Merge two sorted linked lists - GeeksforGeeks

Source: https://www.geeksforgeeks.org/merge-two-sorted-linked-lists/

Courses Tutorials Practice Jobs DSA Tutorial Interview Questions Quizzes Must Do Advanced DSA System Design Aptitude Puzzles Interview Corner DSA Python Technical Scripter 2026 Explore DSA Fundamentals Logic Building Problems Analysis of Algorithms Data Structures Array Data Structure String in Data Structure Hashing in Data Structure Linked List Data Structure Stack Data Structure Queue Data Structure Tree Data Structure Graph Data Structure Trie Data Structure Algorithms Searching Algorithms Sorting Algorithms Introduction to Recursion Greedy Algorithms Tutorial Graph Algorithms Dynamic Programming or DP Bitwise Algorithms Advanced Segment Tree Binary Indexed Tree or Fenwick Tree Square Root (Sqrt) Decomposition Algorithm Binary Lifting Geometry Interview Preparation Interview Corner GfG160 Practice Problem GeeksforGeeks Practice - Leading Online Coding Platform Problem of The Day - Develop the Habit of Coding DSA Course 90% Refund Merge two sorted linked lists Last Updated : 30 Aug, 2025 Given the heads of two sorted linked lists, merge them into a single sorted linked list and return the head of the merged list. Examples: Input: Output: 2 -> 3 -> 5 -> 10 -> 15 -> 20 -> 40 Explanation: Merging two sorted lists [5, 10, 15, 40] and [2, 3, 20] in order gives 2 -> 3 -> 5 -> 10 -> 15 -> 20 -> 40. Input: Output: 1 -> 1 -> 2 -> 4 Explanation: Merging [1 ,1] and [2, 4] in order gives 1 -> 1 -> 2 -> 4. Try it on GfG Practice Table of Content [Naive Approach] By Using Array - O((n+m) × log(n+m)) Time and O(n+m) Space [Better Approach] Using Recursive Merge - O(n+m) Time and O(n+m) Space [Efficient Approach] Using Iterative Merge - O(n+m) Time and O(1) Space [Naive Approach] By Using Array - O((n+m) × log(n+m)) Time and O(n+m) Space The idea is to use an array to store all the node data from both linked lists, sort the array, and then construct the resultant sorted linked list from the array elements. C++ #include <iostream> #include <vector> #include <algorithm> using namespace std ; class Node { public : int data ; Node * next ; Node (int x) { data = x ; next = nullptr ; } }; Node * sortedMerge (Node * head1 , Node * head2) { vector < int > arr ; // pushing the values of the first linked list while (head1 != nullptr) { arr . push_back (head1 -> data); head1 = head1 -> next ; } // pushing the values of the second linked list while (head2 != nullptr) { arr . push_back (head2 -> data); head2 = head2 -> next ; } // sorting the vector sort (arr . begin (), arr . end ()); // creating a new list with sorted values Node * dummy = new Node (-1); Node * curr = dummy ; for (int i = 0 ; i < arr . size (); i ++) { curr -> next = new Node (arr [i]); curr = curr -> next ; } return dummy -> next ; } void printList (Node * curr) { while (curr != nullptr) { cout << curr -> data ; if (curr -> next != nullptr) cout << " -> " ; curr = curr -> next ; } cout << endl ; } int main () { Node * head1 = new Node (5); head1 -> next = new Node (10); head1 -> next -> next = new Node (15); head1 -> next -> next -> next = new Node (40); Node * head2 = new Node (2); head2 -> next = new Node (3); head2 -> next -> next = new Node (20); Node * res = sortedMerge (head1 , head2); printList (res); return 0 ; } Java import java.util.ArrayList ; import java.util.Collections ; class Node { int data ; Node next ; Node (int x) { data = x ; next = null ; } } class GfG { static Node sortedMerge (Node head1 , Node head2) { ArrayList < Integer > arr = new ArrayList <> (); // pushing the values of the first linked list while (head1 != null) { arr . add (head1 . data); head1 = head1 . next ; } // pushing the values of the second linked list while (head2 != null) { arr . add (head2 . data); head2 = head2 . next ; } // sorting the list Collections . sort (arr); // creating a new list with sorted values Node dummy = new Node (- 1); Node curr = dummy ; for (int i = 0 ; i < arr . size (); i ++) { curr . next = new Node (arr . get (i)); curr = curr . next ; } return dummy . next ; } static void printList (Node curr) { while (curr != null) { System . out . print (curr . data); if (curr . next != null) { System . out . print (" -> "); } curr = curr . next ; } System . out . println (); } public static void main (String [] args) { Node head1 = new Node (5); head1 . next = new Node (10); head1 . next . next = new Node (15); head1 . next . next . next = new Node (40); Node head2 = new Node (2); head2 . next = new Node (3); head2 . next . next = new Node (20); Node res = sortedMerge (head1 , head2); printList (res); } } Python class Node : def __init__ (self , x): self . data = x self . next = None def sortedMerge (head1 , head2): arr = [] # pushing the values of the first linked list while head1 is not None : arr . append (head1 . data) head1 = head1 . next # pushing the values of the second linked list while head2 is not None : arr . append (head2 . data) head2 = head2 . next # sorting the list arr . sort () # creating a new list with sorted values dummy = Node (- 1) curr = dummy for value in arr : curr . next = Node (value) curr = curr . next return dummy . next def printList (node): while node is not None : print (f " { node . data } " , end = "") if node . next is not None : print (" -> " , end = "") node = node . next print () if __name__ == "__main__" : head1 = Node (5) head1 . next = Node (10) head1 . next . next = Node (15) head1 . next . next . next = Node (40) head2 = Node (2) head2 . next = Node (3) head2 . next . next = Node (20) res = sortedMerge (head1 , head2) printList (res) C# using System ; using System.Collections.Generic ; class Node { public int data ; public Node next ; public Node (int x) { data = x ; next = null ; } } class GfG { static Node sortedMerge (Node head1 , Node head2) { List < int > arr = new List < int > (); // pushing the values of the first linked list while (head1 != null) { arr . Add (head1 . data); head1 = head1 . next ; } // pushing the values of the second linked list while (head2 != null) { arr . Add (head2 . data); head2 = head2 . next ; } // sorting the list arr . Sort (); // creating a new list with sorted values Node dummy = new Node (- 1); Node curr = dummy ; foreach (int value in arr) { curr . next = new Node (value); curr = curr . next ; } return dummy . next ; } static void printList (Node curr) { while (curr != null) { Console . Write (curr . data); if (curr . next != null) { Console . Write (" -> "); } curr = curr . next ; } Console . WriteLine (); } static void Main (string [] args) { Node head1 = new Node (5); head1 . next = new Node (10); head1 . next . next = new Node (15); head1 . next . next . next = new Node (40); Node head2 = new Node (2); head2 . next = new Node (3); head2 . next . next = new Node (20); Node res = sortedMerge (head1 , head2); printList (res); } } JavaScript class Node { constructor (x) { this . data = x ; this . next = null ; } } function sortedMerge (head1 , head2) { let arr = []; // pushing the values of the first linked list while (head1 !== null) { arr . push (head1 . data); head1 = head1 . next ; } // pushing the values of the second linked list while (head2 !== null) { arr . push (head2 . data); head2 = head2 . next ; } // sorting the array arr . sort ((x , y) => x - y); // creating a new list with sorted values let dummy = new Node (- 1); let curr = dummy ; for (let value of arr) { curr . next = new Node (value); curr = curr . next ; } return dummy . next ; } function printList (node) { while (node !== null) { process . stdout . write (node . data . toString ()); if (node . next !== null) { process . stdout . write (" -> "); } node = node . next ; } } let head1 = new Node (5); head1 . next = new Node (10); head1 . next . next = new Node (15); head1 . next . next . next = new Node (40); let head2 = new Node (2); head2 . next = new Node (3); head2 . next . next = new Node (20); let res = sortedMerge (head1 , head2); printList (res); Output 2 -> 3 -> 5 -> 10 -> 15 -> 20 -> 40 [Better Approach] Using Recursive Merge - O(n+m) Time and O(n+m) Space The idea is to pick the smaller head node at each step and let recursion merge the remaining parts. if one list is empty, return the other; otherwise the smaller node becomes the next node in the merged list and its next is the recursive merge of the rest. Algorithm: If head1 is null, return head2. If head2 is null, return head1. Compare head1.data and head2.data. If head1.data <= head2.data: => set head = head1 => set head.next = merge(head1.next, head2) Else: => set head = head2 => set head.next = merge(head1, head2.next) Return head. C++ #include <iostream> using namespace std ; class Node { public : int data ; Node * next ; Node (int x) { data = x ; next = nullptr ; } }; Node * sortedMerge (Node * head1 , Node * head2) { // base cases if (head1 == nullptr) return head2 ; if (head2 == nullptr) return head1 ; // recursive merging based on smaller value if (head1 -> data <= head2 -> data) { head1 -> next = sortedMerge (head1 -> next , head2); return head1 ; } else { head2 -> next = sortedMerge (head1 , head2 -> next); return head2 ; } } void printList (Node * curr) { while (curr != nullptr) { cout << curr -> data ; if (curr -> next != nullptr) cout << " -> " ; curr = curr -> next ; } cout << endl ; } int main () { Node * head1 = new Node (5); head1 -> next = new Node (10); head1 -> next -> next = new Node (15); head1 -> next -> next -> next = new Node (40); Node * head2 = new Node (2); head2 -> next = new Node (3); head2 -> next -> next = new Node (20); Node * res = sortedMerge (head1 , head2); printList (res); return 0 ; } C #include <stdio.h> #include <stdlib.h> struct Node { int data ; struct Node * next ; }; struct Node * sortedMerge (struct Node * head1 , struct Node * head2) { // base cases if (head1 == NULL) return head2 ; if (head2 == NULL) return head1 ; // recursive merging based on smaller value if (head1 -> data <= head2 -> data) { head1 -> next = sortedMerge (head1 -> next , head2); return head1 ; } else { head2 -> next = sortedMerge (head1 , head2 -> next); return head2 ; } } void printList (struct Node * curr) { while (curr != NULL) { printf ("%d" , curr -> data); if (curr -> next != NULL) { printf (" -> "); } curr = curr -> next ; } printf (" \n "); } struct Node * createNode (int data) { struct Node * newNode = (struct Node *) malloc (sizeof (struct Node)); newNode -> data = data ; newNode -> next = NULL ; return newNode ; } int main () { struct Node * head1 = createNode (5); head1 -> next = createNode (10); head1 -> next -> next = createNode (15); head1 -> next -> next -> next = createNode (40); struct Node * head2 = createNode (2); head2 -> next = createNode (3); head2 -> next -> next = createNode (20); struct Node * res = sortedMerge (head1 , head2); printList (res); return 0 ; } Java class Node { int data ; Node next ; Node (int x) { data = x ; next = null ; } } class GfG { static Node sortedMerge (Node head1 , Node head2) { // base cases if (head1 == null) return head2 ; if (head2 == null) return head1 ; // recursive merging based on smaller value if (head1 . data

<= head2 . data) { head1 . next = sortedMerge (head1 . next , head2); return head1 ; } else { head2 . next = sortedMerge (head1 , head2 . next); return head2 ; } } static void printList (Node curr) { while (curr != null) { System . out . print (curr . data); if (curr . next != null) System . out . print (" -> "); curr = curr . next ; } System . out . println (); } public static void main (String [] args) { Node head1 = new Node (5); head1 . next = new Node (10); head1 . next . next = new Node (15); head1 . next . next . next = new Node (40); Node head2 = new Node (2); head2 . next = new Node (3); head2 . next . next = new Node (20); Node res = sortedMerge (head1 , head2); printList (res); } } Python class Node : def __init__ (self , x): self . data = x self . next = None def sortedMerge (head1 , head2): # base cases if head1 is None : return head2 if head2 is None : return head1 # recursive merging based on smaller value if head1 . data <= head2 . data : head1 . next = sortedMerge (head1 . next , head2) return head1 else : head2 . next = sortedMerge (head1 , head2 . next) return head2 def printList (node): while node is not None : print (f " { node . data } " , end = "") if node . next is not None : print (" -> " , end = "") node = node . next print () if __name__ == "__main__" : head1 = Node (5) head1 . next = Node (10) head1 . next . next = Node (15) head1 . next . next . next = Node (40) head2 = Node (2) head2 . next = Node (3) head2 . next . next = Node (20) res = sortedMerge (head1 , head2) printList (res) C# using System ; class Node { public int data ; public Node next ; public Node (int x) { data = x ; next = null ; } } class GfG { static Node sortedMerge (Node head1 , Node head2) { // base cases if (head1 == null) return head2 ; if (head2 == null) return head1 ; // recursive merging based on smaller value if (head1 . data <= head2 . data) { head1 . next = sortedMerge (head1 . next , head2); return head1 ; } else { head2 . next = sortedMerge (head1 , head2 . next); return head2 ; } } static void printList (Node curr) { while (curr != null) { Console . Write (curr . data); if (curr . next != null) Console . Write (" -> "); curr = curr . next ; } Console . WriteLine (); } static void Main (string [] args) { Node head1 = new Node (5); head1 . next = new Node (10); head1 . next . next = new Node (15); head1 . next . next . next = new Node (40); Node head2 = new Node (2); head2 . next = new Node (3); head2 . next . next = new Node (20); Node res = sortedMerge (head1 , head2); printList (res); } } JavaScript class Node { constructor (x) { this . data = x ; this . next = null ; } } function sortedMerge (head1 , head2) { // base cases if (head1 === null) return head2 ; if (head2 === null) return head1 ; // recursive merging based on smaller value if (head1 . data <= head2 . data) { head1 . next = sortedMerge (head1 . next , head2); return head1 ; } else { head2 . next = sortedMerge (head1 , head2 . next); return head2 ; } } function printList (node) { while (node !