

Assignment- 3

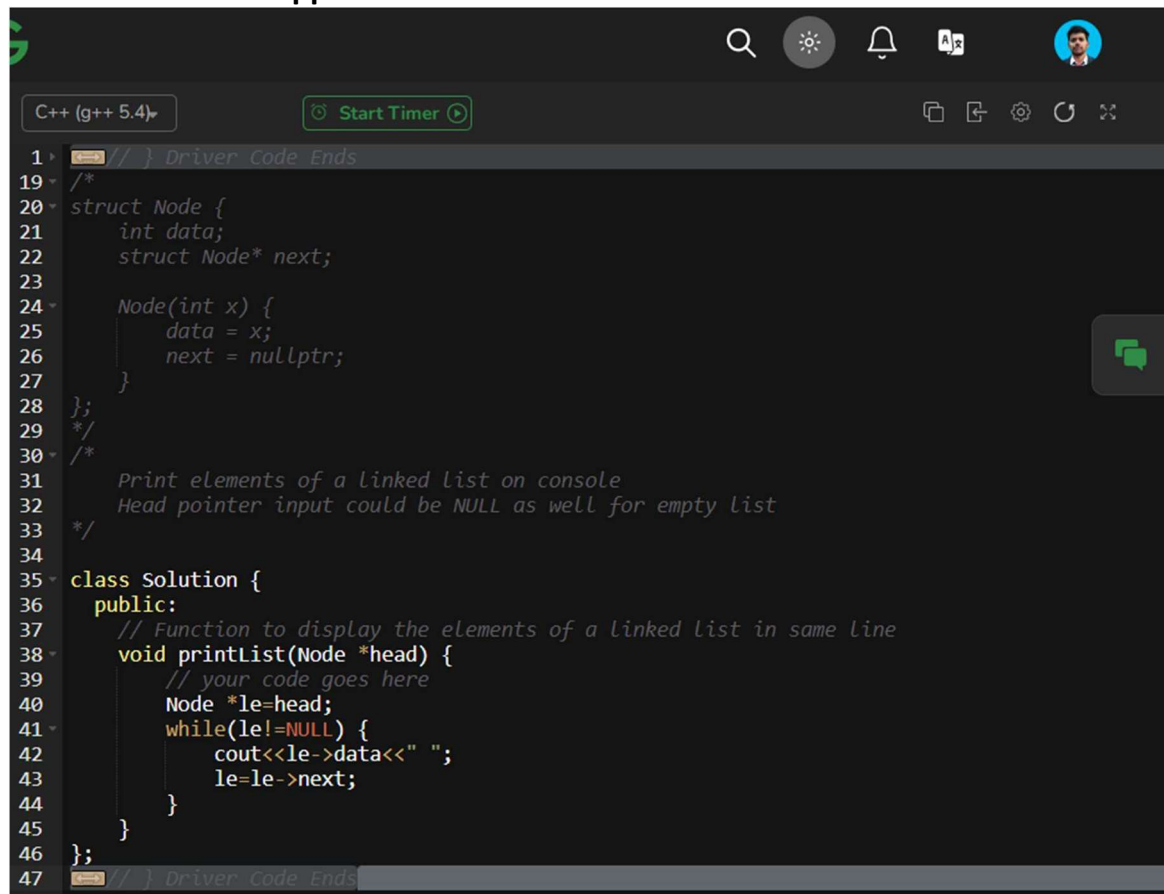
Advanced Programming Lab

Aayush Tewari

UID-22BCS11444

Question 1. Given a linked list. Print all the elements of the linked list separated by space followed.(PRINT LINKEDLIST)

Solution 1. Code Snippet:

A screenshot of a C++ code editor with a dark theme. The editor shows a C++ program for printing a linked list. The code includes a Node struct, a Node constructor, and a printList function within a Solution class. The printList function iterates through the linked list and prints each node's data followed by a space. The editor interface includes a search bar, a 'Start Timer' button, and a user profile icon in the top right corner. The code is as follows:

```
1 // } Driver Code Ends
19 /*
20 struct Node {
21     int data;
22     struct Node* next;
23
24     Node(int x) {
25         data = x;
26         next = nullptr;
27     }
28 };
29 */
30 /*
31     Print elements of a linked list on console
32     Head pointer input could be NULL as well for empty list
33 */
34
35 class Solution {
36 public:
37     // Function to display the elements of a linked list in same line
38     void printList(Node *head) {
39         // your code goes here
40         Node *le=head;
41         while(le!=NULL) {
42             cout<<le->data<<" ";
43             le=le->next;
44         }
45     }
46 };
47 // } Driver Code Ends
```

OUTPUT:

Output Window

Compilation Results

Custom Input

Y.O.G.I. (AI Bot)

Problem Solved Successfully

Suggest Feedback

Test Cases Passed

1112 / 1112

Attempts : Correct / Total

2 / 2

Accuracy : 100%

Time Taken

0.06

Question 2. 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.(REMOVE DUPLICATES FROM LINKEDLIST)

Solution 2.

Code Snippet:

Run Submit

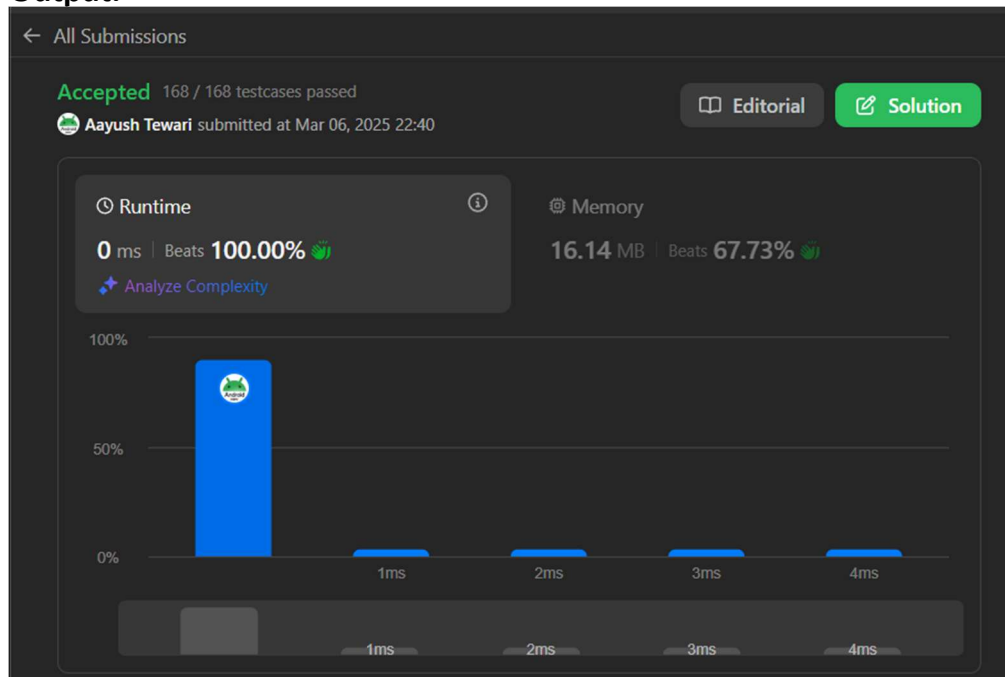
4

</> Code

C++ Auto

```
10 //
11 class Solution {
12 public:
13     ListNode* deleteDuplicates(ListNode* head) {
14         ListNode* curr = head;
15
16         while (curr != nullptr) {
17             while (curr->next && curr->val == curr->next->val)
18                 curr->next = curr->next->next;
19             curr = curr->next;
20         }
21
22         return head;
23     }
24 };
```

Output:



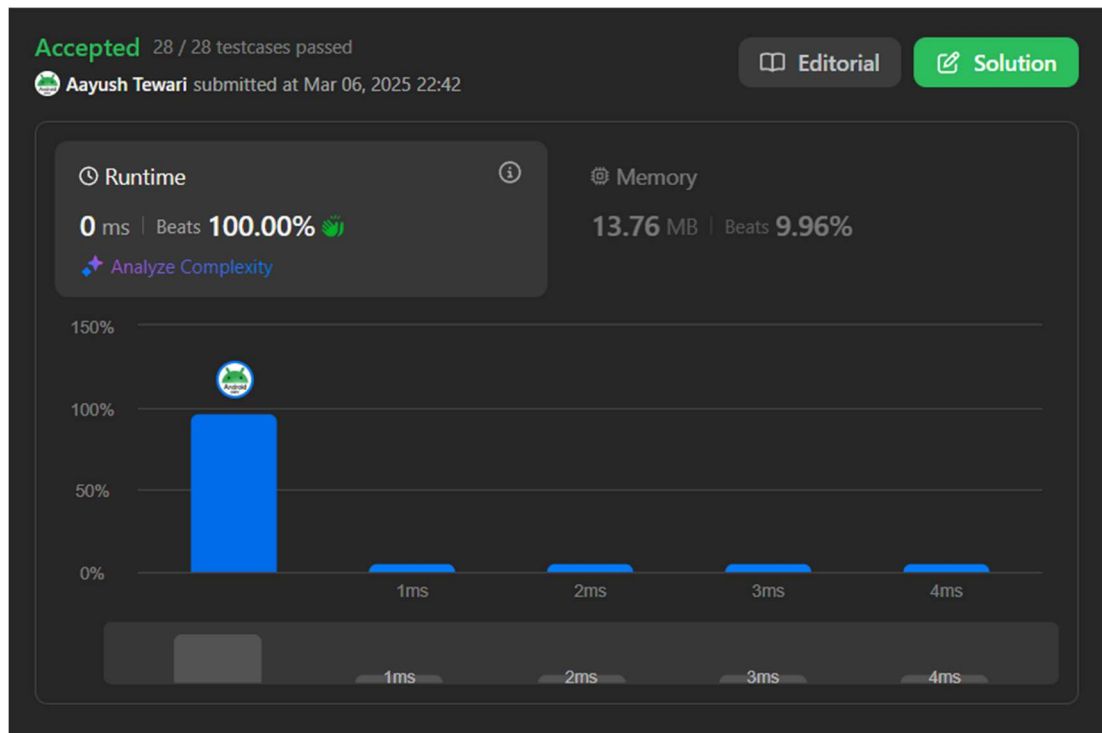
Question 3. Given the head of a singly linked list, reverse the list, and return the reversed list.(REVERSE LINKEDLIST)

Solution 3.

Code Snippet

```
Run Submit
C++ Auto
9 * };
10 */
11 class Solution {
12 public:
13     ListNode* reverseList(ListNode* head) {
14         if (!head || !head->next)
15             return head;
16
17         ListNode* newHead = reverseList(head->next);
18         head->next->next = head;
19         head->next = nullptr;
20         return newHead;
21     }
22 };
```

Output:



Question 4. You are given the head of a linked list. **Delete the middle node**, and return *the head of the modified linked list*. (**Delete the Middle Node of a Linked List**)

Solution 4.

Code Snippet:

```
11 class Solution {
12 public:
13     ListNode* deleteMiddle(ListNode* head) {
14         if (!head || !head->next) return nullptr; // If there's only one node, return nullptr.
15
16         ListNode* slow = head;
17         ListNode* fast = head;
18         ListNode* prev = nullptr;
19
20         while (fast && fast->next) {
21             prev = slow;
22             slow = slow->next;
23             fast = fast->next->next;
24         }
25         prev->next = slow->next;
26         return head;
27     }
28 };
29
```

Output:

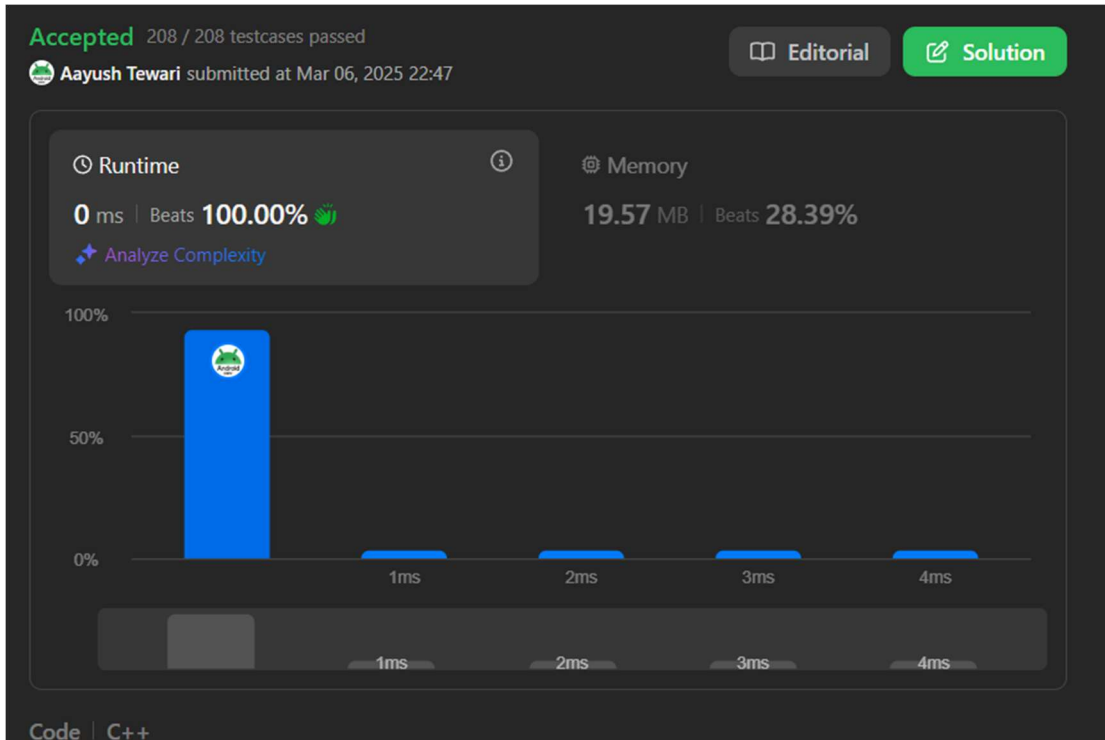


Question 5. You are given the heads of two sorted linked lists list1 and list2. Merge the two lists into one **sorted** list. The list should be made by splicing together the nodes of the first two lists. (**Merge Two Sorted Lists**)

Solution 5:

```
11 class Solution {
12 public:
13     ListNode* mergeTwoLists(ListNode* l1, ListNode* l2) {
14         ListNode *head = new ListNode(-1);
15         ListNode *cur = head;
16
17         while(l1 != nullptr && l2 != nullptr){
18             if(l1->val <= l2->val){
19                 cur->next = l1;
20                 l1 = l1->next;
21             }else{
22                 cur->next = l2;
23                 l2 = l2->next;
24             }
25             cur = cur->next;
26         }
27
28         if(l1 != nullptr) cur->next = l1;
29         else if(l2 != nullptr) cur->next = l2;
30
31         return head->next;
32     }
33 };
```

Output:



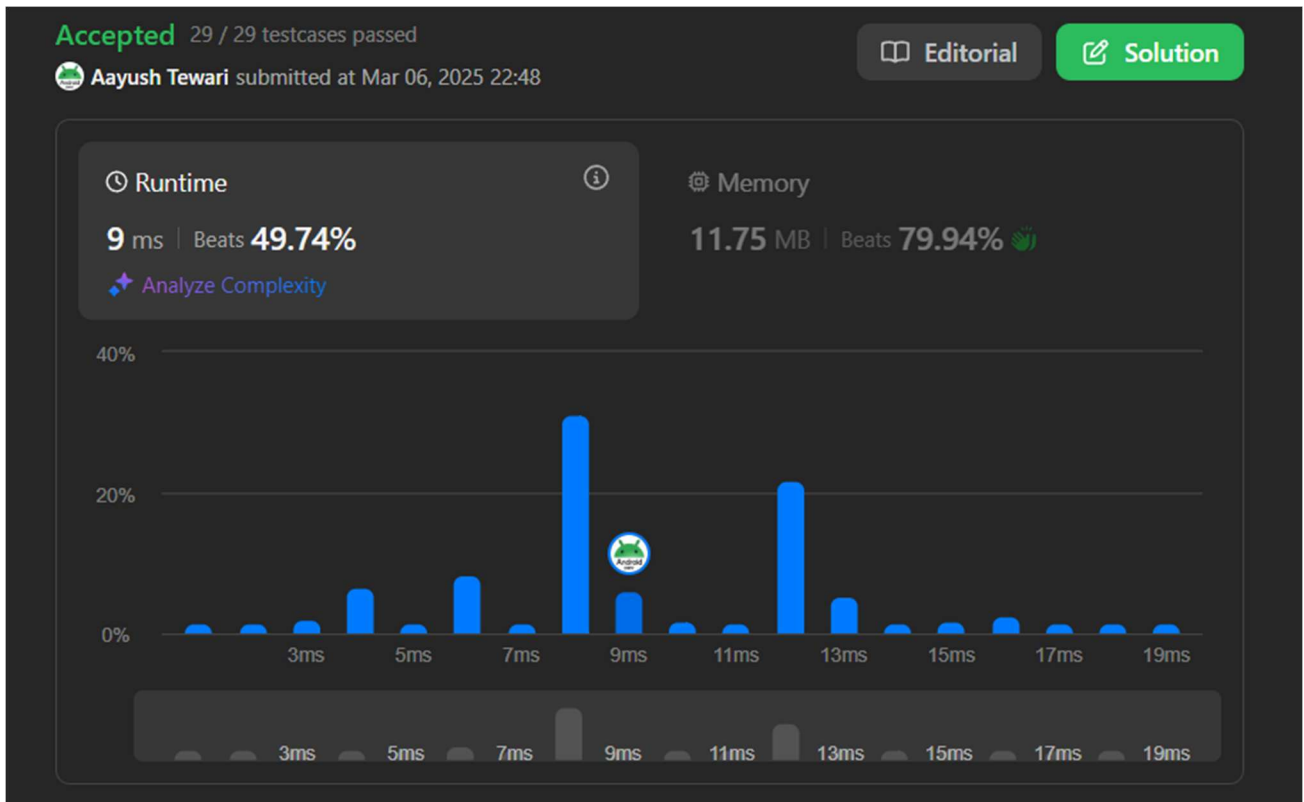
Question 6. Linked List Cycle

Solution 6.

Code Snippet:

```
1  /**
2   * Definition for singly-linked list.
3   * struct ListNode {
4   *     int val;
5   *     ListNode *next;
6   *     ListNode(int x) : val(x), next(NULL) {}
7   * };
8   */
9  class Solution {
10 public:
11     bool hasCycle(ListNode *head) {
12         ListNode *slow = head, *fast = head;
13
14         while (fast != nullptr && fast->next != nullptr) {
15             slow = slow->next;
16             fast = fast->next->next;
17
18             if (slow == fast) return true;
19         }
20
21         return false;
22     }
23 };
24
```

Output:



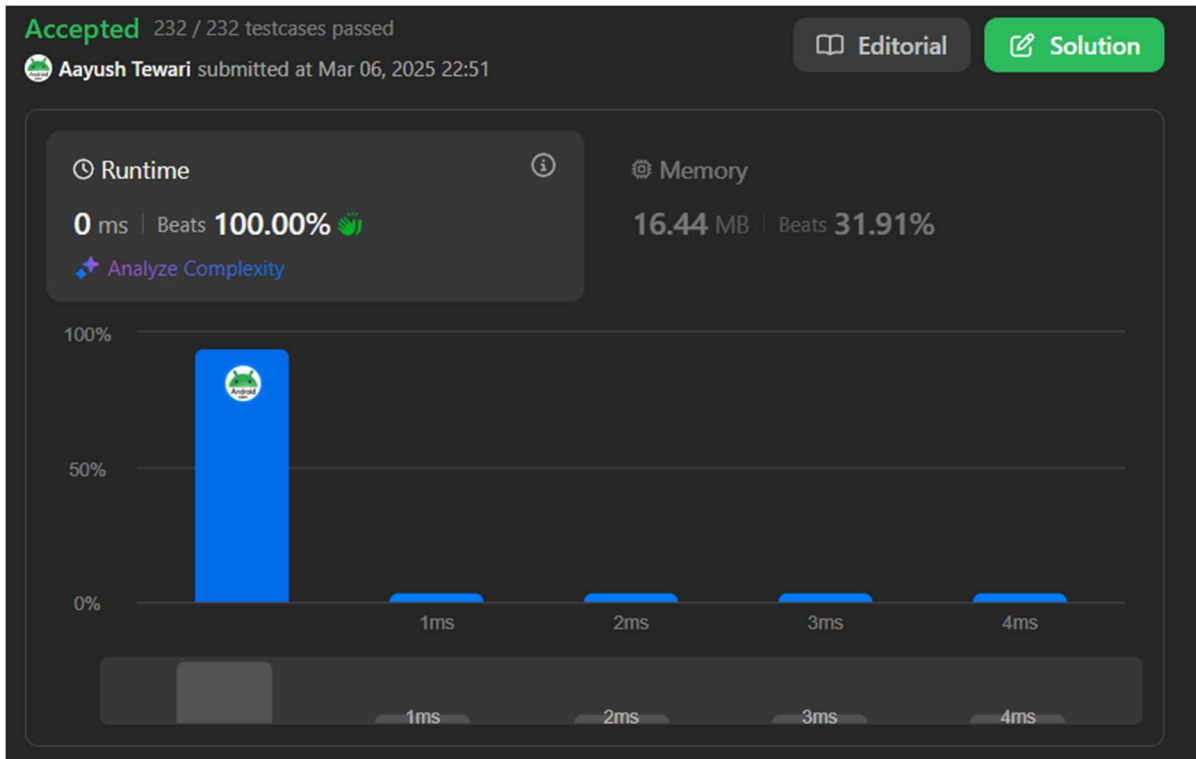
Question 7. Given the head of a linked list, rotate the list to the right by k places. (Rotate List)

Solution 7.

Code Snippet

```
11 class Solution {
12 public:
13     ListNode* rotateRight(ListNode* head, int k) {
14         if (!head || !head->next || k == 0)
15             return head;
16
17         ListNode* tail;
18         int length = 1;
19         for (tail = head; tail->next; tail = tail->next)
20             ++length;
21         tail->next = head; // Circle the list.
22
23         const int t = length - k % length;
24         for (int i = 0; i < t; ++i)
25             tail = tail->next;
26         ListNode* newHead = tail->next;
27         tail->next = nullptr;
28
29         return newHead;
30     }
31 };
```

Output:



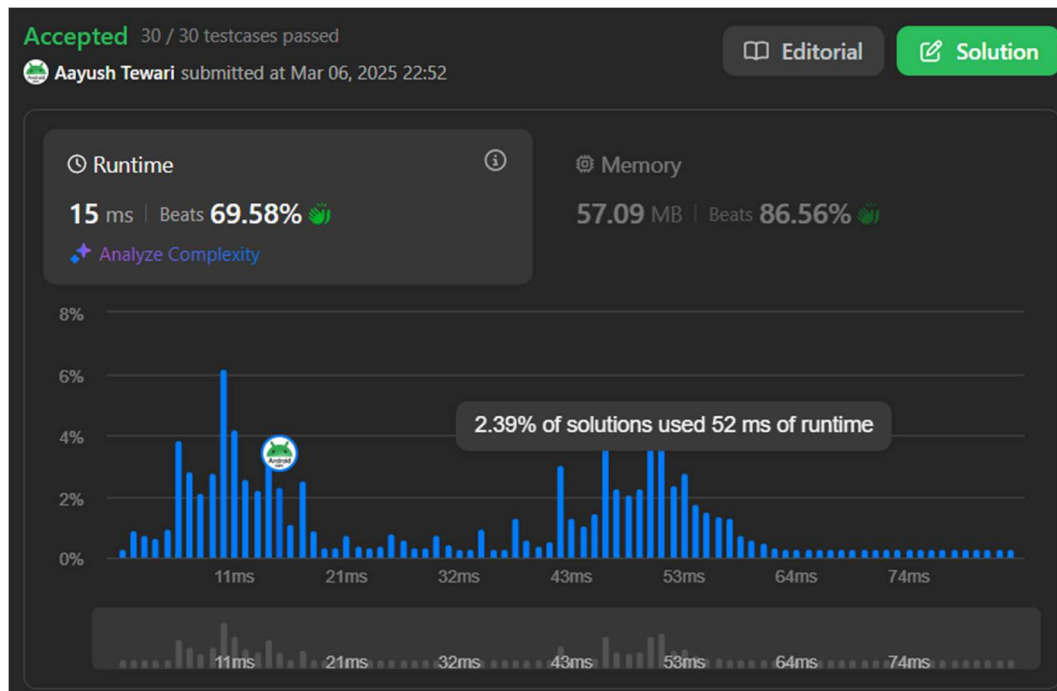
Question 8. Given the head of a linked list, return the list after sorting it in ascending order.(Sort List)

Solution 8.

Code Snippet:

```
11 class Solution {
12 public:
13     ListNode* sortList(ListNode* head) {
14         if (!head || !head->next) {
15             return head;
16         }
17         ListNode* slow = head;
18         ListNode* fast = head->next;
19         while (fast && fast->next) {
20             slow = slow->next;
21             fast = fast->next->next;
22         }
23         ListNode* l1 = head;
24         ListNode* l2 = slow->next;
25         slow->next = nullptr;
26         l1 = sortList(l1);
27         l2 = sortList(l2);
28         ListNode* dummy = new ListNode();
29         ListNode* tail = dummy;
30         while (l1 && l2) {
31             if (l1->val <= l2->val) {
32                 tail->next = l1;
33                 l1 = l1->next;
34             } else {
35                 tail->next = l2;
36                 l2 = l2->next;
37             }
38             tail = tail->next;
39         }
40         tail->next = l1 ? l1 : l2;
41         return dummy->next;
42     }
43 };
```


Output:



Question 9. You are given an array of k linked-lists lists, each linked-list is sorted in ascending order.

Merge all the linked-lists into one sorted linked-list and return it. (**Merge k Sorted Lists**)

Solution 9.

Code Snippet:

```
11 class Solution {
12 public:
13     ListNode* mergeKLists(vector<ListNode*>& lists) {
14         auto cmp = [](ListNode* a, ListNode* b) { return a->val > b->val; };
15         priority_queue<ListNode*, vector<ListNode*>, decltype(cmp)> pq;
16         for (auto head : lists) {
17             if (head) {
18                 pq.push(head);
19             }
20         }
21         ListNode* dummy = new ListNode();
22         ListNode* cur = dummy;
23         while (!pq.empty()) {
24             ListNode* node = pq.top();
25             pq.pop();
26             if (node->next) {
27                 pq.push(node->next);
28             }
29             cur->next = node;
30             cur = cur->next;
31         }
32         return dummy->next;
33     }
34 };
```

Output:

