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;
}
```

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.

Related Topics:

- 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
- 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

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

prk • 5 days ago
simple working implementation in java:

https://ideone.com/q9J8OI

Techie Me • 11 days ago
Wonderful article. Really helped me.

I believe sometimes a video makes the audience understand better. Here is a video I published just to analyse the Merge Sort algorithm https://www.youtube.com/watch?...

• Reply • Share ›

Stack • a month 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 • 4 months 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 • 5 months 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 • 5 months ago

http://ideone.com/D0sbg9

simple implementation

• Reply • Share ›

Amit Yadav • 6 months 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 • 6 months 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

1 • Reply • Share ›

Guest • 7 months 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 • 6 months ago

Sorry wrong post..

• Reply • Share ›

Aadil ➚ Guest • 6 months 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 • 7 months ago

why can't we do it like 1-d array? can anyone pls tell me.

• Reply • Share ›

popeye kajol • 7 months 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

• Reply • Share ›

parit13 • 7 months 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 • 7 months ago

can anyone explain how time

complexity is O(nlogn) ?

• Reply • Share ›

Guru • 7 months ago

slow->next = NULL; Is very important

• Reply • Share ›

deepak Guru • 2 months ago

can you please explain why this is very important?

• Reply • Share ›

chirag agrawal • 8 months 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.

• Reply • Share ›

Rohan • 8 months 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??

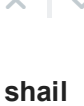
• Reply • Share ›

pawan • 8 months 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 • 8 months ago

http://ideone.com/FNeJiI

• Reply • Share ›

Aveek Biswas • 9 months 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

3 • Reply • Share ›

ANA • 9 months ago

https://ideone.com/eYfGWE

• Reply • Share ›

Himanshu Dagar • a year ago

Can go through below link for code : -

http://ideone.com/WcokHu

1 • Reply • Share ›

Aditya Chhilwar • a year 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 ›

popeye ➚ Aditya Chhilwar • 10 months 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.

• Reply • Share ›

gonecase ➚ popeye • 10 months ago

look, you divide the list(be it array or linked list), log n times(refer to recursion tree), during each partition, given a list of size i=n, n/2, ..., n/2^k, we would take O(i) time to partition the original/already divided list..since sigma O(i)= O(n),we can say , we take O(n) time to partition for any given call of partition(sloppily), so given the time taken to perform a single partition, the question now arises as to how many partitions are going to happen all in all, we observe that the number of partitions at each level i is equal to 2^i , so summing 2^0+2^1+....+2^(lg n) gives us [2(lg n)-1] as the sum which is nothing but (n-1) on simplification , implying that we call partition n-1, (let's approximate it to n), times so , the complexity is atleast big omega of n^2

atleast big omega of n 2..
if i am wrong, please let me know where...thanks:)
• Reply • Share ›

popeye gonecase • 10 months ago
Partition? You mean the split? "number of partitions at each level i is equal to 2^i" - wrong. There is a single split per level of recursion.

I assume you know traditional merge sort on arrays. The recurrence relation is T(n) = 2T(n/2) + O(n). 2T(n/2) corresponding to the 2 recursive calls & O(n) time for the merge operation that happens in each level.

Here, we have a split which takes O(n) time, 2 recursive calls on half size and a merge which takes O(n). So in total it is again T(n) = 2T(n/2) + O(n) giving us the same time complexity as merge sort.
1 • Reply • Share ›

guest11 • a year ago
can anyone explain why slow->next has been set to null
• Reply • Share ›

Atreyee guest11 • 10 months ago
slow is the point before midpoint, so for the first list slow is the last node and for last node, next has to be set to NULL so as to have a check condition for last node
• Reply • Share ›

Ankit • a year ago
better understandable code ...

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

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

struct node * createNode(int value)
{
struct node * N=(struct node *)malloc(sizeof(struct node));
N->next=NULL;
N->data=value;
return N;
}
```

see more

1 • Reply • Share ›

Himanshu Dagar • a year ago

Merge sort whole in recursive way is at below link

http://ideone.com/Wacpan

1 • Reply • Share ›

Anil kumar • a year ago

I hope best way to join sorted list is :

list

merge_sorted_lists(list a, list b)

{

list result, result2;

if(a->data < b->data)

{

result = a;

a = a->next;

}

else

see more

• Reply • Share ›

J • a year ago

This only gives me one half of the list

• Reply • Share ›

Shikha Gupta • 2 years ago

y has not struct node **a been used as parameter in sorted merge function??

• Reply • Share ›

Deepak Shikha Gupta • a year ago

bcz we are returning the result list...

1 • Reply • Share ›

sush • 2 years ago

```
 void SortedMerge(struct node**h,struct node* a, struct node* b)
{


        if(a==NULL)
        {
                *h=b;
                return;
        }
        if(b==NULL)
        {
                *h=a;
                return;
        }
        if(a->data <= b->data)
        {
                *h=a;
                SortedMerge(&((*h)->next),a->next,b);
        }
```

see more

1 ^ | v • Reply • Share ›

Ramesh.Mxian • 2 years ago

I think we can count the number of elements in the list before starting the Merge start and pass the count also as a parameter to merge sort function.

This count can be used to split the list into two parts without having fast and slow pointers.

For example:

assume there are 10 elements in the list. So main function will
call merge sort with MergeSort(Head,10).

We can easily split this list by finding
mid = count/2; //10/2 =5
then traverse the list 5 nodes to find the head of the second part and call merge as
merge(head,mid)
merge(secondHead,count-mid)

everything else will work as usuall

2 ^ | v • Reply • Share ›

Priyanka • 2 years ago

```
I think it should be like this
```

```
I think it should be like this
/* Advance 'fast' two nodes, and advance 'slow' one node */
    while (fast != NULL)
    {
      fast = fast->next;
      if (fast_next != NULL)
      {
        slow = slow->next;
        fast = fast->next;
      }
    }
```

1 • Reply • Share ›

kartik ➦ Priyanka • 2 years ago

@Priyanka: I think the code given is correct. Any reason for this change? What is 'fast_next' in your code?

• Reply • Share ›

priyankasingh.136 ➦ kartik • 2 years ago

[sourcecode language=""]
Sorry, it's fast-> next not fast_next. if we check for if (fast != NULL), slow will point
to middle of the linked list. But as you said, slow should point to one node before the
middle. So we should check for if (fast->next != NULL). Please let me know if it's wrong.

• Reply • Share ›

vishanything • 3 years ago

Can't we just store the addresses of each node in an array and then sort the whole thing on the basis of the data present in those addresses?

• Reply • Share ›

Ankit Gupta • 3 years ago

T(n) = T(n/2) + a * (n/2) + b * n
Where a * (n/2) for computing mid-point (or partitioning) and b * n for merge step.
So O(n log n)

[sourcecode language="java"]
Node mergeSort(Node head, Node rear) {
if (head == rear) {
head.next = null;
return head;
`

```
}
Node mid = getMidNode(head, rear);
Node right = mergeSort(mid.next, rear); // Compute right half first
Node left = mergeSort(head, mid); // Compute left
return mergeList(left, right);
}

Node getMidNode(Node head, Node rear) {
Node it1, it2;
```

see more

• Reply • Share ›

chandeepsingh85 Ankit Gupta • a year ago

Could you please explain why here " while (it2 != rear && it2.next != rear) "

&& is used? I am unable to grasp the logic.

• Reply • Share ›

Chiranjeev Kumar • 3 years ago

```
 /* Paste your code here (You may delete these lines if not writing code)
// Selection Sort
#include<stdio.h>
typedef struct node
{
    int value;
    struct node *next;
}mynode;
void add(mynode **head,int data)
{
    mynode *temp = (mynode *)malloc(sizeof(struct node));
    temp->value = data;
    temp->next = NULL;
    mynode *t = *head;
    if(!t)
    {
        printf(".............Creating SLL.......\n");
        *head = temp;
```

see more

• Reply • Share ›

Rushi Agrawal • 3 years ago

Great code. Very lucid. Would have been better had an explanation on the use of pointers to pointers was given.

pointers was given.

1 • Reply • Share ›

tuhin • 3 years ago

i just wished to know if it meant insitu merging

• Reply • Share ›

rka143 • 4 years ago

I believe in solution 1 should have following condition in place of written one:
Now:
if (fast != NULL) { slow = slow->next; fast = fast->next; }
Correct:
if (fast != NULL && fast->next != NULL) { slow = slow->next; fast = fast->next; }

Reason:
In the case of 2 nodes, it will continue to split the list in infinite loop. Where one part is NULL and other part has 2 nodes.

Please let me know if somebody have differnet thought or need more clarification.

• Reply • Share ›

trinadh • 4 years ago

How about this?

```
 /* Merge two sorted linked lists */
Node * merge(Node *first, Node *second) {
    Node head; //dummy Node which points to merged list
    Node *last_sorted = &head; //points to last Node of sorted list

    while (first != NULL && second != NULL) {
        if (first->data < second->data) {
            last_sorted->next = first;
            last_sorted = first;
            first = first->next;
        }
        else {
            last_sorted->next = second;
            last_sorted = second;
            second = second->next;
        }
```

see more

1 • Reply • Share ›

gantashala venki trinadh • 2 years ago

trinadh i think ur merge function is better because it avoids recursion in merge function....which will prevent stack over flow in the case of large data.:)

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

1 • Reply • Share ›

Load more comments

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

Recent Comments

- Baba

  Traverse the tree in inorder fashion and insert...

  Populate Inorder Successor for all nodes · 54 minutes ago

- creeping_death

  Ruby solution, slightly different, easier to...

Longest Even Length Substring such that Sum of First and Second Half is same · 1 hour ago

- darkprotocol

  Anyone round-1 4th question?

  Amazon Interview | Set 121 (On-Campus for SDE-1) · 1 hour ago

- S.Nkm

  Would a simple answer involving the Unix epoch...

  Find number of days between two given dates · 1 hour ago

- Shomina

  Found Real Job INterview Questions Here...

  Flipkart Interview Experience | Set 20 (For SDE-II) · 2 hours ago

- Shimona

  Found Real Job INterview Questions here...

  Myntra Interview Experience | Set 4 (For Senior Software Engineer) · 2 hours ago