Intersection of two Sorted Linked Lists - GeeksforGeeks

Source: https://www.geeksforgeeks.org/intersection-of-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 Intersection of two Sorted Linked Lists Last Updated : 23 Jul, 2025 Given two lists sorted in increasing order, create and return a new list representing the intersection of the two lists. The new list should be made with its own memory — the original lists should not be changed. Example: Input: First linked list: 1->2->3->4->6 Second linked list be 2->4->6->8, Output: 2->4->6. The elements 2, 4, 6 are common in both the list so they appear in the intersection list. Input: First linked list: 1->2->3->4->5 Second linked list be 2->3->4, Output: 2->3->4 The elements 2, 3, 4 are common in both the list so they appear in the intersection list. Recommended PracticeIntersection of two sorted Linked listsTry It! Method 1 : Using Dummy Node. Approach: The idea is to use a temporary dummy node at the start of the result list. The pointer tail always points to the last node in the result list, so new nodes can be added easily. The dummy node initially gives the tail a memory space to point to. This dummy node is efficient, since it is only temporary, and it is allocated in the stack. The loop proceeds, removing one node from either 'a' or 'b' and adding it to the tail. When the given lists are traversed the result is in dummy. next, as the values are allocated from next node of the dummy. If both the elements are equal then remove both and insert the element to the tail. Else remove the smaller element among both the lists. Below is the implementation of the above approach: Try it on GfG Practice C++ #include <bits/stdc++.h> using namespace std ; /* Link list node */ struct Node { int data ; Node * next ; }; void push (Node ** head_ref , int new_data); /*This solution uses the temporary dummy to build up the result list */ Node * sortedIntersect (Node * a , Node * b) { Node dummy ; Node * tail = & dummy ; dummy . next = NULL ; /* Once one or the other list runs out -- we're done */ while (a != NULL && b != NULL) { if (a -> data == b -> data) { push ((& tail -> next), a -> data); tail = tail -> next ; a = a -> next ; b = b -> next ; } /* advance the smaller list */ else if (a -> data < b -> data) a = a -> next ; else b = b -> next ; } return (dummy . next); } /* UTILITY FUNCTIONS */ /* 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 = (Node *) malloc (sizeof (Node)); /* put in the data */ new_node -> data = new_data ; /* link the old list of the new node */ new_node -> next = (* head_ref); /* move the head to point to the new node */ (* head_ref) = new_node ; } /* Function to print nodes in a given linked list */ void printList (Node * node) { while (node != NULL) { cout << node -> data << " " ; node = node -> next ; } } /* Driver program to test above functions*/ int main () { /* Start with the empty lists */ Node * a = NULL ; Node * b = NULL ; Node * intersect = NULL ; /* Let us create the first sorted linked list to test the functions Created linked list will be 1->2->3->4->5->6 */ push (& a , 6); push (& a , 5); push (& a , 4); push (& a , 3); push (& a , 2); push (& a , 1); /* Let us create the second sorted linked list Created linked list will be 2->4->6->8 */ push (& b , 8); push (& b , 6); push (& b , 4); push (& b , 2); /* Find the intersection two linked lists */ intersect = sortedIntersect (a , b); cout << "Linked list containing common items of a & b \n " ; printList (intersect); } C #include <stdio.h> #include <stdlib.h> /* Link list node */ struct Node { int data ; struct Node * next ; }; void push (struct Node ** head_ref , int new_data); /*This solution uses the temporary dummy to build up the result list */ struct Node * sortedIntersect (struct Node * a , struct Node * b) { struct Node dummy ; struct Node * tail = & dummy ; dummy . next = NULL ; /* Once one or the other list runs out -- we're done */ while (a != NULL && b != NULL) { if (a -> data == b -> data) { push ((& tail -> next), a -> data); tail = tail -> next ; a = a -> next ; b = b -> next ; } /* advance the smaller list */ else if (

a -> data < b -> data) a = a -> next ; else b = b -> next ; } return (dummy . next); } /* UTILITY FUNCTIONS */ /* 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 of the new node */ new_node -> next = (* head_ref); /* move the head to point to the new node */ (* head_ref) = new_node ; } /* Function to print nodes in a given linked list */ void printList (struct Node * node) { while (node != NULL) { printf ("%d " , node -> data); node = node -> next ; } } /* Driver program to test above functions*/ int main () { /* Start with the empty lists */ struct Node * a = NULL ; struct Node * b = NULL ; struct Node * intersect = NULL ; /* Let us create the first sorted linked list to test the functions Created linked list will be 1->2->3->4->5->6 */ push (& a , 6); push (& a , 5); push (& a , 4); push (& a , 3); push (& a , 2); push (& a , 1); /* Let us create the second sorted linked list Created linked list will be 2->4->6->8 */ push (& b , 8); push (& b , 6); push (& b , 4); push (& b , 2); /* Find the intersection two linked lists */ intersect = sortedIntersect (a , b); printf (" \n Linked list containing common items of a & b \n "); printList (intersect); getchar (); } Java import java.util.* ; import java.io.