

QuickSort on Doubly Linked List

Following is a typical recursive implementation of **QuickSort** for arrays. The implementation uses last element as pivot.

```
/* A typical recursive implementation of Quicksort for array*/

/* This function takes last element as pivot, places the pivot element
correct position in sorted array, and places all smaller (smaller than
pivot) to left of pivot and all greater elements to right of pivot */
int partition (int arr[], int l, int h)
{
    int x = arr[h];
    int i = (l - 1);

    for (int j = l; j <= h- 1; j++)
    {
        if (arr[j] <= x)
        {
            i++;
            swap (&arr[i], &arr[j]);
        }
    }
    swap (&arr[i + 1], &arr[h]);
    return (i + 1);
}

/* A[] --> Array to be sorted, l --> Starting index, h --> Ending index */
void quickSort(int A[], int l, int h)
{
    if (l < h)
    {
        int p = partition(A, l, h); /* Partitioning index */
        quickSort(A, l, p - 1);
        quickSort(A, p + 1, h);
    }
}
```

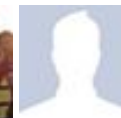
Google™ Custom Search



GeeksforGeeks



53,527 people like GeeksforGeeks.



Interview Experiences

Advanced Data Structures

Dynamic Programming

Greedy Algorithms

Backtracking

Pattern Searching

Divide & Conquer

Mathematical Algorithms

Recursion

Geometric Algorithms

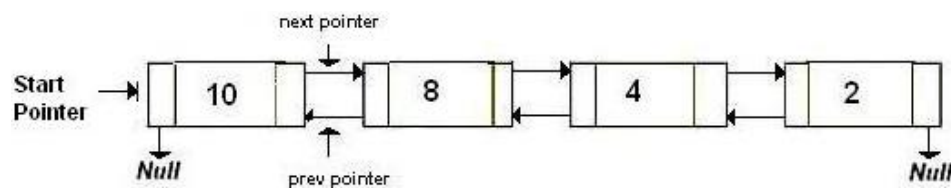
```

    }
}

```

Can we use same algorithm for Linked List?

Following is C++ implementation for doubly linked list. The idea is simple, we first find out pointer to last node. Once we have pointer to last node, we can recursively sort the linked list using pointers to first and last nodes of linked list, similar to the above recursive function where we pass indexes of first and last array elements. The partition function for linked list is also similar to partition for arrays. Instead of returning index of the pivot element, it returns pointer to the pivot element. In the following implementation, quickSort() is just a wrapper function, the main recursive function is _quickSort() which is similar to quickSort() for array implementation.



```

// A C++ program to sort a linked list using Quicksort
#include <iostream>
#include <stdio.h>
using namespace std;

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

/* A utility function to swap two elements */
void swap ( int* a, int* b )
{
    int t = *a;      *a = *b;      *b = t;
}

// A utility function to find last node of linked list
struct node *lastNode (node *root)
{
    while (root && root->next)
        root = root->next;
    return root;
}

```

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

```

/* Considers last element as pivot, places the pivot element at its
   correct position in sorted array, and places all smaller (smaller than
   pivot) to left of pivot and all greater elements to right of pivot
node* partition(node *l, node *h)
{
    // set pivot as h element
    int x = h->data;

    // similar to i = l-1 for array implementation
    node *i = l->prev;

    // Similar to "for (int j = l; j <= h- 1; j++)"
    for (node *j = l; j != h; j = j->next)
    {
        if (j->data <= x)
        {
            // Similar to i++ for array
            i = (i == NULL)? l : i->next;

            swap(&(i->data), &(j->data));
        }
    }
    i = (i == NULL)? l : i->next; // Similar to i++
    swap(&(i->data), &(h->data));
    return i;
}

/* A recursive implementation of quicksort for linked list */
void _quickSort(struct node* l, struct node *h)
{
    if (h != NULL && l != h && l != h->next)
    {
        struct node *p = partition(l, h);
        _quickSort(l, p->prev);
        _quickSort(p->next, h);
    }
}

// The main function to sort a linked list. It mainly calls _quickSort
void quickSort(struct node *head)
{
    // Find last node
    struct node *h = lastNode(head);

    // Call the recursive QuickSort
    _quickSort(head, h);
}

```

Custom market
research at scale.

