LINKED LIST

> https://github.com/SayantanBanerjee16/Striver-DSA-Sheet

> https://github.com/dhruv-yadav-nitj/Striver-A2Z-DSA-Sheet-CPP

Middle of a Linked List | Practice | GeeksforGeeks

Given the head of a linked list, the task is to find the middle. For example, the middle of 1-&gt; 2-&gt;3-&gt;4-&gt;5 is 3. If there are two middle nodes (even count), return the second middle. For example, middle of 1-&gt;2-&gt;3-&gt;4-&gt;5-&gt;6

https://www.geeksforgeeks.org/problems/finding-middle-element-in-a-linked-list/1?itm_source=geeksforgeeks&itm_medium=article&itm_campaign=practice_card

```
class Solution {
public:
    // Function to return the length of the linked list
    int LengthLL(Node* head) {
        int cnt = 0;
        while (head != nullptr) { // Traverse the list to count nodes
            cnt++;
            head = head->next;
        }
        return cnt;
    }

    // Function to return the data of the middle node given the length
    int MiddleLL(Node* head, int mid) {
        int count = 0;
        while (head != nullptr) {
            if (count == mid) // Return the middle node's data
                return head->data;
            count++;
            head = head->next;
        }
        return -1; // Return -1 if mid is out of range
    }

    // Function to get the middle element of the linked list
    int getMiddle(Node* head) {
        int len = LengthLL(head);
        if (len == 0) return -1;  // Return -1 if the list is empty
        int mid = len / 2;        // Calculate the middle index
        return MiddleLL(head, mid); // Get data at the middle index
    }
};

//CAN ALSO USE HARE AND TORTOISE ALGO
```

Reverse a linked list | Practice | GeeksforGeeks

Given the head of a linked list, the task is to reverse this list and return the reversed head. Examples: Input: Linked list: 1-&gt;2-&gt;3-&gt;4-&gt;5-&gt;6 Output: 6-&gt;5-&gt;4-&gt;3-&gt;2-&gt;1 Explanation: Input: Linked list: 2-&gt;7-&gt;10-&gt;

https://www.geeksforgeeks.org/problems/reverse-a-linked-list/1?itm_source=geeksforgeeks&itm_medium=article&itm_campaign=practice_card

Reverse Linked List - LeetCode

Can you solve this real interview question? Reverse Linked List - Given the head of a singly linked list, reverse the list, and return the reversed list. Example 1: [https://assets.leetcode.com/uploads/2021/02/19/rev1ex1.jpg] Input: head =

https://leetcode.com/problems/reverse-linked-list/

```
ListNode* reverseList(ListNode* head) {
        ListNode* curr = head;
        ListNode* prev = NULL;

        while(curr!=NULL){
            ListNode* temp = curr->next;
            curr->next = prev;
            prev = curr;
            curr = temp;
        }

        return prev;
    }
```

```
class Solution {
public:
    ListNode* reverseList(ListNode* head) {
        stack<int> st;

        ListNode* temp = head;
        while (temp != nullptr) {
            st.push(temp->val);
            temp = temp->next;
        }

        temp = head;
        while (temp != nullptr) {
            temp->val = st.top();
            st.pop();
            temp = temp->next;
        }

        return head;
    }
};
```

Add Two Numbers - LeetCode

Can you solve this real interview question? Add Two Numbers - You are given two non-empty linked lists representing two non-negative integers. The digits are stored in reverse order, and each of their nodes contains a single digit. Add the two numbers

https://leetcode.com/problems/add-two-numbers/description/

```
class Solution {
public:
    ListNode* addTwoNumbers(ListNode* l1, ListNode* l2) {
        ListNode* dummy = new ListNode(-1);
        ListNode* curr = dummy;
        int carry = 0;
        while (l1 != nullptr || l2 != nullptr || carry) {
            int sum = 0;

            if (l1 != nullptr) {
                sum = sum + l1->val;
                l1 = l1->next;
            }
            if (l2 != nullptr) {
                sum = sum + l2->val;
                l2 = l2->next;
            }
            sum += carry;
            carry = sum / 10;
            ListNode* ans = new ListNode(sum % 10);
            curr->next = ans;
            curr = curr->next;
        }
        return dummy->next;
    }
};
```

