

GeeksforGeeks

A computer science portal for geeks

GeeksQuiz

- [Home](#)
- [Algorithms](#)
- [DS](#)
- [GATE](#)
- [Interview Corner](#)
- [Q&A](#)
- [C](#)
- [C++](#)
- [Java](#)
- [Books](#)
- [Contribute](#)
- [Ask a Q](#)
- [About](#)

[Array](#)

[Bit Magic](#)

[C/C++](#)

[Articles](#)

[GFacts](#)

[Linked List](#)

[MCQ](#)

[Misc](#)

[Output](#)

[String](#)

[Tree](#)

[Graph](#)

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 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 */
int partition (int arr[], int l, int h)
{
    int x = arr[h];
    int i = (l - 1);
```

```

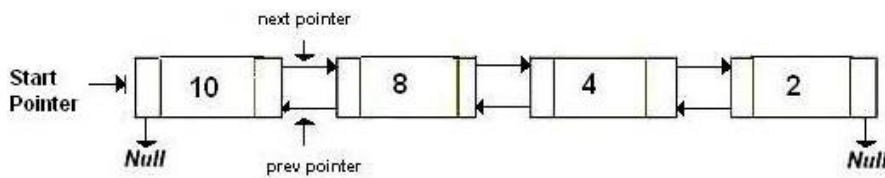
for (int j = 1; j <= h- 1; j++)
{
    if (arr[j] <= x)
    {
        i++;
        swap (&arr[i], &arr[j]);
    }
}
swap (&arr[i + 1], &arr[h]);
return (i + 1);
}

/* 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);
    }
}

```

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;
}

/* 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);
}

// 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 beginning of the Doubly Linked List */
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 beginning, 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";
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.

Can we implement random quick sort for 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 by picking random pivot.

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.

Refer [QuickSort on Singly Linked List](#) for solution.

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

Related Topics:

- [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](#)
- [Generic Linked List in C](#)
- [Clone a linked list with next and random pointer | Set 2](#)
- [Given a linked list of line segments, remove middle points](#)
- [Construct a Maximum Sum Linked List out of two Sorted Linked Lists having some Common nodes](#)
- [Given a linked list, reverse alternate nodes and append at the end](#)



Tweet

6

+1

1

Writing code in comment? Please use [ideone.com](#) and share the link here.

32 Comments

GeeksforGeeks

Login ▾

Recommend 1 Share

Sort by Newest ▾



... ..

[Join the discussion...](#)**Pallavi Singh** • 3 days ago

In one of the posts that i read here, it was said that QuickSort is not the sort for LinkedList as it performs poorly rather merge sort should be used, please clear the confusion someone?

^ | v • Reply • Share ›

**GeeksforGeeks** Mod → Pallavi Singh • 3 days ago

Pallavi,

One possible reason could be caching. QuickSort has good locality of reference when used for arrays, that is one of the main reasons quicksort is preferred over mergesort for arrays.

In linked lists, locality of reference is not possible as nodes are at different locations.

Also, merge sort for arrays requires auxiliary array to merge while QuickSort does sorting in-place. In case of linked lists, merge() operation of mergesort() doesn't require an auxiliary array.

^ | v • Reply • Share ›

**Pallavi Singh** → GeeksforGeeks • 8 hours ago

what do you mean when you say "locality of reference"?

^ | v • Reply • Share ›

**GOPI GOPINATH** → Pallavi Singh • 6 hours ago

Locality of reference is a computer science concept, in simple words, Instead of picking up a single element, we ll pick few elements near to that element because they have maximum possibility to be picked in near future.

^ | v • Reply • Share ›

**Goku** • a month ago

@geeksforgeeks here is the implementation of merge sort for doubly linked list.Please publish this :)

<http://ideone.com/LSnLwA>

^ | v • Reply • Share ›

**Rauf** • 4 months ago

Please Help me...I am doing quick sortBut it does not working fine....i have to enter sorting option again and again to sort data

<http://ideone.com/VXxAul>

^ | v • Reply • Share ›

^ | v • Reply • Share ›



Kerem Sahin • 5 months ago

Implementation by changing links (not data)

<http://ideone.com/qQlvb8>

^ | v • Reply • Share ›



ajayv • 9 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. and its working also and seems to be logical.

that case will never arises bcoz before that they will be cancelled out by the condition (l != h).
please reply ASAP.

^ | v • Reply • Share ›



Vikas Malviya • 10 months ago

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

{

struct node *p = partition(l, h);

_quicksort(l, p->prev);

_quicksort(p->next, h);

}

Why we are doing l!=h->next comparison

5 ^ | v • Reply • Share ›



amateur → Vikas Malviya • 6 months ago

its similar to 'if(l>h)' , which is a condition we don't want.

^ | v • Reply • Share ›



MK → Vikas Malviya • 10 months ago

Did you get answer for this?

^ | v • Reply • Share ›



brahma • a year ago

hear is my implementation swapping nodes instead of data... i used same logic of single linked list

<http://ideone.com/uXGI81>

@geeks can you tell me is there any decrease in run time by implementing in double linked list

^ | v • Reply • Share ›



LinkedListFML • a year ago

Can someone give an implementation based on SWAPPING POINTERS instead of DATA. I tried a lot but was unsuccessful :(

^ | v • Reply • Share ›



Fraklt → LinkedListFML • 10 months ago

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

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

