	Problem	Solution	Time Complexity	Failure cases
116/122/199	Given only a pointer/reference to a node to be deleted in a singly linked list, how do you delete it	Copy the contents of next node to this node and delete the next node.		Doesn't work if the node to be deleted is last node. Possible workaround is to add a dummy node to the list at the end
2	Given a Matrix mat of N*N size, the task is to complete the function constructLinkedMatrix(), that constructs a 2D linked list representation of the given matrix.	Recursively construct the node starting from root.		none
3	Quick Sort on Linked List			
4	Extract Leaves of a Binary Tree in a Doubly Linked List (in place)	Do inorder traversal, maintain a previous pointer at each leaf node, once we move to next leaf assign current left to prev, prev right to current	O(n)	None
5	QuickSort on Doubly Linked List			
6	Convert a Binary Tree to a Circular Doubly Link List	Can use modified inorder traversal, D:\preparation\Rent-a-desk\src\Tree.java	O(n)	None
7	Count pairs from two linked lists whose sum is equal to a given value	Naïve approach: Use 2 loops Sorting: 1st list sort ascending 2nd list descending Follow merge logic Hashing: Store one list in hash map, iterate through other list and check diff is present in hash map	Naïve approach O(n2) Sorting: O(n1logn1)+ O(n2logn2) Hashing: O(n1+n2)	None

			Space : O(n1) assuming n1 is stored in hashmap	
8 1 7 / 1 2 / 1 9	Merge Sort for Linked Lists	MergeSort(headRef) 1) If the head is NULL or there is only one element in the Linked List then return. 2) Else divide the linked list into two halves. get middle using slow and fast pointer method 3) Sort the two halves a and b. MergeSort(a); MergeSort(b);4) Merge the sorted a and b and update the head pointer using headRef. *headRef = SortedMerge(a, b);	O(n Log n)	None
9	Sort linked list which is already sorted on absolute values	1) Insertion or bubble sort 2) Merge Sort 3) efficient solution All negative numbers are present in reverse order. So we traverse the list, whenever we find an element that is out of order, we move it to the front of linked list	O(n2) O(nlogn) O(n)	None

1
0
	Reverse a linked list	Node prev = null;
		Node current = head;
		Node next = null;
		while (current != null)
		{
		next = current.next;
		current.next = prev;
		prev = current;
		current = next;
		}
		return prev;
			Assuming it is O(n) as we only do a single pass over the list.	None

11	Add 1 to a number represented as linked list	Method1: Reverse given linked list. For example, 1-> 9-> 9 -> 9 is converted to 9-> 9 -> 9 ->1. Start traversing linked list from leftmost node and add 1 to it. If there is a carry, move to the next node. Keep moving to the next node while there is a carry. Reverse modified linked list and return head. Method 2: We can recursively reach the last node and forward carry to previous nodes. Recursive solution doesn't require reversing of linked list. We can also use a stack in place of recursion to temporarily hold nodes.	Assumtion O(n)+O(n)+ O(n) 1st and last for reverse the linked list. Middle for actual addition.	None
12	Split a Circular Linked List into two halves	1) Store the mid and last pointers of the circular linked list using tortoise and hare algorithm. (slow pointer, fast pointer) 2) Make the second half circular. 3) Make the first half circular. 4) Set head (or start) pointers of the two linked lists. Hint : If there are odd nodes in the circular list then fast_ptr->next becomes head and for even nodes fast_ptr->next->next becomes head	O(n)	None
13	Reverse a Doubly Linked List	All we need to do is swap prev and next pointers for all nodes, change prev of the head (or start) and	O(n)	

		change the head pointer in the end.		
14	Union of Two Linked Lists	Method 1: Sort both lists using merge sort, and then merge 2 lists using merge. Method 2: Add elements from both list to hashmap. If a number is repeated increment the count. Then form a new list with unique map elements	O(mlogm +nlogn) O(m+n) Space complexity O(m+n)	None
15	Delete Alternate Nodes	Method 1: Take 2 pointer p1 to head p2 to next element. Assign p1.next to p2.next. Then p1=pi1.next...	O(n)	None
16	Deletion in circular linked List	Case 1: List is empty. If the list is empty we will simply return. Case 2:List is not empty If the list is not empty then we define two pointers curr and prev and initialize the pointer curr with the head node. Traverse the list using curr to find the node to be deleted and before moving curr to next node, everytime set prev = curr. If the node is found, check if it is the only node in the list. If yes, set head = NULL and free(curr). If the list has more than one node, check if it is the first node of the list. Condition to check this(curr == head). If yes, then move prev until it reaches the last node. After prev reaches the last node, set head = head		

