GeeksforGeeks

A computer science portal for geeks

Practice IDE Q&A GeeksQuiz

QuickSort on Singly Linked List

QuickSort on Doubly Linked List is discussed here. QuickSort on Singly linked list was given as an exercise. Following is C++ implementation for same. The important things about implementation are, it changes pointers rather swapping data and time complexity is same as the implementation for Doubly Linked List.

In **partition()**, we consider last element as pivot. We traverse through the current list and if a node has value greater than pivot, we move it after tail. If the node has smaller value, we keep it at its current position.

In **QuickSortRecur()**, we first call partition() which places pivot at correct position and returns pivot. After pivot is placed at correct position, we find tail node of left side (list before pivot) and recur for left list. Finally, we recur for right list.

```
// C++ program for Quick Sort on Singly Linled List
#include <iostream>
#include <cstdio>
using namespace std;

/* a node of the singly linked list */
struct node
{
    int data;
    struct node *next;
};

/* A utility function to insert a node at the beginning of linked list */
void push(struct node** head_ref, int new_data)
{
    /* allocate node */
    struct node* new_node = new node;

    /* put in the data  */
    new_node->data  = new_data;

    /* link the old list off the new node */
    new_node->next = (*head_ref);

    /* move the head to point to the new node */
    (*head_ref)    = new_node;
}

/* A utility function to print linked list */
void printList(struct node *node)
{
    while (node != NULL)
    {
        printf("%d  ", node->data);
        node = node->next;
    }
    printf("\n");
}

// Returns the last node of the list
struct node *getTail(struct node *cur)
{
    while (cur != NULL && cur->next != NULL)
        cur = cur->next;
    return cur;
}

// Partitions the list taking the last element as the pivot
struct node *partition(struct node *head, struct node *end,
                       struct node **newHead, struct node **newEnd)
{
    struct node *pivot = end;
    struct node *prev = NULL, *cur = head, *tail = pivot;

    // During partition, both the head and end of the list might change
    // which is updated in the newHead and newEnd variables
    while (cur != pivot)
    {
        if (cur->data < pivot->data)
        {
            // First node that has a value less than the pivot - becomes
            // the new head
            if ((*newHead) == NULL)
                (*newHead) = cur;

            prev = cur;
            cur = cur->next;
        }
        else // If cur node is greater than pivot
        {
            // Move cur node to next of tail, and change tail
            if (prev)
                prev->next = cur->next;
            struct node *tmp = cur->next;
            cur->next = NULL;
            tail->next = cur;
            tail = cur;
            cur = tmp;
        }
    }

    // If the pivot data is the smallest element in the current list,
    // pivot becomes the head
    if ((*newHead) == NULL)
        (*newHead) = pivot;

    // Update newEnd to the current last node
    (*newEnd) = tail;

    // Return the pivot node
    return pivot;
}


//here the sorting happens exclusive of the end node
struct node *quickSortRecur(struct node *head, struct node *end)
{
    // base condition
    if (!head || head == end)
```

```
        return head;

    node *newHead = NULL, *newEnd = NULL;

    // Partition the list, newHead and newEnd will be updated
    // by the partition function
    struct node *pivot = partition(head, end, &newHead, &newEnd);

    // If pivot is the smallest element - no need to recur for
    // the left part.
    if (newHead != pivot)
    {
        // Set the node before the pivot node as NULL
        struct node *tmp = newHead;
        while (tmp->next != pivot)
            tmp = tmp->next;
        tmp->next = NULL;

        // Recur for the list before pivot
        newHead = quickSortRecur(newHead, tmp);

        // Change next of last node of the left half to pivot
        tmp = getTail(newHead);
        tmp->next =  pivot;
    }

    // Recur for the list after the pivot element
    pivot->next = quickSortRecur(pivot->next, newEnd);

    return newHead;
}

// The main function for quick sort. This is a wrapper over recursive
// function quickSortRecur()
void quickSort(struct node **headRef)
{
    (*headRef) = quickSortRecur(*headRef, getTail(*headRef));
    return;
}

// Driver program to test above functions
int main()
{
    struct node *a = NULL;
    push(&a, 5);
    push(&a, 20);
    push(&a, 4);
    push(&a, 3);
    push(&a, 30);

    cout << "Linked List before sorting \n";
    printList(a);

    quickSort(&a);

    cout << "Linked List after sorting \n";
    printList(a);

    return 0;
}
```

Run on IDE

Output:

```
Linked List before sorting
30  3  4  20  5
Linked List after sorting
3  4  5  20  30
```

This article is contributed by **Balasubramanian.N** . Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

49 Comments Category: Linked Lists Sorting

Related Posts:

- Merge two sorted linked lists such that merged list is in reverse order
- Compare two strings represented as linked lists
- Rearrange a given linked list in-place.
- Sort a linked list that is sorted alternating ascending and descending orders?
- Select a Random Node from a Singly Linked List
- Merge Sort for Doubly Linked List
- Point to next higher value node in a linked list with an arbitrary pointer
- Swap nodes in a linked list without swapping data

(Login to Rate and Mark)

3 Average Difficulty : **3/5.0**
Based on **1** vote(s)

Add to TODO List

Mark as DONE

Like Share 87 people like this.

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.