Sort a linked list of 0s, 1s and 2s - GeeksforGeeks

Source: https://www.geeksforgeeks.org/sort-a-linked-list-of-0s-1s-or-2s/

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 Sort a linked list of 0s, 1s and 2s Last Updated : 27 Aug, 2024 Given a linked list of 0s, 1s and 2s , The task is to sort the list in non-decreasing order. Examples : Input: 1 -> 1 -> 2 -> 0 -> 2 -> 0 -> 1 -> NULL Output: 0 -> 0 -> 1 -> 1 -> 1 -> 2 -> 2 -> NULL Input: 1 -> 1 -> 2 -> 1 -> 0 -> NULL Output: 0 -> 1 -> 1 -> 1 -> 2 -> NULL [Expected Approach - 1] By Maintaining Frequency - O(n) Time and O(1) Space: The idea is to traverse the linked List and count the number of nodes having values 0, 1 and 2 and store them in an array of size 3, say cnt[] such that cnt[0] = count of nodes with value 0 cnt[1] = count of nodes with value 1 cnt[2] = count of nodes with value 2 Now, traverse the linked list again to fill the first cnt[0] nodes with 0 , then next cnt[1] nodes with 1 and finally cnt[2] nodes with 2 . Below is the illustration of above approach : Below is the implementation of the above approach. C++ // C++ Program to sort a linked list of 0s, 1s or 2s #include <bits/stdc++.h> using namespace std ; class Node { public : int data ; Node * next ; Node (int new_data) { data = new_data ; next = nullptr ; } }; // Function to sort a linked list of 0s, 1s and 2s void sortList (Node * head) { // Initialize count of '0', '1' and '2' as 0 int cnt [3] = { 0 , 0 , 0 }; Node * ptr = head ; // Traverse and count total number of '0', '1' and '2' // cnt[0] will store total number of '0's // cnt[1] will store total number of '1's // cnt[2] will store total number of '2's while (ptr != NULL) { cnt [ptr -> data] += 1 ; ptr = ptr -> next ; } int idx = 0 ; ptr = head ; // Fill first cnt[0] nodes with value 0 // Fill next cnt[1] nodes with value 1 // Fill remaining cnt[2] nodes with value 2 while (ptr != nullptr) { if (cnt [idx] == 0) idx += 1 ; else { ptr -> data = idx ; cnt [idx] -= 1 ; ptr = ptr -> next ; } } } void printList (Node * node) { while (node != nullptr) { cout << " " << node -> data ; node = node -> next ; } cout << " \n " ; } int main () { // Create a hard-coded linked list: // 1 -> 1 -> 2 -> 1 -> 0 -> NULL Node * head = new Node (1); head -> next = new Node (1); head -> next -> next = new Node (2); head -> next -> next -> next = new Node (1); head -> next -> next -> next -> next = new Node (0); cout << "Linked List before Sorting:" ; printList (head); sortList (head); cout << "Linked List after Sorting:" ; printList (head); return 0 ; } C // C Program to sort a linked list of 0s, 1s or 2s #include <stdio.h> struct Node { int data ; struct Node * next ; }; // Function to sort a linked list of 0s, 1s and 2s void sortList (struct Node * head) { // Initialize count of '0', '1' and '2' as 0 int cnt [3] = { 0 , 0 , 0 }; struct Node * ptr = head ; // Traverse and count total number of '0', '1' and '2' // cnt[0] will store total number of '0's // cnt[1] will store total number of '1's // cnt[2] will store total number of '2's while (ptr != NULL) { cnt [ptr -> data] += 1 ; ptr = ptr -> next ; } int idx = 0 ; ptr = head ; // Fill first cnt[0] nodes with value 0 // Fill next cnt[1] nodes with value 1 // Fill remaining cnt[2] nodes with value 2 while (ptr != NULL) { if (cnt [idx] == 0) { idx += 1 ; } else { ptr -> data = idx ; cnt [idx] -= 1 ; ptr = ptr -> next ; } } } void printList (struct Node * node) { while (node != NULL) { printf (" %d" , node -> data); node = node -> next ; } printf (" \n "); } struct Node * createNode (int new_data) { struct Node * new_node = (struct Node *) malloc (sizeof (struct Node)); new_node -> data = new_data ; new_node -> next = NULL ; return new_node ; } int main () { // Create a hard-coded linked list: // 1 -> 1 -> 2 -> 1 -> 0 -> NULL struct Node * head = createNode (1); head -> next = createNode (1); head -> next -> next = createNode (2); head -> next -> next -> next = createNode (1); head -> next -> next -> next -> next = createNode (0); printf ("Linked List before Sorting:"); printList (head); sortList (head); printf ("Linked List after Sorting:"); printList (head); return 0 ; } Java // Java Program to sort a linked list of 0s, 1s or 2s class Node { int data ; Node next ; Node (int new_data) { data = new_data ; next = null ; } } // Function to sort a linked list of 0s, 1s and 2s class GfG { static void sortList (Node head) { // Initialize count of '0', '1' and '2' as 0 int [] cnt = { 0 , 0 , 0 }; Node ptr = head ; // Traverse and count total number of '0', '1' and '2' // cnt[0] will

