

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 */
```

Google™ Custom Search



GeeksforGeeks



53,527 people like [GeeksforGeeks](#).



Facebook

[Interview Experiences](#)

[Advanced Data Structures](#)

[Dynamic Programming](#)

[Greedy Algorithms](#)

[Backtracking](#)

[Pattern Searching](#)

[Divide & Conquer](#)

[Mathematical Algorithms](#)

[Recursion](#)

```

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;
    }
    prev->next = pivot;
    *newEnd = pivot;
}

```



Popular Posts

All permutations of a given string

Memory Layout of C Programs

Understanding "extern" keyword in C

Median of two sorted arrays

Tree traversal without recursion and without stack!

Structure Member Alignment, Padding and Data Packing

Intersection point of two Linked Lists

Lowest Common Ancestor in a BST.

Check if a binary tree is BST or not

Sorted Linked List to Balanced BST

```

        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;

```

Custom market
research at scale.

Get \$75 off

 Google consumer surveys





Recent Comments

Abhi You live US or India?

Google (Mountain View) interview · 46 minutes ago

Aman Hi, Why arent we checking for conditions...

Write a C program to Delete a Tree. · 1 hour ago

kzs please provide solution for the problem...

Backtracking | Set 2 (Rat in a Maze) · 1 hour ago

Sanjay Agarwal bool

tree::Root_to_leaf_path_given_sum(tree...

Root to leaf path sum equal to a given number · 1 hour ago

GOPI GOPINATH @admin Highlight this sentence "We can easily...

Count trailing zeroes in factorial of a number · 1 hour ago

newCoder3006 If the array contains negative numbers also. We...

Find subarray with given sum · 2 hours ago

AdChoices

[▶ Linked List](#)

[▶ C++ Code](#)

[▶ QuickSort](#)

AdChoices

[▶ QuickSort](#)

```

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;
}

```

Output:

Linked List before sorting

30 3 4 20 5


Linked List after sorting

3 4 5 20 30

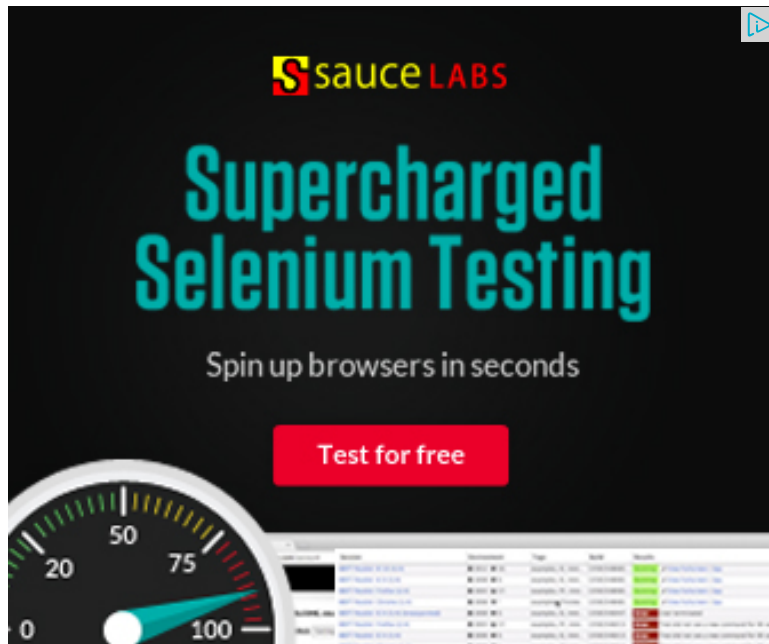
This article is contributed by **Balasubramanian.N** . Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

[▶ C++ Java](#)

[▶ Java Array](#)

AdChoices 

[▶ Node](#)



Related Tpoics:

- [Given a linked list, reverse alternate nodes and append at the end](#)
- [Pairwise swap elements of a given linked list by changing links](#)
- [Self Organizing List | Set 1 \(Introduction\)](#)
- [Merge a linked list into another linked list at alternate positions](#)
- [Delete N nodes after M nodes of a linked list](#)
- [Design a stack with operations on middle element](#)
- [Swap Kth node from beginning with Kth node from end in a Linked List](#)
- [QuickSort on Doubly Linked List](#)



82



Tweet

3



1

Writing code in comment? Please use ideone.com and share the link here.

20 Comments

GeeksforGeeks

Sort by Newest ▼



Join the discussion...



rupali • a month ago

there is an error:

in quickSortRecur function,

before calling it recursively for right part, we need to check if (pivot->next != NU

so it will be

```
if(pivot!=newEnd)
```

```
    pivot->next = quickSortRecur(pivot->next, newEnd);
```

otherwise it may give segmentation error, if such a condition is encountered.

1 ^ | v • Reply • Share ›



Zheng Luo • 2 months ago

Very Good source code, thanks for sharing.

^ | v • Reply • Share ›



Indra Kumar Gurjar • 4 months ago

You can see a very sort programme for it...

<http://ideone.com/T72pBP>

1 ^ | v • Reply • Share ›



Indra Kumar Gurjar • 4 months ago

```
#include<stdio.h>
#define null (node*)(-1)
typedef struct node node;
struct node
{
    node *next;
    int value;
};
node* new_node(int value,node* next)
{
    node* newn;
    newn=(node*)malloc(sizeof(node));
    newn->value=value;
    newn->next=next;
    return newn;
}
```

// function for quick_sort of linked list//

[see more](#)

^ | v • Reply • Share ›



Prasanna • 5 months ago

why not exchange the data instead of changing the pointers? it would be more

^ | v • Reply • Share ›



confused • 9 months ago

Hi,

I am kind of confused with the base case, shouldn't it just be

```
if ( head == end)
```

```
return head;
```

Thanks.

^ | v • Reply • Share ›



Gopi → confused • 9 months ago

The other condition checks for NULL which looks correct.

^ | v • Reply • Share ›



Confused → Gopi • 9 months ago

You mean when the list is empty?

^ | v • Reply • Share ›



Atiqur Rahman • 9 months ago

//Quicksort using singly linked list.