Odd Even Linked List - LeetCode

Can you solve this real interview question? Odd Even Linked List - Given the head of a singly linked list, group all the nodes with odd indices together followed by the nodes with even indices, and return the reordered list. The first node is considered

https://leetcode.com/problems/odd-even-linked-list/

```
BRUTE FORCE

#include <queue>

class Solution {
public:
    ListNode* oddEvenList(ListNode* head) {
        if (!head || !head->next) return head;  // If list has 0 or 1 node, retu

        std::queue<ListNode*> oddQueue;
        std::queue<ListNode*> evenQueue;

        ListNode* current = head;
        bool isOdd = true;  // Flag to alternate between odd and even nodes

        // Separate odd and even nodes into two queues
        while (current != nullptr) {
            if (isOdd) {
                oddQueue.push(current);  // Push odd indexed nodes
            } else {
                evenQueue.push(current);  // Push even indexed nodes
            }
            current = current->next;  // Move to the next node
            isOdd = !isOdd;  // Alternate the flag
        }

        // Reconstruct the linked list
        ListNode* newHead = nullptr;
        ListNode* tail = nullptr;

        // First, add all odd nodes
        while (!oddQueue.empty()) {
            if (newHead == nullptr) {
                newHead = oddQueue.front();  // Initialize the new head
                tail = newHead;  // Initialize the tail
            } else {
                tail->next = oddQueue.front();  // Link the odd nodes
                tail = tail->next;  // Move the tail
            }
            oddQueue.pop();
        }

        // Next, add all even nodes
        while (!evenQueue.empty()) {
            tail->next = evenQueue.front();  // Link the even nodes
            tail = tail->next;  // Move the tail
            evenQueue.pop();
        }

        // End the list
        tail->next = nullptr;

        return newHead;
    }
};
```

```
class Solution {
public:
    ListNode* oddEvenList(ListNode* head) {
        if(head == nullptr || head->next == nullptr) {
            return head;
        }

        ListNode* odd = head;             // Pointer to the odd list
        ListNode* even = head->next;      // Pointer to the even list
        ListNode* evenHead = even;        // Head of the even list

        // Rearrange nodes in odd and even positions
        while(even != nullptr && even->next != nullptr) {
            odd->next = even->next;       // Point odd to the next odd node
            odd = odd->next;              // Move odd pointer forward

            even->next = odd->next;       // Point even to the next even node
            even = even->next;            // Move even pointer forward
        }

        odd->next = evenHead;  // Attach even list after the odd list
        return head;
    }
};
```

`For Brute force solution in LL, always play with the data instead of changing the links.`

`Either use a queue, variables, stack or vector`

`For optimised u have to change the links`

Linked List that is Sorted Alternatingly | Practice | GeeksforGeeks

You are given a Linked list. The list is in alternating ascending and descending orders. Sort&nbsp;the given linked list in non-decreasing order. Examples: Input: LinkedList = 1-&gt;9-&gt;2-&gt;8-&gt;3-&gt;7 Output: 1-&gt;2-&gt;3-&gt;7-&gt;8-&gt;9 Ex

https://www.geeksforgeeks.org/problems/linked-list-that-is-sorted-alternatingly/1?itm_source=geeksforgeeks&itm_medium=article&itm_campaign=practice_card

```
BRUTE FORCE
//use variable cnt0, cnt1, cnt2.

BETTER SOLN
//Similar to odd even just play with links.Consider 3 dummy nodes ie zeroHead,
//oneHead,twoHead

class Solution {
public:
    ListNode* sortList(ListNode* head) {
        if (!head || !head->next) {
            return head; // If the list is empty or has only one element, it's a
        }

        // Create three dummy nodes for 0s, 1s, and 2s
        ListNode* zeroHead = new ListNode(-1);
        ListNode* oneHead = new ListNode(-1);
        ListNode* twoHead = new ListNode(-1);
        // Create three pointers to keep track of the current end of each list
        ListNode* zero = zeroHead;
        ListNode* one = oneHead;
        ListNode* two = twoHead;

        // Traverse the original list and add nodes to the corresponding list
        ListNode* curr = head;
        while (curr) {
            if (curr->val == 0) {
                zero->next = curr;
                zero = zero->next;
            } else if (curr->val == 1) {
                one->next = curr;
                one = one->next;
            } else { // curr->val == 2
                two->next = curr;
                two = two->next;
            }
            curr = curr->next;
        }

        // Link the three lists together
        zero->next = (oneHead->next) ? oneHead->next : twoHead->next; // End of
        one->next = twoHead->next; // End of 1s to start of 2s
        two->next = nullptr; // End of 2s should point to null

        // The new head of the sorted list
        ListNode* sortedHead = zeroHead->next;

        // We don't need to delete the dummy nodes here

        return sortedHead;
    }
};
```