store total number of '0's // cnt[1] will store total number of '1's // cnt[2] will store total number of '2's while (ptr != null) { cnt [ptr . data] += 1 ; ptr = ptr . next ; } int idx = 0 ; ptr = head ; // Fill first cnt[0] nodes with value 0 // Fill next cnt[1] nodes with value 1 // Fill remaining cnt[2] nodes with value 2 while (ptr != null) { if (cnt [idx] == 0) idx += 1 ; else { ptr . data = idx ; cnt [idx] -= 1 ; ptr = ptr . next ; } } } static void printList (Node node) { while (node != null) { System . out . print (" " + node . data); node = node . next ; } System . out . println (); } public static void main (String [] args) { // Create a hard-coded linked list: // 1 -> 1 -> 2 -> 1 -> 0 -> NULL Node head = new Node (1); head . next = new Node (1); head . next . next = new Node (2); head . next . next . next = new Node (1); head . next . next . next . next = new Node (0); System . out . print ("Linked List before Sorting:"); printList (head); sortList (head); System . out . print ("Linked List after Sorting:"); printList (head); } } Python # Python Program to sort a linked list of 0s, 1s or 2s class Node : def __init__ (self , new_data): self . data = new_data self . next = None # Function to sort a linked list of 0s, 1s and 2s def sort_list (head): # Initialize count of '0', '1' and '2' as 0 cnt = [0 , 0 , 0] ptr = head # Traverse and count total number of '0', '1' and '2' # cnt[0] will store total number of '0's # cnt[1] will store total number of '1's # cnt[2] will store total number of '2's while ptr is not None : cnt [ptr . data] += 1 ptr = ptr . next idx = 0 ptr = head # Fill first cnt[0] nodes with value 0 # Fill next cnt[1] nodes with value 1 # Fill remaining cnt[2] nodes with value 2 while ptr is not None : if cnt [idx] == 0 : idx += 1 else : ptr . data = idx cnt [idx] -= 1 ptr = ptr . next def print_list (node): while node is not None : print (f " { node . data } " , end = " ") node = node . next print (" \n ") if __name__ == "__main__" : # Create a hard-coded linked list: # 1 -> 1 -> 2 -> 1 -> 0 -> NULL head = Node (1) head . next = Node (1) head . next . next = Node (2) head . next . next . next = Node (1) head . next . next . next . next = Node (0) print ("Linked List before Sorting:" , end = " ") print_list (head) sort_list (head) print ("Linked List after Sorting:" , end = " ") print_list (head) C# // C# Program to sort a linked list of 0s, 1s or 2s using System ; class Node { public int Data ; public Node Next ; public Node (int newData) { Data = newData ; Next = null ; } } class GfG { // Function to sort a linked list of 0s, 1s, and 2s static void SortList (Node head) { // Initialize count of '0', '1', and '2' as 0 int [] cnt = { 0 , 0 , 0 }; Node ptr = head ; // Traverse and count total number of '0', '1', and '2' // cnt[0] will store total number of '0's // cnt[1] will store total number of '1's // cnt[2] will store total number of '2's while (ptr != null) { cnt [ptr . Data] += 1 ; ptr = ptr . Next ; } int idx = 0 ; ptr = head ; // Fill first cnt[0] nodes with value 0 // Fill next cnt[1] nodes with value 1 // Fill remaining cnt[2] nodes with value 2 while (ptr != null) { if (cnt [idx] == 0) idx += 1 ; else { ptr . Data = idx ; cnt [idx] -= 1 ; ptr = ptr . Next ; } } } static void PrintList (Node node) { while (node != null) { Console . Write (" " + node . Data); node = node . Next ; } Console . WriteLine (); } static void Main () { // Create a hard-coded linked list: // 1 -> 1 -> 2 -> 1 -> 0 -> NULL Node head = new Node (1); head . Next = new Node (1); head . Next . Next = new Node (2); head . Next . Next . Next = new Node (1); head . Next . Next . Next . Next = new Node (0); Console . Write ("Linked List before Sorting:"); PrintList (head); SortList (head); Console . Write ("Linked List after Sorting:"); PrintList (head); } } JavaScript // JavaScript Program to sort a linked list of 0s, 1s or 2s class Node { constructor (newData) { this . data = newData ; this . next = null ; } } // Function to sort a linked list of 0s, 1s and 2s function sortList (head) { // Initialize count of '0', '1' and '2' as 0 const cnt = [0 , 0 , 0]; let ptr = head ; // Traverse and count total number of '0', '1' and '2' // cnt[0] will store total number of '0's // cnt[1] will store total number of '1's // cnt[2] will store total number of '2's while (ptr !== null) { cnt [ptr . data] += 1 ; ptr = ptr . next ; } let idx = 0 ; ptr = head ; // Fill first cnt[0] nodes with value 0 // Fill next cnt[1] nodes with value 1 // Fill remaining cnt[2] nodes with value 2 while (ptr !