* ; public class GFG { // head nodes for pointing to 1st and 2nd linked lists static Node a = null , b = null ; // dummy node for storing intersection static Node dummy = null ; // tail node for keeping track of // last node so that it makes easy for insertion static Node tail = null ; // class - Node static class Node { int data ; Node next ; Node (int data) { this . data = data ; next = null ; } } // function for printing the list void printList (Node start) { Node p = start ; while (p != null) { System . out . print (p . data + " "); p = p . next ; } System . out . println (); } // inserting elements into list void push (int data) { Node temp = new Node (data); if (dummy == null) { dummy = temp ; tail = temp ; } else { tail . next = temp ; tail = temp ; } } // function for finding intersection and adding it to dummy list void sortedIntersect () { // pointers for iterating Node p = a , q = b ; while (p != null && q != null) { if (p . data == q . data) { // add to dummy list push (p . data); p = p . next ; q = q . next ; } else if (p . data < q . data) p = p . next ; else q = q . next ; } } // Driver code public static void main (String args []) { GFG list = new GFG (); // creating first linked list list . a = new Node (1); list . a . next = new Node (2); list . a . next . next = new Node (3); list . a . next . next . next = new Node (4); list . a . next . next . next . next = new Node (6); // creating second linked list list . b = new Node (2); list . b . next = new Node (4); list . b . next . next = new Node (6); list . b . next . next . next = new Node (8); // function call for intersection list . sortedIntersect (); // print required intersection System . out . println ("Linked list containing common items of a & b"); list . printList (dummy); } } // This code is contributed by Likhita AVL Python3 ''' Link list node ''' class Node : def __init__ (self): self . data = 0 self . next = None '''This solution uses the temporary dummy to build up the result list ''' def sortedIntersect (a , b): dummy = Node () tail = dummy ; dummy . next = None ; ''' Once one or the other list runs out -- we're done ''' while (a != None and b != None): if (a . data == b . data): tail . next = push ((tail . next), a . data); tail = tail . next ; a = a . next ; b = b . next ; # advance the smaller list elif (a . data < b . data): a = a . next ; else : b = b . next ; return (dummy . next); ''' UTILITY FUNCTIONS ''' ''' Function to insert a node at the beginning of the linked list ''' def push (head_ref , new_data): ''' allocate node ''' new_node = Node () ''' put in the data ''' new_node . data = new_data ; ''' link the old list of the new node ''' new_node . next = (head_ref); ''' move the head to point to the new node ''' (head_ref) = new_node ; return head_ref ''' Function to print nodes in a given linked list ''' def printList (node): while (node != None): print (node . data , end = ' ') node = node . next ; ''' Driver code''' if __name__ == '__main__' : ''' Start with the empty lists ''' a = None ; b = None ; intersect = None ; ''' Let us create the first sorted linked list to test the functions Created linked list will be 1.2.3.4.5.6 ''' a = push (a , 6); a = push (a , 5); a = push (a , 4); a = push (a , 3); a = push (a , 2); a = push (a , 1); ''' Let us create the second sorted linked list Created linked list will be 2.4.6.8 ''' b = push (b , 8); b = push (b , 6); b = push (b , 4); b = push (b , 2); ''' Find the intersection two linked lists ''' intersect = sortedIntersect (a , b); print ("Linked list containing common items of a & b "); printList (intersect); # This code is contributed by rutvik_56. C# using System ; public class GFG { // dummy node for storing intersection static Node dummy = null ; // tail node for keeping track of // last node so that it makes easy for insertion static Node tail = null ; // class - Node public class Node { public int data ; public Node next ; public Node (int data) { this . data = data ; next = null ; } } // head nodes for pointing to 1st and 2nd linked lists Node a = null , b = null ; // function for printing the list void printList (Node start) { Node p = start ; while (p != null) { Console . Write (p . data + " "); p = p . next ; } Console . WriteLine (); } // inserting elements into list void push (int data) { Node temp = new Node (data); if (dummy == null) { dummy = temp ; tail = temp ; } else { tail . next = temp ; tail = temp ; } } // function for finding intersection and adding it to dummy list void sortedIntersect () { // pointers for iterating Node p = a , q = b ; while (p != null && q != null) { if (p . data == q . data) { // add to dummy list push (p . data);

p = p . next ; q = q . next ; } else if (p . data < q . data) p = p . next ; else q = q . next ; } } // Driver code public static void Main (String [] args) { GFG list = new GFG (); // creating first linked list list . a = new Node (1); list . a . next = new Node (2); list . a . next . next = new Node (3); list . a . next . next . next = new Node (4); list . a . next . next . next . next = new Node (6); // creating second linked list list . b = new Node (2); list . b . next = new Node (4); list . b . next . next = new Node (6); list . b . next . next . next = new Node (8); // function call for intersection list . sortedIntersect (); // print required intersection Console . WriteLine ("Linked list containing common items of a & b"); list . printList (dummy); } } // This code is contributed by aashish1995 JavaScript < script > // head nodes for pointing to // 1st and 2nd linked lists var a = null , b = null ; // dummy node for storing intersection var dummy = null ; // tail node for keeping track of // last node so that it makes easy for insertion var tail = null ; // class - Node class Node { constructor (val) { this . data = val ; this . next = null ; } } // function for printing the list function printList (start) { var p = start ; while (p != null) { document . write (p . data + " "); p = p . next ; } document . write (); } // inserting elements into list function push (data) { var temp = new Node (data); if (dummy == null) { dummy = temp ; tail = temp ; } else { tail . next = temp ; tail = temp ; } } // function for finding intersection and // adding it to dummy list function sortedIntersect () { // pointers for iterating var p = a , q = b ; while (p != null && q != null) { if (p . data == q . data) { // add to dummy list push (p . data); p = p . next ; q = q . next ; } else if (p . data < q . data) p = p . next ; else q = q . next ; } } // Driver code // creating first linked list a = new Node (1); a . next = new Node (2); a . next . next = new Node (3); a . next . next . next = new Node (4); a . next . next . next . next = new Node (6); // creating second linked list b = new Node (2); b . next = new Node (4); b . next . next = new Node (6); b . next . next . next = new Node (8); // function call for intersection sortedIntersect (); // print required intersection document . write ("Linked list containing common items of a & b
"); printList (dummy); // This code is contributed by todaysgaurav < /script> Output Linked list containing common items of a & b 2 4 6 Complexity Analysis: Time Complexity: O(m+n) where m and n are number of nodes in first and second linked lists respectively. Only one traversal of the lists are needed. Auxiliary Space: O(min(m, n)). The output list can store at most min(m,n) nodes . Method 2 : Using Local References. Approach: This solution is structurally very similar to the above, but it avoids using a dummy node Instead, it maintains a struct node** pointer, lastPtrRef, that always points to the last pointer of the result list. This solves the same case that the dummy node did — dealing with the result list when it is empty. If the list is built at its tail, either the dummy node or the struct node** "reference" strategy can be used. Below is the implementation of the above approach: C++14 // C++ program to implement above approach #include <bits/stdc++.h> /* Link list node */ struct Node { int data ; struct Node * next ; }; void push (struct Node ** head_ref , int new_data); /* This solution uses the local reference */ struct Node * sortedIntersect (struct Node * a , struct Node * b) { struct Node * result = NULL ; struct Node ** lastPtrRef = & result ; /* Advance comparing the first nodes in both lists. When one or the other list runs out, we're done. */ while (a != NULL && b != NULL) { if (a -> data == b -> data) { /* found a node for the intersection */ push (lastPtrRef , a -> data); lastPtrRef = & ((* lastPtrRef) -> next); a = a -> next ; b = b -> next ; } else if (a -> data < b -> data) a = a -> next ; /* advance the smaller list */ else b = b -> next ; } return (result); } /* UTILITY FUNCTIONS */ /* 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 of the new node */ new_node -> next = (* head_ref); /* move the head to point to the new node */ (* head_ref) = new_node ; } /* Function to print nodes in a given linked list */ void printList (struct Node * node) { while (node != NULL) { printf ("%d " , node -> data); node = node -> next ; } } /* Driver program to test above functions*/ int main () { /* Start with the empty lists */ struct Node * a = NULL ; struct Node * b = NULL ; struct Node * intersect = NULL ; /* Let us create the first sorted linked list to test the functions Created linked list will be 1->2->3->4->5->6 */ push (& a , 6); push (& a , 5); push (& a , 4); push (& a , 3); push (& a , 2); push (& a , 1); /* Let us create the second sorted linked list Created linked list will be 2->4->6->8 */ push (& b , 8); push (& b , 6); push (& b , 4); push (& b , 2); /* Find the intersection two linked lists */ intersect = sortedIntersect (a , b); printf (" \n Linked list containing common items of a & b \n "); printList (intersect); return 0 ; } //This code is contributed by Abhijeet Kumar(abhijeet19403) C #include <stdio.h> #include <stdlib.h> /* Link list node */ struct Node { int data ; struct Node * next ; }; void push (struct Node ** head_ref , int new_data); /* This solution uses the local reference */ struct Node * sortedIntersect (struct Node * a , struct Node * b) { struct Node * result = NULL ; struct Node ** lastPtrRef = & result ; /* Advance comparing the first nodes in both lists. When one or the other list runs out, we're done. */ while (a != NULL && b != NULL) { if (a -> data == b -> data) { /* found a node for the intersection */ push (

lastPtrRef , a -> data); lastPtrRef = & ((* lastPtrRef) -> next); a = a -> next ; b = b -> next ; } else if (a -> data < b -> data) a = a -> next ; /* advance the smaller list */ else b = b -> next ; } return (result); } /* UTILITY FUNCTIONS */ /* 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 of the new node */ new_node -> next = (* head_ref); /* move the head to point to the new node */ (* head_ref) = new_node ; } /* Function to print nodes in a given linked list */ void printList (struct Node * node) { while (node != NULL) { printf ("%d " , node -> data); node = node -> next ; } } /* Driver program to test above functions*/ int main () { /* Start with the empty lists */ struct Node * a = NULL ; struct Node * b = NULL ; struct Node * intersect = NULL ; /* Let us create the first sorted linked list to test the functions Created linked list will be 1->2->3->4->5->6 */ push (& a , 6); push (& a , 5); push (& a , 4); push (& a , 3); push (& a , 2); push (& a , 1); /* Let us create the second sorted linked list Created linked list will be 2->4->6->8 */ push (& b , 8); push (& b , 6); push (& b , 4); push (& b , 2); /* Find the intersection two linked lists */ intersect = sortedIntersect (a , b); printf (" \n Linked list containing common items of a & b \n "); printList (intersect); getchar (); } Java // Java program to implement above approach import java.util.* ; import java.io.* ; public class GFG { // Link list node static class Node { int data ; Node next ; Node (int d) { data = d ; next = null ; } }; static Node sortedIntersect (Node a , Node b) { Node result = new Node (0); Node curr = result ; /* Advance comparing the first nodes in both lists. When one or the other list runs out, we're done. */ while (a != null && b != null) { if (a . data == b . data) { /* found a node for the intersection */ curr . next = new Node (a . data); curr = curr . next ; a = a . next ; b = b . next ; } else if (a . data < b . data) a = a . next ; /* advance the smaller list */ else b = b . next ; } result = result . next ; return result ; } /* UTILITY FUNCTIONS */ /* Function to insert a node at the beginning of the linked list */ static Node push (Node head_ref , int new_data) { /* Allocate node */ Node new_node = new Node (new_data); /* Link the old list of the new node */ new_node . next = head_ref ; /* Move the head to point to the new node */ head_ref = new_node ; return head_ref ; } /* Function to print nodes in a given linked list */ static void printList (Node node) { while (node != null) { System . out . print (node . data + " "); node = node . next ; } } // Driver code public static void main (String [] args) { /* Start with the empty lists */ Node a = null ; Node b = null ; Node intersect = null ; /* Let us create the first sorted linked list to test the functions Created linked list will be 1.2.3.4.5.6 */ a = push (a , 6); a = push (a , 5); a = push (a , 4); a = push (a , 3); a = push (a , 2); a = push (a , 1); /* Let us create the second sorted linked list Created linked list will be 2.4.6.8 */ b = push (b , 8); b = push (b , 6); b = push (b , 4); b = push (b , 2); /* Find the intersection two linked lists */ intersect = sortedIntersect (a , b); System . out . print ("\n Linked list containing " + "common items of a & b \n "); printList (intersect); } } // This code is contributed by Abhijeet Kumar(abhijeet19403) Python3 # Python3 program to implement above approach # Link list node class Node : def __init__ (self , d): self . data = d self . next = None def sortedIntersect (a , b): result = Node (0) curr = result # Advance comparing the first # nodes in both lists. # When one or the other list runs # out, we're done. while (a != None and b != None): if (a . data == b . data): # found a node for the intersection curr . next = Node (a . data) curr = curr . next a = a . next b = b . next elif (a . data < b . data): a = a . next # advance the smaller list else : b = b . next result = result . next return result # UTILITY FUNCTIONS # Function to insert a node at the beginning of the linked list def push (head_ref , new_data): # Allocate node new_node = Node (new_data) # Link the old list of the new node new_node . next = head_ref # Move the head to point to the new node head_ref = new_node return head_ref # Function to print nodes in a given linked list def printList (node): while (node != None): print (node . data , end = " ") node = node . next # Driver code # Start with the empty lists a = None b = None intersect = None # Let us create the first sorted linked list to test the functions Created # linked list will be 1.2.3.4.5.6 a = push (a , 6) a = push (a , 5) a = push (a , 4) a = push (a , 3) a = push (a , 2) a = push (a , 1) # Let us create the second sorted linked list Created linked list will be # 2.4.6.8 b = push (b , 8) b = push (b , 6) b = push (b , 4) b = push (b , 2) # Find the intersection two linked lists intersect = sortedIntersect (a , b) print ("Linked list containing " + "common items of a & b") printList (intersect) # This code is contributed by Abhijeet Kumar(abhijeet19403) C# // C# program to implement above approach using System ; public class GFG { // Link list node public class Node { public int data ; public Node next ; public Node (int d) { data = d ; next = null ; } }; static Node sortedIntersect (Node a , Node b) { Node result = new Node (0); Node curr = result ; /* Advance comparing the first nodes in both lists. When one or the other list runs out, we're done. */ while (a != null && b != null) { if (a . data == b . data) { /* found a node for the intersection */ curr . next = new Node (a . data); curr = curr . next ; a = a . next ; b = b . next ; } else if (a . data < b . data) a = a . next ; /* advance the smaller list */ else b

