

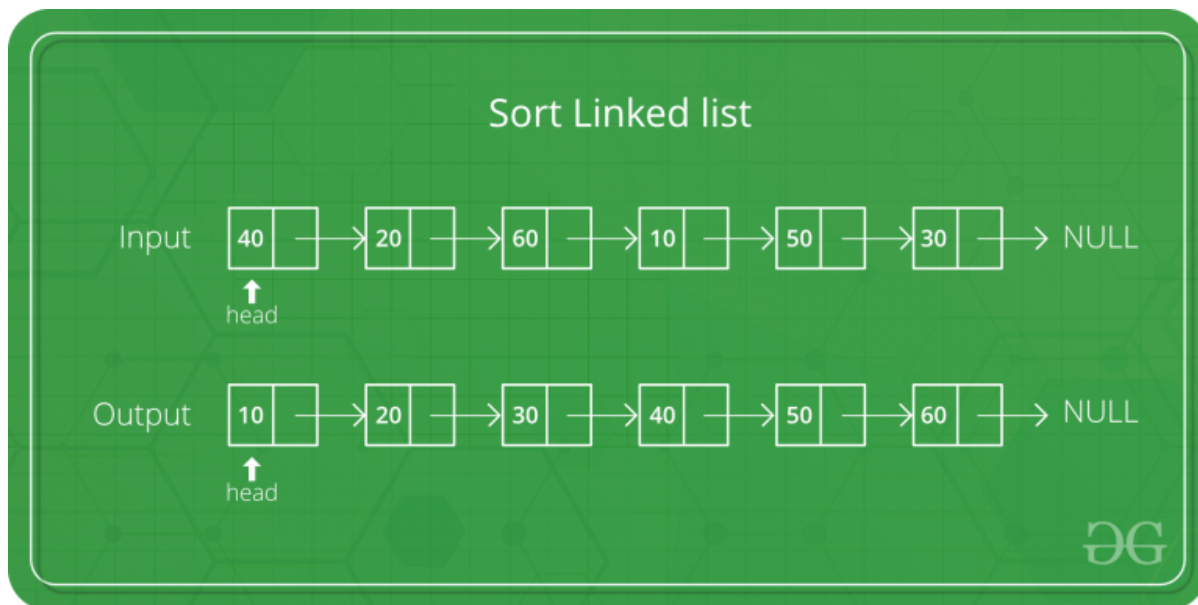
[Array](#) [Matrix](#) [Strings](#) [Hashing](#) [Linked List](#) [Stack](#) [Queue](#) [Binary Tree](#) [Binary Search](#)

Merge Sort for Linked Lists

Difficulty Level : Hard • Last Updated : 26 Jul, 2022



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 the 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



DigitalOcean

Simpler tools lead to happier developers. And
happier developers lead to better results

GET \$100 FREE CREDIT

[HIDE AD](#) • [AD VIA BUYSPELLADS](#)

```
MergeSort(headRef)
```

- 1) If the head is NULL or there is only one element in the Linked 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);`

Recommended Practice

Merge Sort for Linked List

Try It!



Complete Interview Preparation - Self Paced

By Sandeep Jain

Beginner to Advance Level ★★★★★

Find 360 solution to all of your interview woes. Learn 4 years' worth of programming knowledge in just 6 months and ace coding interviews at top tech companies.

[Explore Now](#)

C++

```
// C++ code for linked list merged sort
#include <bits/stdc++.h>
using namespace std;
```



Simpler tools lead to happier developers. And
happier developers lead to better results

GET \$100 FREE CREDIT

[HIDE AD](#) • [AD VIA BUYSPELLADS](#)

```
public:
    int data;
    Node* next;
};

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

/* sorts the linked list by changing next pointers (not data) */
void MergeSort(Node** headRef)
{
    Node* head = *headRef;
    Node* a;
    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 https:// www.geeksforgeeks.org/?p=3622 for details of this
function */
Node* SortedMerge(Node* a, Node* b)
{
    Node* result = NULL;

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