```
#include <stdio.h>
```

```
#include <string.h>
```

```
#include <stdlib.h>
```

```
typedef struct LinkList.
```

```
{
```

```
int data;.
```

```
struct LinkList *next;.
```

```
}List;
```

```
void Insert(List **Head, int value).
```

```
{
```

```
List *ptr=*Head;.
```

```
List *newnode;.
```

```
newnode=(List*)malloc(sizeof(List));.
```

```
newnode->data=value;.
```

```
newnode->next=NULL;.
```



```
if(ptr==NULL).  
*Head=newnode;.
```

[see more](#)

^ | v • Reply • Share ›



atiq • 9 months ago

/ Paste your code here (You may delete these lines if not writing c
//Quicksort using singly linked list.*

```
#include <stdio.h>  
#include <string.h>  
#include <stdlib.h>  
typedef struct LinkList  
{  
    int data;  
    struct LinkList *next;  
}List;  
void Insert(List **Head,int value)  
{  
    List *ptr=*Head;  
    List *newnode;  
    newnode=(List*)malloc(sizeof(List));  
    newnode->data=value;  
    newnode->next=NULL;  
    if(ptr==NULL)
```

[see more](#)

^ | v • Reply • Share ›



LinuxWorld • 9 months ago

///the quick sort

```

#include
#include
struct Node
{
int data ;
struct Node * next ;
};

typedef struct Node Node ;
void partition(Node **head1 , Node ** tail1 , Node ** head2 , Node **tail2 , Node **newhead , Node **newtail)
{
Node *head = *head1 ;
Node *tail = *tail2 ; // tail like null
Node *newhead = head ;
if(head == tail) /// i think there should be some adjustment // i will see it after c
return ;
}

```

[see more](#)

^ | v • Reply • Share ›



Anukul • 10 months ago

My version of quicksort with LinkList as Data Structure

```

/* Paste your code here (You may delete these lines if not writing code)
#include<stdio.h>
#include<conio.h>

struct node
{
int data;
struct node* next;
};

```

```
void add_ll(struct node **,int); //adds nodez to LinkList
void quick_ll(struct node*,struct node*);
struct node *part(struct node*,struct node*);
struct node* search(struct node*,struct node *); //for searching mid
void print(struct node*); //prints LinkList
```

[see more](#)

^ | v • Reply • Share ›



raghson • 10 months ago

Can you please tell me that what is the need of 'prev' pointer in the partition fur

^ | v • Reply • Share ›



skulldude → raghson • 10 months ago

This is just the same as removing a node from a list. When you need to remove a node from a list, you need to make the previous node point to the node to which the

Eg: 1->2->3->4->5

Let's say we are moving 3 to the end. Then, we need to make 2 point to the node to which 3 was pointing. Otherwise, the list becomes inconsistent.

After moving 3 to the end, the list should look like this:

1->2->4->5->3

That is why we are using prev.

Hope it helps.

-Balasubramanian.N

^ | v • Reply • Share ›



raghson → skulldude · 10 months ago

Thanks a lot. I got it. :)

^ | v · Reply · Share ›



venkat_iitg · 11 months ago

In pivot() if the current's data is smaller than pivots data why are u making curi

^ | v · Reply · Share ›



skulldude → venkat_iitg · 11 months ago

Well, a partition function on an array works this way:

- 1) All the elements less than the pivot go before the pivot.
- 2) Those greater than the pivot come after pivot.

So, if the same has to be carried forward to a list, all the nodes smaller and those larger than pivot must come after it.

Here, instead of moving the smaller elements before the pivot, we move the greater elements after the pivot, which has the same effect.

Thus, if there is a smaller element than the pivot in the list, then that will be in the first partition. So, we update the newHead to the first element smaller than the pivot.

Hope this helps.

-Balasubramanian



```
/* Paste your code here (You may delete these lines if not write)
```

^ | v · Reply · Share ›



venkat_iitg → skulldude · 11 months ago

thanks dude. I got it. :)



venkat_iitg → venkat_iitg · 11 months ago

sorry in partition()

1 ^ | v · Reply · Share ›



rohit · 11 months ago

There's another way i found of doing this. I'm using an array of pointers. let the
G[30];

where node is a structure data type.

The linked list is placed on the array starting from G[0].

and you can now quicksort it normally as you now have the indices of all the n

1 ^ | v · Reply · Share ›



Subscribe



Add Disqus to your site

@geeksforgeeks, **Some rights reserved**

Contact Us!

Powered by **WordPress** & **MooTools**, customized by geeksforgeeks team