Remove Nth Node From End of List - LeetCode

Can you solve this real interview question? Remove Nth Node From End of List - Given the head of a linked list, remove the nth node from the end of the list and return its head.   Example 1:

https://leetcode.com/problems/remove-nth-node-from-end-of-list/description/

LeetCode

```
//BRUTE FORCE
//Count the length of LL then do rem=N-2 and reach till the previous node to the
// u want to delete and the do temp->next=te,p->next->next.

//BETTER SOLN
//MAke use of slow and fast pointers.
//Move the fast pointer till u reach the value N, then move slow and fast pointe
//simultaneously till fast pointer reaches NULL.When fast pointer reaches NULL t
//slow pointer has reached the previous node to the node that u want to delete.

 ListNode* removeNthFromEnd(ListNode* head, int n) {
        ListNode* fast = head;
        ListNode* slow = head;


        for(int i = 0;i<n;i++){
            fast = fast->next;
        }
        if(fast==nullptr) return head->next;


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

        ListNode* deleteNode = slow->next;
        slow->next = slow->next->next;
        delete(deleteNode);
        return head;
    }
```

Palindrome Linked List - LeetCode

Can you solve this real interview question? Palindrome Linked List - Given the head of a singly linked list, return true if it is a palindrome or false otherwise.   Example 1: [https://assets.leetcode.com/uploads/2021/03/03/pal1linked-list.jpg]   Input: head =

https://leetcode.com/problems/palindrome-linked-list/description/

LeetCode

```
//BRUTE FORCE
//Make use of stack, push the data in it and then compare the top with head till
```

```
//the stack is empty.If any diff found then it is not palindrome.

class Solution {
public:
    bool isPalindrome(ListNode* head) {
        stack<int> s;
        ListNode* current = head;

        // Traverse the list and push the values onto the stack
        while (current != nullptr) {
            s.push(current->val);
            current = current->next;
        }

        current = head;

        // Traverse the list again and compare with the stack
        while (current != nullptr) {
            if (current->val != s.top()) {
                return false;  // Not a palindrome if mismatch found
            }
            s.pop();
            current = current->next;
        }

        return true;  // Palindrome if all values matched
    }
};
```

```
//OPTIMISED CODE
    Node* reverse (Node* head){

        if(head == NULL or head -> next== NULL){
            return head ;
        }

        Node* prev = NULL , *curr= head ;
        while(curr != NULL){

            Node* temp = curr -> next ;
            curr -> next = prev ;
            prev = curr; curr = temp;
        }
        return prev ;
    }

    //Function to check whether the list is palindrome.
    bool isPalindrome(Node *head){
        //Your code here

        // empty or single element linked list is always a palindrome
        if(head == NULL or head -> next == NULL){
            return true ;
        }

        // finding the mid element
        Node* slow= head , *fast= head ;

        // slightly modified according to the need of the ques
        while(fast -> next != NULL and fast -> next -> next != NULL){

            slow = slow -> next; fast = fast -> next -> next ;

        }

        // reverse all the nodes after mid (or slow here)
        slow -> next = reverse(slow -> next) ;

        Node* head1 = head , *head2 = slow -> next ;
        while(head2 != NULL){

            if(head1 -> data != head2-> data) {
                return false ;
            }

            head1 = head1-> next ; head2 = head2 -> next ;
        }

        return true ;

    }
```

Linked List Cycle - LeetCode

Can you solve this real interview question? Linked List Cycle - Given head, the head of a linked list, determine if the linked list has a cycle in it. There is a cycle in a linked list if there is some node in the list that can be reached again by continuously