```
/* 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(Node* source,
                    Node** frontRef, Node** backRef)
{
    Node* fast;
    Node* slow;
    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 */
```



```
        cout << node->data << " ";
        node = node->next;
    }
}

/* Function to insert a node at the beginning of the linked list */
void push(Node** head_ref, int new_data)
{
    /* allocate node */
    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;
}

/* Driver program to test above functions*/
int main()
{
    /* Start with the empty list */
    Node* res = NULL;
    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);

    cout << "Sorted Linked List is: \n";
    printList(a);
}
```

C

```
// C code for linked list merged sort
#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);
}
```

```
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;
    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;
        }
    }
}
```

```
        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 beginning 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;
}

/* Driver 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, 2);

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

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

getchar();
return 0;
```

Java

```
// Java program to illustrate merge sorted
// of linkedList

public class linkedList {
    node head = null;
    // node a, b;
    static class node {
        int val;
        node next;

        public node(int val)
        {
            this.val = val;
        }
    }

    node sortedMerge(node a, node b)
    {
        node result = null;
        /* Base cases */
        if (a == null)
            return b;
        if (b == null)
            return a;

        /* Pick either a or b, and recur */
        if (a.val <= b.val) {
            result = a;
```

```
        result = b;
        result.next = sortedMerge(a, b.next);
    }
    return result;
}

node mergeSort(node h)
{
    // Base case : if head is null
    if (h == null || h.next == null) {
        return h;
    }

    // get the middle of the list
    node middle = getMiddle(h);
    node nextofmiddle = middle.next;

    // set the next of middle node to null
    middle.next = null;

    // Apply mergeSort on left list
    node left = mergeSort(h);

    // Apply mergeSort on right list
    node right = mergeSort(nextofmiddle);

    // Merge the left and right lists
    node sortedlist = sortedMerge(left, right);
    return sortedlist;
}

// Utility function to get the middle of the linked list
public static node getMiddle(node head)
{
    if (head == null)
        return head;

    node slow = head, fast = head;

    while (fast.next != null && fast.next.next != null) {
        slow = slow.next;
        fast = fast.next.next;
    }
}
```

```
void push(int new_data)
{
    /* allocate node */
    node new_node = new node(new_data);

    /* link the old list off the new node */
    new_node.next = head;

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

// Utility function to print the linked list
void printList(node headref)
{
    while (headref != null) {
        System.out.print(headref.val + " ");
        headref = headref.next;
    }
}

public static void main(String[] args)
{
    linkedList li = new linkedList();
    /*
     * Let us create a unsorted linked list to test the functions
     * created. The list shall be a: 2->3->20->5->10->15
     */
    li.push(15);
    li.push(10);
    li.push(5);
    li.push(20);
    li.push(3);
    li.push(2);

    // Apply merge Sort
    li.head = li.mergeSort(li.head);
    System.out.print("\n Sorted Linked List is: \n");
    li.printList(li.head);
}
```

Python3

```
# Python3 program to merge sort of linked list

# create Node using class Node.
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

class LinkedList:
    def __init__(self):
        self.head = None

    # push new value to linked list
    # using append method
    def append(self, new_value):

        # Allocate new node
        new_node = Node(new_value)

        # if head is None, initialize it to new node
        if self.head is None:
            self.head = new_node
            return
        curr_node = self.head
        while curr_node.next is not None:
            curr_node = curr_node.next

        # Append the new node at the end
        # of the linked list
        curr_node.next = new_node

    def sortedMerge(self, a, b):
        result = None

        # Base cases
        if a == None:
            return b
        if b == None:
            return a
```

```
        result = a
        result.next = self.sortedMerge(a.next, b)
    else:
        result = b
        result.next = self.sortedMerge(a, b.next)
    return result

def mergeSort(self, h):

