GeeksforGeeks

A computer science portal for geeks

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;
}
```

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

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

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

Sunil Sharma • 5 days ago

There is no need to move the elements which are greater than pivot element.

You can just move when the elements are small.
For eg.
C(CURRENT)
N(NEXT) ;

during the first partition call .
4->8->7->21->3->16->11-6
N P
C
here we select the pivot element as 6 .

4<6 swap 4 with curr and move , next element is 8.

4->8->7->21->3->16->11-6
C=4 , N=4.

8>6, don't swap the elements move to next.

---

see more

• Reply • Share ›

Vineeth Reddy • 8 days ago
If pivot is the largest element there is no need to QuickSortRecur the right side of pivot.
• Reply • Share ›

Shivangi Dhakad • 3 months ago
here's my code when instead of exchanging links, we swap the data. Bit easier.
http://ideone.com/bmbZx5
• Reply • Share ›

omar salem • 4 months ago
C# code
http://ideone.com/sL6enU
• Reply • Share ›

Chetan Kamani • 5 months ago
if i want to make middle element as pivot, then its complexity become quadratic weather it is sorted or not. am i right?
• Reply • Share ›

Ashish Jaiswal ↗ Chetan Kamani • 4 months ago
when you make its middle element pivot....its stable...and you get Best case....which is O(nlogn)
Best Case: The best case occurs when the partition process always picks the middle element as pivot. Following is recurrence for best case.

T(n) = 2T(n/2) + \theta(n)

The solution of above recurrence is \theta(nLogn). It can be solved using case 2 of Master Theorem.
• Reply • Share ›

codecrecker • 7 months ago
#include<stdio.h>

#include<stdlib.h>

```
typedef struct nd node;

struct nd{

int d;

node * n;

node * p;

};

node *head=NULL,*cur=NULL,*pivot;int flg=0;node *lp;

void print()

{
```

see more

• Reply • Share ›

Meet • 8 months ago

no need to write struct node *next in the structure (simply write node *next) because structure defines a new data type in C++ not just tag as in C.

• Reply • Share ›

kamran siddique • 9 months ago

http://ideone.com/sTiiAS

• Reply • Share ›

pawan • 9 months ago

```
#include<iostream>

#include<cstdlib>

#include<cmath>

using namespace std;

struct node

{

int data;

struct node *next;

};
```

```
typedef struct node *nodeptr;

void printList(nodeptr head)
```

see more

• Reply • Share ›

xyz • 10 months ago

i am getting segmentation fault error on executing this code.Kindly help

• Reply • Share ›

Mayur ➚ xyz • 10 months ago

Find my Implementation of above code..

http://ideone.com/IuzT8H

• Reply • Share ›

Guest • 10 months ago

we can also use the same approach as quick sort on doubly linked list
we have to just pass the prv node of head on which partition is applied
code is as:

```
#include<iostream>
#include<iomanip>
using namespace std;
typedef struct list
{
int data;
struct list *next;
}node;
node* createlist(node**);
void printlist(node**);
void quicksort(node**,node*,node*);
void partition(node*,node*,node*,node**,node**);
node* findprv(node**,node*);
void swap(int*,int*);
int main()
```

see more

• Reply • Share ›

anand • a year ago

```
void sortList() {

NODE *tmp1=NULL,*tmp2=NULL,*tmp3=NULL;

int num:
```

```
if(start != NULL){

tmp1 = start;

while(tmp1->next != NULL || tmp1 != head)

{

tmp2 =tmp3= tmp1;

num = tmp1->val;

while(tmp2->next != NULL || tmp2 != head)

{
```

see more

• Reply • Share ›

rupali • a year ago

their is an error:

in quickSortRecur function,

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

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.

3 • Reply • Share ›

Zheng Luo • a year ago

Very Good source code, thanks for sharing.

• Reply • Share ›

Indra Kumar Gurjar • a year ago

You can see a very sort programme for it...
http://ideone.com/T72pBP

2 • Reply • Share ›

Indra Kumar Gurjar • a year ago

```
#include<stdio.h>
#define null (node*)( 1)
```

```
#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

• Reply • Share ›

Ashish Jaiswal ➦ Indra Kumar Gurjar • 4 months ago

I really liked this code and just modifying it to make the pivot last node of LL...as in CLRS last node has been choosen as Pivot..
Corrction: return 0;
at last statment of int main()

• Reply • Share ›

Prasanna • a year ago

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

• Reply • Share ›

confused • 2 years ago

Hi,

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

if (head == end)

return head;

Thanks.

• Reply • Share ›

Gopi ➦ confused • 2 years ago

The other condition checks for NULL which looks correct

The other condition checks for NULL which looks correct.

• Reply • Share ›

Confused → Gopi • 2 years ago

You mean when the list is empty?

• Reply • Share ›

Atiqur Rahman • 2 years 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

1 • Reply • Share ›

atiq • 2 years ago

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

```
List *newnode;
newnode=(List*)malloc(sizeof(List));
newnode->data=value;
newnode->next=NULL;
if(ptr==NULL)
```

see more

• Reply • Share ›

LinuxWorld • 2 years ago

///the quick sort

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

typedef struct Node Node ;
void partition(Node **head1 , Node ** tail1 , Node ** head2 , Node **tail2 , Node **key1)
{
Node *head = *head1 ;
Node *tail = *tail2 ; // tail like null
Node *newhead = head ;
if(head == tail) /// i thiink there should be some adjustment // i will see it after completion

see more

• Reply • Share ›

Anukul • 2 years 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-1
void print(struct node*);//prints LinkList
```

see more

• Reply • Share ›

raghson • 2 years ago

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

• Reply • Share ›

skulldude ➚ raghson • 2 years ago

This is just the same as removing a node from a list. When you need to remove a node from a singly linked list, you need to make the previous node point to the node to which the node-to-be-deleted is pointing to.

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

Let's say we are moving 3 to the end. Then, we need to make 2 point to 4, 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

• Reply • Share ›

raghson ➚ skulldude • 2 years ago

Thanks a lot. I got it. :)

• Reply • Share ›

venkat_iitg • 2 years ago

In pivot() if the current's data is smaller than pivots data why are u making current to new head..??

• Reply • Share ›

skulldude ➚ venkat_iitg • 2 years 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 than the pivot must come before pivot and those larger than pivot must come after it.
Here, instead of moving the smaller elements before the pivot, we move the larger 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 the first node in the list after partition.So, we update the newHead to the first element smaller than the pivot that we encounter.

Hope this helps.

-Balasubramanian

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

• Reply • Share ›

venkat_iitg skulldude • 2 years ago
thanks dude. I got it. :)
• Reply • Share ›

venkat_iitg venkat_iitg • 2 years ago
sorry in partition()
1 • Reply • Share ›

rohit • 2 years ago
There's another way i found of doing this. I'm using an array of pointers. let the name of array of pointers be node* 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 nodes!
1 • Reply • Share ›

- 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

- lt_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