https://leetcode.com/problems/linked-list-cycle/description/

```
BRUTE FORCE
//Take a map can insert the entire node in the map and use umap.find to check if
//node is already present or not

class Solution {
public:
    bool hasCycle(ListNode* head) {
        unordered_map<ListNode*, bool> visited;  // Map to store visited nodes

        ListNode* current = head;
        while (current != nullptr) {
            // If current node is already present in the map, it means there is
            if (visited.find(current) != visited.end()) {
                return true;
            }

            // Mark the current node as visited
            visited[current] = true;
            current = current->next;
        }

        // If no cycle is found
        return false;
    }
};
```

```
BETTER SOLN
HARE AND TORTOISE ALGO
//if slow pointer and fast pointer at same index- loop detected.
//slow pointer moves by 1 and fast pointer by 2

class Solution {
public:
    bool hasCycle(ListNode* head) {
        if (head == nullptr || head->next == nullptr) {
            return false;  // No cycle if list is empty or has only one node
        }

        ListNode* slow = head;          // Slow pointer
        ListNode* fast = head->next;    // Fast pointer starts from the next nod

        // Traverse the list
        while (fast != nullptr && fast->next != nullptr) {

        //fast != nullptr for even length and fast->next != nullptr for odd leng

            if (slow == fast) {         // If slow and fast meet, there's a cycl
                return true;
            }

            slow = slow->next;          // Move slow pointer one step
            fast = fast->next->next;    // Move fast pointer two steps
        }

        return false;  // No cycle detected
    }
};

//NET REDUCTION IN DISTANCE B/W FAST AND SLOW IS "1"
```

Find length of Loop | Practice | GeeksforGeeks

Given the head of a linked list, determine whether the list contains a loop. If a loop is present, return the number of nodes in the loop, otherwise return 0. Note: 'c'&nbsp;is the position of the node which is the next pointer of the last node of t

https://www.geeksforgeeks.org/problems/find-length-of-loop/1?itm_source=geeksforgeeks&itm_medium=article&itm_campaign=practice_card

```
//code yet to be appended
```

Delete the Middle Node of a Linked List - LeetCode

Can you solve this real interview question? Delete the Middle Node of a Linked List - You are given the head of a linked list. Delete the middle node, and return the head of the modified linked list. The middle node of a linked list of size n is the ⌊n / 2⌋th node

https://leetcode.com/problems/delete-the-middle-node-of-a-linked-list/description/

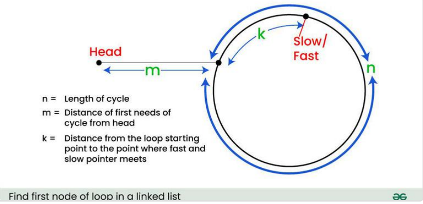
```
class Solution {
public:
    // Function to return the length of the linked list
    int LengthLL(Node* head) {
        int cnt = 0;
        while (head != nullptr) { // Traverse the list to count nodes
            cnt++;
            head = head->next;
        }
        return cnt;
    }

    // Function to delete the middle node given the length
    Node* deleteMiddle(Node* head) {
        int len = LengthLL(head);
        if (len == 0 || len == 1) return nullptr;  // If the list is empty or ha

        int mid = len / 2;  // Calculate the middle index
        Node* temp = head;

        // Traverse to the node just before the middle node
        for (int i = 0; i < mid - 1; i++) {
            temp = temp->next;
        }

        // Delete the middle node
        Node* nodeToDelete = temp->next;
        temp->next = temp->next->next;
        delete nodeToDelete;

        return head;
    }
};
```

```
//HARE AND TORTOISE SOLN

class Solution {
public:
    ListNode* deleteMiddle(ListNode* head) {
        if (!head || !head->next) return nullptr; // If the list is empty or has
        ListNode* slow = head;    // Initialize 'slow' to 'head'
        ListNode* fast = head->next->next; // Initialize 'fast' to the second nc

        // Move 'fast' by 2 steps and 'slow' by 1 step
        while (fast && fast->next) {
            slow = slow->next;        // Move slow by one step
            fast = fast->next->next;  // Move fast by two steps
        }

        // 'slow' is now pointing to the node just before the middle node
        ListNode* middle = slow->next; // Middle node to be deleted
        slow->next = middle->next; // Bypass the middle node
        delete middle;             // Free memory of the deleted node

        return head;
    }
};
```

Find first node of loop in a linked list - GeeksforGeeks

A Computer Science portal for geeks. It contains well written, well thought and well explained computer science and programming articles, quizzes and practice/competitive programming/company interview Questions.

https://www.geeksforgeeks.org/find-first-node-of-loop-in-a-linked-list/


Head
m
k
Slow/
Fast
n
n = Length of cycle
m = Distance of first needs of cycle from head
k = Distance from the loop starting point to the point where fast and slow pointer meets
Find first node of loop in a linked list

```
//BRUTE FORCE SOLN
HASHING CONCEPT
```

```
//OPTIMISED
HARE AND TORTOISE ALGO
slow pointer moves by 1 and fast pointer by 2
step1:if loop exists or not (if fast and slow pointer at same node ie loop exist
step2:Now again put slow pointer at head (keeping fast pointer at the node of cc
and move slow and fast pointer by 1 move each.
Again when it slow and fast reaches at same node that is the starting node of lc
```

Delete all occurrences of a given key in a doubly linked list | Practice | GeeksforGeeks

You are given the head_ref of a doubly Linked List and a Key. Your task is to delete all occurrences of the given key if it is present and return the new DLL. Example1: Input: 2&lt;-&gt;2&lt;-&gt;10&lt;-&gt;8&lt;-&gt;4&lt;-&gt;2&lt;-&gt;5&lt;-&gt;2

https://www.geeksforgeeks.org/problems/delete-all-occurrences-of-a-given-key-in-a-doubly-linked-list/0

```
//yet to code
```

Find pairs with given sum in doubly linked list | Practice | GeeksforGeeks

Given a sorted doubly linked list of positive distinct elements, the task is to find pairs in a doubly-linked list whose sum is equal to given value target. &nbsp; Example 1: Input: 1 &lt;-&gt; 2 &lt;-&gt; 4 &lt;-&gt; 5 &lt;-&gt; 6 &lt;-&gt; 8 &lt;

https://www.geeksforgeeks.org/problems/find-pairs-with-given-sum-in-doubly-linked-list/0

```
//Make use of 2 pointer approach
```

Remove Duplicates from Sorted List - LeetCode

Can you solve this real interview question? Remove Duplicates from Sorted List - Given the head of a sorted linked list, delete all duplicates such that each element appears only once. Return the linked list sorted as well.   Example 1:

https://leetcode.com/problems/remove-duplicates-from-sorted-list/description/

```
class Solution {
public:
    ListNode* deleteDuplicates(ListNode* head) {
        if (head == nullptr || head->next == nullptr) return head; // Return if

        ListNode* temp = head;   // Pointer to traverse the linked list

        while (temp != nullptr && temp->next != nullptr) {
            if (temp->val == temp->next->val) { // If duplicate found
                ListNode* newNode = temp->next;   // Duplicate node
                temp->next = newNode->next;       // Bypass the duplicate node
                delete newNode;                  // Delete the duplicate node
            } else {
                temp = temp->next; // Move to the next node if no duplicate
            }
        }

        return head;
    }
};
```

Rotate List - LeetCode

Can you solve this real interview question? Rotate List - Given the head of a linked list, rotate the list to the right by k places.   Example 1: [https://assets.leetcode.com/uploads/2020/11/13/rotate1.jpg]   Input: head =

https://leetcode.com/problems/rotate-list/description/

```
class Solution {
public:
    ListNode* rotateRight(ListNode* head, int k) {
        if (!head || k == 0) return head;

        // Step 1: Find the length of the list and the last node
        ListNode* old_tail = head;
        int length = 1;
        while (old_tail->next) {
            old_tail = old_tail->next;
            length++;
        }

        // Step 2: Make the list circular
        old_tail->next = head;

        // Step 3: Find the new tail (length - k % length - 1) and new head
        int new_tail_position = length - k % length - 1;
        ListNode* new_tail = head;
        for (int i = 0; i < new_tail_position; ++i) {
            new_tail = new_tail->next;
        }
        ListNode* new_head = new_tail->next;

        // Step 4: Break the circular list
        new_tail->next = nullptr;

        return new_head;
    }
};
```

```
Explanation:
Find Length and Last Node: Traverse the list to find its length and the last noc
Make Circular: Link the last node to the head to form a circular list.
Calculate New Tail: Compute the position of the new tail node.
Break Circular Link: Set the next pointer of the new tail to nullptr to finalize
rotated list.
```

Merge Two Sorted Lists - LeetCode

Can you solve this real interview question? Merge Two Sorted Lists - You are given the heads of two sorted linked lists list1 and list2. Merge the two lists into one sorted list. The list should be made by splicing together the nodes of the first two lists.

