Quick Sort On Linked List

solved by	Senan
Platform	GeeksForGeeks
difficulty	Medium
tags	Linked List Sorting
language	C++
solved on	@29/10/2024
link	https://www.geeksforgeeks.org/problems/quick-sort-on-linked-list/1
Completion	☑

Intuition

The intuition behind this solution is to apply the QuickSort algorithm on a linked list. Instead of using array indices, we work with linked list pointers to partition the list into two parts: one containing nodes with values less than or equal to the pivot, and the other containing nodes with values greater than the pivot. We use a helper function, `getPivot`, to handle the partitioning around the pivot node, keeping the linked list format intact without converting it to an array. The `quickSort` function recursively sorts the two parts and combines them.

Approach

1. **Base Case**: If the list is empty or has only one element (`head == NULL || head->next == NULL`), it's already sorted, so we return `head`.
2. **Partitioning**: We call `getPivot` to split the list into three parts:
   - `less`: nodes with values less than or equal to the pivot.
   - `head`: the pivot node itself.
   - `more`: nodes with values greater than the pivot.

   The `getPivot` function creates two temporary lists for values `<=` and `>` than the pivot. Once partitioning is done, it cleans up the temporary nodes and returns the partitioned parts.
3. **Recursive Sorting**: We recursively call `quickSort` on both `less` and `more`.
4. **Combine Results**: After recursively sorting, we attach the `second` list (greater elements) to the pivot node. Then, if the `first` list (lesser elements) exists, we find its tail and link it to the pivot node to form the complete sorted list.
5. **Return**: If the `first` list exists, it is the new head; otherwise, the pivot node is the new head.

Complexity

Time Complexity:

- **Average Case**: **O(n log n)**, where **n** is the number of nodes in the linked list. The partitioning divides the list around the pivot node, and recursively sorting each partition on average takes **log n** depth.
- **Worst Case**: **O(n^2)**, which can occur if the list is already sorted or if every pivot results in highly unbalanced partitions.

Space Complexity:

- **O(log n)** for the recursion stack due to recursive calls.

Code

```
class Solution {
    vector<Node*> getPivot(Node* head){
        if(head == NULL || head->next == NULL) return {head, NULL};

        Node* less = new Node(-1);
        Node* more = new Node(-1);
        Node* mark1 = less;
        Node* mark2 = more;

        int val = head->data;

        Node* temp = head->next;
        while(temp) {
            if(temp->data <= val) {
                less->next = temp;
                less = temp;
            } else {
                more->next = temp;
                more = temp;
            }
            temp = temp->next;
        }

        head->next = NULL;
        less->next = NULL;
        more->next = NULL;

        less = mark1->next;
        more = mark2->next;

        delete mark1;
        delete mark2;

        return {less, head, more};
    }

  public:
    struct Node* quickSort(struct Node* head) {
        if(head == NULL || head->next == NULL) return head;

        auto pivot = getPivot(head);
        Node* first = quickSort(pivot[0]);
        Node* second = quickSort(pivot[2]);

        head->next = second;

        if(first) {
            Node* temp = first;
            while(temp->next) temp = temp->next;
            temp->next = head;
            return first;
        }
        return head;
    }
};
```