Merge Sort for Linked Lists

Merge sort is often preferred for sorting a linked list. The slow random-access performance of a linked list makes some other algorithms (such as quicksort) perform poorly, and others (such as heapsort) completely impossible.

Let head be the first node of the linked list to be sorted and headRef be the pointer to head. Note that we need a reference to head in MergeSort() as the below implementation changes next links to sort the linked lists (not data at the nodes), so head node has to be changed if the data at original head is not the smallest value in linked list.

```
MergeSort(headRef)
1) If head is NULL or there is only one element in the Linked List
    then return.
2) Else divide the linked list into two halves.
      FrontBackSplit(head, &a, &b); /* a and b are two halves */
3) Sort the two halves a and b.
      MergeSort(a);
      MergeSort(b);
4) Merge the sorted a and b (using SortedMerge() discussed here)
   and update the head pointer using headRef.
     *headRef = SortedMerge(a, b);
```

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

/* Link list node */
struct node
{
    int data;
    struct node* next;
};

/* function prototypes */
struct node* SortedMerge(struct node* a, struct node* b);
void FrontBackSplit(struct node* source,
          struct node** frontRef, struct node** backRef);

/* sorts the linked list by changing next pointers (not data) */
void MergeSort(struct node** headRef)
{
```

```
  struct node* head = *headRef;
  struct node* a;
  struct node* b;

  /* Base case -- length 0 or 1 */
  if ((head == NULL) || (head->next == NULL))
  {
    return;
  }

  /* Split head into 'a' and 'b' sublists */
  FrontBackSplit(head, &a, &b);

  /* Recursively sort the sublists */
  MergeSort(&a);
  MergeSort(&b);

  /* answer = merge the two sorted lists together */
  *headRef = SortedMerge(a, b);
}

/* See http://geeksforgeeks.org/?p=3622 for details of this
   function */
struct node* SortedMerge(struct node* a, struct node* b)
{
  struct node* result = NULL;

  /* Base cases */
  if (a == NULL)
     return(b);
  else if (b==NULL)
     return(a);

  /* Pick either a or b, and recur */
  if (a->data <= b->data)
  {
     result = a;
     result->next = SortedMerge(a->next, b);
  }
  else
  {
     result = b;
     result->next = SortedMerge(a, b->next);
  }
  return(result);
}

/* UTILITY FUNCTIONS */
/* Split the nodes of the given list into front and back halves,
     and return the two lists using the reference parameters.
     If the length is odd, the extra node should go in the front list.
     Uses the fast/slow pointer strategy.  */
void FrontBackSplit(struct node* source,
          struct node** frontRef, struct node** backRef)
{
  struct node* fast;
  struct node* slow;
  if (source==NULL || source->next==NULL)
  {
    /* length < 2 cases */
    *frontRef = source;
    *backRef = NULL;
  }
  else
  {
```

```
    slow = source;
    fast = source->next;

    /* Advance 'fast' two nodes, and advance 'slow' one node */
    while (fast != NULL)
    {
      fast = fast->next;
      if (fast != NULL)
      {
        slow = slow->next;
        fast = fast->next;
      }
    }

    /* 'slow' is before the midpoint in the list, so split it in two
      at that point. */
    *frontRef = source;
    *backRef = slow->next;
    slow->next = NULL;
  }
}

/* Function to print nodes in a given linked list */
void printList(struct node *node)
{
  while(node!=NULL)
  {
   printf("%d ", node->data);
   node = node->next;
  }
}

/* Function to insert a node at the beginging of the linked list */
void push(struct node** head_ref, int new_data)
{
  /* allocate node */
  struct node* new_node =
            (struct node*) malloc(sizeof(struct 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;
}

/* Drier program to test above functions*/
int main()
{
  /* Start with the empty list */
  struct node* res = NULL;
  struct node* a = NULL;

  /* Let us create a unsorted linked lists to test the functions
   Created lists shall be a: 2->3->20->5->10->15 */
  push(&a, 15);
  push(&a, 10);
  push(&a, 5);
  push(&a, 20);
  push(&a, 3);
  push(&a, 2);
```

```
  /* Sort the above created Linked List */
  MergeSort(&a);

  printf("\n Sorted Linked List is: \n");
  printList(a);

  getchar();
  return 0;
}
```

Run on IDE

Time Complexity: O(nLogn)

Sources:

http://en.wikipedia.org/wiki/Merge_sort

http://cslibrary.stanford.edu/105/LinkedListProblems.pdf

Please write comments if you find the above code/algorithm incorrect, or find better ways to solve the same problem.

92 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)

3.5 Average Difficulty : **3.5/5.0**
Based on **11** vote(s)

Add to TODO List

Mark as DONE

Like Share 23 people like this.

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

92 Comments GeeksforGeeks 1 Login

Recommend 5 Share Sort by Newest

Join the discussion…

Jayesh • 11 days ago

Java Implementation. Both recursive and Iterative approach.

http://javabypatel.blogspot.in...

• Reply • Share ›

girlwhoCodes • 20 days ago

For function void FrontBackSplit(struct node* source,struct node** frontRef, struct node** backRef);
Why do we need to pass parameters frontRef/backRef as double pointer? We're not changing the value of the pointer, what else is the reason? Can anyone please explain this to me?

• Reply • Share ›

Sai Teja girlwhoCodes • 20 days ago

thats not mandatory,

• Reply • Share ›

rcgldr • 2 months ago

A bottom up merge sort for lists is faster. A small fixed size (like 26 to 32) array of pointers to lists along with a standard merge lists function is used. Link to example code below. On my system, Intel 2600K, 3.4ghz, it can sort 4 million nodes with 32 bit integers as data in less than 1 second.

http://code.geeksforgeeks.org/...

• Reply • Share ›

Flower girl • 3 months ago

for some function v use struct node * and some just void...y for mergesort(&a) and for sortedmerge(a) no pointer is used for "a"??help me with this confusion..

• Reply • Share ›

Rahul Sarkar ➚ Flower girl • 3 months ago

just keep in mind one thing that we pass pointer to a node only when we have to update the head of the list.

as we can see in the method mergesort(),we are dividing the list into 2 half and them redividing them.BUt later we r joining them using sortedmerge(),and the list which the sortedmerge() is returning is considered as the current list at that point. Therefore its updating *headRef to that list which is being returned at that time.

And since sortedmerge() is just joining the two list which r passed as arguments,therefore it has nothing to do with the updation of any head.Therefore we have not used pointer to any node.
I hope this helped u little bit..

1 • Reply • Share ›

Flower girl ➚ Rahul Sarkar • 3 months ago

really helpful..thanks a lot..

1 • Reply • Share ›

Nishant • 4 months ago

Why slow->next = NULL; is done in FrontBackSplit at the end?

• Reply • Share ›

Flower girl ➚ Nishant • 3 months ago

bcz that first half becomes a separate list and any list shud end with null !!

• Reply • Share ›

Nishant ➚ Flower girl • 3 months ago

Thanks.

• Reply • Share ›

TheCoder • 4 months ago

Java code for merge sort

http://code.geeksforgeeks.org/...

• Reply • Share ›

reaper • 4 months ago

//what is wrong with this code? Mr.Lazy???Help.

```
import java.io.*;

class mergee{

static public void main(String[] args){

Node a = new Node(2);

Node b = new Node(4);Node c = new Node(5);Node d =new Node(1);

a.next = b;b.next =c;c.next = d;

mergesort(a);

Node n = a;

while(n!=null){

System.out.print(n.data+" ");
```

see more

• Reply • Share ›

happyme • 5 months ago

Can anyone point the mistake in the code?

http://ideone.com/2OyeOI

• Reply • Share ›

Prince Bharti • 5 months ago

what would be the auxiliary space needed for this algorithm over here ?

• Reply • Share ›

Mysterious Mind ➚ Prince Bharti • 4 months ago

Nothing except for stack for recursion execution. (log n)

• Reply • Share ›

Merge sort link lists in java • 5 months ago

{{

```
/**
* @author panicker
*/
package standardAlgos;
```

```
public class SinglyLinkList {

private Node head;

public class Node {
private int data;
private Node nextNode;

Node(int d, Node next) {
data = d;
nextNode = next;
}
```

see more

• Reply • Share ›

coolk • 6 months ago

Java version
http://ideone.com/7hZeB9

1 • Reply • Share ›

Holden • 7 months ago

This is Java version: https://ideone.com/SDJyb5

Can anyone tell me why my code drop some elements?
for example:

Before sorting:
4 -> 2 -> 12 -> 6 -> 5 -> 14 -> 8 -> 1
After sorting:
4 -> 5 -> 6 -> 8 -> 12 -> 14

it should be:
1 -> 4 -> 5 -> 6 -> 8 -> 12 -> 14

• Reply • Share ›

Cracker • 7 months ago

My implementation in C++:

http://algods-cracker.blogspot...

• Reply • Share ›

Goku • 9 months ago

Simpler version http://ideone.com/kq0hZP

• Reply • Share ›

prk • 10 months ago

simple working implementation in java:

https://ideone.com/q9J8OI

• Reply • Share ›

Stack • a year ago

Do we actually need to check the if condition in the split function, that part is already checked in the sorted merge function, we will never get to the if condition in the split function?

• Reply • Share ›

Kenneth • a year ago

The given solution uses O(logn) extra space. Here gives my O(1) space complexity algorithm - iteration solution:

http://ideone.com/mxgJUA

1 • Reply • Share ›

Rob • a year ago

Hello! I have a question. What happen with my list. I insert number, then I use this MergeSort, then print list and show sorted, but then I want insert another number and my list lost some data. What happen?

2 • Reply • Share ›

RACHIT SAXENA • a year ago

http://ideone.com/D0sbg9

simple implementation

• Reply • Share ›

Amit Yadav • a year ago