= b . next ; } result = result . next ; return result ; } /* UTILITY FUNCTIONS */ /* * Function to insert a node at the beginning of the * linked list */ static Node push (Node head_ref , int new_data) { /* Allocate node */ Node new_node = new Node (new_data); /* Link the old list of the new node */ new_node . next = head_ref ; /* Move the head to point to the new node */ head_ref = new_node ; return head_ref ; } /* * Function to print nodes in a given linked list */ static void printList (Node node) { while (node != null) { Console . Write (node . data + " "); node = node . next ; } } // Driver code public static void Main (String [] args) { /* Start with the empty lists */ Node a = null ; Node b = null ; Node intersect = null ; /* * Let us create the first sorted linked list to * test the functions Created linked list will * be 1.2.3.4.5.6 */ a = push (a , 6); a = push (a , 5); a = push (a , 4); a = push (a , 3); a = push (a , 2); a = push (a , 1); /* * Let us create the second sorted linked list * Created linked list will be 2.4.6.8 */ b = push (b , 8); b = push (b , 6); b = push (b , 4); b = push (b , 2); /* Find the intersection two linked lists */ intersect = sortedIntersect (a , b); Console . Write ("\n Linked list containing " + "common items of a & b \n "); printList (intersect); } } // This code is contributed by Abhijeet Kumar(abhijeet19403) JavaScript < script > // Javascript program to implement above approach // Link list node class Node { constructor (val) { this . data = val ; this . next = null ; } } // Function to print nodes in a given linked list function printList (node) { while (node != null) { document . write (node . data + " "); node = node . next ; } } // UTILITY FUNCTION // Function to insert a node at the beginning of the linked list function push (head_ref , new_data) { var new_node = new Node (new_data); new_node . next = head_ref ; head_ref = new_node ; return head_ref ; } // This solution uses the local reference function sortedIntersect (a , b){ var result = new Node (0); var lastptrRef = result ; // Advance comparing the first nodes in both lists. // When one or the other list runs out, we're done while (a != null && b != null){ if (a . data == b . data){ // found a node for the intersection lastptrRef . next = new Node (a . data); lastptrRef = lastptrRef . next ; a = a . next ; b = b . next ; } else if (a . data < b . data) a = a . next ; else b = b . next ; } result = result . next ; return result ; } // Driver program to test above functions // start with the empty lists var a = null ; var b = null ; var intersect = null ; // let us create the first sorted linked list to test the functions // Created linked list will be // 1->2->3->4->5->6 a = push (a , 6); a = push (a , 5); a = push (a , 4); a = push (a , 3); a = push (a , 2); a = push (a , 1); // let us create the second sorted linked list // Created linked list will be // 2->4->6->8 b = push (b , 8); b = push (b , 6); b = push (b , 4); b = push (b , 2); // find the intersection two linked lists intersect = sortedIntersect (a , b); document . write ("Linked list containing common items of a & b"); printList (intersect); // This code is contributed by Yash Agarwal(yashagarwal2852002) < /script> Output Linked list containing common items of a & b 2 4 6 Complexity Analysis: Time Complexity: O(m+n) where m and n are number of nodes in first and second linked lists respectively. Only one traversal of the lists are needed. Auxiliary Space: O(max(m, n)). The output list can store at most m+n nodes. Method 3 : Recursive Solution. Approach: The recursive approach is very similar to the above two approaches. Build a recursive function that takes two nodes and returns a linked list node. Compare the first element of both the lists. If they are similar then call the recursive function with the next node of both the lists. Create a node with the data of the current node and put the returned node from the recursive function to the next pointer of the node created. Return the node created. If the values are not equal then remove the smaller node of both the lists and call the recursive function. Below is the implementation of the above approach: C++ #include <bits/stdc++.h> using namespace std ; // Link list node struct Node { int data ; struct Node * next ; }; struct Node * sortedIntersect (struct Node * a , struct Node * b) { // base case if (a == NULL || b == NULL) return NULL ; /* If both lists are non-empty */ /* Advance the smaller list and call recursively */ if (a -> data < b -> data) return sortedIntersect (a -> next , b); if (a -> data > b -> data) return sortedIntersect (a , b -> next); // Below lines are executed only // when a->data == b->data struct Node * temp = (struct Node *) malloc (sizeof (struct Node)); temp -> data = a -> data ; // Advance both lists and call recursively temp -> next = sortedIntersect (a -> next , b -> next); return temp ; } /* UTILITY FUNCTIONS */ /* 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 of the new node */ new_node -> next = (* head_ref); /* Move the head to point to the new node */ (* head_ref) = new_node ; } /* Function to print nodes in a given linked list */ void printList (struct Node * node) { while (node != NULL) { cout << " " << node -> data ; node = node -> next ; } } // Driver code int main () { /* Start with the empty lists */ struct Node * a = NULL ; struct Node * b = NULL ; struct Node * intersect = NULL ; /* Let us create the first sorted linked list to test the functions Created linked list will be 1->2->3->4->5->6 */ push (& a , 6); push (& a , 5); push (& a , 4); push (& a , 3); push (& a , 2); push (& a , 1); /* Let us create the second sorted linked list Created linked list will be 2->4->6->8 */