== null) { if (cnt [idx] === 0) { idx += 1 ; } else { ptr . data = idx ; cnt [idx] -= 1 ; ptr = ptr . next ; } } } function printList (node) { while (node !== null) { console . log (node . data); node = node . next ; } console . log (); } // Create a hard-coded linked list: // 1 -> 1 -> 2 -> 1 -> 0 -> NULL let head = new Node (1); head . next = new Node (1); head . next . next = new Node (2); head . next . next . next = new Node (1); head . next . next . next . next = new Node (0); console . log ("Linked List before Sorting:"); printList (head); sortList (head); console . log ("Linked List after Sorting:"); printList (head); Output Linked List before Sorting: 1 1 2 1 0 Linked List after Sorting: 0 1 1 1 2 Time Complexity: O(n) where n is the number of nodes in the linked list. Auxiliary Space: O(1) [Expected Approach - 2] By Updating Links of Nodes - O(n) Time and O(1) Space: The idea is to maintain 3 pointers named zero , one and two to point to current ending nodes of linked lists containing 0, 1, and 2 respectively. For every traversed node, we attach it to the end of its corresponding list. If the current node's value is 0, append it after pointer zero and move pointer zero to current node . If the current node's value is 1, append it after pointer one and move pointer one to current node . If the current node's value is 2, append it after pointer two and move pointer two to current node . Finally, we link all three lists. To avoid many null checks, we use three dummy pointers zeroD , oneD and twoD that work

as dummy headers of three lists. Below is the implementation of the above approach: C++ // C++ Program to sort a linked list 0s, 1s // or 2s by updating links #include <bits/stdc++.h> using namespace std ; class Node { public : int data ; Node * next ; Node (int new_data) { data = new_data ; next = nullptr ; } }; // Sort a linked list of 0s, 1s, and 2s by changing pointers Node * sortList (Node * head) { if (! head || ! (head -> next)) return head ; // Create three dummy nodes to point to the beginning of // three linked lists. These dummy nodes are created to // avoid null checks. Node * zeroD = new Node (0); Node * oneD = new Node (0); Node * twoD = new Node (0); // Initialize current pointers for three lists Node * zero = zeroD , * one = oneD , * two = twoD ; // Traverse the list Node * curr = head ; while (curr != NULL) { if (curr -> data == 0) { // If the data of the current node is 0, // append it to pointer zero and update zero zero -> next = curr ; zero = zero -> next ; } else if (curr -> data == 1) { // If the data of the current node is 1, // append it to pointer one and update one one -> next = curr ; one = one -> next ; } else { // If the data of the current node is 2, // append it to pointer two and update two two -> next = curr ; two = two -> next ; } curr = curr -> next ; } // Combine the three lists if (oneD -> next) zero -> next = oneD -> next ; else zero -> next = twoD -> next ; one -> next = twoD -> next ; two -> next = nullptr ; // Updated head head = zeroD -> next ; return head ; } void printList (Node * node) { while (node != nullptr) { cout << " " << node -> data ; node = node -> next ; } cout << " \n " ; } int main () { // Create a hard-coded linked list: // 1 -> 1 -> 2 -> 1 -> 0 -> NULL Node * head = new Node (1); head -> next = new Node (1); head -> next -> next = new Node (2); head -> next -> next -> next = new Node (1); head -> next -> next -> next -> next = new Node (0); cout << "Linked List before Sorting:" ; printList (head); head = sortList (head); cout << "Linked List after Sorting:" ; printList (head); return 0 ; } C // C Program to sort a linked list 0s, 1s // or 2s by updating links #include <stdio.h> struct Node { int data ; struct Node * next ; }; // Function to create a new node struct Node * createNode (int new_data); // Sort a linked list of 0s, 1s and 2s // by changing pointers struct Node * sortList (struct Node * head) { if (! head || ! (head -> next)) return head ; // Create three dummy nodes to point to beginning of // three linked lists. These dummy nodes are created to // avoid null checks. struct Node * zeroD = createNode (0); struct Node * oneD = createNode (0); struct Node * twoD = createNode (0); // Initialize current pointers for three // lists struct Node * zero = zeroD , * one = oneD , * two = twoD ; // Traverse list struct Node * curr = head ; while (curr) { if (curr -> data == 0) { // If the data of current node is 0, // append it to pointer zero and update zero zero -> next = curr ; zero = zero -> next ; } else if (curr -> data == 1) { // If the data of current node is 1, // append it to pointer one and update one one -> next = curr ; one = one -> next ; } else { // If the data of current node is 2, // append it to pointer two and update two two -> next = curr ; two = two -> next ; } curr = curr -> next ; } // Combine the three lists zero -> next = (oneD -> next) ? (oneD -> next) : (twoD -> next); one -> next = twoD -> next ; two -> next = NULL ; // Updated head head = zeroD -> next ; return head ; } void printList (struct Node * node) { while (node != NULL) { printf (" %d" , node -> data); node = node -> next ; } printf (" \n "); } struct Node * createNode (int new_data) { struct Node * new_node = (struct Node *) malloc (sizeof (struct Node)); new_node -> data = new_data ; new_node -> next = NULL ; return new_node ; } int main () { // Create a hard-coded linked list: // 1 -> 1 -> 2 -> 1 -> 0 -> NULL struct Node * head = createNode (1); head -> next = createNode (1); head -> next -> next = createNode (2); head -> next -> next -> next = createNode (1); head -> next -> next -> next -> next = createNode (0); printf ("Linked List before Sorting:"); printList (head); head = sortList (head); printf ("Linked List after Sorting:"); printList (head); return 0 ; } Java // Java Program to sort a linked list of 0s, 1s // or 2s by updating links class Node { int data ; Node next ; Node (int new_data) { data = new_data ; next = null ; } } class GfG { // Sort a linked list of 0s, 1s and 2s // by changing pointers static Node sortList (Node head) { if (head == null || head . next == null) return head ; // Create three dummy nodes to point to beginning of // three linked lists. These dummy nodes are created to // avoid null checks. Node zeroD = new Node (0); Node oneD = new Node (0); Node twoD = new Node (0); // Initialize current pointers for three // lists Node zero = zeroD , one = oneD , two = twoD ; // Traverse list Node curr = head ; while (curr != null) { if (curr . data == 0) { // If the data of current node is 0, // append it to pointer zero and update zero zero . next = curr ; zero = zero . next ; } else if (curr . data == 1) { // If the data of current node is 1, // append it to pointer one and update one one . next = curr ; one = one . next ; } else { // If the data of current node is 2, // append it to pointer two and update two two . next = curr ; two = two . next ; } curr = curr . next ; } // Combine the three lists zero . next = (oneD . next != null) ? (oneD . next) : (twoD . next); one . next = twoD . next ; two . next = null ; // Updated head head = zeroD . next ; return head ; } static void printList (Node node) { while (node != null) { System . out . print (" " + node . data); node = node . next ; } System . out . println (); } public static void main (String [] args) { // Create a hard-coded linked list: // 1 -> 1 -> 2 -> 1 -> 0 -> NULL Node head = new Node (1); head . next = new Node (1); head . next . next = new Node (2);

head . next . next . next = new Node (1); head . next . next . next . next = new Node (0); System . out . print ("Linked List before Sorting:"); printList (head); head = sortList (head); System . out . print ("Linked List after Sorting:"); printList (head); } } Python # Python Program to sort a linked list 0s, 1s # or 2s by updating links class Node : def __init__ (self , new_data): self . data = new_data self . next = None # Sort a linked list of 0s, 1s and 2s # by changing pointers def sortList (head): if not head or not head . next : return head # Create three dummy nodes to point to beginning of # three linked lists. These dummy nodes are created to # avoid null checks. zeroD = Node (0) oneD = Node (0) twoD = Node (0) # Initialize current pointers for three # lists zero = zeroD one = oneD two = twoD # Traverse list curr = head while curr : if curr . data == 0 : # If the data of current node is 0, # append it to pointer zero and update zero zero . next = curr zero = zero . next elif curr . data == 1 : # If the data of cu # append it to pointer one and update one one . next = curr one = one . next else : # If the data of current node is 2, # append it to pointer two and update two two . next = curr two = two . next curr = curr . next # Combine the three lists zero . next = oneD . next if oneD . next else twoD . next one . next = twoD . next two . next = None # Updated head head = zeroD . next return head def printList (node): while node is not None : print (node . data , end = ' ') node = node . next print () if __name__ == "__main__" : # Create a hard-coded linked list: # 1 -> 1 -> 2 -> 1 -> 0 -> NULL head = Node (1) head . next = Node (1) head . next . next = Node (2) head . next . next . next = Node (1) head . next . next . next . next = Node (0) print ("Linked List before Sorting: " , end = ") printList (head) head = sortList (head) print ("Linked List after Sorting: " , end = ") printList (head) C# // C# Program to sort a linked list 0s, 1s // or 2s by updating links using System ; public class Node { public int Data ; public Node Next ; public Node (int newData) { Data = newData ; Next = null ; } } // Sort a linked list of 0s, 1s and 2s // by changing pointers class GfG { static Node SortList (Node head) { if (head == null || head . Next == null) return head ; // Create three dummy nodes to point to beginning of // three linked lists. These dummy nodes are created to // avoid null checks. Node zeroD = new Node (0); Node oneD = new Node (0); Node twoD = new Node (0); // Initialize current pointers for three // lists Node zero = zeroD , one = oneD , two = twoD ; // Traverse list Node curr = head ; while (curr != null) { if (curr . Data == 0) { // If the data of current node is 0, // append it to pointer zero and update zero zero . Next = curr ; zero = zero . Next ; } else if (curr . Data == 1) { // If the data of current node is 1, // append it to pointer one and update one one . Next = curr ; one = one . Next ; } else { // If the data of current node is 2, // append it to pointer two and update two two . Next = curr ; two = two . Next ; } curr = curr . Next ; } // Combine the three lists zero . Next = (oneD . Next != null) ? (oneD . Next) : (twoD . Next); one . Next = twoD . Next ; two . Next = null ; // Updated head head = zeroD . Next ; // Delete dummy nodes // In C# garbage collection will handle this return head ; } // This function prints the contents // of the linked list starting from the head static void PrintList (Node node) { while (node != null) { Console . Write (" " + node . Data); node = node . Next ; } Console . WriteLine (); } static void Main () { // Create a hard-coded linked list: // 1 -> 1 -> 2 -> 1 -> 0 -> NULL Node head = new Node (1); head . Next = new Node (1); head . Next . Next = new Node (2); head . Next . Next . Next = new Node (1); head . Next . Next . Next . Next = new Node (0); Console . Write ("Linked List before Sorting:"); PrintList (head); head = SortList (head); Console . Write ("Linked List after Sorting:"); PrintList (head); } } JavaScript // Javascript program to sort a linked list of 0s, 1s // or 2s by updating links class Node { constructor (newData) { this . data = newData ; this . next = null ; } } // Sort a linked list of 0s, 1s and 2s // by changing pointers function sortList (head) { if (! head || ! head . next) return head ; // Create three dummy nodes to point to beginning of // three linked lists. These dummy nodes are created to // avoid null checks. let zeroD = new Node (0); let oneD = new Node (0); let twoD = new Node (0); // Initialize current pointers for three // lists let zero = zeroD , one = oneD , two = twoD ; // Traverse list let curr = head ; while (curr) { if (curr . data === 0) { // If the data of current node is 0, // append it to pointer zero and update zero zero . next = curr ; zero = zero . next ; } else if (curr . data === 1) { // If the data of current node is 1, // append it to pointer one and update one one . next = curr ; one = one . next ; } else { // If the data of current node is 2, // append it to pointer two and update two two . next = curr ; two = two . next ; } curr = curr . next ; } // Combine the three lists zero . next = (oneD . next) ? (oneD . next) : (twoD . next); one . next = twoD . next ; two . next = null ; // Updated head head = zeroD . next ; return head ; } function printList (node) { while (node !== null) { console . log (node . data); node = node . next ; } console . log (); } // Create a hard-coded linked list: // 1 -> 1 -> 2 -> 1 -> 0 -> NULL let head = new Node (1); head . next = new Node (1); head . next . next = new Node (2); head . next . next . next = new Node (1); head . next . next . next . next = new Node (0); console . log ("Linked List before Sorting:"); printList (head); head = sortList (head); console . log ("Linked List after Sorting:"); printList (head); Output Linked List before Sorting: 1 1 2 1 0 Linked List after Sorting: 0 1 1 1 2 Time Complexity: O(n),

where n is the number of nodes in the linked list. Auxiliary Space: O(1) Comment Article Tags: Article Tags: Linked List Sorting DSA Microsoft Amazon MakeMyTrip Linked-List-Sorting + 3 More