```
struct node
```

```
{
```

```
int data;
```

```
struct node *next;
```

```
struct node *prev;
```

```
};
```

```
void push(struct node **head, int val)
```

```
{
```

```
struct node *temp = (node*)malloc(sizeof(node));
```

see more

^ | v • Reply • Share ›



theCuriosityEnthusiast • a year ago

I'm new to this so can someone explain why randomized Quicksort cannot be efficiently implemented in a LL??

^ | v • Reply • Share ›



RK → theCuriosityEnthusiast • 9 months ago

random access is not possible in LL

^ | v • Reply • Share ›



madan raj • a year ago

!!=h->next case works

when there are less than 2 elements in sublists

and when pivot pointer is the same as end element

case try for 5 and 8

p will be pointing to 8

quicksort (5,5) //avoided l==h

and

quicksort(null,8) in this case l==h->next so avoided

2 ^ | v • Reply • Share ›



Vishal • a year ago

what does this statement do ?

i = (i == NULL)? l : i->next;

^ | v • Reply • Share ›



edward • 2 years ago

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

^ | v • Reply • Share ›



cfh → edward • 2 years ago

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

^ | v • Reply • Share ›



LinuxWorld • 2 years 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 ;
```

```
if(head->next == NULL)
```

```
return head ;
```

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

```
return head ;
```

[see more](#)

1 ^ | v • Reply • Share ›



Ronny • 2 years 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 • 2 years ago

h is last node of every sublist so h->next is possible and l!= h->next. It simply means that there must be more than two elements in sublists.

^ | v • Reply • Share ›



Ronny → Ronny • 2 years ago

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

^ | v • Reply • Share ›



numid • 2 years 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 · 2 years ago

Hey!.. Here in this code when recursive function "void _quickSort(struct node* l, struct node *h)" is narrowed down to two elements then the pivot element is either the first or second element .so..

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) so the condition $l \neq h \rightarrow \text{next}$ is violated()

case 1: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) so the condition $l \neq h \rightarrow \text{next}$ is violated

Hope it would help you!!!

1 ^ | v · Reply · Share ›

**ajayv** → Pushkar · 9 months ago

the cases which you are talking about they are special cases and they will violate the condition of ($l \neq h$).. . that means you are not getting the question which numid asked.

^ | v · Reply · Share ›

**Akhil** · 2 years 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 of the recursion cost).

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;
```

[see more](#)

1 ^ | v · Reply · Share ›

**Scott Shipp** · 2 years ago

My thought is that this is good overall, but I wanted to mention variable names could be clearer.

Using "l" and "h" for start and end indices is the kind of thing that really interferes with code readability and maintainability.

1 ^ | v • Reply • Share ›



abhishek08aug • 2 years ago

Intelligent :D

^ | v • Reply • Share ›



Ronny → abhishek08aug • 2 years ago

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

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

4 ^ | v • Reply • Share ›



Srinath • 2 years 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 think that is a good practice.You should swap the entire node by adjusting pointers so that no matter what the object that the linked list is holding the code 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 partition before swapping h->data and i->data..then return this pointer instead of i.

then just change quicksort function to accomodate this.

3 ^ | v • Reply • Share ›

[Subscribe](#)

[Add Disqus to your site](#)

[Privacy](#)



- - [Interview Experiences](#)
 - [Advanced Data Structures](#)
 - [Dynamic Programming](#)
 - [Greedy Algorithms](#)
 - [Backtracking](#)
 - [Pattern Searching](#)
 - [Divide & Conquer](#)
 - [Mathematical Algorithms](#)
 - [Recursion](#)
 - [Geometric Algorithms](#)
-

• 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](#)

Follow @GeeksforGeeks

• Recent Comments

- [It_k](#)
i need help for coding this function in java...
[Java Programming Language](#) · [1 hour ago](#)
- [Piyush](#)
What is the purpose of else if (recStack[*i])...
[Detect Cycle in a Directed Graph](#) · [1 hour ago](#)
- [Andy Toh](#)
My compile-time solution, which agrees with the...
[Dynamic Programming | Set 16 \(Floyd Warshall Algorithm\)](#) · [1 hour ago](#)
- [lucy](#)
because we first fill zero in first col and...

[Dynamic Programming | Set 29 \(Longest Common Substring\)](#) · [2 hours ago](#)

- [lucy](#)

@GeeksforGeeks i don't n know what is this long...

[Dynamic Programming | Set 28 \(Minimum insertions to form a palindrome\)](#) · [2 hours ago](#)

- [manish](#)

Because TAN is not a subsequence of RANT. ANT...

[Given two strings, find if first string is a subsequence of second](#) · [2 hours ago](#)

•

@geeksforgeeks, [Some rights reserved](#) ____ [Contact Us!](#)

Powered by [WordPress](#) & [MooTools](#), customized by geeksforgeeks team