		-> next and prev -> next = head. Delete curr. If curr is not first node, we check if it is the last node in the list. Condition to check this is (curr -> next == head). If curr is the last node. Set prev -> next = head and delete the node curr by free(curr). If the node to be deleted is neither the first node nor the last node, then set prev -> next = temp -> next and delete curr.		
17	Merge two sorted linked lists such that merged list is in reverse order	Method 1: Reverse both lists and then merge 2 lists. Method 2: Merge both lists and then reverse the list. Method 3: 1) Initialize result list as empty: res = NULL.2) Let 'a' and 'b' be heads first and second lists respectively.3) While (a != NULL and b != NULL) a) Find the smaller of two (Current 'a' and 'b') b) Insert the smaller value node at the front of result. c) Move ahead in the list of smaller node. 4) If 'b' becomes NULL before 'a', insert all nodes of 'a' into result list at the beginning.5) If 'a' becomes NULL before 'b', insert all nodes of 'a' into result list at the beginning.	Method 1 and 2 does more than 1 traversal of the list. Method 3: does only 1 traversal of the list and with no additional space	None
18	Check if a linked list is Circular Linked List	The idea is to store head of the linked list and traverse it. If we reach NULL, linked list is not circular. If reach head	O(n)	None

		again, linked list is circular		

19	Delete middle of linked list	The idea is to use two pointers, slow_ptr and fast_ptr. Both pointers start from head of list. When fast_ptr reaches end, slow_ptr reaches middle. keep track of previous of middle so that we can delete middle.
		while (fast_ptr != null && fast_ptr.next != null)
		{
		fast_ptr = fast_ptr.next.next;
		prev = slow_ptr;
		slow_ptr = slow_ptr.next;
		}
		
		//Delete the middle node
		prev.next = slow_ptr.next;
			O(n)	

20	Implement Stack using Linked List	Push : add node at the head Pop : remove head and return head.		
21	Reorder List Ex: L0 -> L1 -> … -> Ln-1 -> Ln to L0 -> Ln -> L1 -> Ln-1 -> L2 -> Ln-2 … Input: 1 -> 2 -> 3 -> 4Output: 1 -> 4 -> 2 -> 3	1) Find the middle point using tortoise and hare method.2) Split the linked list into two halves using found middle point in step 1.3) Reverse the second half.4) Do alternate merge of first and second halves.	O(n)	
22	Segregate even and odd nodes in a Linked List	Method 1: Go to end of list, then start traversing from front. When a odd number is encountered add that at the end of list. Method 2: split the list into 2 lists even and odd list.	O(n) for both methods.	

		After that append odd list at the end of even list.		

23	Clone a linked list with next and random pointer	Method 1: Create the copy of node 1 and insert it between node 1 & node 2 in original Linked List, create the copy of 2 and insert it between 2 & 3.. Continue in this fashion, add the copy of N afte the Nth node
		2) Now copy the arbitrary link in this fashion
		original->next->arbitrary = original->arbitrary->next; /*TRAVERSE TWO NODES*/
		Now restore the original and copy linked lists in this fashion in a single loop.
		
		original->next = original->next->next;
		copy->next = copy->next->next;
		4) Make sure that last element of original->next is NULL.
			O(n)
			Space O(1)
				None

24	Intersection Point in Y Shapped Linked Lists	Get count of the nodes in the first list, let count be c1. Get count of the nodes in the second list, let count be c2. Get the difference of counts d = abs(c1 – c2) Now traverse the bigger list from the first node till d nodes so that from here onwards both the lists have equal no of nodes. Then we can traverse both the lists in parallel till we come across a common node. (Note that getting a common node is done by comparing the address of the nodes)	O(m+n) Space O(1)	None