    # Base case if head is None
    if h == None or h.next == None:
        return h

    # get the middle of the list
    middle = self.getMiddle(h)
    nexttomiddle = middle.next

    # set the next of middle node to None
    middle.next = None

    # Apply mergeSort on left list
    left = self.mergeSort(h)

    # Apply mergeSort on right list
    right = self.mergeSort(nexttomiddle)

    # Merge the left and right lists
    sortedlist = self.sortedMerge(left, right)
    return sortedlist

# Utility function to get the middle
# of the linked list
def getMiddle(self, head):
    if (head == None):
        return head

    slow = head
    fast = head

    while (fast.next != None and
           fast.next.next != None):
        slow = slow.next
```

```
# Utility function to print the linked list
def printList(head):
    if head is None:
        print(' ')
        return
    curr_node = head
    while curr_node:
        print(curr_node.data, end = " ")
        curr_node = curr_node.next
    print(' ')

# Driver Code
if __name__ == '__main__':
    li = LinkedList()

    # Let us create a unsorted linked list
    # to test the functions created.
    # The list shall be a: 2->3->20->5->10->15
    li.append(15)
    li.append(10)
    li.append(5)
    li.append(20)
    li.append(3)
    li.append(2)

    # Apply merge Sort
    li.head = li.mergeSort(li.head)
    print ("Sorted Linked List is:")
    printList(li.head)

# This code is contributed by Vikas Chitturi
```

C#

```
// C# program to illustrate merge sorted
// of linkedList
using System;

public class linkedList {
    node head = null;
```

```
public int val;
public node next;

public node(int val)
{
    this.val = val;
}
}

node sortedMerge(node a, node b)
{
    node result = null;
    /* Base cases */
    if (a == null)
        return b;
    if (b == null)
        return a;

    /* Pick either a or b, and recur */
    if (a.val <= b.val) {
        result = a;
        result.next = sortedMerge(a.next, b);
    }
    else {
        result = b;
        result.next = sortedMerge(a, b.next);
    }
    return result;
}

node mergeSort(node h)
{
    // Base case : if head is null
    if (h == null || h.next == null) {
        return h;
    }

    // get the middle of the list
    node middle = getMiddle(h);
    node nextofmiddle = middle.next;

    // set the next of middle node to null
```

```
node left = mergeSort(h);

// Apply mergeSort on right list
node right = mergeSort(nextofmiddle);

// Merge the left and right lists
node sortedlist = sortedMerge(left, right);
return sortedlist;
}

// Utility function to get the
// middle of the linked list
node getMiddle(node h)
{
    // Base case
    if (h == null)
        return h;
    node fastptr = h.next;
    node slowptr = h;

    // Move fastptr by two and slow ptr by one
    // Finally slowptr will point to middle node
    while (fastptr != null) {
        fastptr = fastptr.next;
        if (fastptr != null) {
            slowptr = slowptr.next;
            fastptr = fastptr.next;
        }
    }
    return slowptr;
}

void push(int new_data)
{
    /* allocate node */
    node new_node = new node(new_data);

    /* link the old list off the new node */
    new_node.next = head;

    /* move the head to point to the new node */
    head = new_node;
}
```



```
void printList(node headref)
{
    while (headref != null) {
        Console.Write(headref.val + " ");
        headref = headref.next;
    }
}

// Driver code
public static void Main(String[] args)
{
    linkedList li = new linkedList();
    /*
    * Let us create a unsorted linked list to test the functions
    * created. The list shall be a: 2->3->20->5->10->15
    */
    li.push(15);
    li.push(10);
    li.push(5);
    li.push(20);
    li.push(3);
    li.push(2);

    // Apply merge Sort
    li.head = li.mergeSort(li.head);
    Console.WriteLine("\n Sorted Linked List is: \n");
    li.printList(li.head);
}

// This code is contributed by Arnab Kundu
```

Javascript

```
<script>

// Javascript program to
// illustrate merge sorted
// of linkedList
```

```
// node a, b;
class node {
    constructor(val) {
        this.val = val;
        this.next = null;
    }
}

function sortedMerge( a, b)
{
    var result = null;
    /* Base cases */
    if (a == null)
        return b;
    if (b == null)
        return a;

