AlgoExpert   Quad Layout   C++   12px   Sublime   Monok

Prompt   Scratchpad   Our Solution(s)   Video Explanation   Run Code

Solution 1

```cpp
// Copyright © 2020 AlgoExpert, LLC. All rights reserved.

using namespace std;

class LinkedList {
public:
  int value;
  LinkedList *next;

  LinkedList(int value) {
    this->value = value;
    next = NULL;
  }
};

struct LinkedListPair {
  LinkedList *head;
  LinkedList *tail;
};

LinkedListPair growLinkedList(LinkedList *head, LinkedList *tail,
                              LinkedList *node);
LinkedListPair connectLinkedLists(LinkedList *headOne, LinkedList *tailOne,
                                  LinkedList *headTwo, LinkedList *tailTwo);

// O(n) time | O(1) space - where n is the number of nodes in the Linked List
LinkedList *rearrangeLinkedList(LinkedList *head, int k) {
  LinkedList *smallerListHead = NULL;
  LinkedList *smallerListTail = NULL;
  LinkedList *equalListHead = NULL;
  LinkedList *equalListTail = NULL;
  LinkedList *greaterListHead = NULL;
  LinkedList *greaterListTail = NULL;

  LinkedList *node = head;
  while (node != NULL) {
    if (node->value < k) {
      LinkedListPair smallerList =
          growLinkedList(smallerListHead, smallerListTail, node);
      smallerListHead = smallerList.head;
      smallerListTail = smallerList.tail;
    } else if (node->value > k) {
      LinkedListPair greaterList =
          growLinkedList(greaterListHead, greaterListTail, node);
      greaterListHead = greaterList.head;
      greaterListTail = greaterList.tail;
    } else {
      LinkedListPair equalList =
          growLinkedList(equalListHead, equalListTail, node);
      equalListHead = equalList.head;
      equalListTail = equalList.tail;
    }

    LinkedList *prevNode = node;
    node = node->next;
    prevNode->next = NULL;
  }

  LinkedListPair first = connectLinkedLists(smallerListHead, smallerListTail,
                                            equalListHead, equalListTail);
  LinkedListPair final = connectLinkedLists(first.head, first.tail,
                                            greaterListHead, greaterListTail);
  return final.head;
}

LinkedListPair growLinkedList(LinkedList *head, LinkedList *tail,
                              LinkedList *node) {
  LinkedList *newHead = head;
  LinkedList *newTail = node;

  if (newHead == NULL)
    newHead = node;
  if (tail != NULL)
    tail->next = node;

  return LinkedListPair{newHead, newTail};
}

LinkedListPair connectLinkedLists(LinkedList *headOne, LinkedList *tailOne,
                                  LinkedList *headTwo, LinkedList *tailTwo) {
  LinkedList *newHead = headOne == NULL ? headTwo : headOne;
  LinkedList *newTail = tailTwo == NULL ? tailOne : tailTwo;

  if (tailOne != NULL)
    tailOne->next = headTwo;

  return LinkedListPair{newHead, newTail};
}
```