== null) { process . stdout . write (node . data . toString ()); if (node . next !== null) { process . stdout . write (" -> "); } node = node . next ; } } // Driver Code let head1 = new Node (5); head1 . next = new Node (10); head1 . next . next = new Node (15); head1 . next . next . next = new Node (40); let head2 = new Node (2); head2 . next = new Node (3); head2 . next . next = new Node (20); let res = sortedMerge (head1 , head2); printList (res); Output 2 -> 3 -> 5 -> 10 -> 15 -> 20 -> 40 [Efficient Approach] Using Iterative Merge - O(n+m) Time and O(1) Space The idea is to iteratively merge two sorted linked lists using a dummy node to simplify the process. A current pointer tracks the last node of the merged list. We compare the nodes from both lists and append the smaller node to the merged list. Once one list is fully traversed, the remaining nodes from the other list are appended. The merged list is returned starting from the node after the dummy node. Working: C++ #include <iostream> using namespace std ; class Node { public : int data ; Node * next ; Node (int x) { data = x ; next = nullptr ; } }; Node * sortedMerge (Node * head1 , Node * head2) { // create a dummy node to simplify // the merging process Node * dummy = new Node (-1); Node * curr = dummy ; // iterate through both linked lists while (head1 != nullptr && head2 != nullptr) { // add the smaller node to the merged list if (head1 -> data <= head2 -> data) { curr -> next = head1 ; head1 = head1 -> next ; } else { curr -> next = head2 ; head2 = head2 -> next ; } curr = curr -> next ; } // if any list is left, append it to // the merged list if (head1 != nullptr) { curr -> next = head1 ; } else { curr -> next = head2 ; } // return the merged list starting // from the next of dummy node return dummy -> next ; } void printList (Node * head) { while (head != nullptr) { cout << head -> data ; if (head -> next != nullptr) cout << " -> " ; head = head -> next ; } cout << endl ; } int main () { Node * head1 = new Node (5); head1 -> next = new Node (10); head1 -> next -> next = new Node (15); head1 -> next -> next -> next = new Node (40); Node * head2 = new Node (2); head2 -> next = new Node (3); head2 -> next -> next = new Node (20); Node * res = sortedMerge (head1 , head2); printList (res); return 0 ; } C #include <stdio.h> #include <stdlib.h> struct Node { int data ; struct Node * next ; }; struct Node * createNode (int data); struct Node * sortedMerge (struct Node * head1 , struct Node * head2) { // create a dummy node to simplify // the merging process struct Node * dummy = createNode (-1); struct Node * curr = dummy ; // iterate through both linked lists while (head1 != NULL && head2 != NULL) { // add the smaller node to the merged list if (head1 -> data <= head2 -> data) {

curr -> next = head1 ; head1 = head1 -> next ; } else { curr -> next = head2 ; head2 = head2 -> next ; } curr = curr -> next ; } // if any list is left, append it to // the merged list if (head1 != NULL) { curr -> next = head1 ; } else { curr -> next = head2 ; } // return the merged list starting // from the next of dummy node return dummy -> next ; } void printList (struct Node * head) { while (head != NULL) { printf ("%d" , head -> data); if (head -> next != NULL) { printf (" -> "); } head = head -> next ; } printf (" \n "); } struct Node * createNode (int data) { struct Node * newNode = (struct Node *) malloc (sizeof (struct Node)); newNode -> data = data ; newNode -> next = NULL ; return newNode ; } int main () { struct Node * head1 = createNode (5); head1 -> next = createNode (10); head1 -> next -> next = createNode (15); head1 -> next -> next -> next = createNode (40); struct Node * head2 = createNode (2); head2 -> next = createNode (3); head2 -> next -> next = createNode (20); struct Node * res = sortedMerge (head1 , head2); printList (res); return 0 ; } Java class Node { int data ; Node next ; Node (int x) { data = x ; next = null ; } } class GfG { static Node sortedMerge (Node head1 , Node head2) { // create a dummy node to simplify // the merging process Node dummy = new Node (- 1); Node curr = dummy ; // iterate through both linked lists while (head1 != null && head2 != null) { // add the smaller node to the merged list if (head1 . data <= head2 . data) { curr . next = head1 ; head1 = head1 . next ; } else { curr . next = head2 ; head2 = head2 . next ; } curr = curr . next ; } // if any list is left, append it to // the merged list if (head1 != null) { curr . next = head1 ; } else { curr . next = head2 ; } // return the merged list starting from // the next of dummy node return dummy . next ; } static void printList (Node head) { while (head != null) { System . out . print (head . data); if (head . next != null) System . out . print (" -> "); head = head . next ; } System . out . println (); } public static void main (String [] args) { Node head1 = new Node (5); head1 . next = new Node (10); head1 . next . next = new Node (15); head1 . next . next . next = new Node (40); Node head2 = new Node (2); head2 . next = new Node (3); head2 . next . next = new Node (20); Node res = sortedMerge (head1 , head2); printList (res); } } Python class Node : def __init__ (self , x): self . data = x self . next = None def sortedMerge (head1 , head2): # create a dummy node to simplify # the merging process dummy = Node (- 1) curr = dummy # iterate through both linked lists while head1 is not None and head2 is not None : # add the smaller node to the merged list if head1 . data <= head2 . data : curr . next = head1 head1 = head1 . next else : curr . next = head2 head2 = head2 . next curr = curr . next # if any list is left, append it to the merged list if head1 is not None : curr . next = head1 else : curr . next = head2 # return the merged list starting from # the next of dummy node return dummy . next def printList (head): while head is not None : if head . next is not None : print (head . data , end = " -> ") else : print (head . data) head = head . next if __name__ == "__main__" : head1 = Node (5) head1 . next = Node (10) head1 . next . next = Node (15) head1 . next . next . next = Node (40) head2 = Node (2) head2 . next = Node (3) head2 . next . next = Node (20) res = sortedMerge (head1 , head2) printList (res) C# using System ; class Node { public int data ; public Node next ; public Node (int x) { data = x ; next = null ; } } class GfG { static Node sortedMerge (Node head1 , Node head2) { // create a dummy node to simplify the // merging process Node dummy = new Node (- 1); Node curr = dummy ; // iterate through both linked lists while (head1 != null && head2 != null) { // add the smaller node to the merged list if (head1 . data <= head2 . data) { curr . next = head1 ; head1 = head1 . next ; } else { curr . next = head2 ; head2 = head2 . next ; } curr = curr . next ; } // if any list is left, append it to the // merged list if (head1 != null) { curr . next = head1 ; } else { curr . next = head2 ; } // return the merged list starting from // the next of dummy node return dummy . next ; } static void printList (Node head) { while (head != null) { Console . Write (head . data); if (head . next != null) Console . Write (" -> "); head = head . next ; } Console . WriteLine (); } static void Main (string [] args) { Node head1 = new Node (5); head1 . next = new Node (10); head1 . next . next = new Node (15); head1 . next . next . next = new Node (40); Node head2 = new Node (2); head2 . next = new Node (3); head2 . next . next = new Node (20); Node res = sortedMerge (head1 , head2); printList (res); } } JavaScript class Node { constructor (x) { this . data = x ; this . next = null ; } } function sortedMerge (head1 , head2) { // create a dummy node to simplify the merging process let dummy = new Node (- 1); let curr = dummy ; // iterate through both linked lists while (head1 !== null && head2 !== null) { // add the smaller node to the merged list if (head1 . data <= head2 . data) { curr . next = head1 ; head1 = head1 . next ; } else { curr . next = head2 ; head2 = head2 . next ; } curr = curr . next ; } // if any list is left, append it // to the merged list if (head1 !== null) { curr . next = head1 ; } else { curr . next = head2 ; } // return the merged list starting from // the next of dummy node return dummy . next ; } function printList (head) { let result = "" ; while (head !== null) { result += head . data ; if (head . next !== null) { result += " -> " ; } head = head . next ; } console . log (result); } // Driver code let head1 = new Node (5); head1 . next = new Node (10); head1 . next . next = new Node (15); head1 . next . next . next = new Node (40); let head2 = new Node (2); head2

. next = new Node (3); head2 . next . next = new Node (20); let res = sortedMerge (head1 , head2); printList (res); Output 2 -> 3 -> 5 -> 10 -> 15 -> 20 -> 40 Comment Article Tags: Article Tags: Linked List DSA Microsoft Amazon Oracle Flipkart Samsung Belzabar Accolite Zoho MakeMyTrip Synopsys Brocade FactSet OATS Systems Merge Sort + 12 More