    /* Pick either a or b, and recur */
    if (a.val <= b.val) {
        result = a;
        result.next = sortedMerge(a.next, b);
    } else {
        result = b;
        result.next = sortedMerge(a, b.next);
    }
    return result;
}

function mergeSort( h) {
    // Base case : if head is null
    if (h == null || h.next == null) {
        return h;
    }

    // get the middle of the list
    var middle = getMiddle(h);
    var nextofmiddle = middle.next;

    // set the next of middle node to null
    middle.next = null;

    // Apply mergeSort on left list
```

```
    var right = mergeSort(nextofmiddle);

    // Merge the left and right lists
    var sortedlist = sortedMerge(left, right);
    return sortedlist;
}

// Utility function to get the middle
// of the linked list
function getMiddle( head) {
    if (head == null)
        return head;

    var slow = head, fast = head;

    while (fast.next != null && fast.next.next != null)
    {
        slow = slow.next;
        fast = fast.next.next;
    }
    return slow;
}

function push(new_data) {
    /* allocate node */
    var new_node = new node(new_data);

    /* link the old list off the new node */
    new_node.next = head;

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

// Utility function to print the linked list
function printList( headref) {
    while (headref != null) {
        document.write(headref.val + " ");
        headref = headref.next;
    }
}
```

```
    Let us create a unsorted linked
    list to test the functions
    created. The list shall be
    a: 2->3->20->5->10->15
    */
    push(15);
    push(10);
    push(5);
    push(20);
    push(3);
    push(2);

    // Apply merge Sort
    head = mergeSort(head);
    document.write("\n Sorted Linked List is: \n");
    printList(head);

    // This code contributed by umadevi9616
```

Output:

```
Sorted Linked List is:
2 3 5 10 15 20
```

Time Complexity: $O(n \cdot \log n)$

Auxiliary Space: $O(n)$

Approach 2: This approach is simpler and uses $\log n$ space.

mergeSort():

1. If the size of the linked list is 1 then return the head
2. Find mid using The Tortoise and The Hare Approach
3. Store the next of mid in head2 i.e. the right sub-linked list.
4. Now Make the next midpoint null.

6. Call `merge()` given the arguments new heads of left and right sub-linked lists and store the final head returned after merging.
7. Return the final head of the merged linkedlist.

`merge(head1, head2):`

1. Take a pointer say `merged` to store the merged list in it and store a dummy node in it.
2. Take a pointer `temp` and assign `merge` to it.
3. If the data of `head1` is less than the data of `head2`, then, store `head1` in next of `temp` & move `head1` to the next of `head1`.
4. Else store `head2` in next of `temp` & move `head2` to the next of `head2`.
5. Move `temp` to the next of `temp`.
6. Repeat steps 3, 4 & 5 until `head1` is not equal to null and `head2` is not equal to null.
7. Now add any remaining nodes of the first or the second linked list to the merged linked list.
8. Return the next of `merged` (that will ignore the dummy and return the head of the final merged linked list)

C++

```
#include <iostream>
using namespace std;

// Node structure
struct Node {
    int data;
    Node* next;
};

// function to insert in list
void insert(int x, Node** head)
{
```

```
        (*head)->next = NULL;
        return;
    }
    Node* temp = new Node;
    temp->data = (*head)->data;
    temp->next = (*head)->next;
    (*head)->data = x;
    (*head)->next = temp;
}

// function to print the list
void print(Node* head)
{
    Node* temp = head;
    while (temp != NULL) {
        cout << temp->data << " ";
        temp = temp->next;
    }
}

Node* merge(Node* firstNode, Node* secondNode)
{
    Node* merged = new Node;
    Node* temp = new Node;

    // merged is equal to temp so in the end we have the top
    // Node.
    merged = temp;