```
if(source == NULL)
{
*frontRef = *backRef = NULL;
}
else if(source -> next == NULL )
{
*fronRef = source;
*backRef = NULL;
}
else
{
```

.....
}

This code should be put in place of what is there currently in FrontBackSplit() function.Because if source is NULL, source -> next has no meaning.

• Reply • Share ›

Shweta Singh • a year ago

#include<iostream>

#include<stdlib.h>

using namespace std;

struct Node

{

int data;

Node* next;

};

void Insert(Node** head, int item)

{

Node* newnode=new Node();

see more

2 • Reply • Share ›

Justine ➚ Shweta Singh • 4 months ago

hi,
can you help me by explaining the splitInTo function?
i dont understand how the slow and fast works..

• Reply • Share ›

Guest • a year ago

i have a doubt plz clear it.
In merge sort we basically divide the list until we get list of 1 element then start merging and finally we get sorted list which is mainly implemented by recursion.
But here we are just dividing in 2 list then sorting each and merging them.
I think this is not exact merge sort

• Reply • Share ›

Aadil Guest • a year ago
Sorry wrong post..
• Reply • Share ›

Aadil Guest • a year ago
You dont have to consider if and else if separetely.

consider your else if part

if source was NULL then *fronRef=NULL(source);

and *backRef=NULL;

which is same as if part.
• Reply • Share ›

kajol • a year ago
why can't we do it like 1-d array? can anyone pls tell me.
• Reply • Share ›

v3gA kajol • a year ago
We are doing it more or less like 1d array. The list is divided into 2, each is recursively sorted and then merged. The only difference here is that in linked lists, we cannot index into the middle element in O(1) time unlike in arrays. Also, here it is inplace whereas the traditional merge sort on arrays takes O(n) auxiliary space
5 • Reply • Share ›

Holden v3gA • 7 months ago
Great explanation on 2 differences :) thank you
• Reply • Share ›

parit13 • a year ago
one more method for front_back_split:-
void front_back_split(node* main_list , node** front_reference, node** back_reference)
{
node* current=main_list;
int len=length(main_list);
if(len<2)
{
*front_reference=main_list;
*back_reference=NULL;
}

else

```
int count_node=(len-1);
for(int i=0;i<count_node;i++) current="current-">next;

*front_reference=current;
*back_reference=current->next;
current->next=NULL;
}
```

• Reply • Share ›

vishwanath • a year ago

can anyone explain how time

complexity is O(nlogn) ?

• Reply • Share ›

Guru • a year ago

slow->next = NULL; Is very important

1 • Reply • Share ›

deepak ➚ Guru • a year ago

can you please explain why this is very important?

1 • Reply • Share ›

Chaipau ➚ deepak • 5 months ago

If we don't reference the "next" of the end node of first-half of the list to NULL then it's "next" will be the first node of the second of the list...which means we didn't split them into two separate lists.
(Correct me if I am made a mistake).

2 • Reply • Share ›

chirag agrawal • a year ago

This algorithm will throw StackOverFlow error in java if we give a very big Linked List. This solution is not full proof to every kind of input.

1 • Reply • Share ›

Rohan • a year ago

For Doubly Linked List
Can we simply apply the given algo to sort the "next" pointers.
And then traverse the list once again to change the "prev" pointers??

• Reply • Share ›

pawan • a year ago

//merge sort for linked lists

```
#include<iostream>

#include<cstdlib>

using namespace std;

struct node

{

int data;

struct node *next;

};

typedef struct node *nodeptr;

nodeptr getMid(nodeptr head)
```

see more

• Reply • Share ›

shail • a year ago

http://ideone.com/FNeJiI

• Reply • Share ›

Aveek Biswas • 2 years ago

Code in Java using algorithm exactly similar to merge sort in arrays:

1) Break the list into two halves recursively until there is only a single node in each half.

2) Perform SortedMerge() [merging two sorted linked lists] on the two halves.

Link for the code: https://ideone.com/KCcOEp

4 • Reply • Share ›

Holden ➦ Aveek Biswas • 7 months ago

Perfect! Thanks. But when it comes to a list with 2 node, your code fails; for example:
400 -> 200
it only prints: 400
instead of: 200 -> 400

• Reply • Share ›

ANA • 2 years ago

https://ideone.com/eYfGWE

• Reply • Share ›

Himanshu Dagar • 2 years ago

Can go through below link for code : -

http://ideone.com/WcokHu

1 • Reply • Share ›

Prince Bharti Himanshu Dagar • 5 months ago

what would be the auxiliary space needed for this algorithm over here ? considering the stack frame for the recursion.

• Reply • Share ›

Aditya Chhilwar • 2 years ago

FrontBackSplit will be called for every partition. For n elements for first call to this function will traverse n nodes (order of n), for second it will be called two times n/2+n/2 = n and so on. How the time complexity is order of nlogn?

4 • Reply • Share ›

v3gA Aditya Chhilwar • 2 years ago

It is order of O(nlogn). Each recursive call does O(n) work since regardless of whether midpoint-finding is O(1) or O(n), merge() will take O(n) time. So the recurrence relation is the same as that in standard merge sort.

1 • Reply • Share ›

Load more comments