h2) {
        ListNode hn = new ListNode(Integer.MIN_VALUE);
        ListNode c = hn;
        while (h1 != null && h2 != null) {
            if (h1.val < h2.val) {
                c.next = h1;
                h1 = h1.next;
            }
            else {
                c.next = h2;
                h2 = h2.next;
            }
            c = c.next;
        }
        if (h1 != null)
            c.next = h1;
        if (h2 != null)
            c.next = h2;
        return hn.next;
    }
}
```

written by potpie original link here

Solution 2

```
public class Solution {

  public ListNode sortList(ListNode head) {
    if (head == null || head.next == null)
      return head;

    // step 1. cut the list to two halves
    ListNode prev = null, slow = head, fast = head;

    while (fast != null && fast.next != null) {
      prev = slow;
      slow = slow.next;
      fast = fast.next.next;
    }

    prev.next = null;

    // step 2. sort each half
    ListNode l1 = sortList(head);
    ListNode l2 = sortList(slow);

    // step 3. merge l1 and l2
    return merge(l1, l2);
  }

  ListNode merge(ListNode l1, ListNode l2) {
    ListNode l = new ListNode(0), p = l;

    while (l1 != null && l2 != null) {
      if (l1.val < l2.val) {
        p.next = l1;
        l1 = l1.next;
      } else {
        p.next = l2;
        l2 = l2.next;
      }
      p = p.next;
    }

    if (l1 != null)
      p.next = l1;

    if (l2 != null)
      p.next = l2;

    return l.next;
  }

}
```

written by jeantimex original link here

Solution 3

Nice problem. I use a non-recurisve way to write merge sort. For example, the size of ListNode is 8,

Round #1 block_size = 1

(a1, a2), (a3, a4), (a5, a6), (a7, a8)

Compare a1 with a2, a3 with a4 ...

Round #2 block_size = 2

(a1, a2, a3, a4), (a5, a6, a7, a8)

merge two sorted arrays (a1, a2) and (a3, a4), then merge tow sorted arrays(a5, a6) and (a7, a8)

Round #3 block_size = 4

(a1, a2, a3, a4, a5, a6, a7, a8)

merge two sorted arrays (a1, a2, a3, a4), and (a5, a6, a7, a8)

No need for round #4 cause block_size = 8 >= n = 8

```
class Solution {
public:
    int count_size(ListNode *node){
        int n = 0;
        while (node != NULL){
            node = node->next;
            ++n;
        }
        return n;
    }
    ListNode *sortList(ListNode *head) {
        int block_size = 1, n = count_size(head), iter = 0, i = 0, a = 0, b = 0;
        ListNode virtual_head(0);
        ListNode *last = NULL, *it = NULL, *A = NULL, *B = NULL, *tmp = NULL;
        virtual_head.next = head;
        while (block_size < n){
            iter = 0;
            last = &virtual_head;
            it = virtual_head.next;
            while (iter <  n){
                a = min(n - iter, block_size);
                b = min(n - iter - a, block_size);

                A = it;
                if (b != 0){
                    for (i = 0; i < a - 1; ++i) it = it->next;
                    B = it->next;
                    it->next = NULL;
                    it = B;

                    for (i = 0; i < b - 1; ++i) it = it->next;
                    tmp = it->next;
                    it->next = NULL;
                    it = tmp;
                }

                while (A || B){
                    if (B == NULL || (A != NULL && A->val <= B->val)){
                        last->next = A;
                        last = last->next;
                        A = A->next;
                    } else {
                        last->next = B;
                        last = last->next;
                        B = B->next;
                    }
                }
                last->next = NULL;
                iter += a + b;
            }
            block_size <<= 1;
        }
        return virtual_head.next;
    }
};
```

written by vaputa original link here

From Leetcoder.