    // while either firstNode or secondNode becomes NULL
    while (firstNode != NULL && secondNode != NULL) {

        if (firstNode->data <= secondNode->data) {
            temp->next = firstNode;
            firstNode = firstNode->next;
        }

        else {
            temp->next = secondNode;
            secondNode = secondNode->next;
        }

        temp = temp->next;
    }
}
```

```
// inserted in the temp List
while (firstNode != NULL) {
    temp->next = firstNode;
    firstNode = firstNode->next;
    temp = temp->next;
}

while (secondNode != NULL) {
    temp->next = secondNode;
    secondNode = secondNode->next;
    temp = temp->next;
}
// return the head of the sorted list
return merged->next;
}

// function to calculate the middle Element
Node* middle(Node* head)
{
    Node* slow = head;
    Node* fast = head->next;

    while (!slow->next && (!fast && !fast->next)) {
        slow = slow->next;
        fast = fast->next->next;
    }
    return slow;
}

// function to sort the given list
Node* sort(Node* head)
{
    if (head->next == NULL) {
        return head;
    }

    Node* mid = new Node;
    Node* head2 = new Node;
    mid = middle(head);
    head2 = mid->next;
    mid->next = NULL;
```

```

        return finalhead;
    }
    int main(void)
    {
        Node* head = NULL;
        int n[] = { 7, 10, 5, 20, 3, 2 };
        for (int i = 0; i < 6; i++) {
            insert(n[i], &head); // inserting in the list
        }
        cout << "Sorted Linked List is: \n";
        print(sort(head)); // printing the sorted list returned
                           // by sort()

        return 0;
    }

    // This code is contributed by Aditya Kumar (adityakumar120)

```

C

```

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

// Node structure
typedef struct Node {
    int data;
    struct Node* next;
} Node;

// function to insert in list
void insert(int x, Node** head)
{
    if (*head == NULL) {
        *head = (Node*)malloc(sizeof(Node));
        (*head)->data = x;
        (*head)->next = NULL;
        return;
    }
    Node* temp = (Node*)malloc(sizeof(Node));
    temp->data = (*head)->data;
    temp->next = (*head)->next;
    (*head)->data = x;
    (*head)->next = temp;
}

```



```
void print(Node* head)
{
    Node* temp = head;
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
}

Node* merge(Node* firstNode, Node* secondNode)
{
    Node* merged = (Node*)malloc(sizeof(Node));
    Node* temp = (Node*)malloc(sizeof(Node));

    // merged is equal to temp so in the end we have the top
    // Node.
    merged = temp;

    // while either firstNode or secondNode becomes NULL
    while (firstNode != NULL && secondNode != NULL) {

        if (firstNode->data <= secondNode->data) {
            temp->next = firstNode;
            firstNode = firstNode->next;
        }

        else {
            temp->next = secondNode;
            secondNode = secondNode->next;
        }
        temp = temp->next;
    }

    // any remaining Node in firstNode or secondNode gets
    // inserted in the temp List
    while (firstNode != NULL) {
        temp->next = firstNode;
        firstNode = firstNode->next;
        temp = temp->next;
    }

    while (secondNode != NULL) {
```



```
    }
    // return the head of the sorted list
    return merged->next;
}

// function to calculate the middle Element
Node* middle(Node* head)
{
    Node* slow = head;
    Node* fast = head->next;

    while (!slow->next && (!fast && !fast->next)) {
        slow = slow->next;
        fast = fast->next->next;
    }
    return slow;
}

// function to sort the given list
Node* sort(Node* head)
{
    if (head->next == NULL) {
        return head;
    }

    Node* mid = (Node*)malloc(sizeof(Node));
    Node* head2 = (Node*)malloc(sizeof(Node));
    mid = middle(head);
    head2 = mid->next;
    mid->next = NULL;
    // recursive call to sort() hence diving our problem,
    // and then merging the solution
    Node* finalhead = merge(sort(head), sort(head2));
    return finalhead;
}

int main(void)
{
    Node* head = NULL;
    int n[] = { 7, 10, 5, 20, 3, 2 };
    for (int i = 0; i < 6; i++) {
```