Get \$75 off

 Google consumer surveys



```
// A utility function to print contents of arr
void printList(struct node *head)
{
    while (head)
    {
        cout << head->data << " ";
        head = head->next;
    }
    cout << endl;
}

/* Function to insert a node at the beginging of the Doubly Linked Lis
void push(struct node** head_ref, int new_data)
{
    struct node* new_node = new node;    /* allocate node */
    new_node->data = new_data;

    /* since we are adding at the begining, prev is always NULL */
    new_node->prev = NULL;

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

    /* change prev of head node to new node */
    if ((*head_ref) != NULL) (*head_ref)->prev = new_node ;

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

/* Driver program to test above function */
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";
```

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](#)

```
    printList(a);  
  
    return 0;  
}
```

Output :

Linked List before sorting

30 3 4 20 5

Linked List after sorting

3 4 5 20 30

Time Complexity: Time complexity of the above implementation is same as time complexity of QuickSort() for arrays. It takes $O(n^2)$ time in worst case and $O(n\log n)$ in average and best cases. The worst case occurs when the linked list is already sorted.


Exercise:

The above implementation is for doubly linked list. Modify it for singly linked list. Note that we don't have prev pointer in singly linked list.

Quicksort can be implemented for Linked List only when we can pick a fixed point as pivot (like last element in above implementation). Random QuickSort cannot be efficiently implemented for Linked Lists.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

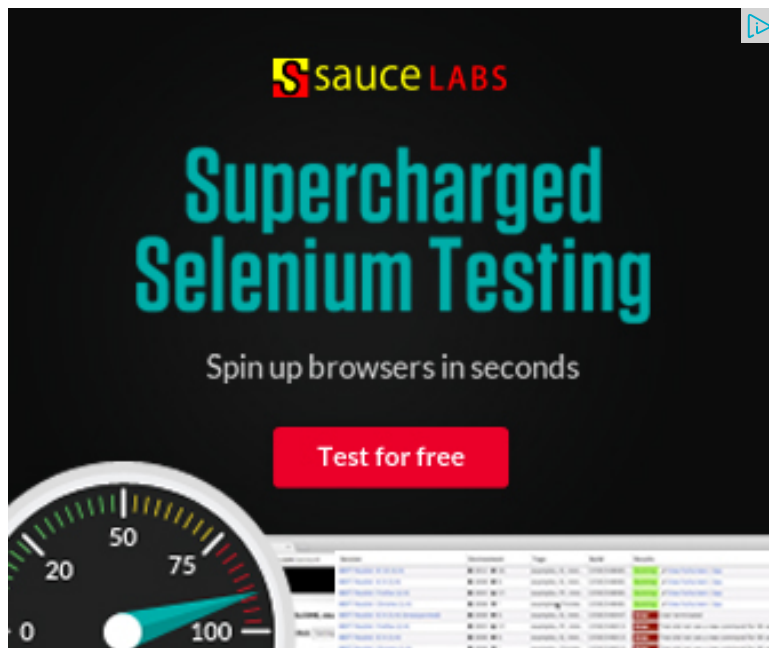
[▶ C++ Java](#)

AdChoices 

[▶ C++ Program](#)

[▶ Test C++](#)

[▶ Null Pointer](#)



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
- QuickSort on Singly Linked List
- 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



33



Tweet

6



1

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

14 Comments

GeeksforGeeks

Sort by Newest ▼





can the algorithm...