		*many solutions exists check gfg		
25	Given a linked list of 0s, 1s and 2s, sort it	Method 1: Traverse the list count the no of 0's, 1's and 2's. Now fill the linked list as per count. Method 2: Take 3 dummy pointers OPtr,1ptr and 2ptr. Now traverse the list, attach each node to corresponding pointer. Then merge the list and remove the dummy pointers	O(n) for both	none
26	Merge two sorted linked lists	REVISIT		
27	Remove duplicates from an unsorted linked list	Method 1: Sort using merge sort Method 2: Use hashmap, remove elements with count 2.	O(nlogn) O(n)	
28	Linked List in Zig-Zag fashion	Take 2 pointer cur and prev Cur points to 2st node, prev to 1st node. Now check cur>prev and cur > cur.next. If not swap corrsponding prev or cur.next. Cur = cur.next.next Prev = prev.next.next; *Mycode check gfg for their sol	O(n)	
29	Remove loop in Linked List	This method is also dependent on Floyd's Cycle detection algorithm. (slow pointer, fast pointer method) Detect Loop using Floyd's Cycle detection algorithm and get the pointer to a loop node.		

		Count the number of nodes in loop. Let the count be k. Fix one pointer to the head and another to a kth node from the head. Move both pointers at the same pace, they will meet at loop starting node. Get a pointer to the last node of the loop and make next of it as NULL.		
30	Find the middle of a given linked list	Traverse linked list using two pointers. Move one pointer by one and other pointer by two. When the fast pointer reaches end slow pointer will reach middle of the linked list.		
31	n'th node from the end of a Linked List	Method 1 (Use length of linked list) 1) Calculate the length of Linked List. Let the length be len. 2) Print the (len – n + 1)th node from the beginning of the Linked List. Method 2 (Use two pointers) Maintain two pointers – reference pointer and main pointer. Initialize both reference and main pointers to head. First, move reference pointer to n nodes from head. Now move both pointers one by one until the reference pointer reaches the end. Now the main pointer will point to nth node from the end	O(n) O(n)	
32	Reverse a Linked List in groups of given size.	REVISIT		
33	Multiply two polynomials	1) Create a product array prod[] of size m+n-1.	O(mn)	

18/12/19		2) Initialize all entries in prod[] as 0. 3) Traverse array A[] and do following for every element A[i] ...(3.a) Traverse array B[] and do following for every element B[j] prod[i+j] = prod[i+j] + A[i] * B[j] 4) Return prod[]. Need to check again		
34	Given a singly linked list, the task is to rearrange it in a way that all odd position nodes are together and all even positions node are together.	Take 2 pointers even pointer and odd pointer. Add elements to above pointers while traversing, then append 2 lists	O(n)	
35	Find length of Loop	Use 2 points to find loop. From that pointer start counting until we again reach that ptr		
36	Insert in a Sorted List	1) If Linked list is empty then make the node as head and return it. 2) If the value of the node to be inserted is smaller than the value of the head node, then insert the node at the start and make it head. 3) In a loop, find the appropriate node after which the input node (let 9) is to be inserted. To find the appropriate node start from the head, keep moving until you reach a node GN (10 in the below diagram) who's value is greater than the input node. The node just before GN is the appropriate node (7).	O(n)	

		4) Insert the node (9) after the appropriate node (7) found in step 3.		
37	Pairwise swap elements of a linked list	Method 1: Swap data while traversing. This might not work if data in each node is huge. Method 2: Swap the pointer for each node.	O(n)	
38	Modify Linked List-1	Method 1: Split the list from the middle. Perform front and back split. If the number of elements is odd, the extra element should go in the 1st(front) list. Reverse the 2nd(back) list. Perfrom the required subtraction while traversing both list simultaneously. Again reverse the 2nd list. Concatenate the 2nd list back to the end of the 1st list. Method 2: . Find the starting point of second half Linked List. 2. Push all elements of second half list into stack s. 3. Traverse list starting from head using temp until stack is not empty and do Modify temp->data by subtracting the top element of stack for every node.	O(n) O(n) Space complexity O(n/2)	
39	Merge K sorted linked lists	Method 1: consider 1st list a base and iteratively add remaining lists.	O(n2)	