```
        print(sort(head)); // printing the sorted list returned
                           // by sort()
    return 0;
}
```

// This code is contributed by Aditya Kumar (adityakumar129)

Java

```
// Java program for the above approach
import java.io.*;
import java.lang.*;
import java.util.*;

// Node Class
class Node {
    int data;
    Node next;
    Node(int key)
    {
        this.data = key;
        next = null;
    }
}

class GFG {

    // Function to merge sort
    static Node mergeSort(Node head)
    {
        if (head.next == null)
            return head;

        Node mid = findMid(head);
        Node head2 = mid.next;
        mid.next = null;
        Node newHead1 = mergeSort(head);
        Node newHead2 = mergeSort(head2);
        Node finalHead = merge(newHead1, newHead2);

        return finalHead;
    }
}
```

```
static Node merge(Node head1, Node head2)
{
    Node merged = new Node(-1);
    Node temp = merged;

    // While head1 is not null and head2
    // is not null
    while (head1 != null && head2 != null) {
        if (head1.data < head2.data) {
            temp.next = head1;
            head1 = head1.next;
        }
        else {
            temp.next = head2;
            head2 = head2.next;
        }
        temp = temp.next;
    }

    // While head1 is not null
    while (head1 != null) {
        temp.next = head1;
        head1 = head1.next;
        temp = temp.next;
    }

    // While head2 is not null
    while (head2 != null) {
        temp.next = head2;
        head2 = head2.next;
        temp = temp.next;
    }
    return merged.next;
}

// Find mid using The Tortoise and The Hare approach
static Node findMid(Node head)
{
    Node slow = head, fast = head.next;
    while (fast != null && fast.next != null) {
        slow = slow.next;
        fast = fast.next.next;
    }
}
```

```
// Function to print list
static void printList(Node head)
{
    while (head != null) {
        System.out.print(head.data + " ");
        head = head.next;
    }
}

// Driver Code
public static void main(String[] args)
{
    Node head = new Node(7);
    Node temp = head;
    temp.next = new Node(10);
    temp = temp.next;
    temp.next = new Node(5);
    temp = temp.next;
    temp.next = new Node(20);
    temp = temp.next;
    temp.next = new Node(3);
    temp = temp.next;
    temp.next = new Node(2);
    temp = temp.next;

    // Apply merge Sort
    head = mergeSort(head);
    System.out.print("\nSorted Linked List is: \n");
    printList(head);
}
}
```

Python3

Python program for the above approach

Node Class

class Node:

```
# Function to merge sort
def mergeSort(head):
    if (head.next == None):
        return head

    mid = findMid(head)
    head2 = mid.next
    mid.next = None
    newHead1 = mergeSort(head)
    newHead2 = mergeSort(head2)
    finalHead = merge(newHead1, newHead2)
    return finalHead

# Function to merge two linked lists
def merge(head1, head2):
    merged = Node(-1)

    temp = merged
    # While head1 is not null and head2
    # is not null
    while (head1 != None and head2 != None):
        if (head1.data < head2.data):
            temp.next = head1
            head1 = head1.next
        else:
            temp.next = head2
            head2 = head2.next
        temp = temp.next

    # While head1 is not null
    while (head1 != None):
        temp.next = head1
        head1 = head1.next
        temp = temp.next

    # While head2 is not null
    while (head2 != None):
        temp.next = head2
        head2 = head2.next
        temp = temp.next

    return merged.next
```

```
slow = head
fast = head.next
while (fast != None and fast.next != None):
    slow = slow.next
    fast = fast.next.next
return slow

# Function to print list
def printList(head):
    while (head != None):
        print(head.data,end=" ")
        head=head.next

# Driver Code
head = Node(7)
temp = head
temp.next = Node(10);
temp = temp.next;
temp.next = Node(5);
temp = temp.next;
temp.next = Node(20);
temp = temp.next;
temp.next = Node(3);
temp = temp.next;
temp.next = Node(2);
temp = temp.next;