Vishal · 10 days ago

what does this statement do ?

```
i = (i == NULL)? I : i->next;
```

^ | v · Reply · Share ›



edward · 8 months ago

why need swap (&arr[i + 1], &arr[h]); after loop?

^ | v · Reply · Share ›



Javed → edward · 7 months ago

to put the pivot which is the last element, at its correct place.

^ | v · Reply · Share ›



LinuxWorld · 9 months ago

sorting the list using linked list

```
#include
```

```
#include
```

```
struct Node
```

```
{
```

```
int data ;
```

```
struct Node * next ;
```

```
};
```

```
typedef struct Node Node ;
```

```
Node *middle(Node *head)
```

```
{
```

```
Node *temp ;
```

```
return head ;  
else if(head->next->next == NULL)  
return head ;
```

[see more](#)

1 ^ | v • Reply • Share ›



Ronny • 11 months ago

@GeeksforGeeks

In the function _quicksort
the condition states that

```
if (h != NULL && l != h && l != h->next)
```

shouldn't it be
h!=l->next

since h pointer contains the last node
and the l pointer contains the first(head)node

so how can head be equal to last->next (Note : It is not a circular linked list)

^ | v • Reply • Share ›



nehamahajan → Ronny • 8 months ago

h is last node of every sublist so h->next is possible and l!= h->next. It
than two elements in sublists.

^ | v • Reply • Share ›



Ronny → Ronny • 11 months ago

its not working if I remove that condition.
Can anyone explain in which case it is required.

^ | v • Reply • Share ›



numid · 11 months ago

One doubt, when and why does low becomes equal to high->next ?
Please comment

```
void _quickSort(struct node* l, struct node *h)
{
    if (h != NULL && l != h && l != h->next)
    {
        struct node *p = partition(l, h);
        _quickSort(l, p->prev);
        _quickSort(p->next, h);
    }
}
```

^ | v · Reply · Share ›



Pushkar → numid · 6 months ago

Hey!... Here in this code when recursive function "void _quickSort(struct node* l, struct node* h)" is called, the pivot element is either the first or second element. There are two cases:
case 1: when pivot element is the first element (p==l)
_quickSort(l, p->prev); set l as l and h as the previous element of p (or l becomes h)
case 2: when pivot element is the second element (p==h)
_quickSort(p->next, h); set h as h and l as the next element of p (or h becomes l)
Hope it would help you!!!

^ | v · Reply · Share ›



Akhil · 11 months ago

@GeeksforGeeks

This partition function gives $O(n^2)$ in all cases.

It would be better to employ selection sort than to use this partition function (because it is $O(n^2)$ in all cases).

Below, i provide a pivoted partition function with complexity $O(n \log n)$.

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

struct dnode
{
    int info;
    struct dnode *b;
    struct dnode *f;
};

typedef struct dnode *Dnode;

void insert(Dnode *head, int num)
```

[see more](#)

^ | v • Reply • Share ›



Scott Shipp • a year ago

My thought is that this is good overall, but I wanted to mention variable names start and end indices is the kind of thing that really interferes with code readab

^ | v • Reply • Share ›



abhishek08aug • a year ago

Intelligent :D

^ | v • Reply • Share ›



Ronny → abhishek08aug • 11 months ago

You seem to have read all the posts/topics of geeksforgeeks.
Cool stuff.

Maybe I should say it for you : "Intelligent :)"

2 ^ | v • Reply • Share ›



Srinath • a year ago

I have one serious doubt...

you are swapping just the data field,in this case it is just a single int but I dont t swap the entire node by adjusting pointers so that no matter what the object tr doesn't change...

You didn't use previous pointer except in one or two cases...

To remove it completely,in partition function store pointer to last element of left i->data..then return this pointer instead of i.

then just change quicksort function to accomodate this.

1 ^ | v • Reply • Share ›



Subscribe



Add Disqus to your site

@geeksforgeeks, **Some rights reserved**

Contact Us!

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