push (& b , 8); push (& b , 6); push (& b , 4); push (& b , 2); /* Find the intersection two linked lists */ intersect = sortedIntersect (a , b); cout << " \n Linked list containing " << "common items of a & b \n " ; printList (intersect); return 0 ; } // This code is contributed by shivanisinghss2110 C #include <stdio.h> #include <stdlib.h> /* Link list node */ struct Node { int data ; struct Node * next ; }; struct Node * sortedIntersect (struct Node * a , struct Node * b) { /* base case */ if (a == NULL || b == NULL) return NULL ; /* If both lists are non-empty */ /* advance the smaller list and call recursively */ if (a -> data < b -> data) return sortedIntersect (a -> next , b); if (a -> data > b -> data) return sortedIntersect (a , b -> next); // Below lines are executed only // when a->data == b->data struct Node * temp = (struct Node *) malloc (sizeof (struct Node)); temp -> data = a -> data ; /* advance both lists and call recursively */ temp -> next = sortedIntersect (a -> next , b -> next); return temp ; } /* UTILITY FUNCTIONS */ /* 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 of the new node */ new_node -> next = (* head_ref); /* move the head to point to the new node */ (* head_ref) = new_node ; } /* Function to print nodes in a given linked list */ void printList (struct Node * node) { while (node != NULL) { printf ("%d " , node -> data); node = node -> next ; } } /* Driver program to test above functions*/ int main () { /* Start with the empty lists */ struct Node * a = NULL ; struct Node * b = NULL ; struct Node * intersect = NULL ; /* Let us create the first sorted linked list to test the functions Created linked list will be 1->2->3->4->5->6 */ push (& a , 6); push (& a , 5); push (& a , 4); push (& a , 3); push (& a , 2); push (& a , 1); /* Let us create the second sorted linked list Created linked list will be 2->4->6->8 */ push (& b , 8); push (& b , 6); push (& b , 4); push (& b , 2); /* Find the intersection two linked lists */ intersect = sortedIntersect (a , b); printf (" \n Linked list containing common items of a & b \n "); printList (intersect); return 0 ; } Java import java.util.* ; import java.io.* ; public class GFG { // Link list node static class Node { int data ; Node next ; }; static Node sortedIntersect (Node a , Node b) { // base case if (a == null || b == null) return null ; /* If both lists are non-empty */ /* Advance the smaller list and call recursively */ if (a . data < b . data) return sortedIntersect (a . next , b); if (a . data > b . data) return sortedIntersect (a , b . next); // Below lines are executed only // when a.data == b.data Node temp = new Node (); temp . data = a . data ; // Advance both lists and call recursively temp . next = sortedIntersect (a . next , b . next); return temp ; } /* UTILITY FUNCTIONS */ /* Function to insert a node at the beginning of the linked list */ static Node 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 of the new node */ new_node . next = head_ref ; /* Move the head to point to the new node */ head_ref = new_node ; return head_ref ; } /* Function to print nodes in a given linked list */ static void printList (Node node) { while (node != null) { System . out . print (" " + node . data); node = node . next ; } } // Driver code public static void main (String [] args) { /* Start with the empty lists */ Node a = null ; Node b = null ; Node intersect = null ; /* Let us create the first sorted linked list to test the functions Created linked list will be 1.2.3.4.5.6 */ a = push (a , 6); a = push (a , 5); a = push (a , 4); a = push (a , 3); a = push (a , 2); a = push (a , 1); /* Let us create the second sorted linked list Created linked list will be 2.4.6.8 */ b = push (b , 8); b = push (b , 6); b = push (b , 4); b = push (b , 2); /* Find the intersection two linked lists */ intersect = sortedIntersect (a , b); System . out . print ("\n Linked list containing " + "common items of a & b \n "); printList (intersect); } } // This code is contributed by umadevi9616 Python3 # Link list node class Node : def __init__ (self): self . data = 0 self . next = None def sortedIntersect (a , b): # base case if (a == None or b == None): return None # If both lists are non-empty # Advance the smaller list and call recursively if (a . data < b . data): return sortedIntersect (a . next , b); if (a . data > b . data): return sortedIntersect (a , b . next); # Below lines are executed only # when a.data == b.data temp = Node (); temp . data = a . data ; # Advance both lists and call recursively temp . next = sortedIntersect (a . next , b . next); return temp ; # UTILITY FUNCTIONS # Function to insert a node at the beginning of the linked list def push (head_ref , new_data): # Allocate node new_node = Node () # Put in the data new_node . data = new_data ; # Link the old list of the new node new_node . next = head_ref ; # Move the head to point to the new node head_ref = new_node ; return head_ref ; # Function to print nodes in a given linked list def printList (node): while (node != None): print (node . data , end = " ") node = node . next ; # Driver code # Start with the empty lists a = None b = None intersect = None # Let us create the first sorted linked list to test the functions Created # linked list will be 1.2.3.4.5.6 a = push (a , 6) a = push (a , 5) a = push (a , 4) a = push (a , 3) a = push (a , 2) a = push (a , 1) # Let us create the second sorted linked list Created linked list will be # 2.4.6.8 b = push (b , 8) b = push (b , 6) b = push (b , 4) b = push (b , 2) # Find the intersection two linked lists intersect = sortedIntersect (a , b) print (" \n Linked list containing " + "common items of a &