# Apply merge Sort
head = mergeSort(head);
print("\nSorted Linked List is: \n");

printList(head);

# This code is contributed by avanitrachhadiya2155
```

C#

```
// C# program for the above approach
using System;
```

```
// Node Class
```

```
    public Node next;

    public Node(int key)
    {
        this.data = key;
        next = null;
    }
}

class GFG{

// Function to merge sort
static Node mergeSort(Node head)
{
    if (head.next == null)
        return head;

    Node mid = findMid(head);
    Node head2 = mid.next;
    mid.next = null;
    Node newHead1 = mergeSort(head);
    Node newHead2 = mergeSort(head2);
    Node finalHead = merge(newHead1, newHead2);

    return finalHead;
}

// Function to merge two linked lists
static Node merge(Node head1, Node head2)
{
    Node merged = new Node(-1);
    Node temp = merged;

    // While head1 is not null and head2
    // is not null
    while (head1 != null && head2 != null)
    {
        if (head1.data < head2.data)
        {
            temp.next = head1;
            head1 = head1.next;
        }
    }
}
```



```
        head2 = head2.next;
    }
    temp = temp.next;
}

// While head1 is not null
while (head1 != null)
{
    temp.next = head1;
    head1 = head1.next;
    temp = temp.next;
}

// While head2 is not null
while (head2 != null)
{
    temp.next = head2;
    head2 = head2.next;
    temp = temp.next;
}
return merged.next;
}

// Find mid using The Tortoise and The Hare approach
static Node findMid(Node head)
{
    Node slow = head, fast = head.next;
    while (fast != null && fast.next != null)
    {
        slow = slow.next;
        fast = fast.next.next;
    }
    return slow;
}

// Function to print list
static void printList(Node head)
{
    while (head != null)
    {
        Console.Write(head.data + " ");
        head = head.next;
    }
}
```

```
// Driver Code
public static void Main(String[] args)
{
    Node head = new Node(7);
    Node temp = head;
    temp.next = new Node(10);
    temp = temp.next;
    temp.next = new Node(5);
    temp = temp.next;
    temp.next = new Node(20);
    temp = temp.next;
    temp.next = new Node(3);
    temp = temp.next;
    temp.next = new Node(2);
    temp = temp.next;

    // Apply merge Sort
    head = mergeSort(head);
    Console.WriteLine("\nSorted Linked List is: \n");
    printList(head);
}
}
```

// This code is contributed by umadevi9616

Javascript

```
<script>

// JavaScript program for the above approach

// Node Class
class Node {
    constructor(val) {
        this.data = val;
        this.next = null;
    }
}

// Function to merge sort
function mergeSort(head) {
    if (head.next == null)
```

```
var head2 = mid.next;
    mid.next = null;
var newHead1 = mergeSort(head);
var newHead2 = mergeSort(head2);
var finalHead = merge(newHead1, newHead2);

    return finalHead;
}

// Function to merge two linked lists
function merge(head1, head2) {
var merged = new Node(-1);
var temp = merged;

    // While head1 is not null and head2
    // is not null
    while (head1 != null && head2 != null) {
        if (head1.data < head2.data) {
            temp.next = head1;
            head1 = head1.next;
        } else {
            temp.next = head2;
            head2 = head2.next;
        }
        temp = temp.next;
    }

    // While head1 is not null
    while (head1 != null) {
        temp.next = head1;
        head1 = head1.next;
        temp = temp.next;
    }

    // While head2 is not null
    while (head2 != null) {
        temp.next = head2;
        head2 = head2.next;
        temp = temp.next;
    }
    return merged.next;
}
```

```
var slow = head, fast = head.next;
    while (fast != null && fast.next != null) {
        slow = slow.next;
        fast = fast.next.next;
    }
    return slow;
}

// Function to print list
function printList(head) {
    while (head != null) {
        document.write(head.data + " ");
        head = head.next;
    }
}