		Method 2: 2 lists can be merged in O(n) time. The idea is to pair up K lists and merge each pair in linear time using O(1) space. After first cycle, K/2 lists are left each of size 2*N. After second cycle, K/4 lists are left each of size 4*N and so on. We repeat the procedure until we have only one list left. Method 3: Create a min heap of size k, from each list add 1st element to min heap. Now pop element from min heap and add to result list. Add next node to min heap from the same list as the poped element. Continue this process. Min heap root is always the smallest element in the tree. PriorityQueue can be used as min heap in java	O(nklogk) K is the number of lists. N is the size of each list. O(nklok)	
40	Check if Linked List is Palindrome	Method 1: find the middle of list, push the 2nd half in to stack. Now traverse from head and pop elements, if head and poped elements match it's a palindrome Method 2: Find middle of list, reverse the 2nd half of list. Now compare both the lists if it matches it's a palindrome.	O(n) Sapace O(n/2) O(n) Space O(1)	
41	Length of longest palindrome in linked list	REVISIT	Only O(n2) solution exists	

42	Delete nodes which have a greater value on right side	1. Reverse the list. 2. Traverse the reversed list. Keep max till now. If next node is less than max, then delete the next node, otherwise max = next node. 3. Reverse the list again to retain the original order.	O(n)	
43	Sort a linked list that is sorted alternating ascending and descending orders?	1. Separate two lists. 2. Reverse the one with descending order 3. Merge both lists.	O(n)	
44	Swap Kth nodes from ends	REVISIT		
45	Polynomial addtion	REVISIT		
46	Find the first non-repeating character from a stream of characters	Create an empty DLL. Also create two arrays inDLL[] and repeated[] of size 256. inDLL is an array of pointers to DLL nodes. repeated[] is a boolean array, repeated[x] is true if x is repeated two or more times, otherwise false. inDLL[x] contains pointer to a DLL node if character x is present in DLL, otherwise NULL. Initialize all entries of inDLL[] as NULL and repeated[] as false. To get the first non-repeating character, return character at head of DLL. Following are steps to process a new character 'x' in a stream. If repeated[x] is true, ignore this character (x is already repeated two or more times in the stream) If repeated[x] is false and inDLL[x] is NULL (x is seen first time). Append x to DLL and store address		

		of new DLL node in inDLL[x]. If repeated[x] is false and inDLL[x] is not NULL (x is seen second time). Get DLL node of x using inDLL[x] and remove the node. Also, mark inDLL[x] as NULL and repeated[x] as true.		
47	Decimal Equivalent of Binary Linked List	Method 1: Initialize result to 0. Traverse the list, at each node multiply the result with 2 and add node data to it. The final result value is the Decimal equivalent. Method 2: Reverse the linked list Start index as 0. Sum =0; For each node multiply the data with 2^index and add it to sum.		
48	Sorted Linked List to Balanced BST (almost same for DLL to Balanced BST)	1) Get the Middle of the linked list and make it root. 2) Recursively do same for the left half and right half. a) Get the middle of the left half and make it left child of the root created in step 1. b) Get the middle of right half and make it the right child of the root created in step 1. Method 2: We first count the number of nodes in the given Linked List. Let the count be n. After counting nodes, we take left n/2 nodes and recursively	O(nlogn) O(n)	

		construct the left subtree. After left subtree is constructed, we allocate memory for root and link the left subtree with root. Finally, we recursively construct the right subtree and link it with root.
		While constructing the BST, we also keep moving the list head pointer to next so that we have the appropriate pointer in each recursive call.
		
		TNode sortedListToBSTRecur(int n)
		{
		/* Base Case */
		if (n <= 0)
		return null;
		
		/* Recursively construct the left subtree */
		TNode left = sortedListToBSTRecur(n / 2);
		
		/* head_ref now refers to middle node,
		make middle node as root of BST*/
		TNode root = new TNode(head.data);
		
		// Set pointer to left subtree
		root.left = left;
		
		/* Change head pointer of Linked List for parent
		recursive calls */
		head = head.next;		

		/* Recursively construct the right subtree and link it
		with root. The number of nodes in right subtree is
		total nodes - nodes in left subtree - 1 (for root) */
		root.right = sortedListToBSTRecur(n - n / 2 - 1);
		
		return root;
		}		