b"); printList (intersect) # This code is contributed by Saurabh Jaiswal C# using System ; public class GFG { // Link list node public class Node { public int data ; public Node next ; }; static Node sortedIntersect (Node a , Node b) { // base case if (a == null || b == null) return null ; /* If both lists are non-empty */ /* * Advance the smaller list and call recursively */ if (a . data < b . data) return sortedIntersect (a . next , b); if (a . data > b . data) return sortedIntersect (a , b . next); // Below lines are executed only // when a.data == b.data Node temp = new Node (); temp . data = a . data ; // Advance both lists and call recursively temp . next = sortedIntersect (a . next , b . next); return temp ; } /* UTILITY FUNCTIONS */ /* * Function to insert a node at the beginning of the linked list */ static Node 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 of the new node */ new_node . next = head_ref ; /* Move the head to point to the new node */ head_ref = new_node ; return head_ref ; } /* * Function to print nodes in a given linked list */ static void printList (Node node) { while (node != null) { Console . Write (" " + node . data); node = node . next ; } } // Driver code public static void Main (String [] args) { /* Start with the empty lists */ Node a = null ; Node b = null ; Node intersect = null ; /* * Let us create the first sorted linked list to test the functions Created * linked list will be 1.2.3.4.5.6 */ a = push (a , 6); a = push (a , 5); a = push (a , 4); a = push (a , 3); a = push (a , 2); a = push (a , 1); /* * Let us create the second sorted linked list Created linked list will be * 2.4.6.8 */ b = push (b , 8); b = push (b , 6); b = push (b , 4); b = push (b , 2); /* Find the intersection two linked lists */ intersect = sortedIntersect (a , b); Console . Write ("\n Linked list containing " + "common items of a & b \n "); printList (intersect); } } // This code is contributed by umadevi9616 JavaScript < script > // Link list node class Node { constructor (){ this . data = 0 ; this . next = null ; } } function sortedIntersect (a , b) { // base case if (a == null || b == null) return null ; /* If both lists are non-empty */ /* * Advance the smaller list and call recursively */ if (a . data < b . data) return sortedIntersect (a . next , b); if (a . data > b . data) return sortedIntersect (a , b . next); // Below lines are executed only // when a.data == b.data var temp = new Node (); temp . data = a . data ; // Advance both lists and call recursively temp . next = sortedIntersect (a . next , b . next); return temp ; } /* UTILITY FUNCTIONS */ /* * Function to insert a node at the beginning of the linked list */ function push (head_ref , new_data) { /* Allocate node */ var new_node = new Node (); /* Put in the data */ new_node . data = new_data ; /* Link the old list of the new node */ new_node . next = head_ref ; /* Move the head to point to the new node */ head_ref = new_node ; return head_ref ; } /* * Function to print nodes in a given linked list */ function printList (node) { while (node != null) { document . write (" " + node . data); node = node . next ; } } // Driver code /* Start with the empty lists */ var a = null ; var b = null ; var intersect = null ; /* * Let us create the first sorted linked list to test the functions Created * linked list will be 1.2.3.4.5.6 */ a = push (a , 6); a = push (a , 5); a = push (a , 4); a = push (a , 3); a = push (a , 2); a = push (a , 1); /* * Let us create the second sorted linked list Created linked list will be * 2.4.6.8 */ b = push (b , 8); b = push (b , 6); b = push (b , 4); b = push (b , 2); /* Find the intersection two linked lists */ intersect = sortedIntersect (a , b); document . write ("\n Linked list containing " + "common items of a & b
 "); printList (intersect); // This code is contributed by Rajput-Ji < /script> Output Linked list containing common items of a & b 2 4 6 Complexity Analysis: Time Complexity: O(m+n) where m and n are number of nodes in first and second linked lists respectively. Only one traversal of the lists are needed. Auxiliary Space: O(max(m, n)). The output list can store at most m+n nodes. Method 4: Use Hashing C++14 // C++ program to implement above approach #include <bits/stdc++.h> using namespace std ; // Link list node struct Node { int data ; struct Node * next ; }; void printList (struct Node * node) { while (node != NULL) { cout << " " << node -> data ; node = node -> next ; } } void append (struct Node ** head_ref , int new_data) { struct Node * new_node = (struct Node *) malloc (sizeof (struct Node)); new_node -> data = new_data ; new_node -> next = (* head_ref); (* head_ref) = new_node ; } vector < int > intersection (struct Node * tmp1 , struct Node * tmp2 , int k) { vector < int > res (k); unordered_set < int > set ; while (tmp1 != NULL) { set . insert (tmp1 -> data); tmp1 = tmp1 -> next ; } int cnt = 0 ; while (tmp2 != NULL) { if (set . find (tmp2 -> data) != set . end ()) { res [cnt] = tmp2 -> data ; cnt ++ ; } tmp2 = tmp2 -> next ; } return res ; } // Driver code int main () { struct Node * ll = NULL ; struct Node * ll1 = NULL ; append (& ll , 7); append (& ll , 6); append (& ll , 5); append (& ll , 4); append (& ll , 3); append (& ll , 2); append (& ll , 1); append (& ll , 0); append (& ll1 , 7); append (& ll1 , 6); append (& ll1 , 5); append (& ll1 , 4); append (& ll1 , 3); append (& ll1 , 12); append (& ll1 , 0); append (& ll1 , 9); vector < int > arr = intersection (ll , ll1 , 6); for (int i : arr) cout << i << " \n " ; return 0 ; } // This code is contributed by Abhijeet Kumar(abhijeet19403) Java import java.io.* ; import java.util.* ; public class LinkedList { Node head ; static class Node { int data ; Node next ; Node (int d) { data = d ; next = null ; } } public void printList () { Node n = head ; while (n != null) { System . out . println (n . data + " "); n = n . next ; } }