// Driver Code

var head = new Node(7);
var temp = head;
temp.next = new Node(10);
temp = temp.next;
temp.next = new Node(5);
temp = temp.next;
temp.next = new Node(20);
temp = temp.next;
temp.next = new Node(3);
temp = temp.next;
temp.next = new Node(2);
temp = temp.next;

// Apply merge Sort
head = mergeSort(head);
document.write("Sorted Linked List is: <br/>");
printList(head);

// This code contributed by gauravrajput1

</script>
```

Output:

Sorted Linked List is:

2 3 5 7 10 20

Time Complexity: $O(n \cdot \log n)$

Auxiliary Space: $O(n)$

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.

MICROSOFT TEST SERIES
To help crack your SDE interview

Enrol Now



Like 273

Previous

Union and Intersection of two
Linked Lists

Next

Adding two polynomials
using Linked List

- 01** Why Quick Sort preferred for Arrays and Merge Sort for Linked Lists?
16, May 15
- 02** Difference of two Linked Lists using Merge sort
07, Apr 20
- 03** C++ Program For Merge Sort Of Linked Lists
06, Nov 21
- 04** Python Program For Merge Sort Of Linked Lists
06, Nov 21
- 05** Javascript Program For Merge Sort Of Linked Lists
06, Nov 21
- 06** Merge Sort for Linked Lists in JavaScript
23, May 18
- 07** Union and Intersection of two linked lists | Set-2 (Using Merge Sort)
14, Jun 17
- 08** Merge Sort with $O(1)$ extra space merge and $O(n \lg n)$ time [Unsigned Integers Only]
06, Aug 18

Article Contributed By :



Vote for difficulty

Current difficulty : [Hard](#)

[Easy](#)[Normal](#)[Medium](#)[Hard](#)[Expert](#)

Improved By : [rathbhupendra](#), [andrew1234](#), [NithinSridhar](#), [nidhi_biet](#),
[Vikas Chitturi](#), [mittulmandhan](#), [umadevi9616](#),
[GauravRajput1](#), [avanitrachhadiya2155](#), [dpatwal8](#),
[rajsng3737](#), [simranarora5sos](#), [surindertarika1234](#),
[2020csb088prince](#), [adityakumar129](#)

Article Tags : [Accolite](#), [Adobe](#), [Linked-List-Sorting](#), [MAQ Software](#),
[Merge Sort](#), [Microsoft](#), [Paytm](#), [Veritas](#), [Linked List](#), [Sorting](#)

Practice Tags : [Paytm](#), [Accolite](#), [Microsoft](#), [MAQ Software](#), [Adobe](#), [Veritas](#),
[Linked List](#), [Sorting](#), [Merge Sort](#)

[Improve Article](#)[Report Issue](#)

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

[Load Comments](#)



A-143, 9th Floor, Sovereign Corporate Tower,
Sector-136, Noida, Uttar Pradesh - 201305

feedback@geeksforgeeks.org

Company

[About Us](#)
[Careers](#)
[In Media](#)
[Contact Us](#)
[Privacy Policy](#)
[Copyright Policy](#)

News

[Top News](#)
[Technology](#)
[Work & Career](#)
[Business](#)
[Finance](#)
[Lifestyle](#)
[Knowledge](#)

Web Development

[Web Tutorials](#)

Learn

[Algorithms](#)
[Data Structures](#)
[SDE Cheat Sheet](#)
[Machine learning](#)
[CS Subjects](#)
[Video Tutorials](#)
[Courses](#)

Languages

[Python](#)
[Java](#)
[CPP](#)
[Golang](#)
[C#](#)
[SQL](#)
[Kotlin](#)

Contribute

[Write an Article](#)



Simpler tools lead to happier developers. And
happier developers lead to better results

GET \$100 FREE CREDIT

[HIDE AD](#) • [AD VIA BUYSSELLADS](#)

NodeJS

@geeksforgeeks , Some rights reserved

Do Not Sell My Personal Information