https://leetcode.com/problems/merge-two-sorted-lists/description/

LeetCode

```
//BRUTE
Take an array store elements of the first ll and then the second ll
Now sort the array
create a new ll and store the sorted elements in it

class Solution {
public:
    ListNode* mergeTwoLists(ListNode* l1, ListNode* l2) {
        std::vector<int> elements;

        // Step 1: Traverse the first list and store elements in the vector
        ListNode* current = l1;
        while (current != nullptr) {
            elements.push_back(current->val);
            current = current->next;
        }

        // Step 2: Traverse the second list and store elements in the vector
        current = l2;
        while (current != nullptr) {
            elements.push_back(current->val);
            current = current->next;
        }

        // Step 3: Sort the vector
        std::sort(elements.begin(), elements.end());

        // Step 4: Create a new sorted linked list from the sorted vector
        ListNode* dummy = new ListNode(-1); // Dummy node to simplify list creat
        ListNode* tail = dummy;

        for (int value : elements) {
            tail->next = new ListNode(value);
            tail = tail->next;
        }

        ListNode* sortedList = dummy->next;
        delete dummy; // Free the dummy node

        return sortedList;
    }
};
```

```
//OPTIMISED

Initialize Two Pointers:

Use two pointers to traverse the two lists: l1 and l2.
Compare the values at the two pointers.
Attach the node with the smaller value to the merged list.
Move the pointer in the list from which the node was taken.
Repeat until one of the lists is exhausted.
If one list is exhausted before the other,
attach the remaining nodes from the non-exhausted list to the end of the merged


class Solution {
public:
    ListNode* mergeTwoLists(ListNode* list1, ListNode* list2) {
        // Create a dummy node and a tail pointer using new keyword
        ListNode* dummy = new ListNode(0);
        ListNode* tail = dummy; // Tail points to the last node in the merged li

        // Traverse both lists
        while (list1 != nullptr && list2 != nullptr) {
            if (list1->val <= list2->val) {
                tail->next = list1; // Attach list1 node to the merged list
                list1 = list1->next; // Move to the next node in list1
            } else {
                tail->next = list2; // Attach list2 node to the merged list
                list2 = list2->next; // Move to the next node in list2
            }
            tail = tail->next; // Move tail to the last node in the merged list
        }

        // Attach the remaining nodes from list1 or list2
        if (list1 != nullptr) {
            tail->next = list1;
        } else {
            tail->next = list2;
        }

        // Store the result list starting from dummy->next
        ListNode* mergedHead = dummy->next;

        // Clean up the dummy node
        delete dummy;

        return mergedHead;
    }
};
```

Flattening a Linked List | Practice | GeeksforGeeks

Given a Linked List, where every node represents a sub-linked-list and contains two pointers:(i) a next pointer to the next node,(ii) a&nbsp;bottom&nbsp;pointer&nbsp;to a linked list where this node is head.Each of the sub-linked lists is in sorted o

https://www.geeksforgeeks.org/problems/flattening-a-linked-list/1?itm_source=geeksforgeeks&itm_medium=article&itm_campaign=practice_card

```
//BRUTE FORCE
The traverseList function traverses through each node in the main list and
its child lists recursively, collecting all node values into a vector<int>.
The values vector is then sorted.
The createSortedList function constructs a new sorted singly linked list
from the sorted values.

class Solution {
public:
    ListNode* flatten(ListNode* head) {
        vector<int> values;
        // Traverse the list to collect all values
        traverseList(head, values);

        // Sort values
        sort(values.begin(), values.end());

        // Create the new flattened list
        return createSortedList(values);
    }

private:
    void traverseList(ListNode* node, vector<int>& values) {
        while (node != nullptr) {
            values.push_back(node->val); // Collect value from the current node
            if (node->child != nullptr) {
                traverseList(node->child, values); // Traverse child list
            }
            node = node->next; // Move to the next node
        }
    }

    ListNode* createSortedList(const vector<int>& values) {
        if (values.empty()) return nullptr;

        ListNode* dummy = new ListNode(0);
        ListNode* tail = dummy;

        for (int val : values) {
            tail->next = new ListNode(val); // Create a new node with the value
            tail = tail->next; // Move tail
        }

        return dummy->next; // Return the flattened sorted list
    }
};

int main() {
    // Example usage (can be adjusted based on input requirements)
    ListNode* head = new ListNode(3, nullptr, new ListNode(7));
    head->next = new ListNode(1, nullptr, new ListNode(2));
    head->next->next = new ListNode(5, nullptr, new ListNode(6));

    Solution solution;
    ListNode* flattenedHead = solution.flatten(head);

    // Output the flattened list
    ListNode* current = flattenedHead;
    while (current != nullptr) {
        cout << current->val << " ";
        current = current->next;
    }

    return 0;
}
```

```
//OPTIMISED SOLN

```