public void append (int d) { Node n = new Node (d); if (head == null) { head = new Node (d); return ; } n . next = null ; Node last = head ; while (last . next != null) { last = last . next ; } last . next = n ; return ; } static int [] intersection (Node tmp1 , Node tmp2 , int k) { int [] res = new int [k] ; HashSet < Integer > set = new HashSet < Integer > (); while (tmp1 != null) { set . add (tmp1 . data); tmp1 = tmp1 . next ; } int cnt = 0 ; while (tmp2 != null) { if (set . contains (tmp2 . data)) { res [cnt] = tmp2 . data ; cnt ++ ; } tmp2 = tmp2 . next ; } return res ; } public static void main (String [] args) { LinkedList ll = new LinkedList (); LinkedList ll1 = new LinkedList (); ll . append (0); ll . append (1); ll . append (2); ll . append (3); ll . append (4); ll . append (5); ll . append (6); ll . append (7); ll1 . append (9); ll1 . append (0); ll1 . append (12); ll1 . append (3); ll1 . append (4); ll1 . append (5); ll1 . append (6); ll1 . append (7); int [] arr = intersection (ll . head , ll1 . head , 6); for (int i : arr) { System . out . println (i); } } } // This code is contributed by ayyuce demirbas Python3 # Python3 program to implement above approach # Link list node class Node : def __init__ (self): self . data = 0 self . next = None def printList (node): while (node != None): print (node . data , end = " ") node = node . next ; def append (head_ref , new_data): new_node = Node () new_node . data = new_data ; new_node . next = head_ref ; head_ref = new_node ; return head_ref ; def intersection (tmp1 , tmp2 , k): res = [0] * k set1 = set () while (tmp1 != None): set1 . add (tmp1 . data) tmp1 = tmp1 . next cnt = 0 while (tmp2 != None): if tmp2 . data in set1 : res [cnt] = tmp2 . data ; cnt += 1 tmp2 = tmp2 . next return res def printList (node): while (node != None): print (node . data , end = " ") node = node . next ; # Driver code # Start with the empty lists ll = None ll1 = None ll = append (ll , 7) ll = append (ll , 6) ll = append (ll , 5) ll = append (ll , 4) ll = append (ll , 3) ll = append (ll , 2) ll = append (ll , 1) ll = append (ll , 0) ll1 = append (ll1 , 7) ll1 = append (ll1 , 6) ll1 = append (ll1 , 5) ll1 = append (ll1 , 4) ll1 = append (ll1 , 3) ll1 = append (ll1 , 12) ll1 = append (ll1 , 0) ll1 = append (ll1 , 9) arr = intersection (ll , ll1 , 6) for i in range (6): print (arr [i]) # This code is contributed by Abhijeet Kumar(abhijeet19403) C# using System ; using System.Collections.Generic ; // This code is contributed by ayyuce demirbas public class List { Node head ; public class Node { public int data ; public Node next ; public Node (int d) { data = d ; next = null ; } } public void printList () { Node n = head ; while (n != null) { Console . WriteLine (n . data + " "); n = n . next ; } } public void append (int d) { Node n = new Node (d); if (head == null) { head = new Node (d); return ; } n . next = null ; Node last = head ; while (last . next != null) { last = last . next ; } last . next = n ; return ; } static int [] intersection (Node tmp1 , Node tmp2 , int k) { int [] res = new int [k]; HashSet < int > set = new HashSet < int > (); while (tmp1 != null) { set . Add (tmp1 . data); tmp1 = tmp1 . next ; } int cnt = 0 ; while (tmp2 != null) { if (set . Contains (tmp2 . data)) { res [cnt] = tmp2 . data ; cnt ++ ; } tmp2 = tmp2 . next ; } return res ; } public static void Main (String [] args) { List ll = new List (); List ll1 = new List (); ll . append (0); ll . append (1); ll . append (2); ll . append (3); ll . append (4); ll . append (5); ll . append (6); ll . append (7); ll1 . append (9); ll1 . append (0); ll1 . append (12); ll1 . append (3); ll1 . append (4); ll1 . append (5); ll1 . append (6); ll1 . append (7); int [] arr = intersection (ll . head , ll1 . head , 6); foreach (int i in arr) { Console . WriteLine (i); } } } // This code is contributed by umadevi9616 JavaScript // JavaScript Program to implement above approach // Link List Node class Node { constructor (data){ this . data = data ; this . next = null ; } } function printList (node){ while (node != null){ document . write (node . data + " "); node = node . next ; } } function append (head_ref , new_data){ new_node = new Node (); new_node . data = new_data ; new_node . next = head_ref ; head_ref = new_node ; return head_ref ; } function intersection (tmp1 , tmp2 , k){ let res = new Array (k); let set = new Set (); while (tmp1 != null){ set . add (tmp1 . data); tmp1 = tmp1 . next ; } let cnt = 0 ; while (tmp2 != null){ if (set . has (tmp2 . data)){ res [cnt] = tmp2 . data ; cnt ++ ; } tmp2 = tmp2 . next ; } return res ; } // Driver Code let ll = null ; let ll1 = null ; ll = append (ll , 7) ll = append (ll , 6) ll = append (ll , 5) ll = append (ll , 4) ll = append (ll , 3) ll = append (ll , 2) ll = append (ll , 1) ll = append (ll , 0) ll1 = append (ll1 , 7) ll1 = append (ll1 , 6) ll1 = append (ll1 , 5) ll1 = append (ll1 , 4) ll1 = append (ll1 , 3) ll1 = append (ll1 , 12) ll1 = append (ll1 , 0) ll1 = append (ll1 , 9) let arr = intersection (ll , ll1 , 6); for (let i = 0 ; i < 6 ; i ++){ document . write (arr [i] + " "); } // This code is contributed by Yash Agarwal Output 0 3 4 5 6 7 Complexity Analysis: Time Complexity: O(n) Space complexity: O(n) since auxiliary space is being used Please write comments if you find the above codes/algorithms incorrect, or find better ways to solve the same problem. References: cslibrary.stanford.edu/105/LinkedListProblems.pdf Comment Article Tags: Article Tags: Linked List Sorting DSA Microsoft Amazon D-E-Shaw Zopper + 3 More